



SAS Publishing

Base SAS® 9 Procedures Guide

Volume 2



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2002. *Base SAS® 9 Procedures Guide*. Cary, NC: SAS Institute Inc.

Base SAS® 9 Procedures Guide

Copyright © 2002 by SAS Institute Inc., Cary, NC, USA

ISBN 1-58025-942-1

All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc. This title includes documentation for early adopter features. THIS DOCUMENTATION FOR AN EARLY ADOPTER FEATURE IS A PRELIMINARY DRAFT AND IS PROVIDED BY SAS INSTITUTE INC. ON AN "AS IS" BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE. The company does not warrant that this documentation is complete, accurate, similar to that which may be released to the general public, or that any such documentation will be released. The company shall not be liable whatsoever for any damages arising out of the use of this documentation, including any direct, indirect, or consequential damages. The company reserves the right to alter or abandon use of this documentation at any time.

U.S. Government Restricted Rights Notice. Use, duplication, or disclosure of the software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227-19 Commercial Computer Software-Restricted Rights (June 1987).

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, June 2002

SAS Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at www.sas.com/pubs or call 1-800-727-3228.

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.® indicates USA registration.

IBM® and DB2® are registered trademarks or trademarks of International Business Machines Corporation. ORACLE® is a registered trademark of Oracle Corporation. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

Contents

What's New **xi**

Overview **xi**

Details **xi**

PART 1 Concepts 1

Chapter 1 △ Choosing the Right Procedure 3

Functional Categories of Base SAS Procedures 3

Report-Writing Procedures 4

Statistical Procedures 6

Utility Procedures 8

Brief Descriptions of Base SAS Procedures 10

Chapter 2 △ Fundamental Concepts for Using Base SAS Procedures 15

Language Concepts 16

Procedure Concepts 19

Output Delivery System 32

Chapter 3 △ Statements with the Same Function in Multiple Procedures 53

Overview 53

Statements 54

PART 2 Procedures 67

Chapter 4 △ The APPEND Procedure 71

Overview: APPEND Procedure 71

Syntax: PROC APPEND 71

Chapter 5 △ The CALENDAR Procedure 73

Overview: CALENDAR Procedure 74

Syntax: CALENDAR Procedure 79

Concepts: CALENDAR Procedure 97

Results: CALENDAR Procedure 107

Examples: CALENDAR Procedure 108

Chapter 6 △ The CATALOG Procedure 143

Overview: CATALOG Procedure 143

Syntax: PROC CATALOG 144

Concepts: CATALOG Procedure 154

Examples: CATALOG Procedure 158

Chapter 7 △ The CHART Procedure 165

Overview: CHART Procedure 165

Syntax: CHART Procedure	170
Concepts: CHART Procedure	183
Results: CHART Procedure	183
Examples: CHART Procedure	184
References	197
Chapter 8 △ The CIMPORT Procedure	199
Overview: CIMPORT Procedure	199
Syntax: PROC CIMPORT	200
Results: CIMPORT Procedure	205
Examples: CIMPORT Procedure	205
Chapter 9 △ The COMPARE Procedure	209
Overview: COMPARE Procedure	209
Syntax: COMPARE Procedure	213
Concepts: COMPARE Procedure	224
Results: COMPARE Procedure	228
Examples: COMPARE Procedure	239
Chapter 10 △ The CONTENTS Procedure	257
Overview: CONTENTS Procedure	257
Syntax: PROC CONTENTS	257
Chapter 11 △ The COPY Procedure	259
Overview: COPY Procedure	259
Syntax: PROC COPY	259
Concepts: COPY Procedure	260
Example: COPY Procedure	260
Chapter 12 △ The CORR Procedure	263
Overview: CORR Procedure	263
Syntax: CORR Procedure	267
Concepts: CORR Procedure	276
Statistical Computations: CORR Procedure	279
Results: CORR Procedure	287
Examples: CORR Procedure	291
References	306
Chapter 13 △ The CPORT Procedure	307
Overview: CPORT Procedure	307
Syntax: PROC CPORT	308
Concepts: CPORT Procedure	316
Results: CPORT Procedure	317
Examples: CPORT Procedure	317
Chapter 14 △ The CV2VIEW Procedure	323
Information about the CV2VIEW Procedure	323

Chapter 15 △ The DATASETS Procedure 325

Overview: DATASETS Procedure 326

Syntax: PROC DATASETS 329

Concepts: DATASETS Procedure 375

Results: DATASETS Procedure 381

Examples: DATASETS Procedure 392

Chapter 16 △ The DBCSTAB Procedure 407

Overview: DBCSTAB Procedure 407

Syntax: DBCSTAB Procedure 407

Details: When Do I Use the DBCSTAB Procedure? 408

Examples: DBCSTAB Procedure 409

See Also 411

Chapter 17 △ The DISPLAY Procedure 413

Overview: DISPLAY Procedure 413

Syntax: DISPLAY Procedure 413

Example: DISPLAY Procedure 414

Chapter 18 △ The DOCUMENT Procedure 417

Information about the DOCUMENT Procedure 417

Chapter 19 △ The EXPLODE Procedure 419

Overview: EXPLODE Procedure 419

Syntax: EXPLODE Procedure 420

Examples: EXPLODE Procedure 423

Chapter 20 △ The EXPORT Procedure 427

Overview: EXPORT Procedure 427

Syntax: PROC EXPORT 428

Examples: PROC EXPORT 434

Chapter 21 △ The FORMAT Procedure 441

Overview: FORMAT Procedure 441

Syntax: FORMAT Procedure 443

Informat and Format Options 462

Specifying Values or Ranges 464

Concepts: FORMAT Procedure 465

Results: FORMAT Procedure 468

Examples: FORMAT Procedure 474

See Also 493

Chapter 22 △ The FORMS Procedure 495

Overview: FORMS Procedure 495

Syntax: FORMS Procedure 497

Concepts: FORMS Procedure 503

Examples: FORMS Procedure 505

Chapter 23 △ The FREQ Procedure 513

Overview: FREQ Procedure 515
 Syntax: FREQ Procedure 518
 Concepts: FREQ Procedure 541
 Statistical Computations: FREQ Procedure 544
 Results: FREQ Procedure 585
 Examples: FREQ Procedure 592
 References 623

Chapter 24 △ The FSLIST Procedure 627

Overview: FSLIST Procedure 627
 Syntax: FSLIST Procedure 627

Chapter 25 △ The IMPORT Procedure 633

Overview: IMPORT Procedure 633
 Syntax: PROC IMPORT 634
 Examples: IMPORT Procedure 641

Chapter 26 △ The MEANS Procedure 649

Overview: MEANS Procedure 650
 Syntax: MEANS Procedure 652
 Concepts: MEANS Procedure 675
 Statistical Computations: MEANS Procedure 678
 Results: MEANS Procedure 681
 Examples: MEANS Procedure 683
 References 712

Chapter 27 △ The OPTIONS Procedure 713

Overview: OPTIONS Procedure 713
 Syntax: OPTIONS Procedure 716
 Results: OPTIONS Procedure 717
 Examples: OPTIONS Procedure 717

Chapter 28 △ The OPTLOAD Procedure 721

Overview: OPTLOAD Procedure 721
 Syntax: OPTLOAD Procedure 721

Chapter 29 △ The OPTSAVE Procedure 723

Overview: OPTSAVE Procedure 723
 Syntax: OPTSAVE Procedure 723

Chapter 30 △ The PLOT Procedure 725

Overview: PLOT Procedure 726
 Syntax: PLOT Procedure 728
 Concepts: PLOT Procedure 744
 Results: PLOT Procedure 749
 Examples: PLOT Procedure 750

Chapter 31 △ The PMENU Procedure 779

Overview: PMENU Procedure 779

Syntax: PMENU Procedure 780

Concepts: PMENU Procedure 793

Examples: PMENU Procedure 796

Chapter 32 △ The PRINT Procedure 817

Overview: PRINT Procedure 817

Syntax: PRINT Procedure 820

Results: Print Procedure 834

Examples: PRINT Procedure 837

Chapter 33 △ The PRINTTO Procedure 879

Overview: PRINTTO Procedure 879

Syntax: PRINTTO Procedure 880

Concepts: PRINTTO Procedure 883

Examples: PRINTTO Procedure 883

Chapter 34 △ The PRTDEF Procedure 893

Overview: PRTDEF Procedure 893

Syntax: PRTDEF Procedure 893

Input Data Set: PRTDEF Procedure 895

Examples: PRTDEF Procedure 899

See Also 903

Chapter 35 △ The PRTEXP Procedure 905

Overview: PRTEXP Procedure 905

Syntax: PRTEXP Procedure 905

Concepts: PRTEXP Procedure 906

Examples: PRTEXP Procedure 907

See Also 908

Chapter 36 △ The RANK Procedure 909

Overview: RANK Procedure 909

Syntax: RANK Procedure 911

Concepts: RANK Procedure 915

Results: RANK Procedure 916

Examples: RANK Procedure 917

References 923

Chapter 37 △ The REGISTRY Procedure 925

Overview: REGISTRY Procedure 925

Syntax: REGISTRY Procedure 925

Creating Registry Files with the REGISTRY Procedure 929

Examples: REGISTRY Procedure 932

See Also 936

Chapter 38 △ The REPORT Procedure 937

Overview: REPORT Procedure 939
 Concepts: REPORT Procedure 944
 Syntax: REPORT Procedure 958
 REPORT Procedure Windows 1000
 How PROC REPORT Builds a Report 1024
 Examples: REPORT Procedure 1037

Chapter 39 △ The SORT Procedure 1091

Overview: SORT Procedure 1091
 Syntax: SORT Procedure 1093
 Concepts: SORT Procedure 1100
 Integrity Constraints: SORT Procedure 1102
 Results: SORT Procedure 1102
 Examples: SORT Procedure 1103

Chapter 40 △ The SQL Procedure 1113

Overview: SQL Procedure 1115
 Syntax: SQL Procedure 1117
 SQL Procedure Component Dictionary 1154
 Concepts: SQL Procedure 1197
 PROC SQL and the ANSI Standard 1204
 Examples: SQL Procedure 1207

Chapter 41 △ The STANDARD Procedure 1243

Overview: STANDARD Procedure 1243
 Syntax: STANDARD Procedure 1245
 Results: STANDARD Procedure 1250
 Statistical Computations: STANDARD Procedure 1250
 Examples: STANDARD Procedure 1251

Chapter 42 △ The SUMMARY Procedure 1257

Overview: SUMMARY Procedure 1257
 Syntax: SUMMARY Procedure 1257

Chapter 43 △ The TABULATE Procedure 1259

Overview: TABULATE Procedure 1260
 Terminology Used with PROC TABULATE 1263
 Syntax: TABULATE Procedure 1266
 Concepts: TABULATE Procedure 1291
 Results: TABULATE Procedure 1299
 Examples: TABULATE Procedure 1310
 References 1361

Chapter 44 △ The TEMPLATE Procedure 1363

Information about the TEMPLATE Procedure 1363

Chapter 45 △ The TIMEPLOT Procedure 1365

Overview: TIMEPLOT Procedure 1365

Syntax: TIMEPLOT Procedure 1367

Results: TIMEPLOT Procedure 1375

Examples: TIMEPLOT Procedure 1376

Chapter 46 △ The TRANSPOSE Procedure 1387

Overview: TRANSPOSE Procedure 1387

Syntax: TRANSPOSE Procedure 1389

Results: TRANSPOSE Procedure 1395

Examples: TRANSPOSE Procedure 1396

Chapter 47 △ The TRANTAB Procedure 1409

Overview: TRANTAB Procedure 1409

Concepts: TRANTAB Procedure 1410

Syntax: TRANTAB Procedure 1413

Examples: TRANTAB Procedure 1419

Chapter 48 △ The UNIVARIATE Procedure 1435

Overview: UNIVARIATE Procedure 1436

Syntax: UNIVARIATE Procedure 1442

Concepts: UNIVARIATE Procedure 1511

Statistical Computations: UNIVARIATE Procedure 1517

Results: UNIVARIATE Procedure 1540

Examples: UNIVARIATE Procedure 1543

References 1572

PART 3 Appendices 1575**Appendix 1 △ SAS Elementary Statistics Procedures 1577**

Overview 1577

Keywords and Formulas 1578

Statistical Background 1586

References 1611

Appendix 2 △ Operating Environment-Specific Procedures 1613

Descriptions of Operating Environment-Specific Procedures 1613

Appendix 3 △ Raw Data and DATA Steps 1615

Overview 1615

AIRCRAFT 1615

CENSUS 1616

CHARITY 1617

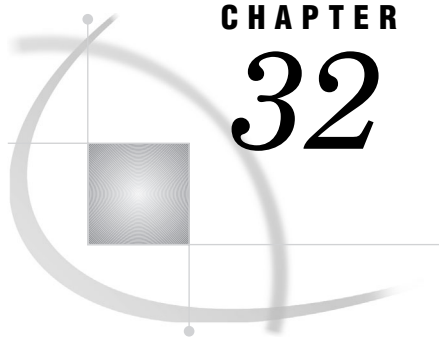
CUSTOMER_RESPONSE 1619

DJIA 1621

EDUCATION 1622

EMPDATA 1623

ENERGY	1625
GROC	1626
HOMELOANS	1627
MATCH_11	1641
PROCLIB.DELAY	1642
PROCLIB.EMP95	1643
PROCLIB.EMP96	1644
PROCLIB.INTERNAT	1645
PROCLIB.LAKES	1646
PROCLIB.MARCH	1646
PROCLIB.PAYLIST2	1647
PROCLIB.PAYROLL	1648
PROCLIB.PAYROLL2	1651
PROCLIB.SCHEDULE	1651
PROCLIB.STAFF	1654
PROCLIB.SUPERV	1657
RADIO	1658
STATEPOP	1670
Appendix 4 △ Recommended Reading	1673
Recommended Reading	1673
Index	1675



CHAPTER 32

The PRINT Procedure

<i>Overview: PRINT Procedure</i>	817
<i>Syntax: PRINT Procedure</i>	820
<i>PROC PRINT Statement</i>	820
<i>BY Statement</i>	829
<i>ID Statement</i>	830
<i>PAGEBY Statement</i>	831
<i>SUM Statement</i>	832
<i>SUMBY Statement</i>	833
<i>VAR Statement</i>	833
<i>Results: Print Procedure</i>	834
<i>Procedure Output</i>	834
<i>Page Layout</i>	834
<i>Column Headings</i>	836
<i>Column Width</i>	836
<i>Examples: PRINT Procedure</i>	837
<i>Example 1: Selecting Variables to Print</i>	837
<i>Example 2: Customizing Text in Column Headers</i>	840
<i>Example 3: Creating Separate Sections of a Report for Groups of Observations</i>	844
<i>Example 4: Summing Numeric Variables with One BY Group</i>	849
<i>Example 5: Summing Numeric Variables with Multiple BY Variables</i>	853
<i>Example 6: Limiting the Number of Sums in a Report</i>	858
<i>Example 7: Controlling the Layout of a Report with Many Variables</i>	863
<i>Example 8: Creating a Customized Layout with BY Groups and ID Variables</i>	869
<i>Example 9: Printing All the Data Sets in a SAS Library</i>	875

Overview: PRINT Procedure

The PRINT procedure prints the observations in a SAS data set, using all or some of the variables. You can create a variety of reports ranging from a simple listing to a highly customized report that groups the data and calculates totals and subtotals for numeric variables.

Output 32.1 on page 818 illustrates the simplest kind of report that you can produce. The statements that produce the output follow. Example 1 on page 837 creates the data set EXPREV.

```
options nodate pageno=1 linesize=64 pagesize=60;

proc print data=exprev;
run;
```

Output 32.1 Simple Listing Report Produced with PROC PRINT

The SAS System						1
Obs	Region	State	Month	Expenses	Revenues	
1	Southern	GA	JAN95	2000	8000	
2	Southern	GA	FEB95	1200	6000	
3	Southern	FL	FEB95	8500	11000	
4	Northern	NY	FEB95	3000	4000	
5	Northern	NY	MAR95	6000	5000	
6	Southern	FL	MAR95	9800	13500	
7	Northern	MA	MAR95	1500	1000	

The following HTML report is a customized report that is produced by PROC PRINT using ODS. The statements that create this report

- ☐ create HTML output
- ☐ customize the appearance of the report
- ☐ customize the title and the column headings
- ☐ place dollar signs and commas in numeric output
- ☐ selectively include and control the order of variables in the report
- ☐ group the data by JobCode
- ☐ sum the values for Salary for each job code and for all job codes.

For an explanation of the program that produces this report, see “Program: Creating an HTML Report with the STYLE= Option” on page 873.

Display 32.1 Customized Report Produced by PROC PRINT Using ODS

*Expenses Incurred for
Salaries for Flight Attendants and Mechanics*

Job Code	Gender	Annual Salary
PA1	F	\$23,177.00
	F	\$22,454.00
	M	\$22,266.00
PA1		\$67,899.00
PA2	F	\$28,608.00
	F	\$27,787.00
	M	\$28,572.00
PA2		\$85,247.00
PA3	F	\$32,886.00
	F	\$33,419.00
	M	\$32,217.00
PA3		\$98,522.00
ME1	M	\$20,769.00
	M	\$20,092.00
	M	\$20,619.00
ME1		\$61,480.00
ME2	F	\$25,108.00
	F	\$24,029.00
	M	\$25,245.00
	M	\$26,025.00
	M	\$25,090.00
	M	\$25,185.00
ME2		\$212,582.00
ME3	M	\$43,025.00
		\$593,735.00

Syntax: PRINT Procedure

Tip: Supports the Output Delivery System. See “Output Delivery System” on page 32 for details.

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, “Statements with the Same Function in Multiple Procedures,” on page 53 for details. You can also use any global statements as well. See “Global Statements” on page 18 for a list.

```

PROC PRINT <option(s)>;
  BY <DESCENDING> variable-1 <...<DESCENDING> variable-n>
    <NOTSORTED>;
  PAGEBY BY-variable;
  SUMBY BY-variable;
  ID variable(s) <option>;
  SUM variable(s) <option>;
  VAR variable(s) <option>;

```

To do this	Use this statement
Produce a separate section of the report for each BY group	BY
Identify observations by the formatted values of the variables that you list instead of by observation numbers	ID
Control page ejects that occur before a page is full	PAGEBY
Limit the number of sums that appear in the report	SUMBY
Total values of numeric variables	SUM
Select variables that appear in the report and determine their order	VAR

PROC PRINT Statement

```

PROC PRINT <option(s)>;

```

To do this	Use this option
Specify text for the HTML contents link to the output	CONTENTS=

To do this	Use this option
Specify the input data set	DATA=
Control general format	
Write a blank line between observations	DOUBLE
Print the number of observations in the data set, in BY groups, or both, and specify explanatory text to print with the number	N=
Suppress the column in the output that identifies each observation by number	NOOBS
Specify a column header for the column that identifies each observation by number	OBS=
Round unformatted numeric values to two decimal places	ROUND
Control page format	
Format the rows on a page	ROWS=
Use each variable's formatted width as its column width on all pages	WIDTH=UNIFORM
Control column format	
Control the orientation of the column headings	HEADING=
Use variables' labels as column headings	LABEL or SPLIT=
Specify the split character, which controls line breaks in column headings	SPLIT=
Specify one or more style elements for the Output Delivery System to use for different parts of the report	STYLE=
Determine the column width for each variable	WIDTH=

Options

CONTENTS=*link-text*

specifies the text for the links in the HTML contents file to the output produced by the PROC PRINT statement. For information on HTML output, see *SAS Output Delivery System User's Guide*.

Restriction: CONTENTS= does not affect the HTML body file. It affects only the HTML contents file.

DATA=*SAS-data-set*

specifies the SAS data set to print.

Main discussion: "Input Data Sets" on page 19

DOUBLE

writes a blank line between observations.

Alias: D

Restriction: This option has no effect on the HTML output.

Featured in: Example 1 on page 837

HEADING=*direction*

controls the orientation of the column headings, where *direction* is one of the following:

HORIZONTAL

prints all column headings horizontally.

Alias: H

VERTICAL

prints all column headings vertically.

Alias: V

Default: Headings are either all horizontal or all vertical. If you omit HEADING=, PROC PRINT determines the direction of the column headings as follows:

- If you do not use LABEL, spacing dictates whether column headings are vertical or horizontal.
- If you use LABEL and at least one variable has a label, all headings are horizontal.

LABEL

uses variables' labels as column headings.

Alias: L

Default: If you omit LABEL, PROC PRINT uses the variable's name as the column heading even if the PROC PRINT step contains a LABEL statement. If a variable does not have a label, PROC PRINT uses the variable's name as the column heading.

Interaction: By default, if you specify LABEL and at least one variable has a label, PROC PRINT prints all column headings horizontally. Therefore, using LABEL may increase the number of pages of output. (Use HEADING=VERTICAL in the PROC PRINT statement to print vertical column headings.)

Interaction: PROC PRINT sometimes conserves space by splitting labels across multiple lines. Use SPLIT= in the PROC PRINT statement to control where these splits occur. You do not need to use LABEL if you use SPLIT=.

Tip: To create a blank column header for a variable, use this LABEL statement in your PROC PRINT step:

```
label variable-name='00'x;
```

See also: For information on using the LABEL statement to create temporary labels in procedures see Chapter 3, "Statements with the Same Function in Multiple Procedures," on page 53.

For information on using the LABEL statement in a DATA step to create permanent labels, see the section on statements in *SAS Language Reference: Dictionary*.

Featured in: Example 3 on page 844

Note: The SAS system option LABEL must be in effect in order for any procedure to use labels. For more information see the section on system options in *SAS Language Reference: Dictionary* △

N<="string-1" <"string-2">>

prints the number of observations in the data set, in BY groups, or both and specifies explanatory text to print with the number.

If you use the N option ...	PROC PRINT ...
with neither a BY nor a SUM statement	prints the number of observations in the data set at the end of the report and labels the number with the value of <i>string-1</i> .
with a BY statement	prints the number of observations in the BY group at the end of each BY group and labels the number with the value of <i>string-1</i> .
with a BY statement and a SUM statement	prints the number of observations in the BY group at the end of each BY group and prints the number of observations in the data set at the end of the report. The numbers for BY groups are labeled with <i>string-1</i> ; the number for the entire data set is labeled with <i>string-2</i> .

Featured in: Example 2 on page 840 (alone)

Example 3 on page 844 (with a BY statement)

Example 4 on page 849 (with a BY statement and a SUM statement)

NOOBS

suppresses the observation number in the output.

Featured in: Example 3 on page 844

OBS=“column-header”

specifies a column header for the column that identifies each observation by number.

Tip: OBS= honors the split character (see the discussion of SPLIT= on page 824).

Featured in: Example 2 on page 840

ROUND

rounds unformatted numeric values to two decimal places. (Formatted values are already rounded by the format to the specified number of decimal places.) For both formatted and unformatted variables, PROC PRINT uses these rounded values to calculate any sums in the report.

If you omit ROUND, PROC PRINT adds the actual values of the rows to obtain the sum *even though it displays the formatted (rounded) values*. Any sums are also rounded by the format, but they include only one rounding error, that of rounding the sum of the actual values. The ROUND option, on the other hand, rounds values before summing them, so there may be multiple rounding errors. The results without ROUND are more accurate, but ROUND is useful for published reports where it is important for the total to be the sum of the printed (rounded) values.

Be aware that the results from PROC PRINT with the ROUND option may differ from the results of summing the same data with other methods such as PROC MEANS or the DATA step. Consider a simple case in which

- ☐ the data set contains three values for X: .003, .004, and .009.
- ☐ X has a format of 5.2.

Depending on how you calculate the sum, you can get three different answers: 0.02, 0.01, and 0.016. The following figure shows the results of calculating the sum with PROC PRINT (without and with the ROUND option) and PROC MEANS.

Figure 32.1 Three Methods of Summing Variables

Actual Values	PROC PRINT without the ROUND option		PROC PRINT with the ROUND option		PROC MEANS
	OBS	X	OBS	X	Analysis Variable : X
.003	1	0.00	1	0.00	Sum
.004	2	0.00	2	0.00	-----
.009	3	0.01	3	0.01	0.0160000
=====		=====		=====	-----
.016		0.02		0.01	

Notice that the sum produced without the ROUND option (.02) is closer to the actual result (0.16) than the sum produced with ROUND (0.01). However, the sum produced with ROUND reflects the numbers displayed in the report.

Alias: R

CAUTION:

Do not use ROUND with PICTURE formats. ROUND is for use with numeric values. SAS procedures treat variables that have picture formats as character variables. Using ROUND with such variables may lead to unexpected results. △

ROWS= *page-format*

formats rows on a page. Currently, PAGE is the only value that you can use for *page-format*:

PAGE

prints only one row of variables for each observation per page. When you use ROWS=PAGE, PROC PRINT does not divide the page into sections; it prints as many observations as possible on each page. If the observations do not fill the last page of the output, PROC PRINT divides the last page into sections and prints all the variables for the last few observations.

Restriction: Physical page size does not mean the same thing in HTML output as it does in traditional procedure output. Therefore, HTML output from PROC PRINT appears the same whether or not you use ROWS=.

Tip: The PAGE value can reduce the number of pages in the output if the data set contains large numbers of variables and observations. However, if the data set contains a large number of variables but few observations, the PAGE value can increase the number of pages in the output.

See also: “Page Layout” on page 834 for discussion of the default layout.

Featured in: Example 7 on page 863

SPLIT= *'split-character'*

specifies the split character, which controls line breaks in column headers. It also uses labels as column headers. PROC PRINT breaks a column heading when it reaches the split character and continues the header on the next line. The split character is not part of the column heading although each occurrence of the split character counts toward the 256-character maximum for a label.

Alias: S=

Interaction: You do not need to use both LABEL and SPLIT= because SPLIT= implies the use of labels.

Interaction: The OBS= option honors the split character. (See the discussion of OBS= on page 823.)

Featured in: Example 2 on page 840

Note: PROC PRINT does not split labels of BY variables in the heading preceding each BY group even if you specify SPLIT=. Instead, PROC PRINT replaces the split character with a blank. △

STYLE=

`<(location(s))>=<style-element-name>[<style-attribute-specification(s)>]`

specifies the style element to use for the specified locations in the report.

Note: You can use braces ({ and }) instead of square brackets ([and]). △

location

identifies the part of the report that the STYLE= option affects. The following table shows the available locations and the other statements in which you can specify them.

Note: Style specifications in a statement other than the PROC PRINT statement override the same style specification in the PROC PRINT statement. However, style attributes that you specify in the PROC PRINT statement are inherited, provided that you do not override the style with style specifications in another statement. For instance, if you specify a blue background and a white foreground for all column headers in the PROC PRINT statement, and you specify a gray background for the column headers of a variable in the VAR statement, the background for that particular column header is gray, and the foreground is white (as specified in the PROC PRINT statement). △

Table 32.1 Specifying Locations in the STYLE= Option

This location	Affects this part of the report	And can also be specified for individual items in this statement
BYLABEL	the label for the BY variable on the line containing the SUM totals	none
DATA	the cells of all columns	VAR ID SUM
GRANDTOTAL	the SUM line containing the grand totals for the whole report	SUM
HEADER	all column headers	VAR ID SUM
N	N= table and contents	none
OBS	the data in the OBS column	none
OBSHEADER	the header of the OBS column	none

This location	Affects this part of the report	And can also be specified for individual items in this statement
TABLE	the structural part of the report - that is, the underlying table used to set things like the width of the border and the space between cells	none
TOTAL	the SUM line containing totals for each BY group	SUM

For your convenience and for consistency with other procedures, the following table shows aliases for the different locations.

Table 32.2 Aliases for Locations

Location	Aliases
BYLABEL	BYSUMLABEL BYLBL BYSUMLBL
DATA	COLUMN COL
GRANDTOTAL	GRANDTOT GRAND GTOTAL GTOT
HEADER	HEAD HDR
N	none
OBS	OBSDATA OBSCOLUMN OBSCOL
OBSHEADER	OBSHEAD OBSHDR
TABLE	REPORT
TOTAL	TOT BYSUMLINE BYLINE BYSUM

style-element-name

is the name of a style element that is part of a style definition that is registered with the Output Delivery System. SAS provides some style definitions. Users can create their own style definitions with PROC TEMPLATE.

When style elements are processed, more specific style elements override less specific style elements.

Default: The following table shows the default style element for each location.

Table 32.3 The Default Style Element for Each Location in PROC PRINT

Location	Default style element
BYLABEL	Header
DATA	Data (for all but ID statement) RowHeader (for ID statement)
GRANDTOTAL	Header
HEADER	Header
N	NoteContent
OBS	RowHeader
OBSHEADER	Header
TABLE	Table
TOTAL	Header

style-attribute-specification

describes the style attribute to change. Each *style-attribute-specification* has this general form:

style-attribute-name=style-attribute-value

You can set these style attributes in the TABLE location:

BACKGROUND=	FONT_WIDTH=*
BACKGROUNDIMAGE=	FOREGROUND=*
BORDERCOLOR=	FRAME=
BORDERCOLORDARK=	HTMLCLASS=
BORDERCOLORLIGHT=	JUST=
BORDERWIDTH=	OUTPUTWIDTH=
CELLPADDING=	POSTHTML=
CELLSPACING=	POSTIMAGE=
FONT=*	POSTTEXT=
FONT_FACE=*	PREHTML=
FONT_SIZE=*	PREIMAGE=
FONT_STYLE=*	PRETEXT=
FONT_WEIGHT=*	RULES=

*When you use these attributes, they affect only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the foreground color or the font for the text that appears in the table, you must set the corresponding attribute in a location that affects the cells rather than the table.

You can set these style attributes in all locations other than TABLE:

ASIS=	FONT_WIDTH=
BACKGROUND=	HREFTARGET=
BACKGROUNDIMAGE=	HTMLCLASS=
BORDERCOLOR=	JUST=
BORDERCOLORDARK=	NOBREAKSPACE=
BORDERCOLORLIGHT=	POSTHTML=
BORDERWIDTH=	POSTIMAGE=
CELLHEIGHT=	POSTTEXT=
CELLWIDTH=	PREHTML=
FLYOVER=	PREIMAGE=
FONT=	PRETEXT=
FONT_FACE=	PROTECTSPECIALCHARS=
FONT_SIZE=	TAGATTR=
FONT_STYLE=	URL=
FONT_WEIGHT=	VJUST=

For information about style attributes, see DEFINE STYLE statement in *SAS Output Delivery System User's Guide*.

Restriction: This option affects all destinations except Listing and Output.

UNIFORM

See WIDTH=UNIFORM on page 828.

WIDTH=*column-width*

determines the column width for each variable. The value of *column-width* must be one of the following:

FULL

uses a variable's formatted width as the column width. If the variable does not have a format that explicitly specifies a field width, PROC PRINT uses the default width. For a character variable, the default width is the length of the variable. For a numeric variable, the default width is 12. When you use WIDTH=FULL, the column widths do not vary from page to page.

Tip: Using WIDTH=FULL can reduce execution time.

MINIMUM

uses for each variable the minimum column width that accommodates all values of the variable.

Alias: MIN

UNIFORM

uses each variable's formatted width as its column width on all pages. If the variable does not have a format that explicitly specifies a field width, PROC PRINT uses the widest data value as the column width. When you specify WIDTH=UNIFORM, PROC PRINT normally needs to read the data set twice.

However, if all the variables in the data set have formats that explicitly specify a field width (for example, BEST12. but not BEST.), PROC PRINT reads the data set only once.

Alias: U

Tip: If the data set is large and you want a uniform report, you can save computer resources by using formats that explicitly specify a field width so that PROC PRINT reads the data only once.

Tip: WIDTH=UNIFORM is the same as UNIFORM.

Restriction: When not all variables have formats that explicitly specify a width, you cannot use WIDTH=UNIFORM with an engine that supports concurrent access if another user is updating the data set at the same time.

UNIFORMBY

formats all columns uniformly within a BY group, using each variable's formatted width as its column width. If the variable does not have a format that explicitly specifies a field width, PROC PRINT uses the widest data value as the column width.

Alias: UBY

Restriction: You cannot use UNIFORMBY with a sequential data set.

Default: If you omit WIDTH= and do not specify the UNIFORM option, PROC PRINT individually constructs each page of output. The procedure analyzes the data for a page and decides how best to display them. Therefore, column widths may differ from one page to another.

Tip: Column width is affected not only by variable width but also by the length of column headings. Long column headings may lessen the usefulness of WIDTH=.

See also: For a discussion of default column widths, see "Column Width" on page 836.

BY Statement

Produces a separate section of the report for each BY group.

Main discussion: "BY" on page 54

Featured in: Example 3 on page 844, Example 4 on page 849, Example 5 on page 853, Example 6 on page 858, and Example 8 on page 869

```
BY <DESCENDING> variable-1
    <...<DESCENDING> variable-n>
    <NOTSORTED>;
```

Required Arguments

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, the observations in the data set must either be sorted by all the variables

that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Options

DESCENDING

specifies that the data set is sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The data are grouped in another way, such as chronological order.

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, the procedure treats each contiguous set as a separate BY group.

Using the BY Statement with an ID Statement

PROC PRINT uses a special layout if all BY variables appear in the same order at the beginning of the ID statement. (See Example 8 on page 869.)

Using the BY Statement with the NOBYLINE Option

If you use the BY statement with the SAS system option NOBYLINE, which suppresses the BY line that normally appears in output produced with BY-group processing, PROC PRINT always starts a new page for each BY group. This behavior ensures that if you create customized BY lines by putting BY-group information in the title and suppressing the default BY lines with NOBYLINE, the information in the titles matches the report on the pages.

ID Statement

Identifies observations by using the formatted values of the variables that you list instead of by using observation numbers.

Featured in: Example 7 on page 863 and Example 8 on page 869

ID *variable(s)* </ STYLE <(location(s))>
 =<style-element-name><[style-attribute-specification(s)]>>;

Required Arguments

variable(s)

specifies one or more variables to print instead of the observation number at the beginning of each row of the report.

Restriction: If the ID variables occupy so much space that no room remains on the line for at least one other variable, PROC PRINT writes a warning to the SAS log and does not treat all ID variables as ID variables.

Interaction: If a variable in the ID statement also appears in the VAR statement, the output contains two columns for that variable.

Options

STYLE *<(location(s))>=<style-element-name>[<style-attribute-specification(s)>]*
specifies the style element to use for ID columns created with the ID statement. For information about the arguments of this option and how it is used, see STYLE= on page 825 in the PROC PRINT statement.

Tip: To specify different style elements for different ID columns, use a separate ID statement for each variable and add a different STYLE= option to each ID statement.

Using the BY Statement with an ID Statement

PROC PRINT uses a special layout if all BY variables appear in the same order at the beginning of the ID statement. (See Example 8 on page 869.)

PAGEBY Statement

Controls page ejects that occur before a page is full.

Requirements: BY statement

Featured in: Example 3 on page 844

PAGEBY *BY-variable*;

Required Arguments

BY-variable

identifies a variable appearing in the BY statement in the PROC PRINT step. If the value of the BY variable changes, or if the value of any BY variable that precedes it in the BY statement changes, PROC PRINT begins printing a new page.

Interaction: If you use the BY statement with the SAS system option NOBYLINE, which suppresses the BY line that normally appears in output produced with BY-group processing, PROC PRINT always starts a new page for each BY group. This behavior ensures that if you create customized BY lines by putting BY-group information in the title and suppressing the default BY lines with NOBYLINE, the information in the titles matches the report on the pages. (See “Creating Titles That Contain BY-Group Information” on page 19.)

SUM Statement

Totals values of numeric variables.

Featured in: Example 4 on page 849, Example 5 on page 853, Example 6 on page 858, and Example 8 on page 869

```
SUM variable(s) </ STYLE <(location(s))>
    =<style-element-name><[style-attribute-specification(s)]>>;
```

Required Arguments

variable(s)

identifies the numeric variables to total in the report.

Option

STYLE <(location(s))>=<style-element-name><[style-attribute-specification(s)]>
 specifies the style element to use for cells containing sums that are created with the SUM statement. For information about the arguments of this option and how it is used, see STYLE= on page 825 in the PROC PRINT statement.

Tip: To specify different style elements for different cells reporting sums, use a separate SUM statement for each variable and add a different STYLE= option to each SUM statement.

Tip: If the STYLE= option is used in multiple SUM statements that affect the same location, the STYLE= option in the last SUM statement will be used.

Using the SUM and BY Statements Together

When you use a SUM statement and a BY statement with one BY variable, PROC PRINT sums the SUM variables for each BY group that contains more than one observation and totals them over all BY groups (see Example 4 on page 849).

When you use a SUM statement and a BY statement with multiple BY variables, PROC PRINT sums the SUM variables for each BY group that contains more than one observation, just as it does if you use only one BY variable. However, it provides sums only for those BY variables whose values change when the BY group changes. (See Example 5 on page 853.)

Note: When the value of a BY variable changes, the SAS System considers that the values of all variables listed after it in the BY statement also change. \triangle

SUMBY Statement

Limits the number of sums that appear in the report.

Requirements: BY statement

Featured in: Example 6 on page 858

SUMBY *BY-variable*;

Required Arguments

BY-variable

identifies a variable that appears in the BY statement in the PROC PRINT step. If the value of the BY variable changes, or if the value of any BY variable that precedes it in the BY statement changes, PROC PRINT prints the sums of all variables listed in the SUM statement.

What Variables Are Summed?

If you use a SUM statement, PROC PRINT subtotals only the SUM variables. Otherwise, PROC PRINT subtotals all the numeric variables in the data set except those listed in the ID and BY statements.

VAR Statement

Selects variables that appear in the report and determines their order.

Tip: If you omit the VAR statement, PROC PRINT prints all variables in the data set.

Featured in: Example 1 on page 837 and Example 8 on page 869

VAR *variable(s)* </ STYLE <(location(s))>
 =<style-element-name><[style-attribute-specification(s)]>>;

Required Arguments

variable(s)

identifies the variables to print. PROC PRINT prints the variables in the order that you list them.

Interaction: In the PROC PRINT output, variables that are listed in the ID statement precede variables that are listed in the VAR statement. If a variable in

the ID statement also appears in the VAR statement, the output contains two columns for that variable.

Option

STYLE *<(location(s))>=<style-element-name>[<style-attribute-specification(s)>*

specifies the style element to use for all columns that are created by a VAR statement. For information about the arguments of this option and how it is used, see STYLE= on page 825 in the PROC PRINT statement.

Tip: To specify different style elements for different columns, use a separate VAR statement to create a column for each variable and add a different STYLE= option to each VAR statement.

Results: Print Procedure

Procedure Output

PROC PRINT always produces a printed report. You control the appearance of the report with statements and options. See “Examples: PRINT Procedure” on page 837 for a sampling of the types of reports that the procedure produces.

Page Layout

By default, PROC PRINT uses an identical layout for all observations on a page of output. First, it attempts to print observations on a single line (see Figure 32.2 on page 834).

Figure 32.2 Printing Observations on a Single Line

Obs	Var_1	Var_2	Var_3	1
1	~~~~	~~~~	~~~~	
2	~~~~	~~~~	~~~~	
3	~~~~	~~~~	~~~~	
4	~~~~	~~~~	~~~~	
5	~~~~	~~~~	~~~~	
6	~~~~	~~~~	~~~~	

If PROC PRINT cannot fit all the variables on a single line, it splits the observations into two or more sections and prints the observation number or the ID variables at the beginning of each line. For example, in Figure 32.3 on page 835, PROC PRINT prints the values for the first three variables in the first section of each page and the values for the second three variables in the second section of each page.

Figure 32.3 Splitting Observations into Multiple Sections on One Page

1			
Obs	Var_1	Var_2	Var_3
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~
Obs	Var_4	Var_5	Var_6
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~

2			
Obs	Var_2	Var_3	
1	~~~~	~~~~	
2	~~~~	~~~~	
3	~~~~	~~~~	

Obs	Var_4	Var_5	Var_6
4	~~~~	~~~~	~~~~
5	~~~~	~~~~	~~~~
6	~~~~	~~~~	~~~~

If PROC PRINT cannot fit all the variables on one page, the procedure prints subsequent pages with the same observations until it has printed all the variables. For example, in Figure 32.4 on page 835, PROC PRINT uses the first two pages to print values for the first three observations and the second two pages to print values for the rest of the observations.

Figure 32.4 Splitting Observations across Multiple Pages

1			
Obs	Var_1	Var_2	Var_3
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~
Obs	Var_4	Var_5	Var_6
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~

2			
Obs	Var_7	Var_8	Var_9
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~
Obs	Var_10	Var_11	Var_12
1	~~~~	~~~~	~~~~
2	~~~~	~~~~	~~~~
3	~~~~	~~~~	~~~~

3			
Obs	Var_1	Var_2	Var_3
4	~~~~	~~~~	~~~~
5	~~~~	~~~~	~~~~
6	~~~~	~~~~	~~~~
Obs	Var_4	Var_5	Var_6
4	~~~~	~~~~	~~~~
5	~~~~	~~~~	~~~~
6	~~~~	~~~~	~~~~

4			
Obs	Var_7	Var_8	Var_9
4	~~~~	~~~~	~~~~
5	~~~~	~~~~	~~~~
6	~~~~	~~~~	~~~~
Obs	Var_10	Var_11	Var_12
4	~~~~	~~~~	~~~~
5	~~~~	~~~~	~~~~
6	~~~~	~~~~	~~~~

Note: You can alter the page layout with the ROWS= option in the PROC PRINT statement (see the discussion of ROWS= on page 824). Δ

Note: PROC PRINT may produce slightly different output if the data set is not RADIX addressable. Version 6 compressed files are not RADIX addressable, while,

beginning with Version 7, compressed files are RADIX addressable. (The integrity of the data is not compromised; the procedure simply numbers the observations differently.) △

Column Headings

By default, spacing dictates whether PROC PRINT prints column headings horizontally or vertically. Figure 32.2 on page 834, Figure 32.3 on page 835, and Figure 32.4 on page 835 all illustrate horizontal headings. Figure 32.5 on page 836 illustrates vertical headings.

Figure 32.5 Using Vertical Headings

	V	V	V	1
	a	a	a	
O	r	r	r	
b	—	—	—	
s	1	2	3	
1	~~~~	~~~~	~~~~	
2	~~~~	~~~~	~~~~	
3	~~~~	~~~~	~~~~	
4	~~~~	~~~~	~~~~	
5	~~~~	~~~~	~~~~	
6	~~~~	~~~~	~~~~	

Note: If you use LABEL and at least one variable has a label, PROC PRINT prints all column headings horizontally unless you specify HEADING=VERTICAL. △

Column Width

By default, PROC PRINT uses a variable's formatted width as the column width. (The WIDTH= option overrides this default behavior.) If the variable does not have a format that explicitly specifies a field width, PROC PRINT uses the widest data value for that variable on that page as the column width.

If the formatted value of a character variable or the data width of an unformatted character variable exceeds the linesize minus the length of all the ID variables, PROC PRINT may truncate the value. Consider the following situation:

- The linesize is 80.
- IdNumber is a character variable with a length of 10. It is used as an ID variable.
- State is a character variable with a length of 2. It is used as an ID variable.
- Comment is a character variable with a length of 200.

When PROC PRINT prints these three variables on a line, it uses 14 print positions for the two ID variables and the space after each one. This leaves 80–14, or 66, print positions for COMMENT. Longer values of COMMENT are truncated.

WIDTH= controls the column width.

Note: Column width is affected not only by variable width but also by the length of column headings. Long column headings may lessen the usefulness of WIDTH=. △

Examples: PRINT Procedure

Example 1: Selecting Variables to Print

Procedure features:

PROC PRINT statement options:

DOUBLE

STYLE=

VAR statement

Other Features:

ODS HTML statement

This example

- selects three variables for the report
- uses variable labels as column headings
- double spaces between rows of the report.

Program: Creating a Listing Report

Set the SAS system options.

```
options nodate pageno=1 linesize=70 pagesize=60;
```

Create the input data set. EXPREV contains information about a company's monthly expenses and revenues for two regions of the United States.

```
data exprev;
  input Region $ State $ Month monyy5.
         Expenses Revenues;
  format month monyy5.;
  datalines;
Southern GA JAN95 2000 8000
Southern GA FEB95 1200 6000
Southern FL FEB95 8500 11000
Northern NY FEB95 3000 4000
Northern NY MAR95 6000 5000
Southern FL MAR95 9800 13500
Northern MA MAR95 1500 1000
;
```

Print the data set EXPREV. DOUBLE inserts a blank line between observations. (This option has no effect on the HTML output.)

```
proc print data=exprev double;
```

Select the variables to include in the report. The VAR statement creates columns for Month, State, and Expenses, in that order.

```
var month state expenses;
```

Specify a title. The TITLE statement specifies a title for the report.

```
title 'Monthly Expenses for Offices in Each State';
run;
```

Output: Listing

Output 32.2 Selecting Variables: Listing Output

By default, PROC PRINT identifies each observation by number under the column heading **Obs**.

Monthly Expenses for Offices in Each State				1
Obs	Month	State	Expenses	
1	JAN95	GA	2000	
2	FEB95	GA	1200	
3	FEB95	FL	8500	
4	FEB95	NY	3000	
5	MAR95	NY	6000	
6	MAR95	FL	9800	
7	MAR95	MA	1500	

Program: Creating an HTML Report

You can easily create HTML output by adding ODS statements. In the following example, ODS statements were added to produce HTML output.

```
options nodate pageno=1 linesize=70 pagesize=60;
```

Create HTML output and specify the file to store the output in. The ODS HTML statement opens the HTML destination. FILE= specifies the external file that you want to contain the HTML output.

```
ods html file='your_file.html';
```



```
proc print data=exprev double;
    var month state expenses;
    title 'Monthly Expenses for Offices in Each State';
run;
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination.

```
ods html close;
```

Output: HTML

Display 32.2 Selecting Variables: Default HTML Output

Monthly Expenses for Offices in Each State

Obs	Month	State	Expenses
1	JAN95	GA	2000
2	FEB95	GA	1200
3	FEB95	FL	8500
4	FEB95	NY	3000
5	MAR95	NY	6000
6	MAR95	FL	9800
7	MAR95	MA	1500

Program: Creating an HTML Report with the STYLE= Option

You can go a step further and add more formatting to your HTML output. The following example uses the STYLE= option to add shading to your HTML report.

```
options nodate pageno=1 linesize=70 pagesize=60;
ods html file='your_file.html';
```

Create stylized HTML output. The first STYLE= option specifies that the column headers be written in white italic font.

The second STYLE= option specifies that SAS change the color of the background of the observations column to red.

```
Proc Print data=exprev double
    style(HEADER) = {font_style=italic foreground = white}
    style(OBS) = {background=red};
    var month state expenses;
    title 'Monthly Expenses for Offices in Each State';
run;
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination.

```
ods html close;
```

Output: HTML Output with Styles

Display 32.3 Selecting Variables: HTML Output Using Styles

Monthly Expenses for Offices in Each State

Obs	Month	State	Expenses
1	JAN95	GA	2000
2	FEB95	GA	1200
3	FEB95	FL	8500
4	FEB95	NY	3000
5	MAR95	NY	6000
6	MAR95	FL	9800
7	MAR95	MA	1500

Example 2: Customizing Text in Column Headers

Procedure features:

PROC PRINT statement options:

N
OBS=
SPLIT=
STYLE=

VAR statement option:

STYLE=

Other features:

LABEL statement
ODS PDF statement

Data set: EXPREV

This example

- customizes and underlines the text in column headings for variables
- customizes the column header for the column that identifies observations by number
- shows the number of observations in the report
- writes the values of Expenses with commas.

Program: Creating a Listing Report

```
options nodate pageno=1 linesize=70 pagesize=60;
```

Print the report and define the column headings. SPLIT= identifies the asterisk as the character that starts a new line in column headers. The N option prints the number of observations at the end of the report. OBS= specifies the column header for the column that identifies each observation by number. The split character (*) starts a new line in the column heading. Therefore, the equal signs (=) in the value of OBS= underline the column header.

```
proc print data=exprev split='*' n obs='Observation*Number*=====';
```

Select the variables to include in the report. The VAR statement creates columns for Month, State, and Expenses, in that order.

```
var month state expenses;
```

Assign the variables' labels as column headings. The LABEL statement associates a label with each variable for the duration of the PROC PRINT step. When you use SPLIT= in the PROC PRINT statement, the procedure uses labels for column headers. The split character (*) starts a new line in the column heading. Therefore, the equal signs (=) in the labels underline the column headers.

```
label month='Month*====='
      state='State*====='
      expenses='Expenses*=====';
```

Specify a title for the report, and format any variable containing numbers. The FORMAT statement assigns a format to use for Expenses in the report. The TITLE statement specifies a title.

```
format expenses comma10.;
title 'Monthly Expenses for Offices in Each State';
run;
```

Output: Listing

Output 32.3 Customizing Text in Column Headers: Listing Output

```

Monthly Expenses for Offices in Each State          1

Observation   Month   State   Expenses
Number
=====
1            JAN95    GA       2,000
2            FEB95    GA       1,200
3            FEB95    FL       8,500
4            FEB95    NY       3,000
5            MAR95    NY       6,000
6            MAR95    FL       9,800
7            MAR95    MA       1,500

N = 7

```

Program: Creating a PDF Report

You can easily create PDF output by adding a few ODS statements. In the following example, ODS statements were added to produce PDF output.

```
options nodate pageno=1 linesize=70 pagesize=60;
```

Create PDF output and specify the file to store the output in. The ODS PDF statement opens the PDF destination and creates PDF output. The FILE= argument specifies your external file that contains the PDF output.

```
ods pdf file='your_file.pdf';

proc print data=exprev split='*' n obs='Observation*Number*=====';
  var month state expenses;
  label month='Month*====='
        state='State*====='
        expenses='Expenses*=====';
  format expenses comma10.;

  title 'Monthly Expenses for Offices in Each State';
run;
```

Close the PDF destination. The ODS PDF CLOSE statement closes the PDF destination.

```
ods pdf close;
```

Output: PDF

Display 32.4 Customizing Text in Column Headers: Default PDF Output

Monthly Expenses for Offices in Each State

Observation Number	Month	State	Expenses
1	JAN95	GA	2,000
2	FEB95	GA	1,200
3	FEB95	FL	8,500
4	FEB95	NY	3,000
5	MAR95	NY	6,000
6	MAR95	FL	9,800
7	MAR95	MA	1,500
N = 7			

Program: Creating a PDF Report with the STYLE= Option

```
options nodate pageno=1 linesize=70 pagesize=60;
ods pdf file='your_file.pdf';
```

Create stylized PDF output. The first STYLE= option specifies that the background color of the cell containing the value for N be changed to blue and that the font style be changed to italic. The second STYLE= option specifies that the background color of the observation column, the observation header, and the other variable's headers be changed to white.

```
proc print data=expres split='*' n obs='Observation*Number*===== '
  style(N) = {font_style=italic background= blue}
  style(HEADER OBS OBSHEADER) = {background=white};
```

Create stylized PDF output. The STYLE= option changes the color of the cells containing data to gray.

```
var month state expenses / style (DATA)= [ background = gray ] ;
label month='Month*====='
      state='State*====='
      expenses='Expenses*=====';
format expenses comma10.;

title 'Monthly Expenses for Offices in Each State';
run;
```

Close the PDF destination. The ODS PDF CLOSE statement closes the PDF destination.

```
ods pdf close;
```

Output: PDF Report with Styles

Display 32.5 Customizing Text in Column Headers: PDF Output Using Styles

Monthly Expenses for Offices in Each State

Observation Number	Month	State	Expenses
1	JAN95	GA	2,000
2	FEB95	GA	1,200
3	FEB95	FL	8,500
4	FEB95	NY	3,000
5	MAR95	NY	6,000
6	MAR95	FL	9,800
7	MAR95	MA	1,500
<i>N = 7</i>			

Example 3: Creating Separate Sections of a Report for Groups of Observations

Procedure features:

PROC PRINT statement options:

LABEL

N=

NOOBS

STYLE=

BY statement

PAGEBY statement

Other features:

SORT procedure

LABEL statement

ODS RTF statement

Data set: EXPREV

This example

- suppresses the printing of observation numbers at the beginning of each row
- presents the data for each state in a separate section of the report

- begins a new page for each region.

Program: Creating a Listing Report

```
options pagesize=60 pageno=1 nodate linesize=70;
```

Sort the EXPREV data set. PROC SORT sorts the observations by Region, State, and Month.

```
proc sort data=exprev;
    by region state month;
run;
```

Print the report, specify the total number of observations in each BY group, and suppress the printing of observation numbers. N= prints the number of observations in a BY group at the end of that BY group. The explanatory text that the N= option provides precedes the number. NOOBS suppresses the printing of observation numbers at the beginning of the rows. LABEL uses variables' labels as column headings.

```
proc print data=exprev n='Number of observations for the state: '
    noobs label;
```

Specify the variables to include in the report. The VAR statement creates columns for Month, Expenses, and Revenues, in that order.

```
var month expenses revenues;
```

Create a separate section for each region of the state and specify page breaks for each BY group of Region. The BY statement produces a separate section of the report for each BY group and prints a heading above each one. The PAGEBY statement starts a new page each time the value of Region changes.

```
by region state;
pageby region;
```

Establish the column headings. The LABEL statement associates a label with the variable Region for the duration of the PROC PRINT step. When you use the LABEL option in the PROC PRINT statement, the procedure uses labels for column headings.

```
label region='Sales Region';
```

Format the columns that contain numbers and specify a title. The FORMAT statement assigns a format to Expenses and Revenues for this report. The TITLE statement specifies a title.

```
format revenues expenses comma10.;
title 'Sales Figures Grouped by Region and State';
```

```
run;
```

Output: Listing

Output 32.4 Creating Separate Sections of a Report for Groups of Observations: Listing Output

Sales Figures Grouped by Region and State			1
----- Sales Region=Northern State=MA -----			
Month	Expenses	Revenues	
MAR95	1,500	1,000	
Number of observations for the state: 1			
----- Sales Region=Northern State=NY -----			
Month	Expenses	Revenues	
FEB95	3,000	4,000	
MAR95	6,000	5,000	
Number of observations for the state: 2			

Sales Figures Grouped by Region and State			2
----- Sales Region=Southern State=FL -----			
Month	Expenses	Revenues	
FEB95	8,500	11,000	
MAR95	9,800	13,500	
Number of observations for the state: 2			
----- Sales Region=Southern State=GA -----			
Month	Expenses	Revenues	
JAN95	2,000	8,000	
FEB95	1,200	6,000	
Number of observations for the state: 2			

Program: Creating an RTF Report

```
options pagesize=60 pageno=1 nodate linesize=70;
```

Create output for Microsoft Word and specify the file to store the output in. The ODS RTF statement opens the RTF destination and creates output formatted for Microsoft Word. The FILE= argument specifies your external file that contains the RTF output.

```
ods rtf file='your_file.rtf';
```



```

proc sort data=exprev;
by region state month;
run;

proc print data=exprev n='Number of observations for the state: '
  noobs label;
  var month expenses revenues;
  by region state;
  pageby region;
  label region='Sales Region';
  format revenues expenses comma10.;
  title 'Sales Figures Grouped by Region
  and State';
run;

```

Close the RTF destination. The ODS RTF CLOSE statement closes the RTF destination.

```
ods rtf close;
```

Output: RTF

Display 32.6 Creating Separate Sections of a Report for Groups of Observations: Default RTF Output

<i>Sales Figures Grouped by Region and State</i>		
Sales Region=Northern State=MA		
Month	Expenses	Revenues
MAR95	1,500	1,000
Number of observations for the state: 1		
Sales Region=Northern State=NY		
Month	Expenses	Revenues
FEB95	5,000	4,000
MAR95	6,000	5,000
Number of observations for the state: 2		
<i>Sales Figures Grouped by Region and State</i>		
Sales Region=Southern State=FL		
Month	Expenses	Revenues
FEB95	8,001	11,001
MAR95	9,001	11,501
Number of observations for the state: 2		
Sales Region=Southern State=GA		
Month	Expenses	Revenues
MAR95	3,001	3,001
FEB95	1,001	6,001
Number of observations for the state: 2		

Program: Creating an RTF Report with the STYLE= Option

```
options pagesize=60 pageno=1 nodate linesize=70;
```

```
ods rtf file='your_file.rtf';

proc sort data=exprev;
by region state month;
run;
```

Create a stylized RTF report. The first STYLE= option specifies that the background color of the cell containing the number of observations be changed to gray.

The second STYLE= option specifies that the background color of the column header for the variable MONTH be changed to white.

The third STYLE= option specifies that the background color of the column header for the variable EXPENSES be changed to blue and the font color be changed to white.

The fourth STYLE= option specifies that the background color of the column header for the variable REVENUES be changed to gray.

```
proc print data=exprev n='Number of observations for the state: '
      noobs label style(N) = {background=gray};
var month / style(HEADER) = [background = white];
var expenses / style(HEADER) = [background = blue foreground=white];
var revenues / style(HEADER) = [background = gray];
by region state;
pageby region;
label region='Sales Region';
format revenues expenses comma10.;
title 'Sales Figures Grouped by Region
and State';
run;

ods rtf close;
```

Output: RTF with Styles

Display 32.7 Creating Separate Sections of a Report for Groups of Observations: RTF Output Using Styles

Sales Figures Grouped by Region and State

Sales Region=Northern State=MA

Month	Expenses	Revenues
MAR95	1,500	1,000
Number of observations for the state: 1		

Sales Region=Northern State=NY

Month	Expenses	Revenues
FEB95	3,000	4,000
MAR95	6,000	5,000
Number of observations for the state: 2		

Sales Figures Grouped by Region and State

Sales Region=Southern State=FL

Month	Expenses	Revenues
FEB95	8,500	11,000
MAR95	9,800	13,500
Number of observations for the state: 2		

Sales Region=Southern State=GA

Month	Expenses	Revenues
JAN95	2,000	8,000
FEB95	1,200	6,000
Number of observations for the state: 2		

Example 4: Summing Numeric Variables with One BY Group

Procedure features:

PROC PRINT statement options:

N=

BY statement

SUM statement

Other features:

ODS MARKUP statement

SORT procedure

TITLE statement

#BYVAL specification

SAS system options:

BYLINE

NOBYLINE

Data set: EXPREV

This example

- sums expenses and revenues for each region and for all regions
- shows the number of observations in each BY group and in the whole report
- creates a customized title, containing the name of the region. This title replaces the default BY line for each BY group.

Program: Creating a Listing Report

Start each BY group on a new page and suppress the printing of the default BY line.

The SAS system option NOBYLINE suppresses the printing of the default BY line. When you use PROC PRINT with NOBYLINE, each BY group starts on a new page.

```
options nodate pageno=1 linesize=70 pagesize=60 nobyline;
```

Sort the data set. PROC SORT sorts the observations by Region.

```
proc sort data=exprev;
    by region;
run;
```

Print the report, suppress the printing of observation numbers, and print the total number of observations for the selected variables. NOOBS suppresses the printing of observation numbers at the beginning of the rows. N= prints the number of observations in a BY group at the end of that BY group and (because of the SUM statement) prints the number of observations in the data set at the end of the report. The first piece of explanatory text that N= provides precedes the number for each BY group. The second piece of explanatory text that N= provides precedes the number for the entire data set.

```
proc print data=exprev noobs
    n='Number of observations for the state: '
    'Number of observations for the data set: ';
```

Sum the values for the selected variables. The SUM statement alone sums the values of Expenses and Revenues for the entire data set. Because the PROC PRINT step contains a BY statement, the SUM statement also sums the values of Expenses and Revenues for each region that contains more than one observation.

```
sum expenses revenues;
by region;
```

Format the numeric values for a specified column. The FORMAT statement assigns the COMMA10. format to Expenses and Revenues for this report.

```
format revenues expenses comma10.;
```

Specify and format a dynamic (or current) title. The TITLE statement specifies a title. The #BYVAL specification places the current value of the BY variable Region in the title. Because NOBYLINE is in effect, each BY group starts on a new page, and the title serves as a BY line.

```
title 'Revenue and Expense Totals for the
#byval(region) Region';
run;
```

Generate the default BY line. The SAS system option BYLINE resets the printing of the default BY line.

```
options byline;
```

Output: Listing

Output 32.5 Summing Numeric Variables with One BY Group: Listing Output

Revenue and Expense Totals for the Northern Region				1
State	Month	Expenses	Revenues	
NY	FEB95	3,000	4,000	
NY	MAR95	6,000	5,000	
MA	MAR95	1,500	1,000	
-----		-----	-----	
Region		10,500	10,000	
Number of observations for the state: 3				

Revenue and Expense Totals for the Southern Region				2
State	Month	Expenses	Revenues	
GA	JAN95	2,000	8,000	
GA	FEB95	1,200	6,000	
FL	FEB95	8,500	11,000	
FL	MAR95	9,800	13,500	
-----		-----	-----	
Region		21,500	38,500	
		=====	=====	
		32,000	48,500	
Number of observations for the state: 4				
Number of observations for the data set: 7				

Program: Creating an XML File

The following example opens the MARKUP destination. The output file will contain only XML tagging unless you have a browser that reads XML.

```
options nodate pageno=1 linesize=70 pagesize=60 nobyline;
```

Produce output that is tagged with Extensible Markup Language (XML) tags and specify the file to store it in. The ODS MARKUP statement opens the MARKUP destination and creates a file containing output that is tagged with XML tags. The FILE= argument specifies your external file that contains the XML output.

```
ods markup file='your_file.xml';
```

```
proc sort data=exprev;
    by region;
run;
```

```
proc print data=exprev noobs
    n='Number of observations for the state: '
    'Number of observations for the data set: ';
```

```
sum expenses revenues;
by region;
```

```
format revenues expenses comma10.;
```

```
title 'Revenue and Expense Totals for the
#byval(region) Region';
run;
```

```
options byline;
```

Close the MARKUP destination. The ODS RTF CLOSE statement closes the MARKUP destination.

```
ods markup close;
```

Output: XML file

Display 32.8 Summing Numeric Variables with One BY Group: XML Output

```

<?xml version="1.0" encoding="windows-1252" ?>
<odsxml>
  <thead>
    <meta operator="jusenp" />
  </thead>
  <body>
    <proc name="Print">
      <label name="IDK" />
      <title class="SystemTitle" toc-level="1">Revenue and Expense Totals for the #byval(region)
        Region</title>
      <branch class="ContentProcName" toc-level="1" label="Print">
        <leaf class="ContentItem" toc-level="2" label="Data Set WORK.EXPREV">
          <output name="Print" label="Data Set WORK.EXPREV" class="Data Set WORK.EXPREV">
            <output-object type="table" class="Table">
              <style>
                <border spacing="1" padding="7" ntees="GROUPS" frame="BOX" />
              </style>
              <colspec columns="4">
                <colgroup>
                  <colspec names="1" width="6" type="string" />
                  <colspec names="2" width="5" type="string" />
                  <colspec names="3" width="10" type="string" />
                  <colspec names="4" width="10" type="string" />
                </colgroup>
              </colspec>
              <output-head>
                <row>
                  <data type="string" class="Header" row="1" column="1">State</data>
                  <data type="string" class="Header" row="1" column="2">Month</data>
                  <data type="string" class="Header" row="1" column="3">Expenses</data>
                  <data type="string" class="Header" row="1" column="4">Revenues</data>
                </row>
              </output-head>
              ... more lines of XML output ...
            </output-object>
          </leaf>
        </branch>
      </proc>
    </body>
  </odsxml>

```

Example 5: Summing Numeric Variables with Multiple BY Variables

Procedure features:

BY statement

SUM statement

Other features: SORT procedure

Data set: EXPREV

This example

- sums expenses and revenues for
 - each region
 - each state with more than one row in the report

- all rows in the report.
- shows the number of observations in each BY group and in the whole report.

Program: Creating a Listing Report

```
options nodate pageno=1 linesize=70 pagesize=60;
```

Sort the data set. PROC SORT sorts the observations by Region and State.

```
proc sort data=exprev;
    by region state;
run;
```

Print the report, suppress the printing of observation numbers, and print the total number of observations for the selected variables. The N option prints the number of observations in a BY group at the end of that BY group and prints the total number of observations used in the report at the bottom of the report. NOOBS suppresses the printing of observation numbers at the beginning of the rows.

```
proc print data=exprev n noobs;
```

Create a separate section of the report for each BY group, and sum the values for the selected variables. The BY statement produces a separate section of the report for each BY group. The SUM statement alone sums the values of Expenses and Revenues for the entire data set. Because the program contains a BY statement, the SUM statement also sums the values of Expenses and Revenues for each BY group that contains more than one observation.

```
    by region state;
    sum expenses revenues;
```

Establish a label for a selected variable, format the values of specified variables, and create a title. The LABEL statement associates a label with the variable Region for the duration of the PROC PRINT step. The BY line at the beginning of each BY group uses the label. The FORMAT statement assigns a format to the variables Expenses and Revenues for this report. The TITLE statement specifies a title.

```
    label region='Sales Region';
    format revenues expenses comma10.;
    title 'Revenue and Expense Totals for Each State and Region';
run;
```

Output: Listing

Output 32.6 Summing Numeric Variables with Multiple BY Variables: Listing Output

The report uses default column headers (variable names) because neither the SPLIT= nor the LABEL option is used. Nevertheless, the BY line at the top of each section of the report shows the BY variables' labels and their values. The name of a BY variable identifies the subtotals in the report.

PROC PRINT sums Expenses and Revenues for each BY group that contains more than one observation. However, sums are shown only for the BY variables whose values change from one BY group to the next. For example, in the third BY group, where the sales region is **Southern** and the state is **FL**, Expenses and Revenues are summed only for the state because the next BY group is for the same region.

Revenue and Expense Totals for Each State and Region			1
----- Sales Region=Northern State=MA -----			
Month	Expenses	Revenues	
MAR95	1,500	1,000	
	N = 1		
----- Sales Region=Northern State=NY -----			
Month	Expenses	Revenues	
FEB95	3,000	4,000	
MAR95	6,000	5,000	
-----	-----	-----	
State	9,000	9,000	
Region	10,500	10,000	
	N = 2		
----- Sales Region=Southern State=FL -----			
Month	Expenses	Revenues	
FEB95	8,500	11,000	
MAR95	9,800	13,500	
-----	-----	-----	
State	18,300	24,500	
	N = 2		
----- Sales Region=Southern State=GA -----			
Month	Expenses	Revenues	
JAN95	2,000	8,000	
FEB95	1,200	6,000	
-----	-----	-----	
State	3,200	14,000	
Region	21,500	38,500	
	=====	=====	
	32,000	48,500	
	N = 2		
	Total N = 7		

Program: Creating an HTML Report

```
options nodate pageno=1 linesize=70 pagesize=60;
```

Produce HTML output and specify the file to store the output in. The ODS HTML statement opens the HTML destination and creates a file that contains HTML output. The FILE= argument specifies your external file that contains the HTML output.

```
ods html file='your_file.html';

proc sort data=exprev;
    by region state;
run;

proc print data=exprev n noobs;proc print
    by region state;
    sum expenses revenues;

    label region='Sales Region';
    format revenues expenses comma10.;
    title 'Revenue and Expense Totals for Each State and Region';
run;
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination.

```
ods html close;
```

Output: HTML

Display 32.9 Summing Numeric Variables with Multiple BY Variables: Default HTML Output

Revenue and Expense Totals for Each State and Region

Sales Region=Northern State=MA		
Month	Expenses	Revenues
MAR95	1,000	1,000
N = 1		
Sales Region=Northern State=NY		
Month	Expenses	Revenues
FEB95	3,000	4,000
MAR95	6,000	5,000
State	9,000	9,000
Region	10,500	10,000
N = 2		
Sales Region=Southern State=FL		
Month	Expenses	Revenues
FEB95	8,500	11,000
MAR95	9,800	13,500
State	12,300	24,500
N = 2		
Sales Region=Southern State=GA		
Month	Expenses	Revenues
JAN95	2,000	3,000
FEB95	1,200	5,000
State	3,200	14,000
Region	21,500	32,500
N = 2		
Total N = 7		

Program: Creating an HTML Report with the STYLE= Option

```
options nodate pageno=1 linesize=70 pagesize=60;

ods html file='your_file.html';

proc sort data=exprev;
  by region state;
run;

proc print data=exprev n noobs;
```

Create stylized HTML output. The STYLE= option in the first SUM statement specifies that the background color of the cell containing the grand total for the variable EXPENSES be changed to white and the font color be changed to dark gray.

The STYLE= option in the second SUM statement specifies that the background color of cells containing totals for the variable REVENUES be changed to blue and the font color be changed to white.

```
  by region state;
sum expenses / style(GRANDTOTAL) = [background =white foreground=blue];
sum revenues / style(TOTAL) = [background =dark gray foreground=white];

label region='Sales Region';
format revenues expenses comma10.;
```

```

        title 'Revenue and Expense Totals for Each State and Region';
run;

ods html close;

```

Output: HTML with Styles

Display 32.10 Summing Numeric Variables with Multiple BY Variables: HTML Output Using Styles

Revenue and Expense Totals for Each State and Region		
Sales Region=Northern State=MA		
Month	Expenses	Revenues
MAR95	1,000	1,000
N = 1		
Sales Region=Northern State=NY		
Month	Expenses	Revenues
FEB95	3,000	4,000
MAR95	6,000	5,000
State	9,000	9,000
Region	10,500	10,000
N = 2		
Sales Region=Southern State=FL		
Month	Expenses	Revenues
FEB95	8,500	11,000
MAR95	9,800	13,500
State	18,300	24,500
N = 2		
Sales Region=Southern State=GA		
Month	Expenses	Revenues
JAN95	2,000	8,000
FEB95	1,200	6,000
State	3,200	14,000
Region	21,500	38,500
N = 2		
Total N = 7		

Example 6: Limiting the Number of Sums in a Report

Features:

- BY statement
- SUM statement
- SUMBY statement

Other features:

- SORT procedure
- LABEL statement

Data set: EXPREV

This example

- creates a separate section of the report for each combination of state and region
- sums expenses and revenues only for each region and for all regions, not for individual states.

Program: Creating a Listing Report

```
options nodate pageno=1 linesize=70 pagesize=60;
```

Sort the data set. PROC SORT sorts the observations by Region and State.

```
proc sort data=exprev;
    by region state;
run;
```

Print the report and remove the observation numbers. NOOBS suppresses the printing of observation numbers at the beginning of the rows.

```
proc print data=exprev noobs;
```

Sum the values for each region. The SUM and BY statements work together to sum the values of Revenues and Expenses for each BY group as well as for the whole report. The SUMBY statement limits the subtotals to one for each region.

```
    by region state;
    sum revenues expenses;
    sumby region;
```

Assign labels to specific variables. The LABEL statement associates a label with the variable Region for the duration of the PROC PRINT step. This label is used in the BY lines.

```
    label region='Sales Region';
```

Assign a format to the necessary variables and specify a title. The FORMAT statement assigns the COMMA10. format to Expenses and Revenues for this report.

```
    format revenues expenses comma10.;
    title 'Revenue and Expense Figures for Each Region';
run;
```

Output: Listing

Output 32.7 Limiting the Number of Sums in a Report: Listing Output

The report uses default column headers (variable names) because neither the `SPLIT=` nor the `LABEL` option is used. Nevertheless, the `BY` line at the top of each section of the report shows the `BY` variables' labels and their values. The name of a `BY` variable identifies the subtotals in the report.

Revenue and Expense Figures for Each Region			1
----- Sales Region=Northern State=MA -----			
Month	Expenses	Revenues	
MAR95	1,500	1,000	
----- Sales Region=Northern State=NY -----			
Month	Expenses	Revenues	
FEB95	3,000	4,000	
MAR95	6,000	5,000	
-----	-----	-----	
Region	10,500	10,000	
----- Sales Region=Southern State=FL -----			
Month	Expenses	Revenues	
FEB95	8,500	11,000	
MAR95	9,800	13,500	
----- Sales Region=Southern State=GA -----			
Month	Expenses	Revenues	
JAN95	2,000	8,000	
FEB95	1,200	6,000	
-----	-----	-----	
Region	21,500	38,500	
	=====	=====	
	32,000	48,500	

Program: Creating a PostScript file

```
options nodate pageno=1 linesize=70 pagesize=60;
```

Produce PostScript output and specify the file to store the output in. The ODS PS statement opens the PS destination and creates a file that contains PostScript output. The `FILE=` argument specifies your external file that contains the PostScript output.

```
ods ps file='your_file.ps';
```

```
proc sort data=exprev;
  by region state;
```

```

run;

proc print data=exprev noobs;

by region state;
  sum revenues expenses;
  sumby region;

label region='Sales Region';

format revenues expenses comma10.;
title 'Revenue and Expense Figures for Each Region';
run;

```

Close the PS destination. The ODS PS CLOSE statement closes the PS destination.

```
ods ps close;
```

Output: PostScript

Display 32.11 Limiting the Number of Sums in a Report: PostScript Output

Revenue and Expense Figures for Each Region

Sales Region=Northern State=MA

Month	Expenses	Revenues
MAR95	1,500	1,000

Sales Region=Northern State=NY

Month	Expenses	Revenues
FEB95	3,000	4,000
MAR95	6,000	5,000
MAY65	10,500	10,000

Sales Region=Southern State=FL

Month	Expenses	Revenues
FEB95	8,500	11,000
MAR95	9,800	13,500

Sales Region=Southern State=GA

Month	Expenses	Revenues
JAN95	2,000	8,000
FEB95	1,200	6,000
MAY00	21,500	38,500
OCT05	32,000	48,500

Program: Creating a PostScript Report with the STYLE= Option

```
options nodate pageno=1 linesize=70 pagesize=60;
```

```
ods ps file='your_file.ps';
```

```
proc sort data=exprev;
  by region state;
run;
```

```
proc print data=exprev noobs;
```

```
  by region state;
```

Create stylized PostScript output. The STYLE= option in the first SUM statement specifies that the background color of cells containing totals for the variable REVENUES be changed to blue and the font color be changed to white.

The STYLE= option in the second SUM statement specifies that the background color of the cell containing the grand total for the EXPENSES variable be changed to white and the font color be changed to dark gray.

```
sum revenues / style(TOTAL) = [background =blue foreground=white];
sum expenses / style(GRANDTOTAL) = [background =white foreground=dark gray];
```

```
label region='Sales Region';
```

```
format revenues expenses comma10.;
title 'Revenue and Expense Figures for Each Region';
run;
```

```
ods ps close;
```


Output: PostScript with Styles

Display 32.12 Limiting the Number of Sums in a Report: PostScript Output Using Styles

Revenue and Expense Figures for Each Region

Sales Region=Northern State=MA

Month	Expenses	Revenues
MAR95	1,500	1,000

Sales Region=Northern State=NY

Month	Expenses	Revenues
FEB95	3,000	4,000
MAR95	6,000	5,000
Region	10,500	10,000

Sales Region=Southern State=FL

Month	Expenses	Revenues
FEB95	8,500	11,000
MAR95	9,800	13,500

Sales Region=Southern State=GA

Month	Expenses	Revenues
JAN95	2,000	8,000
FEB95	1,200	6,000
Region	21,500	38,500
	32,000	48,500

Example 7: Controlling the Layout of a Report with Many Variables

Procedure features:

PROC PRINT statement options:

ROWS=

ID statement options:

STYLE=

Other features:

ODS RTF statement

SAS data set options:

OBS=

This example shows two ways of printing a data set with a large number of variables: one is the default, and the other uses ROWS=. For detailed explanations of the layouts of these two reports, see the ROWS= option on page 824 and see “Page Layout” on page 834.

These reports use a pagesize of 24 and a linesize of 64 to help illustrate the different layouts.

Note: When the two reports are written as HTML output, they do not differ. △

Program: Creating a Listing Report

```
options nodate pageno=1 linesize=64 pagesize=24 ;
```

Create the EMPDATA data set. The data set EMPDATA contains personal and job-related information about a company's employees. A DATA step on page 1623 creates this data set.

```
data empdata;
  input IdNumber $ 1-4 LastName $ 9-19 FirstName $ 20-29
        City $ 30-42 State $ 43-44 /
        Gender $ 1 JobCode $ 9-11 Salary 20-29 @30 Birth date9.
        @43 Hired date9. HomePhone $ 54-65;
  format birth hired date9.;
  datalines;
1919   Adams      Gerald    Stamford    CT
M      TA2        34376    15SEP1948    07JUN1975    203/781-1255
1653   Alexander  Susan     Bridgeport  CT
F      ME2        35108    18OCT1952    12AUG1978    203/675-7715

. . . more lines of data . . .

1407   Grant      Daniel    Mt. Vernon  NY
M      PT1        68096    26MAR1957    21MAR1978    914/468-1616
1114   Green      Janice    New York    NY
F      TA2        32928    21SEP1957    30JUN1975    212/588-1092
;
```

Print only the first 12 observations in a data set. The OBS= data set option uses only the first 12 observations to create the report. (This is just to conserve space here.) The ID statement identifies observations with the formatted value of IdNumber rather than with the observation number. This report is shown in Example 7 on page 863.

```
proc print data=empdata(obs=12);
  id idnumber;
  title 'Personnel Data';
run;
```

Print a report that contains only one row of variables on each page. ROWS=PAGE prints only one row of variables for each observation on a page. This report is shown in Example 7 on page 863.

```
proc print data=empdata(obs=12) rows=page;
  id idnumber;
  title 'Personnel Data';
run;
```

Output: Listing

Output 32.8 Default Layout for a Report with Many Variables: Listing Output

In the traditional procedure output, each page of this report contains values for all variables in each observation. In the HTML output, this report is identical to the report that uses ROWS=PAGE.

Note that PROC PRINT automatically splits the variable names that are used as column headers at a change in capitalization if the entire name does not fit in the column. Compare, for example, the column headers for LastName (which fits in the column) and FirstName (which does not fit in the column).

Personnel Data						1
Id Number	LastName	First Name	City	State	Gender	
1919	Adams	Gerald	Stamford	CT	M	
1653	Alexander	Susan	Bridgeport	CT	F	
1400	Apple	Troy	New York	NY	M	
1350	Arthur	Barbara	New York	NY	F	
1401	Avery	Jerry	Paterson	NJ	M	
1499	Barefoot	Joseph	Princeton	NJ	M	
1101	Baucom	Walter	New York	NY	M	
Id Number	Job Code	Salary	Birth	Hired	HomePhone	
1919	TA2	34376	15SEP48	07JUN75	203/781-1255	
1653	ME2	35108	18OCT52	12AUG78	203/675-7715	
1400	ME1	29769	08NOV55	19OCT78	212/586-0808	
1350	FA3	32886	03SEP53	01AUG78	718/383-1549	
1401	TA3	38822	16DEC38	20NOV73	201/732-8787	
1499	ME3	43025	29APR42	10JUN68	201/812-5665	
1101	SCP	18723	09JUN50	04OCT78	212/586-8060	

Personnel Data						2
Id Number	LastName	First Name	City	State	Gender	
1333	Blair	Justin	Stamford	CT	M	
1402	Blalock	Ralph	New York	NY	M	
1479	Bostic	Marie	New York	NY	F	
1403	Bowden	Earl	Bridgeport	CT	M	
1739	Boyce	Jonathan	New York	NY	M	
Id Number	Job Code	Salary	Birth	Hired	HomePhone	
1333	PT2	88606	02APR49	13FEB69	203/781-1777	
1402	TA2	32615	20JAN51	05DEC78	718/384-2849	
1479	TA3	38785	25DEC56	08OCT77	718/384-8816	
1403	ME1	28072	31JAN57	24DEC79	203/675-3434	
1739	PT1	66517	28DEC52	30JAN79	212/587-1247	

Output 32.9 Layout Produced by the ROWS=PAGE Option: Listing Output

Each page of this report contains values for only some of the variables in each observation. However, each page contains values for more observations than the default report does.

Personnel Data						1
Id Number	LastName	First Name	City	State	Gender	
1919	Adams	Gerald	Stamford	CT	M	
1653	Alexander	Susan	Bridgeport	CT	F	
1400	Apple	Troy	New York	NY	M	
1350	Arthur	Barbara	New York	NY	F	
1401	Avery	Jerry	Paterson	NJ	M	
1499	Barefoot	Joseph	Princeton	NJ	M	
1101	Baucom	Walter	New York	NY	M	
1333	Blair	Justin	Stamford	CT	M	
1402	Blalock	Ralph	New York	NY	M	
1479	Bostic	Marie	New York	NY	F	
1403	Bowden	Earl	Bridgeport	CT	M	
1739	Boyce	Jonathan	New York	NY	M	

Personnel Data						2
Id Number	Job Code	Salary	Birth	Hired	HomePhone	
1919	TA2	34376	15SEP48	07JUN75	203/781-1255	
1653	ME2	35108	18OCT52	12AUG78	203/675-7715	
1400	ME1	29769	08NOV55	19OCT78	212/586-0808	
1350	FA3	32886	03SEP53	01AUG78	718/383-1549	
1401	TA3	38822	16DEC38	20NOV73	201/732-8787	
1499	ME3	43025	29APR42	10JUN68	201/812-5665	
1101	SCP	18723	09JUN50	04OCT78	212/586-8060	
1333	PT2	88606	02APR49	13FEB69	203/781-1777	
1402	TA2	32615	20JAN51	05DEC78	718/384-2849	
1479	TA3	38785	25DEC56	08OCT77	718/384-8816	
1403	ME1	28072	31JAN57	24DEC79	203/675-3434	
1739	PT1	66517	28DEC52	30JAN79	212/587-1247	

Program: Creating an RTF Report

```
options nodate pageno=1 linesize=64 pagesize=24;
```

```
data empdata;
  input IdNumber $ 1-4 LastName $ 9-19 FirstName $ 20-29
        City $ 30-42 State $ 43-44 /
        Gender $ 1 JobCode $ 9-11 Salary 20-29 @30 Birth date9.
        @43 Hired date9. HomePhone $ 54-65;
  format birth hired date9.;
```

```

      datalines;
1919   Adams      Gerald   Stamford   CT
M      TA2        34376   15SEP1948   07JUN1975   203/781-1255
1653   Alexander Susan    Bridgeport CT
F      ME2        35108   18OCT1952   12AUG1978   203/675-7715

      . . . more lines of data . . .

1407   Grant      Daniel   Mt. Vernon NY
M      PT1        68096   26MAR1957   21MAR1978   914/468-1616
1114   Green      Janice   New York   NY
F      TA2        32928   21SEP1957   30JUN1975   212/588-1092
;

```

Create output for Microsoft Word and specify the file to store the output in. The ODS RTF statement opens the RTF destination and creates output formatted for Microsoft Word. The FILE= argument specifies your external file that contains the RTF output.

```

ods rtf file='your_file.rtf';

proc print data=empdata(obs=12);
  id idnumber;
  title 'Personnel Data';
run;

```

Close the RTF destination. The ODS RTF CLOSE statement closes the RTF destination.

```
ods rtf close;
```

Output: RTF

Display 32.13 Layout for a Report with Many Variables: RTF Output

Personnel Data										
IdNumber	LastName	FirstName	City	State	Gender	JobCode	Salary	Birth	Hired	HomePhone
1919	Adams	Gerald	Stamford	CT	M	TA2	34376	15SEP48	07JUN75	203/781-1255
1653	Alexander	Susan	Bridgeport	CT	F	ME2	35108	18OCT52	12AUG78	203/675-7715
1400	Apple	Troy	New York	NY	M	ME1	29769	08NOV55	19OCT78	212/586-0888
1380	Arthur	Barbara	New York	NY	F	FA3	32886	03SEP53	01AUG78	718/383-1349
1401	Avery	Jerry	Paterson	NJ	M	TA3	38822	16DEC38	20NOV73	201/732-8787
1499	Barefoot	Joseph	Princeton	NJ	M	ME3	43025	29APR42	10JUN68	201/812-5665
1101	Beacon	Walter	New York	NY	M	SCP	18723	09JUN50	04OCT78	212/586-8060
1333	Blair	Justin	Stamford	CT	M	PT2	88406	02APR49	13FEB69	203/781-1777
1402	Blalock	Ralph	New York	NY	M	TA2	32615	20JAN51	05DEC78	718/384-2849
1479	Borlar	Mate	New York	NY	F	TA3	38785	25DEC56	08OCT77	718/384-8816
1403	Bowden	Ead	Bridgeport	CT	M	ME1	28972	31JAN57	24DEC79	203/675-3434
1739	Boyce	Jonathan	New York	NY	M	PT1	66517	28DEC52	30JAN79	212/387-1247

Program: Creating an RTF Report with the STYLE= Option

```
options nodate pageno=1 linesize=64 pagesize=24;
```

```

data empdata;
  input IdNumber $ 1-4 LastName $ 9-19 FirstName $ 20-29
        City $ 30-42 State $ 43-44 /
        Gender $ 1 JobCode $ 9-11 Salary 20-29 @30 Birth date9.
        @43 Hired date9. HomePhone $ 54-65;
  format birth hired date9.;
  datalines;
1919    Adams      Gerald    Stamford    CT
M      TA2        34376    15SEP1948    07JUN1975    203/781-1255
1653    Alexander  Susan    Bridgeport  CT
F      ME2        35108    18OCT1952    12AUG1978    203/675-7715

. . . more lines of data . . .

1407    Grant      Daniel    Mt. Vernon  NY
M      PT1        68096    26MAR1957    21MAR1978    914/468-1616
1114    Green      Janice    New York    NY
F      TA2        32928    21SEP1957    30JUN1975    212/588-1092
;

ods rtf file='your_file.rtf';

proc print data=empdata(obs=12);

```

Create stylized output for Microsoft Word.

```

id idnumber / style(DATA) =
               {background = red foreground = white}
style(HEADER) =
               {background = blue foreground = white};

title 'Personnel Data';
run;

ods rtf close;

```

Output: RTF with Styles

Display 32.14 Layout for a Report with Many Variables: RTF Output Using Styles

Personnel Data

IdNumber	LastName	FirstName	City	State	Gender	JobCode	Salary	Birth	Hired	HomePhone
1919	Adams	Gerald	Stanford	CT	M	TA2	34376	15SEP48	07JUN75	203/781-1255
1653	Alexander	Stanes	Bridgeport	CT	F	ME2	35108	18OCT52	12AUG78	203/675-7715
1400	Apple	Troy	New York	NY	M	ME1	29769	08NOV55	19OCT78	212/586-0808
1350	Arthur	Barbara	New York	NY	F	FA3	32836	03SEP53	01AUG78	718/583-1549
1401	Avvey	Jerry	Paterson	NJ	M	TA3	38822	16DEC38	20NOV73	201/732-8787
1499	Barefoot	Joseph	Princeton	NJ	M	ME3	43025	29APR42	10JUN68	201/812-5665
1101	Bascom	Walter	New York	NY	M	SCP	18723	09JUN50	04OCT78	212/586-8060
1335	Blair	Justin	Stanford	CT	M	PT2	88606	02APR49	13FEB69	203/781-1777
1402	Blalock	Ralph	New York	NY	M	TA2	32615	20JAN51	05DEC78	718/584-2849
1479	Bostic	Mate	New York	NY	F	TA3	38785	25DEC56	08OCT77	718/584-8816
1403	Bowden	Ead	Bridgeport	CT	M	ME1	28072	31JAN57	24DEC79	203/675-3434
1739	Bryce	Jonathan	New York	NY	M	PT1	66517	28DEC52	30JAN79	212/587-1247

Example 8: Creating a Customized Layout with BY Groups and ID Variables

Procedure features:

BY statement

ID statement

SUM statement

VAR statement

Other features:

SORT procedure

Data set: EMPDATA on page 864

This customized report

- selects variables to include in the report and controls their order
- selects observations to include in the report
- groups the selected observations by JobCode
- sums the salaries for each job code and for all job codes
- displays numeric data with commas and dollar signs.

Program: Creating a Listing Report

Create and sort a temporary data set. PROC SORT creates a temporary data set in which the observations are sorted by JobCode and Gender.

```
options nodate pageno=1 linesize=64 pagesize=60;
proc sort data=empdata out=tempemp;
    by jobcode gender;
run;
```

Identify the character that starts a new line in column headers. SPLIT= identifies the asterisk as the character that starts a new line in column headers.

```
proc print data=tempemp split='*';
```

Specify the variables to include in the report. The VAR statement and the ID statement together select the variables to include in the report. The ID statement and the BY statement produce the special format.

```
id jobcode;
by jobcode;
var gender salary;
```

Calculate the total value for each BY group. The SUM statement totals the values of Salary for each BY group and for the whole report.

```
sum salary;
```

Assign labels to the appropriate variables. The LABEL statement associates a label with each variable for the duration of the PROC PRINT step. When you use SPLIT= in the PROC PRINT statement, the procedure uses labels for column headings.

```
label jobcode='Job Code*====='
gender='Gender*====='
salary='Annual Salary*=====';
```

Create formatted columns. The FORMAT statement assigns a format to Salary for this report. The WHERE statement selects for the report only the observations for job codes that contain the letters 'FA' or 'ME'. The TITLE statements specify two titles.

```
format salary dollar11.2;
where jobcode contains 'FA' or jobcode contains 'ME';
title 'Expenses Incurred for';
title2 'Salaries for Flight Attendants and Mechanics';
run;
```

Output: Listing

Output 32.10 Creating a Customized Layout with BY Groups and ID Variables:
Listing Output

The ID and BY statements work together to produce this layout. The ID variable is listed only once for each BY group. The BY lines are suppressed. Instead, the value of the ID variable, JobCode, identifies each BY group.

Expenses Incurred for Salaries for Flight Attendants and Mechanics			1
Job Code =====	Gender =====	Annual Salary =====	
FA1	F	\$23,177.00	
	F	\$22,454.00	
	M	\$22,268.00	
-----		-----	
FA1		\$67,899.00	
FA2	F	\$28,888.00	
	F	\$27,787.00	
	M	\$28,572.00	
-----		-----	
FA2		\$85,247.00	
FA3	F	\$32,886.00	
	F	\$33,419.00	
	M	\$32,217.00	
-----		-----	
FA3		\$98,522.00	
ME1	M	\$29,769.00	
	M	\$28,072.00	
	M	\$28,619.00	
-----		-----	
ME1		\$86,460.00	
ME2	F	\$35,108.00	
	F	\$34,929.00	
	M	\$35,345.00	
	M	\$36,925.00	
	M	\$35,090.00	
	M	\$35,185.00	
-----		-----	
ME2		\$212,582.00	
ME3	M	\$43,025.00	
		=====	
		\$593,735.00	

Program: Creating an HTML Report

```
options nodate pageno=1 linesize=64 pagesize=60;
proc sort data=empdata out=tempemp;
  by jobcode gender;
run;
```

Produce HTML output and specify the file to store the output in. The ODS HTML statement opens the HTML destination and creates a file that contains HTML output. The FILE= argument specifies your external file that contains the HTML output.

```
ods html file='your_file.html';

proc print data=tempemp split='*';

  id jobcode;
  by jobcode;
  var gender salary;

  sum salary;

  label jobcode='Job Code*====='
        gender='Gender*====='
        salary='Annual Salary*=====';

  format salary dollar11.2;
  where jobcode contains 'FA' or jobcode contains 'ME';
  title 'Expenses Incurred for';
  title2 'Salaries for Flight Attendants and Mechanics';
run;
```

Close the HTML destination. The ODS HTML CLOSE statement closes the HTML destination.

```
ods html close;
```

Output: HTML

Display 32.15 Creating a Customized Layout with BY Groups and ID Variables: Default HTML Output

Expenses Incurred for Salaries for Flight Attendants and Mechanics		
Job Code	Gender	Annual Salary
FA1	F	\$23,177.00
	F	\$22,464.00
	M	\$22,269.00
FA1		\$67,899.00
FA2	F	\$29,898.00
	F	\$27,787.00
	M	\$28,572.00
FA2		\$85,247.00
FA3	F	\$32,886.00
	F	\$32,419.00
	M	\$32,217.00
FA3		\$98,522.00
ME1	M	\$29,769.00
	M	\$28,872.00
	M	\$28,819.00
ME1		\$86,460.00
ME2	F	\$35,105.00
	F	\$34,929.00
	M	\$35,345.00
	M	\$36,925.00
	M	\$35,090.00
	M	\$35,165.00
ME2		\$212,582.00
ME3	M	\$43,025.00
		\$693,735.00

Program: Creating an HTML Report with the STYLE= Option

```
options nodate pageno=1 linesize=64 pagesize=60;
proc sort data=empdata out=tempemp;
  by jobcode gender;
```

```
run;
```

```
ods html file='your_file.html';
```

Create stylized HTML output. The first STYLE= option specifies that the font of the headers be changed to italic. The second STYLE= option specifies that the background of cells that contain input data be changed to blue and the foreground of these cells be changed to white.

```
proc print data=tempemp split='*' style(HEADER) =
                                {font_style=italic}
                                style(DATA) =
                                {background=blue foreground = white};
```

```
id jobcode;
  by jobcode;
  var gender salary;
```

Create total values that are written in red. The STYLE= option specifies that the color of the foreground of the cell that contain the totals be changed to red.

```
sum salary / style(total)= [foreground=red];

label jobcode='Job Code*======'
      gender='Gender*======'
      salary='Annual Salary*=====';

format salary dollar11.2;
where jobcode contains 'FA' or jobcode contains 'ME';
title 'Expenses Incurred for';
title2 'Salaries for Flight Attendants and Mechanics';
run;

ods html close;
```

Output: HTML with Styles

Display 32.16 Creating a Customized Layout with BY Groups and ID Variables: HTML Output Using Styles

Expenses Incurred for Salaries for Flight Attendants and Mechanics		
Job Code	Gender	Annual Salary
PA1	F	\$23,177.00
	F	\$22,454.00
	M	\$22,266.00
PA1		\$67,899.00
PA2	F	\$28,636.00
	F	\$27,787.00
	M	\$28,572.00
PA2		\$85,247.00
PA3	F	\$32,886.00
	F	\$33,419.00
	M	\$32,217.00
PA3		\$98,522.00
ME1	M	\$21,766.00
	M	\$20,092.00
	M	\$20,619.00
ME1		\$66,460.00
ME2	F	\$25,108.00
	F	\$24,929.00
	M	\$25,245.00
	M	\$26,025.00
	M	\$25,099.00
	M	\$25,185.00
ME2		\$212,582.00
ME3	M	\$43,025.00
		\$593,735.00

Example 9: Printing All the Data Sets in a SAS Library

Features:

Macro facility

DATASETS procedure

PRINT procedure

Data set: EXPREV and LIST on page 506

This example prints all the data sets in a SAS library. You can use the same programming logic with any procedure. Just replace the PROC PRINT step near the end of the example with whatever procedure step you want to execute. The example uses the macro language. For details about the macro language, see *SAS Guide to Macro Processing, Version 6, Second Edition*.

Program

```
libname printlib 'SAS-data-library'
options nodate pageno=1 linesize=80 pagesize=60;
```

Copy the desired data sets from the WORK library to a permanent library. PROC DATASETS copies two data sets from the WORK library to the PRINTLIB library in order to limit the number of data sets available to the example.

```
proc datasets library=work memtype=data nolist;
    copy out=printlib;
        select list exprev;
run;
```

Create a macro and specify the parameters. The %MACRO statement creates the macro PRINTALL. When you call the macro, you can pass one or two parameters to it. The first parameter is the name of the library whose data set you want to print. The second parameter is a library used by the macro. If you do not specify this parameter, the WORK library is the default.

```
%macro printall(libname,worklib=work);
```

Create the local macro variables. The %LOCAL statement creates two local macro variables, NUM and I, to use in a loop.

```
%local num i;
```

Produce an output data set. This PROC DATASETS step reads the library that you specify as a parameter when you invoke the macro. The CONTENTS statement produces an output data set called TEMP1 in WORKLIB. This data set contains an observation for each variable in each data set in the library LIBNAME. By default, each observation includes the name of the data set that the variable is included in as well as other information about the variable. However, the KEEP= data set option writes only the name of the data set to TEMP1.

```
proc datasets library=&libname memtype=data nodetails;
    contents out=&worklib..temp1(keep=memname) data=_all_ noprint;
```

```
run;
```

Specify the unique values in the data set, assign a macro variable to each one, and assign DATA step information to a macro variable. This DATA step increments the value of N each time it reads the last occurrence of a data set name (when IF LAST.MEMNAME is true). The CALL SYMPUT statement uses the current value of N to create a macro variable for each unique value of MEMNAME in the data set TEMP1. The TRIM function removes extra blanks in the TITLE statement in the PROC PRINT step that follows.

```
data _null_;
  set &worklib..temp1 end=final;
  by memname notsorted;
  if last.memname;
  n+1;
  call symput('ds' || left(put(n,8.)),trim(memname));
```

When it reads the last observation in the data set (when FINAL is true), the DATA step assigns the value of N to the macro variable NUM. At this point in the program, the value of N is the number of observations in the data set.

```
if final then call symput('num',put(n,8.));
```

Run the DATA step. The RUN statement is crucial. It forces the DATA step to run, thus creating the macro variables that are used in the CALL SYMPUT statements before the %DO loop, which uses them, executes.

```
run;
```

Print the data sets and end the macro. The %DO loop issues a PROC PRINT step for each data set. The %MEND statement ends the macro.

```
%do i=1 %to &num;
  proc print data=&libname..&ds&i noobs;
    title "Data Set &libname..&ds&i";
  run;
%end;
%mend printall;
```

Print all the data sets in the PRINTLIB library. This invocation of the PRINTALL macro prints all the data sets in the library PRINTLIB.

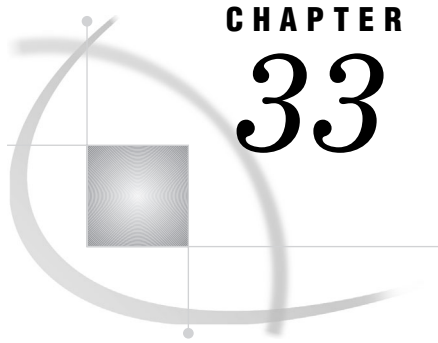
```
options nodate pageno=1 linesize=70 pagesize=60;
%printall(printlib)
```

Output

Output 32.11 Printing All the Data Sets in a SAS Library: Listing Output

Data Set printlib.EXPREV					1
Region	State	Month	Expenses	Revenues	
Northern	MA	MAR95	1500	1000	
Northern	NY	FEB95	3000	4000	
Northern	NY	MAR95	6000	5000	
Southern	FL	FEB95	8500	11000	
Southern	FL	MAR95	9800	13500	
Southern	GA	JAN95	2000	8000	
Southern	GA	FEB95	1200	6000	

Data Set printlib.LIST					2
Name	Street	City	State	Zip	
Gabrielli, Theresa	24 Ridgetop Rd.	Westboro	MA	01581	
Clayton, Aria	314 Bridge St.	Hanover	NH	03755	
Dix, Martin L.	4 Shepherd St.	Norwich	VT	05055	
Slater, Emily C.	2009 Cherry St.	York	PA	17407	
Ericson, Jane	211 Clancey Court	Chapel Hill	NC	27514	
An, Ing	95 Willow Dr.	Charlotte	NC	28211	
Jacobson, Becky	7 Lincoln St.	Tallahassee	FL	32312	
Misiewicz, Jeremy	43-C Lakeview Apts.	Madison	WI	53704	
Ahmadi, Hafez	5203 Marston Way	Boulder	CO	80302	
Archuleta, Ruby	Box 108	Milagro	NM	87429	



The PRINTTO Procedure

<i>Overview: PRINTTO Procedure</i>	879
<i>Syntax: PRINTTO Procedure</i>	880
<i>PROC PRINTTO Statement</i>	880
<i>Concepts: PRINTTO Procedure</i>	883
<i>Page Numbering</i>	883
<i>Routing SAS Log or Procedure Output Directly to a Printer</i>	883
<i>Examples: PRINTTO Procedure</i>	883
<i>Example 1: Routing to External Files</i>	883
<i>Example 2: Routing to SAS Catalog Entries</i>	886
<i>Example 3: Using Procedure Output as an Input File</i>	889
<i>Example 4: Routing to a Printer</i>	892

Overview: PRINTTO Procedure

The PRINTTO procedure defines destinations for SAS procedure output and for the SAS log. By default, SAS procedure output and the SAS log are routed to the default procedure output file and the default SAS log file for your method of operation. See Table 33.1 on page 879. You can store the SAS log or procedure output in an external file or in a SAS catalog entry. With additional programming, you can use SAS output as input data within the same job.

Table 33.1 Default Destinations for SAS Log and Procedure Output

Method of running the SAS System	SAS log destination	Procedure output destination
windowing environment	the LOG window	the OUTPUT window
interactive line mode	the display monitor (as statements are entered)	the display monitor (as each step executes)
noninteractive mode or batch mode	depends on the host operating system	depends on the operating environment

Operating Environment Information: For information and examples specific to your operating system or environment, see the appropriate SAS Companion or technical report. Δ

Syntax: PRINTTO Procedure

PROC PRINTTO *<option(s)>*;

PROC PRINTTO Statement

Tip: To reset the destination for the SAS log and procedure output to the default, use the PROC PRINTTO statement without options.

Tip: To route the SAS log and procedure output to the same file, specify the same file with both the LOG= and PRINT= options.

Restriction: To route SAS log and procedure output directly to a printer, you must use a FILENAME statement with the PROC PRINTTO statement. See Example 4 on page 892.

PROC PRINTTO *<option(s)>*;

To do this	Use this option
provide a description for a SAS log or procedure output stored in a SAS catalog entry	LABEL=
route the SAS log to a permanent external file or SAS catalog entry	LOG=
combine the SAS log and procedure output into a single file	LOG= and PRINT= with same destination
replace the file instead of appending to it	NEW
route procedure output to a permanent external file or SAS catalog entry or printer.	PRINT=

Without Options

Using a PROC PRINTTO statement with no options

- closes any files opened by a PROC PRINTTO statement
- points both the SAS log and SAS procedure output to their default destinations.

Interaction: To close the appropriate file and to return only the SAS log or procedure output to its default destination, use LOG=LOG or PRINT=PRINT.

Featured in: Example 1 on page 883 and Example 2 on page 886

Options

LABEL=*'description'*

provides a description for a catalog entry that contains a SAS log or procedure output.

Range: 1 to 256 characters

Interaction: Use the LABEL= option only when you specify a catalog entry as the value for the LOG= or the PRINT= option.

Featured in: Example 2 on page 886

LOG=LOG | *file-specification* | *SAS-catalog-entry*

routes the SAS log to one of three locations:

LOG

routes the SAS log to its default destination.

file-specification

routes the SAS log to an external file. It is one of the following:

'external-file'

the name of an external file specified in quotation marks.

fileref

a fileref previously assigned to an external file.

SAS-catalog-entry

routes the SAS log to a SAS catalog entry. By default, *libref* is SASUSER, *catalog* is PROFILE, and *type* is LOG. Express *SAS-catalog-entry* in one of the following ways:

libref.catalog.entry<LOG>

a SAS catalog entry stored in the SAS data library and SAS catalog specified.

catalog.entry<LOG>

a SAS catalog entry stored in the specified SAS catalog in the default SAS data library SASUSER.

entry.LOG

a SAS catalog entry stored in the default SAS library and catalog:
SASUSER.PROFILE.

fileref

a fileref previously assigned to a SAS catalog entry. Search for "FILENAME, CATALOG Access Method" in the SAS online documentation.

Default: LOG.

Tip: After routing the log to an external file or a catalog entry, you can specify LOG to route the SAS log back to its default destination.

Tip: When routing the SAS log, include a RUN statement in the PROC PRINTTO statement. If you omit the RUN statement, the first line of the following DATA or PROC step is not routed to the new file. (This occurs because a statement does not execute until a step boundary is crossed.)

Interaction: The SAS log and procedure output cannot be routed to the same catalog entry at the same time.

Interaction: The NEW option replaces the existing contents of a file with the new log. Otherwise, the new log is appended to the file.

Interaction: To route the SAS log and procedure output to the same file, specify the same file with both the LOG= and PRINT= options.

Interaction: When routing the log to a SAS catalog entry, you can use the LABEL option to provide a description for the entry in the catalog directory.

Featured in: Example 1 on page 883, Example 2 on page 886, and Example 3 on page 889

NEW

clears any information that exists in a file and prepares the file to receive the SAS log or procedure output.

Default: If you omit NEW, the new information is appended to the existing file.

Interaction: If you specify both LOG= and PRINT=, NEW applies to both.

Featured in: Example 1 on page 883, Example 2 on page 886, and Example 3 on page 889

PRINT= PRINT | *file-specification* | *SAS-catalog-entry*

routes procedure output to one of three locations:

PRINT

routes procedure output to its default destination. After routing it to an external file or a catalog entry, you can specify PRINT to route subsequent procedure output to its default destination.

file-specification

routes procedure output to an external file. It is one of the following:

'external-file'

the name of an external file specified in quotation marks.

fileref

a fileref previously assigned to an external file.

SAS-catalog-entry

routes procedure output to a SAS catalog entry. By default, *libref* is SASUSER, *catalog* is PROFILE, and *type* is OUTPUT. Express *SAS-catalog-entry* in one of the following ways:

libref.catalog.entry<.OUTPUT>

a SAS catalog entry stored in the SAS data library and SAS catalog specified.

catalog.entry<.OUTPUT>

a SAS catalog entry stored in the specified SAS catalog in the default SAS data library SASUSER.

entry.OUTPUT

a SAS catalog entry stored in the default SAS library and catalog:
SASUSER.PROFILE.

fileref

a fileref previously assigned to a SAS catalog entry. Search for "FILENAME, CATALOG Access Method" in the SAS online documentation.

Aliases: FILE=, NAME=

Default: PRINT

Interaction: The procedure output and the SAS log cannot be routed to the same catalog entry at the same time.

Interaction: The NEW option replaces the existing contents of a file with the new procedure output. If you omit NEW, the new output is appended to the file.

Interaction: To route the SAS log and procedure output to the same file, specify the same file with both the LOG= and PRINT= options.

Interaction: When routing procedure output to a SAS catalog entry, you can use the LABEL option to provide a description for the entry in the catalog directory.

Featured in: Example 3 on page 889

UNIT=*nn*

routes the output to the file identified by the fileref FTnnF001, where *nn* is an integer between 1 and 99.

Range: 1 to 99, integer only.

Tip: You can define this fileref yourself; however, some operating systems predefine certain filerefs in this form.

Concepts: PRINTTO Procedure

Page Numbering

- When the SAS system option NUMBER is in effect, there is a single page-numbering sequence for all output in the current job or session. When NONUMBER is in effect, output pages are not numbered.
- You can specify the beginning page number for the output you are currently producing by using the PAGENO= in an OPTIONS statement.

Routing SAS Log or Procedure Output Directly to a Printer

To route SAS log or procedure output directly to a printer, use a FILENAME statement to associate a fileref with the printer name, and then use that fileref in the LOG= or PRINT= option. For an example, see Example 4 on page 892.

For more information see the FILENAME statement in *SAS Language Reference: Dictionary*.

Operating Environment Information: For examples of printer names, see the documentation for your operating system. △

Examples: PRINTTO Procedure

Example 1: Routing to External Files

Procedure features:

PRINTTO statement:

Without options

Options:

LOG=

NEW

PRINT=

This example uses PROC PRINTTO to route the log and procedure output to an external file and then reset both destinations to the default.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. The SOURCE option writes lines of source code to the default destination for the SAS log.

```
options nodate pageno=1 linesize=80 pagesize=60 source;
```

Route the SAS log to an external file. PROC PRINTTO uses the LOG= option to route the SAS log to an external file. By default, this log is appended to the current contents of **log-file**.

```
proc printto log='log-file';
run;
```

Create the NUMBERS data set. The DATA step uses list input to create the NUMBERS data set.

```
data numbers;
    input x y z;
    datalines;
14.2    25.2    96.8
10.8    51.6    96.8
 9.5    34.2   138.2
 8.8    27.6    83.2
11.5    49.4   287.0
 6.3    42.0   170.7
;
```

Route the procedure output to an external file. PROC PRINTTO routes output to an external file. Because NEW is specified, any output written to **output-file** will overwrite the file's current contents.

```
proc printto print='output-file' new;
run;
```

Print the NUMBERS data set. The PROC PRINT output is written to the specified external file.

```
proc print data=numbers;
    title 'Listing of NUMBERS Data Set';
run;
```

Reset the SAS log and procedure output destinations to default. PROC PRINTTO routes subsequent logs and procedure output to their default destinations and closes both of the current files.

```
proc printto;
run;
```

Log

Output 33.1 Portion of Log Routed to the Default Destination

```
1      options nodate pageno=1 linesize=80 pagesize=60 source;
2      proc printto log='log-file';
3      run;
```

Output 33.2 Portion of Log Routed to an External File

```
5
6      data numbers;
7      input x y z;
8      datalines;

NOTE: The data set WORK.NUMBERS has 6 observations and 3 variables.
NOTE: DATA statement used:
      real time          0.00 seconds
      cpu time           0.00 seconds

15     ;
16     proc printto print='output-file' new;
16
17     run;

NOTE: PROCEDURE PRINTTO used:
      real time          0.00 seconds
      cpu time           0.00 seconds

18
19     proc print data=numbers;
20     title 'Listing of NUMBERS Data Set';
21     run;

NOTE: The PROCEDURE PRINT printed page 1.
NOTE: PROCEDURE PRINT used:
      real time          0.00 seconds
      cpu time           0.00 seconds

22
23     proc printto;
24     run;
```

Output

Output 33.3 Procedure Output Routed to an External File

Listing of NUMBERS Data Set				1
OBS	x	y	z	
1	14.2	25.2	96.8	
2	10.8	51.6	96.8	
3	9.5	34.2	138.2	
4	8.8	27.6	83.2	
5	11.5	49.4	287.0	
6	6.3	42.0	170.7	

Example 2: Routing to SAS Catalog Entries

Procedure features:

PRINTTO statement:

Without options

Options:

LABEL=

LOG=

NEW

PRINT=

This example uses PROC PRINTTO to route the SAS log and procedure output to a SAS catalog entry and then to reset both destinations to the default.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60 source;
```

Assign a libname.

```
libname lib1 'SAS-data-library';
```

Route the SAS log to a SAS catalog entry. PROC PRINTTO routes the SAS log to a SAS catalog entry named SASUSER.PROFILE.TEST.LOG. The PRINTTO procedure uses the default libref and catalog SASUSER.PROFILE because only the entry name and type are specified. LABEL= assigns a description for the catalog entry.


```
proc printto log=test.log label='Inventory program' new;
run;
```

Create the LIB1.INVENTORY data set. The DATA step creates a permanent SAS data set.

```
data lib1.inventory;
    length Dept $ 4 Item $ 6 Season $ 6 Year 4;
    input dept item season year @@;
    datalines;
3070 20410   spring 1996 3070 20411   spring 1997
3070 20412   spring 1997 3070 20413   spring 1997
3070 20414   spring 1996 3070 20416   spring 1995
3071 20500   spring 1994 3071 20501   spring 1995
3071 20502   spring 1996 3071 20503   spring 1996
3071 20505   spring 1994 3071 20506   spring 1994
3071 20507   spring 1994 3071 20424   spring 1994
;
```

Route the procedure output to a SAS catalog entry. PROC PRINTTO routes procedure output from the subsequent PROC REPORT step to the SAS catalog entry LIB1.CAT1.INVENTORY.OUTPUT. LABEL= assigns a description for the catalog entry.

```
proc printto print=lib1.cat1.inventory.output
              label='Inventory program' new;
run;

proc report data=lib1.inventory nowindows headskip;
    column dept item season year;
    title 'Current Inventory Listing';
run;
```

Reset the SAS log and procedure output back to the default and close the file. PROC PRINTTO closes the current files that were opened by the previous PROC PRINTTO step and reroutes subsequent SAS logs and procedure output to their default destinations.

```
proc printto;
run;
```

Log

Output 33.4 SAS Log Routed to SAS Catalog Entry SASUSER.PROFILE.TEST.LOG.

You can view this catalog entry in the BUILD window of the SAS Explorer.

```
8
9   data lib1.inventory;
10      length Dept $ 4 Item $ 6 Season $ 6 Year 4;
11      input dept item season year @@;
12      datalines;

NOTE: SAS went to a new line when INPUT statement reached past the end of a
      line.
NOTE: The data set LIB1.INVENTORY has 14 observations and 4 variables.
NOTE: DATA statement used:
      real time           0.00 seconds
      cpu time            0.00 seconds

20  ;
21
22  proc printto print=lib1.cat1.inventory.output
23      label='Inventory program' new;
24  run;

NOTE: PROCEDURE PRINTTO used:
      real time           0.00 seconds
      cpu time            0.00 seconds

25
26  proc report data=lib1.inventory nowindows headskip;
27      column dept item season year;
28      title 'Current Inventory Listing';
29  run;

NOTE: PROCEDURE REPORT used:
      real time           0.00 seconds
      cpu time            0.00 seconds

30
31  proc printto;
32  run;
```

Output

Output 33.5 Procedure Output Routed to SAS Catalog Entry LIB1.CAT1.INVENTORY.OUTPUT.

You can view this catalog entry in the BUILD window of the SAS Explorer.

Current Inventory Listing				1
Dept	Item	Season	Year	
3070	20410	spring	1996	
3070	20411	spring	1997	
3070	20412	spring	1997	
3070	20413	spring	1997	
3070	20414	spring	1996	
3070	20416	spring	1995	
3071	20500	spring	1994	
3071	20501	spring	1995	
3071	20502	spring	1996	
3071	20503	spring	1996	
3071	20505	spring	1994	
3071	20506	spring	1994	
3071	20507	spring	1994	
3071	20424	spring	1994	

Example 3: Using Procedure Output as an Input File

Procedure features:

PRINTTO statement:

Without options

Options:

LOG=

NEW

PRINT=

This example uses PROC PRINTTO to route procedure output to an external file and then uses that file as input to a DATA step.

Generate random values for the variables. The DATA step uses the RANUNI function to randomly generate values for the variables X and Y in the data set A.

```
data test;
  do n=1 to 1000;
    x=int(ranuni(77777)*7);
    y=int(ranuni(77777)*5);
    output;
  end;
run;
```

Assign a fileref and route procedure output to the file that is referenced. The FILENAME statement assigns a fileref to an external file. PROC PRINTTO routes subsequent procedure output to the file that is referenced by the fileref ROUTED. See Output 33.6 on page 890.

```
filename routed 'output-filename';

proc printto print=routed new;
run;
```

Produce the frequency counts. PROC FREQ computes frequency counts and a chi-square analysis of the variables X and Y in the data set TEST. This output is routed to the file that is referenced as ROUTED.

```
proc freq data=test;
    tables x*y / chisq;
run;
```

Close the file. You must use another PROC PRINTTO to close the file that is referenced by fileref ROUTED so that the following DATA step can read it. The step also routes subsequent procedure output to the default destination. PRINT= causes the step to affect only procedure output, not the SAS log.

```
proc printto print=print;
run;
```

Create the data set PROBTTEST. The DATA step uses ROUTED, the file containing PROC FREQ output, as an input file and creates the data set PROBTTEST. This DATA step reads all records in ROUTED but creates an observation only from a record that begins with **Chi-Squa**.

```
data probtest;
    infile routed;
    input word1 $ @;
    if word1='Chi-Squa' then
        do;
            input df chisq prob;
            keep chisq prob;
            output;
        end;
run;
```

Print the PROBTTEST data set. PROC PRINT produces a simple listing of data set PROBTTEST. This output is routed to the default destination. See Output 33.7 on page 891.

```
proc print data=probtest;
    title 'Chi-Square Analysis for Table of X by Y';
run;
```

Output 33.6 PROC FREQ Output Routed to the External File Referenced as ROUTED

The FREQ Procedure						
Table of x by y						
x	y					
Frequency						
Percent						
Row Pct						
Col Pct	0	1	2	3	4	Total
0	29	33	12	25	27	126
	2.90	3.30	1.20	2.50	2.70	12.60
	23.02	26.19	9.52	19.84	21.43	
	15.18	16.18	6.25	11.74	13.50	
1	23	26	29	20	19	117
	2.30	2.60	2.90	2.00	1.90	11.70
	19.66	22.22	24.79	17.09	16.24	
	12.04	12.75	15.10	9.39	9.50	
2	28	26	32	30	25	141
	2.80	2.60	3.20	3.00	2.50	14.10
	19.86	18.44	22.70	21.28	17.73	
	14.66	12.75	16.67	14.08	12.50	
3	26	24	36	32	45	163
	2.60	2.40	3.60	3.20	4.50	16.30
	15.95	14.72	22.09	19.63	27.61	
	13.61	11.76	18.75	15.02	22.50	
4	25	31	28	36	29	149
	2.50	3.10	2.80	3.60	2.90	14.90
	16.78	20.81	18.79	24.16	19.46	
	13.09	15.20	14.58	16.90	14.50	
5	32	29	26	33	27	147
	3.20	2.90	2.60	3.30	2.70	14.70
	21.77	19.73	17.69	22.45	18.37	
	16.75	14.22	13.54	15.49	13.50	
6	28	35	29	37	28	157
	2.80	3.50	2.90	3.70	2.80	15.70
	17.83	22.29	18.47	23.57	17.83	
	14.66	17.16	15.10	17.37	14.00	
Total	191	204	192	213	200	1000
	19.10	20.40	19.20	21.30	20.00	100.00

2

The FREQ Procedure

Statistics for Table of x by y

Statistic	DF	Value	Prob
Chi-Square	24	27.2971	0.2908
Likelihood Ratio Chi-Square	24	28.1830	0.2524
Mantel-Haenszel Chi-Square	1	0.6149	0.4330
Phi Coefficient		0.1652	
Contingency Coefficient		0.1630	
Cramer's V		0.0826	

Sample Size = 1000

Output 33.7 PROC PRINT Output of Data Set PROBTTEST, Routed to Default Destination

Chi-Square Analysis for Table of X by Y			3
Obs	chisq	prob	
1	27.297	0.291	

Example 4: Routing to a Printer

Procedure features:

PRINTTO statement:

Option:

PRINT= option

This example uses PROC PRINTTO to route procedure output directly to a printer.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

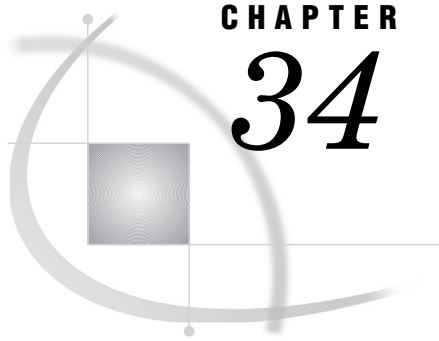
```
options nodate pageno=1 linesize=80 pagesize=60;
```

Associate a fileref with the printer name. The FILENAME statement associates a fileref with the printer name that you specify. If you want to associate a fileref with the default printer, omit *'printer-name'*.

```
filename your_fileref printer 'printer-name';
```

Specify the file to route to the printer. The PRINT= option specifies the file that PROC PRINTTO routes to the printer.

```
proc printto print=your_fileref;
run;
```



CHAPTER 34

The PRTDEF Procedure

<i>Overview: PRTDEF Procedure</i>	893
<i>Syntax: PRTDEF Procedure</i>	893
<i>PROC PRTDEF Statement</i>	893
<i>Input Data Set: PRTDEF Procedure</i>	895
<i>Summary of Valid Variables</i>	895
<i>Required Variables</i>	896
<i>Optional Variables</i>	897
<i>Examples: PRTDEF Procedure</i>	899
<i>Example 1: Defining Multiple Printer Definitions</i>	899
<i>Example 2: Creating a Ghostview Printer in SASUSER to Preview PostScript Printer Output in SASUSER</i>	900
<i>Example 3: Creating a Single Printer Definition That Is Available to All Users</i>	901
<i>Example 4: Adding, Modifying, and Deleting Printer Definitions</i>	902
<i>See Also</i>	905

Overview: PRTDEF Procedure

The PRTDEF procedure creates printer definitions in batch mode either for an individual user or for all SAS users at your site. Your system administrator can create printer definitions in the SAS registry and make these printers available to all SAS users at your site by using PROC PRTDEF with the USESASHELP option. An individual user can create personal printer definitions in the SAS registry by using PROC PRTDEF.

Syntax: PRTDEF Procedure

```
PROC PRTDEF <option(s)>;
```

PROC PRTDEF Statement

```
PROC PRTDEF <option(s)>;
```

To do this	Use this option
Specify the input data set that contains the printer attributes	DATA=
Specify that the default operation is to delete the printer definitions from the registry	DELETE
Specify that the registry entries are being created for export to a different host	FOREIGN
Specify that a list of printers that are created or replaced will be written to the log	LIST
Specify that any printer name that already exists will be modified by using the information in the printer attributes data set	REPLACE
Specify whether the printer definitions are available to all users or just the users running PROC PRTDEF	USESASHELP

Options

DATA=SAS-*data-set*

specifies the SAS input data set that contains the printer attributes.

Requirements: Printer attributes variables that must be specified are DEST, DEVICE, MODEL, and NAME, except when the value of the variable OPCODE is DELETE, in which case only the NAME variable is required.

DELETE

specifies that the default operation is to delete the printer definitions from the registry.

Interaction: If both DELETE and REPLACE are specified, then DELETE is the default operation.

Tip: If the user-defined printer definition is deleted, then the administrator-defined printer may still appear if it exists in the SASHELP catalog.

FOREIGN

specifies that the registry entries are being created for export to a different host. As a consequence, tests of any host-dependent items, such as the TRANTAB, are skipped.

LIST

specifies that a list of printers that are created or replaced will be written to the log.

REPLACE

specifies that the default operation is to modify existing printer definitions. Any printer name that already exists will be modified by using the information in the printer attributes data set. Any printer name that does not exist will be added.

Interaction: If both REPLACE and DELETE are specified, then a DELETE will be performed.

USESASHELP

specifies that the printer definitions that are to be placed in the SASHELP library, where they are available to all users.

If the USESASHELP option is not specified, then the printer definitions that are placed in the current SASUSER library, where they are available to the local user only.

Restriction: To use the USESASHELP option, you must have permission to write to the SASHELP catalog.

Operating Environment Information: You can create printer definitions with PROC PRTDEF in the Windows operating environment. However, because Universal Printing is turned off by default in Windows, these printer definitions do not appear in the Print window.

If you want to use your printer definitions when Universal Printing is turned off, then do one of the following:

- ☐ specify the printer definition as part of the PRINTERPATH system option
- ☐ from the Output Delivery System (ODS), issue the following code:

```
ODS PRINTER SAS PRINTER=myprinter;
```

where *myprinter* is the name of your printer definition.

Δ

Input Data Set: PRTDEF Procedure

Summary of Valid Variables

To create your printer definitions, you must create a SAS data set whose variables contain the appropriate printer attributes. The following table lists and describes both the required and the optional variables for this data set.

Variable Name	Variable Description
Required	
DEST	Destination
DEVICE	Device
MODEL	Prototype
NAME	Printer name
Optional	
BOTTOM	Default bottom margin
CHARSET	Default font character set
DESC	Description
FONTSIZE	Point size of the default font
HOSTOPT	Host options
LEFT	Default left margin
LRECL	Output buffer size
OPCODE	Operation code
PAPERIN	Paper source or input tray
PAPEROUT	Paper destination or output tray
PAPERSIZ	Paper size

Variable Name	Variable Description
PAPERTYP	Paper type
PREVIEW	Preview
PROTOCOL	Protocol
RES	Default printer resolution
RIGHT	Default right margin
STYLE	Default font style
TOP	Default top margin
TRANTAB	Translation table
TYPEFACE	Default font
UNITS	CM or IN units
VIEWER	Viewer
WEIGHT	Default font weight

Required Variables

To create or modify a printer, you must supply the NAME, MODEL, DEVICE, and DEST variables. All the other variables use default values from the printer prototype that is specified by the MODEL variable. To delete a printer, specify only the required NAME variable.

The following variables are required in the input data set:

- DEST** specifies the output destination for the printer.
- Operating Environment Information:* DEST is case sensitive for some devices. \triangle
- Restriction:** DEST is limited to 1023 characters.
- DEVICE** specifies the type of I/O device to use when sending output to the printer. Valid devices are listed in the Printer Definition wizard and in the SAS Registry Editor.
- Restriction:** DEVICE is limited to 31 characters.
- MODEL** specifies the printer prototype to use when defining the printer.
- For a valid list of prototypes or model descriptions, you can look in the SAS Registry Editor under CORE\PRINTING\PROTOTYPES.
- Tip:** While in interactive mode, you can invoke the registry with the REGEDIT command.
- Tip:** While in interactive mode, you can invoke the Print Setup dialog (DMPRTSETUP) and press **New** to view the list that is specified in the second window of the Printer Definition wizard.
- Restriction:** MODEL is limited to 127 characters.
- NAME** specifies the printer definition name that will be associated with the rest of the attributes in the printer definition.
- The name is unique within a given registry. If a new printer definition contains a name that already exists, then the record will

not be processed unless the REPLACE option has been specified or unless the value of the OPCODE variable is **Modify**.

Restriction: NAME must have the following features:

- It is limited to 127 characters.
- It must have at least one nonblank character.
- It cannot contain a backslash.

Note: Leading and trailing blanks will be stripped from the name. △

Optional Variables

The following variables are optional in the input data set:

BOTTOM

specifies the default bottom margin in the units that are specified by the UNITS variable.

CHARSET

specifies the default font character set.

Restriction: The value must be one of the character set names in the typeface that is specified by the TYPEFACE variable.

Restriction: CHARSET is limited to 31 characters.

DESC

specifies the description of the printer.

Restriction: The description can have a maximum of 1023 characters.

Default: DESC defaults to the prototype that is used to create the printer.

FONTSIZE

specifies the point size of the default font.

HOSTOPT

specifies any host options for the output destination. The host options are not case sensitive.

Restriction: The host options can have a maximum of 1023 characters.

LEFT

specifies the default left margin in the units that are specified by the UNITS variable.

LRECL

specifies the buffer size or record length to use when sending output to the printer.

Default: If LRECL is less than zero when modifying an existing printer definition that does not use the default buffer size, then the printer's non-default buffer size will be replaced by the default buffer size.

OPCODE

is a character variable that specifies what action (Add, Delete, or Modify) to perform on the printer definition.

Add

creates a new printer definition in the registry. If the REPLACE option has been specified, then this operation will also modify an existing printer definition.

Delete

removes an existing printer definition from the registry.

Restriction: This operation requires only the NAME variable to be defined. The other variables are ignored.

Modify

changes an existing printer definition in the registry or adds a new one.

Tip: If a user modifies and saves new attributes on a printer in the SASHELP library, then these modifications are stored in the SASUSER library. Values that are specified by the user will override values that are set by the administrator, but they will not replace them.

Restriction: OPTCODE is limited to 8 characters.

PAPERIN

specifies the default paper source or input tray.

Restriction: The value of PAPERIN must be one of the paper source names in the printer prototype that is specified by the MODEL variable.

Restriction: PAPERIN is limited to 31 characters.

PAPEROUT

specifies the default paper destination or output tray.

Restriction: The value of PAPEROUT must be one of the paper destination names in the printer prototype that is specified by the MODEL variable.

Restriction: PAPEROUT is limited to 31 characters.

PAPERSIZ

specifies the default paper source or input tray.

Restriction: The value of PAPERSIZ must be one of the paper size names listed in the printer prototype that is specified by the MODEL variable.

Restriction: PAPERSIZ is limited to 31 characters.

PAPERTYP

specifies the default paper type.

Restriction: The value of PAPERTYP must be one of the paper source names listed in the printer prototype that is specified by the MODEL variable.

Restriction: PAPERTYP is limited to 31 characters.

PREVIEW

specifies the printer application to use for print preview.

Restriction: PREVIEW is limited to 127 characters.

PROTOCOL

specifies the I/O protocol to use when sending output to the printer.

Operating Environment Information: On mainframe systems, the protocol describes how to convert the output to a format that can be processed by a protocol converter that connects the mainframe to an ASCII device. △

Restriction: PROTOCOL is limited to 31 characters.

RES

specifies the default printer resolution.

Restriction: The value of RES must be one of the resolution values available to the printer prototype that is specified by the MODEL variable.

Restriction: RES is limited to 31 characters.

RIGHT

specifies the default right margin in the units that are specified by the UNITS variable.

STYLE

specifies the default font style.

Restriction: The value of STYLE must be one of the styles available to the typeface that is specified by the TYPEFACE variable.

Restriction: STYLE is limited to 31 characters.

TOP

specifies the default top margin in the units that are specified by the UNITS variable.

TRANTAB

specifies which translation table to use when sending output to the printer.

Operating Environment Information: The translation table is needed when an EBCDIC host sends data to an ASCII device. \triangle

Restriction: TRANTAB is limited to 8 characters.

TYPEFACE

specifies the typeface of the default font.

Restriction: The typeface must be one of the typeface names available to the printer prototype that is specified by the MODEL variable.

Restriction: TYPEFACE is limited to 63 characters.

UNITS

specifies the units CM or IN that are used by margin variables.

VIEWER

specifies the host system command that is to be used during print previews. As a result, PROC PRTDEF causes a preview printer to be created.

Preview printers are specialized printers that are used to display printer output on the screen before printing.

Tip: The values of the PREVIEW, PROTOCOL, DEST, and HOSTOPT variables are ignored when a value for VIEWER has been specified. Place %s where the input filename would normally be in the viewer command. The %s can be used as many times as needed.

Restriction: VIEWER is limited to 127 characters.

WEIGHT

specifies the default font weight.

Restriction: The value must be one of the valid weights for the typeface that is specified by the TYPEFACE variable.

Examples: PRTDEF Procedure

Example 1: Defining Multiple Printer Definitions

Procedure features:

PROC PRTDEF statement option:

```
DATA=
USESASHELP
```

This example shows you how to set up various printers.

Program

Create the PRINTERS data set. The INPUT statement contains the names of the four required variables. Each data line contains the information that is needed to produce a single printer definition.

The & specifies that two or more blanks separate character values. This allows the name and model value to contain blanks.

```
data printers;
input name $&  model $& device $& dest $&;
datalines;
Myprinter          PostScript Level 1  PRINTER  printer1
Laserjet           PCL 5 Printer        PIPE      lp -dprinter5
Color LaserJet     PostScript Level 2   PIPE      lp -dprinter2
;
```

Specify the input data set that contains the printer attributes, create the printer definitions, and make the definitions available to all users. The DATA= option specifies PRINTERS as the input data set that contains the printer attributes.

PROC PRTDEF creates the printer definitions for the SAS registry, and the USESASHELP option specifies that the printer definitions will be available to all users.

```
proc prtdef data=printers usesashelp;
run;
```

Example 2: Creating a Ghostview Printer in SASUSER to Preview PostScript Printer Output in SASUSER

Procedure features:

PROC PRTDEF statement options:

```
DATA=
LIST
REPLACE
```

This example creates a Ghostview printer definition in the SASUSER library for previewing PostScript output.

Program

Create the GSVIEW data set, and specify the printer name, printer description, printer prototype, and commands to be used for print preview. The GSVIEW data set contains the variables whose values contain the information that is needed to produce the printer definitions.

The NAME variable specifies the printer name that will be associated with the rest of the attributes in the printer definition data record.

The DESC variable specifies the description of the printer.

The MODEL variable specifies the printer prototype to use when defining this printer.

The VIEWER variable specifies the host system commands to be used for print preview. GSVIEW must be installed on your system and the value for VIEWER must include the path to find it. You must enclose the value in single quotation marks because of the %s. If you use double quotation marks, SAS will assume that %s is a macro variable.

DEVICE and DEST are required variables, but no value is needed in this example. Therefore, a “dummy” or blank value should be assigned.

```
data gsview;
  name = "Ghostview";
  desc = "Print Preview with Ghostview";
  model= "PostScript Level 2 (Color)";
  viewer = 'ghostview %s';
  device = "Dummy";
  dest = " ";
```

Specify the input data set that contains the printer attributes, create the printer definitions, write the printer definitions to the SAS log, and replace a printer definition in the SAS registry. The DATA= option specifies GSVIEW as the input data set that contains the printer attributes.

PROC PRTDEF creates the printer definitions.

The LIST option specifies that a list of printers that are created or replaced will be written to the SAS log.

The REPLACE option specifies that a printer definition will replace a printer definition in the registry if the name of the printer definition matches a name already in the registry. If the printer definition names do not match, then the new printer definition is added to the registry.

```
proc prtdef data=gsview list replace;
run;
```

Example 3: Creating a Single Printer Definition That Is Available to All Users

Procedure features:

PROC PRTDEF statement option:

DATA=

This example creates a definition for a Tektronix Phaser 780 printer with a Ghostview print previewer with the following specifications:

- ☐ bottom margin set to 1 inch
- ☐ font size set to 14 point

- paper size set to A4.

Program

Create the TEK780 data set and supply appropriate information for the printer destination. The TEK780 data set contains the variables whose values contain the information that is needed to produce the printer definitions.

In the example, assignment statements are used to assign these variables.

The NAME variable specifies the printer name that will be associated with the rest of the attributes in the printer definition data record.

The DESC variable specifies the description of the printer.

The MODEL variable specifies the printer prototype to use when defining this printer.

The DEVICE variable specifies the type of I/O device to use when sending output to the printer.

The DEST variable specifies the output destination for the printer.

The PREVIEW variable specifies which printer will be used for print preview.

The UNITS variable specifies whether the margin variables are measured in centimeters or inches.

The BOTTOM variable specifies the default bottom margin in the units that are specified by the UNITS variable.

The FONTSIZE variable specifies the point size of the default font.

The PAPERSIZ variable specifies the default paper size.

```
data tek780;
  name = "Tek780";
  desc = "Test Lab Phaser 780P";
  model = "Tek Phaser 780 Plus";
  device = "PRINTER";
  dest = "testlab3";
  preview = "Ghostview";
  units = "in";
  bottom = 1;
  fontsize = 14;
  papersiz = "ISO A4";
run;
```

Create the TEK780 printer definition. The DATA= option specifies TEK780 as the input data set.

```
proc prtdef data=tek780;
run;
```

Example 4: Adding, Modifying, and Deleting Printer Definitions

Procedure features:

PROC PRTDEF statement options:

DATA=
LIST

This example

- adds two printer definitions
- modifies a printer definition
- deletes two printer definitions.

Program

Create the PRINTERS data set and specify which actions to perform on the printer definitions. The PRINTERS data set contains the variables whose values contain the information that is needed to produce the printer definitions.

The MODEL variable specifies the printer prototype to use when defining this printer.

The DEVICE variable specifies the type of I/O device to use when sending output to the printer.

The DEST variable specifies the output destination for the printer.

The OPCODE variable specifies which action (add, delete, or modify) to perform on the printer definition.

The first Add operation creates a new printer definition for Color PostScript in the SAS registry, and the second Add operation creates a new printer definition for ColorPS in the SAS registry.

The Mod operation modifies the existing printer definition for LaserJet 5 in the registry.

The Del operation deletes the printer definitions for Gray PostScript and test from the registry.

The & specifies that two or more blanks separate character values. This allows the name and model value to contain blanks.

```
data printers;
  length name    $ 80
         model   $ 80
         device  $ 8
         dest    $ 80
         opcode  $ 3
  ;
  input opcode $& name $& model $& device $& dest $&;
  datalines;
add  Color PostScript   PostScript Level 2 (Color)      DISK    sasprt.ps
mod  LaserJet 5         PCL 5                          DISK    sasprt.pcl
del  Gray PostScript    PostScript Level 2 (Gray Scale) DISK    sasprt.ps
del  test               PostScript Level 2 (Color)      DISK    sasprt.ps
add  ColorPS            PostScript Level 2 (Color)      DISK    sasprt.ps
  ;
```

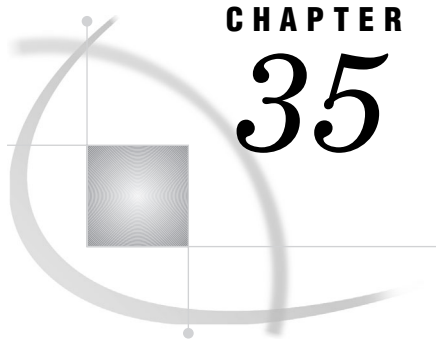
Create multiple printer definitions and write them to the SAS log. The DATA= option specifies the input data set PRINTERS that contains the printer attributes. PROC PRTDEF creates five printer definitions, two of which have been deleted. The LIST option specifies that a list of printers that are created or replaced will be written to the log.

```
proc prtdef data=printers library=sasuser list;
run;
```

See Also

Procedures

Chapter 35, “The PRTEXP Procedure,” on page 905



CHAPTER 35

The PRTEXP Procedure

<i>Overview: PRTEXP Procedure</i>	905
<i>Syntax: PRTEXP Procedure</i>	905
<i>PROC PRTEXP Statement</i>	905
<i>EXCLUDE Statement</i>	906
<i>SELECT Statement</i>	906
<i>Concepts: PRTEXP Procedure</i>	906
<i>Examples: PRTEXP Procedure</i>	907
<i>Example 1: Writing Attributes to the SAS Log</i>	907
<i>Example 2: Writing Attributes to a SAS Data Set</i>	907
<i>See Also</i>	909

Overview: PRTEXP Procedure

The PRTEXP procedure enables you to extract printer attributes from the SAS registry for replication and modification. PROC PRTEXP then writes these attributes to the SAS log or to a SAS data set. You can specify that PROC PRTEXP search for these attributes in the SASHELP portion of the registry or the entire SAS registry.

Syntax: PRTEXP Procedure

Note: If neither the SELECT nor the EXCLUDE statement is used, then all of the printers will be included in the output.

```
PROC PRTEXP<option(s)>;
    <SELECT printer_1 ...<printer_n>>;
    <EXCLUDE printer_1 ... <printer_n>>;
```

PROC PRTEXP Statement

```
PROC PRTEXP<option(s)>;
```

Options

USESASHELP

specifies that SAS search only the SASHELP portion of the registry for printer definitions.

Default: The default is to search both the SASUSER and SASHELP portions of the registry for printer definitions.

OUT=SAS-*data-set*

specifies the SAS data set to create that contains the printer definitions.

The data set that is specified by the OUT=SAS-*data-set* option is the same type of data set that is specified by the DATA=SAS-*data-set* option in PROC PRTDEF to define each printer.

Default: If OUT=SAS-*data-set* is not specified, then the data that is needed to define each printer is written to the SAS log.

EXCLUDE Statement

The EXCLUDE statement will cause the output to contain information from all those printers that are not listed.

```
EXCLUDE printer_1 ... <printer_n>;
```

Required Arguments

printer_1 printer_n

specifies the printer(s) that you do not want the output to contain information about.

SELECT Statement

The SELECT statement will cause the output to contain information from only those printers that are listed.

```
SELECT printer_1 ... <printer_n>;
```

Required Arguments

printer_1 printer_n

specifies the printer(s) that you would like the output to contain information about.

Concepts: PRTEXP Procedure

The PRTEXP procedure, along with the PRTDEF procedure, can replicate, modify, and create printer definitions either for an individual user or for all SAS users at your

site. PROC PRTEXP can extract only the attributes that are used to create printer definitions from the registry. If you write them to a SAS data set, then you can later replicate and modify them. You can then use PROC PRTDEF to create the printer definitions in the SAS registry from your input data set. For a complete discussion of PROC PRTDEF and the variables and attributes that are used to create the printer definitions, see “Input Data Set: PRTDEF Procedure” on page 895.

Examples: PRTEXP Procedure

Example 1: Writing Attributes to the SAS Log

Procedure Features:

PROC PRTEXP statement option:

SELECT statement

USESASHELP option

This example shows you how to write the attributes that are used to define a printer to the SAS log.

Program

Specify the printer that you want information about, specify that only the SASHELP portion of the registry be searched, and write the information to the SAS log. The SELECT statement specifies that you want the attribute information that is used to define the printer Postscript to be included in the output. The USESASHELP option specifies that only the SASHELP registry is to be searched for Postscript's printer definitions. The data that is needed to define each printer is written to the SAS log because the OUT= option was not used to specify a SAS data set.

```
proc prtexp usesashelp;  
select postscript;  
run;
```

Example 2: Writing Attributes to a SAS Data Set

Procedure Features:

PROC PRTEXP statement option:

OUT= option

SELECT statement

This example shows you how to create a SAS data set that contains the data that PROC PRTDEF would use to define the printers PCL4, PCL5, PCL5E, and PCLC.

Program

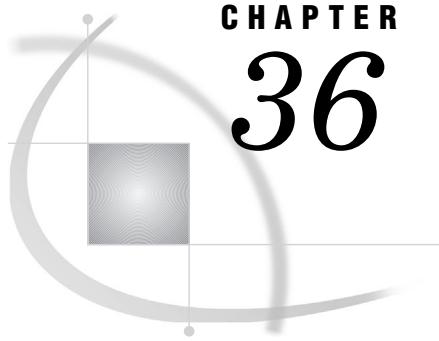
Specify the printers that you want information about and create the PRDVTER data set. The SELECT statement specifies the printers PCL4, PCL5, PCL5E, and PCLC. The OUT= option creates the SAS data set PRDVTER, which contains the same attributes that are used by PROC PRTDEF to define the printers PCL4, PCL5, PCL5E, and PCLC. SAS will search both the SASUSER and SASHELP registries, because USESASHELP was not specified.

```
proc prtexp out=PRDVTER;  
select pcl4 pcl5 pcl5e pcl5c;  
run;
```

See Also

Procedures

Chapter 34, “The PRTDEF Procedure,” on page 893



CHAPTER

36

The RANK Procedure

<i>Overview: RANK Procedure</i>	909
<i>Syntax: RANK Procedure</i>	911
<i>PROC RANK Statement</i>	911
<i>BY Statement</i>	914
<i>RANKS Statement</i>	915
<i>VAR Statement</i>	915
<i>Concepts: RANK Procedure</i>	915
<i>Computer Resources</i>	916
<i>Statistical Applications</i>	916
<i>Results: RANK Procedure</i>	916
<i>Missing Values</i>	916
<i>Output Data Set</i>	916
<i>Examples: RANK Procedure</i>	917
<i>Example 1: Ranking Values of Multiple Variables</i>	917
<i>Example 2: Ranking Values within BY Groups</i>	918
<i>Example 3: Partitioning Observations into Groups Based on Ranks</i>	920
<i>References</i>	923

Overview: RANK Procedure

The RANK procedure computes ranks for one or more numeric variables across the observations of a SAS data set and outputs the ranks to a new SAS data set. PROC RANK by itself produces no printed output.

Output 36.1 on page 909 shows the results of ranking the values of one variable with a simple PROC RANK step. In this example, the new ranking variable shows the order of finish of five golfers over a four-day competition. The player with the lowest number of strokes finishes in first place. The following statements produce the output:

```
proc rank data=golf out=rankings;
    var strokes;
    ranks Finish;
run;

proc print data=rankings;
run;
```

Output 36.1 Assignment of the Lowest Rank Value to the Lowest Variable Value

The SAS System				1
Obs	Player	Strokes	Finish	
1	Jack	279	2	
2	Jerry	283	3	
3	Mike	274	1	
4	Randy	296	4	
5	Tito	302	5	

In Output 36.2 on page 910, the candidates for city council are ranked by district according to the number of votes that they received in the election and according to the number of years that they have served in office.

This example shows how PROC RANK can

- ☐ reverse the order of the rankings so that the highest value receives the rank of 1, the next highest value receives the rank of 2, and so on
- ☐ rank the observations separately by values of multiple variables
- ☐ rank the observations within BY groups
- ☐ handle tied values.

For an explanation of the program that produces this report, see Example 2 on page 918.

Output 36.2 Assignment of the Lowest Rank Value to the Highest Variable Value within Each BY Group

Results of City Council Election						1
----- District=1 -----						
OBS	Candidate	Vote	Years	Vote Rank	Years Rank	
1	Cardella	1689	8	1	1	
2	Latham	1005	2	3	2	
3	Smith	1406	0	2	3	
4	Walker	846	0	4	3	
N = 4						
----- District=2 -----						
OBS	Candidate	Vote	Years	Vote Rank	Years Rank	
5	Hinkley	912	0	3	3	
6	Kreitemeyer	1198	0	2	3	
7	Lundell	2447	6	1	1	
8	Thrash	912	2	3	2	
N = 4						

Syntax: RANK Procedure

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, “Statements with the Same Function in Multiple Procedures,” on page 53 for details. You can also use any global statements as well. See “Global Statements” on page 18 for a list.

```
PROC RANK <option(s)>;
  BY <DESCENDING> variable-1
    <...<DESCENDING> variable-n>
    <NOTSORTED>;
  VAR data-set-variables(s);
  RANKS new-variables(s);
```

To do this	Use this statement
Calculate a separate set of ranks for each BY group	BY
Identify a variables that contain the ranks	RANKS
Specify the variables to rank	VAR

PROC RANK Statement

```
PROC RANK <option(s)>;
```

To do this	Use this option
Specify the input data set	DATA=
Create an output data set	OUT=
Specify the ranking method	
Compute fractional ranks	FRACTION or NPLUS1
Partition observations into groups	GROUPS=
Compute normal scores	NORMAL=
Compute percentages	PERCENT
Compute Savage scores	SAVAGE
Reverse the order of the rankings	DESCENDING
Specify how to rank tied values	TIES=

Note: You can specify only one ranking method in a single PROC RANK step. Δ

Options

DATA=SAS-*data-set*

specifies the input SAS data set.

Main discussion: “Input Data Sets” on page 19

Restriction: You cannot use PROC RANK with an engine that supports concurrent access if another user is updating the data set at the same time.

DESCENDING

reverses the direction of the ranks. With DESCENDING, the largest value receives a rank of 1, the next largest value receives a rank of 2, and so on. Otherwise, values are ranked from smallest to largest.

Featured in: Example 1 on page 917 and Example 2 on page 918

FRACTION

computes fractional ranks by dividing each rank by the number of observations having nonmissing values of the ranking variable.

Alias: F

Interaction: TIES=HIGH is the default with the FRACTION option. With TIES=HIGH, fractional ranks are considered values of a right-continuous empirical cumulative distribution function.

See also: NPLUS1 option

GROUPS=number-of-groups

assigns group values ranging from 0 to *number-of-groups* minus 1. Common specifications are GROUPS=100 for percentiles, GROUPS=10 for deciles, and GROUPS=4 for quartiles. For example, GROUPS=4 partitions the original values into four groups, with the smallest values receiving, by default, a quartile value of 0 and the largest values receiving a quartile value of 3.

The formula for calculating group values is

$$\text{FLOOR}(\text{rank} * k / (n + 1))$$

where FLOOR is the FLOOR function, *rank* is the value's order rank, *k* is the value of GROUPS=, and *n* is the number of observations having nonmissing values of the ranking variable.

If the number of observations is evenly divisible by the number of groups, each group has the same number of observations, provided there are no tied values at the boundaries of the groups. Grouping observations by a variable that has many tied values can result in unbalanced groups because PROC RANK always assigns observations with the same value to the same group.

Tip: Use DESCENDING to reverse the order of the group values.

Featured in: Example 3 on page 920

NORMAL=BLOM | TUKEY | VW

computes normal scores from the ranks. The resulting variables appear normally distributed. The formulas are

$$\text{BLOM} \quad y_i = \Phi^{-1}(r_i - 3/8) / (n + 1/4)$$

TUKEY $y_i = \Phi^{-1}(r_i - 1/3)/(n + 1/3)$

VW $y_i = \Phi^{-1}(r_i)/(n + 1)$

where Φ^{-1} is the inverse cumulative normal (PROBIT) function, r_i is the rank of the i th observation, and n is the number of nonmissing observations for the ranking variable.

VW stands for van der Waerden. With NORMAL=VW, you can use the scores for a nonparametric location test. All three normal scores are approximations to the exact expected order statistics for the normal distribution, also called *normal scores*. The BLOM version appears to fit slightly better than the others (Blom 1958; Tukey 1962).

Interaction: If you specify the TIES= option, then PROC RANK computes the normal score from the ranks based on non-tied values and applies the TIES= specification to the resulting normal score.

NPLUS1

computes fractional ranks by dividing each rank by the denominator $n + 1$, where n is the number of observations having nonmissing values of the ranking variable.

Aliases: FN1, N1

Interaction: TIES=HIGH is the default with the NPLUS1 option.

See also: FRACTION option

OUT=SAS-data-set

names the output data set. If *SAS-data-set* does not exist, PROC RANK creates it. If you omit OUT=, the data set is named using the *DATA* n naming convention.

PERCENT

divides each rank by the number of observations that have nonmissing values of the variable and multiplies the result by 100 to get a percentage.

Alias: P

Interaction: TIES=HIGH is the default with the PERCENT option.

Tip: You can use PERCENT to calculate cumulative percentages, but use GROUPS=100 to compute percentiles.

SAVAGE

computes Savage (or exponential) scores from the ranks by the following formula (Lehman 1998):

$$y_i = \left[\sum_{j=n-r_i+1}^n \left(\frac{1}{j} \right) \right] - 1$$

TIES=HIGH | LOW | MEAN

specifies how to compute normal scores or ranks for tied data values.

HIGH

assigns the largest of the corresponding ranks (or largest of the normal scores when NORMAL= is specified).

LOW

assigns the smallest of the corresponding ranks (or smallest of the normal scores when NORMAL= is specified).

MEAN

assigns the mean of the corresponding rank (or mean of the normal scores when NORMAL= is specified).

Default: MEAN (unless the FRACTION option or PERCENT option is in effect).

Interaction: If you specify the NORMAL= option, then the TIES= specification applies to the normal score, not to the rank that is used to compute the normal score.

Featured in: Example 1 on page 917 and Example 2 on page 918

BY Statement

Produces a separate set of ranks for each BY group.

Main discussion: “BY” on page 54

Featured in: Example 2 on page 918 and Example 3 on page 920

```
BY <DESCENDING> variable-1
    <...<DESCENDING> variable-n>
    <NOTSORTED>;
```

Required Arguments

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, the observations in the data set must either be sorted by all the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Options

DESCENDING

specifies that the observations are sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The observations are grouped in another way, such as chronological order.

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, the procedure treats each contiguous set as a separate BY group.

RANKS Statement

Creates new variables for the rank values.

Requirement: If you use the RANKS statement, you must also use the VAR statement.

Default: If you omit the RANKS statement, the rank values replace the original variable values in the output data set.

Featured in: Example 1 on page 917 and Example 2 on page 918

RANKS *new-variables(s)*;

Required Arguments

new-variable(s)

specifies one or more new variables that contain the ranks for the variable(s) listed in the VAR statement. The first variable listed in the RANKS statement contains the ranks for the first variable listed in the VAR statement, the second variable listed in the RANKS statement contains the ranks for the second variable listed in the VAR statement, and so forth.

VAR Statement

Specifies the input variables.

Default: If you omit the VAR statement, PROC RANK computes ranks for all numeric variables in the input data set.

Featured in: Example 1 on page 917, Example 2 on page 918, and Example 3 on page 920

VAR *data-set-variables(s)*;

Required Arguments

data-set-variable(s)

specifies one or more variables for which ranks are computed.

Using the VAR Statement with the RANKS Statement

The VAR statement is required when you use the RANKS statement. Using these statements together creates the ranking variables named in the RANKS statement that correspond to the input variables specified in the VAR statement. If you omit the RANKS statement, the rank values replace the original values in the output data set.

Computer Resources

PROC RANK stores all values in memory of the variables for which it computes ranks.

Statistical Applications

Ranks are useful for investigating the distribution of values for a variable. The ranks divided by n or $n+1$ form values in the range 0 to 1, and these values estimate the cumulative distribution function. You can apply inverse cumulative distribution functions to these fractional ranks to obtain probability quantile scores, which you can compare to the original values to judge the fit to the distribution. For example, if a set of data has a normal distribution, the normal scores should be a linear function of the original values, and a plot of scores versus original values should be a straight line.

Many nonparametric methods are based on analyzing ranks of a variable:

- A two-sample t -test applied to the ranks is equivalent to a Wilcoxon rank sum test using the t approximation for the significance level. If you apply the t -test to the normal scores rather than to the ranks, the test is equivalent to the van der Waerden test. If you apply the t -test to median scores (GROUPS=2), the test is equivalent to the median test.
- A one-way analysis of variance applied to ranks is equivalent to the Kruskal-Wallis k -sample test; the F-test generated by the parametric procedure applied to the ranks is often better than the X^2 approximation used by Kruskal-Wallis. This test can be extended to other rank scores (Quade 1966).
- You can obtain a Friedman's two-way analysis for block designs by ranking within BY groups and then performing a main-effects analysis of variance on these ranks (Conover 1998).
- You can investigate regression relationships by using rank transformations with a method described by Iman and Conover (1979).

Results: RANK Procedure

Missing Values

Missing values are not ranked and are left missing when ranks or rank scores replace the original values in the output data set.

Output Data Set

The RANK procedure creates a SAS data set containing the ranks or rank scores but does not create any printed output. You can use PROC PRINT, PROC REPORT, or another SAS reporting tool to print the output data set.

The output data set contains all the variables from the input data set plus the variables named in the RANKS statement. If you omit the RANKS statement, the rank values replace the original variable values in the output data set.

Examples: RANK Procedure

Example 1: Ranking Values of Multiple Variables

Procedure features:

PROC RANK statement options:

DESCENDING

TIES=

RANKS statement

VAR statement

Other features:

PRINT procedure

This example

- ☐ reverses the order of the ranks so that the highest value receives the rank of 1
- ☐ assigns tied values the best possible rank
- ☐ creates ranking variables and prints them with the original variables.

Program

Set the SAS system options. The NODATE option specifies to omit the date and time when the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines for a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the CAKE data set. This data set contains each participant's last name, score for presentation, and score for taste in a cake-baking contest.

```
data cake;
    input Name $ 1-10 Present 12-13 Taste 15-16;
    datalines;
Davis          77 84
Orlando        93 80
Ramey          68 72
Roe            68 75
Sanders        56 79
Simms          68 77
Strickland     82 79
    ;
```

Generate the ranks for the numeric variables in descending order and create the output data set ORDER. DESCENDING reverses the order of the ranks so that the high score receives the rank of 1. TIES=LOW gives tied values the best possible rank. OUT= creates the output data set ORDER.

```
proc rank data=cake out=order descending ties=low;
```

Create two new variables that contain ranks. The VAR statement specifies the variables to rank. The RANKS statement creates two new variables, PresentRank and TasteRank, that contain the ranks for the variables Present and Taste, respectively.

```
var present taste;
ranks PresentRank TasteRank;
run;
```

Print the data set. PROC PRINT prints the ORDER data set. The TITLE statement specifies a title.

```
proc print data=order;
title "Rankings of Participants' Scores";
run;
```

Output

Rankings of Participants' Scores						1
OBS	Name	Present	Taste	Present Rank	Taste Rank	
1	Davis	77	84	3	1	
2	Orlando	93	80	1	2	
3	Ramey	68	72	4	7	
4	Roe	68	75	4	6	
5	Sanders	56	79	7	3	
6	Simms	68	77	4	5	
7	Strickland	82	79	2	3	

Example 2: Ranking Values within BY Groups

Procedure features:

PROC RANK statement options:

DESCENDING

TIES=

BY statement

RANKS statement

VAR statement

Other features:

PRINT procedure

This example

- ranks observations separately within BY groups
- reverses the order of the ranks so that the highest value receives the rank of 1
- assigns tied values the best possible rank
- creates ranking variables and prints them with the original variables.

Program

Set the SAS system options. The NODATE option specifies to omit the date and time when the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines for a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the ELECT data set. This data set contains each candidate's last name, district number, vote total, and number of years' experience on the city council.

```
data elect;
    input Candidate $ 1-11 District 13 Vote 15-18 Years 20;
    datalines;
Cardella      1 1689 8
Latham        1 1005 2
Smith         1 1406 0
Walker        1  846 0
Hinkley       2  912 0
Kreitemeyer  2 1198 0
Lundell       2 2447 6
Thrash        2  912 2
    ;
```

Generate the ranks for the numeric variables in descending order and create the output data set RESULTS. DESCENDING reverses the order of the ranks so that the highest vote total receives the rank of 1. TIES=LOW gives tied values the best possible rank. OUT= creates the output data set RESULTS.

```
proc rank data=elect out=results ties=low descending;
```

Create a separate set of ranks for each BY group. The BY statement separates the rankings by values of District.

```
by district;
```

Create two new variables that contain ranks. The VAR statement specifies the variables to rank. The RANKS statement creates the new variables, VoteRank and YearsRank, that contain the ranks for the variables Vote and Years, respectively.

```
var vote years;
ranks VoteRank YearsRank;
run;
```

Print the data set. PROC PRINT prints the RESULTS data set. The N option prints the number of observations in each BY group. The TITLE statement specifies a title.

```
proc print data=results n;
  by district;
  title 'Results of City Council Election';
run;
```

Output

In the second district, Hinkley and Thrash tied with 912 votes. They both receive a rank of 3 because TIES=LOW.

Results of City Council Election						1
----- District=1 -----						
OBS	Candidate	Vote	Years	Vote Rank	Years Rank	
1	Cardella	1689	8	1	1	
2	Latham	1005	2	3	2	
3	Smith	1406	0	2	3	
4	Walker	846	0	4	3	
N = 4						
----- District=2 -----						
OBS	Candidate	Vote	Years	Vote Rank	Years Rank	
5	Hinkley	912	0	3	3	
6	Kreitemeyer	1198	0	2	3	
7	Lundell	2447	6	1	1	
8	Thrash	912	2	3	2	
N = 4						

Example 3: Partitioning Observations into Groups Based on Ranks

Procedure features:

PROC RANK statement option:

GROUPS=
 BY statement
 VAR statement
Other features:
 PRINT procedure
 SORT procedure

This example

- ☐ partitions observations into groups on the basis of values of two input variables
- ☐ groups observations separately within BY groups
- ☐ replaces the original variable values with the group values.

Program

Set the SAS system options. The NODATE option specifies to omit the date and time when the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines for a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the SWIM data set. This data set contains swimmers' first names and their times, in seconds, for the backstroke and the freestyle. This example groups the swimmers into pairs, within male and female classes, based on times for both strokes so that every swimmer is paired with someone who has a similar time for each stroke.

```
data swim;
  input Name $ 1-7 Gender $ 9 Back 11-14 Free 16-19;
  datalines;
Andrea F 28.6 30.3
Carole F 32.9 24.0
Clayton M 27.0 21.9
Curtis M 29.0 22.6
Doug M 27.3 22.4
Ellen F 27.8 27.0
Jan F 31.3 31.2
Jimmy M 26.3 22.5
Karin F 34.6 26.2
Mick M 29.0 25.4
Richard M 29.7 30.2
Sam M 27.2 24.1
Susan F 35.1 36.1
;
```

Sort the SWIM data set and create the output data set PAIRS. PROC SORT sorts the data set by Gender. This is required to obtain a separate set of ranks for each group. OUT= creates the output data set PAIRS.

```
proc sort data=swim out=pairs;
  by gender;
run;
```

Generate the ranks that are partitioned into three groups and create an output data set. GROUPS=3 assigns one of three possible group values (0,1,2) to each swimmer for each stroke. OUT= creates the output data set RANKPAIR.

```
proc rank data=pairs out=rankpair groups=3;
```

Create a separate set of ranks for each BY group. The BY statement separates the rankings by Gender.

```
  by gender;
```

Replace the original values of the variables with the rank values. The VAR statement specifies that Back and Free are the variables to rank. With no RANKS statement, PROC RANK replaces the original variable values with the group values in the output data set.

```
  var back free;
run;
```

Print the data set. PROC PRINT prints the RANKPAIR data set. The N option prints the number of observations in each BY group. The TITLE statement specifies a title.

```
proc print data=rankpair n;
  by gender;
  title 'Pairings of Swimmers for Backstroke and Freestyle';
run;
```

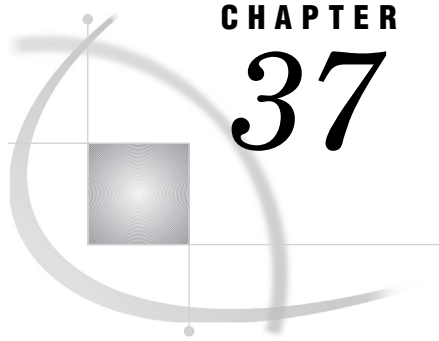
Output

The group values pair up swimmers with similar times to work on each stroke. For example, Andrea and Ellen work together on the backstroke because they have the fastest times in the female class. The groups of male swimmers are unbalanced because there are seven male swimmers; for each stroke, one group has three swimmers.

Pairings of Swimmers for Backstroke and Freestyle				1
----- Gender=F -----				
OBS	Name	Back	Free	
1	Andrea	0	1	
2	Carole	1	0	
3	Ellen	0	1	
4	Jan	1	2	
5	Karin	2	0	
6	Susan	2	2	
N = 6				
----- Gender=M -----				
OBS	Name	Back	Free	
7	Clayton	0	0	
8	Curtis	2	1	
9	Doug	1	0	
10	Jimmy	0	1	
11	Mick	2	2	
12	Richard	2	2	
13	Sam	1	1	
N = 7				

References

- Blom, G. (1958), *Statistical Estimates and Transformed Beta Variables*, New York: John Wiley & Sons, Inc.
- Conover, W.J. (1998), *Practical Nonparametric Statistics, Third Edition*, New York: John Wiley & Sons, Inc.
- Conover, W.J. and Iman, R.L. (1976), "On Some Alternative Procedures Using Ranks for the Analysis of Experimental Designs," *Communications in Statistics*, A5, 14, 1348–1368.
- Conover, W.J. and Iman, R.L. (1981), "Rank Transformations as a Bridge between Parametric and Nonparametric Statistics," *The American Statistician*, 35, 124–129.
- Iman, R.L. and Conover, W.J. (1979), "The Use of the Rank Transform in Regression," *Technometrics*, 21, 499–509.
- Lehman, E.L. (1998), *Nonparametrics: Statistical Methods Based on Ranks*, New Jersey: Prentice Hall .
- Quade, D. (1966), "On Analysis of Variance for the k -Sample Problem," *Annals of Mathematical Statistics*, 37, 1747–1758.
- Tukey, John W. (1962), "The Future of Data Analysis," *Annals of Mathematical Statistics*, 33, 22.



CHAPTER 37

The REGISTRY Procedure

<i>Overview: REGISTRY Procedure</i>	925
<i>Syntax: REGISTRY Procedure</i>	925
<i>PROC REGISTRY Statement</i>	926
<i>Creating Registry Files with the REGISTRY Procedure</i>	929
<i>Structure of a Registry File</i>	930
<i>Specifying Key Names</i>	930
<i>Specifying Values for Keys</i>	930
<i>Sample Registry Entries</i>	931
<i>Examples: REGISTRY Procedure</i>	932
<i>Example 1: Importing a File to the Registry</i>	932
<i>Example 2: Listing and Exporting the Registry</i>	933
<i>Example 3: Comparing the Registry to an External File</i>	934
<i>Example 4: Comparing Registry Files</i>	935
<i>See Also</i>	937

Overview: REGISTRY Procedure

The REGISTRY procedure maintains the SAS registry. The registry consists of two parts. One part is stored in the SASHELP library, and the other part is stored in the SASUSER library.

The REGISTRY procedure enables you to

- ☐ import registry files to populate the SASHELP and SASUSER registries
- ☐ export all or part of the registry to another file
- ☐ list the contents of the registry in the SAS log
- ☐ compare the contents of the registry to a file
- ☐ uninstall a registry file
- ☐ deliver detailed status information when a key or value will be overwritten or uninstalled
- ☐ clear out entries in the SASUSER registry
- ☐ validate that the registry exists
- ☐ list diagnostic information.

Syntax: REGISTRY Procedure

```
PROC REGISTRY <option(s)>;
```

PROC REGISTRY Statement

PROC REGISTRY *<option(s)>*;

To do this	Use this option
Erase the contents of the SASUSER registry	CLEARASASUSER
Compare two registry files	COMPAREREG1 and COMPAREREG2
Compare the contents of a registry to a file	COMPARETO
Enable registry debugging	DEBUGON
Disable registry debugging	DEBUGOFF
Write the contents of a registry to the specified file	EXPORT=
Provide additional information in the SAS log about the results of the IMPORT= and the UNINSTALL options	FULLSTATUS
Import the specified file to a registry	IMPORT=
Write the contents of the registry to the SAS log. Used with the STARTAT= option to list specific keys.	LIST
Write the contents of the SASHELP portion of the registry to the SAS log	LISTHELP
Send the contents of a registry to the log	LISTREG
Write the contents of the SASUSER portion of the registry to the SAS log	LISTUSER
Start exporting or writing or comparing the contents of a registry at the specified key	STARTAT=
Delete from the specified registry all the keys and values that are in the specified file	UNINSTALL
Uppercase all incoming key names	UPCASE
Perform the specified operation on the SASHELP portion of the SAS registry	USESASHELP

Options

CLEARASASUSER

erases the content of the SASUSER portion of the SAS registry.

COMPAREREG1=*"libname.registry name-1"*

specifies one of two registries to compare. The results appear in the SAS log.

libname

is the name of the library in which the registry file resides.

registry name-1

is the name of the first registry.

Requirement: Must be used with COMPAREREG2.

Interaction: To specify a single key and all of its subkeys, specify the STARTAT= option.

COMPAREREG2=“*libname.registry name-2*”

specifies the second of two registries to compare. The results appear in the SAS log.

libname

is the name of the library in which the registry file resides.

registry name-2

is the name of the second registry.

Requirement: Must be used with COMPAREREG1.

COMPARETO=*file-specification*

compares the contents of a file that contains registry information to a registry. It returns information about keys and values that it finds in the file that are not in the registry. It reports as differences

- keys that are defined in the external file but not in the registry
- value names for a given key that are in the external file but not in the registry
- differences in the content of like-named values in like-named keys.

COMPARETO= does not report as differences any keys and values that are in the registry but not in the file because the registry could easily be composed of pieces from many different files.

file-specification is one of the following:

'external-file'

is the path and name of an external file that contains the registry information.

fileref

is a fileref that has been assigned to an external file.

Requirement: You must have previously associated the fileref with an external file in a FILENAME statement, a FILENAME function, the Explorer window, or an appropriate operating environment command.

Interaction: By default, PROC REGISTRY compares *file-specification* to the SASUSER portion of the registry. To compare *file-specification* to the SASHELP portion of the registry, specify the option USESASHELP.

See also: For information about how to structure a file that contains registry information, see “Creating Registry Files with the REGISTRY Procedure” on page 929.

DEBUGON

enables registry debugging by providing more descriptive log entries.

DEBUGOFF

disables registry debugging.

EXPORT=*file-specification*

writes the contents of a registry to the specified file, where *file-specification* is one of the following:

'external-file'

is the name of an external file that contains the registry information.

fileref

is a fileref that has been assigned to an external file.

Requirement: You must have previously associated the fileref with an external file in a FILENAME statement, a FILENAME function, the Explorer window, or an appropriate operating environment command.

If *file-specification* already exists, PROC REGISTRY overwrites it. Otherwise, PROC REGISTRY creates the file.

Interaction: By default, EXPORT= writes the SASUSER portion of the registry to the specified file. To write the SASHELP portion of the registry, specify the USESASHELP option. You must have write permission to the SASHELP library to use USESASHELP.

Interaction: To export a single key and all of its subkeys, specify the STARTAT= option.

FULLSTATUS

lists the keys, subkeys, and values that were added or deleted as a result of running the IMPORT= and the UNINSTALL options.

IMPORT=*file-specification*

specifies the file to import into the SAS registry. PROC REGISTRY does not overwrite the existing registry. Instead, it updates the existing registry with the contents of the specified file.

Note: .sasxreg file extension is not required. Δ

file-specification is one of the following:

'external-file'

is the path and name of an external file that contains the registry information.

fileref

is a fileref that has been assigned to an external file.

Requirement: You must have previously associated the fileref with an external file in a FILENAME statement, a FILENAME function, the Explorer window, or an appropriate operating environment command.

Interaction: By default, IMPORT= imports the file to the SASUSER portion of the SAS registry. To import the file to the SASHELP portion of the registry, specify the USESASHELP option. You must have write permission to SASHELP to use USESASHELP.

Interaction: To obtain additional information in the SAS log as you import a file, use FULLSTATUS.

See also: For information about how to structure a file that contains registry information, see “Creating Registry Files with the REGISTRY Procedure” on page 929.

LIST

writes the contents of the entire SAS registry to the SAS log.

Interaction: To write a single key and all of its subkeys, use the STARTAT= option.

LISTHELP=

writes the contents of the SASHELP portion of the registry to the SAS log.

Interaction: To write a single key and all of its subkeys, use the STARTAT= option.

LISTREG=*“libname.registry name”*

lists the contents of the specified registry in the log.

libname

is the name of the library in which the registry file resides.

registry name

is the name of the registry.

Example:

```
proc registry listreg='sashelp.regstry';
run;
```

Interaction: To list a single key and all of its subkeys, use the STARTAT= option.

LISTUSER

writes the contents of the SASUSER portion of the registry to the SAS log.

Interaction: To write a single key and all of its subkeys, use the STARTAT= option.

STARTAT=*'key-name'*

exports or writes the contents of a single key and all of its subkeys.

Interaction: USE STARTAT= with the EXPORT=, LIST, LISTHELP, LISTUSER, COMPAREREG1=, COMPAREREG2= and the LISTREG option.

UNINSTALL=*file-specification*

deletes from the specified registry all the keys and values that are in the specified file. *file-specification* is one of the following:

'external-file'

is the name of an external file that contains the keys and values to delete.

fileref

is a fileref that has been assigned to an external file. To assign a fileref you can

- ☐ use the Explorer Window
- ☐ use the FILENAME statement. (For information about the FILENAME statement, see the section on statements in *SAS Language Reference: Dictionary*.)

Interaction: By default, UNINSTALL deletes the keys and values from the SASUSER portion of the SAS registry. To delete the keys and values from the SASHELP portion of the registry, specify the USESASHELP option. You must have write permission to SASHELP to use this option.

Interaction: Use FULLSTATUS to obtain additional information in the SAS log as you uninstall a registry.

See also: For information about how to structure a file that contains registry information, see “Creating Registry Files with the REGISTRY Procedure” on page 929.

UPCASE

uppercases all incoming key names.

USESASHELP

performs the specified operation on the SASHELP portion of the SAS registry.

Interaction: Use USESASHELP with the IMPORT=, EXPORT=, COMPARETO, or UNINSTALL option. To use USESASHELP with IMPORT= or UNINSTALL, you must have write permission to SASHELP.

Creating Registry Files with the REGISTRY Procedure

Structure of a Registry File

You can create registry files with the SAS Registry Editor or with any text editor.

A registry file must have a particular structure. Each entry in the registry file consists of a key name, followed on the next line by one or more values. The key name identifies the key or subkey that you are defining. Any values that follow specify the names or data to associate with the key.

Specifying Key Names

Key names are entered on a single line between square brackets ([and]). To specify a subkey, enter multiple key names between the brackets, starting with the root key. Separate the names in a sequence of key names with a backslash (\). The length of a single key name or a sequence of key names cannot exceed 255 characters (including the square brackets and the backslashes). Key names can contain any character except the backslash.

Examples of valid key name sequences follow. These sequences are typical of the SAS registry:

```
[CORE\EXPLORER\MENUS\ENTRIES\CLASS]
[CORE\EXPLORER\NEWMEMBER\CATALOG]
[CORE\EXPLORER\NEWENTRY\CLASS]
[CORE\EXPLORER\ICONS\ENTRIES\LOG]
```

Specifying Values for Keys

Enter each value on the line that follows the key name that it is associated with. You can specify multiple values for each key, but each value must be on a separate line.

The general form of a value is

value-name=value-content

A *value-name* can be an at sign (@), which indicates the default value name, or it can be any text string in double quotation marks. If the text string contains an ampersand (&), then the character (either uppercase or lowercase) that follows the ampersand is a shortcut for the value name. See “Sample Registry Entries” on page 931.

The entire text string cannot contain more than 255 characters (including quotation marks and ampersands). It can contain any character except a backslash (\).

Value-content can be any of the following:

- the string **double:** followed by a numeric value.
- a string. You can put anything inside the quotes, including nothing ("").

Note: To include a backslash in the quoted string, use two adjacent backslashes. To include a double quotation mark, use two adjacent double quotation marks. △

- the string **hex:** followed by any number of hexadecimal characters, up to the 255-character limit, separated by commas. If you extend the hexadecimal characters beyond a single line, end the line with a backslash to indicate that the data continues on the next line. Hex values may also be referred to as “binary values” in the Registry Editor.
- the string **dword:** followed by an unsigned long hexadecimal value.
- the string **int:** followed by a signed long integer value.
- the string **uint:** followed by an unsigned long integer value.

The following display shows how the different types of values that are described above appear in the Registry Editor:

Display 37.1 Types of Registry Values, Displayed in the Registry Editor

Name	Data
A double value	2.4E-44
A string	"my data"
Binary data	01,00,76,63,62,6B
Dword	66051
Signed integer value	-123
Unsigned integer value (decimal)	123456

The following list contains a sample of valid registry values:

- A double value=double:2.4E-44
- A string="my data"
- Binary data=hex: 01,00,76,63,62,6B
- Dword=dword:00010203
- Signed integer value=int:-123
- Unsigned integer value (decimal)=dword:0001E240

Sample Registry Entries

Registry entries can vary in content and appearance, depending on their purpose. The following display shows a registry entry that contains default PostScript printer settings.

Display 37.2 Portion of a Registry Editor Showing Settings for a PostScript Printer

Contents of 'DEFAULT SETTINGS'	
Name	Data
Font Character Set	"Western"
Font Size	12
Font Style	"Regular"
Font Typeface	"Courier"
Font Weight	"Normal"
Margin Bottom	0.5
Margin Left	0.5
Margin Right	0.5
Margin Top	0.5
Margin Units	"IN"
Paper Destination	""
Paper Size	"Letter"
Paper Source	""
Paper Type	""
Resolution	"300 DPI"

To see what the actual registry text file looks like, you can use PROC REGISTRY to write the contents of the registry key to the SAS log, using the LISTUSER and STARTAT= options:

Example Code 37.1 SAS code for sending a SASUSER registry entry to the log

```
proc registry
  listuser
```

```

startat="<sasuser registry key name>";
run;

```

Example Code 37.2 SAS code for sending a SASUSER registry entry to the log

```

proc registry
  listuser
  startat="HKEY_SYSTEM_ROOT\CORE\PRINTING\PRINTERS\PostScript\DEFAULT SETTINGS";
run;

```

For example, the list below begins at the
CORE\PRINTING\PRINTERS\PostScript\DEFAULT SETTINGS key.

Output 37.1 Log Output of a Registry Entry for a PostScript Printer

```

NOTE: Contents of SASUSER REGISTRY starting at subkey [CORE\
PRINTING\PRINTERS\PostScript\DEFAULT SETTINGS key]

```

```

Font Character Set="Western"
Font Size=double:12
Font Style="Regular"
Font Typeface="Courier"
Font Weight="Normal"
Margin Bottom=double:0.5
Margin Left=double:0.5
Margin Right=double:0.5
Margin Top=double:0.5
Margin Units="IN"
Paper Destination=""
Paper Size="Letter"
Paper Source=""
Paper Type=""
Resolution="300 DPI"

```

```

NOTE: PROCEDURE REGISTRY used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds

```

Examples: REGISTRY Procedure

Example 1: Importing a File to the Registry

Procedure features: IMPORT=

Other features: FILENAME statement

This example imports a file into the SASUSER portion of the SAS registry.

Source File

The following file contains examples of valid key name sequences in a registry file:

```
[HKEY_USER_ROOT\AllGoodPeopleComeToTheAidOfTheirCountry]
@="This is a string value"
"Value2"=""
"Value3"="C:\\This\\Is\\Another\\String\\Value"
```

Program

Assign a fileref to a file that contains valid text for the registry. The FILENAME statement assigns the fileref SOURCE to the external file that contains the text to read into the registry.

```
filename source 'external-file';
```

Invoke PROC REGISTRY to import the file that contains input for the registry. PROC REGISTRY reads the input file that is identified by the fileref SOURCE. IMPORT= writes to the SASUSER portion of the SAS registry by default.

```
proc registry    import=source;
run;
```

SAS Log

```
1  filename source 'external-file';
2  proc registry
3      import=source;
4  run;
Parsing REG file and loading the registry please wait....
Registry IMPORT is now complete.
```

Example 2: Listing and Exporting the Registry

Procedure features:

```
EXPORT=
LISTUSER
```

This example lists the SASUSER portion of the SAS registry and exports it to an external file.

Note: This is usually a very large file. To export a portion of the registry, use the STARTAT= option. △

Program

Write the contents of the SASUSER portion of the registry to the SAS log. The LISTUSER option causes PROC REGISTRY to write the entire SASUSER portion of the registry to the log.

```
proc registry
  listuser
```

Export the registry to the specified file. The EXPORT= option writes a copy of the SASUSER portion of the SAS registry to the external file.

```
    export='external-file';
run;
```

SAS Log

```
1  proc registry listuser export='external-file';
2  run;
Starting to write out the registry file, please wait...
The export to file external-file is now complete.
Contents of SASUSER REGISTRY.
[  HKEY_USER_ROOT]
[    CORE]
[      EXPLORER]
[        CONFIGURATION]
[          Initialized= "True"
[      FOLDERS]
[        UNXHOST1]
[          Closed= "658"
[          Icon= "658"
[          Name= "Home Directory"
[          Open= "658"
[          Path= "~"
```

Example 3: Comparing the Registry to an External File

Procedure features: COMPARETO=

Other features: FILENAME statement

This example compares the SASUSER portion of the SAS registry to an external file. Comparisons such as this are useful if you want to know the difference between a backup file that was saved with a .txt file extension and the current registry file.

Note: To compare the SASHELP portion of the registry with an external file, specify the USESASHELP option. Δ

Program

Assign a fileref to the external file that contains the text to compare to the registry. The FILENAME statement assigns the fileref TESTREG to the external file.

```
filename testreg 'external-file';
```


Compare the specified file to the SASUSER portion of the SAS registry. The COMPARETO option compares the contents of a file to a registry. It returns information about keys and values that it finds in the file that are not in the registry.

```
proc registry
  compareto=testreg;
run;
```

SAS Log

This SAS log shows two differences between the SASUSER portion of the registry and the specified external file. In the registry, the value of “Initialized” is “True”; in the external file, it is “False”. In the registry, the value of “Icon” is “658”; in the external file it is “343”.

```
1  filename testreg 'external-file';
2  proc registry
3    compareto=testreg;
4  run;
Parsing REG file and comparing the registry please wait....
COMPARE DIFF: Value "Initialized" in
[HKEY_USER_ROOT\CORE\EXPLORER\CONFIGURATION]: REGISTRY TYPE=STRING, CURRENT
VALUE="True"
COMPARE DIFF: Value "Initialized" in
[HKEY_USER_ROOT\CORE\EXPLORER\CONFIGURATION]: FILE TYPE=STRING, FILE
VALUE="False"
COMPARE DIFF: Value "Icon" in
[HKEY_USER_ROOT\CORE\EXPLORER\FOLDERS\UNXHOST1]: REGISTRY TYPE=STRING,
CURRENT VALUE="658"
COMPARE DIFF: Value "Icon" in
[HKEY_USER_ROOT\CORE\EXPLORER\FOLDERS\UNXHOST1]: FILE TYPE=STRING, FILE
VALUE="343"
Registry COMPARE is now complete.
COMPARE: There were differences between the registry and the file.
```

Example 4: Comparing Registry Files

This example uses the REGISTRY procedure options COMPAREREG1 and COMPAREREG2 to specify two registry files for comparison.

Start PROC REGISTRY and specify the first registry file to be used in the comparison.

```
proc registry comparereg1="sasuser.regstry"
```

Limit the comparison to the registry keys including and following the specified registry key. The STARTAT= option limits the scope of the comparison. By default the comparison includes the entire contents of both registries.

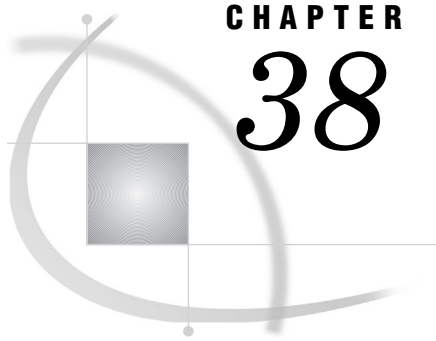
```
startat="HKEY_USER_ROOT\COLORNAMES"
```

Specify the second registry file to be used in the comparison.

```
comparereg2="sasuser.registry_copy";  
run;
```

See Also

SAS registry chapter in *SAS Language Reference: Concepts*



CHAPTER

38

The REPORT Procedure

Overview: <i>REPORT</i> Procedure	939
What Does the <i>REPORT</i> Procedure Do?	939
What Types of Reports Can <i>PROC REPORT</i> Produce?	939
What Do the Various Types of Reports Look Like?	939
Concepts: <i>REPORT</i> Procedure	944
Laying Out a Report	944
Usage of Variables in a Report	945
Display Variables	945
Order Variables	945
Across Variables	946
Group Variables	946
Analysis Variables	946
Computed Variables	947
Interactions of Position and Usage	947
Statistics That Are Available in <i>PROC REPORT</i>	949
Using Compute Blocks	949
The Purpose of Compute Blocks	950
The Contents of Compute Blocks	950
Four Ways to Reference Report Items in a Compute Block	951
Compute Block Processing	952
Using Break Lines	952
Creating Break Lines	952
Order of Break Lines	953
The Automatic Variable <i>_BREAK_</i>	953
Using Style Elements in <i>PROC REPORT</i>	953
Printing a Report	956
Printing with ODS	956
Printing from the <i>REPORT</i> Window	956
Printing with a Form	956
Printing from the Output Window	957
Printing from Noninteractive or Batch Mode	957
Printing from Interactive Line Mode	957
Using <i>PROC PRINTTO</i>	957
Storing and Reusing a Report Definition	957
Syntax: <i>REPORT</i> Procedure	958
<i>PROC REPORT</i> Statement	959
<i>BREAK</i> Statement	973
<i>BY</i> Statement	978
<i>CALL DEFINE</i> Statement	979
<i>COLUMN</i> Statement	981
<i>COMPUTE</i> Statement	983

<i>DEFINE Statement</i>	985
<i>ENDCOMP Statement</i>	994
<i>FREQ Statement</i>	994
<i>LINE Statement</i>	995
<i>RBREAK Statement</i>	996
<i>WEIGHT Statement</i>	1000
<i>REPORT Procedure Windows</i>	1000
<i>BREAK</i>	1001
<i>COMPUTE</i>	1004
<i>COMPUTED VAR</i>	1004
<i>DATA COLUMNS</i>	1005
<i>DATA SELECTION</i>	1005
<i>DEFINITION</i>	1006
<i>DISPLAY PAGE</i>	1011
<i>EXPLORE</i>	1012
<i>FORMATS</i>	1013
<i>LOAD REPORT</i>	1013
<i>MESSAGES</i>	1014
<i>PROFILE</i>	1014
<i>PROMPTER</i>	1015
<i>REPORT</i>	1016
<i>ROPTIONS</i>	1016
<i>SAVE DATA SET</i>	1021
<i>SAVE DEFINITION</i>	1021
<i>SOURCE</i>	1022
<i>STATISTICS</i>	1022
<i>WHERE</i>	1023
<i>WHERE ALSO</i>	1023
<i>How PROC REPORT Builds a Report</i>	1024
<i>Sequence of Events</i>	1024
<i>Construction of Summary Lines</i>	1025
<i>Using Compound Names</i>	1025
<i>Building a Report That Uses Groups and a Report Summary</i>	1026
<i>Building a Report That Uses DATA Step Variables</i>	1030
<i>Examples: REPORT Procedure</i>	1037
<i>Example 1: Selecting Variables for a Report</i>	1037
<i>Example 2: Ordering the Rows in a Report</i>	1040
<i>Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable</i>	1043
<i>Example 4: Consolidating Multiple Observations into One Row of a Report</i>	1047
<i>Example 5: Creating a Column for Each Value of a Variable</i>	1049
<i>Example 6: Displaying Multiple Statistics for One Variable</i>	1053
<i>Example 7: Storing and Reusing a Report Definition</i>	1055
<i>Example 8: Condensing a Report into Multiple Panels</i>	1058
<i>Example 9: Writing a Customized Summary on Each Page</i>	1060
<i>Example 10: Calculating Percentages</i>	1064
<i>Example 11: How PROC REPORT Handles Missing Values</i>	1067
<i>Example 12: Creating and Processing an Output Data Set</i>	1070
<i>Example 13: Storing Computed Variables as Part of a Data Set</i>	1072
<i>Example 14: Using a Format to Create Groups</i>	1075
<i>Example 15: Specifying Style Elements for ODS Output in the PROC REPORT Statement</i>	1078
<i>Example 16: Specifying Style Elements for ODS Output in Multiple Statements</i>	1083

Overview: REPORT Procedure

What Does the REPORT Procedure Do?

The REPORT procedure combines features of the PRINT, MEANS, and TABULATE procedures with features of the DATA step in a single report-writing tool that can produce a variety of reports. You can use PROC REPORT in three ways:

- in a windowing environment with a prompting facility that guides you as you build a report.
- in a windowing environment without the prompting facility.
- in a nonwindowing environment. In this case, you submit a series of statements with the PROC REPORT statement, just as you do in other SAS procedures. You can submit these statements from the Program Editor with the NOWINDOWS option in the PROC REPORT statement, or you can run SAS in batch, noninteractive, or interactive line mode (see the information about running SAS in *SAS Language Reference: Concepts*).

This documentation provides reference information about using PROC REPORT in a windowing or nonwindowing environment. For task-oriented documentation for the nonwindowing environment, see SAS Technical Report P-258, *Using the REPORT Procedure in a Nonwindowing Environment*, Release 6.07.

What Types of Reports Can PROC REPORT Produce?

A *detail report* contains one row for every observation selected for the report. Each of these rows is a *detail row*. A *summary report* consolidates data so that each row represents multiple observations. Each of these rows is also called a detail row.

Both detail and summary reports can contain *summary lines* as well as detail rows. A summary line summarizes numerical data for a set of detail rows or for all detail rows. PROC REPORT provides both default and customized summaries (see “Using Break Lines” on page 952).

This overview illustrates the kinds of reports that PROC REPORT can produce. The statements that create the data sets and formats used in these reports are in Example 1 on page 1037. The formats are stored in a permanent SAS data library. See “Examples: REPORT Procedure” on page 1037 for more reports and for the statements that create them.

What Do the Various Types of Reports Look Like?

The data set that these reports use contains one day’s sales figures for eight stores in a chain of grocery stores.

A simple PROC REPORT step produces a report similar to one produced by a simple PROC PRINT step. Figure 38.1 on page 940 illustrates the simplest kind of report that you can produce with PROC REPORT. The statements that produce the report follow. The data set and formats that the program uses are created in Example 1 on page 1037. Although the WHERE and FORMAT statements are not essential, here they limit the amount of output and make the values easier to understand.

```
libname proclib 'SAS-data-library';
```

```

options nodate pageno=1 linesize=64 pagesize=60
      fmtsearch=(proclib);

proc report data=grocery nowd;
  where sector='se';
  format sector $sctrfmt. manager $mgrfmt.
         dept $deptfmt. sales dollar10.2;
run;

```

Figure 38.1 Simple Detail Report with a Detail Row for Each Observation

The SAS System			
Sector	Manager	Department	Sales
Southeast	Smith	Paper	\$50.00
Southeast	Smith	Meat/Dairy	\$100.00
Southeast	Smith	Canned	\$120.00
Southeast	Smith	Produce	\$80.00
Southeast	Jones	Paper	\$40.00
Southeast	Jones	Meat/Dairy	\$300.00
Southeast	Jones	Canned	\$220.00
Southeast	Jones	Produce	\$70.00

The report in Figure 38.2 on page 940 uses the same observations as those in Figure 38.1 on page 940. However, the statements that produce this report

- ☐ order the rows by the values of Manager and Department
- ☐ create a default summary line for each value of Manager
- ☐ create a customized summary line for the whole report. A customized summary lets you control the content and appearance of the summary information, but you must write additional PROC REPORT statements to create one.

For an explanation of the program that produces this report, see Example 2 on page 1040.

Figure 38.2 Ordered Detail Report with Default and Customized Summaries

Sales for the Southeast Sector		
Manager	Department	Sales

Jones	Paper	\$40.00
	Canned	\$220.00
	Meat/Dairy	\$300.00
	Produce	\$70.00

Jones		\$630.00

Smith	Paper	\$50.00
	Canned	\$120.00
	Meat/Dairy	\$100.00
	Produce	\$80.00

Smith		\$350.00

Total sales for these stores were: \$980.00		

The summary report in Figure 38.3 on page 941 contains one row for each store in the northern sector. Each detail row represents four observations in the input data set, one observation for each department. Information about individual departments does not appear in this report. Instead, the value of Sales in each detail row is the sum of the values of Sales in all four departments. In addition to consolidating multiple observations into one row of the report, the statements that create this report

- customize the text of the column headers
- create default summary lines that total the sales for each sector of the city
- create a customized summary line that totals the sales for both sectors.

For an explanation of the program that produces this report, see Example 4 on page 1047.

Figure 38.3 Summary Report with Default and Customized Summaries

Sales Figures for Northern Sectors			Default summary line for Sector
			1
Sector	Manager	Sales	
-----	-----	-----	
Northeast	Alomar	786.00	
	Andrews	1,045.00	

		\$1,831.00	←
Northwest	Brown	598.00	
	Pelfrey	746.00	
	Reveiz	1,110.00	

		\$2,454.00	
Combined sales for the northern sectors were \$4,285.00.			←
			Customized summary line for the whole report

The summary report in Figure 38.4 on page 942 is similar to Figure 38.3 on page 941. The major difference is that it also includes information for individual departments. Each selected value of Department forms a column in the report. In addition, the statements that create this report

- compute and display a variable that is not in the input data set
- double-space the report
- put blank lines in some of the column headers.

For an explanation of the program that produces this report, see Example 5 on page 1049.

Figure 38.4 Summary Report with a Column for Each Value of a Variable

Sales Figures for Perishables in Northern Sectors				
Sector	Manager	Department		Perishable Total
		Meat/Dairy	Produce	
Northeast	Alomar	\$190.00	\$86.00	\$276.00
	Andrews	\$300.00	\$125.00	\$425.00
Northwest	Brown	\$250.00	\$73.00	\$323.00
	Pelfrey	\$205.00	\$76.00	\$281.00
	Reveiz	\$600.00	\$30.00	\$630.00
Combined sales for meat and dairy : \$1,545.00 Combined sales for produce : \$390.00 Combined sales for all perishables: \$1,935.00				

The customized report in Figure 38.5 on page 943 shows each manager's store on a separate page. Only the first two pages appear here. The statements that create this report create

- ☐ a customized header for each page of the report
- ☐ a computed variable (Profit) that is not in the input data set
- ☐ a customized summary with text that is dependent on the total sales for that manager's store.

For an explanation of the program that produces this report, see Example 9 on page 1060.

Figure 38.5 Customized Summary Report

Detail row

Computed variable

Sales for Individual Stores 1

Northeast Sector
Store managed by Alomar

Department	Sales	Profit
Canned	\$420.00	\$168.00
Meat/Dairy	\$190.00	\$47.50
Paper	\$90.00	\$36.00
Produce	\$86.00	\$21.50
	\$786.00	\$196.50

Customized summary line for Manager: Sales are in the target region.

Default summary line for Manager

Detail row

Computed variable

Sales for Individual Stores 2

Northeast Sector
Store managed by Andrews

Department	Sales	Profit
Canned	\$420.00	\$168.00
Meat/Dairy	\$300.00	\$75.00
Paper	\$200.00	\$80.00
Produce	\$125.00	\$31.25
	\$1,045.00	\$261.25

Customized summary line for Manager: Sales exceeded goal!

Default summary line for Manager

The report in Figure 38.6 on page 944 uses customized style elements to control things like font faces, font sizes, and justification, as well as the width of the border of the table and the width of the spacing between cells. This report was created by using the HTML destination of the Output Delivery System (ODS) and the STYLE= option in several statements in the procedure.

For an explanation of the program that produces this report, see Example 16 on page 1083. For information on ODS, see “Output Delivery System” on page 32.

Figure 38.6 HTML Output

Sales for the Southeast Sector		
Manager	Department	Sales
Jones	Paper	40
	Canned	220
	Meat/Dairy	300
	Produce	70
Jones		630
Subtotal for Jones is \$630.00.		
Smith	Paper	50
	Canned	120
	Meat/Dairy	100
	Produce	80
Smith		350
Subtotal for Smith is \$350.00.		
Total for all departments is: \$980.00.		

Concepts: REPORT Procedure

Laying Out a Report

Report writing is simplified if you approach it with a clear understanding of what you want the report to look like. The most important thing to determine is the layout of the report. To design the layout, ask yourself the following kinds of questions:

- ☐ What do I want to display in each column of the report?
- ☐ In what order do I want the columns to appear?
- ☐ Do I want to display a column for each value of a particular variable?
- ☐ Do I want a row for every observation in the report, or do I want to consolidate information for multiple observations into one row?
- ☐ In what order do I want the rows to appear?

When you understand the layout of the report, use the COLUMN and DEFINE statements in PROC REPORT to construct the layout.

The COLUMN statement lists the items that appear in the columns of the report, describes the arrangement of the columns, and defines headers that span multiple columns. A report item can be

- a data set variable
- a statistic calculated by the procedure
- a variable that you compute from other items in the report.

Omit the COLUMN statement if you want to include all variables in the input data set in the same order as they occur in the data set.

Note: If you start PROC REPORT in the windowing environment without the COLUMN statement, then the initial report includes only as many variables as will fit on one page. △

The DEFINE statement (or, in the windowing environment, the DEFINITION window) defines the characteristics of an item in the report. These characteristics include how PROC REPORT uses the item in the report, the text of the column header, and the format to use to display values.

Usage of Variables in a Report

Much of a report's layout is determined by the usages that you specify for variables in the DEFINE statements or DEFINITION windows. For data set variables, these usages are

DISPLAY

ORDER

ACROSS

GROUP

ANALYSIS

A report can contain variables that are not in the input data set. These variables must have a usage of COMPUTED.

Display Variables

A report that contains one or more display variables has a row for every observation in the input data set. Display variables do not affect the order of the rows in the report. If no order variables appear to the left of a display variable, then the order of the rows in the report reflects the order of the observations in the data set. By default, PROC REPORT treats all character variables as display variables.

Featured in: Example 1 on page 1037

Order Variables

A report that contains one or more order variables has a row for every observation in the input data set. If no display variable appears to the left of an order variable, then PROC REPORT orders the detail rows according to the ascending, formatted values of the order variable. You can change the default order with ORDER= and DESCENDING in the DEFINE statement or with the DEFINITION window.

If the report contains multiple order variables, then PROC REPORT establishes the order of the detail rows by sorting these variables from left to right in the report. PROC REPORT does not repeat the value of an order variable from one row to the next if the value does not change, unless an order variable to its left changes values.

Featured in: Example 2 on page 1040

Across Variables

PROC REPORT creates a column for each value of an across variable. PROC REPORT orders the columns by the ascending, formatted values of the across variable. You can change the default order with ORDER= and DESCENDING in the DEFINE statement or with the DEFINITION window. If no other variable helps define the column (see “COLUMN Statement” on page 981), then PROC REPORT displays the N statistic (the number of observations in the input data set that belong to that cell of the report).

If you are familiar with procedures that use class variables, then you will see that across variables are class variables that are used in the column dimension.

Featured in: Example 5 on page 1049

Group Variables

If a report contains one or more group variables, then PROC REPORT tries to consolidate into one row all observations from the data set that have a unique combination of formatted values for all group variables.

When PROC REPORT creates groups, it orders the detail rows by the ascending, formatted values of the group variable. You can change the default order with ORDER= and DESCENDING in the DEFINE statement or with the DEFINITION window.

If the report contains multiple group variables, then the REPORT procedure establishes the order of the detail rows by sorting these variables from left to right in the report. PROC REPORT does not repeat the values of a group variable from one row to the next if the value does not change, unless a group variable to its left changes values.

If you are familiar with procedures that use class variables, then you will see that group variables are class variables that are used in the row dimension.

Note: You cannot always create groups. PROC REPORT cannot consolidate observations into groups if the report contains any order variables or any display variables that do not have one or more statistics associated with them (see “COLUMN Statement” on page 981). In the windowing environment, if PROC REPORT cannot immediately create groups, then the procedure changes all display and order variables to group variables so that it can create the group variable that you requested. In the nonwindowing environment, it returns to the SAS log a message that explains why it could not create groups. Instead, it creates a detail report that displays group variables the same way as it displays order variables. Even when PROC REPORT creates a detail report, the variables that you define as group variables retain that usage in their definitions. △

Featured in: Example 4 on page 1047

Analysis Variables

An analysis variable is a numeric variable that is used to calculate a statistic for all the observations represented by a cell of the report. (Across variables, in combination with group variables or order variables, determine which observations a cell represents.) You associate a statistic with an analysis variable in the variable’s definition or in the COLUMN statement. By default, PROC REPORT uses numeric variables as analysis variables that are used to calculate the Sum statistic.

The value of an analysis variable depends on where it appears in the report:

- In a detail report, the value of an analysis variable in a detail row is the value of the statistic associated with that variable calculated for a single observation. Calculating a statistic for a single observation is not practical; however, using the variable as an analysis variable enables you to create summary lines for sets of observations or for all observations.

- In a summary report, the value displayed for an analysis variable is the value of the statistic that you specify calculated for the set of observations represented by that cell of the report.
- In a summary line for any report, the value of an analysis variable is the value of the statistic that you specify calculated for all observations represented by that cell of the summary line.

See also: “BREAK Statement” on page 973 and “RBREAK Statement” on page 996

Featured in: Example 2 on page 1040, Example 3 on page 1043, Example 4 on page 1047, and Example 5 on page 1049

Note: Be careful when you use SAS dates in reports that contain summary lines. SAS dates are numeric variables. Unless you explicitly define dates as some other kind of variable, PROC REPORT summarizes them. △

Computed Variables

Computed variables are variables that you define for the report. They are not in the input data set, and PROC REPORT does not add them to the input data set. However, computed variables are included in an output data set if you create one.

In the windowing environment, you add a computed variable to a report from the COMPUTED VAR window.

In the nonwindowing environment, you add a computed variable by

- including the computed variable in the COLUMN statement
- defining the variable’s usage as COMPUTED in the DEFINE statement
- computing the value of the variable in a compute block associated with the variable.

Featured in: Example 5 on page 1049, Example 10 on page 1064, and Example 13 on page 1072

Interactions of Position and Usage

The position and usage of each variable in the report determine the report’s structure and content. PROC REPORT orders the detail rows of the report according to the values of order and group variables, considered from left to right in the report. Similarly, PROC REPORT orders columns for an across variable from left to right, according to the values of the variable.

Several items can collectively define the contents of a column in a report. For instance, in Figure 38.7 on page 948, the values that appear in the third and fourth columns are collectively determined by Sales, an analysis variable, and by Department, an across variable. You create this kind of report with the COLUMN statement or, in the windowing environment, by placing report items above or below each other. This is called stacking items in the report because each item generates a header, and the headers are stacked one above the other.

Figure 38.7 Stacking Department and Sales

Sales Figures for Perishables in Northern Sectors				
Sector	Manager	Department		Perishable Total
		Meat/Dairy	Produce	
Northeast	Alomar	\$190.00	\$86.00	\$276.00
	Andrews	\$300.00	\$125.00	\$425.00
Northwest	Brown	\$250.00	\$73.00	\$323.00
	Pelfrey	\$205.00	\$76.00	\$281.00
	Reveiz	\$600.00	\$30.00	\$630.00

When you use multiple items to define the contents of a column, at most one of the following can be in a column:

- ☐ a display variable with or without a statistic above or below it
- ☐ an analysis variable with or without a statistic above or below it
- ☐ an order variable
- ☐ a group variable
- ☐ a computed variable.

More than one of these items in a column creates a conflict for PROC REPORT about which values to display.

Table 38.1 on page 948 shows which report items can share a column.

Note: You cannot stack order variables with other report items. △

Table 38.1 Report Items That Can Share Columns

	Display	Analysis	Order	Group	Computed	Across	Statistic
Display						X*	X
Analysis						X	X
Order							
Group						X	
Computed variable						X	
Across	X*	X			X	X	X
Statistic	X	X				X	

*When a display variable and an across variable share a column, the report must also contain another variable that is not in the same column.

The following items can stand alone in a column:

- ☐ display variable
- ☐ analysis variable
- ☐ order variable
- ☐ group variable
- ☐ computed variable

- ☐ across variable
- ☐ N statistic.

Note: The values in a column that is occupied only by an across variable are frequency counts. Δ

Statistics That Are Available in PROC REPORT

Descriptive statistic keywords

CSS	PCTSUM
CV	RANGE
MAX	STDDEV STD
MEAN	STDERR
MIN	SUM
N	SUMWGT
NMISS	USS
PCTN	VAR

Quantile statistic keywords

MEDIAN P50	Q3 P75
P1	P90
P5	P95
P10	P99
Q1 P25	QRANGE

Hypothesis testing keyword

PROBT	T
-------	---

These statistics, the formulas that are used to calculate them, and their data requirements are discussed in “Keywords and Formulas” on page 1578.

To compute standard error and the Student’s *t*-test you must use the default value of VARDEF=, which is DF.

Every statistic except N must be associated with a variable. You associate a statistic with a variable either by placing the statistic above or below a numeric display variable or by specifying the statistic as a usage option in the DEFINE statement or in the DEFINITION window for an analysis variable.

You can place N anywhere because it is the number of observations in the input data set that contribute to the value in a cell of the report. The value of N does not depend on a particular variable.

Note: If you use the MISSING option in the PROC REPORT statement, then N includes observations with missing group, order, or across variables. Δ

Using Compute Blocks

A *compute block* is one or more programming statements that appear either between a COMPUTE and an ENDCOMP statement or in a COMPUTE window. PROC

REPORT executes these statements as it builds the report. A compute block can be associated with a report item (a data set variable, a statistic, or a computed variable) or with a location (at the top or bottom of the report; before or after a set of observations). You create a compute block with the COMPUTE window or with the COMPUTE statement. One form of the COMPUTE statement associates the compute block with a report item. Another form associates the compute block with a location in the report (see “Using Break Lines” on page 952).

Note: When you use the COMPUTE statement, you do not have to use a corresponding BREAK or RBREAK statement. (See Example 2 on page 1040, which uses COMPUTE AFTER but does not use the RBREAK statement). Use these statements only when you want to implement one or more BREAK statement or RBREAK statement options (see Example 9 on page 1060, which uses both COMPUTE AFTER MANAGER and BREAK AFTER MANAGER. △

The Purpose of Compute Blocks

A compute block that is associated with a report item can

- define a variable that appears in a column of the report but is not in the input data set
- define display attributes for a report item (see “CALL DEFINE Statement” on page 979).

A compute block that is associated with a location can write a customized summary.

In addition, all compute blocks can use most SAS language elements to perform calculations (see “The Contents of Compute Blocks” on page 950). A PROC REPORT step can contain multiple compute blocks, but they cannot be nested.

The Contents of Compute Blocks

In the windowing environment, a compute block is in a COMPUTE window. In the nonwindowing environment, a compute block begins with a COMPUTE statement and ends with an ENDCOMP statement. Within a compute block, you can use these SAS language elements:

- DM statement
- %INCLUDE statement
- these DATA step statements:

ARRAY	IF-THEN/ELSE
assignment	LENGTH
CALL	RETURN
DO (all forms)	SELECT
END	sum

- comments
- null statements
- macro variables and macro invocations
- all DATA step functions.

For information about SAS language elements see the appropriate section in *SAS Language Reference: Dictionary*.

Within a compute block, you can also use these PROC REPORT features:

- Compute blocks for a customized summary can contain one or more **LINE** statements, which place customized text and formatted values in the summary. (See “LINE Statement” on page 995.)
- Compute blocks for a report item can contain one or more **CALL DEFINE** statements, which set attributes like color and format each time a value for the item is placed in the report. (See “CALL DEFINE Statement” on page 979.)
- Any compute block can contain the automatic variable `_BREAK_` (see “The Automatic Variable `_BREAK_`” on page 953.

Four Ways to Reference Report Items in a Compute Block

A compute block can reference any report item that forms a column in the report (whether or not the column is visible). You reference report items in a compute block in one of four ways:

- by name.
- by a compound name that identifies both the variable and the name of the statistic that you calculate with it. A compound name has this form

variable-name.statistic

- by an alias that you create in the **COLUMN** statement or in the **DEFINITION** window.
- by column number, in the form

`'_Cn_'`

where *n* is the number of the column (from left to right) in the report.

Note: Even though the columns that you define with **NOPRINT** and **NOZERO** do not appear in the report, you must count them when you are referencing columns by number. See the discussion of **NOPRINT** on page 990 and **NOZERO** on page 991. Δ

Note: Referencing variables that have missing values leads to missing values. If a compute block references a variable that has a missing value, then PROC REPORT displays that variable as a blank (for character variables) or as a period (for numeric variables). Δ

The following table shows how to use each type of reference in a compute block.

If the variable that you reference is this type...	Then refer to it by...	For example...
group	name*	Department
order	name*	Department
computed	name*	Department
display	name*	Department
display sharing a column with a statistic	a compound name*	Sales.sum
analysis	a compound name*	Sales.mean
any type sharing a column with an across variable	column number**	'_c3_'

If the variable that you reference is this type...	Then refer to it by...	For example...
*If the variable has an alias, then you must reference it with the alias.		
**Even if the variable has an alias, you must reference it by column number.		

Featured in: Example 3 on page 1043, which references analysis variables by their aliases; Example 5 on page 1049, which references variables by column number; and Example 10 on page 1064, which references group variables and computed variables by name.

Compute Block Processing

PROC REPORT processes compute blocks in two different ways.

- If a compute block is associated with a location, then PROC REPORT executes the compute block only at that location. Because PROC REPORT calculates statistics for groups before it actually constructs the rows of the report, statistics for sets of detail rows are available before or after the rows are displayed, as are values for any variables based on these statistics.
- If a compute block is associated with a report item, then PROC REPORT executes the compute block on every row of the report when it comes to the column for that item. The value of a computed variable in any row of a report is the last value assigned to that variable during that execution of the DATA step statements in the compute block. PROC REPORT assigns values to the columns in a row of a report from left to right. Consequently, you cannot base the calculation of a computed variable on any variable that appears to its right in the report.

Note: PROC REPORT recalculates computed variables at breaks. For details on compute block processing see “How PROC REPORT Builds a Report” on page 1024. △

Using Break Lines

Break lines are lines of text (including blanks) that appear at particular locations, called *breaks*, in a report. A report can contain multiple breaks. Generally, break lines are used to visually separate parts of a report, to summarize information, or both. They can occur

- at the beginning or end of a report
- at the top or bottom of each page
- between sets of observations (whenever the value of a group or order variable changes).

Break lines can contain

- text
- values calculated for either a set of rows or for the whole report.

Creating Break Lines

There are two ways to create break lines. The first way is simpler. It produces a default summary. The second way is more flexible. It produces a customized summary

and provides a way to slightly modify a default summary. Default summaries and customized summaries can appear at the same location in a report.

Default summaries are produced with the BREAK statement, the RBREAK statement, or the BREAK window. You can use default summaries to visually separate parts of the report, to summarize information for numeric variables, or both. Options provide some control over the appearance of the break lines, but if you choose to summarize numeric variables, then you have no control over the content and the placement of the summary information. (A break line that summarizes information is a summary line.)

Customized summaries are produced in a compute block. You can control both the appearance and content of a customized summary, but you must write the code to do so.

Order of Break Lines

You control the order of the lines in a customized summary. However, PROC REPORT controls the order of lines in a default summary and the placement of a customized summary relative to a default summary. When a default summary contains multiple break lines, the order in which the break lines appear is

- 1 overlining or double overlining (in traditional SAS monospace output only)
- 2 summary line
- 3 underlining or double underlining (in traditional SAS monospace output only)
- 4 blank line
- 5 page break.

In traditional SAS monospace output only, if you define a customized summary for the same location, then customized break lines appear after underlining or double underlining.

The Automatic Variable `_BREAK_`

PROC REPORT automatically creates a variable called `_BREAK_`. This variable contains

- ☐ a blank if the current line is not part of a break
- ☐ the value of the break variable if the current line is part of a break between sets of observations
- ☐ the value **RBREAK** if the current line is part of a break at the beginning or end of the report
- ☐ the value `_PAGE_` if the current line is part of a break at the beginning or end of a page.

Using Style Elements in PROC REPORT

If you use the Output Delivery System to create HTML, RTF, or Printer output from PROC REPORT, then you can specify style elements for the procedure to use for various parts of the report. Style elements determine presentation attributes like font face, font weight, color, and so forth. For information about the attributes that you can set for a style, see *SAS Output Delivery System User's Guide*.

You specify style elements for PROC REPORT with the `STYLE=` option. The general form of the `STYLE=` option is

STYLE<(location(s))>=<style-element-name><[style-attribute-specification(s)]>

Note: You can use braces ({ and }) instead of square brackets ([and]). Δ

location(s)

identifies the part of the report that the `STYLE=` option affects. The following table shows what parts of a report are affected by values of *location*.

Table 38.2 Location Values

Location Value	Part of Report Affected
CALLDEF	Cells identified by a CALL DEFINE statement
COLUMN	Column cells
HEADER HDR	Column headers
LINES	Lines generated by LINE statements
REPORT	Report as a whole
SUMMARY	Summary lines

The valid and default values for *location* vary by what statement the STYLE= option appears in. Table 38.3 on page 954 shows valid and default values for *location* for each statement. To specify more than one value of *location* in the same STYLE= option, separate each value with a space.

style-element-name

is the name of a style element that is part of a style definition that is registered with the Output Delivery System. SAS provides some style definitions. Users can create their own style definitions with the TEMPLATE procedure (see *SAS Output Delivery System User's Guide* for information about PROC TEMPLATE). The following table shows the default style elements for each statement.

Table 38.3 Locations and Default Style Elements for Each Statement in PROC REPORT

Statement	Valid Location Values	Default Location Value	Default Style Element
PROC REPORT	REPORT, COLUMN, HEADER HDR, SUMMARY, LINES, CALLDEF	REPORT	Table
BREAK	SUMMARY, LINES	SUMMARY	DataEmphasis
CALL DEFINE	CALLDEF	CALLDEF	Data
COMPUTE	LINES	LINES	NoteContent
DEFINE	COLUMN, HEADER HDR	COLUMN and HEADER	COLUMN: Data HEADER: Header
RBREAK	SUMMARY, LINES	SUMMARY	DataEmphasis

style-attribute-specification(s)

describes the style attribute to change. Each *style-attribute-specification* has this general form:

style-attribute-name=style-attribute-value

To specify more than one *style-attribute-specification*, separate each one with a space.

The following table shows valid values of *style-attribute-name* for the REPORT location. Note that not all style attributes are valid in all destinations. See *SAS Output Delivery System User's Guide* for more information on these style attributes, their valid values, and their applicable destinations.

BACKGROUND=	FONT_WIDTH=*
BACKGROUNDIMAGE=	FOREGROUND=*
BORDERCOLOR=	FRAME=
BORDERCOLORDARK=	HTMLCLASS=
BORDERCOLORLIGHT=	JUST=
BORDERWIDTH=	OUTPUTWIDTH=
CELLPADDING=	POSTHTML=
CELLSPACING=	POSTIMAGE=
FONT=*	POSTTEXT=
FONT_FACE=*	PREHTML=
FONT_SIZE=*	PREIMAGE=
FONT_STYLE=*	PRETEXT=
FONT_WEIGHT=*	RULES=

* When you use these attributes in this location, they affect only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the foreground color or the font for the text that appears in the table, you must set the corresponding attribute in a location that affects the cells rather than the table.

The following table shows valid values of *style-attribute-name* for the CALLDEF, COLUMN, HEADER, LINES, and SUMMARY locations. Note that not all style attributes are valid in all destinations. See *SAS Output Delivery System User's Guide* for more information on these style attributes, their valid values, and their applicable destinations.

ASIS=	FONT_WIDTH=
BACKGROUND=	HREFTARGET=
BACKGROUNDIMAGE=	HTMLCLASS=
BORDERCOLOR=	JUST=
BORDERCOLORDARK=	NOBREAKSPACE=
BORDERCOLORLIGHT=	POSTHTML=
BORDERWIDTH=	POSTIMAGE=
CELLHEIGHT=	POSTTEXT=
CELLWIDTH=	PREHTML=
FLYOVER=	PREIMAGE=
FONT=	PRETEXT=
FONT_FACE=	PROTECTSPECIALCHARS=
FONT_SIZE=	TAGATTR=
FONT_STYLE=	URL=
FONT_WEIGHT=	VJUST=

Specifications in a statement other than the PROC REPORT statement override the same specification in the PROC REPORT statement. However, any style attributes that you specify in the PROC REPORT statement and do not override in another statement are inherited. For instance, if you specify a blue background and a white foreground for all column headings in the PROC REPORT statement, and you specify a gray background for the column headings of a variable in the DEFINE statement, then the background for that particular column heading is gray, and the foreground is white (as specified in the PROC REPORT statement).

You can use a format to assign a style attribute value. For example, the following code assigns a red background color to cells in the Profit column for which the value is negative, and a green background color where the values are positive:

```
proc format;
    value proffmt low-<0='red'
                    0-high='green';
run;
ods html body='external-HTML-file';
proc report data=profits nowd;
    title 'Profits for Individual Stores';
    column Store Profit;
    define Store / display 'Store';
    define Profit / display 'Profit' style=[background=proffmt.];
run;
ods html close;
```

Printing a Report

Printing with ODS

Printing reports with the Output Delivery System is much simpler and provides more attractive output than the older methods of printing that are documented here. For best results, use an output destination such as Printer or RTF. For details on these destinations and on using the ODS statement, see *SAS Output Delivery System User's Guide*.

Printing from the REPORT Window

By default, if you print from the REPORT window, then the report is routed directly to your printer. If you want, you can specify a form to use for printing (see “Printing with a Form” on page 956). Forms specify things like the type of printer that you are using, text format, and page orientation.

Note: Forms are available only when you run SAS from a windowing environment. \triangle

Operating Environment Information: Printing is implemented differently in different operating environments. For information related to printing, consult *SAS Language Reference: Concepts*. Additional information may be available in the SAS documentation for your operating environment. \triangle

Printing with a Form

To print with a form from the REPORT window:

- 1 Specify a form. You can specify a form with the FORMNAME command or, in some cases, through the **File** menu.
- 2 Specify a print file if you want the output to go to a file instead of directly to the printer. You can specify a print file with the PRTPFILE command or, in some cases, through the **File** menu.
- 3 Issue the PRINT or PRINT PAGE command from the command line or from the **File** menu.
- 4 If you specified a print file, then do the following:
 - a Free the print file. You can free a file with the FREE command or, in some cases, through **Print utilities** in the **File** menu. You cannot view or print the file until you free it.
 - b Use operating environment commands to send the file to the printer.

Printing from the Output Window

If you are running PROC REPORT with the NOWINDOWS option, then the default destination for the output is the Output window. Use the commands in the **File** menu to print the report.

Printing from Noninteractive or Batch Mode

If you use noninteractive or batch mode, then SAS writes the output either to the display or to external files, depending on the operating environment and on the SAS options that you use. Refer to the SAS documentation for your operating environment for information about how these files are named and where they are stored.

You can print the output file directly or use PROC PRINTTO to redirect the output to another file. In either case, no form is used, but carriage control characters are written if the destination is a print file.

Use operating environment commands to send the file to the printer.

Printing from Interactive Line Mode

If you use interactive line mode, then by default the output and log are displayed on the screen immediately following the programming statements. Use PROC PRINTTO to redirect the output to an external file. Then use operating environment commands to send the file to the printer.

Using PROC PRINTTO

PROC PRINTTO defines destinations for the SAS output and the SAS log (see Chapter 33, “The PRINTTO Procedure,” on page 879).

PROC PRINTTO does not use a form, but it does write carriage control characters if you are writing to a print file.

Note: You need two PROC PRINTTO steps. The first PROC PRINTTO step precedes the PROC REPORT step. It redirects the output to a file. The second PROC PRINTTO step follows the PROC REPORT step. It reestablishes the default destination and frees the output file. You cannot print the file until PROC PRINTTO frees it. Δ

Storing and Reusing a Report Definition

The OUTREPT= option in the PROC REPORT statement stores a report definition in the specified catalog entry. If you are working in the nonwindowing environment, then

the definition is based on the PROC REPORT step that you submit. If you are in the windowing environment, then the definition is based on the report that is in the REPORT window when you end the procedure. SAS assigns an entry type of REPT to the entry.

In the windowing environment, you can save the definition of the current report by selecting

File ► **Save Report**

A report definition may differ from the SAS program that creates the report (see the discussion of OUTREPT= on page 968).

You can use a report definition to create an identically structured report for any SAS data set that contains variables with the same names as the ones that are used in the report definition. Use the REPORT= option in the PROC REPORT statement to load a report definition when you start PROC REPORT. In the windowing environment, load a report definition from the LOAD REPORT window by selecting

File ► **Open Report**

Syntax: REPORT Procedure

Tip: Supports the Output Delivery System. See “Output Delivery System” on page 32 for details.

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, “Statements with the Same Function in Multiple Procedures,” on page 53 for details. You can also use any global statements as well. See “Global Statements” on page 18 for a list.

```
PROC REPORT <option(s)>;
  BREAK location break-variable</ option(s)>;
  BY <DESCENDING> variable-1
    <...<DESCENDING> variable-n> <NOTSORTED>;
  COLUMN column-specification(s);
  COMPUTE location <target>
    </ STYLE=<style-element-name>
    <[style-attribute-specification(s)]>>;
  LINE specification(s);
  . . . select SAS language elements . . .
  ENDCOMP;
  COMPUTE report-item </ type-specification>;
  CALL DEFINE (column-id, 'attribute-name', value);
  . . . select SAS language elements . . .
  ENDCOMP;
  DEFINE report-item / <usage>
    <attribute(s)>
    <option(s)>
    <justification>
    <COLOR=color>
    <'column-header-1' <...column-header-n'>>
    <style>;
```


FREQ *variable*;
RBREAK *location* *</ option(s)>*;
WEIGHT *variable*;

To do this	Use this statement
Produce a default summary at a change in the value of a group or order variable	BREAK
Create a separate report for each BY group	BY
Set the value of an attribute for a particular column in the current row	CALL DEFINE
Describe the arrangement of all columns and of headers that span more than one column	COLUMN
Specify one or more programming statements that PROC REPORT executes as it builds the report	COMPUTE and ENDCOMP
Describe how to use and display a report item	DEFINE
Treat observations as if they appear multiple times in the input data set	FREQ
Provide a subset of features of the PUT statement for writing customized summaries	LINE
Produce a default summary at the beginning or end of a report or at the beginning and end of each BY group	RBREAK
Specify weights for analysis variables in the statistical calculations	WEIGHT

PROC REPORT Statement

PROC REPORT *<option(s)>*;

To do this	Use this option
Specify the input data set	DATA=
Specify the output data set	OUT=
Select the windowing or the nonwindowing environment	WINDOWS NOWINDOWS

To do this	Use this option
Use a report that was created before compute blocks required aliases (before Release 6.11)	NOALIAS
Control the statistical analysis	
Specify the divisor to use in the calculation of variances	VARDEF=
Specify the sample size to use for the P^2 quantile estimation method	QMARKERS=
Specify the quantile estimation method	QMETHOD=
Specify the mathematical definition to calculate quantiles	QNTLDEF=
Exclude observations with nonpositive weight values from the analysis.	EXCLNPWGT
Control classification levels	
Create all possible combinations of the across variable values	COMPLETECOLS NOCOMPLETECOLS
Create all possible combinations of the group variable values	COMPLETEROWS NOCOMPLETEROWS
Control the layout of the report	
Use formatting characters to add line-drawing characters to the report	BOX*
Specify whether to center or left-justify the report and summary text	CENTER NOCENTER
Specify the default number of characters for columns containing computed variables or numeric data set variables	COLWIDTH=*
Define the characters to use as line-drawing characters in the report	FORMCHAR=*
Specify the length of a line of the report	LS=*
Consider missing values as valid values for group, order, or across variables	MISSING
Specify the number of panels on each page of the report	PANELS=*
Specify the number of lines in a page of the report	PS=
Specify the number of blank characters between panels	PSPACE=*
Override options in the DEFINE statement that suppress the display of a column	SHOWALL
Specify the number of blank characters between columns	SPACING=*

To do this	Use this option
Display one value from each column of the report, on consecutive lines if necessary, before displaying another value from the first column	WRAP
Customize column headers	
Underline all column headers and the spaces between them	HEADLINE*
Write a blank line beneath all column headers	HEADSKIP*
Suppress column headers	NOHEADER
Write <i>name=</i> in front of each value in the report, where <i>name=</i> is the column header for the value	NAMED
Specify the split character	SPLIT=
Control ODS output	
Specify one or more style elements (for the Output Delivery System) to use for different parts of the report	STYLE=
Specify text for the HTML or PDF table of contents entry for the output	CONTENTS=
Store and retrieve report definitions, PROC REPORT statements, and your report profile	
Write to the SAS log the PROC REPORT code that creates the current report	LIST
Suppress the building of the report	NOEXEC
Store in the specified catalog the report definition that is defined by the PROC REPORT step that you submit	OUTREPT=
Identify the report profile to use	PROFILE=
Specify the report definition to use	REPORT=
Control the windowing environment	
Display command lines rather than menu bars in all REPORT windows	COMMAND
Identify the library and catalog containing user-defined help for the report	HELP=
Open the REPORT window and start the PROMPT facility	PROMPT

* Traditional SAS monospace output only.

Options

BOX

uses formatting characters to add line-drawing characters to the report. These characters

- ☐ surround each page of the report
- ☐ separate column headers from the body of the report
- ☐ separate rows and columns from each other
- ☐ separate values in a summary line from other values in the same columns
- ☐ separate a customized summary from the rest of the report.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: You cannot use BOX if you use WRAP in the PROC REPORT statement or in the ROPTIONS window or if you use FLOW in any item definition.

See also: the discussion of FORMCHAR= on page 963

Featured in: Example 12 on page 1070

CENTER|NOCENTER

specifies whether to center or left-justify the report and summary text (customized break lines).

PROC REPORT honors the first of these centering specifications that it finds:

- ☐ the CENTER or NOCENTER option in the PROC REPORT statement or the CENTER toggle in the ROPTIONS window
- ☐ the CENTER or NOCENTER option stored in the report definition that is loaded with REPORT= in the PROC REPORT statement
- ☐ the SAS system option CENTER or NOCENTER.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: When CENTER is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

COLWIDTH=column-width

specifies the default number of characters for columns containing computed variables or numeric data set variables.

Default: 9

Range: 1 to the linesize

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: When setting the width for a column, PROC REPORT first looks at WIDTH= in the definition for that column. If WIDTH= is not present, then PROC REPORT uses a column width large enough to accommodate the format for the item. (For information about formats see the discussion of FORMAT= on page 989.)

If no format is associated with the item, then the column width depends on variable type:

If the variable is a...	Then the column width is the...
character variable in the input data set	length of the variable
numeric variable in the input data set	value of the COLWIDTH= option
computed variable (numeric or character)	value of the COLWIDTH= option

Featured in: Example 2 on page 1040

COMMAND

displays command lines rather than menu bars in all REPORT windows.

After you have started PROC REPORT in the windowing environment, you can display the menu bars in the current window by issuing the COMMAND command. You can display the menu bars in all PROC REPORT windows by issuing the PMENU command. The PMENU command affects all the windows in your SAS session. Both of these commands are toggles.

You can store a setting of COMMAND in your report profile. PROC REPORT honors the first of these settings that it finds:

- ☐ the COMMAND option in the PROC REPORT statement
- ☐ the setting in your report profile.

Restriction: This option has no effect in the nonwindowing environment.

COMPLETECOLS|NOCOMPLETECOLS

creates all possible combinations for the values of the across variables even if one or more of the combinations do not occur within the input data set. Consequently, the column headings are the same for all logical pages of the report within a single BY group.

Default: COMPLETECOLS

Interaction: The PRELOADFMT option in the DEFINE statement ensures that PROC REPORT uses all user-defined format ranges for the combinations of across variables, even when a frequency is zero.

COMPLETEROWS|NOCOMPLETEROWS

displays all possible combinations of the values of the group variables, even if one or more of the combinations do not occur in the input data set. Consequently, the row headings are the same for all logical pages of the report within a single BY group.

Default: NOCOMPLETEROWS

Interaction: The PRELOADFMT option in the DEFINE statement ensures that PROC REPORT uses all user-defined format ranges for the combinations of group variables, even when a frequency is zero.

CONTENTS=*'link-text'*

specifies the text for the entries in the HTML contents file or PDF table of contents for the output that is produced by PROC REPORT. For information on HTML and PDF output, see “Output Delivery System” on page 32.

Restriction: For HTML output, the CONTENTS= option has no effect on the HTML body file. It affects only the HTML contents file.

DATA=SAS-*data-set*

specifies the input data set.

Main discussion: “Input Data Sets” on page 19

EXCLNPWGT

excludes observations with nonpositive weight values (zero or negative) from the analysis. By default, PROC REPORT treats observations with negative weights like those with zero weights and counts them in the total number of observations.

Alias: EXCLNPWGTS

Requirement: You must use a WEIGHT statement.

See also: “WEIGHT Statement” on page 1000

FORMCHAR <(position(s))>=*'formatting-character(s)'*

defines the characters to use as line-drawing characters in the report.

position(s)

identifies the position of one or more characters in the SAS formatting-character string. A space or a comma separates the positions.

Default: Omitting (*position(s)*) is the same as specifying all 20 possible SAS formatting characters, in order.

Range: PROC REPORT uses 12 of the 20 formatting characters that SAS provides.

Table 38.4 on page 964 shows the formatting characters that PROC REPORT uses. Figure 38.8 on page 965 illustrates the use of some commonly used formatting character in the output from PROC REPORT.

formatting-character(s)

lists the characters to use for the specified positions. PROC REPORT assigns characters in *formatting-character(s)* to *position(s)*, in the order that they are listed. For instance, the following option assigns the asterisk (*) to the third formatting character, the pound sign (#) to the seventh character, and does not alter the remaining characters:

```
formchar(3,7)='*#'
```

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: The SAS system option FORMCHAR= specifies the default formatting characters. The system option defines the entire string of formatting characters. The FORMCHAR= option in a procedure can redefine selected characters.

Tip: You can use any character in *formatting-characters*, including hexadecimal characters. If you use hexadecimal characters, then you must put an **x** after the closing quotation mark. For instance, the following option assigns the hexadecimal character 2D to the third formatting character, the hexadecimal character 7C to the seventh character, and does not alter the remaining characters:

```
formchar(3,7)='2D7C'x
```

Table 38.4 Formatting Characters Used by PROC REPORT

Position	Default	Used to draw
1		the right and left borders and the vertical separators between columns
2	-	the top and bottom borders and the horizontal separators between rows; also underlining and overlining in break lines as well as the underlining that the HEADLINE option draws
3	-	the top character in the left border
4	-	the top character in a line of characters that separates columns

5	-	the top character in the right border
6		the leftmost character in a row of horizontal separators
7	+	the intersection of a column of vertical characters and a row of horizontal characters
8		the rightmost character in a row of horizontal separators
9	-	the bottom character in the left border
10	-	the bottom character in a line of characters that separate columns
11	-	the bottom character in the right border
13	=	double overlining and double underlining in break lines

Figure 38.8 Formatting Characters in PROC REPORT Output

Sales for Northern Sectors			1
Sector	Manager	Sales	
-----○			2
Northeast	Alomar	786.00	
	Andrews	1,045.00	
		-----○	
		1,831.00	2
		-----○	
Northwest	Brown	598.00	
	Pelfrey	746.00	
	Reveiz	1,110.00	

		2,454.00	

		=====○	
		4,285.00	13
		=====○	

HEADLINE

underlines all column headers and the spaces between them at the top of each page of the report.

The HEADLINE option underlines with the second formatting character. (See the discussion of FORMCHAR= on page 963 .)

Default: hyphen (-)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Tip: In traditional (monospace) SAS output, you can underline column headers without underlining the spaces between them, by using two hyphens ('--') as the last line of each column header instead of using HEADLINE.

Featured in: Example 2 on page 1040 and Example 8 on page 1058

HEADSKIP

writes a blank line beneath all column headers (or beneath the underlining that the HEADLINE option writes) at the top of each page of the report.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Featured in: Example 2 on page 1040

HELP=libref.catalog

identifies the library and catalog containing user-defined help for the report. This help can be in CBT or HELP catalog entries. You can write a CBT or HELP entry for each item in the report with the BUILD procedure in SAS/AF software. Store all such entries for a report in the same catalog.

Specify the entry name for help for a particular report item in the DEFINITION window for that report item or in a DEFINE statement.

Restriction: This option has no effect in the nonwindowing environment or on ODS destinations other than traditional SAS monospace output.

LIST

writes to the SAS log the PROC REPORT code that creates the current report. This listing may differ in these ways from the statements that you submit:

- ☐ It shows some defaults that you may not have specified.
- ☐ It omits some statements that are not specific to the REPORT procedure, whether you submit them with the PROC REPORT step or had previously submitted them. These statements include

BY

FOOTNOTE

FREQ

TITLE

WEIGHT

WHERE

- ☐ It omits these PROC REPORT statement options:

LIST

OUT=

OUTREPT=

PROFILE=

REPORT=

WINDOWS|NOWINDOWS

- ☐ It omits SAS system options.
- ☐ It resolves automatic macro variables.

Restriction: This option has no effect in the windowing environment. In the windowing environment, you can write the report definition for the report that is currently in the REPORT window to the SOURCE window by selecting

Tools

►

Report Statements

LS=*line-size*

specifies the length of a line of the report.

PROC REPORT honors the first of these line size specifications that it finds:

- the LS= option in the PROC REPORT statement or Linesize= in the ROPTIONS window
- the LS= setting stored in the report definition loaded with REPORT= in the PROC REPORT statement
- the SAS system option LINESIZE=.

Range: 64-256 (integer)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Featured in: Example 6 on page 1053 and Example 8 on page 1058

MISSING

considers missing values as valid values for group, order, or across variables. Special missing values used to represent numeric values (the letters A through Z and the underscore () character) are each considered as a different value. A group for each missing value appears in the report. If you omit the MISSING option, then PROC REPORT does not include observations with a missing value for any group, order, or across variables in the report.

See also: For information about special missing values, see the section on missing values in *SAS Language Reference: Concepts*.

Featured in: Example 11 on page 1067

NAMED

writes *name*= in front of each value in the report, where *name* is the column header for the value.

Interaction: When you use the NAMED option, PROC REPORT automatically uses the NOHEADER option.

Tip: Use NAMED in conjunction with the WRAP option to produce a report that wraps all columns for a single row of the report onto consecutive lines rather than placing columns of a wide report on separate pages.

Featured in: Example 7 on page 1055

NOALIAS

lets you use a report that was created before compute blocks required aliases (before Release 6.11). If you use NOALIAS, then you cannot use aliases in compute blocks.

NOCENTER

See CENTER|NOCENTER on page 962.

NOCOMLETECOLS

See COMPLETECOLS|NOCOMLETECOLS on page 963.

NOCOMPLETEROWS

See COMPLETEROWS|NOCOMPLETEROWS on page 963.

NOEXEC

suppresses the building of the report. Use NOEXEC with OUTREPT= to store a report definition in a catalog entry. Use NOEXEC with LIST and REPORT= to display a listing of the specified report definition.

NOHEADER

suppresses column headers, including those that span multiple columns.

When you suppress the display of column headers in the windowing environment, you cannot select any report items.

NOWINDOWS

Alias: NOWD

See WINDOWS|NOWINDOWS on page 973.

OUT=SAS-data-set

names the output data set. If this data set does not exist, then PROC REPORT creates it. The data set contains one observation for each detail row of the report and one observation for each unique summary line. If you use both customized and default summaries at the same place in the report, then the output data set contains only one observation because the two summaries differ only in how they present the data. Information about customization (underlining, color, text, and so forth) is not data and is not saved in the output data set.

The output data set contains one variable for each column of the report. PROC REPORT tries to use the name of the report item as the name of the corresponding variable in the output data set. However, this is not possible if a data set variable is under or over an across variable or if a data set variable appears multiple times in the COLUMN statement without aliases. In these cases, the name of the variable is based on the column number (_C1_, _C2_, and so forth).

Output data set variables that are derived from input data set variables retain the formats of their counterparts in the input data set. PROC REPORT derives labels for these variables from the corresponding column headers in the report unless the only item defining the column is an across variable. In that case, the variables have no label. If multiple items are stacked in a column, then the labels of the corresponding output data set variables come from the analysis variable in the column.

The output data set also contains a character variable named `_BREAK_`. If an observation in the output data set derives from a detail row in the report, then the value of `_BREAK_` is missing. If the observation derives from a summary line, then the value of `_BREAK_` is the name of the break variable that is associated with the summary line, or `_RBREAK_`. If the observation derives from a COMPUTE BEFORE `_PAGE_` or COMPUTE AFTER `_PAGE_` statement, then the value of `_BREAK_` is “`_PAGE_`”.

Interaction: You cannot use OUT= in a PROC REPORT step that uses a BY statement.

Featured in: Example 12 on page 1070 and Example 13 on page 1072

OUTREPT=libref.catalog.entry

stores in the specified catalog entry the REPORT definition that is defined by the PROC REPORT step that you submit. PROC REPORT assigns the entry a type of REPT.

The stored report definition may differ in these ways from the statements that you submit:

- It omits some statements that are not specific to the REPORT procedure, whether you submit them with the PROC REPORT step or whether they are already in effect when you submit the step. These statements include

BY

FOOTNOTE

FREQ

TITLE

WEIGHT

WHERE

- It omits these PROC REPORT statement options:

LIST

NOALIAS

OUT=

OUTREPT=

PROFILE=

REPORT=

WINDOWS|NOWINDOWS

- It omits SAS system options.
- It resolves automatic macro variables.

Featured in: Example 7 on page 1055

PANELS=number-of-panels

specifies the number of panels on each page of the report. If the width of a report is less than half of the line size, then you can display the data in multiple sets of columns so that rows that would otherwise appear on multiple pages appear on the same page. Each set of columns is a *panel*. A familiar example of this kind of report is a telephone book, which contains multiple panels of names and telephone numbers on a single page.

When PROC REPORT writes a multipanel report, it fills one panel before beginning the next.

The number of panels that fits on a page depends on the

- width of the panel
- space between panels
- line size.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output. However, the COLUMNS= option in the ODS PRINTER or ODS PDF statement produces similar results. For details, see the chapter on ODS statements in *SAS Output Delivery System User's Guide*.

Default: 1

Tip: If *number-of-panels* is larger than the number of panels that can fit on the page, then PROC REPORT creates as many panels as it can. Let PROC REPORT put your data in the maximum number of panels that can fit on the page by specifying a large number of panels (for example, 99).

See also: For information about the space between panels and the line size, see the discussions of PSPACE= on page 970 and the discussion of LS= on page 967.

Featured in: Example 8 on page 1058

PROFILE=libref.catalog

identifies the report profile to use. A profile

- specifies the location of menus that define alternative menu bars and pull-down menus for the REPORT and COMPUTE windows.
- sets defaults for WINDOWS, PROMPT, and COMMAND.

PROC REPORT uses the entry REPORT.PROFILE in the catalog that you specify as your profile. If no such entry exists, or if you do not specify a profile, then PROC REPORT uses the entry REPORT.PROFILE in SASUSER.PROFILE. If you have no profile, then PROC REPORT uses default menus and the default settings of the options.

You create a profile from the PROFILE window while using PROC REPORT in a windowing environment. To create a profile

1 Invoke PROC REPORT with the WINDOWS option.

2 Select

Tools ► Report Profile

3 Fill in the fields to suit your needs.

4 Select **OK** to exit the PROFILE window. When you exit the window, PROC REPORT stores the profile in SASUSER.PROFILE.REPORT.PROFILE. Use the CATALOG procedure or the Explorer window to copy the profile to another location.

Note: If, after opening the PROFILE window, you decide not to create a profile, then select **CANCEL** to close the window. \triangle

PROMPT

opens the REPORT window and starts the PROMPT facility. This facility guides you through creating a new report or adding more data set variables or statistics to an existing report.

If you start PROC REPORT with prompting, then the first window gives you a chance to limit the number of observations that are used during prompting. When you exit the prompter, PROC REPORT removes the limit.

Restriction: When you use the PROMPT option, you open the REPORT window.

When the REPORT window is open, you cannot send procedure output to any ODS destination.

Tip: You can store a setting of PROMPT in your report profile. PROC REPORT honors the first of these settings that it finds:

- ☐ the PROMPT option in the PROC REPORT statement
- ☐ the setting in your report profile.

If you omit PROMPT from the PROC REPORT statement, then the procedure uses the setting in your report profile, if you have one. If you do not have a report profile, then PROC REPORT does not use the prompt facility. For information on report profiles, see “PROFILE” on page 1014.

PS=*page-size*

specifies the number of lines in a page of the report.

PROC REPORT honors the first of these page size specifications that it finds:

- ☐ the PS= option in the PROC REPORT statement
- ☐ the PS= setting in the report definition specified with REPORT= in the PROC REPORT statement
- ☐ the SAS system option PAGESIZE=.

Range: 15-32,767 (integer)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Featured in: Example 6 on page 1053 and Example 8 on page 1058

PSPACE=*space-between-panels*

specifies the number of blank characters between panels. PROC REPORT separates all panels in the report by the same number of blank characters. For each panel, the sum of its width and the number of blank characters separating it from the panel to its left cannot exceed the line size.

Default: 4

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Featured in: Example 8 on page 1058

QMARKERS=number

specifies the default number of markers to use for the P^2 estimation method. The number of markers controls the size of fixed memory space.

Default: The default value depends on which quantiles you request. For the median (P50), *number* is 7. For the quartiles (P25 and P75), *number* is 25. For the quantiles P1, P5, P10, P90, P95, or P99, *number* is 105. If you request several quantiles, then PROC REPORT uses the largest default value of *number*.

Range: any odd integer greater than 3

Tip: Increase the number of markers above the default settings to improve the accuracy of the estimates; you can reduce the number of markers to conserve computing resources.

QMETHOD=OS|P2

specifies the method that PROC REPORT uses to process the input data when it computes quantiles. If the number of observations is less than or equal to the value of the QMARKERS= option, and the value of the QNTLDEF= option is 5, then both methods produce the same results.

OS

uses order statistics. This is the technique that PROC UNIVARIATE uses.

Note: This technique can be very memory intensive. Δ

P2

uses the P^2 method to approximate the quantile.

Default: OS

Restriction: When QMETHOD=P2, PROC REPORT does not compute weighted quantiles.

Tip: When QMETHOD=P2, reliable estimates of some quantiles (P1, P5, P95, P99) might not be possible for some data sets such as those with heavily tailed or skewed distributions.

QNTLDEF=number

specifies the mathematical definition that the procedure uses to calculate quantiles when the value of the QMETHOD= option is OS. When QMETHOD=P2, you must use QNTLDEF=5.

Default: 5

Range: any integer from 1 to 5, inclusive

Alias: PCTLDEF=

REPORT=libref.catalog.entry

specifies the report definition to use. PROC REPORT stores all report definitions as entries of type REPT in a SAS catalog.

Interaction: If you use REPORT=, then you cannot use the COLUMN statement.

See also: OUTREPT= on page 968

Featured in: Example 7 on page 1055

SHOWALL

overrides options in the DEFINE statement that suppress the display of a column.

See also: NOPRINT and NOZERO in “DEFINE Statement” on page 985

SPACING=space-between-columns

specifies the number of blank characters between columns. For each column, the sum of its width and the blank characters between it and the column to its left cannot exceed the line size.

Default: 2

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: PROC REPORT separates all columns in the report by the number of blank characters specified by SPACING= in the PROC REPORT statement unless you use SPACING= in the DEFINE statement to change the spacing to the left of a specific item.

Interaction: When CENTER is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

Featured in: Example 2 on page 1040

SPLIT='character'

specifies the split character. PROC REPORT breaks a column header when it reaches that character and continues the header on the next line. The split character itself is not part of the column header although each occurrence of the split character counts toward the 256-character maximum for a label.

Default: slash (/)

Interaction: The FLOW option in the DEFINE statement honors the split character.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Featured in: Example 5 on page 1049

STYLE<(location(s))>=<style-element-name>[<style-attribute-specification(s)>]

specifies the style element to use for the specified locations in the report. See “Using Style Elements in PROC REPORT” on page 953 for details.

Restriction: This option affects only the HTML, RTF, and Printer output.

Featured in: Example 15 on page 1078 and Example 16 on page 1083

VARDEF=divisor

specifies the divisor to use in the calculation of the variance and standard deviation. Table 38.5 on page 972 shows the possible values for *divisor* and associated divisors.

Table 38.5 Possible Values for VARDEF=

Value	Divisor	Formula for Divisor
DF	degrees of freedom	$n - 1$
N	number of observations	n
WDF	sum of weights minus one	$(\sum_i w_i) - 1$
WEIGHT WGT	sum of weights	$\sum_i w_i$

The procedure computes the variance as $CSS/divisor$, where CSS is the corrected sums of squares and equals $\sum (x_i - \bar{x})^2$. When you weight the analysis variables, CSS equals $\sum w_i (x_i - \bar{x}_w)^2$, where \bar{x}_w is the weighted mean.

Default: DF

Requirement: To compute the standard error of the mean and Student's t -test, use the default value of VARDEF=.

Tip: When you use the WEIGHT statement and VARDEF=DF, the variance is an estimate of σ^2 , where the variance of the i th observation is $\text{var}(x_i) = \sigma^2/w_i$ and w_i is the weight for the i th observation. This yields an estimate of the variance of an observation with unit weight.

Tip: When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large n) an estimate of σ^2/\bar{w} , where \bar{w} is the average weight. This yields an asymptotic estimate of the variance of an observation with average weight.

See also: “WEIGHT” on page 59

WINDOWS|NOWINDOWS

selects a windowing or nonwindowing environment.

When you use WINDOWS, SAS opens the REPORT window, which enables you to modify a report repeatedly and to see the modifications immediately. When you use NOWINDOWS, PROC REPORT runs without the REPORT window and sends its output to the open output destination(s).

Alias: WD|NOWD

Restriction: When you use the WINDOWS option, you cannot send procedure output to the HTML, RTF, or Printer destination.

Tip: You can store a setting of WINDOWS in your report profile, if you have one. If you do not specify WINDOWS or NOWINDOWS in the PROC REPORT statement, then the procedure uses the setting in your report profile. If you do not have a report profile, then PROC REPORT looks at the setting of the SAS system option DMS. If DMS is ON, then PROC REPORT uses the windowing environment; if DMS is OFF, then it uses the nonwindowing environment.

See also: For a discussion of the report profile see the discussion of PROFILE= on page 969.

Featured in: Example 1 on page 1037

WRAP

displays one value from each column of the report, on consecutive lines if necessary, before displaying another value from the first column. By default, PROC REPORT displays values for only as many columns as it can fit on one page. It fills a page with values for these columns before starting to display values for the remaining columns on the next page.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: When WRAP is in effect, PROC REPORT ignores PAGE in any item definitions.

Tip: Typically, you use WRAP in conjunction with the NAMED option in order to avoid wrapping column headers.

Featured in: Example 7 on page 1055

BREAK Statement

Produces a default summary at a break (a change in the value of a group or order variable). The information in a summary applies to a set of observations. The observations share a unique

combination of values for the break variable and all other group or order variables to the left of the break variable in the report.

Featured in: Example 4 on page 1047 and Example 5 on page 1049.

BREAK *location break-variable* </ option(s)>;

To do this	Use this option
Specify the color of the break lines in the REPORT window	COLOR=
Double overline each value	DOL*
Double underline each value	DUL*
Overline each value	OL*
Start a new page after the last break line	PAGE
Write a blank line for the last break line	SKIP
Specify a style element for default summary lines, customized summary lines or both	STYLE=
Write a summary line in each group of break lines	SUMMARIZE
Suppress the printing of the value of the break variable in the summary line and of any underlining or overlining in the break lines in the column containing the break variable	SUPPRESS
Underline each value	UL*

* Traditional SAS monospace output only.

Required Arguments

location

controls the placement of the break lines and is either

AFTER

places the break lines immediately after the last row of each set of rows that have the same value for the break variable.

BEFORE

places the break lines immediately before the first row of each set of rows that have the same value for the break variable.

break-variable

is a group or order variable. The REPORT procedure writes break lines each time the value of this variable changes.

Options

COLOR=*color*

specifies the color of the break lines in the REPORT window. You can use the following colors:

BLACK	MAGENTA
BLUE	ORANGE
BROWN	PINK
CYAN	RED
GRAY	WHITE
GREEN	YELLOW

Default: The color of **Foreground** in the SASCOLOR window. (For more information, see the online help for the SASCOLOR window.)

Restriction: This option affects output in the windowing environment only.

Note: Not all operating environments and devices support all colors, and on some operating systems and devices, one color may map to another color. For example, if the DEFINITION window displays the word BROWN in yellow characters, then selecting BROWN results in a yellow item. Δ

DOL

(for double overlining) uses the thirteenth formatting character to overline each value

- ☐ that appears in the summary line
- ☐ that would appear in the summary line if you specified the SUMMARIZE option.

Default: equals sign (=)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the OL and DOL options, then PROC REPORT honors only OL.

See also: the discussion of FORMCHAR= on page 963.

DUL

(for double underlining) uses the thirteenth formatting character to underline each value

- ☐ that appears in the summary line
- ☐ that would appear in the summary line if you specified the SUMMARIZE option.

Default: equals sign (=)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the UL and DUL options, then PROC REPORT honors only UL.

See also: the discussion of FORMCHAR= on page 963.

OL

(for overlining) uses the second formatting character to overline each value

- ☐ that appears in the summary line
- ☐ that would appear in the summary line if you specified the SUMMARIZE option.

Default: hyphen (-)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the OL and DOL options, then PROC REPORT honors only OL.

See also: the discussion of FORMCHAR= on page 963.

Featured in: Example 2 on page 1040 and Example 9 on page 1060

PAGE

starts a new page after the last break line.

Interaction: If you use PAGE in the BREAK statement and you create a break at the end of the report, then the summary for the whole report appears on a separate page.

Featured in: Example 9 on page 1060

SKIP

writes a blank line for the last break line.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Featured in: Example 2 on page 1040, Example 4 on page 1047, Example 5 on page 1049, and Example 8 on page 1058

STYLE<location(s)>=<style-element-name><[style-attribute-specification(s)]>

specifies the style element to use for default summary lines that are created with the BREAK statement. See “Using Style Elements in PROC REPORT” on page 953 for details.

Restriction: This option affects only the HTML, RTF, and Printer output.

SUMMARIZE

writes a summary line in each group of break lines. A summary line for a set of observations contains values for

- ☐ the break variable (which you can suppress with the SUPPRESS option)
- ☐ other group or order variables to the left of the break variable
- ☐ statistics
- ☐ analysis variables
- ☐ computed variables.

The following table shows how PROC REPORT calculates the value for each kind of report item in a summary line that is created by the BREAK statement:

If the report item is...	Then its value is...
the break variable	the current value of the variable (or a missing value if you use SUPPRESS)
a group or order variable to the left of the break variable	the current value of the variable
a group or order variable to the right of the break variable, or a display variable anywhere in the report	missing*
a statistic	the value of the statistic over all observations in the set
an analysis variable	the value of the statistic specified as the usage option in the item's definition. PROC REPORT calculates the value of the statistic over all observations in the set. The default usage is SUM.

If the report item is...	Then its value is...
a computed variable	the results of the calculations based on the code in the corresponding compute block (see “COMPUTE Statement” on page 983).

* If you reference a variable with a missing value in a customized summary line, then PROC REPORT displays that variable as a blank (for character variables) or a period (for numeric variables).

Note: PROC REPORT cannot create groups in a report that contains order or display variables. Δ

Featured in: Example 2 on page 1040, Example 4 on page 1047, and Example 9 on page 1060

SUPPRESS

suppresses printing of

- ☐ the value of the break variable in the summary line
- ☐ any underlining and overlining in the break lines in the column that contains the break variable.

Interaction: If you use SUPPRESS, then the value of the break variable is unavailable for use in customized break lines unless you assign a value to it in the compute block that is associated with the break (see “COMPUTE Statement” on page 983).

Featured in: Example 4 on page 1047

UL

(for underlining) uses the second formatting character to underline each value

- ☐ that appears in the summary line
- ☐ that would appear in the summary line if you specified the SUMMARIZE option.

Default: hyphen (-)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the UL and DUL options, then PROC REPORT honors only UL.

See also: the discussion of FORMCHAR= on page 963.

Order of Break Lines

When a default summary contains more than one break line, the order in which the break lines appear is

- 1 overlining or double overlining (OL or DOL)
- 2 summary line (SUMMARIZE)
- 3 underlining or double underlining (UL or DUL)
- 4 skipped line (SKIP)
- 5 page break (PAGE).

Note: If you define a customized summary for the break, then customized break lines appear after underlining or double underlining. For more information about customized break lines, see “COMPUTE Statement” on page 983 and “LINE Statement” on page 995. Δ

BY Statement

Creates a separate report on a separate page for each BY group.

Restriction: If you use the BY statement, then you must use the NOWINDOWS option in the PROC REPORT statement.

Restriction: You cannot use the OUT= option when you use a BY statement.

Interaction: If you use the RBREAK statement in a report that uses BY processing, then PROC REPORT creates a default summary for each BY group. In this case, you cannot summarize information for the whole report.

Tip: Using the BY statement does not make the FIRST. and LAST. variables available in compute blocks.

Main discussion: “BY” on page 54

```
BY <DESCENDING> variable-1
    <...<DESCENDING> variable-n> <NOTSORTED>;
```

Required Arguments

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then the observations in the data set either must be sorted by all the variables that you specify or must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Options

DESCENDING

specifies that the data set is sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The data are grouped in another way, for example, chronological order.

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

CALL DEFINE Statement

Sets the value of an attribute for a particular column in the current row.

Restriction: Valid only in a compute block that is attached to a report item.

Featured in: Example 4 on page 1047

CALL DEFINE (*column-id* | *_ROW_*, '*attribute-name*', *value*);

The CALL DEFINE statement is often used to write report definitions that other people will use in a windowing environment. Only the FORMAT, URL, URLBP, and URLP attributes have an effect in the nonwindowing environment. In fact, URL, URLBP, and URLP are effective only in the nonwindowing environment. The STYLE= and URL attributes are effective only when you are using the Output Delivery System to create HTML, RTF, or Printer output. (See Table 38.6 on page 979 for descriptions of the available attributes.)

Required Arguments

column-id

specifies a column name or a column number (that is, the position of the column from the left edge of the report). A column ID can be one of the following:

- a character literal (in quotation marks) that is the column name
- a character expression that resolves to the column name
- a numeric literal that is the column number
- a numeric expression that resolves to the column number
- a name of the form '*_Cn_*', where *n* is the column number
- the automatic variable *_COL_*, which identifies the column that contains the report item that the compute block is attached to

attribute-name

is the attribute to define. For attribute names, refer to Table 38.6 on page 979.

ROW

is an automatic variable that indicates the entire current row.

value

sets the value for the attribute. For values for each attribute, refer to Table 38.6 on page 979.

Table 38.6 Attribute Descriptions

Attribute	Description	Values	Affects
BLINK	Controls blinking of current value	1 turns blinking on; 0 turns it off	windowing environment
COLOR	Controls the color of the current value in the REPORT window	'blue', 'red', 'pink', 'green', 'cyan', 'yellow', 'white', 'orange', 'black', 'magenta', 'gray', 'brown'	windowing environment

Attribute	Description	Values	Affects
COMMAND	Specifies that a series of commands follows	a quoted string of SAS commands to submit to the command line	windowing environment
FORMAT	Specifies a format for the column	a SAS format or a user-defined format	windowing and nonwindowing environments
HIGHLIGHT	Controls highlighting of the current value	1 turns highlighting on; 0 turns it off	windowing environment
RVSVIDEO	Controls display of the current value	1 turns reverse video on; 0 turns it off	windowing environment
STYLE=	Specifies the style element for the Output Delivery System	See “Using the STYLE= Attribute” on page 981	HTML, RTF, and Printer output
URL	Makes the contents of each cell of the column a link to the specified Uniform Resource Locator (URL)*	a quoted URL (either single or double quotation marks can be used)	HTML, RTF, and Printer output
URLBP	Makes the contents of each cell of the column a link. The link points to a Uniform Resource Locator that is a concatenation of <ol style="list-style-type: none"> the string that is specified by the BASE= option in the ODS HTML statement the string that is specified by the PATH= option in the ODS HTML statement the value of the URLBP attribute*[#] 	a quoted URL (either single or double quotation marks can be used)	HTML output
URLP	Makes the contents of each cell of the column a link. The link points to a Uniform Resource Locator that is a concatenation of <ol style="list-style-type: none"> the string that is specified by the PATH= option in the ODS HTML statement the value of the URLP attribute*[#] 	a quoted URL (either single or double quotation marks can be used)	HTML output

* The total length of the URL that you specify (including any characters that come from the BASE= and PATH= options) cannot exceed the line size. Use the LS= option in the PROC REPORT statement to alter the line size for the PROC REPORT step.

For information on the BASE= and PATH= options, see the documentation for the ODS HTML statement in *SAS Output Delivery System User's Guide*.

Note: The attributes BLINK, HIGHLIGHT, and RVSVIDEO do not work on all devices. \triangle

Using the STYLE= Attribute

The STYLE= attribute specifies the style element to use in the cells that are affected by the CALL DEFINE statement.

The STYLE= attribute functions like the STYLE= option in other statements in PROC REPORT. However, instead of acting as an option in a statement, it becomes the value for the STYLE= attribute. For instance, the following CALL DEFINE statement sets the background color to yellow and the font size to 7 for the specified column:

```
call define(_col_, "style",
           "style=[background=yellow font_size=7]");
```

See “Using Style Elements in PROC REPORT” on page 953 for details.

Restriction: This option affects only the HTML, RTF, Printer destinations.

Interaction: If you set a style element for the CALLDEF location in the PROC REPORT statement and you want to use that exact style element in a CALL DEFINE statement, then use an empty string as the value for the STYLE attribute, as shown here:

```
call define (_col_, "STYLE", "" );
```

Featured in: Example 16 on page 1083

COLUMN Statement

Describes the arrangement of all columns and of headers that span more than one column.

Restriction: You cannot use the COLUMN statement if you use REPORT= in the PROC REPORT statement.

Featured in: Example 1 on page 1037, Example 3 on page 1043, Example 5 on page 1049, Example 6 on page 1053, Example 10 on page 1064, and Example 11 on page 1067

COLUMN *column-specification(s)*;

Required Arguments

column-specification(s)

is one or more of the following:

- *report-item(s)*
- *report-item-1, report-item-2* <. . . , *report-item-n*>
- ('*header-1*' <. . . '*header-n*'> *report-item(s)*)
- *report-item=name*

where *report-item* is the name of a data set variable, a computed variable, or a statistic. See “Statistics That Are Available in PROC REPORT” on page 949 for a list of available statistics.

report-item(s)

identifies items that each form a column in the report.

Featured in: Example 1 on page 1037 and Example 11 on page 1067

report-item-1, report-item-2 <. . . , report-item-n>

identifies report items that collectively determine the contents of the column or columns. These items are said to be stacked in the report because each item generates a header, and the headers are stacked one above the other. The header for the leftmost item is on top. If one of the items is an analysis variable, a computed variable, a group variable, or a statistic, then its values fill the cells in that part of the report. Otherwise, PROC REPORT fills the cells with frequency counts.

If you stack a statistic with an analysis variable, then the statistic that you name in the column statement overrides the statistic in the definition of the analysis variable. For example, the following PROC REPORT step produces a report that contains the minimum value of Sales for each sector:

```
proc report data=grocery;
  column sector sales,min;
  define sector/group;
  define sales/analysis sum;
run;
```

If you stack a display variable under an across variable, then all the values of that display variable appear in the report.

Interaction: A series of stacked report items can include only one analysis variable or statistic. If you include more than one analysis variable or statistic, then PROC REPORT returns an error because it cannot determine which values to put in the cells of the report.

Tip: You can use parentheses to group report items whose headers should appear at the same level rather than stacked one above the other.

Featured in: Example 5 on page 1049, Example 6 on page 1053, and Example 10 on page 1064

('header-1' <... 'header-n' > report-item(s))

creates one or more headers that span multiple columns.

header

is a string of characters that spans one or more columns in the report. PROC REPORT prints each header on a separate line. You can use split characters in a header to split one header over multiple lines. See the discussion of SPLIT= on page 972.

In traditional (monospace) SAS output, if the first and last characters of a header are one of the following characters, then PROC REPORT uses that character to expand the header to fill the space over the column or columns:

```
: - = \ _ . * +
```

Similarly, if the first character of a header is < and the last character is >, or vice-versa, then PROC REPORT expands the header to fill the space over the column by repeating the first character before the text of the header and the last character after it.

report-item(s)

specifies the columns to span.

Featured in: Example 10 on page 1064

report-item=name

specifies an alias for a report item. You can use the same report item more than once in a COLUMN statement. However, you can use only one DEFINE statement

for any given name. (The DEFINE statement designates characteristics such as formats and customized column headers. If you omit a DEFINE statement for an item, then the REPORT procedure uses defaults.) Assigning an alias in the COLUMN statement does not by itself alter the report. However, it does enable you to use separate DEFINE statements for each occurrence of a variable or statistic.

Featured in: Example 3 on page 1043

Note: You cannot always use an alias. When you refer in a compute block to a report item that has an alias, you must usually use the alias. However, if the report item shares a column with an across variable, then you must reference the column by column number (see “Four Ways to Reference Report Items in a Compute Block” on page 951). △

COMPUTE Statement

Starts a *compute block*. A compute block contains one or more programming statements that PROC REPORT executes as it builds the report.

Interaction: An ENDCOMP statement must mark the end of the group of statements in the compute block.

Featured in: Example 2 on page 1040, Example 3 on page 1043, Example 4 on page 1047, Example 5 on page 1049, Example 9 on page 1060, and Example 10 on page 1064

```

COMPUTE location <target>
    </ STYLE=<style-element-name>
    <[style-attribute-specification(s)]>>;
    LINE specification(s);
    . . . select SAS language elements . . .
ENDCOMP;

COMPUTE report-item </ type-specification>;
    CALL DEFINE (column-id, 'attribute-name', value);
    . . . select SAS language elements . . .
ENDCOMP;

```

A compute block can be associated with a report item or with a location (at the top or bottom of a report; at the top or bottom of a page; before or after a set of observations). You create a compute block with the COMPUTE window or with the COMPUTE statement. One form of the COMPUTE statement associates the compute block with a report item. Another form associates the compute block with a location.

For a list of the SAS language elements that you can use in compute blocks, see “The Contents of Compute Blocks” on page 950.

Required Arguments

You must specify either a location or a report item in the COMPUTE statement.

location

determines where the compute block executes in relation to *target*.

AFTER

executes the compute block at a break in one of the following places:

- immediately after the last row of a set of rows that have the same value for the variable that you specify as *target* or, if there is a default summary on that variable, immediately after the creation of the preliminary summary line (see “How PROC REPORT Builds a Report” on page 1024).
- except in Printer and RTF output, near the bottom of each page, immediately before any footnotes, if you specify `_PAGE_` as *target*.
- at the end of the report if you omit a target.

BEFORE

executes the compute block at a break in one of the following places:

- immediately before the first row of a set of rows that have the same value for the variable that you specify as *target* or, if there is a default summary on that variable, immediately after the creation of the preliminary summary line (see “How PROC REPORT Builds a Report” on page 1024).
- except in Printer and RTF output, near the top of each page, between any titles and the column headings, if you specify `_PAGE_` as *target*.
- immediately before the first detail row if you omit a target.

Featured in: Example 3 on page 1043 and Example 9 on page 1060

report-item

specifies a data set variable, a computed variable, or a statistic to associate the compute block with. If you are working in the nonwindowing environment, then you must include the report item in the COLUMN statement. If the item is a computed variable, then you must include a DEFINE statement for it.

Featured in: Example 4 on page 1047 and Example 5 on page 1049

Note: The position of a computed variable is important. PROC REPORT assigns values to the columns in a row of a report from left to right. Consequently, you cannot base the calculation of a computed variable on any variable that appears to its right in the report. Δ

Options

STYLE<(location(s))>=<style-element-name><[style-attribute-specification(s)]>

specifies the style to use for the text that is created by any LINE statements in this compute block. See “Using Style Elements in PROC REPORT” on page 953 for details.

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Featured in: Example 16 on page 1083

target

controls when the compute block executes. If you specify a location (BEFORE or AFTER) for the COMPUTE statement, then you can also specify *target*, which can be one of the following:

break-variable

is a group or order variable.

When you specify a break variable, PROC REPORT executes the statements in the compute block each time the value of the break variable changes.

`_PAGE_ </ justification>`

except in Printer and RTF output, causes the compute block to execute once for each page, either immediately after printing any titles or immediately before

printing any footnotes. *justification* controls the placement of text and values. It can be one of the following:

CENTER	centers each line that the compute block writes.
LEFT	left-justifies each line that the compute block writes.
RIGHT	right-justifies each line that the compute block writes.
Default: CENTER	

Featured in: Example 9 on page 1060

type-specification

specifies the type and, optionally, the length of *report-item*. If the report item that is associated with a compute block is a computed variable, then PROC REPORT assumes that it is a numeric variable unless you use a type specification to specify that it is a character variable. A type specification has the form

CHARACTER <LENGTH=*length*>

where

CHARACTER

specifies that the computed variable is a character variable. If you do not specify a length, then the variable's length is 8.

Alias: CHAR

Featured in: Example 10 on page 1064

LENGTH=*length*

specifies the length of a computed character variable.

Default: 8

Range: 1 to 200

Interaction: If you specify a length, then you must use CHARACTER to indicate that the computed variable is a character variable.

Featured in: Example 10 on page 1064

DEFINE Statement

Describes how to use and display a report item.

Tip: If you do not use a DEFINE statement, then PROC REPORT uses default characteristics.

Featured in: Example 2 on page 1040, Example 3 on page 1043, Example 4 on page 1047, Example 5 on page 1049, Example 6 on page 1053, Example 9 on page 1060, and Example 10 on page 1064

DEFINE *report-item* / <*option(s)*>;

To do this	Use this option
Specify how to use a report item (see “Usage of Variables in a Report” on page 945)	
Define the item, which must be a data set variable, as an across variable	ACROSS
Define the item, which must be a data set variable, as an analysis variable	ANALYSIS
Define the item as a computed variable	COMPUTED
Define the item, which must be a data set variable, as a display variable	DISPLAY
Define the item, which must be a data set variable, as a group variable	GROUP
Define the item, which must be a data set variable, as an order variable	ORDER
Specify style attributes for a report item	
Exclude all combinations of the item that are not found in the preloaded range of user-defined formats	EXCLUSIVE
Assign a SAS or user-defined format to the item	FORMAT=
Reference a HELP or CBT entry that contains Help information for the report item	ITEMHELP=
Consider missing values as valid values for the item	MISSING
Order the values of a group, order, or across variable according to the specified order	ORDER=
Specify that all formats are preloaded for the item.	PRELOADFMT
For traditional SAS monospace output, define the number of blank characters to leave between the column being defined and the column immediately to its left	SPACING=
Associate a statistic with an analysis variable	<i>statistic</i>
Specify a numeric variable whose values weight the value of the analysis variable	WEIGHT=
Define the width of the column in which PROC REPORT displays the report item	WIDTH=
Specify options for a report item	
Reverse the order in which PROC REPORT displays rows or values of a group, order, or across variable	DESCENDING
Wrap the value of a character variable in its column	FLOW
Specify that the item that you are defining is an ID variable	ID
Suppress the display of the report item	NOPRINT
Suppress the display of the report item if its values are all zero or missing	NOZERO
Insert a page break just before printing the first column containing values of the report item	PAGE
Control the placement of values and column headers	

To do this	Use this option
Center the formatted values of the report item within the column width and center the column header over the values	CENTER
Left-justify the formatted values of the report item within the column width and left-justify the column headers over the values	LEFT
Right-justify the formatted values of the report item within the column width and right-justify the column headers over the values	RIGHT
Specify the color in the REPORT window of the column header and of the values of the item that you define	COLOR=
Define the column header for the report item	<i>column-header</i>
Specify a style element (for the Output Delivery System) for the report item	STYLE=

Required Arguments

report-item

specifies the name or alias (established in the COLUMN statement) of the data set variable, computed variable, or statistic to define.

Note: Do not specify a usage option in the definition of a statistic. The name of the statistic tells PROC REPORT how to use it. Δ

Options

ACROSS

defines *report-item*, which must be a data set variable, as an across variable. (See “Across Variables” on page 946.)

Featured in: Example 5 on page 1049

ANALYSIS

defines *report-item*, which must be a data set variable, as an analysis variable. (See “Analysis Variables” on page 946.)

By default, PROC REPORT calculates the Sum statistic for an analysis variable. Specify an alternate statistic with the *statistic* option in the DEFINE statement.

Note: Naming a statistic in the DEFINE statement implies the ANALYSIS option, so you never need to specify ANALYSIS. However, specifying ANALYSIS may make your code easier for novice users to understand. Δ

Featured in: Example 2 on page 1040, Example 3 on page 1043, and Example 4 on page 1047

CENTER

centers the formatted values of the report item within the column width and centers the column header over the values. This option has no effect on the CENTER option in the PROC REPORT statement, which centers the report on the page.

COLOR=*color*

specifies the color in the REPORT window of the column header and of the values of the item that you are defining. You can use the following colors:

BLACK	MAGENTA
BLUE	ORANGE
BROWN	PINK
CYAN	RED
GRAY	WHITE
GREEN	YELLOW

Default: The color of **Foreground** in the SASCOLOR window. (For more information, see the online Help for the SASCOLOR window.)

Restriction: This option affects output in the windowing environment only.

Note: Not all operating environments and devices support all colors, and in some operating environments and devices, one color may map to another color. For example, if the DEFINITION window displays the word BROWN in yellow characters, then selecting BROWN results in a yellow item. \triangle

column-header

defines the column header for the report item. Enclose each header in single or double quotation marks. When you specify multiple column headers, PROC REPORT uses a separate line for each one. The split character also splits a column header over multiple lines.

In traditional (monospace) SAS output, if the first and last characters of a heading are one of the following characters, then PROC REPORT uses that character to expand the heading to fill the space over the column:

```
:- = \_ . * +
```

Similarly, if the first character of a header is < and the last character is >, or vice-versa, then PROC REPORT expands the header to fill the space over the column by repeating the first character before the text of the header and the last character after it.

Default:

Item	Header
variable without a label	variable name
variable with a label	variable label
statistic	statistic name

Tip: If you want to use names when labels exist, then submit the following SAS statement before invoking PROC REPORT:

```
options nolabel;
```

Tip: HEADLINE underlines all column headers and the spaces between them. In traditional (monospace) SAS output, you can underline column headers without

underlining the spaces between them, by using the special characters '---' as the last line of each column header instead of using HEADLINE (see Example 4 on page 1047).

See also: SPLIT= on page 972

Featured in: Example 3 on page 1043, Example 4 on page 1047, and Example 5 on page 1049

COMPUTED

defines the specified item as a computed variable. Computed variables are variables that you define for the report. They are not in the input data set, and PROC REPORT does not add them to the input data set.

In the windowing environment, you add a computed variable to a report from the COMPUTED VAR window.

In the nonwindowing environment, you add a computed variable by

- ☐ including the computed variable in the COLUMN statement
- ☐ defining the variable's usage as COMPUTED in the DEFINE statement
- ☐ computing the value of the variable in a compute block associated with the variable.

Featured in: Example 5 on page 1049 and Example 10 on page 1064

DESCENDING

reverses the order in which PROC REPORT displays rows or values of a group, order, or across variable.

Tip: By default, PROC REPORT orders group, order, and across variables by their formatted values. Use the ORDER= option in the DEFINE statement to specify an alternate sort order.

DISPLAY

defines *report-item*, which must be a data set variable, as a display variable. (See "Display Variables" on page 945.)

EXCLUSIVE

excludes from the report and the output data set all combinations of the group variables and the across variables that are not found in the preloaded range of user-defined formats.

Requirement: You must specify the PRELOADFMT option in the DEFINE statement in order to preload the variable formats.

FLOW

wraps the value of a character variable in its column. The FLOW option honors the split character. If the text contains no split character, then PROC REPORT tries to split text at a blank.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Featured in: Example 10 on page 1064

FORMAT=*format*

assigns a SAS or user-defined format to the item. This format applies to *report-item* as PROC REPORT displays it; the format does not alter the format associated with a variable in the data set. For data set variables, PROC REPORT honors the first of these formats that it finds:

- ☐ the format assigned with FORMAT= in the DEFINE statement
- ☐ the format assigned in a FORMAT statement when you invoke PROC REPORT
- ☐ the format associated with the variable in the data set.

If none of these is present, then PROC REPORT uses BEST w . for numeric variables and \$ w . for character variables. The value of w is the default column width. For character variables in the input data set, the default column width is the variable's length. For numeric variables in the input data set and for computed variables (both numeric and character), the default column width is the value specified by COLWIDTH= in the PROC REPORT statement or in the ROPTIONS window.

In the windowing environment, if you are unsure what format to use, then type a question mark (?) in the format field in the DEFINITION window to access the FORMATS window.

Featured in: Example 2 on page 1040 and Example 6 on page 1053

GROUP

defines *report-item*, which must be a data set variable, as a group variable. (See “Group Variables” on page 946.)

Featured in: Example 4 on page 1047, Example 6 on page 1053, and Example 14 on page 1075

ID

specifies that the item that you are defining is an ID variable. An ID variable and all columns to its left appear at the left of every page of a report. ID ensures that you can identify each row of the report when the report contains more columns than will fit on one page.

Featured in: Example 6 on page 1053

ITEMHELP=*entry-name*

references a HELP or CBT entry that contains help information for the report item. Use PROC BUILD in SAS/AF software to create a HELP or CBT entry for a report item. All HELP and CBT entries for a report must be in the same catalog, and you must specify that catalog with the HELP= option in the PROC REPORT statement or from the **User Help** fields in the ROPTIONS window.

Of course, you can access these entries only from a windowing environment. To access a Help entry from the report, select the item and issue the HELP command. PROC REPORT first searches for and displays an entry named *entry-name*.CBT. If no such entry exists, then PROC REPORT searches for *entry-name*.HELP. If neither a CBT nor a HELP entry for the selected item exists, then the opening frame of the Help for PROC REPORT is displayed.

LEFT

left-justifies the formatted values of the report item within the column width and left-justifies the column headers over the values. If the format width is the same as the width of the column, then the LEFT option has no effect on the placement of values.

MISSING

considers missing values as valid values for the report item. Special missing values that represent numeric values (the letters A through Z and the underscore (_)) character) are each considered as a separate value.

Default: If you omit the MISSING option, then PROC REPORT excludes from the report and the output data sets all observations that have a missing value for any group, order, or across variable.

NOPRINT

suppresses the display of the report item. Use this option

- ☐ if you do not want to show the item in the report but you need to use its values to calculate other values that you use in the report
- ☐ to establish the order of rows in the report

- if you do not want to use the item as a column but want to have access to its values in summaries (see Example 9 on page 1060).

Interaction: Even though the columns that you define with NOPRINT do not appear in the report, you must count them when you are referencing columns by number (see “Four Ways to Reference Report Items in a Compute Block” on page 951).

Interaction: SHOWALL in the PROC REPORT statement or the ROPTIONS window overrides all occurrences of NOPRINT.

Featured in: Example 3 on page 1043, Example 9 on page 1060, and Example 12 on page 1070

NOZERO

suppresses the display of the report item if its values are all zero or missing.

Interaction: Even though the columns that you define with NOZERO do not appear in the report, you must count them when you are referencing columns by number (see “Four Ways to Reference Report Items in a Compute Block” on page 951).

Interaction: SHOWALL in the PROC REPORT statement or in the ROPTIONS window overrides all occurrences of NOZERO.

ORDER

defines *report-item*, which must be a data set variable, as an order variable. (See “Order Variables” on page 945.)

Featured in: Example 2 on page 1040

ORDER=DATA|FORMATTED|FREQ|INTERNAL

orders the values of a group, order, or across variable according to the specified order, where

DATA

orders values according to their order in the input data set.

FORMATTED

orders values by their formatted (external) values. If no format has been assigned to a class variable, then the default format, BEST12., is used.

FREQ

orders values by ascending frequency count.

INTERNAL

orders values by their unformatted values, which yields the same order that PROC SORT would yield. This order is operating environment-dependent. This sort sequence is particularly useful for displaying dates chronologically.

Default: FORMATTED

Interaction: DESCENDING in the item’s definition reverses the sort sequence for an item. By default, the order is ascending.

Featured in: Example 2 on page 1040

Note: The default value for the ORDER= option in PROC REPORT is not the same as the default value in other SAS procedures. In other SAS procedures, the default is ORDER=INTERNAL. The default for the option in PROC REPORT may change in a future release to be consistent with other procedures. Therefore, in production jobs where it is important to order report items by their formatted values, specify ORDER=FORMATTED even though it is currently the default. Doing so ensures that PROC REPORT will continue to produce the reports you expect even if the default changes. Δ

PAGE

inserts a page break just before printing the first column containing values of the report item.

Interaction: PAGE is ignored if you use WRAP in the PROC REPORT statement or in the ROPTIONS window.

PRELOADFMT

specifies that the format is preloaded for the variable.

Restriction: PRELOADFMT applies only to group and across variables.

Requirement: PRELOADFMT has no effect unless you specify either EXCLUSIVE or ORDER=DATA and you assign a format to the variable.

Interaction: To limit the report to the combination of formatted variable values that are present in the input data set, use the EXCLUSIVE option in the DEFINE statement.

Interaction To include all ranges and values of the user-defined formats in the output, use the COMPLETEROWS option in the PROC REPORT statement.

Note: If you do not specify NOCOMPLETECOLS when you define the across variables, then the report includes a column for every formatted variable. If you specify COMPLETEROWS when you define the group variables, then the report includes a row for every formatted value. Some combinations of rows and columns might not make sense when the report includes a column for every formatted value of the across variable and a row for every formatted value of the group variable. \triangle

RIGHT

right-justifies the formatted values of the specified item within the column width and right-justifies the column headers over the values. If the format width is the same as the width of the column, then RIGHT has no effect on the placement of values.

SPACING=*horizontal-positions*

defines the number of blank characters to leave between the column being defined and the column immediately to its left. For each column, the sum of its width and the blank characters between it and the column to its left cannot exceed the line size.

Default: 2

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: When PROC REPORT's CENTER option is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

Interaction: SPACING= in an item's definition overrides the value of SPACING= in the PROC REPORT statement or in the ROPTIONS window.

statistic

associates a statistic with an analysis variable. You must associate a statistic with every analysis variable in its definition. PROC REPORT uses the statistic that you specify to calculate values for the analysis variable for the observations that are represented by each cell of the report. You cannot use *statistic* in the definition of any other kind of variable.

See "Statistics That Are Available in PROC REPORT" on page 949 for a list of available statistics.

Default: SUM

Featured in: Example 2 on page 1040, Example 3 on page 1043, and Example 4 on page 1047

Note: PROC REPORT uses the name of the analysis variable as the default header for the column. You can customize the column header with the *column-header* option in the DEFINE statement. \triangle

STYLE<(location(s))>=<style-element-name><[style-attribute-specification(s)]>

specifies the style element to use for column headers and for text inside cells for this report item. See “Using Style Elements in PROC REPORT” on page 953 for details.

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Featured in: Example 16 on page 1083

WEIGHT=*weight-variable*

specifies a numeric variable whose values weight the values of the analysis variable that is specified in the DEFINE statement. The variable value does not have to be an integer. The following table describes how PROC REPORT treats various values of the WEIGHT variable.

Weight Value	PROC REPORT Response
0	counts the observation in the total number of observations
less than 0	converts the value to zero and counts the observation in the total number of observations
missing	excludes the observation

To exclude observations that contain negative and zero weights from the analysis, use the EXCLNPWGT option in the PROC REPORT statement. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default.

Restriction: to compute weighted quantiles, use QMETHOD=OS in the PROC REPORT statement.

Tip: When you use the WEIGHT= option, consider which value of the VARDEF= option in the PROC REPORT statement is appropriate.

Tip: Use the WEIGHT= option in separate variable definitions in order to specify different weights for the variables.

Note: Prior to Version 7 of SAS, the REPORT procedure did not exclude the observations with missing weights from the count of observations. Δ

WIDTH=*column-width*

defines the width of the column in which PROC REPORT displays *report-item*.

Default: A column width that is just large enough to handle the format. If there is no format, then PROC REPORT uses the value of the COLWIDTH= option in the PROC REPORT statement.

Range: 1 to the value of the SAS system option LINESIZE=

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: WIDTH= in an item definition overrides the value of COLWIDTH= in the PROC REPORT statement or the ROPTIONS window.

Tip: When you stack items in the same column in a report, the width of the item that is at the bottom of the stack determines the width of the column.

Featured in: Example 10 on page 1064

ENDCOMP Statement

Marks the end of one or more programming statements that PROC REPORT executes as it builds the report.

Restriction: A COMPUTE statement must precede the ENDCOMP statement.

ENDCOMP;

See also: COMPUTE statement

Featured in: Example 2 on page 1040

FREQ Statement

Treats observations as if they appear multiple times in the input data set.

Tip: The effects of the FREQ and WEIGHT statements are similar except when calculating degrees of freedom.

See also: For an example that uses the FREQ statement, see “Example” on page 57

FREQ *variable*;

Required Arguments

variable

specifies a numeric variable whose value represents the frequency of the observation. If you use the FREQ statement, then the procedure assumes that each observation represents n observations, where n is the value of *variable*. If n is not an integer, then SAS truncates it. If n is less than 1 or is missing, then the procedure does not use that observation to calculate statistics.

Frequency Information Is Not Saved

When you store a report definition, PROC REPORT does not store the FREQ statement.

LINE Statement

Provides a subset of the features of the PUT statement for writing customized summaries.

Restriction: This statement is valid only in a compute block that is associated with a location in the report.

Restriction: You cannot use the LINE statement in conditional statements (IF-THEN, IF-THEN/ELSE, and SELECT) because it is not executed until PROC REPORT has executed all other statements in the compute block.

Featured in: Example 2 on page 1040, Example 3 on page 1043, and Example 9 on page 1060

LINE *specification(s)*;

Required Arguments

specification(s)

can have one of the following forms. You can mix different forms of specifications in one LINE statement.

item item-format

specifies the item to display and the format to use to display it, where

item

is the name of a data set variable, a computed variable, or a statistic in the report. For information about referencing report items see “Four Ways to Reference Report Items in a Compute Block” on page 951.

item-format

is a SAS format or user-defined format. You must specify a format for each item.

Featured in: Example 2 on page 1040

'character-string'

specifies a string of text to display. When the string is a blank and nothing else is in *specification(s)*, PROC REPORT prints a blank line.

Featured in: Example 2 on page 1040

number-of-repetitions'character-string'*

specifies a character string and the number of times to repeat it.

Featured in: Example 3 on page 1043

pointer-control

specifies the column in which PROC REPORT displays the next specification. You can use either of the following forms for pointer controls:

@column-number

specifies the number of the column in which to begin displaying the next item in the specification list.

+column-increment

specifies the number of columns to skip before beginning to display the next item in the specification list.

Both *column-number* and *column-increment* can be either a variable or a literal value.

Restriction: The pointer controls are designed for monospace output. They have no effect on the HTML, RTF, or Printer output.

Featured in: Example 3 on page 1043 and Example 5 on page 1049

Differences between the LINE and PUT Statements

The LINE statement does not support the following features of the PUT statement:

- automatic labeling signaled by an equals sign (=), also known as named output
- the _ALL_, _INFILE_, and _PAGE_ arguments and the OVERPRINT option
- grouping items and formats to apply one format to a list of items
- pointer control using expressions
- line pointer controls (# and /)
- trailing at signs (@ and @@)
- format modifiers
- array elements.

RBREAK Statement

Produces a default summary at the beginning or end of a report or at the beginning or end of each BY group.

Featured in: Example 1 on page 1037 and Example 10 on page 1064

RBREAK *location* *</ option(s)>*;

To do this	Use this option
Specify the color of the break lines in the REPORT window	COLOR=
Double overline each value	DOL*
Double underline each value	DUL*
Overline each value	OL*
Start a new page after the last break line of a break located at the beginning of the report	PAGE
Write a blank line for the last break line of a break located at the beginning of the report	SKIP*
Specify a style element (for the Output Delivery System) for default summary lines, customized summary lines, or both	STYLE=
Include a summary line as one of the break lines	SUMMARIZE
Underline each value	UL*

* Traditional SAS monospace output only.

Required Arguments

location

controls the placement of the break lines and is either of the following:

AFTER

places the break lines at the end of the report.

BEFORE

places the break lines at the beginning of the report.

Options

COLOR=*color*

specifies the color of the break lines in the REPORT window. You can use the following colors:

BLACK	MAGENTA
BLUE	ORANGE
BROWN	PINK
CYAN	RED
GRAY	WHITE
GREEN	YELLOW

Default: The color of **Foreground** in the SASCOLOR window. (For more information, see the online Help for the SASCOLOR window.)

Restriction: This option affects output in the windowing environment only.

Note: Not all operating environments and devices support all colors, and in some operating environments and devices, one color may map to another color. For example, if the DEFINITION window displays the word BROWN in yellow characters, then selecting BROWN results in a yellow item. △

DOL

(for double overlining) uses the thirteenth formatting character to overline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option.

Default: equals sign (=)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the OL and DOL options, then PROC REPORT honors only OL.

See also: the discussion of FORMCHAR= on page 963.

Featured in: Example 1 on page 1037

DUL

(for double underlining) uses the thirteenth formatting character to underline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option.

Default: equals sign (=)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the UL and DUL options, then PROC REPORT honors only UL.

See also: the discussion of FORMCHAR= on page 963.

OL

(for overlining) uses the second formatting character to overline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option.

Default: hyphen (-)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the OL and DOL options, then PROC REPORT honors only OL.

See also: the discussion of FORMCHAR= on page 963.

Featured in: Example 10 on page 1064

PAGE

starts a new page after the last break line of a break located at the beginning of the report.

SKIP

writes a blank line after the last break line of a break located at the beginning of the report.

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

STYLE<(location(s))>=<style-element-name><[style-attribute-specification(s)]>

specifies the style element to use for default summary lines that are created with the RBREAK statement. See “Using Style Elements in PROC REPORT” on page 953 for details.

Restriction: This option affects only the HTML, RTF, and Printer destinations.

SUMMARIZE

includes a summary line as one of the break lines. A summary line at the beginning or end of a report contains values for

- statistics
- analysis variables
- computed variables.

The following table shows how PROC REPORT calculates the value for each kind of report item in a summary line created by the RBREAK statement:

If the report item is...	Then its value is...
a statistic	the value of the statistic over all observations in the set
an analysis variable	the value of the statistic specified as the usage option in the DEFINE statement. PROC REPORT calculates the value of the statistic over all observations in the set. The default usage is SUM.
a computed variable	the results of the calculations based on the code in the corresponding compute block (see “COMPUTE Statement” on page 983).

Featured in: Example 1 on page 1037 and Example 10 on page 1064

UL

(for underlining) uses the second formatting character to underline each value

- that appears in the summary line
- that would appear in the summary line if you specified the SUMMARIZE option.

Default: hyphen (-)

Restriction: This option has no effect on ODS destinations other than traditional SAS monospace output.

Interaction: If you specify both the UL and DUL options, then PROC REPORT honors only UL.

See also: the discussion of FORMCHAR= on page 963.

Order of Break Lines

When a default summary contains more than one break line, the order in which the break lines appear is

- 1 overlining or double overlining (OL or DOL, traditional SAS monospace output only)
- 2 summary line (SUMMARIZE)
- 3 underlining or double underlining (UL or DUL, traditional SAS monospace output only)
- 4 skipped line (SKIP, traditional SAS monospace output only)
- 5 page break (PAGE).

Note: If you define a customized summary for the break, then customized break lines appear after underlining or double underlining. For more information about customized break lines, see “COMPUTE Statement” on page 983 and “LINE Statement” on page 995. △

WEIGHT Statement

Specifies weights for analysis variables in the statistical calculations.

See also: For information about calculating weighted statistics see “Calculating Weighted Statistics” on page 60. For an example that uses the WEIGHT statement, see “Weighted Statistics Example” on page 60.

WEIGHT *variable*;

Required Arguments

variable

specifies a numeric variable whose values weight the values of the analysis variables. The value of the variable does not have to be an integer. If the value of *variable* is

Weight value...	PROC REPORT...
0	counts the observation in the total number of observations
less than 0	converts the value to zero and counts the observation in the total number of observations
missing	excludes the observation

To exclude observations that contain negative and zero weights from the analysis, use EXCLNPWGT. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default.

Tip: When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See VARDEF= on page 972 and the calculation of weighted statistics in “Keywords and Formulas” on page 1578 for more information.

Note: Prior to Version 7 of SAS, the procedure did not exclude the observations with missing weights from the count of observations. \triangle

Weight Information Is Not Saved

When you store a report definition, PROC REPORT does not store the WEIGHT statement.

REPORT Procedure Windows

The windowing environment in PROC REPORT provides essentially the same functionality as the statements, with one major exception: you cannot use the Output Delivery System from the windowing environment.

BREAK

Controls PROC REPORT's actions at a change in the value of a group or order variable or at the top or bottom of a report.

Path

Edit ► Summarize information

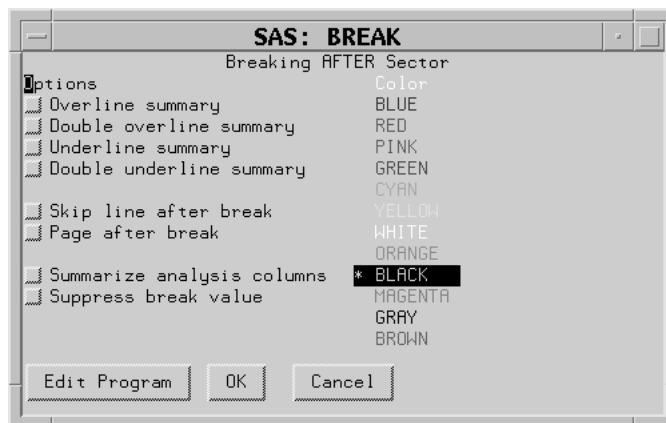
After you select **Summarize Information**, PROC REPORT offers you four choices for the location of the break:

- ☐ **Before Item**
- ☐ **After Item**
- ☐ **At the top**
- ☐ **At the bottom.**

After you select a location, the BREAK window opens.

Note: To create a break before or after detail lines (when the value of a group or order variable changes), you must select a variable before you open the BREAK window. △

Description



Note: For information about changing the formatting characters that are used by the line drawing options in this window, see the discussion of FORMCHAR= on page 963. △

Options

Overline summary

uses the second formatting character to overline each value

- ☐ that appears in the summary line
- ☐ that would appear in the summary line if you specified the **SUMMARIZE** option.

Default: hyphen (-)

Interaction: If you specify options to overline and to double overline, then PROC REPORT overlines.

Double overline summary

uses the thirteenth formatting character to overline each value

- ☐ that appears in the summary line
- ☐ that would appear in the summary line if you specified the **SUMMARIZE** option.

Default: equals sign (=)

Interaction: If you specify options to overline and to double overline, then PROC REPORT overlines.

Underline summary

uses the second formatting character to underline each value

- ☐ that appears in the summary line
- ☐ that would appear in the summary line if you specified the **SUMMARIZE** option.

Default: hyphen (-)

Interaction: If you specify options to underline and to double underline, then PROC REPORT underlines.

Double underline summary

uses the thirteenth formatting character to underline each value

- ☐ that appears in the summary line
- ☐ that would appear in the summary line if you specified the **SUMMARIZE** option.

Default: equals sign (=)

Interaction: If you specify options to underline and to double underline, then PROC REPORT underlines.

Skip line after break

writes a blank line for the last break line.

This option has no effect if you use it in a break at the end of a report.

Page after break

starts a new page after the last break line. This option has no effect in a break at the end of a report.

Interaction: If you use this option in a break on a variable and you create a break at the end of the report, then the summary for the whole report is on a separate page.

Summarize analysis columns

writes a summary line in each group of break lines. A summary line contains values for

- ☐ statistics
- ☐ analysis variables
- ☐ computed variables.

A summary line between sets of observations also contains

- ☐ the break variable (which you can suppress with **Suppress break value**)
- ☐ other group or order variables to the left of the break variable.

The following table shows how PROC REPORT calculates the value for each kind of report item in a summary line created by the **BREAK** window:

If the report item is...	Then its value is...
the break variable	the current value of the variable (or a missing value if you select suppress break value)
a group or order variable to the left of the break variable	the current value of the variable
a group or order variable to the right of the break variable, or a display variable anywhere in the report	missing*
a statistic	the value of the statistic over all observations in the set
an analysis variable	the value of the statistic specified as the usage option in the item's definition. PROC REPORT calculates the value of the statistic over all observations in the set. The default usage is SUM.
a computed variable	the results of the calculations based on the code in the corresponding compute block (see "COMPUTE Statement" on page 983).

*If you reference a variable with a missing value in a customized summary line, then PROC REPORT displays that variable as a blank (for character variables) or a period (for numeric variables).

Suppress break value

suppresses printing of

- ☐ the value of the break variable in the summary line
- ☐ any underlining and overlining in the break lines in the column containing the break variable.

If you select **Suppress break value**, then the value of the break variable is unavailable for use in customized break lines unless you assign it a value in the compute block that is associated with the break.

Color

From the list of colors, select the one to use in the REPORT window for the column header and the values of the item that you are defining.

Default: The color of **Foreground** in the SASCOLOR window. (For more information, see the online Help for the SASCOLOR window.)

Note: Not all operating environments and devices support all colors, and in some operating environments and devices, one color may map to another color. For example, if the DEFINITION window displays the word BROWN in yellow characters, then selecting BROWN results in a yellow item.

Buttons

Edit Program

opens the COMPUTE window and enables you to associate a compute block with a location in the report.

OK

applies the information in the BREAK window to the report and closes the window.

Cancel

closes the BREAK window without applying information to the report.

COMPUTE

Attaches a compute block to a report item or to a location in the report. Use the SAS Text Editor commands to manipulate text in this window.

Path

From **Edit Program** in the COMPUTED VAR, DEFINITION, or BREAK window.

Description

For information about the SAS language features that you can use in the COMPUTE window, see “The Contents of Compute Blocks” on page 950.

COMPUTED VAR

Adds a variable that is not in the input data set to the report.

Path

Select a column. Then select


Edit ► **Add Item** ► **Computed Column**

After you select **Computed Column**, PROC REPORT prompts you for the location of the computed column relative to the column that you have selected. After you select a location, the COMPUTED VAR window opens.

Description

Enter the name of the variable at the prompt. If it is a character variable, then select the **Character data** check box and, if you want, enter a value in the **Length** field. The length can be any integer between 1 and 200. If you leave the field blank, then PROC REPORT assigns a length of 8 to the variable.

After you enter the name of the variable, select **Edit Program** to open the COMPUTE window. Use programming statements in the COMPUTE window to define the computed variable. After closing the COMPUTE and COMPUTED VAR windows, open the DEFINITION window to describe how to display the computed variable.

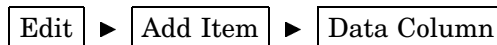
Note: The position of a computed variable is important. PROC REPORT assigns values to the columns in a row of a report from left to right. Consequently, you cannot base the calculation of a computed variable on any variable that appears to its right in the report. 

DATA COLUMNS

Lists all variables in the input data set so that you can add one or more data set variables to the report.

Path

Select a report item. Then select



After you select **Data column**, PROC REPORT prompts you for the location of the computed column relative to the column that you have selected. After you select a location, the DATA COLUMNS window opens.

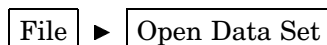
Description

Select one or more variables to add to the report. When you select the first variable, it moves to the top of the list in the window. If you select multiple variables, then subsequent selections move to the bottom of the list of selected variables. An asterisk (*) identifies each selected variable. The order of selected variables from top to bottom determines their order in the report from left to right.

DATA SELECTION

Loads a data set into the current report definition.

Path



Description

The first list box in the DATA SELECTION window lists all the librefs defined for your SAS session. The second one lists all the SAS data sets in the selected library.

Note: You must use data that is compatible with the current report definition. The data set that you load must contain variables whose names are the same as the variable names in the current report definition. △

Buttons

OK

loads the selected data set into the current report definition.

Cancel

closes the DATA SELECTION window without loading new data.

DEFINITION

Displays the characteristics associated with an item in the report and lets you change them.

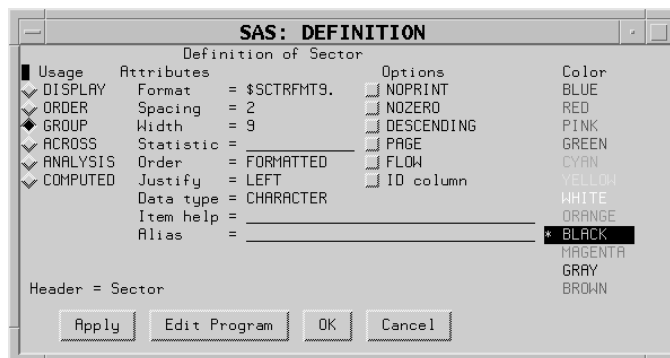
Path

Select a report item. Then select



Note: Alternatively, double-click on the selected item. (Not all operating environments support this method of opening the DEFINITION window.) △

Description



Usage

For an explanation of each type of usage see “Laying Out a Report” on page 944.

DISPLAY

defines the selected item as a display variable. DISPLAY is the default for character variables.

ORDER

defines the selected item as an order variable.

GROUP

defines the selected item as a group variable.

ACROSS

defines the selected item as an across variable.

ANALYSIS

defines the selected item as an analysis variable. You must specify a statistic (see the discussion of the `Statistic=` attribute on page 1008) for an analysis variable.

ANALYSIS is the default for numeric variables.

COMPUTED

defines the selected item as a computed variable. Computed variables are variables that you define for the report. They are not in the input data set, and PROC REPORT does not add them to the input data set. However, computed variables are included in an output data set if you create one.

In the windowing environment, you add a computed variable to a report from the COMPUTED VAR window.

Attributes

Format=

assigns a SAS or user-defined format to the item. This format applies to the selected item as PROC REPORT displays it; the format does not alter the format associated with a variable in the data set. For data set variables, PROC REPORT honors the first of these formats that it finds:

- ☐ the format assigned with `FORMAT=` in the DEFINITION window
- ☐ the format assigned in a `FORMAT` statement when you start PROC REPORT
- ☐ the format associated with the variable in the data set.

If none of these is present, then PROC REPORT uses `BEST w` for numeric variables and `$ w` for character variables. The value of w is the default column width. For character variables in the input data set, the default column width is the variable's length. For numeric variables in the input data set and for computed variables (both numeric and character), the default column width is the value of the `COLWIDTH=` attribute in the ROPTIONS window.

If you are unsure what format to use, then type a question mark (?) in the format field in the DEFINITION window to access the FORMATS window.

Spacing=

defines the number of blank characters to leave between the column being defined and the column immediately to its left. For each column, the sum of its width and the blank characters between it and the column to its left cannot exceed the line size.

Default: 2

Interaction: When PROC REPORT's `CENTER` option is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

Interaction: `SPACING=` in an item definition overrides the value of `SPACING=` in the PROC REPORT statement or the ROPTIONS window.

Width=

defines the width of the column in which PROC REPORT displays the selected item.

Range: 1 to the value of the SAS system option `LINESIZE=`

Default: A column width that is just large enough to handle the format. If there is no format, then PROC REPORT uses the value of `COLWIDTH=`.

Note: When you stack items in the same column in a report, the width of the item that is at the bottom of the stack determines the width of the column. Δ

Statistic=

associates a statistic with an analysis variable. You must associate a statistic with every analysis variable in its definition. PROC REPORT uses the statistic that you specify to calculate values for the analysis variable for the observations represented by each cell of the report. You cannot use *statistic* in the definition of any other kind of variable.

Default: SUM

Note: PROC REPORT uses the name of the analysis variable as the default header for the column. You can customize the column header with the **Header** field of the DEFINITION window. \triangle

You can use the following values for *statistic*:

Descriptive statistic keywords

CSS	PCTSUM
CV	RANGE
MAX	STDDEV STD
MEAN	STDERR
MIN	SUM
N	SUMWGT
NMISS	USS
PCTN	VAR

Quantile statistic keywords

MEDIAN Q2 P50	Q3 P75
P1	P90
P5	P95
P10	P99
Q1 P25	QRANGE

Hypothesis testing keyword

PROBT	T
-------	---

Explanations of the keywords, the formulas that are used to calculate them, and the data requirements are discussed in Appendix 1, “SAS Elementary Statistics Procedures,” on page 1577.

Requirement: To compute standard error and the Student’s *t*-test you must use the default value of VARDEF= which is DF.

See also: For definitions of these statistics, see “Keywords and Formulas” on page 1578.

Order=

orders the values of a GROUP, ORDER, or ACROSS variable according to the specified order, where

DATA

orders values according to their order in the input data set.

FORMATTED

orders values by their formatted (external) values. By default, the order is ascending.

FREQ

orders values by ascending frequency count.

INTERNAL

orders values by their unformatted values, which yields the same order that PROC SORT would yield. This order is operating environment-dependent. This sort sequence is particularly useful for displaying dates chronologically.

Default: FORMATTED

Interaction: DESCENDING in the item's definition reverses the sort sequence for an item.

Note: The default value for the ORDER= option in PROC REPORT is not the same as the default value in other SAS procedures. In other SAS procedures, the default is ORDER=INTERNAL. The default for the option in PROC REPORT may change in a future release to be consistent with other procedures. Therefore, in production jobs where it is important to order report items by their formatted values, specify ORDER=FORMATTED even though it is currently the default. Doing so ensures that PROC REPORT will continue to produce the reports you expect even if the default changes. Δ

Justify=

You can justify the placement of the column header and of the values of the item that you are defining within a column in one of three ways:

LEFT

left-justifies the formatted values of the item that you are defining within the column width and left-justifies the column header over the values. If the format width is the same as the width of the column, then LEFT has no effect on the placement of values.

RIGHT

right-justifies the formatted values of the item that you are defining within the column width and right-justifies the column header over the values. If the format width is the same as the width of the column, then RIGHT has no effect on the placement of values.

CENTER

centers the formatted values of the item that you are defining within the column width and centers the column header over the values. This option has no effect on the setting of the SAS system option CENTER.

When justifying values, PROC REPORT justifies the field width defined by the format of the item within the column. Thus, numbers are always aligned.

Data type=

shows you if the report item is numeric or character. You cannot change this field.

Item Help=

references a HELP or CBT entry that contains help information for the selected item. Use PROC BUILD in SAS/AF software to create a HELP or CBT entry for a report item. All HELP and CBT entries for a report must be in the same catalog, and you must specify that catalog with the HELP= option in the PROC REPORT statement or from the **User Help** fields in the ROPTIONS window.

To access a help entry from the report, select the item and issue the HELP command. PROC REPORT first searches for and displays an entry named *entry-name.CBT*. If no such entry exists, then PROC REPORT searches for

entry-name.HELP. If neither a CBT nor a HELP entry for the selected item exists, then the opening frame of the help for PROC REPORT is displayed.

Alias=

By entering a name in the **Alias** field, you create an alias for the report item that you are defining. Aliases let you distinguish between different uses of the same report item. When you refer in a compute block to a report item that has an alias, you must use the alias (see Example 3 on page 1043).

Options

NOPRINT

suppresses the display of the item that you are defining. Use this option

- if you do not want to show the item in the report but you need to use the values in it to calculate other values that you use in the report
- to establish the order of rows in the report
- if you do not want to use the item as a column but want to have access to its values in summaries (see Example 9 on page 1060).

Interaction: Even though the columns that you define with NOPRINT do not appear in the report, you must count them when you are referencing columns by number (see “Four Ways to Reference Report Items in a Compute Block” on page 951).

Interaction: SHOWALL in the PROC REPORT statement or the ROPTIONS window overrides all occurrences of NOPRINT.

NOZERO

suppresses the display of the item that you are defining if its values are all zero or missing.

Interaction: Even though the columns that you define with NOZERO do not appear in the report, you must count them when you are referencing columns by number (see “Four Ways to Reference Report Items in a Compute Block” on page 951).

Interaction: SHOWALL in the PROC REPORT statement or the ROPTIONS window overrides all occurrences of NOZERO.

DESCENDING

reverses the order in which PROC REPORT displays rows or values of a group, order, or across variable.

PAGE

inserts a page break just before printing the first column containing values of the selected item.

Interaction: PAGE is ignored if you use WRAP in the PROC REPORT statement or in the ROPTIONS window.

FLOW

wraps the value of a character variable in its column. The FLOW option honors the split character. If the text contains no split character, then PROC REPORT tries to split text at a blank.

ID column

specifies that the item that you are defining is an ID variable. An ID variable and all columns to its left appear at the left of every page of a report. ID ensures that you can identify each row of the report when the report contains more columns than will fit on one page.

Color

From the list of colors, select the one to use in the REPORT window for the column header and the values of the item that you are defining.

Default: The color of **Foreground** in the SASCOLOR window. (For more information, see the online Help for the SASCOLOR window.)

Note: Not all operating environments and devices support all colors, and in some operating environments and devices, one color may map to another color. For example, if the DEFINITION window displays the word BROWN in yellow characters, then selecting BROWN results in a yellow item.

Buttons

Apply

applies the information in the open window to the report and keeps the window open.

Edit Program

opens the COMPUTE window and enables you to associate a compute block with the variable that you are defining.

OK

applies the information in the DEFINITION window to the report and closes the window.

Cancel

closes the DEFINITION window without applying changes made with **APPLY**.

DISPLAY PAGE

Displays a particular page of the report.

Path

View ► **Display Page**

Description

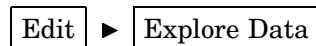
You can get to the last page of the report by entering a large number for the page number. When you are on the last page of the report, PROC REPORT sends a note to the message line of the REPORT window.

EXPLORE

Lets you experiment with your data.

Restriction: You cannot open the EXPLORE window unless your report contains at least one group or order variable.

Path



Description

In the EXPLORE window you can

- ☐ subset the data with list boxes
- ☐ suppress the display of a column with the **Remove Column** check box
- ☐ change the order of the columns with [Rotate columns](#).

Note: The results of your manipulations in the EXPLORE window appear in the REPORT window but are not saved in report definitions. △

Window Features

list boxes

The EXPLORE window contains three list boxes. These boxes contain the value **All levels** as well as actual values for the first three group or order variables in your report. The values reflect any WHERE clause processing that is in effect. For example, if you use a WHERE clause to subset the data so that it includes only the northeast and northwest sectors, then the only values that appear in the list box for Sector are **All levels**, **Northeast**, and **Northwest**. Selecting **All levels** in this case displays rows of the report for only the northeast and northwest sectors. To see data for all the sectors, you must clear the WHERE clause before you open the EXPLORE window.

Selecting values in the list boxes restricts the display in the REPORT window to the values that you select. If you select incompatible values, then PROC REPORT returns an error.

Remove Column

Above each list box in the EXPLORE window is a check box labeled **Remove Column**. Selecting this check box and applying the change removes the column from the REPORT window. You can easily restore the column by clearing the check box and applying that change.

Buttons



applies the information in the EXPLORE window to the report and closes the window.

Apply

applies the information in the EXPLORE window to the report and keeps the window open.

Rotate columns

changes the order of the variables displayed in the list boxes. Each variable that can move one column to the left does; the leftmost variable moves to the third column.

Cancel

closes the EXPLORE window without applying changes made with **APPLY**.

FORMATS

Displays a list of formats and provides a sample of each one.

Path

From the DEFINE window, type a question mark (?) in the **Format** field and select any of the Buttons except **Cancel**, or press RETURN.

Description

When you select a format in the FORMATS window, a sample of that format appears in the **Sample:** field. Select the format that you want to use for the variable that you are defining.

Buttons**OK**

writes the format that you have selected into the **Format** field in the DEFINITION window and closes the FORMATS window. To see the format in the report, select **Apply** in the DEFINITION window.

Cancel

closes the FORMATS window without writing a format into the **Format** field.

LOAD REPORT

Loads a stored report definition.

Path

File ► Open Report

Description

The first list box in the LOAD REPORT window lists all the librefs that are defined for your SAS session. The second list box lists all the catalogs that are in the selected

library. The third list box lists descriptions of all the stored report definitions (entry types of REPT) that are in the selected catalog. If there is no description for an entry, then the list box contains the entry's name.

Buttons

OK

loads the current data into the selected report definition.

Cancel

closes the LOAD REPORT window without loading a new report definition.

Note: Issuing the END command in the REPORT window returns you to the previous report definition (with the current data). △

MESSAGES

Automatically opens to display notes, warnings, and errors returned by PROC REPORT.

You must close the MESSAGES window by selecting **OK** before you can continue to use PROC REPORT.

PROFILE

Customizes some features of the PROC REPORT environment by creating a report profile.

Path

Tools ► Report Profile

Description

The PROFILE window creates a report profile that

- specifies the SAS library, catalog, and entry that define alternative menus to use in the REPORT and COMPUTE windows. Use PROC PMENU to create catalog entries of type PMENU that define these menus. PMENU entries for both windows must be in the same catalog.
- sets defaults for WINDOWS, PROMPT, and COMMAND. PROC REPORT uses the default option whenever you start the procedure unless you specifically override the option in the PROC REPORT statement.

Specify the catalog that contains the profile to use with the PROFILE= option in the PROC REPORT statement (see the discussion of PROFILE= on page 969).

Buttons

OK

stores your profile in a file that is called SASUSER.PROFILE.REPORT.PROFILE.

Note: Use PROC CATALOG or the EXPLORER window to copy the profile to another location. △

Cancel

closes the window without storing the profile.

PROMPTER

Prompts you for information as you add items to a report.

Path

Specify the PROMPT option when you start PROC REPORT or select PROMPT from the ROPTIONS window. The PROMPTER window opens the next time that you add an item to the report.

Description

The prompter guides you through parts of the windows that are most commonly used to build a report. As the content of the PROMPTER window changes, the title of the window changes to the name of the window that you would use to perform a task if you were not using the prompter. The title change is to help you begin to associate the windows with their functions and to learn what window to use if you later decide to change something.

If you start PROC REPORT with prompting, then the first window gives you a chance to limit the number of observations that are used during prompting. When you exit the prompter, PROC REPORT removes the limit.

Buttons

OK

applies the information in the open window to the report and continues the prompting process.

Note: When you select **OK** from the last prompt window, PROC REPORT removes any limit on the number of observations that it is working with. △

Apply

applies the information in the open window to the report and keeps the window open.

Backup

returns you to the previous PROMPTER window.

Exit Prompter

closes the PROMPTER window without applying any more changes to the report. If you have limited the number of observations to use during prompting, then PROC REPORT removes the limit.

REPORT

Is the surface on which the report appears.

Path

Use WINDOWS or PROMPT in the PROC REPORT statement.

Description

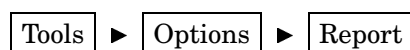
You cannot write directly in any part of the REPORT window except column headers. To change other aspects of the report, you select a report item (for example, a column heading) as the target of the next command and issue the command. To select an item, use a mouse or cursor keys to position the cursor over it. Then click the mouse button or press ENTER. To execute a command, make a selection from the menu bar at the top of the REPORT window. PROC REPORT displays the effect of a command immediately unless the DEFER option is on.

Note: Issuing the END command in the REPORT window returns you to the previous report definition with the current data. If there is no previous report definition, then END closes the REPORT window. △

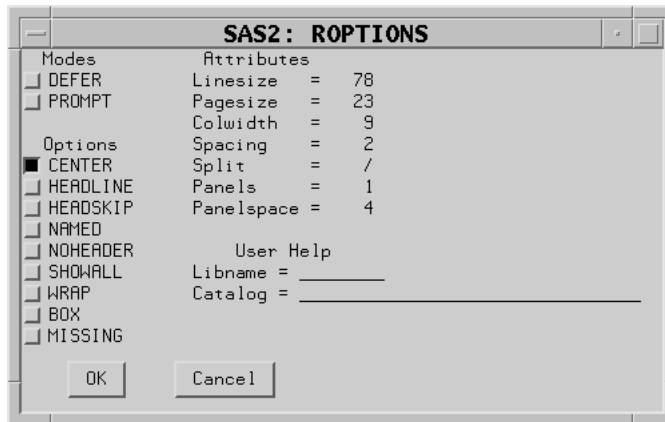
ROPTIONS

Displays choices that control the layout and display of the entire report and identifies the SAS data library and catalog containing CBT or HELP entries for items in the report.

Path



Description



Modes

DEFER

stores the information for changes and makes the changes all at once when you turn DEFER mode off or select

►

DEFER is particularly useful when you know that you need to make several changes to the report but do not want to see the intermediate reports.

By default, PROC REPORT redisplay the report in the REPORT window each time you redefine the report by adding or deleting an item, by changing information in the DEFINITION window, or by changing information in the BREAK window.

PROMPT

opens the PROMPTER window the next time that you add an item to the report.

Options

CENTER

centers the report and summary text (customized break lines). If CENTER is not selected, then the report is left-justified.

PROC REPORT honors the first of these centering specifications that it finds:

- ☐ the CENTER or NOCENTER option in the PROC REPORT statement or the CENTER toggle in the ROPTIONS window
- ☐ the CENTER or NOCENTER option stored in the report definition loaded with REPORT= in the PROC REPORT statement
- ☐ the SAS system option CENTER or NOCENTER.

When PROC REPORT's CENTER option is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

HEADLINE

underlines all column headers and the spaces between them at the top of each page of the report.

HEADLINE underlines with the second formatting character. (See the discussion of FORMCHAR= on page 963.)

Default: hyphen (-)

Tip: In traditional (monospace) SAS output, you can underline column headers without underlining the spaces between them, by using ‘--’ as the last line of each column header instead of using HEADLINE.

HEADSKIP

writes a blank line beneath all column headers (or beneath the underlining that the HEADLINE option writes) at the top of each page of the report.

NAMED

writes *name=* in front of each value in the report, where *name* is the column header for the value.

Tip: Use NAMED in conjunction with WRAP to produce a report that wraps all columns for a single row of the report onto consecutive lines rather than placing columns of a wide report on separate pages.

Interaction: When you use NAMED, PROC REPORT automatically uses NOHEADER.

NOHEADER

suppresses column headers, including those that span multiple columns.

Once you suppress the display of column headers in the windowing environment, you cannot select any report items.

SHOWALL

overrides the parts of a definition that suppress the display of a column (NOPRINT and NOZERO). You define a report item with a DEFINE statement or in the DEFINITION window.

WRAP

displays one value from each column of the report, on consecutive lines if necessary, before displaying another value from the first column. By default, PROC REPORT displays values for only as many columns as it can fit on one page. It fills a page with values for these columns before starting to display values for the remaining columns on the next page.

Interaction: When WRAP is in effect, PROC REPORT ignores PAGE in any item definitions.

Tip: Typically, you use WRAP in conjunction with NAMED to avoid wrapping column headers.

BOX

uses formatting characters to add line-drawing characters to the report. These characters

- ☐ surround each page of the report
- ☐ separate column headers from the body of the report
- ☐ separate rows and columns from each other.

Interaction: You cannot use BOX if you use WRAP in the PROC REPORT statement or ROPTIONS window or if you use FLOW in any item’s definition.

See also: For information about formatting characters, see the discussion of FORMCHAR= on page 963.

MISSING

considers missing values as valid values for group, order, or across variables. Special missing values that are used to represent numeric values (the letters A through Z and the underscore (_) character) are each considered as a different value. A group for each missing value appears in the report. If you omit the MISSING option, then PROC REPORT does not include observations with a missing value for one or more group, order, or across variables in the report.

Attributes

Linesize

specifies the line size for a report. PROC REPORT honors the first of these line-size specifications that it finds:

- LS= in the PROC REPORT statement or Linesize= in the ROPTIONS window
- the LS= setting stored in the report definition loaded with REPORT= in the PROC REPORT statement
- the SAS system option LINESIZE=.

Range: 64-256 (integer)

Tip: If the line size is greater than the width of the REPORT window, then use SAS windowing environment commands RIGHT and LEFT to display portions of the report that are not currently in the display.

Pagesize

specifies the page size for a report. PROC REPORT honors the first of these page size specifications that it finds:

- PS= in the PROC REPORT statement or Pagesize= in the ROPTIONS window
- the PS= setting stored in the report definition loaded with REPORT= in the PROC REPORT statement
- the SAS system option PAGESIZE=.

Range: 15-32,767 (integer)

Colwidth

specifies the default number of characters for columns containing computed variables or numeric data set variables.

Range: 1 to the linesize

Default: 9

Interaction: When setting the width for a column, PROC REPORT first looks at WIDTH= in the definition for that column. If WIDTH= is not present, then PROC REPORT uses a column width large enough to accommodate the format for the item. (For information about formats, see the discussion of Format= on page 1007.) If no format is associated with the item, then the column width depends on variable type:

If the variable is a...	Then the column width is the...
character variable in the input data set	length of the variable
numeric variable in the input data set	value of the COLWIDTH= option
computed variable (numeric or character)	value of the COLWIDTH= option

SPACING=*space-between-columns*

specifies the number of blank characters between columns. For each column, the sum of its width and the blank characters between it and the column to its left cannot exceed the line size.

Default: 2

Interaction: PROC REPORT separates all columns in the report by the number of blank characters specified by SPACING= in the PROC REPORT statement or the ROPTIONS window unless you use SPACING= in the definition of a particular item to change the spacing to the left of that item.

Interaction: When CENTER is in effect, PROC REPORT ignores spacing that precedes the leftmost variable in the report.

SPLIT='character'

specifies the split character. PROC REPORT breaks a column header when it reaches that character and continues the header on the next line. The split character itself is not part of the column header although each occurrence of the split character counts toward the 40-character maximum for a label.

Default: slash (/)

Interaction: The FLOW option in the DEFINE statement honors the split character.

Note: If you are typing over a header (rather than entering one from the PROMPTER or DEFINITION window), then you do not see the effect of the split character until you refresh the screen by adding or deleting an item, by changing the contents of a DEFINITION or a BREAK window, or by selecting

View ► Refresh

PANELS=number-of-panels

specifies the number of panels on each page of the report. If the width of a report is less than half of the line size, then you can display the data in multiple sets of columns so that rows that would otherwise appear on multiple pages appear on the same page. Each set of columns is a *panel*. A familiar example of this kind of report is a telephone book, which contains multiple panels of names and telephone numbers on a single page.

When PROC REPORT writes a multipanel report, it fills one panel before beginning the next.

The number of panels that fits on a page depends on the

- ☐ width of the panel
- ☐ space between panels
- ☐ line size.

Default: 1

Tip: If *number-of-panels* is larger than the number of panels that can fit on the page, then PROC REPORT creates as many panels as it can. Let PROC REPORT put your data in the maximum number of panels that can fit on the page by specifying a large number of panels (for example, 99).

See also: For information about specifying the space between panels see the discussion of PSPACE= on page 1020. For information about setting the linesize, see the discussion of Linesize on page 1019).

PSPACE=space-between-panels

specifies the number of blank characters between panels. PROC REPORT separates all panels in the report by the same number of blank characters. For each panel, the sum of its width and the number of blank characters separating it from the panel to its left cannot exceed the line size.

Default: 4

User Help

identifies the library and catalog containing user-defined help for the report. This help can be in CBT or HELP catalog entries. You can write a CBT or HELP entry for each item in the report with the BUILD procedure in SAS/AF software. You must store all such entries for a report in the same catalog.

Specify the entry name for help for a particular report item in the DEFINITION window for that report item or in a DEFINE statement.

SAVE DATA SET

Lets you specify an output data set in which to store the data from the current report.

Path

File ► Save Data Set

Description

To specify an output data set, enter the name of the SAS data library and the name of the data set (called **member** in the window) that you want to create in the Save Data Set window.

Buttons

OK

Creates the output data set and closes the Save Data Set window.

Cancel

Closes the Save Data Set window without creating an output data set.

SAVE DEFINITION

Saves a report definition for subsequent use with the same data set or with a similar data set.

Path

File ► Save Report

Description

The SAVE DEFINITION window prompts you for the complete name of the catalog entry in which to store the definition of the current report and for an optional description of the report. This description shows up in the LOAD REPORT window and helps you to select the appropriate report.

SAS stores the report definition as a catalog entry of type REPT. You can use a report definition to create an identically structured report for any SAS data set that contains variables with the same names as those used in the report definition.

Buttons

OK

Creates the report definition and closes the SAVE DEFINITION window.

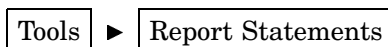
Cancel

Closes the SAVE DEFINITION window without creating a report definition.

SOURCE

Lists the PROC REPORT statements that build the current report.

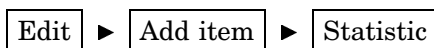
Path



STATISTICS

Displays statistics that are available in PROC REPORT.

Path



After you select **Statistic**, PROC REPORT prompts you for the location of the statistic relative to the column that you have selected. After you select a location, the STATISTICS window opens.

Description

Select the statistics that you want to include in your report and close the window. When you select the first statistic, it moves to the top of the list in the window. If you select multiple statistics, then subsequent selections move to the bottom of the list of selected statistics. An asterisk (*) indicates each selected statistic. The order of selected statistics from top to bottom determines their order in the report from left to right.

Note: If you double-click on a statistic, then PROC REPORT immediately adds it to the report. The STATISTICS window remains open. \triangle

To compute standard error and the Student's t test you must use the default value of VARDEF= which is DF.

To add all selected statistics to the report, select

►

Selecting

►

closes the STATISTICS window without adding the selected statistics to the report.

WHERE

Selects observations from the data set that meet the conditions that you specify.

Path

►

Description

Enter a *where-expression* in the **Enter where clause** field. A *where-expression* is an arithmetic or logical expression that generally consists of a sequence of operands and operators. For information about constructing a *where-expression*, see the documentation of the WHERE statement in the section on statements in *SAS Language Reference: Dictionary*.

Note: You can clear all *where-expressions* by leaving the **Enter where clause** field empty and by selecting . Δ

Buttons

Applies the *where-expression* to the report and closes the WHERE window.

Closes the WHERE window without altering the report.

WHERE ALSO

Selects observations from the data set that meet the conditions that you specify and any other conditions that are already in effect.

Path



Description

Enter a *where-expression* in the **Enter where also clause** field. A *where-expression* is an arithmetic or logical expression that generally consists of a sequence of operands and operators. For information about constructing a *where-expression*, see the documentation of the WHERE statement in the chapter on statements in *SAS Language Reference: Dictionary*.

Buttons

OK

Adds the *where-expression* to any other *where-expressions* that are already in effect and applies them all to the report. It also closes the WHERE ALSO window.

Cancel

Closes the WHERE ALSO window without altering the report.

How PROC REPORT Builds a Report

This section first explains the process of building a report. Following this explanation are illustrations of how PROC REPORT creates two sample reports. The examples use programming statements; you can construct the same reports in the windowing environment.

To understand the process of building a report, you must understand the difference between report variables and DATA step variables. Variables that appear only in one or more compute blocks are *DATA step variables*. Variables that appear in one or more columns of the report are *report variables*. A report variable may or may not appear in a compute block.

Sequence of Events

PROC REPORT constructs a report as follows:

- 1 It consolidates the data by group, order, and across variables. It calculates all statistics for the report, those for detail rows as well as those for summary lines in breaks. Statistics include those computed for analysis variables. PROC REPORT calculates statistics for summary lines whether or not they appear in the report. It stores all this information in a temporary file.
- 2 It initializes all DATA step variables to missing.
- 3 It begins constructing the rows of the report.
 - a At the beginning of each row, it initializes all report variables to missing.
 - b It fills in values for report variables from left to right.
 - Values for computed variables come from executing the statements in the corresponding compute blocks.

- Values for all other variables come from the temporary file created at the beginning of the report-building process.
- Whenever it comes to a break, PROC REPORT first constructs the break lines created with the BREAK or RBREAK statement or with options in the BREAK window. It then executes the statements in the compute block attached to the break (if there is one).

Note: Because of the way PROC REPORT builds a report, you can

- use group statistics in compute blocks for a break before the group variable.
- use statistics for the whole report in a compute block at the beginning of the report.

This document references these statistics with the appropriate compound name. For information about referencing report items in a compute block, see “Four Ways to Reference Report Items in a Compute Block” on page 951. △

Construction of Summary Lines

PROC REPORT constructs a summary line for a break if either of the following conditions is true:

- You summarize numeric variables in the break.
- You use a compute block at the break. (You can attach a compute block to a break without using a BREAK or RBREAK statement or without selecting any options in the BREAK window.)

For more information about using compute blocks, see “Using Compute Blocks” on page 949 and “COMPUTE Statement” on page 983.

The summary line that PROC REPORT constructs at this point is preliminary. If no compute block is attached to the break, then the preliminary summary line becomes the final summary line. However, if a compute block is attached to the break, then the statements in the compute block can alter the values in the preliminary summary line.

PROC REPORT prints the summary line only if you summarize numeric variables in the break.

Using Compound Names

When you use a statistic in a report, you generally refer to it in compute blocks by a compound name like Sales.sum. However, in different parts of the report, that same name has different meanings. Consider the report in Output 38.1 on page 1026. The statements that create the output follow. The user-defined formats that are used are created by a PROC FORMAT step on page 1039.

```
libname proclib 'SAS-data-library';

options nodate pageno=1 linesize=64
       pagesize=60 fmtsearch=(proclib);
proc report data=grocery nowindows;
  column sector manager sales;
  define sector / group format=$sctrfmt.;
  define sales / analysis sum
               format=dollar9.2;
  define manager / group format=$mgrfmt.;
  break after sector / summarize skip 01;
```

```

rbreak after / summarize dol dul;
compute after;
    sector='Total: ';
endcomp;
run;

```

Output 38.1 Three Different Meanings of Sales.sum

The SAS System			1
Sector	Manager	Sales	
Northeast	Alomar	\$786.00	❶
	Andrews	\$1,045.00	
-----		-----	
Northeast		\$1,831.00	❷
Northwest	Brown	\$598.00	
	Pelfrey	\$746.00	
	Reveiz	\$1,110.00	
-----		-----	
Northwest		\$2,454.00	
Southeast	Jones	\$630.00	
	Smith	\$350.00	
-----		-----	
Southeast		\$980.00	
Southwest	Adams	\$695.00	
	Taylor	\$353.00	
-----		-----	
Southwest		\$1,048.00	
=====		=====	
Total:		\$6,313.00	❸
=====		=====	

Here Sales.sum has three different meanings:

- ❶ In detail rows, the value is the sales for one manager's store in a sector of the city. For example, the first detail row of the report shows that the sales for the store that Alomar manages were \$786.00.
- ❷ In the group summary lines, the value is the sales for all the stores in one sector. For example, the first group summary line shows that sales for the Northeast sector were \$1,831.00.
- ❸ In the report summary line, the value (\$6,313.00) is the sales for all stores in the city.

Note: Unless you use the NOALIAS option in the PROC REPORT statement, when you refer in a compute block to a statistic that has an alias, you do not use a compound name. Generally, you must use the alias. However, if the statistic shares a column with an across variable, then you must reference it by column number (see “Four Ways to Reference Report Items in a Compute Block” on page 951). Δ

Building a Report That Uses Groups and a Report Summary

The report in Output 38.2 on page 1027 contains five columns:

- ☐ Sector and Department are group variables.
- ☐ Sales is an analysis variable that is used to calculate the Sum statistic.

- Profit is a computed variable whose value is based on the value of Department.
- The N statistic indicates how many observations each row represents.

At the end of the report a break summarizes the statistics and computed variables in the report and assigns to Sector the value of **TOTALS:**.

The following statements produce Output 38.2 on page 1027. The user-defined formats that are used are created by a PROC FORMAT step on page 1039.

```
libname proclib 'SAS-data-library';

options nodate pageno=1 linesize=64
      pagesize=60 fmtsearch=(proclib);
proc report data=grocery headline headskip;
  column sector department sales Profit N;
  define sector / group format=$sctrfmt.;
  define department / group format=$deptfmt.;
  define sales / analysis sum
      format=dollar9.2;
  define profit / computed format=dollar9.2;

  compute profit;
    if department='np1' or department='np2'
      then profit=0.4*sales.sum;
    else profit=0.25*sales.sum;
  endcomp;

  rbreak after / dol dul summarize;
  compute after;
    sector='TOTALS: ';
  endcomp;

  where sector contains 'n';
  title 'Report for Northeast and Northwest Sectors';
run;
```

Output 38.2 Report with Groups and a Report Summary

Report for Northeast and Northwest Sectors					1
Sector	Department	Sales	Profit	N	

Northeast	Canned	\$840.00	\$336.00	2	
	Meat/Dairy	\$490.00	\$122.50	2	
	Paper	\$290.00	\$116.00	2	
	Produce	\$211.00	\$52.75	2	
Northwest	Canned	\$1,070.00	\$428.00	3	
	Meat/Dairy	\$1,055.00	\$263.75	3	
	Paper	\$150.00	\$60.00	3	
	Produce	\$179.00	\$44.75	3	
=====		=====	=====	=====	
TOTALS:		\$4,285.00	\$1,071.25	20	
=====		=====	=====	=====	

A description of how PROC REPORT builds this report follows:

- 1 PROC REPORT starts building the report by consolidating the data (Sector and Department are group variables) and by calculating the statistics (Sales.sum and N) for each detail row and for the break at the end of the report. It stores these values in a temporary file.
- 2 Now, PROC REPORT is ready to start building the first row of the report. This report does not contain a break at the beginning of the report or a break before any groups, so the first row of the report is a detail row. The procedure initializes all report variables to missing, as Figure 38.9 on page 1028 illustrates. Missing values for a character variable are represented by a blank, and missing values for a numeric variable are represented by a period.

Figure 38.9 First Detail Row with Values Initialized

Sector	Department	Sales	Profit	N
		.	.	.

- 3 Figure 38.10 on page 1028 illustrates the construction of the first three columns of the row. PROC REPORT fills in values for the row from left to right. Values come from the temporary file that is created at the beginning of the report-building process.

Figure 38.10 First Detail Row with Values Filled in from Left to Right

Sector	Department	Sales	Profit	N
Northeast		.	.	.

Sector	Department	Sales	Profit	N
Northeast	Canned	.	.	.

Sector	Department	Sales	Profit	N
Northeast	Canned	\$840.00	.	.

- 4 The next column in the report contains the computed variable Profit. When it gets to this column, PROC REPORT executes the statements in the compute block that is attached to Profit. Nonperishable items (which have a value of **np1** or **np2**) return a profit of 40%; perishable items (which have a value of **p1** or **p2**) return a profit of 25%.

```

if department='np1' or department='np2'
  then profit=0.4*sales.sum;
else profit=0.25*sales.sum;

```

The row now looks like Figure 38.11 on page 1029.

Note: The position of a computed variable is important. PROC REPORT assigns values to the columns in a row of a report from left to right. Consequently, you cannot base the calculation of a computed variable on any variable that appears to its right in the report. Δ

Figure 38.11 A Computed Variable Added to the First Detail Row

Sector	Department	Sales	Profit	N
Northeast	Canned	\$840.00	\$336.00	.

- 5 Next, PROC REPORT fills in the value for the N statistic. The value comes from the temporary file created at the beginning of the report-building process. Figure 38.12 on page 1029 illustrates the completed row.

Figure 38.12 First Complete Detail Row

Sector	Department	Sales	Profit	N
Northeast	Canned	\$840.00	\$336.00	2

- 6 The procedure writes the completed row to the report.
- 7 PROC REPORT repeats steps 2, 3, 4, 5, and 6 for each detail row in the report.
- 8 At the break at the end of the report, PROC REPORT constructs the break lines described by the RBREAK statement. These lines include double underlining, double overlining, and a preliminary version of the summary line. The statistics for the summary line were calculated earlier (see step 1). The value for the computed variable is calculated when PROC REPORT reaches the appropriate column, just as it is in detail rows. PROC REPORT uses these values to create the preliminary version of the summary line (see Figure 38.13 on page 1029).

Figure 38.13 Preliminary Summary Line

Sector	Department	Sales	Profit	N
		\$4,285.00	\$1,071.25	20

- 9 If no compute block is attached to the break, then the preliminary version of the summary line is the same as the final version. However, in this example, a compute block is attached to the break. Therefore, PROC REPORT now executes the statements in that compute block. In this case, the compute block contains one statement:

```
sector='TOTALS:';
```

This statement replaces the value of Sector, which in the summary line is missing by default, with the word **TOTALS:**. After PROC REPORT executes the statement, it modifies the summary line to reflect this change to the value of Sector. The final version of the summary line appears in Figure 38.14 on page 1030.

Figure 38.14 Final Summary Line

Sector	Department	Sales	Profit	N
TOTALS:		\$4,285.00	\$1,071.25	20

- 10 Finally, PROC REPORT writes all the break lines, with underlining, overlining, and the final summary line, to the report.

Building a Report That Uses DATA Step Variables

PROC REPORT initializes report variables to missing at the beginning of each row of the report. The value for a DATA step variable is initialized to missing before PROC REPORT begins to construct the rows of the report, and it remains missing until you specifically assign a value to it. PROC REPORT retains the value of a DATA step variable from the execution of one compute block to another.

Because all compute blocks share the current values of all variables, you can initialize DATA step variables at a break at the beginning of the report or at a break before a break variable. This report initializes the DATA step variable Sctrtot at a break before Sector.

Note: PROC REPORT creates a preliminary summary line for a break before it executes the corresponding compute block. If the summary line contains computed variables, then the computations are based on the values of the contributing variables in the preliminary summary line. If you want to recalculate computed variables based on values that you set in the compute block, then you must do so explicitly in the compute block. This report illustrates this technique.

If no compute block is attached to a break, then the preliminary summary line becomes the final summary line. Δ

The report in Output 38.3 on page 1033 contains five columns:

- ☐ Sector and Department are group variables.
- ☐ Sales is an analysis variable that is used twice in this report: once to calculate the Sum statistic, and once to calculate the Pctsum statistic.
- ☐ Sctrpct is a computed variable whose values are based on the values of Sales and a DATA step variable, Sctrtot, which is the total sales for a sector.

At the beginning of the report, a customized report summary tells what the sales for all stores are. At a break before each group of observations for a department, a default summary summarizes the data for that sector. At the end of each group a break inserts a blank line.

The following statements produce Output 38.3 on page 1033. The user-defined formats that are used are created by a PROC FORMAT step on page 1039.

Note: Calculations of the percentages do not multiply their results by 100 because PROC REPORT prints them with the PERCENT. format. Δ

```
libname proclib 'SAS-data-library';

options nodate pageno=1 linesize=64
       pagesize=60 fmtsearch=(proclib);
proc report data=grocery noheader nowindows;
  column sector department sales
         Sctrpct sales=Salespct;

  define sector      / 'Sector' group
                    format=$sctrfmt.;
  define department / group format=$deptfmt.;
  define sales       / analysis sum
                    format=dollar9.2 ;
  define sctrpct     / computed
                    format=percent9.2 ;
  define salespct    / pctsum format=percent9.2;

  compute before;
    line ' ';
    line @16 'Total for all stores is '
          sales.sum dollar9.2;
    line ' ';
    line @29 'Sum of' @40 'Percent'
          @51 'Percent of';
    line @6 'Sector' @17 'Department'
          @29 'Sales'
          @40 'of Sector' @51 'All Stores';
    line @6 55* '=';
    line ' ';
  endcomp;

  break before sector / summarize ul;
  compute before sector;
    sctrtot=sales.sum;
    sctrpct=sales.sum/sctrtot;
  endcomp;

  compute sctrpct;
    sctrpct=sales.sum/sctrtot;
  endcomp;

  break after sector/skip;
```

```
    where sector contains 'n';  
    title 'Report for Northeast and Northwest Sectors';  
run;
```

Output 38.3 Report with DATA Step Variables

Report for Northeast and Northwest Sectors					1
Total for all stores is \$4,285.00					
Sector	Department	Sum of Sales	Percent of Sector	Percent of All Stores	
=====					
Northeast		\$1,831.00	100.00%	42.73%	

Northeast	Canned	\$840.00	45.88%	19.60%	
	Meat/Dairy	\$490.00	26.76%	11.44%	
	Paper	\$290.00	15.84%	6.77%	
	Produce	\$211.00	11.52%	4.92%	

Northwest		\$2,454.00	100.00%	57.27%	

Northwest	Canned	\$1,070.00	43.60%	24.97%	
	Meat/Dairy	\$1,055.00	42.99%	24.62%	
	Paper	\$150.00	6.11%	3.50%	
	Produce	\$179.00	7.29%	4.18%	

A description of how PROC REPORT builds this report follows:

- 1 PROC REPORT starts building the report by consolidating the data (Sector and Department are group variables) and by calculating the statistics (Sales.sum and Sales.pctsum) for each detail row, for the break at the beginning of the report, for the breaks before each group, and for the breaks after each group. It stores these values in a temporary file.
- 2 PROC REPORT initializes the DATA step variable, Sctrtot, to missing (see Figure 38.15 on page 1033).

Figure 38.15 Initialized DATA Step Variables

Report Variables					DATA Step Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
	

- 3 Because this PROC REPORT step contains a COMPUTE BEFORE statement, the procedure constructs a preliminary summary line for the break at the beginning of the report. This preliminary summary line contains values for the statistics (Sales.sum and Sales.pctsum) and the computed variable (Sctrpct).

At this break, Sales.sum is the sales for all stores, and Sales.pctsum is the percentage those sales represent for all stores (100%). PROC REPORT takes the values for these statistics from the temporary file that it created at the beginning of the report-building process.

The value for Sctrpct comes from executing the statements in the corresponding compute block. Because the value of Sctrtot is missing, PROC REPORT cannot calculate a value for Sctrpct. Therefore, in the preliminary summary line (which is

not printed in this case), this variable also has a missing value (see Figure 38.16 on page 1034).

The statements in the COMPUTE BEFORE block do not alter any variables. Therefore, the final summary line is the same as the preliminary summary line.

Note: The COMPUTE BEFORE statement creates a break at the beginning of the report. You do not need to use an RBREAK statement. \triangle

Figure 38.16 Preliminary and Final Summary Line for the Break at the Beginning of the Report

Report Variables					DATA Step Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
		\$4,285.00	.	100.00%	.

- 4 Because the program does not include an RBREAK statement with the SUMMARIZE option, PROC REPORT does not write the final summary line to the report. Instead, it uses LINE statements to write a customized summary that embeds the value of Sales.sum into a sentence and to write customized column headers. (The NOHEADER option in the PROC REPORT statement suppresses the default column headers, which would have appeared before the customized summary.)
- 5 Next, PROC REPORT constructs a preliminary summary line for the break before the first group of observations. (This break both uses the SUMMARIZE option in the BREAK statement and has a compute block attached to it. Either of these conditions generates a summary line.) The preliminary summary line contains values for the break variable (Sector), the statistics (Sales.sum and Sales.pctsum), and the computed variable (Sctrpct). At this break, Sales.sum is the sales for one sector (the northeast sector). PROC REPORT takes the values for Sector, Sales.sum, and Sales.pctsum from the temporary file that it created at the beginning of the report-building process.

The value for Sctrpct comes from executing the statements in the corresponding compute blocks. Because the value of Sctrtot is still missing, PROC REPORT cannot calculate a value for Sctrpct. Therefore, in the preliminary summary line, Sctrpct has a missing value (see Figure 38.17 on page 1034).

Figure 38.17 Preliminary Summary Line for the Break before the First Group of Observations

Report Variables					DATA Step Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast		\$1,831.00	.	42.73%	.

- 6 PROC REPORT creates the final version of the summary line by executing the statements in the COMPUTE BEFORE SECTOR compute block. These statements execute once each time the value of Sector changes.
 - ☐ The first statement assigns the value of Sales.sum, which in that part of the report represents total sales for one Sector, to the variable Sctrtot.

- The second statement completes the summary line by recalculating Sctrpct from the new value of Sctrtot. Figure 38.18 on page 1035 shows the final summary line.

Note: In this example, you must recalculate the value for Sctrpct in the final summary line. If you do not recalculate the value for Sctrpct, then it will be missing because the value of Sctrtot is missing at the time that the COMPUTE Sctrpct block executes. △

Figure 38.18 Final Summary Line for the Break before the First Group of Observations

Report Variables					DATA Step Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast		\$1,831.00	100.00%	42.73%	\$1,831.00

- 7 Because the program contains a BREAK BEFORE statement with the SUMMARIZE option, PROC REPORT writes the final summary line to the report. The UL option in the BREAK statement underlines the summary line.
- 8 Now, PROC REPORT is ready to start building the first detail row of the report. It initializes all report variables to missing. Values for DATA step variables do not change. Figure 38.19 on page 1035 illustrates the first detail row at this point.

Figure 38.19 First Detail Row with Initialized Values

Report Variables					DATA Step Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
		.	.	.	\$1,831.00

- 9 Figure 38.20 on page 1035 illustrates the construction of the first three columns of the row. PROC REPORT fills in values for the row from left to right. The values come from the temporary file that it created at the beginning of the report-building process.

Figure 38.20 Filling in Values from Left to Right

Report Variables					DATA Step Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast		.	.	.	\$1,831.00

Report Variables					DATA Step Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast	Canned	.	.	.	\$1,831.00

Report Variables					DATA Step Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast	Canned	\$840.00	.	.	\$1,831.00

- 10 The next column in the report contains the computed variable `Sctrpct`. When it gets to this column, PROC REPORT executes the statement in the compute block attached to `Sctrpct`. This statement calculates the percentage of the sector's total sales that this department accounts for:

```
sctrpct=sales.sum/sctrtot;
```

The row now looks like Figure 38.21 on page 1036.

Figure 38.21 First Detail Row with the First Computed Variable Added

Report Variables					DATA Step Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast	Canned	\$840.00	45.88%	.	\$1,831.00

- 11 The next column in the report contains the statistic `Sales.pctsum`. PROC REPORT gets this value from the temporary file. The first detail row is now complete (see Figure 38.22 on page 1036).

Figure 38.22 First Complete Detail Row

Report Variables					DATA Step Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northeast	Canned	\$840.00	45.88%	19.60%	\$1,831.00

- 12 PROC REPORT writes the detail row to the report. It repeats steps 8, 9, 10, 11, and 12 for each detail row in the group.
- 13 After writing the last detail row in the group to the report, PROC REPORT constructs the default group summary. Because no compute block is attached to this break and because the `BREAK AFTER` statement does not include the `SUMMARIZE` option, PROC REPORT does not construct a summary line. The only action at this break is that the `SKIP` option in the `BREAK AFTER` statement writes a blank line after the last detail row of the group.
- 14 Now the value of the break variable changes from **Northeast** to **Northwest**. PROC REPORT constructs a preliminary summary line for the break before this group of observations. As at the beginning of any row, PROC REPORT initializes all report variables to missing but retains the value of the DATA step variable. Next, it completes the preliminary summary line with the appropriate values for the break variable (Sector), the statistics (`Sales.sum` and `Sales.pctsum`), and the computed variable (`Sctrpct`). At this break, `Sales.sum` is the sales for the Northwest sector. Because the `COMPUTE BEFORE Sector` block has not yet executed, the value of `Sctrtot` is still \$1,831.00, the value for the Northeast sector. Thus, the value that PROC REPORT calculates for `Sctrpct` in this preliminary summary line is incorrect (see Figure 38.23 on page 1037). The statements in the compute block for this break calculate the correct value (see the following step).

Figure 38.23 Preliminary Summary Line for the Break before the Second Group of Observations

Report Variables					DATA Step Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northwest		\$2,454.00	134.00%	57.27%	\$1,831.00

CAUTION:

Synchronize values for computed variables in break lines to prevent incorrect results.

If the PROC REPORT step does not recalculate Sctrpct in the compute block that is attached to the break, then the value in the final summary line will not be synchronized with the other values in the summary line, and the report will be incorrect. △

15 PROC REPORT creates the final version of the summary line by executing the statements in the COMPUTE BEFORE Sector compute block. These statements execute once each time the value of Sector changes.

- The first statement assigns the value of Sales.sum, which in that part of the report represents sales for the Northwest sector, to the variable Sctrtot.
- The second statement completes the summary line by recalculating Sctrpct from the new, appropriate value of Sctrtot. Figure 38.24 on page 1037 shows the final summary line.

Figure 38.24 Final Summary Line for the Break before the Second Group of Observations

Report Variables					DATA Step Variable
Sector	Department	Sales.sum	Sctrpct	Sales.pctsum	Sctrtot
Northwest		\$2,454.00	100.00%	57.27%	\$2,454.00

Because the program contains a BREAK BEFORE statement with the SUMMARIZE option, PROC REPORT writes the final summary line to the report. The UL option in the BREAK statement underlines the summary line.

16 Now, PROC REPORT is ready to start building the first row for this group of observations. It repeats steps 8 through 16 until it has processed all observations in the input data set (stopping with step 14 for the last group of observations).

Examples: REPORT Procedure

Example 1: Selecting Variables for a Report

Procedure features:

PROC REPORT statement options:

NOWD

COLUMN statement

default variable usage

RBREAK statement options:

DOL

SUMMARIZE

Other features:

FORMAT statement

FORMAT procedure:

LIBRARY=

SAS system options:

FMTSEARCH=

Automatic macro variables:

SYSDATE

This example uses a permanent data set and permanent formats to create a report that contains

- ☐ one row for every observation
- ☐ a default summary for the whole report.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=64 pagesize=60;
```

Create the GROCERY data set. GROCERY contains one day's sales figures for eight stores in the Grocery Mart chain. Each observation contains one day's sales data for one department in one store.

```
data grocery;
  input Sector $ Manager $ Department $ Sales @@;
  datalines;
se 1 np1 50      se 1 p1 100      se 1 np2 120      se 1 p2 80
se 2 np1 40      se 2 p1 300      se 2 np2 220      se 2 p2 70
nw 3 np1 60      nw 3 p1 600      nw 3 np2 420      nw 3 p2 30
nw 4 np1 45      nw 4 p1 250      nw 4 np2 230      nw 4 p2 73
nw 9 np1 45      nw 9 p1 205      nw 9 np2 420      nw 9 p2 76
sw 5 np1 53      sw 5 p1 130      sw 5 np2 120      sw 5 p2 50
```



```

sw 6 np1 40      sw 6 p1 350      sw 6 np2 225      sw 6 p2 80
ne 7 np1 90      ne 7 p1 190      ne 7 np2 420      ne 7 p2 86
ne 8 np1 200     ne 8 p1 300      ne 8 np2 420      ne 8 p2 125
;

```

Create the \$SCTRFMT., \$MGRFMT., and \$DEPTFMT. formats. PROC FORMAT creates permanent formats for Sector, Manager, and Department. The LIBRARY= option specifies a permanent storage location so that the formats are available in subsequent SAS sessions. These formats are used for examples throughout this section.

```

proc format library=proclib;
    value $sctrfmt 'se' = 'Southeast'
                  'ne' = 'Northeast'
                  'nw' = 'Northwest'
                  'sw' = 'Southwest';

    value $mgrfmt  '1' = 'Smith'   '2' = 'Jones'
                  '3' = 'Reveiz'  '4' = 'Brown'
                  '5' = 'Taylor'  '6' = 'Adams'
                  '7' = 'Alomar'  '8' = 'Andrews'
                  '9' = 'Pelfrey';

    value $deptfmt 'np1' = 'Paper'
                  'np2' = 'Canned'
                  'p1'  = 'Meat/Dairy'
                  'p2'  = 'Produce';
run;

```

Specify the format search library. The SAS system option FMTSEARCH= adds the SAS data library PROCLIB to the search path that is used to locate formats.

```
options fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs the REPORT procedure without the REPORT window and sends its output to the open output destination(s).

```
proc report data=grocery nowd;
```

Specify the report columns. The report contains a column for Manager, Department, and Sales. Because there is no DEFINE statement for any of these variables, PROC REPORT uses the character variables (Manager and Department) as display variables and the numeric variable (Sales) as an analysis variable that is used to calculate the sum statistic.

```
    column manager department sales;
```

Produce a report summary. The RBREAK statement produces a default summary at the end of the report. DOL writes a line of equal signs (=) above the summary information. SUMMARIZE sums the value of Sales for all observations in the report.

```
rbreak after / dol summarize;
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the southeast sector.

```
where sector='se';
```

Format the report columns. The FORMAT statement assigns formats to use in the report. You can use the FORMAT statement only with data set variables.

```
format manager $mgrfmt. department $deptfmt.
       sales dollar11.2;
```

Specify the titles. SYSDATE is an automatic macro variable that returns the date when the SAS job or SAS session began. The TITLE2 statement uses double rather than single quotation marks so that the macro variable resolves.

```
title 'Sales for the Southeast Sector';
title2 "for &sysdate";
run;
```

Output

Sales for the Southeast Sector			1
for 04JAN02			
Manager	Department	Sales	
Smith	Paper	\$50.00	
Smith	Meat/Dairy	\$100.00	
Smith	Canned	\$120.00	
Smith	Produce	\$80.00	
Jones	Paper	\$40.00	
Jones	Meat/Dairy	\$300.00	
Jones	Canned	\$220.00	
Jones	Produce	\$70.00	
		=====	
		\$980.00	

Example 2: Ordering the Rows in a Report

Procedure features:

PROC REPORT statement options:

COLWIDTH=

HEADLINE

HEADSKIP

SPACING=

BREAK statement options:

OL
SKIP
SUMMARIZE

COMPUTE statement arguments:

AFTER

DEFINE statement options:

ANALYSIS
FORMAT=
ORDER
ORDER=
SUM

ENDCOMP statement

LINE statement:

with quoted text
with variable values

Data set: GROCERY on page 1038

Formats: \$MGRFMT. and \$DEPTFMT. on page 1039

This example

- ☐ arranges the rows alphabetically by the formatted values of Manager and the internal values of Department (so that sales for the two departments that sell nonperishable goods precede sales for the two departments that sell perishable goods)
- ☐ controls the default column width and the spacing between columns
- ☐ underlines the column headers and writes a blank line beneath the underlining
- ☐ creates a default summary of Sales for each manager
- ☐ creates a customized summary of Sales for the whole report.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
      fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). COLWIDTH=10 sets the default column width to 10 characters. SPACING= puts five blank characters between columns. HEADLINE underlines all column headers and the spaces between them at the top of each page of the report. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes.

```
proc report data=grocery nowd
           colwidth=10
           spacing=5
           headline headskip;
```

Specify the report columns. The report contains a column for Manager, Department, and Sales.

```
column manager department sales;
```

Define the sort order variables. The values of all variables with the ORDER option in the DEFINE statement determine the order of the rows in the report. In this report, PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement) and then by the values of Department.

ORDER= specifies the sort order for a variable. This report arranges the rows according to the formatted values of Manager and the internal values of Department (np1, np2, p1, and p2).

FORMAT= specifies the formats to use in the report.

```
define manager / order order=formatted format=$mgrfmt.;
define department / order order=internal format=$deptfmt.;
```

Define the analysis variable. Sum calculates the sum statistic for all observations that are represented by the current row. In this report each row represents only one observation. Therefore, the Sum statistic is the same as the value of Sales for that observation in the input data set. Using Sales as an analysis variable in this report enables you to summarize the values for each group and at the end of the report.

```
define sales / analysis sum format=dollar7.2;
```

Produce a report summary. This BREAK statement produces a default summary after the last row for each manager. OL writes a row of hyphens above the summary line. SUMMARIZE writes the value of Sales (the only analysis or computed variable) in the summary line. PROC REPORT sums the values of Sales for each manager because Sales is an analysis variable that is used to calculate the Sum statistic. SKIP writes a blank line after the summary line.

```
break after manager / ol
                    summarize
                    skip;
```

Produce a customized summary. This COMPUTE statement begins a compute block that produces a customized summary at the end of the report. The LINE statement places the quoted text and the value of Sales.sum (with the DOLLAR9.2 format) in the summary. An ENDCOMP statement must end the compute block.

```

compute after;
  line 'Total sales for these stores were: '
    sales.sum dollar9.2;
endcomp;

```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the southeast sector.

```
where sector='se';
```

Specify the title.

```

title 'Sales for the Southeast Sector';
run;

```

Output

Sales for the Southeast Sector			1
Manager	Department	Sales	

Jones	Paper	\$40.00	
	Canned	\$220.00	
	Meat/Dairy	\$300.00	
	Produce	\$70.00	
-----		-----	
Jones		\$630.00	
Smith	Paper	\$50.00	
	Canned	\$120.00	
	Meat/Dairy	\$100.00	
	Produce	\$80.00	
-----		-----	
Smith		\$350.00	
Total sales for these stores were:		\$980.00	

Example 3: Using Aliases to Obtain Multiple Statistics for the Same Variable

Procedure features:

COLUMN statement:

with aliases

COMPUTE statement arguments:

AFTER

DEFINE statement options:

ANALYSIS

MAX

MIN

NOPRINT

customizing column headers

LINE statement:

pointer controls

quoted text

repeating a character string

variable values and formats

writing a blank line

Other features:

automatic macro variables:

SYSDATE

Data set: GROCERY on page 1038

Formats: \$MGRFMT. and \$DEPTFMT. on page 1039

The customized summary at the end of this report displays the minimum and maximum values of Sales over all departments for stores in the southeast sector. To determine these values, PROC REPORT needs the MIN and MAX statistic for Sales in every row of the report. However, to keep the report simple, the display of these statistics is suppressed.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
      fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). HEADLINE underlines all column headers and the spaces between them at the top of each page of the report. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes.

```
proc report data=grocery nowd headline headskip;
```

Specify the report columns. The report contains columns for Manager and Department. It also contains three columns for Sales. The column specifications SALES=SALESMIN and SALES=SALESMAX create aliases for Sales. These aliases enable you to use a separate definition of Sales for each of the three columns.

```

column manager department sales
      sales=salesmin
      sales=salesmax;

```

Define the sort order variables. The values of all variables with the ORDER option in the DEFINE statement determine the order of the rows in the report. In this report, PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement) and then by the values of Department. The ORDER= option specifies the sort order for a variable. This report arranges the values of Manager by their formatted values and arranges the values of Department by their internal values (np1, np2, p1, and p2). FORMAT= specifies the formats to use in the report. Text in quotation marks specifies column headings.

```

define manager / order
      order=formatted
      format=$mgrfmt.
      'Manager';
define department / order
      order=internal
      format=$deptfmt.
      'Department';

```

Define the analysis variable. The value of an analysis variable in any row of a report is the value of the statistic that is associated with it (in this case Sum), calculated for all observations that are represented by that row. In a detail report each row represents only one observation. Therefore, the Sum statistic is the same as the value of Sales for that observation in the input data set.

```

define sales / analysis sum format=dollar7.2 'Sales';

```

Define additional analysis variables for use in the summary. These DEFINE statements use aliases from the COLUMN statement to create separate columns for the MIN and MAX statistics for the analysis variable Sales. NOPRINT suppresses the printing of these statistics. Although PROC REPORT does not print these values in columns, it has access to them so that it can print them in the summary.

```

define salesmin / analysis min noprint;
define salesmax / analysis max noprint;

```

Print a horizontal line at the end of the report. This COMPUTE statement begins a compute block that executes at the end of the report. The first LINE statement writes a blank line. The second LINE statement writes 53 hyphens (-), beginning in column 7. Note that the pointer control (@) has no effect on ODS destinations other than traditional SAS monospace output.

```

compute after;
  line ' ';
  line @7 53*'-';

```

Produce a customized summary. The first line of this LINE statement writes the text in quotation marks, beginning in column 7. The second line writes the value of Salesmin with the DOLLAR7.2 format, beginning in the next column. The cursor then moves one column to the right (+1), where PROC REPORT writes the text in quotation marks. Again, the cursor moves one column to the right, and PROC REPORT writes the value of Salesmax with the DOLLAR7.2 format. (Note that the program must reference the variables by their aliases.) The third line writes the text in quotation marks, beginning in the next column. Note that the pointer control (@) is designed for the Listing destination (traditional SAS output). It has no effect on ODS destinations other than traditional SAS monospace output. The ENDCOMP statement ends the compute block.

```

line @7 '| Departmental sales ranged from'
      salesmin dollar7.2 +1 'to' +1 salesmax dollar7.2
      '. |';
line @7 53*'-' ;
endcomp;

```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the southeast sector.

```

where sector='se';

```

Specify the titles. SYSDATE is an automatic macro variable that returns the date when the SAS job or SAS session began. The TITLE2 statement uses double rather than single quotation marks so that the macro variable resolves.

```

title 'Sales for the Southeast Sector';
title2 "for &sysdate";
run;

```

Output

Sales for the Southeast Sector			1
for 04JAN02			
Manager	Department	Sales	

Jones	Paper	\$40.00	
	Canned	\$220.00	
	Meat/Dairy	\$300.00	
	Produce	\$70.00	
Smith	Paper	\$50.00	
	Canned	\$120.00	
	Meat/Dairy	\$100.00	
	Produce	\$80.00	

Departmental sales ranged from \$40.00 to \$300.00.			

Example 4: Consolidating Multiple Observations into One Row of a Report

Procedure features:

BREAK statement options:

OL
SKIP
SUMMARIZE
SUPPRESS

CALL DEFINE statement

Compute block

associated with a data set variable

COMPUTE statement arguments:

AFTER
a data set variable as *report-item*

DEFINE statement options:

ANALYSIS
GROUP
SUM
customizing column headers

LINE statement:

quoted text
variable values

Data set: GROCERY on page 1038

Formats: \$MGRFMT. and \$DEPTFMT. on page 1039

This example creates a summary report that

- ☐ consolidates information for each combination of Sector and Manager into one row of the report
- ☐ contains default summaries of sales for each sector
- ☐ contains a customized summary of sales for all sectors
- ☐ uses one format for sales in detail rows and a different format in summary rows
- ☐ uses customized column headers.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
      fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). HEADLINE underlines all column headings and the spaces between them at the top of each page of the report. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes.

```
proc report data=grocery nowd headline headskip;
```

Specify the report columns. The report contains columns for Sector, Manager, and Sales.

```
column sector manager sales;
```

Define the group and analysis variables. In this report, Sector and Manager are group variables. Sales is an analysis variable that is used to calculate the Sum statistic. Each detail row represents a set of observations that have a unique combination of formatted values for all group variables. The value of Sales in each detail row is the sum of Sales for all observations in the group. FORMAT= specifies the format to use in the report. Text in quotation marks in a DEFINE statement specifies the column heading.

```
define sector / group
      format=$sctrfmt.
      'Sector';
define manager / group
      format=$mgrfmt.
      'Manager';
define sales / analysis sum
      format=comma10.2
      'Sales';
```

Produce a report summary. This BREAK statement produces a default summary after the last row for each sector. OL writes a row of hyphens above the summary line. SUMMARIZE writes the value of Sales in the summary line. PROC REPORT sums the values of Sales for each manager because Sales is an analysis variable used to calculate the Sum statistic. SUPPRESS prevents PROC REPORT from displaying the value of Sector in the summary line. SKIP writes a blank line after the summary line.

```
break after sector / ol
      summarize
      suppress
      skip;
```

Produce a customized summary. This compute block creates a customized summary at the end of the report. The LINE statement writes the quoted text and the value of Sales.sum (with a format of DOLLAR9.2) in the summary. An ENDCOMP statement must end the compute block.

```
compute after;
      line 'Combined sales for the northern sectors were '
```

```

        sales.sum dollar9.2 '.';
    endcomp;

```

Specify a format for the summary rows. In detail rows, PROC REPORT displays the value of Sales with the format that is specified in its definition (COMMA10.2). The compute block specifies an alternate format to use in the current column on summary rows. Summary rows are identified as a value other than a blank for `_BREAK_`.

```

compute sales;
    if _break_ ne ' ' then
        call define(_col_,"format","dollar11.2");
endcomp;

```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the northeast and northwest sectors. The TITLE statement specifies the title.

```

    where sector contains 'n';

```

Specify the title.

```

    title 'Sales Figures for Northern Sectors';
run;

```

Output

Sales Figures for Northern Sectors			1
Sector	Manager	Sales	

Northeast	Alomar	786.00	
	Andrews	1,045.00	

		\$1,831.00	
Northwest	Brown	598.00	
	Pelfrey	746.00	
	Reveiz	1,110.00	

		\$2,454.00	
Combined sales for the northern sectors were \$4,285.00.			

Example 5: Creating a Column for Each Value of a Variable

Procedure features:

PROC REPORT statement options:

SPLIT=

BREAK statement options:

SKIP

COLUMN statement:

stacking variables

COMPUTE statement arguments:

with a computed variable as *report-item*

AFTER

DEFINE statement options:

ACROSS

ANALYSIS

COMPUTED

SUM

LINE statement:

pointer controls

Data set: GROCERY on page 1038

Formats: \$SCTRFMT., \$MGRFMT., and \$DEPTFMT. on page 1039

The report in this example

- ☐ consolidates multiple observations into one row
- ☐ contains a column for each value of Department that is selected for the report (the departments that sell perishable items)
- ☐ contains a variable that is not in the input data set
- ☐ uses customized column headers, some of which contain blank lines
- ☐ double-spaces between detail rows
- ☐ uses pointer controls to control the placement of text and variable values in a customized summary.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
      fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). HEADLINE underlines the column headings. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes. SPLIT= defines the split character as an asterisk (*) because the default split character (/) is part of the name of a department.

```
proc report data=grocery nowd
      headline
      headskip
      split='*';
```

Specify the report columns. Department and Sales are separated by a comma in the COLUMN statement, so they collectively determine the contents of the column that they define. Each item generates a header, but the header for Sales is set to blank in its definition. Because Sales is an analysis variable, its values fill the cells that are created by these two variables.

```
      column sector manager department,sales perish;
```

Define the group variables. In this report, Sector and Manager are group variables. Each detail row of the report consolidates the information for all observations with the same values of the group variables. FORMAT= specifies the formats to use in the report. Text in quotation marks in the DEFINE statements specifies column headings. These statements illustrate two ways to write a blank line in a column header. 'Sector' '' writes a blank line because each quoted string is a line of the column heading. The two adjacent quotation marks write a blank line for the second line of the heading. 'Manager* ' writes a blank line because the split character (*) starts a new line of the heading. That line contains only a blank.

```
      define sector / group format=$sctrfmt. 'Sector' '';
      define manager / group format=$mgrfmt. 'Manager* ';
```

Define the across variable. PROC REPORT creates a column and a column heading for each formatted value of the across variable Department. PROC REPORT orders the columns by these values. PROC REPORT also generates a column heading that spans all these columns. Quoted text in the DEFINE statement for Department customizes this heading. In traditional (monospace) SAS output, PROC REPORT expands the heading with underscores to fill all columns that are created by the across variable.

```
      define department / across format=$deptfmt. '_Department_';
```

Define the analysis variable. Sales is an analysis variable that is used to calculate the sum statistic. In each case, the value of Sales is the sum of Sales for all observations in one department in one group. (In this case, the value represents a single observation.)

```
      define sales / analysis sum format=dollar11.2 ' ';
```

Define the computed variable. The COMPUTED option indicates that PROC REPORT must compute values for Perish. You compute the variable's values in a compute block that is associated with Perish.

```
      define perish / computed format=dollar11.2
      'Perishable*Total';
```

Produce a report summary. This BREAK statement creates a default summary after the last row for each value of Manager. The only option that is in use is SKIP, which writes a blank line. You can use this technique to double-space in many reports that contains a group or order variable.

```
break after manager / skip;
```

Calculate values for the computed variable. This compute block computes the value of Perish from the values for the Meat/Dairy department and the Produce department. Because the variables Sales and Department collectively define these columns, there is no way to identify the values to PROC REPORT by name. Therefore, the assignment statement uses column numbers to unambiguously specify the values to use. Each time PROC REPORT needs a value for Perish, it sums the values in the third and fourth columns of that row of the report.

```
compute perish;
    perish=sum(_c3_, _c4_);
endcomp;
```

Produce a customized summary. This compute block creates a customized summary at the end of the report. The first LINE statement writes 57 hyphens (-) starting in column 4. Subsequent LINE statements write the quoted text in the specified columns and the values of the variables _C3_, _C4_, and _C5_ with the DOLLAR11.2 format. Note that the pointer control (@) is designed for the Listing destination. It has no effect on ODS destinations other than traditional SAS monospace output.

```
compute after;
    line @4 57*'-';
    line @4 '|    Combined sales for meat and dairy : '
        @46 _c3_ dollar11.2 '    |';
    line @4 '|    Combined sales for produce : '
        @46 _c4_ dollar11.2 '    |';
    line @4 '| ' @60 '|';
    line @4 '|    Combined sales for all perishables: '
        @46 _c5_ dollar11.2 '    |';
    line @4 57*'-';
endcomp;
```

Select the observations to process. The WHERE statement selects for the report only the observations for departments p1 and p2 in stores in the northeast or northwest sector.

```
where sector contains 'n'
    and (department='p1' or department='p2');
```

Specify the title.

```
title 'Sales Figures for Perishables in Northern Sectors';
run;
```

Output

Sales Figures for Perishables in Northern Sectors					1
Sector	Manager	Department		Perishable Total	
		Meat/Dairy	Produce		
Northeast	Alomar	\$190.00	\$86.00	\$276.00	
	Andrews	\$300.00	\$125.00	\$425.00	
Northwest	Brown	\$250.00	\$73.00	\$323.00	
	Pelfrey	\$205.00	\$76.00	\$281.00	
	Reveiz	\$600.00	\$30.00	\$630.00	
Combined sales for meat and dairy :			\$1,545.00		
Combined sales for produce :			\$390.00		
Combined sales for all perishables:			\$1,935.00		

Example 6: Displaying Multiple Statistics for One Variable

Procedure features:

PROC REPORT statement options:

LS=

PS=

COLUMN statement:

specifying statistics for stacked variables

DEFINE statement options:

FORMAT=

GROUP

ID

Data set: GROCERY on page 1038

Formats: \$MGRFMT. on page 1039

The report in this example displays six statistics for the sales for each manager's store. The output is too wide to fit all the columns on one page, so three of the statistics appear on the second page of the report. In order to make it easy to associate the statistics on the second page with their group, the report repeats the values of Manager and Sector on every page of the report.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=80 pagesize=60
      fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). HEADLINE underlines all column headings and the spaces between them at the top of each page of the report. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes. LS= sets the line size for the report to 66, and PS= sets the page size to 18.

```
proc report data=grocery nowd headline headskip
      ls=66 ps=18;
```

Specify the report columns. This COLUMN statement creates a column for Sector, Manager, and each of the six statistics that are associated with Sales.

```
column sector manager (Sum Min Max Range Mean Std),sales;
```

Define the group variables and the analysis variable. ID specifies that Manager is an ID variable. An ID variable and all columns to its left appear at the left of every page of a report. In this report, Sector and Manager are group variables. Each detail row of the report consolidates the information for all observations with the same values of the group variables. FORMAT= specifies the formats to use in the report.

```
define manager / group format=$mgrfmt. id;
define sector / group format=$sctrfmt.;
define sales / format=dollar11.2 ;
```

Specify the title.

```
title 'Sales Statistics for All Sectors';
run;
```


Output

Sales Statistics for All Sectors					1
Sector	Manager	Sum Sales	Min Sales	Max Sales	

Northeast	Alomar	\$786.00	\$86.00	\$420.00	
	Andrews	\$1,045.00	\$125.00	\$420.00	
Northwest	Brown	\$598.00	\$45.00	\$250.00	
	Pelfrey	\$746.00	\$45.00	\$420.00	
Southeast	Reveiz	\$1,110.00	\$30.00	\$600.00	
	Jones	\$630.00	\$40.00	\$300.00	
Southwest	Smith	\$350.00	\$50.00	\$120.00	
	Adams	\$695.00	\$40.00	\$350.00	
	Taylor	\$353.00	\$50.00	\$130.00	

Sales Statistics for All Sectors					2
Sector	Manager	Range Sales	Mean Sales	Std Sales	

Northeast	Alomar	\$334.00	\$196.50	\$156.57	
	Andrews	\$295.00	\$261.25	\$127.83	
Northwest	Brown	\$205.00	\$149.50	\$105.44	
	Pelfrey	\$375.00	\$186.50	\$170.39	
Southeast	Reveiz	\$570.00	\$277.50	\$278.61	
	Jones	\$260.00	\$157.50	\$123.39	
Southwest	Smith	\$70.00	\$87.50	\$29.86	
	Adams	\$310.00	\$173.75	\$141.86	
	Taylor	\$80.00	\$88.25	\$42.65	

Example 7: Storing and Reusing a Report Definition

Procedure features:

PROC REPORT statement options:

NAMED
OUTREPT=
REPORT=
WRAP

Other features:

TITLE statement
WHERE statement

Data set: GROCERY on page 1038

Formats: \$SCTRFMT., \$MGRFMT. and \$DEPTFMT. on page 1039

The first PROC REPORT step in this example creates a report that displays one value from each column of the report, using two rows to do so, before displaying another value from the first column. (By default, PROC REPORT displays values for only as

many columns as it can fit on one page. It fills a page with values for these columns before starting to display values for the remaining columns on the next page.)

Each item in the report is identified in the body of the report rather than in a column header.

The report definition created by the first PROC REPORT step is stored in a catalog entry. The second PROC REPORT step uses it to create a similar report for a different sector of the city.

Program to Store a Report Definition

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=80 pagesize=60
      fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). NAMED writes *name=* in front of each value in the report, where *name=* is the column heading for the value. When you use NAMED, PROC REPORT suppresses the display of column headings at the top of each page.

```
proc report data=grocery nowd
      named
      wrap
      ls=64 ps=36
      outrept=proclib.reports.namewrap;
```

Specify the report columns. The report contains a column for Sector, Manager, Department, and Sales.

```
column sector manager department sales;
```

Define the display and analysis variables. Because no usage is specified in the DEFINE statements, PROC REPORT uses the defaults. The character variables (Sector, Manager, and Department) are display variables. Sales is an analysis variable that is used to calculate the sum statistic. FORMAT= specifies the formats to use in the report.

```
define sector / format=$sctrfmt.;
define manager / format=$mgrfmt.;
define department / format=$deptfmt.;
define sales / format=dollar11.2;
```

Select the observations to process. A report definition might differ from the SAS program that creates the report. In particular, PROC REPORT stores neither WHERE statements nor TITLE statements.

```
where manager='1';
```

Specify the title. SYSDATE is an automatic macro variable that returns the date when the SAS job or SAS session began. The TITLE statement uses double rather than single quotation marks so that the macro variable resolves.

```
title "Sales Figures for Smith on &sysdate";
run;
```

Output

This is the output from the first PROC REPORT step, which creates the report definition.

Sales Figures for Smith on 04JAN02			1
Sector=Southeast	Manager=Smith	Department=Paper	
Sales=	\$50.00		
Sector=Southeast	Manager=Smith	Department=Meat/Dairy	
Sales=	\$100.00		
Sector=Southeast	Manager=Smith	Department=Canned	
Sales=	\$120.00		
Sector=Southeast	Manager=Smith	Department=Produce	
Sales=	\$80.00		

Program to Use a Report Definition

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 fmtsearch=(proclib);
```

Specify the report options, load the report definition, and select the observations to process. REPORT= uses the report definition that is stored in PROCLIB.REPORTS.NAMEWRAP to produce the report. The second report differs from the first one because it uses different WHERE and TITLE statements.

```
proc report data=grocery report=proclib.reports.namewrap
nowd;
where sector='sw';
title "Sales Figures for the Southwest Sector on &sysdate";
run;
```

Output

Sales Figures for the Southwest Sector on 04JAN02			1
Sector=Southwest	Manager=Taylor	Department=Paper	
Sector=Southwest	Manager=Taylor	Department=Meat/Dairy	
Sector=Southwest	Manager=Taylor	Department=Canned	
Sector=Southwest	Manager=Taylor	Department=Produce	
Sector=Southwest	Manager=Adams	Department=Paper	
Sector=Southwest	Manager=Adams	Department=Meat/Dairy	
Sector=Southwest	Manager=Adams	Department=Canned	
Sector=Southwest	Manager=Adams	Department=Produce	

Sales Figures for the Southwest Sector on 04JAN02			2
Sales=	\$53.00		
Sales=	\$130.00		
Sales=	\$120.00		
Sales=	\$50.00		
Sales=	\$40.00		
Sales=	\$350.00		
Sales=	\$225.00		
Sales=	\$80.00		

Example 8: Condensing a Report into Multiple Panels**Procedure features:**

PROC REPORT statement options:

FORMCHAR=

HEADLINE

LS=

PANELS=

PS=

PSPACE=

BREAK statement options:

SKIP

Other features:

SAS system option FORMCHAR=

Data set: GROCERY on page 1038**Formats:** \$MGRFMT. and \$DEPTFMT. on page 1039

The report in this example

- uses panels to condense a two-page report to one page. Panels compactly present information for long, narrow reports by placing multiple rows of information side by side.
- uses a default summary to place a blank line after the last row for each manager.
- changes the default underlining character for the duration of this PROC REPORT step.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=80 pagesize=60
      fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). HEADLINE underlines all column headings and the spaces between them at the top of each panel of the report. FORMCHAR= sets the value of the second formatting character (the one that HEADLINE uses) to the tilde (~). Therefore, the tilde underlines the column headings in the output. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes. LS= sets the line size for the report to 64, and PS= sets the page size to 18. PANELS= creates a multipanel report. Specifying PANELS=99 ensures that PROC REPORT fits as many panels as possible on one page. PSPACE=6 places six spaces between panels.

```
proc report data=grocery nowd headline
      formchar(2)='~'
      panels=99 pspace=6
      ls=64 ps=18;
```

Specify the report columns. The report contains a column for Manager, Department, and Sales.

```
column manager department sales;
```

Define the sort order and analysis columns. The values of all variables with the ORDER option in the DEFINE statement determine the order of the rows in the report. In this report, PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement) and then, within each value of Manager, by the values of Department. The ORDER= option specifies the sort order for a variable. This report arranges the values of Manager by their formatted values and arranges the values of Department by their internal values (np1, np2, p1, and p2). FORMAT= specifies the formats to use in the report.

```
define manager / order
      order=formatted
      format=$mgrfmt.;
define department / order
      order=internal
      format=$deptfmt.;
```

```
define sales / format=dollar7.2;
```

Produce a report summary. This BREAK statement produces a default summary after the last row for each manager. Because SKIP is the only option in the BREAK statement, each break consists of only a blank line.

```
break after manager / skip;
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the northwest or southwest sector.

```
where sector='nw' or sector='sw';
```

Specify the title.

```
title 'Sales for the Western Sectors';
run;
```

Output

Sales for the Western Sectors						1
Manager	Department	Sales	Manager	Department	Sales	
Adams	Paper	\$40.00	Reveiz	Paper	\$60.00	
	Canned	\$225.00		Canned	\$420.00	
	Meat/Dairy	\$350.00		Meat/Dairy	\$600.00	
	Produce	\$80.00		Produce	\$30.00	
Brown	Paper	\$45.00	Taylor	Paper	\$53.00	
	Canned	\$230.00		Canned	\$120.00	
	Meat/Dairy	\$250.00		Meat/Dairy	\$130.00	
	Produce	\$73.00		Produce	\$50.00	
Pelfrey	Paper	\$45.00				
	Canned	\$420.00				
	Meat/Dairy	\$205.00				
	Produce	\$76.00				

Example 9: Writing a Customized Summary on Each Page

Procedure features:

BREAK statement options:

OL

PAGE

SUMMARIZE

COMPUTE statement arguments:

```

with a computed variable as report-item
BEFORE break-variable
AFTER break-variable with conditional logic
BEFORE _PAGE_
DEFINE statement options:
  NOPRINT
LINE statement:
  pointer controls
  quoted text
  repeating a character string
  variable values and formats
Data set:  GROCERY on page 1038
Formats:  $SCTRFMT., $MGRFMT., and $DEPTFMT. on page 1039

```

The report in this example displays a record of one day's sales for each store. The rows are arranged so that all the information about one store is together, and the information for each store begins on a new page. Some variables appear in columns. Others appear only in the page header that identifies the sector and the store's manager.

The header that appears at the top of each page is created with the `_PAGE_` argument in the `COMPUTE` statement.

Profit is a computed variable based on the value of Sales and Department.

The text that appears at the bottom of the page depends on the total of Sales for the store. Only the first two pages of the report appear here.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=30
      fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). NOHEADER in the PROC REPORT statement suppresses the default column headings.

```
proc report data=grocery nowd
      headline headskip;
```

Specify the title.

```
title 'Sales for Individual Stores';
```

Specify the report columns. The report contains a column for Sector, Manager, Department, Sales, and Profit, but the NOPRINT option suppresses the printing of the columns for Sector and Manager. The page heading (created later in the program) includes their values. To get these variable values into the page heading, Sector and Manager must be in the COLUMN statement.

```
column sector manager department sales Profit;
```

Define the group, computed, and analysis variables. In this report, Sector, Manager, and Department are group variables. Each detail row of the report consolidates the information for all observations with the same values of the group variables. Profit is a computed variable whose values are calculated in the next section of the program. FORMAT= specifies the formats to use in the report.

```
define sector / group noprint;
define manager / group noprint;
define profit / computed format=dollar11.2;
define sales / analysis sum format=dollar11.2;
define department / group format=$deptfmt.;
```

Calculate the computed variable. Profit is computed as a percentage of Sales. For nonperishable items, the profit is 40% of the sale price. For perishable items the profit is 25%. Notice that in the compute block you must reference the variable Sales with a compound name (Sales.sum) that identifies both the variable and the statistic that you calculate with it.

```
compute profit;
  if department='np1' or department='np2'
    then profit=0.4*sales.sum;
  else profit=0.25*sales.sum;
endcomp;
```

Create a customized page header. This compute block executes at the top of each page, after PROC REPORT writes the title. It writes the page heading for the current manager's store. The LEFT option left-justifies the text in the LINE statements. Each LINE statement writes the text in quotation marks just as it appears in the statement. The first two LINE statements write a variable value with the format specified immediately after the variable's name.

```
compute before _page_ / left;
  line sector $sctrfmt. ' Sector';
  line 'Store managed by ' manager $mgrfmt.;
  line ' ';
  line ' ';
  line ' ';
endcomp;
```


Produce a report summary. This BREAK statement creates a default summary after the last row for each manager. OL writes a row of hyphens above the summary line. SUMMARIZE writes the value of Sales (the only analysis or computed variable) in the summary line. The PAGE option starts a new page after each default summary so that the page heading that is created in the preceding compute block always pertains to the correct manager.

```
break after manager / ol summarize page;
```

Produce a customized summary. This compute block places conditional text in a customized summary that appears after the last detail row for each manager.

```
compute after manager;
```

Specify the length of the customized summary text. The LENGTH statement assigns a length of 35 to the DATA step variable TEXT. In this particular case, the LENGTH statement is unnecessary because the longest version appears in the first IF/THEN statement. However, using the LENGTH statement ensures that even if the order of the conditional statements changes, TEXT will be long enough to hold the longest version.

```
length text $ 35;
```

Specify the conditional logic for the customized summary text. You cannot use the LINE statement in conditional statements (IF-THEN, IF-THEN/ELSE, and SELECT) because it does not take effect until PROC REPORT has executed all other statements in the compute block. These IF-THEN/ELSE statements assign a value to TEXT based on the value of Sales.sum in the summary row. A LINE statement writes that variable, whatever its value happens to be.

```
if sales.sum lt 500 then
    text='Sales are below the target region.';
else if sales.sum ge 500 and sales.sum lt 1000 then
    text='Sales are in the target region.';
else if sales.sum ge 1000 then
    text='Sales exceeded goal!';
line ' ';
line text $35.;
endcomp;
run;
```

Output

Sales for Individual Stores			1
Northeast Sector			
Store managed by Alomar			
Department	Sales	Profit	

Canned	\$420.00	\$168.00	
Meat/Dairy	\$190.00	\$47.50	
Paper	\$90.00	\$36.00	
Produce	\$86.00	\$21.50	
	-----	-----	
	\$786.00	\$196.50	
Sales are in the target region.			

Sales for Individual Stores			2
Northeast Sector			
Store managed by Andrews			
Department	Sales	Profit	

Canned	\$420.00	\$168.00	
Meat/Dairy	\$300.00	\$75.00	
Paper	\$200.00	\$80.00	
Produce	\$125.00	\$31.25	
	-----	-----	
	\$1,045.00	\$261.25	
Sales exceeded goal!			

Example 10: Calculating Percentages**Procedure features:**

COLUMN statement arguments:

PCTSUM

SUM

spanning headers

COMPUTE statement options:

CHAR

LENGTH=

DEFINE statement options:

COMPUTED

FLOW

WIDTH=

RBREAK statement options:

OL

SUMMARIZE

Other features:

TITLE statement

Data set: GROCERY on page 1038**Formats:** \$MGRFMT. and \$DEPTFMT. on page 1039

The summary report in this example shows the total sales for each store and the percentage that these sales represent of sales for all stores. Each of these columns has its own header. A single header also spans all the columns. This header looks like a title, but it differs from a title because it would be stored in a report definition. You must submit a null TITLE statement whenever you use the report definition, or the report will contain both a title and the spanning header.

The report includes a computed character variable, COMMENT, that flags stores with an unusually high percentage of sales. The text of COMMENT wraps across multiple rows. It makes sense to compute COMMENT only for individual stores. Therefore, the compute block that does the calculation includes conditional code that prevents PROC REPORT from calculating COMMENT on the summary line.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
      fmtsearch=(proclib);
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). HEADLINE underlines all column headings and the spaces between them at the top of each page of the report. The null TITLE statement suppresses the title of the report.

```
proc report data=grocery nowd headline;
  title;
```

Specify the report columns. The COLUMN statement uses the text in quotation marks as a spanning heading. The heading spans all the columns in the report because they are all included in the pair of parentheses that contains the heading. The COLUMN statement associates two statistics with Sales: Sum and Pctsum. The Sum statistic sums the values of Sales for all observations that are included in a row of the report. The Pctsum statistic shows what percentage of Sales that sum is for all observations in the report.

```
column ('Individual Store Sales as a Percent of All Sales'
       sector manager sales,(sum pctsum) comment);
```

Define the group and analysis columns. In this report, Sector and Manager are group variables. Each detail row represents a set of observations that have a unique combination of formatted values for all group variables. Sales is, by default, an analysis variable that is used to calculate the Sum statistic. However, because statistics are associated with Sales in the column statement, those statistics override the default. `FORMAT=` specifies the formats to use in the report. Text between quotation marks specifies the column heading.

```
define manager / group
               format=$mgrfmt.;
define sector / group
               format=$sctrfmt.;
define sales / format=dollar11.2
             '';
define sum / format=dollar9.2
           'Total Sales';
```

Define the percentage and computed columns. The `DEFINE` statement for `Pctsum` specifies a column heading, a format, and a column width of 8. The `PERCENT.` format presents the value of `Pctsum` as a percentage rather than a decimal. The `DEFINE` statement for `COMMENT` defines it as a computed variable and assigns it a column width of 20 and a blank column heading. The `FLOW` option wraps the text for `COMMENT` onto multiple lines if it exceeds the column width.

```
define pctsum / 'Percent of Sales' format=percent6. width=8;
define comment / computed width=20 '' flow;
```

Calculate the computed variable. Options in the `COMPUTE` statement define `COMMENT` as a character variable with a length of 40.

```
compute comment / char length=40;
```

Specify the conditional logic for the computed variable. For every store where sales exceeded 15% of the sales for all stores, this compute block creates a comment that says **Sales substantially above expectations**. Of course, on the summary row for the report, the value of `Pctsum` is 100. However, it is inappropriate to flag this row as having exceptional sales. The automatic variable `_BREAK_` distinguishes detail rows from summary rows. In a detail row, the value of `_BREAK_` is blank. The `THEN` statement executes only on detail rows where the value of `Pctsum` exceeds 0.15.

```
if sales.pctsum gt .15 and _break_ = ' '
then comment='Sales substantially above expectations.';
else comment=' ';
endcomp;
```

Produce the report summary. This RBREAK statement creates a default summary at the end of the report. OL writes a row of hyphens above the summary line. SUMMARIZE writes the values of Sales.sum and Sales.pctsum in the summary line.

```
rbreak after / ol summarize;
run;
```

Output

Individual Store Sales as a Percent of All Sales				
Sector	Manager	Total Sales	Percent of Sales	

Northeast	Alomar	\$786.00	12%	
	Andrews	\$1,045.00	17%	Sales substantially above expectations.
Northwest	Brown	\$598.00	9%	
	Pelfrey	\$746.00	12%	
	Reveiz	\$1,110.00	18%	Sales substantially above expectations.
Southeast	Jones	\$630.00	10%	
	Smith	\$350.00	6%	
Southwest	Adams	\$695.00	11%	
	Taylor	\$353.00	6%	
		-----	-----	
		\$6,313.00	100%	

Example 11: How PROC REPORT Handles Missing Values

Procedure features:

PROC REPORT statement options:

MISSING

COLUMN statement

with the N statistic

Other features:

TITLE statement

Formats: \$MGRFMT. on page 1039

This example illustrates the difference between the way PROC REPORT handles missing values for group (or order or across) variables with and without the MISSING option. The differences in the reports are apparent if you compare the values of N for each row and compare the totals in the default summary at the end of the report.

Program with Data Set with No Missing Values

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
      fmtsearch=(proclib);
```

Create the GROCMISS data set. GROCMISS is identical to GROCERY except that it contains some observations with missing values for Sector, Manager, or both.

```
data grocmis;
  input Sector $ Manager $ Department $ Sales @@;
datalines;
se 1 np1 50      . 1 p1 100      se . np2 120      se 1 p2 80
se 2 np1 40      se 2 p1 300      se 2 np2 220      se 2 p2 70
nw 3 np1 60      nw 3 p1 600      . 3 np2 420      nw 3 p2 30
nw 4 np1 45      nw 4 p1 250      nw 4 np2 230      nw 4 p2 73
nw 9 np1 45      nw 9 p1 205      nw 9 np2 420      nw 9 p2 76
sw 5 np1 53      sw 5 p1 130      sw 5 np2 120      sw 5 p2 50
. . np1 40      sw 6 p1 350      sw 6 np2 225      sw 6 p2 80
ne 7 np1 90      ne . p1 190      ne 7 np2 420      ne 7 p2 86
ne 8 np1 200     ne 8 p1 300      ne 8 np2 420      ne 8 p2 125
;
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). HEADLINE underlines all column headings and the spaces between them.

```
proc report data=grocmis nowd headline;
```

Specify the report columns. The report contains columns for Sector, Manager, the N statistic, and Sales.

```
column sector manager N sales;
```

Define the group and analysis variables. In this report, Sector and Manager are group variables. Sales is, by default, an analysis variable that is used to calculate the Sum statistic. Each detail row represents a set of observations that have a unique combination of formatted values for all group variables. The value of Sales in each detail row is the sum of Sales for all observations in the group. In this PROC REPORT step, the procedure does not include observations with a missing value for the group variable. FORMAT= specifies formats to use in the report.

```
define sector / group format=$sctrfmt.;
define manager / group format=$mgrfmt.;
```

```
define sales / format=dollar9.2;
```

Produce a report summary. This RBREAK statement creates a default summary at the end of the report. DOL writes a row of equal signs above the summary line. SUMMARIZE writes the values of N and Sales.sum in the summary line.

```
rbreak after / dol summarize;
```

Specify the title.

```
title 'Summary Report for All Sectors and Managers';
run;
```

Output with No Missing Values

Summary Report for All Sectors and Managers				1
Sector	Manager	N	Sales	

Northeast	Alomar	3	\$596.00	
	Andrews	4	\$1,045.00	
Northwest	Brown	4	\$598.00	
	Pelfrey	4	\$746.00	
	Reveiz	3	\$690.00	
Southeast	Jones	4	\$630.00	
	Smith	2	\$130.00	
Southwest	Adams	3	\$655.00	
	Taylor	4	\$353.00	
		=====	=====	
		31	\$5,443.00	

Program with Data Set with Missing Values

Include the missing values. The MISSING option in the second PROC REPORT step includes the observations with missing values for the group variable.

```
proc report data=grocmiss nowd headline missing;
  column sector manager N sales;
  define sector / group format=$sctrfmt.;
  define manager / group format=$mgrfmt.;
  define sales / format=dollar9.2;
  rbreak after / dol summarize;
run;
```

Output with Missing Values

Summary Report for All Sectors and Managers				2
Sector	Manager	N	Sales	

		1	\$40.00	
	Reveiz	1	\$420.00	
	Smith	1	\$100.00	
Northeast		1	\$190.00	
	Alomar	3	\$596.00	
	Andrews	4	\$1,045.00	
Northwest	Brown	4	\$598.00	
	Pelfrey	4	\$746.00	
	Reveiz	3	\$690.00	
Southeast		1	\$120.00	
	Jones	4	\$630.00	
	Smith	2	\$130.00	
Southwest	Adams	3	\$655.00	
	Taylor	4	\$353.00	
		=====	=====	
		36	\$6,313.00	

Example 12: Creating and Processing an Output Data Set

Procedure features:

PROC REPORT statement options:

BOX

OUT=

DEFINE statement options:

ANALYSIS

GROUP

NOPRINT

SUM

Other features:

Data set options:

WHERE=

Data set: GROCERY on page 1038

Formats: \$MGRFMT. on page 1039

This example uses WHERE processing as it builds an output data set. This technique enables you to do WHERE processing after you have consolidated multiple observations into a single row.

The first PROC REPORT step creates a report (which it does not display) in which each row represents all the observations from the input data set for a single manager. The second PROC REPORT step builds a report from the output data set. This report uses line-drawing characters to separate the rows and columns.

Program to Create Output Data Set

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
      fmtsearch=(proclib);
```

Specify the report options and columns. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). OUT= creates the output data set TEMP. The output data set contains a variable for each column in the report (Manager and Sales) as well as for the variable _BREAK_, which is not used in this example. Each observation in the data set represents a row of the report. Because Manager is a group variable and Sales is an analysis variable that is used to calculate the Sum statistic, each row in the report (and therefore each observation in the output data set) represents multiple observations from the input data set. In particular, each value of Sales in the output data set is the total of all values of Sales for that manager. The WHERE= data set option in the OUT= option filters those rows as PROC REPORT creates the output data set. Only those observations with sales that exceed \$1,000 become observations in the output data set.

```
proc report data=grocery nowd
      out=temp( where=(sales gt 1000) );
  column manager sales;
```

Define the group and analysis variables. Because the definitions of all report items in this report include the NOPRINT option, PROC REPORT does not print a report. However, the PROC REPORT step does execute and create an output data set.

```
define manager / group noprint;
define sales / analysis sum noprint;
run;
```

Output Showing the Output Data Set

This is the output data set that PROC REPORT creates. It is used as the input set in the second PROC REPORT step.

The Data Set TEMP			1
Manager	Sales	BREAK	
3	1110		
8	1045		

Program That Uses the Output Data Set

Specify the report options and columns, define the group and analysis columns, and specify the titles. DATA= specifies the output data set from the first PROC REPORT step as the input data set for this report. The BOX option draws an outline around the output, separates the column headings from the body of the report, and separates rows and columns of data. The TITLE statements specify a title for the report.

```
proc report data=temp box nowd;
  column manager sales;
  define manager / group format=$mgrfmt.;
  define sales / analysis sum format=dollar11.2;
  title 'Managers with Daily Sales';
  title2 'of over';
  title3 'One Thousand Dollars';
run;
```

Report Based on the Output Data Set

Managers with Daily Sales		1
of over		
One Thousand Dollars		

Manager	Sales	

Andrews	\$1,045.00	

Reveiz	\$1,110.00	

Example 13: Storing Computed Variables as Part of a Data Set

Procedure features:

PROC REPORT statement options:

OUT=

COMPUTE statement:

with a computed variable as *report-item*

DEFINE statement options:

COMPUTED

Other features: CHART procedure

Data set: GROCERY on page 1038

Formats: \$SCTRFMT. on page 1039

The report in this example

- ☐ creates a computed variable

- stores it in an output data set
- uses that data set to create a chart based on the computed variable.

Program That Creates the Output Data Set

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
      fmtsearch=(proclib);
```

Delete any existing titles.

```
title;
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window and sends its output to the open output destination(s). OUT= creates the output data set PROFIT.

```
proc report data=grocery nowd out=profit;
```

Specify the report columns. The report contains columns for Manager, Department, Sales, and Profit, which is not in the input data set. Because the purpose of this report is to generate an output data set to use in another procedure, the report layout simply uses the default usage for all the data set variables to list all the observations. DEFINE statements for the data set variables are unnecessary.

```
column sector manager department sales Profit;
```

Define the computed column. The COMPUTED option tells PROC REPORT that Profit is defined in a compute block somewhere in the PROC REPORT step.

```
define profit / computed;
```

Calculate the computed column. Profit is computed as a percentage of Sales. For nonperishable items, the profit is 40% of the sale price. For perishable items the profit is 25%. Notice that in the compute block, you must reference the variable Sales with a compound name (Sales.sum) that identifies both the variable and the statistic that you calculate with it.

```

/* Compute values for Profit. */
compute profit;
  if department='np1' or department='np2' then profit=0.4*sales.sum;
  else profit=0.25*sales.sum;
endcomp;
run;

```

The Output Data Set

This is the output data set that is created by PROC REPORT. It is used as input for PROC CHART.

The Data Set PROFIT					1
Sector	Manager	Department	Sales	Profit	__BREAK__
se	1	np1	50	20	
se	1	p1	100	25	
se	1	np2	120	48	
se	1	p2	80	20	
se	2	np1	40	16	
se	2	p1	300	75	
se	2	np2	220	88	
se	2	p2	70	17.5	
nw	3	np1	60	24	
nw	3	p1	600	150	
nw	3	np2	420	168	
nw	3	p2	30	7.5	
nw	4	np1	45	18	
nw	4	p1	250	62.5	
nw	4	np2	230	92	
nw	4	p2	73	18.25	
nw	9	np1	45	18	
nw	9	p1	205	51.25	
nw	9	np2	420	168	
nw	9	p2	76	19	
sw	5	np1	53	21.2	
sw	5	p1	130	32.5	
sw	5	np2	120	48	
sw	5	p2	50	12.5	
sw	6	np1	40	16	
sw	6	p1	350	87.5	
sw	6	np2	225	90	
sw	6	p2	80	20	
ne	7	np1	90	36	
ne	7	p1	190	47.5	
ne	7	np2	420	168	
ne	7	p2	86	21.5	
ne	8	np1	200	80	
ne	8	p1	300	75	
ne	8	np2	420	168	
ne	8	p2	125	31.25	

Program That Uses the Output Data Set

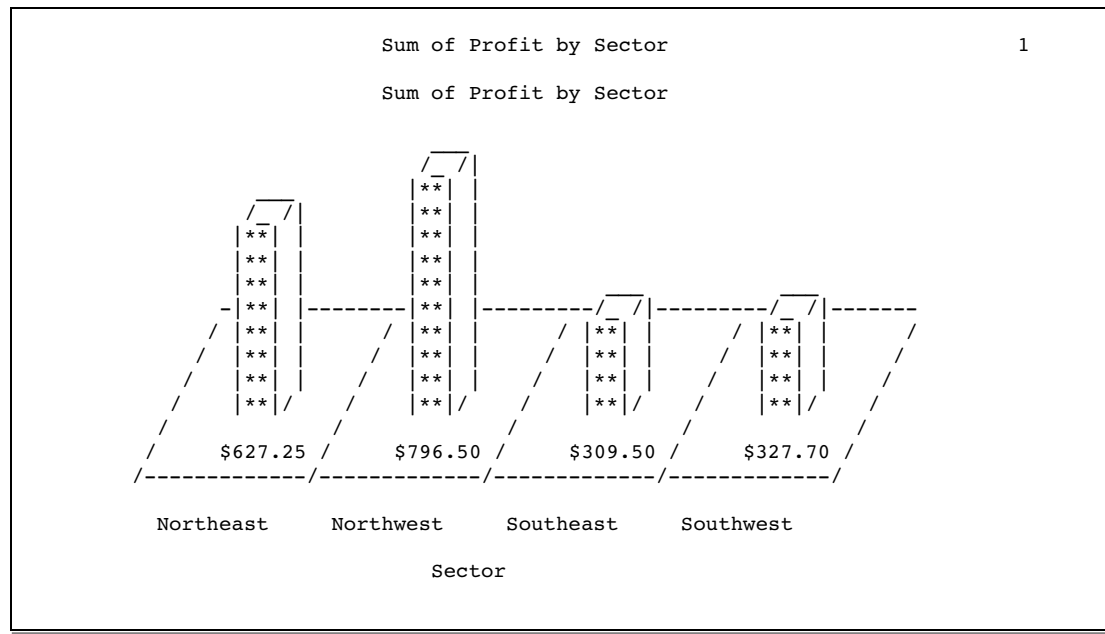
Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=80 pagesize=60
      fmtsearch=(proclib);
```

Chart the data in the output data set. PROC CHART uses the output data set from the previous PROC REPORT step to chart the sum of Profit for each sector.

```
proc chart data=profit;
  block sector / sumvar=profit;
  format sector $ctrfmt.;
  format profit dollar7.2;
  title 'Sum of Profit by Sector';
run;
```

Output from Processing the Output Data Set



Example 14: Using a Format to Create Groups

Procedure features:

DEFINE statement options:

GROUP

Other features: FORMAT procedure

Data set: GROCERY on page 1038

Formats: \$MGRFMT. on page 1039

This example shows how to use formats to control the number of groups that PROC REPORT creates. The program creates a format for Department that classifies the four departments as one of two types: perishable or nonperishable. Consequently, when Department is an across variable, PROC REPORT creates only two columns instead of four. The column header is the formatted value of the variable.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. FMTSEARCH= specifies the library to include when searching for user-created formats.

```
options nodate pageno=1 linesize=64 pagesize=60
      fmtsearch=(proclib);
```

Create the \$PERISH. format. PROC FORMAT creates a format for Department. This variable has four different values in the data set, but the format has only two values.

```
proc format;
  value $perish 'p1','p2'='Perishable'
               'np1','np2'='Nonperishable';
run;
```

Specify the report options. The NOWD option runs the REPORT procedure without the REPORT window and sends its output to the open output destination(s). HEADLINE underlines all column headings and the spaces between them at the top of each page of the report. HEADSKIP writes a blank line beneath the underlining that HEADLINE writes.

```
proc report data=grocery nowd
      headline
      headskip;
```

Specify the report columns. Department and Sales are separated by a comma in the COLUMN statement, so they collectively determine the contents of the column that they define. Because Sales is an analysis variable, its values fill the cells that are created by these two variables. The report also contains a column for Manager and a column for Sales by itself (which is the sales for all departments).

```
column manager department,sales sales;
```

Define the group and across variables. Manager is a group variable. Each detail row of the report consolidates the information for all observations with the same value of Manager. Department is an across variable. PROC REPORT creates a column and a column heading for each formatted value of Department. ORDER=FORMATTED arranges the values of Manager and Department alphabetically according to their formatted values. FORMAT= specifies the formats to use. The empty quotation marks in the definition of Department specify a blank column heading, so no heading spans all the departments. However, PROC REPORT uses the formatted values of Department to create a column heading for each individual department.

```
define manager / group order=formatted
                format=$mgrfmt.;
define department / across order=formatted
                format=$perish. '';
```

Define the analysis variable. Sales is an analysis variable that is used to calculate the Sum statistic. Sales appears twice in the COLUMN statement, and the same definition applies to both occurrences. FORMAT= specifies the format to use in the report. WIDTH= specifies the width of the column. Notice that the column headings for the columns that both Department and Sales create are a combination of the heading for Department and the (default) heading for Sales.

```
define sales / analysis sum
                format=dollar9.2 width=13;
```

Produce a customized summary. This COMPUTE statement begins a compute block that produces a customized summary at the end of the report. The LINE statement places the quoted text and the value of Sales.sum (with the DOLLAR9.2 format) in the summary. An ENDCOMP statement must end the compute block.

```
compute after;
  line ' ';
  line 'Total sales for these stores were: '
      sales.sum dollar9.2;
endcomp;
```

Specify the title.

```
title 'Sales Summary for All Stores';
run;
```

Output

Sales Summary for All Stores				1
Manager	Nonperishable Sales	Perishable Sales	Sales	

Adams	\$265.00	\$430.00	\$695.00	
Alomar	\$510.00	\$276.00	\$786.00	
Andrews	\$620.00	\$425.00	\$1,045.00	
Brown	\$275.00	\$323.00	\$598.00	
Jones	\$260.00	\$370.00	\$630.00	
Pelfrey	\$465.00	\$281.00	\$746.00	
Reveiz	\$480.00	\$630.00	\$1,110.00	
Smith	\$170.00	\$180.00	\$350.00	
Taylor	\$173.00	\$180.00	\$353.00	
Total sales for these stores were: \$6,313.00				

Example 15: Specifying Style Elements for ODS Output in the PROC REPORT Statement

Procedure features: STYLE= option in the PROC REPORT statement

Other features:

ODS HTML statement

ODS PDF statement

ODS RTF statement

Data set: GROCERY on page 1038

Formats: \$MGRFMT. and \$DEPTFMT. on page 1039

This example creates HTML, PDF, and RTF files and sets the style elements for each location in the report in the PROC REPORT statement.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. FMTSEARCH= specifies the library to include when searching for user-created formats. LINESIZE= and PAGESIZE= are not set for this example because they have no effect on HTML, RTF, and Printer output.

```
options nodate pageno=1 fmtsearch=(proclib);
```

Specify the ODS output filenames. By opening multiple ODS destinations, you can produce multiple output files in a single execution. The ODS HTML statement produces output that is written in HTML. The ODS PDF statement produces output in Portable Document Format (PDF). The ODS RTF statement produces output in Rich Text Format (RTF). The output from PROC REPORT goes to each of these files.


```
ods html body='external-HTML-file';
ods pdf file='external-PDF-file';
ods rtf file='external-RTF-file';
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window. In this case, SAS writes the output to the traditional procedure output, the HTML body file, and the RTF and PDF files.

```
proc report data=grocery nowd headline headskip
```

Specify the style attributes for the report. This STYLE= option sets the style element for the structural part of the report. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for CELLSPACING=, BORDERWIDTH=, and BORDERCOLOR=.

```
style(report)=[cellspacing=5 borderwidth=10 bordercolor=blue]
```

Specify the style attributes for the column headings. This STYLE= option sets the style element for all column headings. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for those that are specified here.

```
style(header)=[foreground=yellow
               font_style=italic font_size=6]
```

Specify the style attributes for the report columns. This STYLE= option sets the style element for all the cells in all the columns. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for those that are specified here.

```
style(column)=[foreground=moderate brown
               font_face=helvetica font_size=4]
```

Specify the style attributes for the compute block lines. This STYLE= option sets the style element for all the LINE statements in all compute blocks. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for those that are specified here.

```
style(lines)=[foreground=white background=black
               font_style=italic font_weight=bold font_size=5]
```

Specify the style attributes for report summaries. This STYLE= option sets the style element for all the default summary lines. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for those that are specified here.

```
style(summary)=[foreground=cx3e3d73 background=cxaeadd9
                 font_face=helvetica font_size=3 just=r];
```

Specify the report columns. The report contains columns for Manager, Department, and Sales.

```
column manager department sales;
```

Define the sort order variables. In this report Manager and Department are order variables. PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement), then by the value of Department. For Manager, ORDER= specifies that values of Manager are arranged according to their formatted values; similarly, for Department, ORDER= specifies that values of Department are arranged according to their internal values. FORMAT= specifies the format to use for each variable. Text in quotation marks specifies the column headings.

```
define manager / order
    order=formatted
    format=$mgrfmt.
    'Manager';
define department / order
    order=internal
    format=$deptfmt.
    'Department';
```

Produce a report summary. The BREAK statement produces a default summary after the last row for each manager. SUMMARIZE writes the values of Sales (the only analysis or computed variable in the report) in the summary line. PROC REPORT sums the values of Sales for each manager because Sales is an analysis variable that is used to calculate the Sum statistic.

```
break after manager / summarize;
```

Produce a customized summary. The COMPUTE statement begins a compute block that produces a customized summary after each value of Manager. The LINE statement places the quoted text and the values of Manager and Sales.sum (with the formats \$MGRFMT. and DOLLAR7.2) in the summary. An ENDCOMP statement must end the compute block.

```
compute after manager;
    line 'Subtotal for ' manager $mgrfmt. 'is '
        sales.sum dollar7.2 '.';
endcomp;
```

Produce a customized end-of-report summary. This COMPUTE statement begins a compute block that executes at the end of the report. The LINE statement writes the quoted text and the value of Sales.sum (with the DOLLAR7.2 format). An ENDCOMP statement must end the compute block.

```
compute after;
    line 'Total for all departments is: '
        sales.sum dollar7.2 '.';
endcomp;
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the southeast sector.

```
where sector='se';
```

Specify the title.

```
title 'Sales for the Southeast Sector';
run;
```

Close the ODS destinations.

```
ods html close;
ods pdf close;
ods rtf close;
```

HTML Output

Sales for the Southeast Sector		
<i>Manager</i>	<i>Department</i>	<i>Sales</i>
Jones	Paper	40
	Canned	220
	Meat/Dairy	300
	Produce	70
<i>Jones</i>		<i>630</i>
Subtotal for Jones is \$630.00.		
Smith	Paper	50
	Canned	120
	Meat/Dairy	100
	Produce	80
<i>Smith</i>		<i>350</i>
Subtotal for Smith is \$350.00.		
Total for all departments is: \$980.00.		

PDF Output*Sales for the Southeast Sector*

<i>Manager</i>	<i>Department</i>	<i>Sales</i>
Jones	Paper	40
	Canned	220
	Meat/Dairy	300
	Produce	70
<i>Jones</i>		<i>630</i>
<i>Subtotal for Jones is \$630.00.</i>		
Smith	Paper	50
	Canned	120
	Meat/Dairy	100
	Produce	80
<i>Smith</i>		<i>350</i>
<i>Subtotal for Smith is \$350.00.</i>		
<i>Total for all departments is: \$980.00.</i>		

RTF Output*Sales for the Southeast Sector*

<i>Manager</i>	<i>Department</i>	<i>Sales</i>
Jones	Paper	40
	Canned	220
	Meat/Dairy	300
	Produce	70
<i>Jones</i>		<i>630</i>
<i>Subtotal for Jones is \$630.00.</i>		
Smith	Paper	50
	Canned	120
	Meat/Dairy	100
	Produce	80
<i>Smith</i>		<i>350</i>
<i>Subtotal for Smith is \$350.00.</i>		
<i>Total for all departments is: \$980.00.</i>		

Example 16: Specifying Style Elements for ODS Output in Multiple Statements**Procedure features:**

STYLE= option in
 PROC REPORT statement
 CALL DEFINE statement
 COMPUTE statement
 DEFINE statement

Other features:

ODS HTML statement

ODS PDF statement

ODS RTF statement

Data set: GROCERY on page 1038

Formats: \$MGRFMT. on page 1039 and \$DEPTFMT. on page 1039

This example creates HTML, PDF, and RTF files and sets the style elements for each location in the report in the PROC REPORT statement. It then overrides some of these settings by specifying style elements in other statements.

Program

Declare the PROCLIB library. The PROCLIB library is used to store user-created formats.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. FMTSEARCH= specifies the library to include when searching for user-created formats. LINESIZE= and PAGESIZE= are not set for this example because they have no effect on HTML, RTF, and Printer output.

```
options nodate pageno=1 fmtsearch=(proclib);
```

Specify the ODS output filenames. By opening multiple ODS destinations, you can produce multiple output files in a single execution. The ODS HTML statement produces output that is written in HTML. The ODS PDF statement produces output in Portable Document Format (PDF). The ODS RTF statement produces output in Rich Text Format (RTF). The output from PROC REPORT goes to each of these files.

```
ods html body='external-HTML-file';
ods pdf file='external-PDF-file';
ods rtf file='external-RTF-file';
```

Specify the report options. The NOWD option runs PROC REPORT without the REPORT window. In this case, SAS writes the output to the traditional procedure output, the HTML body file, and the RTF and PDF files.

```
proc report data=grocery nowd headline headskip
```

Specify the style attributes for the report. This STYLE= option sets the style element for the structural part of the report. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for those that are specified here.

```
style(report)=[cellspacing=5 borderwidth=10 bordercolor=blue]
```

Specify the style attributes for the column headings. This STYLE= option sets the style element for all column headings. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for those that are specified here.

```
style(header)=[foreground=yellow
               font_style=italic font_size=6]
```

Specify the style attributes for the report columns. This STYLE= option sets the style element for all the cells in all the columns. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for those that are specified here.

```
style(column)=[foreground=moderate brown
               font_face=helvetica font_size=4]
```

Specify the style attributes for the compute block lines. This STYLE= option sets the style element for all the LINE statements in all compute blocks. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for those that are specified here.

```
style(lines)=[foreground=white background=black
              font_style=italic font_weight=bold font_size=5]
```

Specify the style attributes for the report summaries. This STYLE= option sets the style element for all the default summary lines. Because no style element is specified, PROC REPORT uses all the style attributes of the default style element for this location except for those that are specified here.

```
style(summary)=[foreground=cx3e3d73 background=cxaeadd9
                font_face=helvetica font_size=3 just=r];
```

Specify the report columns. The report contains columns for Manager, Department, and Sales.

```
column manager department sales;
```

Define the first sort order variable. In this report Manager is an order variable. PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement). ORDER= specifies that values of Manager are arranged according to their formatted values. FORMAT= specifies the format to use for this variable. Text in quotation marks specifies the column headings.

```
define manager / order
              order=formatted
              format=$mgrfmt.
              'Manager'
```

Specify the style attributes for the first sort order variable column heading. The STYLE= option sets the foreground and background colors of the column heading for Manager. The other style attributes for the column heading will match those that were established for the HEADER location in the PROC REPORT statement.

```
style(header)=[foreground=white
               background=black];
```

Define the second sort order variable. In this report Department is an order variable. PROC REPORT arranges the rows first by the value of Manager (because it is the first variable in the COLUMN statement), then by the value of Department. ORDER= specifies that values of Department are arranged according to their internal values. FORMAT= specifies the format to use for this variable. Text in quotation marks specifies the column heading.

```
define department / order
                  order=internal
```

```
format=$deptfmt.
'Department'
```

Specify the style attributes for the second sort order variable column. The STYLE= option sets the font of the cells in the column Department to italic. The other style attributes for the cells will match those that were established for the COLUMN location in the PROC REPORT statement.

```
style(column)=[font_style=italic];
```

Produce a report summary. The BREAK statement produces a default summary after the last row for each manager. SUMMARIZE writes the values of Sales (the only analysis or computed variable in the report) in the summary line. PROC REPORT sums the values of Sales for each manager because Sales is an analysis variable that is used to calculate the Sum statistic.

```
break after manager / summarize;
```

Produce a customized summary. The COMPUTE statement begins a compute block that produces a customized summary at the end of the report. This STYLE= option specifies the style element to use for the text that is created by the LINE statement in this compute block. This style element switches the foreground and background colors that were specified for the LINES location in the PROC REPORT statement. It also changes the font style, the font weight, and the font size.

```
compute after manager
  / style=[font_style=roman font_size=3 font_weight=bold
    background=white foreground=black];
```

Specify the text for the customized summary. The LINE statement places the quoted text and the values of Manager and Sales.sum (with the formats \$MGRFMT. and DOLLAR7.2) in the summary. An ENDCOMP statement must end the compute block.

```
line 'Subtotal for ' manager $mgrfmt. 'is '
    sales.sum dollar7.2 '.';
endcomp;
```

Produce a customized background for the analysis column. This compute block specifies a background color and a bold font for all cells in the Sales column that contain values of 100 or greater and that are not summary lines.

```
compute sales;
  if sales.sum>100 and _break_=' ' then
    call define(_col_, "style",
      "style=[background=yellow
        font_face=helvetica
        font_weight=bold]");
endcomp;
```

Produce a customized end-of-report summary. This COMPUTE statement begins a compute block that executes at the end of the report. The LINE statement writes the quoted text and the value of Sales.sum (with the DOLLAR7.2 format). An ENDCOMP statement must end the compute block.

```
compute after;
  line 'Total for all departments is: '
    sales.sum dollar7.2 '.';
```



```
endcomp;
```

Select the observations to process. The WHERE statement selects for the report only the observations for stores in the southeast sector.

```
where sector='se';
```

Specify the title.

```
title 'Sales for the Southeast Sector';  
run;
```

Close the ODS destinations.

```
ods html close;  
ods pdf close;  
ods rtf close;
```

HTML Body File**Sales for the Southeast Sector**

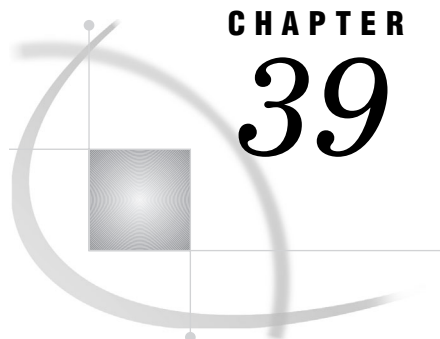
Manager	Department	Sales
Jones	Paper	40
	Canned	220
	Meat/Dairy	300
	Produce	70
Jones		630
Subtotal for Jones is \$630.00.		
Smith	Paper	50
	Canned	120
	Meat/Dairy	100
	Produce	80
Smith		350
Subtotal for Smith is \$350.00.		
Total for all departments is: \$980.00.		

PDF Output*Sales for the Southeast Sector*

<i>Manager</i>	<i>Department</i>	<i>Sales</i>
Jones	<i>Paper</i>	40
	<i>Canned</i>	220
	<i>Meat/Dairy</i>	300
	<i>Produce</i>	70
<i>Jones</i>		630
Subtotal for Jones is \$630.00.		
Smith	<i>Paper</i>	50
	<i>Canned</i>	120
	<i>Meat/Dairy</i>	100
	<i>Produce</i>	80
<i>Smith</i>		350
Subtotal for Smith is \$350.00.		
<i>Total for all departments is: \$980.00.</i>		

RTF Output*Sales for the Southeast Sector*

<i>Manager</i>	<i>Department</i>	<i>Sales</i>
Jones	<i>Paper</i>	40
	<i>Canned</i>	220
	<i>Meat/Dairy</i>	300
	<i>Produce</i>	70
Jones		630
Subtotal for Jones is \$630.00.		
Smith	<i>Paper</i>	50
	<i>Canned</i>	120
	<i>Meat/Dairy</i>	100
	<i>Produce</i>	80
Smith		350
Subtotal for Smith is \$350.00.		
<i>Total for all departments is: \$980.00.</i>		



CHAPTER

39

The SORT Procedure

<i>Overview: SORT Procedure</i>	1091
<i>Syntax: SORT Procedure</i>	1093
<i>PROC SORT Statement</i>	1093
<i>BY Statement</i>	1099
<i>Concepts: SORT Procedure</i>	1100
<i>Multi-threaded Sorting</i>	1100
<i>Sorting Orders for Numeric Variables</i>	1100
<i>Sorting Orders for Character Variables</i>	1101
<i>EBCDIC Order</i>	1101
<i>ASCII Order</i>	1101
<i>Specifying Sorting Orders for Character Variables</i>	1101
<i>Stored Sort Information</i>	1102
<i>Integrity Constraints: SORT Procedure</i>	1102
<i>Results: SORT Procedure</i>	1102
<i>Procedure Output</i>	1103
<i>Output Data Set</i>	1103
<i>Examples: SORT Procedure</i>	1103
<i>Example 1: Sorting by the Values of Multiple Variables</i>	1103
<i>Example 2: Sorting in Descending Order</i>	1105
<i>Example 3: Maintaining the Relative Order of Observations in Each BY Group</i>	1107
<i>Example 4: Retaining the First Observation of Each BY Group</i>	1110

Overview: SORT Procedure

The SORT procedure sorts observations in a SAS data set by one or more character or numeric variables. The SORT procedure either replaces the original data set or creates a new data set. PROC SORT produces only an output data set. For more information, see “Procedure Output” on page 1103.

Operating Environment Information: The sorting capabilities that are described in this chapter are available for all operating environments. In addition, if you use the HOST value of the SAS system option SORTPGM=, you might be able to use other sorting options that are available only for your operating environment. Refer to the SAS documentation for your operating environment for information about other sorting capabilities Δ

In the following example, the original data set was in alphabetical order by last name. PROC SORT replaces the original data set with a data set that is sorted by employee identification number. Output 39.1 on page 1092 shows the log that results from running this PROC SORT step. Output 39.2 on page 1092 shows the results of the PROC PRINT step. The statements that produce the output follow:

```

proc sort data=employee;
    by idnumber;
run;

proc print data=employee;
run;

```

Output 39.1 SAS Log Generated by PROC SORT

```

NOTE: There were 6 observations read from the data set WORK.EMPLOYEE.
NOTE: The data set WORK.EMPLOYEE has 6 observations and 3 variables.
NOTE: PROCEDURE SORT used:
      real time          0.01 seconds
      cpu time           0.01 seconds

```

Output 39.2 Observations Sorted by the Values of One Variable

The SAS System			1
Obs	Name	IDnumber	
1	Belloit	1988	
2	Wesley	2092	
3	Lemeux	4210	
4	Arnsbarger	5466	
5	Pierce	5779	
6	Capshaw	7338	

The following output shows the results of a more complicated sort by three variables. The businesses in this example are sorted by town, then by debt from highest amount to lowest amount, then by account number. For an explanation of the program that produces this output, see Example 2 on page 1105.

Output 39.3

Customers with Past-Due Accounts Listed by Town, Amount, Account Number					1
Obs	Company	Town	Debt	Account Number	
1	Paul's Pizza	Apex	83.00	1019	
2	Peter's Auto Parts	Apex	65.79	7288	
3	Watson Tabor Travel	Apex	37.95	3131	
4	Tina's Pet Shop	Apex	37.95	5108	
5	Apex Catering	Apex	37.95	9923	
6	Deluxe Hardware	Garner	467.12	8941	
7	Boyd & Sons Accounting	Garner	312.49	4762	
8	World Wide Electronics	Garner	119.95	1122	
9	Elway Piano and Organ	Garner	65.79	5217	
10	Ice Cream Delight	Holly Springs	299.98	2310	
11	Tim's Burger Stand	Holly Springs	119.95	6335	
12	Strickland Industries	Morrisville	657.22	1675	
13	Pauline's Antiques	Morrisville	302.05	9112	
14	Bob's Beds	Morrisville	119.95	4998	

Syntax: SORT Procedure

Requirements: BY statement

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, “Statements with the Same Function in Multiple Procedures,” on page 53 for details. You can also use any global statements as well. See “Global Statements” on page 18 for a list.

PROC SORT *<collating-sequence-option>* *<other option(s)>*;

BY *<DESCENDING>* *variable-1* *<...<DESCENDING>* *variable-n*;

PROC SORT Statement

PROC SORT *<collating-sequence-option>* *<other option(s)>*;

To do this

Specify the collating sequence

Specify ASCII

Specify EBCDIC

Specify Danish

Specify Finnish

Use this option

ASCII

EBCDIC

DANISH

FINNISH

To do this	Use this option
Specify Norwegian	NORWEGIAN
Specify Swedish	SWEDISH
Specify a customized sequence	NATIONAL
Specify any of these collating sequences: ASCII, EBCDIC, DANISH, FINNISH, ITALIAN, NORWEGIAN, SPANISH, SWEDISH	SORTSEQ=
Specify the input data set	DATA=
Sort a SAS data set without changing the created and modified dates	DATECOPY
Create an output data set	OUT=
Specify the output order	
Reverse the collation order for character variables	REVERSE
Maintain relative order within BY groups	EQUALS
Do not maintain relative order within BY groups	NOEQUALS
Eliminate duplicate observations	
Delete observations with duplicate BY values	NODUPKEY
Delete duplicate observations	NODUPRECS
Specify the available memory	SORTSIZE=
Force redundant sorting	FORCE
Reduce temporary disk usage	TAGSORT
Override SAS system option THREADS	
Enable multi-threaded sorting	THREADS
Prevent multi-threaded sorting	NOTHREADS

Options

Options can include one *collating-sequence-option* and multiple *other options*. The order of the two types of options does not matter and both types are not necessary in the same PROC SORT step.

Collating-Sequence-Options

Operating Environment Information: For information about behavior specific to your operating environment for the DANISH, FINNISH, NORWEGIAN, or SWEDISH *collating-sequence-option*, see the SAS documentation for your operating environment. Δ

Restriction: You can specify only one *collating-sequence-option* in a PROC SORT step.

ASCII

sorts character variables using the ASCII collating sequence. You need this option only when you sort by ASCII on a system where EBCDIC is the native collating sequence.

See also: “Sorting Orders for Character Variables” on page 1101

DANISH

NORWEGIAN

sort characters according to the Danish and Norwegian national standard.

The Danish and Norwegian collating sequence is shown in Figure 39.1 on page 1096.

EBCDIC

sorts character variables using the EBCDIC collating sequence. You need this option only when you sort by EBCDIC on a system where ASCII is the native collating sequence.

See also: “Sorting Orders for Character Variables” on page 1101

FINNISH

SWEDISH

sorts characters according to the Finnish and Swedish national standard. The Finnish and Swedish collating sequence is shown in Figure 39.1 on page 1096.

NATIONAL

sorts character variables using an alternate collating sequence, as defined by your installation, to reflect a country’s National Use Differences. To use this option, your site must have a customized national sort sequence defined. Check with the SAS Installation Representative at your site to determine if a customized national sort sequence is available.

NORWEGIAN

See DANISH.

SORTSEQ=collating-sequence

specifies the collating sequence. The value of *collating-sequence* can be any one of the *collating-sequence-options* in the PROC SORT statement, or the value can be the name of a translation table, either a default translation table or one that you have created in the TRANTAB procedure. For an example of using PROC TRANTAB and PROC SORT with SORTSEQ=, see Example 6 on page 1429. The available translation tables are

- Danish
- Finnish
- Italian
- Norwegian
- Spanish
- Swedish

To see how the alphanumeric characters in each language will sort, refer to Figure 39.1 on page 1096.

Figure 39.1 National Collating Sequences of Alphanumeric Characters

```

Danish:      0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅabcdefghijklmnopqrstuvwxyzæøå
Finnish:     0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÄÅöabcdefghijklmnopqrstuvwxyzääö
Italian:     0123456789AÀBCÇDEÈÊFGHIÌJJKLMNOÒPQRSTUÙVWXYZaàbcçdeèèfgghiìjklmnoòpqrstuùvwxyz
Norwegian:   0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅabcdefghijklmnopqrstuvwxyzæøå
Spanish:     0123456789AÁaáBbCcDdEéEéFfGgHhIíIíJjKkLlMmNnÑñOóOóPpQqRrSsTtUúUúVvWwXxYyZz
Swedish:     0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÄÅöabcdefghijklmnopqrstuvwxyzääö

```

SWEDISH

See FINNISH.

Other Options**DATA=SAS-data-set**

identifies the input SAS data set.

Main discussion: “Input Data Sets” on page 19**DATECOPY**

copies the SAS internal date and time when the SAS data set was created and the date and time when it was last modified prior to the sort to the resulting sorted data set. Note that the operating environment date and time are not preserved.

Restriction: DATECOPY can be used only when the resulting data set uses the V8 or V9 engine.**Tip:** You can alter the file creation date and time with the DTC= option in the MODIFY statement in PROC DATASETS. For more information, see “MODIFY Statement” on page 366.**EQUALS | NOEQUALS**

specifies the order of the observations in the output data set. For observations with identical BY-variable values, EQUALS maintains the relative order of the observations within the input data set in the output data set. NOEQUALS does not necessarily preserve this order in the output data set.

Default: EQUALS**Interaction:** When you use NODUPRECS or NODUPKEY to remove observations in the output data set, the choice of EQUALS or NOEQUALS can have an effect on which observations are removed.**Tip:** Using NOEQUALS can save CPU time and memory.**Interaction:** The EQUALS option is supported by the multi-threaded sort.

However, I/O performance may be reduced when using the EQUALS option with the multi-threaded sort because partitioned data sets will be processed as if they are non-partitioned data sets.

Interaction: The NOEQUALS option is supported by the multi-threaded sort. The order of observations within BY groups returned by the multi-threaded sort may not be consistent between runs. Therefore, using the NOEQUALS option can produce inconsistent results in your output data sets.**FORCE**

sorts and replaces an indexed data set when the OUT= option is not specified.

Without the FORCE option, PROC SORT does not sort and replace an indexed data

set because sorting destroys user-created indexes for the data set. When you specify **FORCE**, PROC SORT sorts and replaces the data set and destroys all user-created indexes for the data set. Indexes that were created or required by integrity constraints are preserved.

Tip: PROC SORT checks for the sort information before it sorts a data set so that data is not re-sorted unnecessarily. By default, PROC SORT does not sort a data set if the sort information matches the requested sort. You can use **FORCE** to override this behavior. You might need to use **FORCE** if SAS cannot verify the sort specification in the data set option **SORTEDBY=**. For more information about **SORTEDBY=**, see the chapter on SAS data set options in *SAS Language Reference: Dictionary*.

Restriction: If you use PROC SORT with the **FORCE** option on data sets that were created with the Version 5 compatibility engine or with a sequential engine such as a tape format engine, you must also specify the **OUT=** option.

NODUPKEY

checks for and eliminates observations with duplicate BY values. If you specify this option, then PROC SORT compares all BY values for each observation to those for the previous observation that is written to the output data set. If an exact match is found, then the observation is not written to the output data set.

Operating Environment Information: If you use the VMS operating environment sort, then the observation that is written to the output data set is not always the first observation of the BY group. △

Note: See **NODUPRECS** for information about eliminating duplicate observations. △

Interaction: When you are removing observations with duplicate BY values with **NODUPKEY**, the choice of **EQUALS** or **NOEQUALS** can have an effect on which observations are removed.

Tip: Use the **EQUALS** option with the **NODUPKEY** option for consistent results in your output data sets.

Featured in: Example 4 on page 1110

NODUPRECS

checks for and eliminates duplicate observations. If you specify this option, then PROC SORT compares all variable values for each observation to those for the previous observation that was written to the output data set. If an exact match is found, then the observation is not written to the output data set.

Note: See **NODUPKEY** for information about eliminating observations with duplicate BY values. △

Alias : **NODUP**

Interaction: When you are removing consecutive duplicate observations in the output data set with **NODUPRECS**, the choice of **EQUALS** or **NOEQUALS** can have an effect on which observations are removed.

Tip: Use the **EQUALS** option with the **NODUPRECS** option for consistent results in your output data sets.

Interaction: The action of **NODUPRECS** is directly related to the setting of the **SORTDUP=** system option. When **SORTDUP=** is set to **LOGICAL**, **NODUPRECS** removes duplicate observations based on the examination of the variables that remain after a **DROP** or **KEEP** operation on the input data set. Setting **SORTDUP=LOGICAL** increases the number of duplicate observations that are removed, because it eliminates variables before observation comparisons take place. Also, setting **SORTDUP=LOGICAL** can improve performance, because

dropping variables before sorting reduces the amount of memory required to perform the sort. When SORTDUP= is set to PHYSICAL, NODUPRECS examines all variables in the data set, regardless of whether they have been kept or dropped. For more information about SORTDUP=, see the chapter on SAS system options in *SAS Language Reference: Dictionary*.

Tip: Because NODUPRECS checks only consecutive observations, some nonconsecutive duplicate observations might remain in the output data set. You can remove all duplicates with this option by sorting on all variables.

NOEQUALS

See EQUALS | NOEQUALS.

NOTHREADS

See THREADS | NOTHREADS.

OUT=SAS-data-set

names the output data set. If *SAS-data-set* does not exist, then PROC SORT creates it.

CAUTION:

Use care when you use PROC SORT without OUT=. Without OUT=, data could be lost if your system failed during execution of PROC SORT. Δ

Default: Without OUT=, PROC SORT overwrites the original data set.

Tip : You can use data set options with OUT=.

Featured in: Example 1 on page 1103

REVERSE

sorts character variables using a collating sequence that is reversed from the normal collating sequence.

Operating Environment Information: For information about the normal collating sequence for your operating environment, see “EBCDIC Order” on page 1101, “ASCII Order” on page 1101, and the SAS documentation for your operating environment. Δ

Interaction: Using REVERSE with the DESCENDING option in the BY statement restores the sequence to the normal order.

Restriction: The REVERSE option cannot be used with a *collating-sequence-option*. You can specify either a *collating-sequence-option* or the REVERSE option in a PROC SORT, but you cannot specify both.

See also: The DESCENDING option in the BY statement. The difference is that the DESCENDING option can be used with both character and numeric variables.

SORTSIZE=memory-specification

specifies the maximum amount of memory that is available to PROC SORT. Valid values for *memory-specification* are as follows:

MAX

specifies that all available memory can be used.

n

specifies the amount of memory in bytes, where *n* is a real number.

*n*K

specifies the amount of memory in kilobytes, where *n* is a real number.

*n*M

specifies the amount of memory in megabytes, where *n* is a real number.

*n*G

specifies the amount of memory in gigabytes, where *n* is a real number.

Specifying the SORTSIZE= option in the PROC SORT statement temporarily overrides the SAS system option SORTSIZE=. For more information about SORTSIZE=, see the chapter on SAS system options in *SAS Language Reference: Dictionary*.

Operating Environment Information: Some system sort utilities may treat this option differently. Refer to the SAS documentation for your operating environment. △

Default: the value of the SAS system option SORTSIZE=

Tip: Setting the SORTSIZE= option in the PROC SORT statement to MAX or 0, or not setting the SORTSIZE= option, limits the PROC SORT to the available physical memory based on the settings of the SAS system options that relate to memory and information regarding available memory that is gathered from the operating environment.

Operating Environment Information: For information about the SAS system options that relate to memory, see the SAS documentation for your operating environment. △

TAGSORT

stores only the BY variables and the observation numbers in temporary files. The BY variables and the observation numbers are called *tags*. At the completion of the sorting process, PROC SORT uses the tags to retrieve records from the input data set in sorted order.

Interaction: The TAGSORT option is not supported by the multi-threaded sort.

Tip: When the total length of BY variables is small compared with the record length, TAGSORT reduces temporary disk usage considerably. However, processing time may be much higher.

THREADS | NOTTHREADS

enables or prevents the activation of multi-threaded sorting.

Default: the value of the SAS system option THREADS

Interaction: THREAD|NOTTHREADS overrides the value of the SAS system option THREADS. For more information about THREADS, see the chapter on SAS system options in *SAS Language Reference: Dictionary*.

Interaction: The TAGSORT option is not supported by the multi-threaded sort.

Note: If THREADS is specified (either as a SAS system option or in the PROC SORT statement) and another program has the input data set open for reading, writing, or updating, then PROC SORT might fail to open the input data set. In this case, PROC SORT stops processing and writes a message to the SAS log.

See also: “Multi-threaded Sorting” on page 1100

BY Statement

Specifies the sorting variables.

Featured in: Example 1 on page 1103, Example 2 on page 1105, and Example 4 on page 1110

BY <DESCENDING> *variable-1* <...> <DESCENDING> *variable-n*>;

Required Arguments

variable

specifies the variable by which PROC SORT sorts the observations. PROC SORT first arranges the data set by the values in ascending order, by default, of the first BY variable. PROC SORT then arranges any observations that have the same value of the first BY variable by the values of the second BY variable in ascending order. This sorting continues for every specified BY variable.

Option

DESCENDING

reverses the sort order for the variable that immediately follows in the statement so that observations are sorted from the largest value to the smallest value.

Featured in: Example 2 on page 1105

Concepts: SORT Procedure

Multi-threaded Sorting

The SAS system option THREADS activates multi-threaded sorting, which is new with Version 9. Multi-threaded sorting achieves a degree of parallelism in the sorting operations. This parallelism is intended to reduce the real-time to completion for a given operation at the possible cost of additional CPU resources. For more information, see the chapter on “Support for Parallel Processing” in *SAS Language Reference: Concepts*.

The performance of the multi-threaded sort will be affected by the value of the SAS system option CPUCOUNT=. CPUCOUNT= suggests how many system CPUs are available for use by the multi-threaded sort.

The multi-threaded sort supports concurrent input from the partitions of a partitioned data set.

Note: These partitioned data sets should not be confused with partitioned data sets on OS/390. △

Operating Environment Information: For information about the support of partitioned data sets in your operating environment, see the SAS documentation for your operating environment. △

For more information about THREADS and CPUCOUNT=, see the chapter on SAS system options in *SAS Language Reference: Dictionary*.

Sorting Orders for Numeric Variables

For numeric variables, the smallest-to-largest comparison sequence is

- 1 SAS missing values (shown as a period or special missing value)
- 2 negative numeric values
- 3 zero

4 positive numeric values.

Sorting Orders for Character Variables

By default PROC SORT uses either the EBCDIC or the ASCII collating sequence when it compares character values, depending on the environment under which the procedure is running.

EBCDIC Order

The OS/390 operating environment uses the EBCDIC collating sequence. The sorting order of the English-language EBCDIC sequence is

```
blank . < ( + | & ! $ * ); ~ - / , % _ > ? : # @ ' = "
a b c d e f g h i j k l m n o p q r ~ s t u v w x y z
{ A B C D E F G H I } J K L M N O P Q R \ S T
U V W X Y Z
0 1 2 3 4 5 6 7 8 9
```

The main features of the EBCDIC sequence are that lowercase letters are sorted before uppercase letters, and uppercase letters are sorted before digits. Note also that some special characters interrupt the alphabetic sequences. The blank is the smallest displayable character.

ASCII Order

The operating environments that use the ASCII collating sequence include

- ☐ UNIX and its derivatives
- ☐ OpenVMS
- ☐ Windows.

From the smallest to largest displayable character, the English-language ASCII sequence is

```
blank ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _
a b c d e f g h i j k l m n o p q r s t u v w x y z { } ~
```

The main features of the ASCII sequence are that digits are sorted before uppercase letters, and uppercase letters are sorted before lowercase letters. The blank is the smallest displayable character.

Specifying Sorting Orders for Character Variables

The options EBCDIC, ASCII, NATIONAL, DANISH, SWEDISH, and REVERSE specify collating sequences that are stored in the HOST catalog.

If you want to provide your own collating sequences or change a collating sequence provided for you, then use the TRANTAB procedure to create or modify translation tables. For complete details about the TRANTAB procedure, see Chapter 47, “The TRANTAB Procedure,” on page 1409. When you create your own translation tables, they are stored in your PROFILE catalog, and they override any translation tables that have the same name in the HOST catalog.

Note: System managers can modify the HOST catalog by copying newly created tables from the PROFILE catalog to the HOST catalog. Then all users can access the new or modified translation table. △

Stored Sort Information

PROC SORT records the BY variables, collating sequence, and character set that it uses to sort the data set. This information is stored with the data set to help avoid unnecessary sorts.

Before PROC SORT sorts a data set, it checks the stored sort information. If you try to sort a data set the way that it is currently sorted, then PROC SORT does not perform the sort and writes a message to the log to that effect. To override this behavior, use the FORCE option. If you try to sort a data set the way that it is currently sorted and you specify an OUT= data set, then PROC SORT simply makes a copy of the DATA= data set.

To override the sort information that PROC SORT stores, use the _NULL_ value with the SORTEDBY= data set option. For more information about SORTEDBY=, see the chapter on SAS data set options in *SAS Language Reference: Dictionary*.

If you want to change the sort information for an existing data set, then use the SORTEDBY= data set option in the MODIFY statement in the DATASETS procedure. For more information, see “MODIFY Statement” on page 366.

To access the sort information that is stored with a data set, use the CONTENTS statement in PROC DATASETS. For more information, see “CONTENTS Statement” on page 344.

Integrity Constraints: SORT Procedure

Sorting the input data set and replacing it with the sorted data set preserves both referential and general integrity constraints, as well as any indexes that they may require. A sort that creates a new data set will not preserve any integrity constraints or indexes. For more information about implicit replacement, explicit replacement, and no replacement with and without the OUT= option, see “Output Data Set” on page 1103. For more information about integrity constraints, see the chapter on SAS data files in *SAS Language Reference: Concepts*.

Results: SORT Procedure

Procedure Output

PROC SORT produces only an output data set. To see the output data set, you can use PROC PRINT, PROC REPORT, or another of the many available methods of printing in SAS.

Output Data Set

Without the OUT= option, PROC SORT replaces the original data set with the sorted observations when the procedure executes without errors. When you specify the OUT= option using a new data set name, PROC SORT creates a new data set that contains the sorted observations.

To do this	Use this statement
implicit replacement of input data set	proc sort data=names;
explicit replacement of input data set	proc sort data=names out=names;
no replacement of input data set	proc sort data=names out=namesbyid;

With all three replacement options (implicit replacement, explicit replacement, and no replacement) there must be at least enough space in the output data library for a copy of the original data set.

You can also sort compressed data sets. If you specify a compressed data set as the input data set and omit the OUT= option, then the input data set is sorted and remains compressed. If you specify an OUT= data set, then the resulting data set is compressed only if you choose a compression method with the COMPRESS= data set option. For more information about COMPRESS=, see the chapter on SAS data set options in *SAS Language Reference: Dictionary*.

Note: If the SAS system option NOREPLACE is in effect, then you cannot replace an original permanent data set with a sorted version. You must either use the OUT= option or specify the SAS system option REPLACE in an OPTIONS statement. The SAS system option NOREPLACE does not affect temporary SAS data sets. △

Examples: SORT Procedure

Example 1: Sorting by the Values of Multiple Variables

Procedure features:

PROC SORT statement option:

OUT=

BY statement

Other features:

PROC PRINT

This example

- ☐ sorts the observations by the values of two variables
- ☐ creates an output data set for the sorted observations
- ☐ prints the results.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the input data set ACCOUNT. ACCOUNT contains the name of each business that owes money, the amount of money that it owes on its account, the account number, and the town where the business is located.

```
data account;
    input Company $ 1-22 Debt 25-30 AccountNumber 33-36
           Town $ 39-51;
    datalines;
Paul's Pizza          83.00  1019  Apex
World Wide Electronics 119.95  1122  Garner
Strickland Industries 657.22  1675  Morrisville
Ice Cream Delight     299.98  2310  Holly Springs
Watson Tabor Travel   37.95  3131  Apex
Boyd & Sons Accounting 312.49  4762  Garner
Bob's Beds           119.95  4998  Morrisville
Tina's Pet Shop       37.95  5108  Apex
Elway Piano and Organ 65.79  5217  Garner
Tim's Burger Stand    119.95  6335  Holly Springs
Peter's Auto Parts    65.79  7288  Apex
Deluxe Hardware       467.12  8941  Garner
Pauline's Antiques    302.05  9112  Morrisville
Apex Catering         37.95  9923  Apex
;
```

Create the output data set BYTOWN. OUT= creates a new data set for the sorted observations.

```
proc sort data=account out=bytown;
```

Sort by two variables. The BY statement specifies that the observations should be first ordered alphabetically by town and then by company.

```
by town company;
run;
```

Print the output data set BYTOWN. PROC PRINT prints the data set BYTOWN.

```
proc print data=bytown;
```

Specify the variables to print. The VAR statement specifies the variables to print and their column order in the output.

```
var company town debt accountnumber;
```

Specify the titles.

```
title 'Customers with Past-Due Accounts';
title2 'Listed Alphabetically within Town';
run;
```

Output

Customers with Past-Due Accounts Listed Alphabetically within Town					1
Obs	Company	Town	Debt	Account Number	
1	Apex Catering	Apex	37.95	9923	
2	Paul's Pizza	Apex	83.00	1019	
3	Peter's Auto Parts	Apex	65.79	7288	
4	Tina's Pet Shop	Apex	37.95	5108	
5	Watson Tabor Travel	Apex	37.95	3131	
6	Boyd & Sons Accounting	Garner	312.49	4762	
7	Deluxe Hardware	Garner	467.12	8941	
8	Elway Piano and Organ	Garner	65.79	5217	
9	World Wide Electronics	Garner	119.95	1122	
10	Ice Cream Delight	Holly Springs	299.98	2310	
11	Tim's Burger Stand	Holly Springs	119.95	6335	
12	Bob's Beds	Morrisville	119.95	4998	
13	Pauline's Antiques	Morrisville	302.05	9112	
14	Strickland Industries	Morrisville	657.22	1675	

Example 2: Sorting in Descending Order

Procedure features:

This example BY statement option:

DESCENDING

Other features

PROC PRINT

Data set: ACCOUNT on page 1104

- ☐ sorts the observations by the values of three variables
- ☐ sorts one of the variables in descending order
- ☐ prints the results.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the output data set SORTED. OUT= creates a new data set for the sorted observations.

```
proc sort data=account out=sorted;
```

Sort by three variables with one in descending order. The BY statement specifies that observations should be first ordered alphabetically by town, then by descending value of amount owed, then by ascending value of the account number.

```
    by town descending debt accountnumber;
run;
```

Print the output data set SORTED. PROC PRINT prints the data set SORTED.

```
proc print data=sorted;
```

Specify the variables to print. The VAR statement specifies the variables to print and their column order in the output.

```
    var company town debt accountnumber;
```

Specify the titles.

```
    title 'Customers with Past-Due Accounts';
    title2 'Listed by Town, Amount, Account Number';
run;
```

Output

Note that sorting last by AccountNumber puts the businesses in Apex with a debt of \$37.95 in order of account number.

Customers with Past-Due Accounts Listed by Town, Amount, Account Number					1
Obs	Company	Town	Debt	Account Number	
1	Paul's Pizza	Apex	83.00	1019	
2	Peter's Auto Parts	Apex	65.79	7288	
3	Watson Tabor Travel	Apex	37.95	3131	
4	Tina's Pet Shop	Apex	37.95	5108	
5	Apex Catering	Apex	37.95	9923	
6	Deluxe Hardware	Garner	467.12	8941	
7	Boyd & Sons Accounting	Garner	312.49	4762	
8	World Wide Electronics	Garner	119.95	1122	
9	Elway Piano and Organ	Garner	65.79	5217	
10	Ice Cream Delight	Holly Springs	299.98	2310	
11	Tim's Burger Stand	Holly Springs	119.95	6335	
12	Strickland Industries	Morrisville	657.22	1675	
13	Pauline's Antiques	Morrisville	302.05	9112	
14	Bob's Beds	Morrisville	119.95	4998	

Example 3: Maintaining the Relative Order of Observations in Each BY Group

Procedure features:

PROC SORT statement option:

EQUALS|NOEQUALS

Other features: PROC PRINT

This example

- sorts the observations by the value of the first variable
- maintains the relative order with the EQUALS option
- does not maintain the relative order with the NOEQUALS option.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the input data set INSURANCE. INSURANCE contains the number of years worked by all insured employees and their insurance ids.

```
data insurance;
    input YearsWorked 1 InsuranceID 3-5;
    datalines;
5 421
5 336
1 209
1 564
3 711
3 343
4 212
4 616
;
```

Create the output data set BYYEARS1 with the EQUALS option. OUT= creates a new data set for the sorted observations. The EQUALS option maintains the order of the observations relative to each other.

```
proc sort data=insurance out=byyears1 equals;
```

Sort by the first variable. The BY statement specifies that the observations should be ordered numerically by the number of years worked.

```
    by yearsworked;
run;
```

Print the output data set BYYEARS1. PROC PRINT prints the data set BYYEARS1.

```
proc print data=byyears1;
```

Specify the variables to print. The VAR statement specifies the variables to print and their column order in the output.

```
    var yearsworked insuranceid;
```

Specify the title.

```
    title 'Sort with EQUALS';
run;
```

Create the output data set BYYEARS2. OUT= creates a new data set for the sorted observations. The NOEQUALS option will not maintain the order of the observations relative to each other.

```
proc sort data=insurance out=byyears2 noequals;
```

Sort by the first variable. The BY statement specifies that the observations should be ordered numerically by the number of years worked.

```
by yearsworked;
run;
```

Print the output data set BYYEARS2. PROC PRINT prints the data set BYYEARS2.

```
proc print data=byyears2;
```

Specify the variables to print. The VAR statement specifies the variables to print and their column order in the output.

```
var yearsworked insuranceid;
```

Specify the title.

```
title 'Sort with NOEQUALS';
run;
```

Output

Note that sorting with the EQUALS option versus sorting with the NOEQUALS option causes a different sort order for the observations where YearsWorked=3.

Sort with EQUALS			1
Obs	Years Worked	Insurance ID	
1	1	209	
2	1	564	
3	3	711	
4	3	343	
5	4	212	
6	4	616	
7	5	421	
8	5	336	
Sort with NOEQUALS			1
Obs	Years Worked	Insurance ID	
1	1	209	
2	1	564	
3	3	343	
4	3	711	
5	4	212	
6	4	616	
7	5	421	
8	5	336	

Example 4: Retaining the First Observation of Each BY Group

Procedure features:

PROC SORT statement option:

NODUPKEY

BY statement

Other features:

PROC PRINT

Data set: ACCOUNT on page 1104

Interaction: The EQUALS option, which is the default, must be in effect to ensure that the first observation for each BY group is the one that is retained by the NODUPKEY option. If the NOEQUALS option has been specified, then one observation for each BY group will still be retained by the NODUPKEY option, but not necessarily the first observation.

In this example, PROC SORT creates an output data set that contains only the first observation of each BY group. The NODUPKEY option prevents an observation from being written to the output data set when its BY value is identical to the BY value of the last observation written to the output data set. The resulting report contains one observation for each town where the businesses are located.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the output data set TOWNS but include only the first observation of each BY group. NODUPKEY writes only the first observation of each BY group to the new data set TOWNS.

Operating Environment Information: If you use the VMS operating environment sort, then the observation that is written to the output data set is not always the first observation of the BY group. \triangle

```
proc sort data=account out=towns nodupkey;
```

Sort by one variable. The BY statement specifies that observations should be ordered by town.

```
by town;
run;
```

Print the output data set TOWNS. PROC PRINT prints the data set TOWNS.


```
proc print data=towns;
```

Specify the variables to print. The VAR statement specifies the variables to print and their column order in the output.

```
var town company debt accountnumber;
```

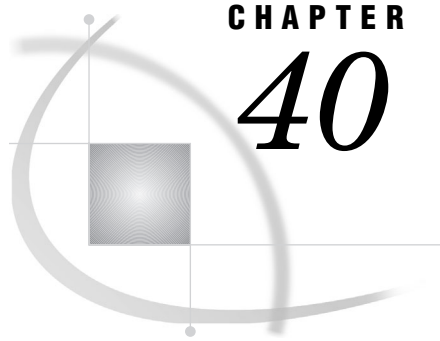
Specify the title.

```
title 'Towns of Customers with Past-Due Accounts';
run;
```

Output

The output data set contains only four observations, one for each town in the input data set.

Towns of Customers with Past-Due Accounts					1
Obs	Town	Company	Debt	Account Number	
1	Apex	Paul's Pizza	83.00	1019	
2	Garner	World Wide Electronics	119.95	1122	
3	Holly Springs	Ice Cream Delight	299.98	2310	
4	Morrisville	Strickland Industries	657.22	1675	



CHAPTER 40

The SQL Procedure

<i>Overview: SQL Procedure</i>	1115
<i>What Is the SQL Procedure?</i>	1115
<i>What Are PROC SQL Tables?</i>	1115
<i>What Are Views?</i>	1115
<i>SQL Procedure Coding Conventions</i>	1116
<i>Syntax: SQL Procedure</i>	1117
<i>PROC SQL Statement</i>	1119
<i>ALTER TABLE Statement</i>	1124
<i>CONNECT Statement</i>	1128
<i>CREATE INDEX Statement</i>	1128
<i>CREATE TABLE Statement</i>	1130
<i>CREATE VIEW Statement</i>	1133
<i>DELETE Statement</i>	1135
<i>DESCRIBE Statement</i>	1136
<i>DISCONNECT Statement</i>	1137
<i>DROP Statement</i>	1138
<i>EXECUTE Statement</i>	1139
<i>INSERT Statement</i>	1139
<i>RESET Statement</i>	1141
<i>SELECT Statement</i>	1142
<i>UPDATE Statement</i>	1153
<i>VALIDATE Statement</i>	1154
<i>SQL Procedure Component Dictionary</i>	1154
<i>BETWEEN condition</i>	1155
<i>BTRIM function</i>	1155
<i>CALCULATED</i>	1156
<i>CASE expression</i>	1157
<i>COALESCE Function</i>	1158
<i>column-definition</i>	1159
<i>column-modifier</i>	1160
<i>column-name</i>	1161
<i>CONNECTION TO</i>	1162
<i>CONTAINS condition</i>	1162
<i>EXISTS condition</i>	1163
<i>IN condition</i>	1163
<i>IS condition</i>	1164
<i>joined-table</i>	1165
<i>LIKE condition</i>	1174
<i>LOWER function</i>	1176
<i>query-expression</i>	1176
<i>sql-expression</i>	1182

<i>SUBSTRING</i> function	1189
summary-function	1190
table-expression	1196
<i>UPPER</i> function	1197
Concepts: <i>SQL Procedure</i>	1197
Using SAS Data Set Options with <i>PROC SQL</i>	1197
Connecting to a DBMS Using the <i>SQL Procedure Pass-Through Facility</i>	1198
Return Codes	1198
Connecting to a DBMS Using the <i>LIBNAME</i> Statement	1198
Using the <i>DICTIONARY</i> Tables	1199
What Are <i>DICTIONARY</i> Tables?	1199
Retrieving Information about <i>DICTIONARY</i> Tables and <i>SASHELP</i> Views	1200
Using <i>DICTIONARY</i> Tables	1201
<i>DICTIONARY</i> Tables and Performance	1201
Using Macro Variables Set by <i>PROC SQL</i>	1202
Updating <i>PROC SQL</i> and <i>SAS/ACCESS</i> Views	1203
<i>PROC SQL</i> and the ANSI Standard	1204
<i>SQL Procedure Enhancements</i>	1204
Reserved Words	1204
Column Modifiers	1205
Alternate Collating Sequences	1205
<i>ORDER BY</i> Clause in a View Definition	1205
In-Line Views	1205
Outer Joins	1205
Arithmetic Operators	1205
Orthogonal Expressions	1205
Set Operators	1206
Statistical Functions	1206
<i>SAS DATA</i> Step Functions	1206
<i>SQL Procedure Omissions</i>	1206
<i>COMMIT</i> Statement	1206
<i>ROLLBACK</i> Statement	1206
Identifiers and Naming Conventions	1206
Granting User Privileges	1207
Three-Valued Logic	1207
Embedded <i>SQL</i>	1207
Examples: <i>SQL Procedure</i>	1207
Example 1: Creating a Table and Inserting Data into It	1207
Example 2: Creating a Table from a Query's Result	1209
Example 3: Updating Data in a <i>PROC SQL</i> Table	1211
Example 4: Joining Two Tables	1213
Example 5: Combining Two Tables	1216
Example 6: Reporting from <i>DICTIONARY</i> Tables	1218
Example 7: Performing an Outer Join	1220
Example 8: Creating a View from a Query's Result	1224
Example 9: Joining Three Tables	1227
Example 10: Querying an In-Line View	1230
Example 11: Retrieving Values with the <i>SOUNDS-LIKE</i> Operator	1231
Example 12: Joining Two Tables and Calculating a New Value	1233
Example 13: Producing All the Possible Combinations of the Values in a Column	1235
Example 14: Matching Case Rows and Control Rows	1238
Example 15: Counting Missing Values with a SAS Macro	1240

Overview: SQL Procedure

What Is the SQL Procedure?

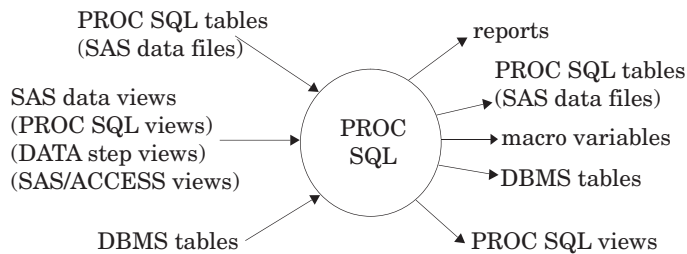
The SQL procedure implements Structured Query Language (SQL) for SAS. SQL is a standardized, widely used language that retrieves data from and updates data in tables and the views that are based on those tables.

The SAS SQL procedure enables you to

- retrieve and manipulate data that is stored in tables or views.
- create tables, views, and indexes on columns in tables.
- create SAS macro variables that contain values from rows in a query's result.
- add or modify the data values in a table's columns or insert and delete rows. You can also modify the table itself by adding, modifying, or dropping columns.
- send DBMS-specific SQL statements to a database management system (DBMS) and retrieve DBMS data.

The following figure summarizes the variety of source material that you can use with PROC SQL and what the procedure can produce.

Figure 40.1 PROC SQL Input and Output



What Are PROC SQL Tables?

A PROC SQL *table* is synonymous with a SAS data file and has a member type of DATA. You can use PROC SQL tables as input into DATA steps and procedures.

You create PROC SQL tables from SAS data files, from SAS data views, or from DBMS tables by using PROC SQL's Pass-Through Facility or the SAS/ACCESS LIBNAME statement. The Pass-Through Facility is described in "Connecting to a DBMS Using the SQL Procedure Pass-Through Facility" on page 1198. The SAS/ACCESS LIBNAME statement is described in "Connecting to a DBMS Using the LIBNAME Statement" on page 1198.

In PROC SQL terminology, a *row* in a table is the same as an *observation* in a SAS data file. A *column* is the same as a *variable*.

What Are Views?

A SAS *data view* defines a virtual data set that is named and stored for later use. A view contains no data but describes or defines data that is stored elsewhere. There are three types of SAS data views:

- ☐ PROC SQL views
- ☐ SAS/ACCESS views
- ☐ DATA step views.

You can refer to views in queries as if they were tables. The view derives its data from the tables or views that are listed in its FROM clause. The data that is accessed by a view is a subset or superset of the data that is in its underlying table(s) or view(s).

A PROC SQL view is a SAS data set of type VIEW that is created by PROC SQL. A PROC SQL view contains no data. It is a stored query expression that reads data values from its underlying files, which can include SAS data files, SAS/ACCESS views, DATA step views, other PROC SQL views, or DBMS data. When executed, a PROC SQL view's output can be a subset or superset of one or more underlying files.

SAS/ACCESS views and DATA step views are similar to PROC SQL views in that they are both stored programs of member type VIEW. SAS/ACCESS views describe data in DBMS tables from other software vendors. DATA step views are stored DATA step programs.

Note: Starting in Version 9, PROC SQL views, the Pass-Through Facility, and the SAS/ACCESS LIBNAME statement are the preferred ways to access relational DBMS data; SAS/ACCESS views are no longer recommended. You can convert existing SAS/ACCESS views to PROC SQL views by using the CV2VIEW procedure. See The CV2VIEW Procedure in *SAS/ACCESS for Relational Databases: Reference* for more information. \triangle

You can update data through a PROC SQL or SAS/ACCESS view with certain restrictions. See “Updating PROC SQL and SAS/ACCESS Views” on page 1203.

You can use all types of views as input to DATA steps and procedures.

Note: In this chapter, the term *view* collectively refers to PROC SQL views, DATA step views, and SAS/ACCESS views, unless otherwise noted. \triangle

Note: When the contents of an SQL view are processed (by a DATA step or a procedure), the referenced data set must be opened to retrieve information about the variables that is not stored in the view. If that data set has a libref associated with it that is not defined in the current SAS code, then an error will result. \triangle

SQL Procedure Coding Conventions

Because PROC SQL implements Structured Query Language, it works somewhat differently from other base SAS procedures, as described here:

- ☐ When a PROC SQL statement is executed, PROC SQL continues to run until a QUIT statement, a DATA step, or another SAS procedure is executed. Therefore, you do not need to repeat the PROC SQL statement with each SQL statement. You need to repeat the PROC SQL statement only if you execute a QUIT statement, a DATA step, or another SAS procedure between SQL statements.
- ☐ SQL procedure statements are divided into clauses. For example, the most basic SELECT statement contains the SELECT and FROM clauses. Items within clauses are separated with commas in SQL, not with blanks as in other SAS code. For example, if you list three columns in the SELECT clause, then the columns are separated with commas.
- ☐ The SELECT statement, which is used to retrieve data, also automatically writes the output data to the Output window unless you specify the NOPRINT option in the PROC SQL statement. Therefore, you can display your output or send it to a list file without specifying the PRINT procedure.

- The ORDER BY clause sorts data by columns. In addition, tables do not need to be presorted by a variable for use with PROC SQL. Therefore, you do not need to use the SORT procedure with your PROC SQL programs.
- A PROC SQL statement runs when you submit it; you do not have to specify a RUN statement. If you follow a PROC SQL statement with a RUN statement, then SAS ignores the RUN statement and submits the statements as usual.

Syntax: SQL Procedure

Tip: Supports the Output Delivery System. See “Output Delivery System” on page 32 for details.

Reminder: You can use any global statements. See Chapter 2, “Fundamental Concepts for Using Base SAS Procedures,” on page 15 for a list.

Reminder: You can use data set options any time a table name or view name is specified. See “Using SAS Data Set Options with PROC SQL” on page 1197 for details.

Note:

Regular type indicates the name of a component that is described in “SQL Procedure Component Dictionary” on page 1154.

view-name indicates a SAS data view of any type.

PROC SQL *<option(s)>*;

ALTER TABLE *table-name*

<ADD <CONSTRAINT> constraint-clause<, ... constraint-clause>>

<ADD column-definition<, ... column-definition>>

<DROP CONSTRAINT constraint-name <, ... constraint-name>>

<DROP column<, ... column>>

<DROP FOREIGN KEY constraint-name> [Note: This is a DB2 extension.]

<DROP PRIMARY KEY> [Note: This is a DB2 extension.]

<MODIFY column-definition<, ... column-definition>>

;

CREATE <UNIQUE> INDEX *index-name*

ON table-name (column <, ... column>);

CREATE TABLE *table-name*

(column-specification<, ...column-specification | constraint-specification>)

;

CREATE TABLE *table-name* **LIKE** *table-name2*;

CREATE TABLE *table-name* **AS** *query-expression*

<ORDER BY order-by-item<, ... order-by-item>>;

CREATE VIEW *proc-sql-view* **AS** *query-expression*

<ORDER BY order-by-item<, ... order-by-item>>

<USING libname-clause<, ... libname-clause>> ;

DELETE

FROM table-name | proc-sql-view | sas/access-view <AS alias>

<WHERE sql-expression>;

DESCRIBE TABLE *table-name <, ... table-name>;*

DESCRIBE VIEW *proc-sql-view <, ... proc-sql-view>;*

```

DESCRIBE TABLE CONSTRAINTS table-name <, ... table-name>;
DROP INDEX index-name <, ... index-name>
FROM table-name;
DROP TABLE table-name <, ... table-name>;
DROP VIEW view-name <, ... view-name>;
INSERT INTO table-name | sas/access-view | proc-sql-view <(column<, ... column>)>
SET column=sql-expression
    <, ... column=sql-expression>
    <SET column=sql-expression
    <, ... column=sql-expression>>;
INSERT INTO table-name | sas/access-view | proc-sql-view <(column<, ... column>)>
VALUES (value <, ... value>)
    <... VALUES (value <, ... value>)>;
INSERT INTO table-name | sas/access-view | proc-sql-view
    <(column<, ...column>)> query-expression;
RESET <option(s)>;
SELECT <DISTINCT> object-item <, ...object-item>
    <INTO macro-variable-specification
    <, ... macro-variable-specification>>
FROM from-list
    <WHERE sql-expression>
    <GROUP BY group-by-item
    <, ... group-by-item>>
    <HAVING sql-expression>
    <ORDER BY order-by-item
    <, ... order-by-item>>;
UPDATE table-name | sas/access-view | proc-sql-view <AS alias>
SET column=sql-expression
    <, ... column=sql-expression>
    <SET column=sql-expression
    <, ... column=sql-expression>>
    <WHERE sql-expression>;
VALIDATE query-expression;

```

To connect to a DBMS and send it a DBMS-specific nonquery SQL statement, use this form:

```

PROC SQL;
CONNECT TO dbms-name <AS alias>
    <(connect-statement-argument-1=value <...
    connect-statement-argument-n=value>)>
    <(dbms-argument-1=value <... dbms-argument-n=value>)>;
EXECUTE (dbms-SQL-statement)
BY dbms-name | alias;
<DISCONNECT FROM dbms-name | alias>;
<QUIT>;

```

To connect to a DBMS and query the DBMS data, use this form:

PROC SQL;

CONNECT TO *dbms-name* <AS *alias*>
 <(connect-statement-argument-1=value <...
 connect-statement-argument-n=value>)>
 <(dbms-argument-1=value <... dbms-argument-n=value>)>;

SELECT *column-list*
FROM CONNECTION TO *dbms-name* | *alias*
 (*dbms-query*)
 optional *PROC SQL* clauses;

<**DISCONNECT FROM** *dbms-name* | *alias*;>
 <**QUIT**;>

To do this	Use this statement
Modify, add, or drop columns	ALTER TABLE
Establish a connection with a DBMS	CONNECT TO
Create an index on a column	CREATE INDEX
Create a PROC SQL table	CREATE TABLE
Create a PROC SQL view	CREATE VIEW
Delete rows	DELETE
Display a definition of a table or view	DESCRIBE
Terminate the connection with a DBMS	DISCONNECT FROM
Delete tables, views, or indexes	DROP
Send a DBMS-specific nonquery SQL statement to a DBMS	EXECUTE
Add rows	INSERT
Reset options that affect the procedure environment without restarting the procedure	RESET
Select and execute rows	SELECT
Query a DBMS	CONNECTION TO
Modify values	UPDATE
Verify the accuracy of your query	VALIDATE

PROC SQL Statement

PROC SQL <*option(s)*>;

To do this	Use this option
Control output	
Double-space the report	DOUBLE NODOUBLE
Write a statement to the SAS log that expands the query	FEEDBACK NOFEEDBACK
Flow characters within a column	FLOW NOFLOW
Include a column of row numbers	NUMBER NONUMBER
Specify whether PROC SQL prints the query's result	PRINT NOPRINT
Specify whether PROC SQL should display sorting information	SORTMSG NOSORTMSG
Specify a collating sequence	SORTSEQ=
Control execution	
Specify the number of INSERT or UPDATE statements that can be bundled together and executed at once	CACHED_UPDATES=
Allow PROC SQL to use names other than SAS names	DQUOTE=
Specify whether PROC SQL should stop executing after an error	ERRORSTOP NOERRORSTOP
Specify whether PROC SQL should execute statements	EXEC NOEXEC
Restrict the number of input rows	INOBS=
Restrict the number of output rows	OUTOBS=
Restrict the number of loops	LOOPS=
Specify whether PROC SQL prompts you when a limit is reached with the INOBS=, OUTOBS=, or LOOPS= options	PROMPT NOPROMPT
Specify whether PROC SQL writes timing information for each statement to the SAS log	STIMER NOSTIMER
Override the SAS system option THREADS NOTHEADS	THREADS NOTHEADS
Specify how PROC SQL handles updates when there is an interruption	UNDO_POLICY=

Options

CACHED_UPDATES=*n*

Specifies that *n* INSERT or UPDATE statements, operating on a single table from a single PROC SQL step, can be bundled together and executed at one time. For PROC SQL steps that have many INSERT or UPDATE statements, setting CACHED_UPDATES= can enhance performance.

In order to qualify to be included in a single bundle, the statements

- must be either all INSERT statements or all UPDATE statements
- must appear consecutively, with no other statements among them
- must have the same target table with the same set of data set option values.

If fewer than n consecutive statements qualify for a single bundle, those statements are executed at one time after the end of the last qualifying INSERT or UPDATE statement.

Default: 1

Tip: While the maximum value is limited only by the largest integer that your operating environment allows, the optimal value is typically between 20 and 30.

Tip: The CACHED_UPDATES= option is useful mainly for processes that generate PROC SQL code with hundreds or thousands of INSERT or UPDATE statements.

DOUBLE|NODOUBLE

double-spaces the report.

Default: NODOUBLE

Featured in: Example 5 on page 1216

DQUOTE=ANSI|SAS

specifies whether PROC SQL treats values within double quotation marks (" ") as variables or strings. With DQUOTE=ANSI, PROC SQL treats a quoted value as a variable. This feature enables you to use the following as table names, column names, or aliases:

- reserved words such as AS, JOIN, GROUP, and so on
- DBMS names and other names that are not normally permissible in SAS.

The quoted value can contain any character.

With DQUOTE=SAS, values within double quotation marks are treated as strings.

Default: SAS

ERRORSTOP|NOERRORSTOP

specifies whether PROC SQL stops executing if it encounters an error. In a batch or noninteractive session, ERRORSTOP instructs PROC SQL to stop executing the statements but to continue checking the syntax after it has encountered an error.

NOERRORSTOP instructs PROC SQL to execute the statements and to continue checking the syntax after an error occurs.

Default: NOERRORSTOP in an interactive SAS session; ERRORSTOP in a batch or noninteractive session

Interaction: This option is useful only when the EXEC option is in effect.

Tip: ERRORSTOP has an effect only when SAS is running in the batch or noninteractive execution mode.

Tip: NOERRORSTOP is useful if you want a batch job to continue executing SQL procedure statements after an error is encountered.

EXEC|NOEXEC

specifies whether a statement should be executed after its syntax is checked for accuracy.

Default: EXEC

Tip: NOEXEC is useful if you want to check the syntax of your SQL statements without executing the statements.

See also: ERRORSTOP on page 1121

FEEDBACK|NOFEEDBACK

specifies whether PROC SQL displays, in the SAS log, PROC SQL statements after view references are expanded or certain other transformations of the statement are made.

This option has the following effects:

- ☐ Any asterisk (for example, **SELECT ***) is expanded into the list of qualified columns that it represents.
- ☐ Any PROC SQL view is expanded into the underlying query.
- ☐ Macro variables are resolved.
- ☐ Parentheses are shown around all expressions to further indicate their order of evaluation.
- ☐ Comments are removed.

Default: NOFEEDBACK

FLOW=<n <m>>|NOFLOW

specifies that character columns longer than *n* are flowed to multiple lines. PROC SQL sets the column width at *n* and specifies that character columns longer than *n* are flowed to multiple lines. When you specify FLOW=*n m*, PROC SQL floats the width of the columns between these limits to achieve a balanced layout. Specifying FLOW without arguments is equivalent to specifying FLOW=12 200.

Default: NOFLOW

INOBS=*n*

restricts the number of rows (observations) that PROC SQL retrieves from any single source.

Tip: This option is useful for debugging queries on large tables.

LOOPS=*n*

restricts PROC SQL to *n* iterations through its inner loop. You use the number of iterations reported in the SQLLOOPS macro variable (after each SQL statement is executed) to discover the number of loops. Set a limit to prevent queries from consuming excessive computer resources. For example, joining three large tables without meeting the join-matching conditions could create a huge internal table that would be inefficient to execute.

See also: “Using Macro Variables Set by PROC SQL” on page 1202

NODOUBLE

See DOUBLE|NODOUBLE on page 1121.

NOERRORSTOP

See ERRORSTOP|NOERRORSTOP on page 1121.

NOEXEC

See EXEC|NOEXEC on page 1121.

NOFEEDBACK

See FEEDBACK|NOFEEDBACK on page 1121.

NOFLOW

See FLOW|NOFLOW on page 1122.

NONUMBER

See NUMBER|NONUMBER on page 1123.

NOPRINT

See PRINT|NOPRINT on page 1123.

NOPROMPT

See PROMPT|NOPROMPT on page 1123.

NOSORTMSG

See SORTMSG|NOSORTMSG on page 1123.

NOSTIMER

See STIMER|NOSTIMER on page 1123.

NOTHREADS

See THREADS|NOTHREADS.

NUMBER|NONUMBER

specifies whether the SELECT statement includes a column called ROW, which is the row (or observation) number of the data as the rows are retrieved.

Default: NONUMBER

Featured in: Example 4 on page 1213

OUTOBS=*n*

restricts the number of rows (observations) in the output. For example, if you specify OUTOBS=10 and insert values into a table using a query-expression, then the SQL procedure inserts a maximum of 10 rows. Likewise, OUTOBS=10 limits the output to 10 rows.

PRINT|NOPRINT

specifies whether the output from a SELECT statement is printed.

Default: PRINT

Tip: NOPRINT is useful when you are selecting values from a table into macro variables and do not want anything to be displayed.

PROMPT|NOPROMPT

modifies the effect of the INOBS=, OUTOBS=, and LOOPS= options. If you specify the PROMPT option and reach the limit specified by INOBS=, OUTOBS=, or LOOPS=, then PROC SQL prompts you to stop or continue. The prompting repeats if the same limit is reached again.

Default: NOPROMPT

SORTMSG|NOSORTMSG

Certain operations, such as ORDER BY, may sort tables internally using PROC SORT. Specifying SORTMSG requests information from PROC SORT about the sort and displays the information in the log.

Default: NOSORTMSG

SORTSEQ=*sort-table*

specifies the collating sequence to use when a query contains an ORDER BY clause. Use this option only if you want a collating sequence other than your system's or installation's default collating sequence.

See also: SORTSEQ= option in *SAS Language Reference: Dictionary*.

STIMER|NOSTIMER

specifies whether PROC SQL writes timing information to the SAS log for each statement, rather than as a cumulative value for the entire procedure. For this option to work, you must also specify the SAS system option STIMER. Some operating environments require that you specify this system option when you invoke SAS. If you use the system option alone, then you receive timing information for the entire SQL procedure, not on a statement-by-statement basis.

Default: NOSTIMER

THREADS|NOTHREADS

overrides the SAS system option THREADS|NOTHREADS for a particular invocation of PROC SQL. THREADS|NOTHREADS can also be specified in a

RESET statement for use in particular queries. When THREADS is specified, PROC SQL uses parallel processing in order to increase the performance of sorting operations that involve large amounts of data. For more information about parallel processing, see *SAS Language Reference: Concepts*.

Default: value of SAS system option THREADS|NOTTHREADS.

Note: When THREADS|NOTTHREADS has been specified in a PROC SQL statement or a RESET statement, there is no way to reset the option to its default (that is, the value of the SAS system option THREADS|NOTTHREADS) for that invocation of PROC SQL. Δ

UNDO_POLICY=NONE|OPTIONAL|REQUIRED

specifies how PROC SQL handles updated data if errors occur while you are updating data. You can use UNDO_POLICY= to control whether your changes will be permanent:

NONE

keeps any updates or inserts.

OPTIONAL

reverses any updates or inserts that it can reverse reliably.

REQUIRED

reverses all inserts or updates that have been done to the point of the error. In some cases, the UNDO operation cannot be done reliably. For example, when a program uses a SAS/ACCESS view, it may not be able to reverse the effects of the INSERT and UPDATE statements without reversing the effects of other changes at the same time. In that case, PROC SQL issues an error message and does not execute the statement. Also, when a SAS data set is accessed through a SAS/SHARE server and is opened with the data set option CNTLLEV=RECORD, you cannot reliably reverse your changes.

This option may enable other users to update newly inserted rows. If an error occurs during the insert, then PROC SQL can delete a record that another user updated. In that case, the statement is not executed, and an error message is issued.

Default: REQUIRED

Note: Options can be added, removed, or changed between PROC SQL statements with the RESET statement. Δ

ALTER TABLE Statement

Adds columns to, drops columns from, and changes column attributes in an existing table. Adds, modifies, and drops integrity constraints from an existing table.

Restriction: You cannot use any type of view in an ALTER TABLE statement.

Restriction: You cannot use ALTER TABLE on a table that is accessed by an engine that does not support UPDATE processing.

Restriction: You must use at least one ADD, DROP, or MODIFY clause in the ALTER TABLE statement.

Featured in: Example 3 on page 1211

ALTER TABLE *table-name*

```

<ADD <CONSTRAINT> constraint-clause<, ... constraint-clause>>
<ADD column-definition<, ... column-definition>>
<DROP CONSTRAINT constraint-name <, ... constraint-name>>
<DROP column<, ... column>>
<DROP FOREIGN KEY constraint-name>
<DROP PRIMARY KEY>
<MODIFY column-definition<, ... column-definition>>
;

```

Arguments

<ADD <CONSTRAINT> *constraint-clause*<, ... *constraint-clause*>>
 adds the integrity constraint that is specified in *constraint-clause*.

<ADD *column-definition*<, ... *column-definition*>>
 adds the column(s) that are specified in each *column-definition*.

column
 names a column in *table-name*.

column-definition
 See “column-definition” on page 1159.

constraint
 is one of the following integrity constraints:

CHECK (*WHERE-clause*)
 specifies that all rows in *table-name* satisfy the *WHERE-clause*.

DISTINCT (*column*<, ... *column*>)
 specifies that the values of each *column* must be unique. This constraint is identical to UNIQUE.

FOREIGN KEY (*column*<, ... *column*>)
REFERENCES *table-name*
<ON DELETE *referential-action*> <ON UPDATE *referential-action*>
 specifies a foreign key, that is, a set of *columns* whose values are linked to the values of the primary key variable in another table (the *table-name* that is specified for REFERENCES). The *referential-actions* are performed when the values of a primary key column that is referenced by the foreign key are updated or deleted.

NOT NULL (*column*)
 specifies that *column* does not contain a null or missing value, including special missing values.

PRIMARY KEY (*column*<, ... *column*>)
 specifies one or more primary key columns, that is, columns that do not contain missing values and whose values are unique.

UNIQUE (*column*<, ... *column*>)
 specifies that the values of each *column* must be unique. This constraint is identical to DISTINCT.

constraint-clause
 consists of

constraint-name constraint <MESSAGE='message-string'
<MSGTYPE=message-type>>

constraint-name

specifies a name for the constraint that is being specified. The name must be a valid SAS name.

Note: The names PRIMARY, FOREIGN, MESSAGE, UNIQUE, DISTINCT, CHECK, and NOT cannot be used as values for *constraint-name*. △

<DROP column<, ... column>>

deletes each *column* from the table.

<DROP CONSTRAINT *constraint-name*<, ...> (*constraint-name*)>>

deletes the integrity constraint that is referenced by each *constraint-name*. To find the name of an integrity constraint, use the DESCRIBE TABLE CONSTRAINTS clause (see “DESCRIBE Statement” on page 1136).

<DROP FOREIGN KEY *constraint-name*>

Removes the foreign key constraint that is referenced by *constraint-name*.

Note: The DROP FOREIGN KEY clause is a DB2 extension. △

<DROP PRIMARY KEY>

Removes the primary key constraint from *table-name*.

Note: The DROP PRIMARY KEY clause is a DB2 extension. △

message-string

specifies the text of an error message that is written to the log when the integrity constraint is not met. The maximum length of *message-string* is 250 characters.

message-type

specifies how the error message is displayed in the SAS log when an integrity constraint is not met.

NEWLINE

the text that is specified for MESSAGE= is displayed as well as the default error message for that integrity constraint.

USER

only the text that is specified for MESSAGE= is displayed.

<MODIFY column-definition<, ... column-definition>>

changes one or more attributes of the column that is specified in each column-definition.

referential-action

specifies the type of action to be performed on all matching foreign key values.

CASCADE

allows primary key data values to be updated, and updates matching values in the foreign key to the same values. This referential action is currently supported for updates only.

RESTRICT

prevents the update or deletion of primary key data values if a matching value exists in the foreign key. This referential action is the default.

SET NULL

allows primary key data values to be updated, and sets all matching foreign key values to NULL.

table-name

- in the ALTER TABLE statement, refers to the name of the table that is to be altered.
- in the REFERENCES clause, refers to the name of table that contains the primary key that is referenced by the foreign key.

table-name can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

WHERE-clause

specifies a SAS WHERE clause. Do not include the WHERE keyword in the WHERE clause.

Specifying Initial Values of New Columns

When the ALTER TABLE statement adds a column to the table, it initializes the column's values to missing in all rows of the table. Use the UPDATE statement to add values to the new column(s).

Changing Column Attributes

If a column is already in the table, then you can change the following column attributes by using the MODIFY clause: length, informat, format, and label. The values in a table are either truncated or padded with blanks (if character data) as necessary to meet the specified length attribute.

You cannot change a character column to numeric and vice versa. To change a column's data type, drop the column and then add it (and its data) again, or use the DATA step.

Note: You cannot change the length of a numeric column with the ALTER TABLE statement. Use the DATA step instead. Δ

Renaming Columns

To change a column's name, you must use the SAS data set option RENAME=. You cannot change this attribute with the ALTER TABLE statement. RENAME= is described in the section on SAS data set options in *SAS Language Reference: Dictionary*.

Indexes on Altered Columns

When you alter the attributes of a column and an index has been defined for that column, the values in the altered column continue to have the index defined for them. If you drop a column with the ALTER TABLE statement, then all the indexes (simple and composite) in which the column participates are also dropped. See "CREATE INDEX Statement" on page 1128 for more information about creating and using indexes.

Integrity Constraints

Use ALTER TABLE to modify integrity constraints for existing tables. Use the CREATE TABLE statement to attach integrity constraints to new tables. For more information on integrity constraints, see the section on SAS files in *SAS Language Reference: Concepts*.

CONNECT Statement

Establishes a connection with a DBMS that is supported by SAS/ACCESS software.

Requirement: SAS/ACCESS software is required. For more information about this statement, refer to your SAS/ACCESS documentation.

See also: “Connecting to a DBMS Using the SQL Procedure Pass-Through Facility” on page 1198

```
CONNECT TO dbms-name <AS alias>
    <(connect-statement-argument-1=value <...
    connect-statement-argument-n=value>)>
    <(dbms-argument-1=value <... dbms-argument-n=value>)>;
```

Arguments

alias

specifies an alias that has 1 to 32 characters. The keyword AS must precede *alias*. Some DBMSs allow more than one connection. The optional AS clause enables you to name the connections so that you can refer to them later.

connect-statement-argument=value

specifies values for arguments that indicate whether you can make multiple connections, shared or unique connections, and so on, to the database. These arguments are optional, but if they are included, then they must be enclosed in parentheses. See *SAS/ACCESS for Relational Databases: Reference* for more information about these arguments.

database-connection-argument=value

specifies values for the DBMS-specific arguments that are needed by PROC SQL in order to connect to the DBMS. These arguments are optional for most databases, but if they are included, then they must be enclosed in parentheses. For more information, see the SAS/ACCESS documentation for your DBMS.

dbms-name

identifies the DBMS that you want to connect to (for example, ORACLE or DB2).

CREATE INDEX Statement

Creates indexes on columns in tables.

Restriction: You cannot use CREATE INDEX on a table that is accessed with an engine that does not support UPDATE processing.

```
CREATE <UNIQUE> INDEX index-name
    ON table-name ( column <, ... column>);
```

Arguments

column

specifies a column in *table-name*.

index-name

names the index that you are creating. If you are creating an index on one column only, then *index-name* must be the same as *column*. If you are creating an index on more than one column, then *index-name* cannot be the same as any column in the table.

table-name

specifies a PROC SQL table.

Indexes in PROC SQL

An *index* stores both the values of a table's columns and a system of directions that enable access to rows in that table by index value. Defining an index on a column or set of columns enables SAS, under certain circumstances, to locate rows in a table more quickly and efficiently. Indexes enable PROC SQL to execute the following classes of queries more efficiently:

- comparisons against a column that is indexed
- an IN subquery where the column in the inner subquery is indexed
- correlated subqueries, where the column being compared with the correlated reference is indexed
- join-queries, where the join-expression is an equals comparison and all the columns in the join-expression are indexed in one of the tables being joined.

SAS maintains indexes for all changes to the table, whether the changes originate from PROC SQL or from some other source. Therefore, if you alter a column's definition or update its values, then the same index continues to be defined for it. However, if an indexed column in a table is dropped, then the index on it is also dropped.

You can create simple or composite indexes. A *simple index* is created on one column in a table. A simple index must have the same name as that column. A *composite index* is one index name that is defined for two or more columns. The columns can be specified in any order, and they can have different data types. A composite index name cannot match the name of any column in the table. If you drop a composite index, then the index is dropped for all the columns named in that composite index.

UNIQUE Keyword

The UNIQUE keyword causes SAS to reject any change to a table that would cause more than one row to have the same index value. Unique indexes guarantee that data in one column, or in a composite group of columns, remain unique for every row in a table. For this reason, a unique index cannot be defined for a column that includes NULL or missing values.

Managing Indexes

You can use the CONTENTS statement in the DATASETS procedure to display a table's index names and the columns for which they are defined. You can also use the DICTIONARY tables INDEXES, TABLES, and COLUMNS to list information about indexes. For more information, see "Using the DICTIONARY Tables" on page 1199.

See the section on SAS files in *SAS Language Reference: Dictionary* for a further description of when to use indexes and how they affect SAS statements that handle BY-group processing.

CREATE TABLE Statement

Creates PROC SQL tables.

Featured in: Example 1 on page 1207 and Example 2 on page 1209

- ❶ **CREATE TABLE** *table-name*
 (*column-specification*<, ...*column-specification* | *constraint-specification*>)
 ;
- ❷ **CREATE TABLE** *table-name* **LIKE** *table-name2*;
- ❸ **CREATE TABLE** *table-name* **AS** *query-expression*
 <**ORDER BY** *order-by-item*<, ... *order-by-item*>>;

Arguments

column-constraint

is one of the following:

CHECK (*WHERE-clause*)

specifies that all rows in *table-name* satisfy the *WHERE-clause*.

DISTINCT

specifies that the values of the column must be unique. This constraint is identical to **UNIQUE**.

NOT NULL

specifies that the column does not contain a null or missing value, including special missing values.

PRIMARY KEY

specifies that the column is a primary key column, that is, a column that does not contain missing values and whose values are unique.

REFERENCES *table-name*

 <**ON DELETE** *referential-action*> <**ON UPDATE** *referential-action*>

specifies that the column is a foreign key, that is, a column whose values are linked to the values of the primary key variable in another table (the *table-name* that is specified for **REFERENCES**). The *referential-actions* are performed when the values of a primary key column that is referenced by the foreign key are updated or deleted.

UNIQUE

specifies that the values of the column must be unique. This constraint is identical to **DISTINCT**.

Note: If you specify *column-constraint*, then SAS automatically assigns a name to the constraint. The constraint name has the form

Default name	Constraint type
CKxxxx	Check
FKxxxx	Foreign key
NMxxxx	Not Null
PKxxxx	Primary key
UNxxxx	Unique

where *xxxx* is a counter that begins at 0001.

△

column-definition

See “column-definition” on page 1159.

column-specification

consists of

column-definition *<column-constraint>*

constraint

is one of the following:

CHECK (*WHERE-clause*)

specifies that all rows in *table-name* satisfy the *WHERE-clause*.

DISTINCT (*column<, ... column>*)

specifies that the values of each *column* must be unique. This constraint is identical to UNIQUE.

FOREIGN KEY (*<column<, ... column>*)

REFERENCES *table-name*

<ON DELETE referential-action> <ON UPDATE referential-action>

specifies a foreign key, that is, a set of *columns* whose values are linked to the values of the primary key variable in another table (the *table-name* that is specified for REFERENCES). The *referential-actions* are performed when the values of a primary key column that is referenced by the foreign key are updated or deleted.

NOT NULL (*column*)

specifies that *column* does not contain a null or missing value, including special missing values.

PRIMARY KEY (*column<, ... column>*)

specifies one or more primary key columns, that is, columns that do not contain missing values and whose values are unique.

UNIQUE (*column<, ... column>*)

specifies that the values of each *column* must be unique. This constraint is identical to DISTINCT.

constraint-name

specifies a name for the constraint that is being specified. The name must be a valid SAS name.

Note: The names PRIMARY, FOREIGN, MESSAGE, UNIQUE, DISTINCT, CHECK, and NOT cannot be used as values for *constraint-name*. △

constraint-specification

consists of

CONSTRAINT *constraint-name constraint* <MESSAGE='message-string'
<MSGTYPE=message-type>>

message-string

specifies the text of an error message that is written to the log when the integrity constraint is not met. The maximum length of *message-string* is 250 characters.

message-type

specifies how the error message is displayed in the SAS log when an integrity constraint is not met.

NEWLINE

the text that is specified for MESSAGE= is displayed as well as the default error message for that integrity constraint.

USER

only the text that is specified for MESSAGE= is displayed.

ORDER BY order-by-item

sorts the rows in *table-name* by the values of each *order-by-item*. See ORDER BY Clause on page 1151.

query-expression

creates *table-name* from the results of a query. See “query-expression” on page 1176.

referential-action

specifies the type of action to be performed on all matching foreign key values.

CASCADE

allows primary key data values to be updated, and updates matching values in the foreign key to the same values. This referential action is currently supported for updates only.

RESTRICT

occurs only if there are matching foreign key values. This referential action is the default.

SET NULL

sets all matching foreign key values to NULL.

table-name

- in the CREATE TABLE statement, refers to the name of the table that is to be created. You can use data set options by placing them in parentheses immediately after *table-name*. See “Using SAS Data Set Options with PROC SQL” on page 1197 for details.
- in the REFERENCES clause, refers to the name of table that contains the primary key that is referenced by the foreign key.

table-name2

creates *table-name* with the same column names and column attributes as *table-name2*, but with no rows.

WHERE-clause

specifies a SAS WHERE clause. Do not include the WHERE keyword in the WHERE clause.

Creating a Table without Rows

- ❶ The first form of the CREATE TABLE statement creates tables that automatically map SQL data types to those that are supported by SAS. Use this form when you want to create a new table with columns that are not present in existing tables. It is also useful if you are running SQL statements from an SQL application in another SQL-based database.
- ❷ The second form uses a LIKE clause to create a table that has the same column names and column attributes as another table. To drop any columns in the new table, you can specify the DROP= data set option in the CREATE TABLE statement. The specified columns are dropped when the table is created. Indexes are not copied to the new table.

Both of these forms create a table without rows. You can use an INSERT statement to add rows. Use an ALTER TABLE statement to modify column attributes or to add or drop columns.

Creating a Table from a Query Expression

- ❸ The third form of the CREATE TABLE statement stores the results of any query-expression in a table and does not display the output. It is a convenient way to create temporary tables that are subsets or supersets of other tables.

When you use this form, a table is physically created as the statement is executed. The newly created table does not reflect subsequent changes in the underlying tables (in the query-expression). If you want to continually access the most current data, then create a view from the query expression instead of a table. See “CREATE VIEW Statement” on page 1133.

CAUTION:

Recursive table references can cause data integrity problems. While it is possible to recursively reference the target table of a CREATE TABLE AS statement, doing so can cause data integrity problems and incorrect results. Constructions such as the following should be avoided:

```
proc sql;
  create table a as
    select var1, var2
    from a;
```

Δ

Integrity Constraints

You can attach integrity constraints when you create a new table. To modify integrity constraints, use the ALTER TABLE statement. For more information on integrity constraints, see the section on SAS files in *SAS Language Reference: Concepts*.

CREATE VIEW Statement

Creates a PROC SQL view from a query-expression.

See also: “What Are Views?” on page 1115

Featured in: Example 8 on page 1224

```

CREATE VIEW proc-sql-view AS query-expression
    <ORDER BY order-by-item<, ... order-by-item>>
    <USING libname-clause<, ... libname-clause>> ;

```

Arguments

query-expression

See “query-expression” on page 1176.

libname-clause

is one of the following:

```

LIBNAME libref <engine> 'SAS-data-library' <option(s)> <engine-host-option(s)>

```

```

LIBNAME libref SAS/ACCESS-engine-name
    <SAS/ACCESS-engine-connection-option(s)>
    <SAS/ACCESS-engine-LIBNAME-option(s)>

```

See the *SAS Language Reference: Dictionary* for information about the base SAS LIBNAME statement and the SAS/ACCESS LIBNAME statement.

order-by-item

See ORDER BY Clause on page 1151.

proc-sql-view

specifies the name for the PROC SQL view that you are creating. See “What Are Views?” on page 1115 for a definition of a PROC SQL view.

Sorting Data Retrieved by Views

PROC SQL enables you to specify the ORDER BY clause in the CREATE VIEW statement. When a view with an ORDER BY clause is accessed, and the ORDER BY clause directly affects the order of the results, its data is sorted and displayed as specified by the ORDER BY clause. However, if the ORDER BY clause does not directly affect the order of the results (for instance, if the view is specified as part of a join), then PROC SQL ignores the ORDER BY clause in order to enhance performance.

Note: If you specify the NUMBER option in the PROC SQL statement when you create your view, then the ROW column appears in the output. However, you cannot order by the ROW column in subsequent queries. See the description of NUMBER|NONNUMBER on page 1123. △

Librefs and Stored Views

You can refer to a table name alone (without the libref) in the FROM clause of a CREATE VIEW statement if the table and view reside in the same SAS data library, as in this example:

```

create view proclib.view1 as
  select *
    from invoice
   where invqty>10;

```

In this view, VIEW1 and INVOICE are stored permanently in the SAS data library referenced by PROCLIB. Specifying a libref for INVOICE is optional.

Updating Views

You can update a view's underlying data with some restrictions. See “Updating PROC SQL and SAS/ACCESS Views” on page 1203.

Embedded LIBNAME Statements

The USING clause enables you to store DBMS connection information in a view by *embedding* the SAS/ACCESS LIBNAME statement inside the view. When PROC SQL executes the view, the stored query assigns the libref and establishes the DBMS connection using the information in the LIBNAME statement. The scope of the libref is local to the view, and will not conflict with any identically named librefs in the SAS session. When the query finishes, the connection to the DBMS is terminated and the libref is deassigned.

The USING clause must be the last clause in the CREATE VIEW statement. Multiple LIBNAME statements can be specified, separated by commas. In the following example, a connection is made and the libref ACCREC is assigned to an ORACLE database.

```
create view proclib.view1 as
  select *
    from accrec.invoices as invoices
  using libname accrec oracle
        user=username pass=password
        path='dbms-path';
```

For more information on the SAS/ACCESS LIBNAME statement, see the SAS/ACCESS documentation for your DBMS.

Note: Starting in Version 9, PROC SQL views, the Pass-Through Facility, and the SAS/ACCESS LIBNAME statement are the preferred ways to access relational DBMS data; SAS/ACCESS views are no longer recommended. You can convert existing SAS/ACCESS views to PROC SQL views by using the CV2VIEW procedure. See “The CV2VIEW Procedure” in *SAS/ACCESS for Relational Databases: Reference* for more information. △

You can also embed a SAS LIBNAME statement in a view with the USING clause. This enables you to store SAS libref information in the view. Just as in the embedded SAS/ACCESS LIBNAME statement, the scope of the libref is local to the view, and it will not conflict with an identically named libref in the SAS session.

```
create view work.tableview as
  select * from proclib.invoices
  using libname proclib 'sas-data-library';
```

DELETE Statement

Removes one or more rows from a table or view that is specified in the FROM clause.

Restriction: You cannot use DELETE FROM on a table that is accessed by an engine that does not support UPDATE processing.

Featured in: Example 5 on page 1216

DELETE

FROM *table-name* | *sas/access-view* | *proc-sql-view* <**AS** *alias*>
 <**WHERE** *sql-expression*>;

Arguments

alias

assigns an alias to *table-name*, *sas/access-view*, or *proc-sql-view*.

sas/access-view

specifies a SAS/ACCESS view that you are deleting rows from.

proc-sql-view

specifies a PROC SQL view that you are deleting rows from. *proc-sql-view* can be a one-level name, a two-level *libref.view* name, or a physical pathname that is enclosed in single quotation marks.

sql-expression

See “*sql-expression*” on page 1182.

table-name

specifies the table that you are deleting rows from. *table-name* can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

CAUTION:

Recursive table references can cause data integrity problems. While it is possible to recursively reference the target table of a DELETE statement, doing so can cause data integrity problems and incorrect results. Constructions such as the following should be avoided:

```
proc sql;
  delete from a
    where var1 > (select min(var2) from a);
```

\triangle

Deleting Rows through Views

You can delete one or more rows from a view’s underlying table, with some restrictions. See “Updating PROC SQL and SAS/ACCESS Views” on page 1203.

CAUTION:

If you omit a WHERE clause, then the DELETE statement deletes all the rows from the specified table or the table that is described by a view. \triangle

DESCRIBE Statement

Displays a PROC SQL definition in the SAS log.

Restriction: PROC SQL views are the only type of view allowed in a DESCRIBE VIEW statement.

Featured in: Example 6 on page 1218

DESCRIBE TABLE *table-name* <, ... *table-name*>;

DESCRIBE VIEW *proc-sql-view* <, ... *proc-sql-view*>;

DESCRIBE TABLE CONSTRAINTS *table-name* <, ... *table-name*>;

Arguments

table-name

specifies a PROC SQL table. *table-name* can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

proc-sql-view

specifies a PROC SQL view. *proc-sql-view* can be a one-level name, a two-level *libref.view* name, or a physical pathname that is enclosed in single quotation marks.

Details

- The DESCRIBE TABLE statement writes a CREATE TABLE statement to the SAS log for the table specified in the DESCRIBE TABLE statement, regardless of how the table was originally created (for example, with a DATA step). If applicable, SAS data set options are included with the table definition. If indexes are defined on columns in the table, then CREATE INDEX statements for those indexes are also written to the SAS log.

When you are transferring a table to a DBMS that is supported by SAS/ACCESS software, it is helpful to know how it is defined. To find out more information about a table, use the FEEDBACK option or the CONTENTS statement in the DATASETS procedure.

- The DESCRIBE VIEW statement writes a view definition to the SAS log. If you use a PROC SQL view in the DESCRIBE VIEW statement that is based on or derived from another view, then you might want to use the FEEDBACK option in the PROC SQL statement. This option displays in the SAS log how the underlying view is defined and expands any expressions that are used in this view definition. The CONTENTS statement in DATASETS procedure can also be used with a view to find out more information.
- The DESCRIBE TABLE CONSTRAINTS statement lists the integrity constraints that are defined for the specified table(s).

DISCONNECT Statement

Ends the connection with a DBMS that is supported by a SAS/ACCESS interface.

Requirement: SAS/ACCESS software is required. For more information on this statement, refer to your SAS/ACCESS documentation.

See also: “Connecting to a DBMS Using the SQL Procedure Pass-Through Facility” on page 1198

DISCONNECT FROM *dbms-name* | *alias*;

Arguments

alias

specifies the alias that is defined in the CONNECT statement.

dbms-name

specifies the DBMS from which you want to end the connection (for example, DB2 or ORACLE). The name you specify should match the name that is specified in the CONNECT statement.

Details

- An implicit COMMIT is performed before the DISCONNECT statement ends the DBMS connection. If a DISCONNECT statement is not submitted, then implicit DISCONNECT and COMMIT actions are performed and the connection to the DBMS is broken when PROC SQL terminates.
- PROC SQL continues executing until you submit a QUIT statement, another SAS procedure, or a DATA step.

DROP Statement

Deletes tables, views, or indexes.

Restriction: You cannot use DROP TABLE or DROP INDEX on a table that is accessed by an engine that does not support UPDATE processing.

DROP TABLE *table-name* <, ... *table-name*>;

DROP VIEW *view-name* <, ... *view-name*>;

DROP INDEX *index-name* <, ... *index-name*>
FROM *table-name*;

Arguments***index-name***

specifies an index that exists on *table-name*.

table-name

specifies a PROC SQL table. *table-name* can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

view-name

specifies a SAS data view of any type: PROC SQL view, SAS/ACCESS view, or DATA step view. *view-name* can be a one-level name, a two-level *libref.view* name, or a physical pathname that is enclosed in single quotation marks.

Details

- If you drop a table that is referenced in a view definition and try to execute the view, then an error message is written to the SAS log that states that the table does not exist. Therefore, remove references in queries and views to any table(s) and view(s) that you drop.

- If you drop a table with indexed columns, then all the indexes are automatically dropped. If you drop a composite index, then the index is dropped for all the columns that are named in that index.
- You can use the DROP statement to drop a table or view in an external database that is accessed with the Pass-Through Facility or SAS/ACCESS LIBNAME statement, but not for an external database table or view that is described by a SAS/ACCESS view.

EXECUTE Statement

Sends a DBMS-specific SQL statement to a DBMS that is supported by a SAS/ACCESS interface.

Requirement: SAS/ACCESS software is required. For more information on this statement, refer to your SAS/ACCESS documentation.

See also: “Connecting to a DBMS Using the SQL Procedure Pass-Through Facility” on page 1198 and the SQL documentation for your DBMS.

```
EXECUTE (dbms-SQL-statement)
      BY dbms-name | alias;
```

Arguments

alias

specifies an optional alias that is defined in the CONNECT statement. Note that *alias* must be preceded by the keyword BY.

dbms-name

identifies the DBMS to which you want to direct the DBMS statement (for example, ORACLE or DB2).

dbms-SQL-statement

is any DBMS-specific SQL statement, except the SELECT statement, that can be executed by the DBMS-specific dynamic SQL.

Details

- If your DBMS supports multiple connections, then you can use the alias that is defined in the CONNECT statement. This alias directs the EXECUTE statements to a specific DBMS connection.
- Any return code or message that is generated by the DBMS is available in the macro variables SQLXRC and SQLXMSG after the statement completes.

INSERT Statement

Adds rows to a new or existing table or view.

Restriction: You cannot use INSERT INTO on a table that is accessed with an engine that does not support UPDATE processing.

Featured in: Example 1 on page 1207

- ❶ **INSERT INTO** *table-name* | *sas/access-view* | *proc-sql-view* <(column<, ... column>)>
SET *column*=sql-expression
 <,> ... *column*=sql-expression>
 <**SET** *column*=sql-expression
 <,> ... *column*=sql-expression>;
- ❷ **INSERT INTO** *table-name* | *sas/access-view* | *proc-sql-view* <(column<, ... column>)>
VALUES (*value* <,> ... *value*>)
 <... **VALUES** (*value* <,> ... *value*>)>;
- ❸ **INSERT INTO** *table-name* | *sas/access-view* | *proc-sql-view*
 <(column<, ...column>)> query-expression;

Arguments

column

specifies the column into which you are inserting rows.

proc-sql-view

specifies a PROC SQL view into which you are inserting rows. *proc-sql-view* can be a one-level name, a two-level *libref.view* name, or a physical pathname that is enclosed in single quotation marks.

query-expression

See “query-expression” on page 1176.

sas/access-view

specifies a SAS/ACCESS view into which you are inserting rows.

sql-expression

See “sql-expression” on page 1182.

table-name

specifies a PROC SQL table into which you are inserting rows. *table-name* can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

value

is a data value.

CAUTION:

Recursive table references can cause data integrity problems. While it is possible to recursively reference the target table of an INSERT statement, doing so can cause data integrity problems and incorrect results. Constructions such as the following should be avoided:

```
proc sql;
  insert into a
    select var1, var2
  from a
  where var1 > 0;
```

\triangle

Methods for Inserting Values

- ❶ The first form of the INSERT statement uses the SET clause, which specifies or alters the values of a column. You can use more than one SET clause per INSERT

statement, and each SET clause can set the values in more than one column. Multiple SET clauses are not separated by commas. If you specify an optional list of columns, then you can set a value only for a column that is specified in the list of columns to be inserted.

- ② The second form of the INSERT statement uses the VALUES clause. This clause can be used to insert lists of values into a table. You can either give a value for each column in the table or give values just for the columns specified in the list of column names. One row is inserted for each VALUES clause. Multiple VALUES clauses are not separated by commas. The order of the values in the VALUES clause matches the order of the column names in the INSERT column list or, if no list was specified, the order of the columns in the table.
- ③ The third form of the INSERT statement inserts the results of a query-expression into a table. The order of the values in the query-expression matches the order of the column names in the INSERT column list or, if no list was specified, the order of the columns in the table.

Note: If the INSERT statement includes an optional list of column names, then only those columns are given values by the statement. Columns that are in the table but not listed are given missing values. Δ

Inserting Rows through Views

You can insert one or more rows into a table through a view, with some restrictions. See “Updating PROC SQL and SAS/ACCESS Views” on page 1203.

Adding Values to an Indexed Column

If an index is defined on a column and you insert a new row into the table, then that value is added to the index. You can display information about indexes with

- ❑ the CONTENTS statement in the DATASETS procedure. See “CONTENTS Statement” on page 344.
- ❑ the DICTIONARY.INDEXES table. See “Using the DICTIONARY Tables” on page 1199 for more information.

For more information on creating and using indexes, see “CREATE INDEX Statement” on page 1128.

RESET Statement

Resets PROC SQL options without restarting the procedure.

Featured in: Example 5 on page 1216

RESET <option(s)>;

The RESET statement enables you to add, drop, or change the options in PROC SQL without restarting the procedure. See “PROC SQL Statement” on page 1119 for a description of the options.

SELECT Statement

Selects columns and rows of data from tables and views.

Restriction: The clauses in the SELECT statement must appear in the order shown.

See also: “table-expression” on page 1196, “query-expression” on page 1176

```

SELECT <DISTINCT> object-item <, ...object-item>
    <INTO macro-variable-specification
      <[, ... macro-variable-specification>>
    FROM from-list
    <WHERE sql-expression>
    <GROUP BY group-by-item
      <[, ... group-by-item>>
    <HAVING sql-expression>
    <ORDER BY order-by-item
      <[, ... order-by-item>>;

```

SELECT Clause

Lists the columns that will appear in the output.

See Also: “column-definition” on page 1159

Featured in: Example 1 on page 1207 and Example 2 on page 1209

```

SELECT <DISTINCT> object-item <, ... object-item>

```

Arguments

alias

assigns a temporary, alternate name to the column.

DISTINCT

eliminates duplicate rows.

Featured in: Example 13 on page 1235

object-item

is one of the following:

*

represents all columns in a tables or views that are listed in the FROM clause.

case-expression <AS *alias*>

derives a column from a CASE expression. See “CASE expression” on page 1157.

column-name <<AS> *alias*>

<column-modifier <... column-modifier>>

names a single column. See “column-name” on page 1161 and “column-modifier” on page 1160.

sql-expression <AS *alias*>

<column-modifier <... column modifier>>

derives a column from an sql-expression. See “sql-expression” on page 1182 and “column-modifier” on page 1160.

table-name.*

specifies all columns in the PROC SQL table that is specified in *table-name*.

table-alias.*

specifies all columns in the PROC SQL table that has the alias that is specified in *table-alias*.

view-name.*

specifies all columns in the SAS data view that is specified in *view-name*.

view-alias.*

specifies all columns in the SAS data view that has the alias that is specified in *view-alias*.

Asterisk (*) Notation

The asterisk (*) represents all columns of the table(s) listed in the FROM clause. When an asterisk is not prefixed with a table name, all the columns from all tables in the FROM clause are included; when it is prefixed (for example, *table-name*.* or *table-alias*.*), all the columns from that table only are included.

Column Aliases

A column alias is a temporary, alternate name for a column. Aliases are specified in the SELECT clause to name or rename columns so that the result table is clearer or easier to read. Aliases are often used to name a column that is the result of an arithmetic expression or summary function. An alias is one word only. If you need a longer column name, then use the LABEL= column-modifier, as described in “column-modifier” on page 1160. The keyword AS is not required with a column alias.

Column aliases are optional, and each column name in the SELECT clause can have an alias. After you assign an alias to a column, you can use the alias to refer to that column in other clauses.

If you use a column alias when creating a PROC SQL view, then the alias becomes the permanent name of the column for each execution of the view.

INTO Clause

Stores the value of one or more columns for use later in another PROC SQL query or SAS statement.

Restriction: An INTO clause cannot be used in a CREATE TABLE statement.

See also: “Using Macro Variables Set by PROC SQL” on page 1202

INTO *macro-variable-specification*

<, ... *macro-variable-specification*>

Arguments

macro-variable

specifies a SAS macro variable that stores the values of the rows that are returned.

macro-variable-specification

is one of the following:

:macro-variable <SEPARATED BY 'character(s)' <NOTRIM>>

stores the values that are returned into a single macro variable.

:macro-variable-1 – :macro-variable-n <NOTRIM>

stores the values that are returned into a range of macro variables.

NOTRIM

protects the leading and trailing blanks from being deleted from values that are stored in a range of macro variables or multiple values that are stored in a single macro variable.

SEPARATED BY 'character'

specifies a character that separates the values of the rows.

Details

- Use the INTO clause only in the outer query of a SELECT statement and not in a subquery.
- When storing a single value into a macro variable, PROC SQL preserves leading or trailing blanks. However, when storing values into a range of macro variables, or when using the SEPARATED BY option to store multiple values in one macro variable, PROC SQL trims leading or trailing blanks unless you use the NOTRIM option.
- You can put multiple rows of the output into macro variables. You can check the PROC SQL macro variable SQLOBS to see the number of rows that are produced by a query-expression. See “Using Macro Variables Set by PROC SQL” on page 1202 for more information on SQLOBS.

Examples

These examples use the PROCLIB.HOUSES table:

The SAS System		1
Style	SqFeet	
CONDO	900	
CONDO	1000	
RANCH	1200	
RANCH	1400	
SPLIT	1600	
SPLIT	1800	
TWOSTORY	2100	
TWOSTORY	3000	
TWOSTORY	1940	
TWOSTORY	1860	

With the *macro-variable-specification*, you can do the following:

- You can create macro variables based on the first row of the result.

```
proc sql noprint;
  select style, sqfeet
    into :style, :sqfeet
    from proclib.houses;

%put &style &sqfeet;
```

The results are written to the SAS log:

```
1  proc sql noprint;
2    select style, sqfeet
3      into :style, :sqfeet
4      from proclib.houses;
5
6  %put &style &sqfeet;
CONDO          900
```

- You can create one new macro variable per row in the result of the SELECT statement. This example shows how you can request more values for one column than for another. The hyphen (-) is used in the INTO clause to imply a range of macro variables. You can use either of the keywords THROUGH or THRU instead of a hyphen.

The following PROC SQL step puts the values from the first four rows of the PROCLIB.HOUSES table into macro variables:

```
proc sql noprint;
  select distinct Style, SqFeet
    into :style1 - :style3, :sqfeet1 - :sqfeet4
    from proclib.houses;

%put &style1 &sqfeet1;
%put &style2 &sqfeet2;
%put &style3 &sqfeet3;
%put &sqfeet4;
```

The %PUT statements write the results to the SAS log:

```
1  proc sql noprint;
2    select distinct style, sqfeet
3      into :style1 - :style3, :sqfeet1 - :sqfeet4
4      from proclib.houses;
5
6  %put &style1 &sqfeet1;
CONDO 900
7  %put &style2 &sqfeet2;
CONDO 1000
8  %put &style3 &sqfeet3;
RANCH 1200
9  %put &sqfeet4;
1400
```

- You can concatenate the values of one column into one macro variable. This form is useful for building up a list of variables or constants.

```
proc sql noprint;
  select distinct style
    into :s1 separated by ','
    from proclib.houses;
```

```
%put &s1;
```

The results are written to the SAS log:

```
3   proc sql noprint;
4       select distinct style
5           into :s1 separated by ','
6       from proclib.houses;
7
8   %put &s1

CONDO,RANCH,SPLIT,TWOSTORY
```

- You can use leading zeros in order to create a range of macro variable names, as shown in the following example:

```
proc sql noprint;
    select SqFeet
        into :sqfeet01 - :sqfeet10
        from proclib.houses;

%put &sqfeet01 &sqfeet02 &sqfeet03 &sqfeet04 &sqfeet05;
%put &sqfeet06 &sqfeet07 &sqfeet08 &sqfeet09 &sqfeet10;
```

The results are written to the SAS log:

```
11  proc sql noprint;
12      select sqfeet
13          into :sqfeet01 - :sqfeet10
14      from proclib.houses;

15  %put &sqfeet01 &sqfeet02 &sqfeet03 &sqfeet04 &sqfeet05;
900 1000 1200 1400 1600
16  %put &sqfeet06 &sqfeet07 &sqfeet08 &sqfeet09 &sqfeet10;
1800 2100 3000 1940 1860
```

- You can prevent leading and trailing blanks from being trimmed from values that are stored in macro variables. By default, when storing values in a range of macro variables or when storing multiple values in one macro variable (with the SEPARATED BY option), PROC SQL trims the leading and trailing blanks from the values before creating the macro variables. If you do not want the blanks to be trimmed, then add the NOTRIM option, as shown in the following example:

```
proc sql noprint;
    select style, sqfeet
        into :style1 - :style4 notrim,
            :sqfeet separated by ',' notrim
        from proclib.houses;

%put *&style1* *&sqfeet*;
%put *&style2* *&sqfeet*;
%put *&style3* *&sqfeet*;
%put *&style4* *&sqfeet*;
```

The results are written to the SAS log, as shown in the following output:

```

3  proc sql noprint;
4      select style, sqfeet
5          into :style1 - :style4 notrim,
6              :sqfeet separated by ',' notrim
7          from proclib.houses;
8
9  %put *%style1* *%sqfeet*;
*CONDO * *      900,    1000,    1200,    1400,    1600,    1800,    2100,
3000,    1940,    1860*
10 %put *%style2* *%sqfeet*;
*CONDO * *      900,    1000,    1200,    1400,    1600,    1800,    2100,
3000,    1940,    1860**
11 %put *%style3* *%sqfeet*;
*RANCH * *      900,    1000,    1200,    1400,    1600,    1800,    2100,
3000,    1940,    1860**
12 %put *%style4* *%sqfeet*;
*RANCH * *      900,    1000,    1200,    1400,    1600,    1800,    2100,
3000,    1940,    1860**

```

FROM Clause

Specifies source tables or views.

Featured in: Example 1 on page 1207, Example 4 on page 1213, Example 9 on page 1227, and Example 10 on page 1230

FROM *from-list*

Arguments

alias

specifies a temporary, alternate name for a table, view, or in-line view that is specified in the FROM clause.

column

names the column that appears in the output. The column names that you specify are matched by position to the columns in the output.

from-list

is one of the following:

table-name <<AS> *alias*>

names a single PROC SQL table. *table-name* can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

view-name <<AS> *alias*>

names a single SAS data view. *view-name* can be a one-level name, a two-level *libref.view* name, or a physical pathname that is enclosed in single quotation marks.

joined-table

specifies a join. See “joined-table” on page 1165.

(query-expression) <<AS *alias*>
 <(column <, ... column>)>>

specifies an in-line view. See “query-expression” on page 1176.

CONNECTION TO

specifies a DBMS table. See “CONNECTION TO” on page 1162.

Note: With *table-name* and *view-name*, you can use data set options by placing them in parentheses immediately after *table-name* or *view-name*. See “Using SAS Data Set Options with PROC SQL” on page 1197 for details. △

Table Aliases

A table alias is a temporary, alternate name for a table that is specified in the FROM clause. Table aliases are prefixed to column names to distinguish between columns that are common to multiple tables. Column names in reflexive joins (joining a table with itself) must be prefixed with a table alias in order to distinguish which copy of the table the column comes from. Column names in other kinds of joins must be prefixed with table aliases or table names unless the column names are unique to those tables.

The optional keyword AS is often used to distinguish a table alias from other table names.

In-Line Views

The FROM clause can itself contain a query-expression that takes an optional table alias. This kind of nested query-expression is called an *in-line view*. An in-line view is any query-expression that would be valid in a CREATE VIEW statement. PROC SQL can support many levels of nesting, but it is limited to 32 tables in any one query. The 32-table limit includes underlying tables that may contribute to views that are specified in the FROM clause.

An in-line view saves you a programming step. Rather than creating a view and referring to it in another query, you can specify the view *in-line* in the FROM clause.

Characteristics of in-line views include the following:

- An in-line view is not assigned a permanent name, although it can take an alias.
- An in-line view can be referred to only in the query in which it is defined. It cannot be referenced in another query.
- You cannot use an ORDER BY clause in an in-line view.
- The names of columns in an in-line view can be assigned in the object-item list of that view or with a parenthesized list of names following the alias. This syntax can be useful for renaming columns. See Example 10 on page 1230 for an example.
- In order to visually separate an in-line view from the rest of the query, you can enclose the in-line view in any number of pairs of parentheses. Note that if you specify an alias for the in-line view, the alias specification must appear outside the outermost pair of parentheses for that in-line view.

WHERE Clause

Subsets the output based on specified conditions.

Featured in: Example 4 on page 1213 and Example 9 on page 1227

WHERE sql-expression

Argument

sql-expression

See “sql-expression” on page 1182.

Details

- When a condition is met (that is, the condition resolves to true), those rows are displayed in the result table; otherwise, no rows are displayed.
- You cannot use summary functions that specify only one column. For example:

```
where max(measure1) > 50;
```

However, this WHERE clause will work:

```
where max(measure1,measure2) > 50;
```

GROUP BY Clause

Specifies how to group the data for summarizing.

Featured in: Example 8 on page 1224 and Example 12 on page 1233

GROUP BY *group-by-item* <, ..., *group-by-item*>

Arguments

group-by-item

is one of the following:

integer

is a positive integer that equates to a column's position.

column-name

is the name of a column or a column alias. See “column-name” on page 1161.

sql-expression

See “sql-expression” on page 1182.

Details

- You can specify more than one *group-by-item* to get more detailed reports. Both the grouping of multiple items and the BY statement of a PROC step are evaluated in similar ways. If more than one *group-by-item* is specified, then the first one determines the major grouping.
- Integers can be substituted for column names (that is, SELECT object-items) in the GROUP BY clause. For example, if the *group-by-item* is 2, then the results are grouped by the values in the second column of the SELECT clause list. Using

integers can shorten your coding and enable you to group by the value of an unnamed expression in the SELECT list. Note that if you use a floating-point value (for example, 2.3), then PROC SQL ignores the decimal portion.

- The data does not have to be sorted in the order of the group-by values because PROC SQL handles sorting automatically. You can use the ORDER BY clause to specify the order in which rows are displayed in the result table.
- If you specify a GROUP BY clause in a query that does not contain a summary function, then your clause is transformed into an ORDER BY clause and a message to that effect is written to the SAS log.
- You can group the output by the values that are returned by an expression. For example, if X is a numeric variable, then the output of the following is grouped by the integer portion of values of X:

```
select x, sum(y)
from table1
group by int(x);
```

Similarly, if Y is a character variable, then the output of the following is grouped by the second character of values of Y:

```
select sum(x), y
from table1
group by substring(y from 2 for 1);
```

Note that an expression that contains only numeric literals (and functions of numeric literals) or only character literals (and functions of character literals) is ignored.

An expression in a GROUP BY clause cannot be a summary function. For example, the following GROUP BY clause is not valid:

```
group by sum(x)
```

HAVING Clause

Subsets grouped data based on specified conditions.

Featured in: Example 8 on page 1224 and Example 12 on page 1233

HAVING sql-expression

Argument

sql-expression

See “sql-expression” on page 1182.

Subsetting Grouped Data

The HAVING clause is used with at least one summary function and an optional GROUP BY clause to summarize groups of data in a table. A HAVING clause is any valid SQL expression that is evaluated as either true or false for each group in a query. Alternatively, if the query involves remerged data, then the HAVING expression is evaluated for each row that participates in each group. The query must include one or more summary functions.

Typically, the GROUP BY clause is used with the HAVING expression and defines the group(s) to be evaluated. If you omit the GROUP BY clause, then the summary function and the HAVING clause treat the table as one group.

The following PROC SQL step uses the PROCLIB.PAYROLL table (shown in Example 2 on page 1209) and groups the rows by Gender to determine the oldest employee of each gender. In SAS, dates are stored as integers. The lower the birth date as an integer, the greater the age. The expression **birth=min(birth)** is evaluated for each row in the table. When the minimum birth date is found, the expression becomes true and the row is included in the output.

```
proc sql;
  title 'Oldest Employee of Each Gender';
  select *
    from proclib.payroll
  group by gender
  having birth=min(birth);
```

Note: This query involves remerged data because the values returned by a summary function are compared to values of a column that is not in the GROUP BY clause. See “Remerging Data” on page 1192 for more information about summary functions and remerging data. \triangle

ORDER BY Clause

Specifies the order in which rows are displayed in a result table.

See also: “query-expression” on page 1176

Featured in: Example 11 on page 1231

ORDER BY *order-by-item* <ASC|DESC><, ... *order-by-item* <ASC|DESC>>;

Arguments

order-by-item

is one of the following:

integer

equates to a column's position.

column-name

is the name of a column or a column alias. See “column-name” on page 1161.

sql-expression

See “sql-expression” on page 1182.

ASC

orders the data in ascending order. This is the default order; if neither ASC nor DESC is specified, the data is ordered in ascending order.

DESC

orders the data in descending order.

Details

- The ORDER BY clause sorts the result of a query expression according to the order specified in that query. When this clause is used, the default ordering sequence is ascending, from the lowest value to the highest. You can use the SORTSEQ= option to change the collating sequence for your output. See “PROC SQL Statement” on page 1119.
- If an ORDER BY clause is omitted, then a particular order to the output rows, such as the order in which the rows are encountered in the queried table, cannot be guaranteed. Without an ORDER BY clause, the order of the output rows is determined by the internal processing of PROC SQL, the default collating sequence of SAS, and your operating environment. Therefore, if you want your result table to appear in a particular order, then use the ORDER BY clause.
- If more than one *order-by-item* is specified (separated by commas), then the first one determines the major sort order.
- Integers can be substituted for column names (that is, SELECT object-items) in the ORDER BY clause. For example, if the *order-by-item* is 2 (an integer), then the results are ordered by the values of the second column. If a query-expression includes a set operator (for example, UNION), then use integers to specify the order. Doing so avoids ambiguous references to columns in the table expressions. Note that if you use a floating-point value (for example, 2.3) instead of an integer, then PROC SQL ignores the decimal portion.
- In the ORDER BY clause, you can specify any column of a table or view that is specified in the FROM clause of a query-expression, regardless of whether that column has been included in the query’s SELECT clause. For example, this query produces a report ordered by the descending values of the population change for each country from 1990 to 1995:

```
proc sql;
  select country
    from census
   order by pop95-pop90 desc;
```

NOTE: The query as specified involves ordering by an item that doesn’t appear in its SELECT clause.

- You can order the output by the values that are returned by an expression. For example, if X is a numeric variable, then the output of the following is ordered by the integer portion of values of X:

```
select x, y
  from table1
 order by int(x);
```

Similarly, if Y is a character variable, then the output of the following is ordered by the second character of values of Y:

```
select x, y
  from table1
 order by substring(y from 2 for 1);
```

Note that an expression that contains only numeric literals (and functions of numeric literals) or only character literals (and functions of character literals) is ignored.

UPDATE Statement

Modifies a column's values in existing rows of a table or view.

Restriction: You cannot use UPDATE on a table that is accessed by an engine that does not support UPDATE processing.

Featured in: Example 3 on page 1211

```
UPDATE table-name | sas/access-view | proc-sql-view <AS alias>
  SET column=sql-expression
    <, ... column=sql-expression>
  <SET column=sql-expression
    <, ... column=sql-expression>>
  <WHERE sql-expression>;
```

Arguments

alias

assigns an alias to *table-name*, *sas/access-view*, or *proc-sql-view*.

column

specifies a column in *table-name*, *sas/access-view*, or *proc-sql-view*.

sas/access-view

specifies a SAS/ACCESS view.

sql-expression

See “sql-expression” on page 1182.

table-name

specifies a PROC SQL table. *table-name* can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

proc-sql-view

specifies a PROC SQL view. *proc-sql-view* can be a one-level name, a two-level *libref.view* name, or a physical pathname that is enclosed in single quotation marks.

Updating Tables through Views

You can update one or more rows of a table through a view, with some restrictions. See “Updating PROC SQL and SAS/ACCESS Views” on page 1203.

Details

- Any column that is not modified retains its original values, except in certain queries using the CASE expression. See “CASE expression” on page 1157 for a description of CASE expressions.

- To add, drop, or modify a column's definition or attributes, use the ALTER TABLE statement, described in “ALTER TABLE Statement” on page 1124.
- In the SET clause, a column reference on the left side of the equal sign can also appear as part of the expression on the right side of the equal sign. For example, you could use this expression to give employees a \$1,000 holiday bonus:

```
set salary=salary + 1000
```

- If you omit the WHERE clause, then all the rows are updated. When you use a WHERE clause, only the rows that meet the WHERE condition are updated.
- When you update a column and an index has been defined for that column, the values in the updated column continue to have the index defined for them.

VALIDATE Statement

Checks the accuracy of a query-expression's syntax and semantics without executing the expression.

VALIDATE query-expression;

Argument

query-expression

See “query-expression” on page 1176.

Details

- The VALIDATE statement writes a message in the SAS log that states that the query is valid. If there are errors, then VALIDATE writes error messages to the SAS log.
- The VALIDATE statement can also be included in applications that use the macro facility. When used in such an application, VALIDATE returns a value that indicates the query-expression's validity. The value is returned through the macro variable SQLRC (a short form for SQL return code). For example, if a SELECT statement is valid, then the macro variable SQLRC returns a value of 0. See “Using Macro Variables Set by PROC SQL” on page 1202 for more information.

SQL Procedure Component Dictionary

This section describes the components that are used in SQL procedure statements. *Components* are the items in PROC SQL syntax that appear in roman type.

Most components are contained in clauses within the statements. For example, the basic SELECT statement is composed of the SELECT and FROM clauses, where each clause contains one or more components. Components can also contain other components.

For easy reference, components appear in alphabetical order, and some terms are referred to before they are defined. Use the index or the “See Also” references to refer to other statement or component descriptions that may be helpful.

BETWEEN condition

Selects rows where column values are within a range of values.

```
sql-expression <NOT> BETWEEN sql-expression
      AND sql-expression
```

Argument

sql-expression

is described in “sql-expression” on page 1182.

Details

- The sql-expressions must be of compatible data types. They must be either all numeric or all character types.
- Because a BETWEEN condition evaluates the boundary values as a range, it is not necessary to specify the smaller quantity first.
- You can use the NOT logical operator to exclude a range of numbers, for example, to eliminate customer numbers between 1 and 15 (inclusive) so that you can retrieve data on more recently acquired customers.
- PROC SQL supports the same comparison operators that the DATA step supports. For example:

```
x between 1 and 3
x between 3 and 1
1<=x<=3
x>=1 and x<=3
```

BTRIM function

Removes blanks or specified characters from the beginning, the end, or both the beginning and end of a character string.

```
BTRIM (<<btrim-specification> <'btrim-character' FROM>> sql-expression)
```

Arguments

btrim-specification

is one of the following:

LEADING

removes the blanks or specified characters from the beginning of the character string.

TRAILING

removes the blanks or specified characters from the end of the character string.

BOTH

removes the blanks or specified characters from both the beginning and the end of the character string.

Default: BOTH

btrim-character

is a single character that is to be removed from the character string. The default character is a blank.

sql-expression

must resolve to a character string or character variable and is described in “sql-expression” on page 1182.

Details

The BTRIM function operates on character strings. BTRIM removes one or more instances of a single character (the value of *btrim-character*) from the beginning, the end, or both the beginning and end of a string, depending whether LEADING, TRAILING, or BOTH is specified. If *btrim-specification* is not specified, then BOTH is used. If *btrim-character* is omitted, then blanks are removed.

Note: SAS adds trailing blanks to character values that are shorter than the length of the variable. Suppose you have a character variable Z, with length 10, and a value **xxabcxx**. SAS stores the value with three blanks after the last x (for a total length of 10). If you attempt to remove all the x characters with

```
btrim(both 'x' from z)
```

then the result is **abcxx** because PROC SQL sees the trailing characters as blanks, not the x character. In order to remove all the x characters, use

```
btrim(both 'x' from btrim(z))
```

The inner BTRIM function removes the trailing blanks before passing the value to the outer BTRIM function. △

CALCULATED

Refers to columns already calculated in the SELECT clause.

CALCULATED *column-alias*

Argument

column-alias

is the name that is assigned to the column in the SELECT clause.

Referencing a CALCULATED Column

CALCULATED enables you to use the results of an expression in the same SELECT clause or in the WHERE clause. It is valid only when used to refer to columns that are calculated in the immediate query expression.

CASE expression

Selects result values that satisfy specified conditions.

Featured in: Example 3 on page 1211 and Example 13 on page 1235

```

CASE <case-operand>
  WHEN when-condition THEN result-expression
  <...WHEN when-condition THEN result-expression>
  <ELSE result-expression>
END

```

Arguments

case-operand

is a valid sql-expression that resolves to a table column whose values are compared to all the *when-conditions*. See “sql-expression” on page 1182.

when-condition

- When *case-operand* is specified, *when-condition* is a shortened sql-expression that assumes *case-operand* as one of its operands and that resolves to true or false.
- When *case-operand* is not specified, *when-condition* is an sql-expression that resolves to true or false.

result-expression

is an sql-expression that resolves to a value.

Details

The CASE expression selects values if certain conditions are met. A CASE expression returns a single value that is conditionally evaluated for each row of a table (or view). Use the WHEN-THEN clauses when you want to execute a CASE expression for some but not all of the rows in the table that is being queried or created. An optional ELSE expression gives an alternative action if no THEN expression is executed.

When you omit *case-operand*, *when-condition* is evaluated as a Boolean (true or false) value. If *when-condition* returns a nonzero, nonmissing result, then the WHEN clause is true. If *case-operand* is specified, then it is compared with *when-condition* for equality. If *case-operand* equals *when-condition*, then the WHEN clause is true.

If the *when-condition* is true for the row being executed, then the *result-expression* following THEN is executed. If *when-condition* is false, then PROC SQL evaluates the next *when-condition* until they are all evaluated. If every *when-condition* is false, then PROC SQL executes the ELSE expression, and its result becomes the CASE expression’s result. If no ELSE expression is present and every *when-condition* is false, then the result of the CASE expression is a missing value.

You can use CASE expressions in the SELECT, UPDATE, and INSERT statements, and as either operand in an sql-expression.

Example

The following two PROC SQL steps show two equivalent CASE expressions that create a character column with the strings in the THEN clause. The CASE expression in the second PROC SQL step is a shorthand method that is useful when all the comparisons are with the same column.

```
proc sql;
  select *, case
    when degrees > 80 then 'Hot'
    when degrees < 40 then 'Cold'
    else 'Mild'
  end
  from temperatures;

proc sql;
  select *, case Degrees
    when > 80 then 'Hot'
    when < 40 then 'Cold'
    else 'Mild'
  end
  from temperatures;
```

COALESCE Function

Returns the first nonmissing value from a list of columns.

Featured in: Example 7 on page 1220

COALESCE (column-name <, ... column-name>)

Arguments

column-name

is described in “column-name” on page 1161.

Details

COALESCE accepts one or more column names of the same data type. The COALESCE function checks the value of each column in the order in which they are listed and returns the first nonmissing value. If only one column is listed, the COALESCE function returns the value of that column. If all the values of all arguments are missing, the COALESCE function returns a missing value.

In some SQL DBMSs, the COALESCE function is called the IFNULL function. See “PROC SQL and the ANSI Standard” on page 1204 for more information.

Note: If your query contains a large number of COALESCE function calls, it might be more efficient to use a natural join instead. See “Natural Joins” on page 1171. \triangle

column-definition

Defines PROC SQL's data types and dates.

See also: “column-modifier” on page 1160

Featured in: Example 1 on page 1207

column data-type <column-modifier <... column-modifier>>

Arguments

column

is a column name.

column-modifier

is described in “column-modifier” on page 1160.

data-type

is one of the following data types:

CHARACTER | VARCHAR <(width)>

indicates a character column with a column width of *width*. The default column width is eight characters.

INTEGER | SMALLINT

indicates an integer column.

DECIMAL | NUMERIC | FLOAT <(width<, ndec>)>

indicates a floating-point column with a column width of *width* and *ndec* decimal places.

REAL | DOUBLE PRECISION

indicates a floating-point column.

DATE

indicates a date column.

Details

- SAS supports many but not all of the data types that SQL-based databases support.
- For all the numeric data types (INTEGER, SMALLINT, DECIMAL, NUMERIC, FLOAT, REAL, DOUBLE PRECISION, and DATE), the SQL procedure defaults to the SAS data type NUMERIC. The *width* and *ndec* arguments are ignored; PROC SQL creates all numeric columns with the maximum precision allowed by SAS. If you want to create numeric columns that use less storage space, then use the LENGTH statement in the DATA step. The various numeric data type names, along with the *width* and *ndec* arguments, are included for compatibility with other SQL software.
- For the character data types (CHARACTER and VARCHAR), the SQL procedure defaults to the SAS data type CHARACTER. The *width* argument is honored.

- The CHARACTER, INTEGER, and DECIMAL data types can be abbreviated to CHAR, INT, and DEC, respectively.
- A column that is declared with DATE is a SAS numeric variable with a date informat or format. You can use any of the column-modifiers to set the appropriate attributes for the column that is being defined. See *SAS Language Reference: Dictionary* for more information on dates.

column-modifier

Sets column attributes.

See also: “column-definition” on page 1159 and SELECT Clause on page 1142

Featured in: Example 1 on page 1207 and Example 2 on page 1209

column-modifier

Arguments

column-modifier

is one of the following:

INFORMAT=*informatw.d*

specifies a SAS informat to be used when SAS accesses data from a table or view. You can change one permanent informat to another by using the ALTER statement. PROC SQL stores informats in its table definitions so that other SAS procedures and the DATA step can use this information when they reference tables created by PROC SQL.

See *SAS Language Reference: Dictionary* for more information about informats.

FORMAT=*formatw.d*

specifies a SAS format for determining how character and numeric values in a column are displayed by the query-expression. If the FORMAT= modifier is used in the ALTER, CREATE TABLE, or CREATE VIEW statements, then it specifies the permanent format to be used when SAS displays data from that table or view. You can change one permanent format to another by using the ALTER statement.

See *SAS Language Reference: Dictionary* for more information about formats.

LABEL=*'label'*

specifies a column label. If the LABEL= modifier is used in the ALTER, CREATE TABLE, or CREATE VIEW statements, then it specifies the permanent label to be used when displaying that column. You can change one permanent label to another by using the ALTER statement.

A label can begin with the following characters: a through z, A through Z, 0 through 9, an underscore (_), or a blank space. If you begin a label with any other character, such as pound sign (#), then that character is used as a split character and it splits the label onto the next line wherever it appears. For example:

```
select dropout label=
'#Percentage of#Students Who#Dropped Out'
from educ(obs=5);
```

If a special character must appear as the first character in the output, then precede it with a space or a forward slash (/).

You can omit the LABEL= part of the column-modifier and still specify a label. Be sure to enclose the label in quotation marks, as in this example:

```
select empname "Names of Employees"
  from sql.employees;
```

If an apostrophe must appear in the label, then type it twice so that SAS reads the apostrophe as a literal. Alternatively, you can use single and double quotation marks alternately (for example, “Date Rec’d”).

LENGTH=*length*
specifies the length of the column.

Details

If you refer to a labeled column in the ORDER BY or GROUP BY clause, then you must use either the column name (not its label), the column’s alias, or its ordering integer (for example, **ORDER BY 2**). See the section on SAS statements in *SAS Language Reference: Dictionary* for more information about labels.

column-name

Specifies the column to select.

See also: “column-modifier” on page 1160 and SELECT Clause on page 1142

column-name

column-name

is one of the following:

column

is the name of a column.

table-name.column

is the name of a column in the table *table-name*.

table-alias.column

is the name of a column in the table that is referenced by *table-alias*.

view-name.column

is the name of a column in the view *view-name*.

view-alias.column

is the name of a column in the view that is referenced by *view-alias*.

Details

A column can be referred to by its name alone if it is the only column by that name in all the tables or views listed in the current query-expression. If the same column name exists in more than one table or view in the query-expression, then you must

qualify each use of the column name by prefixing a reference to the table that contains it. Consider the following examples:

```
SALARY      /* name of the column */
EMP.SALARY  /* EMP is the table or view name */
E.SALARY    /* E is an alias for the table
              or view that contains the
              SALARY column */
```

CONNECTION TO

Retrieves and uses DBMS data in a PROC SQL query or view.

Tip: You can use CONNECTION TO in the SELECT statement's FROM clause as part of the from-list.

See also: "Connecting to a DBMS Using the SQL Procedure Pass-Through Facility" on page 1198 and your SAS/ACCESS documentation.

CONNECTION TO *dbms-name* (*dbms-query*)

CONNECTION TO *alias* (*dbms-query*)

Arguments

alias

specifies an alias, if one was defined in the CONNECT statement.

dbms-name

identifies the DBMS that you are using.

dbms-query

specifies the query to send to a DBMS. The query uses the DBMS's dynamic SQL. You can use any SQL syntax that the DBMS understands, even if that is not valid for PROC SQL. However, your DBMS query cannot contain a semicolon because that represents the end of a statement to SAS.

The number of tables that you can join with *dbms-query* is determined by the DBMS. Each CONNECTION TO component counts as one table toward the 32-table PROC SQL limit for joins.

See *SAS/ACCESS for Relational Databases: Reference* for more information about DBMS queries.

CONTAINS condition

Tests whether a string is part of a column's value.

Restriction: The CONTAINS condition is used only with character operands.

Featured in: Example 7 on page 1220

sql-expression <NOT> **CONTAINS** sql-expression

Argument

sql-expression

is described in “sql-expression” on page 1182.

EXISTS condition

Tests if a subquery returns one or more rows.

See also: “Query Expressions (Subqueries)” on page 1185

<NOT> **EXISTS** (query-expression)

Argument

query-expression

is described in “query-expression” on page 1176.

Details

The EXISTS condition is an operator whose right operand is a subquery. The result of an EXISTS condition is true if the subquery resolves to at least one row. The result of a NOT EXISTS condition is true if the subquery evaluates to zero rows. For example, the following query subsets PROCLIB.PAYROLL (which is shown in Example 2 on page 1209) based on the criteria in the subquery. If the value for STAFF.IDNUM is on the same row as the value **CT** in PROCLIB.STAFF (which is shown in Example 4 on page 1213), then the matching IDNUM in PROCLIB.PAYROLL is included in the output. Thus, the query returns all the employees from PROCLIB.PAYROLL who live in **CT**.

```
proc sql;
  select *
    from proclib.payroll p
   where exists (select *
                  from proclib.staff s
                 where p.idnumber=s.idnum
                   and state='CT');
```

IN condition

Tests set membership.

Featured in: Example 4 on page 1213

sql-expression <NOT> IN (query-expression | *constant* <, ... *constant*>)

Arguments

constant

is a number or a quoted character string (or other special notation) that indicates a fixed value. Constants are also called *literals*.

query-expression

is described in “query-expression” on page 1176.

sql-expression

is described in “sql-expression” on page 1182.

Details

An IN condition tests if the column value that is returned by the sql-expression on the left is a member of the set (of constants or values returned by the query-expression) on the right. The IN condition is true if the value of the left-hand operand is in the set of values that are defined by the right-hand operand.

IS condition

Tests for a missing value.

Featured in: Example 5 on page 1216

sql-expression IS <NOT> NULL | MISSING

Argument

sql-expression

is described in “sql-expression” on page 1182.

Details

IS NULL and IS MISSING are predicates that test for a missing value. IS NULL and IS MISSING are used in the WHERE, ON, and HAVING expressions. Each predicate resolves to true if the sql-expression’s result is missing and false if it is not missing.

SAS stores a numeric missing value as a period (.) and a character missing value as a blank space. Unlike missing values in some versions of SQL, missing values in SAS always appear first in the collating sequence. Therefore, in Boolean and comparison operations, the following expressions resolve to true in a predicate:

```
3>null
-3>null
0>null
```

The SAS way of evaluating missing values differs from that of the ANSI Standard for SQL. According to the Standard, these expressions are NULL. See “sql-expression” on page 1182 for more information on predicates and operators. See “PROC SQL and the ANSI Standard” on page 1204 for more information on the ANSI Standard.

joined-table

Joins a table with itself or with other tables or views.

Restrictions: Joins are limited to 32 tables.

See also: FROM Clause on page 1147 and “query-expression” on page 1176

Featured in: Example 4 on page 1213, Example 7 on page 1220, Example 9 on page 1227, Example 13 on page 1235, and Example 14 on page 1238

- ❶ *table-name* <<**AS**> *alias*>, *table-name* <<**AS**> *alias*>
<, ... *table-name* <<**AS**> *alias*>>
- ❷ <(>*table-name* <**INNER**> **JOIN** *table-name*
ON *sql-expression*<)>
- ❸ <(>*table-name* **LEFT JOIN** | **RIGHT JOIN** | **FULL JOIN**
table-name ON *sql-expression*<)>
- ❹ <(>*table-name* **CROSS JOIN** *table-name*<)>
- ❺ <(>*table-name* **UNION JOIN** *table-name*<)>
- ❻ <(>*table-name* **NATURAL**
 <**INNER** | **FULL** <**OUTER**> | **LEFT** <**OUTER**> | **RIGHT** <**OUTER**>>
JOIN *table-name*<)>

Arguments

alias

specifies an alias for *table-name*. The AS keyword is optional.

sql-expression

is described in “sql-expression” on page 1182.

table-name

can be one of the following:

- ❑ the name of a PROC SQL table.
- ❑ the name of a SAS data view or PROC SQL view.
- ❑ a query-expression. A query-expression in the FROM clause is usually referred to as an *in-line view*. See “FROM Clause” on page 1147 for more information about in-line views.
- ❑ a connection to a DBMS in the form of the CONNECTION TO component. See “CONNECTION TO” on page 1162 for more information.

table-name can be a one-level name, a two-level *libref.table* name, or a physical pathname that is enclosed in single quotation marks.

Note: If you include parentheses, then be sure to include them in pairs. Parentheses are not valid around comma joins (type ❶). △

Types of Joins

- ❶❷ Inner join. See “Inner Joins” on page 1167.
- ❸ Outer join. See “Outer Joins” on page 1169.
- ❹ Cross join. See “Cross Joins” on page 1170.
- ❺ Union join. See “Union Joins” on page 1171.
- ❻ Natural join. See “Natural Joins” on page 1171.

Joining Tables

When multiple tables, views, or query-expressions are listed in the FROM clause, they are processed to form one table. The resulting table contains data from each contributing table. These queries are referred to as *joins*.

Conceptually, when two tables are specified, each row of table A is matched with all the rows of table B to produce an internal or intermediate table. The number of rows in the intermediate table (*Cartesian product*) is equal to the product of the number of rows in each of the source tables. The intermediate table becomes the input to the rest of the query in which some of its rows may be eliminated by the WHERE clause or summarized by a summary function.

A common type of join is an *equijoin*, in which the values from a column in the first table must equal the values of a column in the second table.

Table Limit

PROC SQL can process a maximum of 32 tables for a join. If you are using views in a join, then the number of tables on which the views are based count toward the 32-table limit. Each CONNECTION TO component in the Pass-Through Facility counts as one table.

Specifying the Rows to Be Returned

The WHERE clause or ON clause contains the conditions (sql-expression) under which the rows in the Cartesian product are kept or eliminated in the result table. WHERE is used to select rows from inner joins. ON is used to select rows from inner or outer joins.

The expression is evaluated for each row from each table in the intermediate table described earlier in “Joining Tables” on page 1166. The row is considered to be matching if the result of the expression is true (a nonzero, nonmissing value) for that row.

Note: You can follow the ON clause with a WHERE clause to further subset the query result. See Example 7 on page 1220 for an example. △

Table Aliases

Table aliases are used in joins to distinguish the columns of one table from those in the other table(s). A table name or alias must be prefixed to a column name when you are joining tables that have matching column names. See FROM Clause on page 1147 for more information on table aliases.

Joining a Table with Itself

A single table can be joined with itself to produce more information. These joins are sometimes called *reflexive joins*. In these joins, the same table is listed twice in the FROM clause. Each instance of the table must have a table alias or you will not be able to distinguish between references to columns in either instance of the table. See Example 13 on page 1235 and Example 14 on page 1238 for examples.

Inner Joins

An *inner join* returns a result table for all the rows in a table that have one or more matching rows in the other table(s), as specified by the sql-expression. Inner joins can be performed on up to 32 tables in the same query-expression.

You can perform an inner join by using a list of table-names separated by commas or by using the INNER, JOIN, and ON keywords.

The LEFTTAB and RIGHTTAB tables are used to illustrate this type of join:

Left Table - LEFTTAB		
Continent	Export	Country
NA	wheat	Canada
EUR	corn	France
EUR	rice	Italy
AFR	oil	Egypt

Right Table - RIGHTTAB		
Continent	Export	Country
NA	sugar	USA
EUR	corn	Spain
EUR	beets	Belgium
ASIA	rice	Vietnam

The following example joins the LEFTTAB and RIGHTTAB tables to get the *Cartesian product* of the two tables. The Cartesian product is the result of combining every row from one table with every row from another table. You get the Cartesian product when you join two tables and do not subset them with a WHERE clause or ON clause.

```
proc sql;
  title 'The Cartesian Product of';
  title2 'LEFTTAB and RIGHTTAB';
  select *
    from lefttab, righttab;
```

The Cartesian Product of LEFTTAB and RIGHTTAB					
Continent	Export	Country	Continent	Export	Country
NA	wheat	Canada	NA	sugar	USA
NA	wheat	Canada	EUR	corn	Spain
NA	wheat	Canada	EUR	beets	Belgium
NA	wheat	Canada	ASIA	rice	Vietnam
EUR	corn	France	NA	sugar	USA
EUR	corn	France	EUR	corn	Spain
EUR	corn	France	EUR	beets	Belgium
EUR	corn	France	ASIA	rice	Vietnam
EUR	rice	Italy	NA	sugar	USA
EUR	rice	Italy	EUR	corn	Spain
EUR	rice	Italy	EUR	beets	Belgium
EUR	rice	Italy	ASIA	rice	Vietnam
AFR	oil	Egypt	NA	sugar	USA
AFR	oil	Egypt	EUR	corn	Spain
AFR	oil	Egypt	EUR	beets	Belgium
AFR	oil	Egypt	ASIA	rice	Vietnam

The LEFTTAB and RIGHTTAB tables can be joined by listing the table names in the FROM clause. The following query represents an equijoin because the values of Continent from each table are matched. The column names are prefixed with the table aliases so that the correct columns can be selected.

```
proc sql;
  title 'Inner Join';
  select *
    from lefttab as l, righttab as r
   where l.continent=r.continent;
```

Inner Join					
Continent	Export	Country	Continent	Export	Country
NA	wheat	Canada	NA	sugar	USA
EUR	corn	France	EUR	corn	Spain
EUR	corn	France	EUR	beets	Belgium
EUR	rice	Italy	EUR	corn	Spain
EUR	rice	Italy	EUR	beets	Belgium

The following PROC SQL step is equivalent to the previous one and shows how to write an equijoin using the INNER JOIN and ON keywords.

```
proc sql;
  title 'Inner Join';
  select *
    from lefttab as l inner join
      righttab as r
   on l.continent=r.continent;
```

See Example 4 on page 1213, Example 13 on page 1235, and Example 14 on page 1238 for more examples.

Outer Joins

Outer joins are inner joins that have been augmented with rows that did not match with any row from the other table in the join. The three types of outer joins are left, right, and full.

A left outer join, specified with the keywords `LEFT JOIN` and `ON`, has all the rows from the Cartesian product of the two tables for which the sql-expression is true, plus rows from the first (`LEFTTAB`) table that do not match any row in the second (`RIGHTTAB`) table.

```
proc sql;
  title 'Left Outer Join';
  select *
    from lefttab as l left join
      righttab as r
   on l.continent=r.continent;
```

Left Outer Join					
Continent	Export	Country	Continent	Export	Country
AFR	oil	Egypt			
EUR	rice	Italy	EUR	beets	Belgium
EUR	corn	France	EUR	beets	Belgium
EUR	rice	Italy	EUR	corn	Spain
EUR	corn	France	EUR	corn	Spain
NA	wheat	Canada	NA	sugar	USA

A right outer join, specified with the keywords `RIGHT JOIN` and `ON`, has all the rows from the Cartesian product of the two tables for which the sql-expression is true, plus rows from the second (`RIGHTTAB`) table that do not match any row in the first (`LEFTTAB`) table.

```
proc sql;
  title 'Right Outer Join';
  select *
    from lefttab as l right join
      righttab as r
   on l.continent=r.continent;
```

Right Outer Join					
Continent	Export	Country	Continent	Export	Country
			ASIA	rice	Vietnam
EUR	rice	Italy	EUR	beets	Belgium
EUR	rice	Italy	EUR	corn	Spain
EUR	corn	France	EUR	beets	Belgium
EUR	corn	France	EUR	corn	Spain
NA	wheat	Canada	NA	sugar	USA

A full outer join, specified with the keywords `FULL JOIN` and `ON`, has all the rows from the Cartesian product of the two tables for which the sql-expression is true, plus rows from each table that do not match any row in the other table.

```

proc sql;
  title 'Full Outer Join';
  select *
    from lefttab as l full join
      righttab as r
    on l.continent=r.continent;

```

Full Outer Join					
Continent	Export	Country	Continent	Export	Country
AFR	oil	Egypt	ASIA	rice	Vietnam
EUR	rice	Italy	EUR	beets	Belgium
EUR	rice	Italy	EUR	corn	Spain
EUR	corn	France	EUR	beets	Belgium
EUR	corn	France	EUR	corn	Spain
NA	wheat	Canada	NA	sugar	USA

See Example 7 on page 1220 for another example.

Cross Joins

A cross join returns as its result table the product of the two tables.

Using the LEFTTAB and RIGHTTAB example tables, the following program demonstrates the cross join:

```

proc sql;
  title 'Cross Join';
  select *
    from lefttab as l cross join
      righttab as r;

```

Cross Join					
Continent	Export	Country	Continent	Export	Country
NA	wheat	Canada	NA	sugar	USA
NA	wheat	Canada	EUR	corn	Spain
NA	wheat	Canada	EUR	beets	Belgium
NA	wheat	Canada	ASIA	rice	Vietnam
EUR	corn	France	NA	sugar	USA
EUR	corn	France	EUR	corn	Spain
EUR	corn	France	EUR	beets	Belgium
EUR	corn	France	ASIA	rice	Vietnam
EUR	rice	Italy	NA	sugar	USA
EUR	rice	Italy	EUR	corn	Spain
EUR	rice	Italy	EUR	beets	Belgium
EUR	rice	Italy	ASIA	rice	Vietnam
AFR	oil	Egypt	NA	sugar	USA
AFR	oil	Egypt	EUR	corn	Spain
AFR	oil	Egypt	EUR	beets	Belgium
AFR	oil	Egypt	ASIA	rice	Vietnam

The cross join is not functionally different from a cartesian product join. You would get the same result by submitting the following program:

```
proc sql;
  select *
    from lefttab, righttab;
```

Do not use an ON clause with a cross join. An ON clause will cause a cross join to fail. However, you can use a WHERE clause to subset the output.

Union Joins

A union join returns a union of the columns of both tables. The union join places in the results all rows with their respective column values from each input table. Columns that do not exist in one table will have null (missing) values for those rows in the result table. The following example demonstrates a union join.

```
proc sql;
  title 'Union Join';
  select *
    from lefttab union join righttab;
```

Union Join					
Continent	Export	Country	Continent	Export	Country
			NA	sugar	USA
			EUR	corn	Spain
			EUR	beets	Belgium
			ASIA	rice	Vietnam
NA	wheat	Canada			
EUR	corn	France			
EUR	rice	Italy			
AFR	oil	Egypt			

Using a union join is similar to concatenating tables with the OUTER UNION set operator. See “query-expression” on page 1176 for more information.

Do not use an ON clause with a union join. An ON clause will cause a union join to fail.

Natural Joins

A natural join selects rows from two tables that have equal values in columns that share the same name and the same type. An error results if two columns have the same name but different types. If *join-specification* is omitted when specifying a natural join, then INNER is implied. If no like columns are found, then a cross join is performed.

The following examples use these two tables:

table1		
x	y	z
1	2	3
2	1	8
6	5	4
2	5	6

table2		
x	b	z

1	5	3
3	5	4
2	7	8
6	0	4

The following program demonstrates a natural inner join.

```
proc sql;
  title 'Natural Inner Join';
  select *
  from table1 natural join table2;
```

Natural Inner Join			
x	z	b	y

1	3	5	2
2	8	7	1
6	4	0	5

The following program demonstrates a natural left outer join.

```
proc sql;
  title 'Natural Left Outer Join';
  select *
  from table1 natural left join table2;
```

Natural Left Outer Join			
x	z	b	y

1	3	5	2
2	6	.	5
2	8	7	1
6	4	0	5

Do not use an ON clause with a natural join. An ON clause will cause a natural join to fail. When using a natural join, an ON clause is implied, matching all like columns.

Joining More Than Two Tables

Inner joins are usually performed on two or three tables, but they can be performed on up to 32 tables in PROC SQL. A join on three tables is described here to explain how and why the relationships work among the tables.

In a three-way join, the sql-expression consists of two conditions: one relates the first table to the second table and the other relates the second table to the third table. It is possible to break this example into stages, performing a two-way join into a temporary table and then joining that table with the third one for the same result. However, PROC SQL can do it all in one step as shown in the next example.

The example shows the joining of three tables: COMM, PRICE, and AMOUNT. To calculate the total revenue from exports for each country, you need to multiply the

amount exported (AMOUNT table) by the price of each unit (PRICE table), and you must know the commodity that each country exports (COMM table).

COMM Table		
Continent	Export	Country
NA	wheat	Canada
EUR	corn	France
EUR	rice	Italy
AFR	oil	Egypt

PRICE Table	
Export	Price
rice	3.56
corn	3.45
oil	18
wheat	2.98

AMOUNT Table	
Country	Quantity
Canada	16000
France	2400
Italy	500
Egypt	10000

```
proc sql;
  title 'Total Export Revenue';
  select c.Country, p.Export, p.Price,
         a.Quantity, a.quantity*p.price
         as Total
  from comm c, price p, amount a
  where c.export=p.export
        and c.country=a.country;
```

Total Export Revenue				
Country	Export	Price	Quantity	Total
Italy	rice	3.56	500	1780
France	corn	3.45	2400	8280
Egypt	oil	18	10000	180000
Canada	wheat	2.98	16000	47680

See Example 9 on page 1227 for another example.

Comparison of Joins and Subqueries

You can often use a subquery or a join to get the same result. However, it is often more efficient to use a join if the outer query and the subquery do not return duplicate rows. For example, the following queries produce the same result. The second query is more efficient:

```
proc sql;
  select IDNumber, Birth
    from proclib.payroll
   where IDNumber in (select idnum
                      from proclib.staff
                      where lname like 'B%');

proc sql;
  select  p.IDNumber, p.Birth
    from proclib.payroll p, proclib.staff s
   where p.idnumber=s.idnum
        and s.lname like 'B%';
```

Note: PROCLIB.PAYROLL is shown in Example 2 on page 1209. △

LIKE condition

Tests for a matching pattern.

sql-expression <NOT> **LIKE** sql-expression <ESCAPE *character-expression*>

Arguments

sql-expression

is described in “sql-expression” on page 1182.

character-expression

is an sql-expression that evaluates to a single character. The operands of *character-expression* must be character or string literals; they cannot be column names.

Note: If you use an ESCAPE clause, then the pattern-matching specification must be a quoted string or quoted concatenated string; it cannot contain column names. △

Details

The LIKE condition selects rows by comparing character strings with a pattern-matching specification. It resolves to true and displays the matched string(s) if the left operand matches the pattern specified by the right operand.

The ESCAPE clause is used to search for literal instances of the percent (%) and underscore () characters, which are usually used for pattern matching.

Patterns for Searching

Patterns are composed of three classes of characters:

underscore (`_`)
 matches any single character.

percent sign (`%`)
 matches any sequence of zero or more characters.

any other character
 matches that character.

These patterns can appear before, after, or on both sides of characters that you want to match. The LIKE condition is case-sensitive.

The following list uses these values: **Smith**, **Smooth**, **Smothers**, **Smart**, and **Smuggle**.

`'Sm%'`
 matches **Smith**, **Smooth**, **Smothers**, **Smart**, **Smuggle**.

`'%th'`
 matches **Smith**, **Smooth**.

`'S__gg%'`
 matches **Smuggle**.

`'S_o'`
 matches a three-letter word, so it has no matches here.

`'S_o%'`
 matches **Smooth**, **Smothers**.

`'S%th'`
 matches **Smith**, **Smooth**.

`'z'`
 matches the single, uppercase character **z** only, so it has no matches here.

Searching for Literal % and _

Because the `%` and `_` characters have special meaning in the context of the LIKE condition, you must use the ESCAPE clause to search for these character literals in the input character string.

These example use the values **app**, **a_**, **a__**, **bbaa1**, and **ba_1**.

- The condition **like** `'a_ %'` matches **app**, **a_**, and **a__**, because the underscore (`_`) in the search pattern matches any single character (including the underscore), and the percent (`%`) in the search pattern matches zero or more characters, including `' %'` and `'_'`.
- The condition **like** `'a_^ %'` **escape** `'^'` matches only **a_**, because the escape character (`^`) specifies that the pattern search for a literal `' %'`.
- The condition **like** `'a_ %'` **escape** `'_'` matches none of the values, because the escape character (`_`) specifies that the pattern search for an `'a'` followed by a literal `' %'`, which does not apply to any of these values.

Searching for Mixed-Case Strings

To search for mixed-case strings, use the UPCASE function to make all the names uppercase before entering the LIKE condition:

```
uppercase(name) like 'SM%';
```

Note: When you are using the `%` character, be aware of the effect of trailing blanks. You may have to use the TRIM function to remove trailing blanks in order to match values. △

LOWER function

Converts the case of a character string to lowercase.

See also: “UPPER function” on page 1197

LOWER (sql-expression)

Argument

sql-expression

must resolve to a character string and is described in “sql-expression” on page 1182.

Details

The LOWER function operates on character strings. LOWER changes the case of its argument to all lowercase.

Note: The LOWER function is provided for compatibility with the ANSI SQL standard. You can also use the SAS function LOWCASE. △

query-expression

Retrieves data from tables.

See also: “table-expression” on page 1196, “Query Expressions (Subqueries)” on page 1185, and “In-Line Views” on page 1148

table-expression <set-operator table-expression> <...set-operator table-expression>

Arguments

table-expression

is described in “table-expression” on page 1196.

set-operator

is one of the following:

INTERSECT <CORRESPONDING> <ALL>

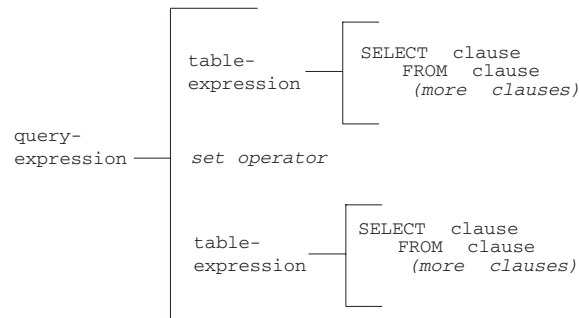
OUTER UNION <CORRESPONDING>

UNION <CORRESPONDING> <ALL>

EXCEPT <CORRESPONDING> <ALL>

Query Expressions and Table Expressions

A query-expression is one or more table-expressions. Multiple table expressions are linked by set operators. The following figure illustrates the relationship between table-expressions and query-expressions.



Set Operators

PROC SQL provides these set operators:

OUTER UNION

concatenates the query results.

UNION

produces all unique rows from both queries.

EXCEPT

produces rows that are part of the first query only.

INTERSECT

produces rows that are common to both query results.

A query-expression with set operators is evaluated as follows.

- Each table-expression is evaluated to produce an (internal) intermediate result table.
- Each intermediate result table then becomes an operand linked with a set operator to form an expression, for example, A UNION B.
- If the query-expression involves more than two table-expressions, then the result from the first two becomes an operand for the next set operator and operand, such as (A UNION B) EXCEPT C, ((A UNION B) EXCEPT C) INTERSECT D, and so on.
- Evaluating a query-expression produces a single output table.

Set operators follow this order of precedence unless they are overridden by parentheses in the expression(s): INTERSECT is evaluated first. OUTER UNION, UNION, and EXCEPT have the same level of precedence.

PROC SQL performs set operations even if the tables or views that are referred to in the table-expressions do not have the same number of columns. The reason for this behavior is that the ANSI Standard for SQL requires that tables or views that are involved in a set operation have the same number of columns and that the columns have matching data types. If a set operation is performed on a table or view that has fewer columns than the one(s) with which it is being linked, then PROC SQL extends the table or view with fewer columns by creating columns with missing values of the appropriate data type. This temporary alteration enables the set operation to be performed correctly.

CORRESPONDING (CORR) Keyword

The CORRESPONDING keyword is used only when a set operator is specified. CORR causes PROC SQL to match the columns in table-expressions *by name* and not

by ordinal position. Columns that do not match by name are excluded from the result table, except for the OUTER UNION operator. See “OUTER UNION” on page 1178.

For example, when performing a set operation on two table-expressions, PROC SQL matches the first specified column-name (listed in the SELECT clause) from one table-expression with the first specified column-name from the other. If CORR is omitted, then PROC SQL matches the columns by ordinal position.

ALL Keyword

The set operators automatically eliminate duplicate rows from their output tables. The optional ALL keyword preserves the duplicate rows, reduces the execution by one step, and thereby improves the query-expression’s performance. You use it when you want to display all the rows resulting from the table-expressions, rather than just the rows that are output because duplicates have been deleted. The ALL keyword is used only when a set operator is also specified.

OUTER UNION

Performing an OUTER UNION is very similar to performing the SAS DATA step with a SET statement. The OUTER UNION concatenates the intermediate results from the table-expressions. Thus, the result table for the query-expression contains all the rows produced by the first table-expression followed by all the rows produced by the second table-expression. Columns with the same name are in separate columns in the result table.

For example, the following query expression concatenates the ME1 and ME2 tables but does not overlay like-named columns. Output 40.1 on page 1178 shows the result.

```
proc sql;
  title 'ME1 and ME2: OUTER UNION';
  select *
    from me1
  outer union
  select *
    from me2;
```

ME1			
IDnum	Jobcode	Salary	Bonus
1400	ME1	29769	587
1403	ME1	28072	342
1120	ME1	28619	986
1120	ME1	28619	986

ME2		
IDnum	Jobcode	Salary
1653	ME2	35108
1782	ME2	35345
1244	ME2	36925

Output 40.1 OUTER UNION of ME1 and ME2 Tables

ME1 and ME2: OUTER UNION						
IDnum	Jobcode	Salary	Bonus	IDnum	Jobcode	Salary
1400	ME1	29769	587			.
1403	ME1	28072	342			.
1120	ME1	28619	986			.
1120	ME1	28619	986			.
		.	.	1653	ME2	35108
		.	.	1782	ME2	35345
		.	.	1244	ME2	36925

Concatenating tables with the OUTER UNION set operator is similar to performing a union join. See “Union Joins” on page 1171 for more information.

To overlay columns with the same name, use the CORRESPONDING keyword.

```
proc sql;
  title 'ME1 and ME2: OUTER UNION CORRESPONDING';
  select *
    from me1
  outer union corr
  select *
    from me2;
```

ME1 and ME2: OUTER UNION CORRESPONDING				
IDnum	Jobcode	Salary	Bonus	
1400	ME1	29769	587	
1403	ME1	28072	342	
1120	ME1	28619	986	
1120	ME1	28619	986	
1653	ME2	35108	.	
1782	ME2	35345	.	
1244	ME2	36925	.	

In the resulting concatenated table, notice the following:

- OUTER UNION CORRESPONDING retains all nonmatching columns.
- For columns with the same name, if a value is missing from the result of the first table-expression, then the value in that column from the second table-expression is inserted.
- The ALL keyword is not used with OUTER UNION because this operator’s default action is to include all rows in a result table. Thus, both rows from the table ME1 where IDnum is **1120** appear in the output.

UNION

The UNION operator produces a table that contains all the unique rows that result from both table-expressions. That is, the output table contains rows produced by the first table-expression, the second table-expression, or both.

Columns are appended by position in the tables, regardless of the column names. However, the data type of the corresponding columns must match or the union will not occur. PROC SQL issues a warning message and stops executing.

The names of the columns in the output table are the names of the columns from the first table-expression unless a column (such as an expression) has no name in the first table-expression. In such a case, the name of that column in the output table is the name of the respective column in the second table-expression.

In the following example, PROC SQL combines the two tables:

```
proc sql;
  title 'ME1 and ME2: UNION';
  select *
    from me1
  union
  select *
    from me2;
```

ME1 and ME2: UNION			
IDnum	Jobcode	Salary	Bonus
1120	ME1	28619	986
1244	ME2	36925	.
1400	ME1	29769	587
1403	ME1	28072	342
1653	ME2	35108	.
1782	ME2	35345	.

In the following example, ALL includes the duplicate row from ME1. In addition, ALL changes the sorting by specifying that PROC SQL make one pass only. Thus, the values from ME2 are simply appended to the values from ME1.

```
proc sql;
  title 'ME1 and ME2: UNION ALL';
  select *
    from me1
  union all
  select *
    from me2;
```

ME1 and ME2: UNION ALL			
IDnum	Jobcode	Salary	Bonus
1400	ME1	29769	587
1403	ME1	28072	342
1120	ME1	28619	986
1120	ME1	28619	986
1653	ME2	35108	.
1782	ME2	35345	.
1244	ME2	36925	.

See Example 5 on page 1216 for another example.

EXCEPT

The EXCEPT operator produces (from the first table-expression) an output table that has unique rows that are not in the second table-expression. If the intermediate result from the first table-expression has at least one occurrence of a row that is not in the intermediate result of the second table-expression, then that row (from the first table-expression) is included in the result table.

In the following example, the IN_USA table contains flights to cities within and outside the USA. The OUT_USA table contains flights only to cities outside the USA. This example returns only the rows from IN_USA that are not also in OUT_USA:

```
proc sql;
  title 'Flights from IN_USA Only';
  select * from in_usa
  except
  select * from out_usa;
```

IN_USA	
Flight	Dest

145	ORD
156	WAS
188	LAX
193	FRA
207	LON

OUT_USA	
Flight	Dest

193	FRA
207	LON
311	SJA

Flights from IN_USA Only	
Flight	Dest

145	ORD
156	WAS
188	LAX

INTERSECT

The INTERSECT operator produces an output table that has rows that are common to both tables. For example, using the IN_USA and OUT_USA tables shown above, the following example returns rows that are in both tables:

```
proc sql;
  title 'Flights from Both IN_USA and OUT_USA';
  select * from in_usa
  intersect
  select * from out_usa;
```

Flights from Both IN_USA and OUT_USA	
Flight	Dest
193	FRA
207	LON

sql-expression

Produces a value from a sequence of operands and operators.

operand operator operand

Arguments

operand

is one of the following:

- a *constant*, which is a number or a quoted character string (or other special notation) that indicates a fixed value. Constants are also called *literals*. Constants are described in *SAS Language Reference: Dictionary*.
- a column-name, which is described in “column-name” on page 1161.
- a SAS function, which is any SAS function except LAG, DIF, and SOUND. Functions are described in *SAS Language Reference: Dictionary*.
- the ANSI SQL functions COALESCE, BTRIM, LOWER, UPPER, and SUBSTRING.
- a summary-function, which is described in “summary-function” on page 1190.
- a query-expression, which is described in “query-expression” on page 1176.
- the USER literal, which references the userid of the person who submitted the program. The userid that is returned is operating environment-dependent, but PROC SQL uses the same value that the &SYSJOBID macro variable has on the operating environment.

operator

is described in “Operators and the Order of Evaluation” on page 1183.

Note: SAS functions, including summary functions, can stand alone as SQL expressions. For example

```
select min(x) from table;
```

```
select scan(y,4) from table;
```

△

SAS Functions

PROC SQL supports the same SAS functions as the DATA step, except for the functions LAG, DIF, and SOUND. For example, the SCAN function is used in the following query:


```
select style, scan(street,1) format=$15.
from houses;
```

See *SAS Language Reference: Dictionary* for complete documentation on SAS functions. Summary functions are also SAS functions. See “summary-function” on page 1190 for more information.

USER Literal

USER can be specified in a view definition, for example, to create a view that restricts access to those in the user’s department:

```
create view myemp as
select * from dept12.employees
where manager=user;
```

This view produces a different set of employee information for each manager who references it.

Operators and the Order of Evaluation

The order in which operations are evaluated is the same as in the DATA step with this one exception: NOT is grouped with the logical operators AND and OR in PROC SQL; in the DATA step, NOT is grouped with the unary plus and minus signs.

Unlike missing values in some versions of SQL, missing values in SAS always appear first in the collating sequence. Therefore, in Boolean and comparison operations, the following expressions resolve to true in a predicate:

```
3>null
-3>null
0>null
```

You can use parentheses to group values or to nest mathematical expressions. Parentheses make expressions easier to read and can also be used to change the order of evaluation of the operators. Evaluating expressions with parentheses begins at the deepest level of parentheses and moves outward. For example, SAS evaluates $A+B*C$ as $A+(B*C)$, although you can add parentheses to make it evaluate as $(A+B)*C$ for a different result.

Higher priority operations are performed first: that is, group 0 operators are evaluated before group 5 operators. The following table shows the operators and their order of evaluation, including their priority groups.

Table 40.1 Operators and Order of Evaluation

Group	Operator	Description
0	()	forces the expression enclosed to be evaluated first
1	case-expression	selects result values that satisfy specified conditions
2	**	raises to a power
	unary +, unary -	indicates a positive or negative number
3	*	multiplies
	/	divides
4	+	adds
	-	subtracts

Group	Operator	Description
5		concatenates
6	<NOT> BETWEEN condition	See “BETWEEN condition” on page 1155.
	<NOT> CONTAINS condition	see “CONTAINS condition” on page 1162.
	<NOT> EXISTS condition	See “EXISTS condition” on page 1163.
	<NOT> IN condition	See “IN condition” on page 1163.
	IS <NOT> condition	See “IS condition” on page 1164.
	<NOT> LIKE condition	See “LIKE condition” on page 1174.
7	=, eq	equals
	≠, ^=, < >, ne	does not equal
	>, gt	is greater than
	<, lt	is less than
	>=, ge	is greater than or equal to
	<=, le	is less than or equal to
	=*	sounds like (use with character operands only). See Example 11 on page 1231.
	eqt	equal to truncated strings (use with character operands only). See “Truncated String Comparison Operators” on page 1184.
	gtt	greater than truncated strings
	ltt	less than truncated strings
	get	greater than or equal to truncated strings
	let	less than or equal to truncated strings
	net	not equal to truncated strings
8	&, AND	indicates logical AND
9	, OR	indicates logical OR
10	¬, ^, NOT	indicates logical NOT

Symbols for operators might vary, depending on your operating environment. See *SAS Language Reference: Dictionary* for more information on operators and expressions.

Truncated String Comparison Operators

PROC SQL supports truncated string comparison operators (see Group 7 in Table 40.1 on page 1183). In a truncated string comparison, the comparison is performed after making the strings the same length by truncating the longer string to be the same length as the shorter string. For example, the expression `'TWOSTORY' eqt 'TWO'` is true because the string "TWOSTORY" is reduced to "TWO" before the comparison is performed. Note that the truncation is performed internally; neither operand is permanently changed.

Note: Unlike the DATA step, PROC SQL does not support the colon operators (such as =:, >:, and <=:) for truncated string comparisons. Use the alphabetic operators (such as EQT, GTT, and LET). △

Query Expressions (Subqueries)

A query-expression is called a *subquery* when it is used in a WHERE or HAVING clause. A subquery is a query-expression that is nested as part of another query-expression. A subquery selects one or more rows from a table based on values in another table.

Depending on the clause that contains it, a subquery can return a single value or multiple values. If more than one subquery is used in a query-expression, then the innermost query is evaluated first, then the next innermost query, and so on, moving outward.

PROC SQL allows a subquery (contained in parentheses) at any point in an expression where a simple column value or constant can be used. In this case, a subquery must return a *single value*, that is, one row with only one column.

The following is an example of a subquery that returns one value. This PROC SQL step subsets the PROCLIB.PAYROLL table based on information in the PROCLIB.STAFF table. (PROCLIB.PAYROLL is shown in Example 2 on page 1209, and PROCLIB.STAFF is shown in Example 4 on page 1213.) PROCLIB.PAYROLL contains employee identification numbers (IdNumber) and their salaries (Salary) but does not contain their names. If you want to return only the row from PROCLIB.PAYROLL for one employee, then you can use a subquery that queries the PROCLIB.STAFF table, which contains the employees' identification numbers and their names (Lname and Fname).

```
options ls=64 nodate nonumber;
proc sql;
  title 'Information for Earl Bowden';
  select *
    from proclib.payroll
   where idnumber=
      (select idnum
        from proclib.staff
       where upcase(lname)='BOWDEN');
```

Information for Earl Bowden					
Id Number	Gender	Jobcode	Salary	Birth	Hired
1403	M	ME1	28072	28JAN69	21DEC91

Subqueries can return *multiple values*. The following example uses the tables PROCLIB.DELAY and PROCLIB.MARCH. These tables contain information about the same flights and have the Flight column in common. The following subquery returns all the values for Flight in PROCLIB.DELAY for international flights. The values from the subquery complete the WHERE clause in the outer query. Thus, when the outer query is executed, only the international flights from PROCLIB.MARCH are in the output.

```
options ls=64 nodate nonumber;
proc sql outobs=5;
  title 'International Flights from';
  title2 'PROCLIB.MARCH';
  select Flight, Date, Dest, Boarded
    from proclib.march
```

```

where flight in
(select flight
 from proclib.delay
 where destype='International');

```

International Flights from PROCLIB.MARCH			
Flight	Date	Dest	Boarded
219	01MAR94	LON	198
622	01MAR94	FRA	207
132	01MAR94	YYZ	115
271	01MAR94	PAR	138
219	02MAR94	LON	147

Sometimes it is helpful to compare a value with a set of values returned by a subquery. The keywords ANY or ALL can be specified before a subquery when the subquery is the right-hand operand of a comparison. If ALL is specified, then the comparison is true only if it is true for all values that are returned by the subquery. If a subquery returns no rows, then the result of an ALL comparison is true for each row of the outer query.

If ANY is specified, then the comparison is true if it is true for any one of the values that are returned by the subquery. If a subquery returns no rows, then the result of an ANY comparison is false for each row of the outer query.

The following example selects all those in PROCLIB.PAYROLL who earn more than the highest paid **ME3**:

```

options ls=64 nodate nonumber ;
proc sql;
title  ''Employees who Earn More than'';
title2  ''All ME's'';
select *
  from proclib.payroll
 where salary > all (select salary
                    from proclib.payroll
                    where jobcode='ME3');

```

Employees who Earn More than All ME's					
Id Number	Gender	Jobcode	Salary	Birth	Hired
1333	M	PT2	88606	30MAR61	10FEB81
1739	M	PT1	66517	25DEC64	27JAN91
1428	F	PT1	68767	04APR60	16NOV91
1404	M	PT2	91376	24FEB53	01JAN80
1935	F	NA2	51081	28MAR54	16OCT81
1905	M	PT1	65111	16APR72	29MAY92
1407	M	PT1	68096	23MAR69	18MAR90
1410	M	PT2	84685	03MAY67	07NOV86
1439	F	PT1	70736	06MAR64	10SEP90
1545	M	PT1	66130	12AUG59	29MAY90
1106	M	PT2	89632	06NOV57	16AUG84
1442	F	PT2	84536	05SEP66	12APR88
1417	M	NA2	52270	27JUN64	07MAR89
1478	M	PT2	84203	09AUG59	24OCT90
1556	M	PT1	71349	22JUN64	11DEC91
1352	M	NA2	53798	02DEC60	16OCT86
1890	M	PT2	91908	20JUL51	25NOV79
1107	M	PT2	89977	09JUN54	10FEB79
1830	F	PT2	84471	27MAY57	29JAN83
1928	M	PT2	89858	16SEP54	13JUL90
1076	M	PT1	66558	14OCT55	03OCT91

Note: See the first item in “Subqueries and Efficiency” on page 1188 for a note about efficiency when using ALL. △

In order to visually separate a subquery from the rest of the query, you can enclose the subquery in any number of pairs of parentheses.

Correlated Subqueries

In a correlated subquery, the WHERE expression in a subquery refers to values in a table in the outer query. The correlated subquery is evaluated for each row in the outer query. With correlated subqueries, PROC SQL executes the subquery and the outer query together.

The following example uses the PROCLIB.DELAY and PROCLIB.MARCH tables. A DATA step (“PROCLIB.DELAY” on page 1642) creates PROCLIB.DELAY. PROCLIB.MARCH is shown in Example 13 on page 1235. PROCLIB.DELAY has the Flight, Date, Orig, and Dest columns in common with PROCLIB.MARCH:

```
proc sql outobs=5;
  title 'International Flights';
  select *
    from proclib.march
   where 'International' in
      (select destype
        from proclib.delay
       where march.Flight=delay.Flight);
```

The subquery resolves by substituting every value for MARCH.Flight into the subquery's WHERE clause, one row at a time. For example, when MARCH.Flight=219, the subquery resolves as follows:

- 1 PROC SQL retrieves all the rows from DELAY where Flight=219 and passes their DESTYPE values to the WHERE clause.
- 2 PROC SQL uses the DESTYPE values to complete the WHERE clause:

```
where 'International' in
      ('International','International', ...)
```

- 3 The WHERE clause checks to see if **International** is in the list. Because it is, all rows from MARCH that have a value of 219 for Flight become part of the output.

The following output contains the rows from MARCH for international flights only.

Output 40.2 International Flights for March

International Flights							
Flight	Date	Depart	Orig	Dest	Miles	Boarded	Capacity
219	01MAR94	9:31	LGA	LON	3442	198	250
622	01MAR94	12:19	LGA	FRA	3857	207	250
132	01MAR94	15:35	LGA	YYZ	366	115	178
271	01MAR94	13:17	LGA	PAR	3635	138	250
219	02MAR94	9:31	LGA	LON	3442	147	250

Subqueries and Efficiency

- Use the MAX function in a subquery instead of the ALL keyword before the subquery. For example, the following queries produce the same result, but the second query is more efficient:

```
proc sql;
  select * from proclib.payroll
  where salary > all(select salary
                    from proclib.payroll
                    where jobcode='ME3');
```

```
proc sql;
  select * from proclib.payroll
  where salary > (select max(salary)
                 from proclib.payroll
                 where jobcode='ME3');
```

- With subqueries, use IN instead of EXISTS when possible. For example, the following queries produce the same result, but the second query is usually more efficient:

```
proc sql;
  select *
  from proclib.payroll p
  where exists (select *
               from staff s
               where p.idnum=s.idnum
                  and state='CT');
```

```

proc sql;
  select *
    from proclib.payroll
   where idnum in (select idnum
                  from staff
                  where state='CT');

```

SUBSTRING function

Returns a part of a character expression.

SUBSTRING (sql-expression FROM *start* <FOR *length*>)

- sql-expression must be a character string and is described in “sql-expression” on page 1182.
- *start* is a number (not a variable or column name) that specifies the position, counting from the left end of the character string, at which to begin extracting the substring.
- *length* is a number (not a variable or column name) that specifies the length of the substring that is to be extracted.

Details

The SUBSTRING function operates on character strings. SUBSTRING returns a specified part of the input character string, beginning at the position that is specified by *start*. If *length* is omitted, then the SUBSTRING function returns all characters from *start* to the end of the input character string. The values of *start* and *length* must be numbers (not variables) and can be positive, negative, or zero.

If *start* is greater than the length of the input character string, then the SUBSTRING function returns a zero-length string.

If *start* is less than 1, then the SUBSTRING function begins extraction at the beginning of the input character string.

If *length* is specified, then the sum of *start* and *length* cannot be less than *start* or an error is returned. If the sum of *start* and *length* is greater than the length of the input character string, then the SUBSTRING function returns all characters from *start* to the end of the input character string. If the sum of *start* and *length* is less than 1, then the SUBSTRING function returns a zero-length string.

Note: The SUBSTRING function is provided for compatibility with the ANSI SQL standard. You can also use the SAS function SUBSTR. Δ

summary-function

Performs statistical summary calculations.

Restriction: A summary function cannot appear in an ON clause or a WHERE clause.

See also: GROUP BY on page 1149, HAVING Clause on page 1150, SELECT Clause on page 1142, and “table-expression” on page 1196

Featured in: Example 8 on page 1224, Example 12 on page 1233, and Example 15 on page 1240

summary-function (<DISTINCT | ALL> sql-expression)

Arguments

summary-function

is one of the following:

AVG|MEAN

arithmetic mean or average of values

COUNT|FREQ|N

number of nonmissing values

CSS

corrected sum of squares

CV

coefficient of variation (percent)

MAX

largest value

MIN

smallest value

NMISS

number of missing values

PRT

probability of a greater absolute value of Student's t

RANGE

range of values

STD

standard deviation

STDERR

standard error of the mean

SUM

sum of values

SUMWGT
sum of the WEIGHT variable values*

T
Student's t value for testing the hypothesis that the population mean is zero

USS
uncorrected sum of squares

VAR
variance
For a description and the formulas used for these statistics, see Appendix 1, "SAS Elementary Statistics Procedures," on page 1577.

DISTINCT
specifies that only the unique values of sql-expression be used in the calculation.

ALL
specifies that all values of sql-expression be used in the calculation. If neither DISTINCT nor ALL is specified, then ALL is used.

sql-expression
is described in "sql-expression" on page 1182.

Summarizing Data

Summary functions produce a statistical summary of the entire table or view that is listed in the FROM clause or for each group that is specified in a GROUP BY clause. If GROUP BY is omitted, then all the rows in the table or view are considered to be a single group. These functions reduce all the values in each row or column in a table to one *summarizing* or *aggregate* value. For this reason, these functions are often called *aggregate functions*. For example, the sum (one value) of a column results from the addition of all the values in the column.

Counting Rows

The COUNT function counts rows. COUNT(*) returns the total number of rows in a group or in a table. If you use a column name as an argument to COUNT, then the result is the total number of rows in a group or in a table that have a nonmissing value for that column. If you want to count the unique values in a column, then specify COUNT(DISTINCT *column*).

If the SELECT clause of a table-expression contains one or more summary functions and that table-expression resolves to no rows, then the summary function results are missing values. The following are exceptions that return zeros:

COUNT(*)
COUNT(<DISTINCT> sql-expression)
NMISS(<DISTINCT> sql-expression)

See Example 8 on page 1224 and Example 15 on page 1240 for examples.

Calculating Statistics Based on the Number of Arguments

The number of arguments that is specified in a summary function affects how the calculation is performed. If you specify a single argument, then the values in the

* Currently, there is no way to designate a WEIGHT variable for a table in PROC SQL. Thus, each row (or observation) has a weight of 1.

column are calculated. If you specify multiple arguments, then the arguments or columns that are listed are calculated for each row. For example, consider calculations on the following table.

```
proc sql;
  title 'Summary Table';
  select * from summary;
```

Summary Table			
	X	Y	Z
	1	3	4
	2	4	5
	8	9	4
	4	5	4

If you use one argument in the function, then the calculation is performed on that column only. If you use more than one argument, then the calculation is performed on each row of the specified columns. In the following PROC SQL step, the MIN and MAX functions return the minimum and maximum of the columns they are used with. The SUM function returns the sum of each row of the columns specified as arguments:

```
proc sql;
  select min(x) as Colmin_x,
         min(y) as Colmin_y,
         max(z) as Colmax_z,
         sum(x,y,z) as Rowsum
  from summary;
```

Summary Table			
Colmin_x	Colmin_y	Colmax_z	Rowsum
1	3	5	8
1	3	5	11
1	3	5	21
1	3	5	13

Remerging Data

When you use a summary function in a SELECT clause or a HAVING clause, you might see the following message in the SAS log:

```
NOTE: The query requires remerging summary
       statistics back with the original
       data.
```

The process of *remerging* involves two passes through the data. On the first pass, PROC SQL

- calculates and returns the value of summary functions. It then uses the result to calculate the arithmetic expressions in which the summary function participates.
- groups data according to the GROUP BY clause.

On the second pass, PROC SQL retrieves any additional columns and rows that it needs to show in the output.

The following examples use the PROCLIB.PAYROLL table (shown in Example 2 on page 1209) to show when remerging of data is and is not necessary.

The first query requires remerging. The first pass through the data groups the data by Jobcode and resolves the AVG function for each group. However, PROC SQL must make a second pass in order to retrieve the values of IdNumber and Salary.

```
proc sql outobs=10;
  title 'Salary Information';
  title2 '(First 10 Rows Only)';
  select IdNumber, Jobcode, Salary,
         avg(salary) as AvgSalary
  from proclib.payroll
  group by jobcode;
```

Salary Information (First 10 Rows Only)				
Id Number	Jobcode	Salary	AvgSalary	
1704	BCK	25465	25794.22	
1677	BCK	26007	25794.22	
1383	BCK	25823	25794.22	
1845	BCK	25996	25794.22	
1100	BCK	25004	25794.22	
1663	BCK	26452	25794.22	
1673	BCK	25477	25794.22	
1389	BCK	25028	25794.22	
1834	BCK	26896	25794.22	
1132	FA1	22413	23039.36	

You can change the previous query to return only the average salary for each jobcode. The following query does not require remerging because the first pass of the data does the summarizing and the grouping. A second pass is not necessary.

```
proc sql outobs=10;
  title 'Average Salary for Each Jobcode';
  select Jobcode, avg(salary) as AvgSalary
  from proclib.payroll
  group by jobcode;
```

Average Salary for Each Jobcode

Jobcode	AvgSalary

BCK	25794.22
FA1	23039.36
FA2	27986.88
FA3	32933.86
ME1	28500.25
ME2	35576.86
ME3	42410.71
NA1	42032.2
NA2	52383
PT1	67908

When you use the HAVING clause, PROC SQL may have to remerge data to resolve the HAVING expression.

First, consider a query that uses HAVING but that does not require remerging. The query groups the data by values of Jobcode, and the result contains one row for each value of Jobcode and summary information for people in each Jobcode. On the first pass, the summary functions provide values for the **Number**, **Average Age**, and **Average Salary** columns. The first pass provides everything that PROC SQL needs to resolve the HAVING clause, so no remerging is necessary.

```
proc sql outobs=10;
title 'Summary Information for Each Jobcode';
title2 '(First 10 Rows Only)';
select Jobcode,
       count(jobcode) as number
       label='Number',
       avg(int((today()-birth)/365.25))
       as avgage format=2.
       label='Average Age',
       avg(salary) as avgsal format=dollar8.
       label='Average Salary'
from proclib.payroll
group by jobcode
having avgage ge 30;
```

Summary Information for Each Jobcode
(First 10 Rows Only)

Jobcode	Number	Average Age	Average Salary

BCK	9	36	\$25,794
FA1	11	33	\$23,039
FA2	16	37	\$27,987
FA3	7	39	\$32,934
ME1	8	34	\$28,500
ME2	14	39	\$35,577
ME3	7	42	\$42,411
NA1	5	30	\$42,032
NA2	3	42	\$52,383
PT1	8	38	\$67,908

In the following query, PROC SQL remerges the data because the HAVING clause uses the SALARY column in the comparison and SALARY is not in the GROUP BY clause.

```
proc sql outobs=10;
title 'Employees who Earn More than the';
title2 'Average for Their Jobcode';
title3 '(First 10 Rows Only)';
select Jobcode, Salary,
       avg(salary) as AvgSalary
from proclib.payroll
group by jobcode
having salary > AvgSalary;
```

Employees who Earn More than the Average for Their Jobcode (First 10 Rows Only)		
Jobcode	Salary	AvgSalary
BCK	26007	25794.22
BCK	25823	25794.22
BCK	25996	25794.22
BCK	26452	25794.22
BCK	26896	25794.22
FA1	23177	23039.36
FA1	23738	23039.36
FA1	23979	23039.36
FA1	23916	23039.36
FA1	23644	23039.36

Keep in mind that PROC SQL remerges data when

- the values returned by a summary function are used in a calculation. For example, the following query returns the values of X and the percent of the total for each row. On the first pass, PROC SQL computes the sum of X, and on the second pass PROC SQL computes the percentage of the total for each value of X:

```
proc sql;
title 'Percentage of the Total';
select X, (100*x/sum(X)) as Pct_Total
from summary;
```

Percentage of the Total	
x	Pct_Total
32	14.81481
86	39.81481
49	22.68519
49	22.68519

- the values returned by a summary function are compared to values of a column that is not specified in the GROUP BY clause. For example, the following query uses the PROCLIB.PAYROLL table. PROC SQL remerges data because the column Salary is not specified in the GROUP BY clause:

```
proc sql;
  select  jobcode, salary,
          avg(salary) as avsal
  from    proclib.payroll
  group by jobcode
  having  salary > avsal;
```

- a column from the input table is specified in the SELECT clause and is not specified in the GROUP BY clause. This rule does not refer to columns used as arguments to summary functions in the SELECT clause.

For example, in the following query, the presence of IdNumber in the SELECT clause causes PROC SQL to remerge the data because IdNumber is not involved in grouping or summarizing during the first pass. In order for PROC SQL to retrieve the values for IdNumber, it must make a second pass through the data.

```
proc sql;
  select IdNumber, jobcode,
          avg(salary) as avsal
  from    proclib.payroll
  group by jobcode;
```

table-expression

Defines part or all of a query-expression.

See also: “query-expression” on page 1176

SELECT <DISTINCT> *object-item*<, ... *object-item*>

<INTO :*macro-variable-specification*
<, ... :*macro-variable-specification*>>

FROM *from-list*

<**WHERE** *sql-expression*>

<**GROUP BY** *group-by-item* <, ... *group-by-item*>>

<**HAVING** *sql-expression*>

See “SELECT Statement” on page 1142 for complete information on the SELECT statement.

Details

A table-expression is a SELECT statement. It is the fundamental building block of most SQL procedure statements. You can combine the results of multiple table-expressions with set operators, which creates a query-expression. Use one ORDER BY clause for an entire query-expression. Place a semicolon only at the end of the entire query-expression. A query-expression is often only one SELECT statement or table-expression.

UPPER function

Converts the case of a character string to uppercase.

See also: “LOWER function” on page 1176

UPPER (sql-expression)

- sql-expression must be a character string and is described in “sql-expression” on page 1182.

Details

The UPPER function operates on character strings. UPPER converts the case of its argument to all uppercase.

Concepts: SQL Procedure

Using SAS Data Set Options with PROC SQL

In PROC SQL, you can apply most of the SAS data set options, such as KEEP= and DROP=, to tables or SAS/ACCESS views any time that you specify a table or SAS/ACCESS view. In the SQL procedure, SAS data set options that are separated by spaces are enclosed in parentheses, and they follow immediately after the table or SAS/ACCESS view name. In the following PROC SQL step, RENAME= renames LNAME to LASTNAME for the STAFF1 table. OBS= restricts the number of rows written to STAFF1 to 15:

```
proc sql;
  create table
    staff1(rename=(lname=lastname)) as
  select *
    from staff(obs=15);
```

SAS data set options can be combined with SQL statement arguments:

```
proc sql;
  create table test
    (a character, b numeric, pw=cat);
  create index staffidx on
    staff1 (lastname, alter=dog);
```

You cannot use SAS data set options with DICTIONARY tables because DICTIONARY tables are read-only objects.

The only SAS data set options that you can use with PROC SQL views are those that assign and provide SAS passwords: READ=, WRITE=, ALTER=, and PW=.

See *SAS Language Reference: Dictionary* for a description of SAS data set options.

Connecting to a DBMS Using the SQL Procedure Pass-Through Facility

The SQL Procedure Pass-Through Facility enables you to send DBMS-specific SQL statements directly to a DBMS for execution. The Pass-Through Facility uses a SAS/ACCESS interface engine to connect to the DBMS. Therefore, you must have SAS/ACCESS software installed for your DBMS.

You submit SQL statements that are DBMS-specific. For example, you pass Transact-SQL statements to a SYBASE database. The Pass-Through Facility's basic syntax is the same for all the DBMSs. Only the statements that are used to connect to the DBMS and the SQL statements are DBMS-specific.

With the Pass-Through Facility, you can perform the following tasks:

- ☐ establish a connection with the DBMS using a CONNECT statement and terminate the connection with the DISCONNECT statement.
- ☐ send nonquery DBMS-specific SQL statements to the DBMS using the EXECUTE statement.
- ☐ retrieve data from the DBMS to be used in a PROC SQL query with the CONNECTION TO component in a SELECT statement's FROM clause.

You can use the Pass-Through Facility statements in a query, or you can store them in a PROC SQL view. When a view is stored, any options that are specified in the corresponding CONNECT statement are also stored. Thus, when the PROC SQL view is used in a SAS program, SAS can automatically establish the appropriate connection to the DBMS.

See "CONNECT Statement" on page 1128, "DISCONNECT Statement" on page 1137, "EXECUTE Statement" on page 1139, "CONNECTION TO" on page 1162, and "The Pass-Through Facility for Relational Databases" in *SAS/ACCESS for Relational Databases: Reference*.

Return Codes

As you use PROC SQL statements that are available in the Pass-Through Facility, any errors are written to the SAS log. The return codes and messages that are generated by the Pass-Through Facility are available to you through the SQLXRC and SQLXMSG macro variables. Both macro variables are described in "Using Macro Variables Set by PROC SQL" on page 1202.

Connecting to a DBMS Using the LIBNAME Statement

For many DBMSs, you can directly access DBMS data by assigning a libref to the DBMS using the SAS/ACCESS LIBNAME statement. Once you have associated a libref with the DBMS, you can specify a DBMS table in a two-level SAS name and work with the table like any SAS data set. You can also embed the LIBNAME statement in a PROC SQL view (see "CREATE VIEW Statement" on page 1133).

PROC SQL will take advantage of the capabilities of a DBMS by passing it certain operations whenever possible. For example, before implementing a join, PROC SQL checks to see if the DBMS can do the join. If it can, then PROC SQL passes the join to the DBMS. This enhances performance by reducing data movement and translation. If the DBMS cannot do the join, then PROC SQL processes the join. Using the SAS/ACCESS LIBNAME statement can often provide you with the performance benefits of the SQL Procedure Pass-Through Facility without having to write DBMS-specific code.

To use the SAS/ACCESS LIBNAME statement, you must have SAS/ACCESS software installed for your DBMS. For more information about the SAS/ACCESS LIBNAME statement, refer to the SAS/ACCESS documentation for your DBMS.

Using the DICTIONARY Tables

What Are DICTIONARY Tables?

DICTIONARY tables are special, read-only SAS data views that contain information about your SAS session. For example, the DICTIONARY.COLUMNS table contains information, such as name, type, length, and format, about all columns in all tables that are known to the current SAS session. DICTIONARY tables are accessed by using the libref DICTIONARY in the FROM clause in a SELECT statement in PROC SQL. Additionally, there are PROC SQL views, stored in the SASHELP library and known as *SASHELP views*, that reference the DICTIONARY tables and that can be used in other SAS procedures and in the DATA step.

Note: You cannot use data set options with DICTIONARY tables. Δ

For an example that demonstrates the use of a DICTIONARY table, see Example 6 on page 1218.

The following table describes the DICTIONARY tables that are available and shows the associated SASHELP view(s) for each table.

Table 40.2 DICTIONARY Tables and Associated SASHELP Views

DICTIONARY table	SASHELP view	Description
CATALOGS	VCATALG	Contains information about known SAS catalogs.
CHECK_CONSTRAINTS	VCHKCON	Contains information about known check constraints.
COLUMNS	VCOLUMN	Contains information about columns in all known tables.
CONSTRAINT_COLUMN_USAGE	VCNCOLU	Contains information about columns that are referred to by integrity constraints.
CONSTRAINT_TABLE_USAGE	VCNTABU	Contains information about tables that have integrity constraints defined on them.
DICTIONARIES	VDCTNRY	Contains information about all DICTIONARY tables.
EXTFILES	VEXTFL	Contains information about known external files.
FORMATS	VFORMAT	Contains information about currently accessible formats and informats.
GOPTIONS	VGOPT VALLOPT	Contains information about currently defined graphics options (SAS/GRAPH software). SASHELP.VALLOPT includes SAS system options as well as graphics options.
INDEXES	VINDEX	Contains information about known indexes.
LIBNAMES	VLIBNAM	Contains information about currently defined SAS data libraries.

DICTIONARY table	SASHELP view	Description
MACROS	VMACRO	Contains information about currently defined macros.
MEMBERS	VMEMBER VSACCES VSCATLG VSLIB VSTABLE VSTABVW VSVIEW	Contains information about all objects that are in currently defined SAS data libraries. SASHELP.VMEMBER contains information for all member types; the other SASHELP views are specific to particular member types (such as tables or views).
OPTIONS	VOPTION VALLOPT	Contains information on SAS system options. SASHELP.VALLOPT includes graphics options as well as SAS system options.
REFERENTIAL_CONSTRAINTS	VREFCON	Contains information about referential constraints.
STYLES		Contains information about known ODS styles.
TABLE_CONSTRAINTS	VTABCON	Contains information about integrity constraints in all known tables.
TABLES	VTABLE	Contains information about known tables.
TITLES	VTITLE	Contains information about currently defined titles and footnotes.
VIEWS	VVIEW	Contains information about known data views.

Retrieving Information about **DICTIONARY** Tables and **SASHELP** Views

To see how each **DICTIONARY** table is defined, submit a **DESCRIBE TABLE** statement. After you know how a table is defined, you can use its column names in a subsetting **WHERE** clause in order to retrieve more specific information. For example:

```
proc sql;
    describe table dictionary.indexes;
```

The results are written to the SAS log:

```
6    proc sql;
7        describe table dictionary.indexes;
NOTE: SQL table DICTIONARY.INDEXES was created like:

create table DICTIONARY.INDEXES
(
    libname char(8) label='Library Name',
    memname char(32) label='Member Name',
    memtype char(8) label='Member Type',
    name char(32) label='Column Name',
    idxusage char(9) label='Column Index Type',
    indxname char(32) label='Index Name',
    indxpos num label='Position of Column in Concatenated Key',
    nomiss char(3) label='Nomiss Option',
    unique char(3) label='Unique Option'
);
```

Use the DESCRIBE VIEW statement in PROC SQL to find out how a SASHELP view is defined. Here's an example:

```
proc sql;
    describe view sashelp.vstabvw;
```

The results are written to the SAS log:

```
6    proc sql;
7    describe view sashelp.vstabvw;
NOTE: SQL view SASHELP.VSTABVW is defined as:

      select libname, memname, memtype
      from DICTIONARY.MEMBERS
      where (memtype='VIEW') or (memtype='DATA')
      order by libname asc, memname asc;
```

Using DICTIONARY Tables

DICTIONARY tables are commonly used to monitor and manage SAS sessions because the data is more easily manipulated than the output from, for example, PROC DATASETS. You can query DICTIONARY tables the same way that you query any other table, including subsetting with a WHERE clause, ordering the results, and creating PROC SQL views. Because DICTIONARY tables are read-only objects, you cannot insert rows or columns, alter column attributes, or add integrity constraints to them.

DICTIONARY Tables and Performance

When querying a DICTIONARY table, SAS launches a discovery process that gathers information that is pertinent to that table. Depending on the DICTIONARY table that is being queried, this discovery process can search libraries, open tables, and execute views. Unlike other SAS procedures and the DATA step, PROC SQL can mitigate this process by optimizing the query before the discovery process is launched. Therefore, although it is possible to access DICTIONARY table information with SAS procedures or the DATA step by using the SASHELP views, it is often more efficient to use PROC SQL instead.

For example, the following programs both produce the same result, but the PROC SQL step runs much faster because the WHERE clause is processed prior to opening the tables that are referenced by the SASHELP.VCOLUMN view:

```
data mytable;
    set sashelp.vcolumn;
    where libname='WORK' and memname='SALES';
run;

proc sql;
    create table mytable as
        select * from sashelp.vcolumn
        where libname='WORK' and memname='SALES';
quit;
```

Note: SAS does not maintain DICTIONARY table information between queries. Each query of a DICTIONARY table launches a new discovery process. Δ

If you are querying the same DICTIONARY table several times in a row, you can get even faster performance by creating a temporary SAS data set (with the DATA step SET statement or PROC SQL CREATE TABLE AS statement) with the information that you want and running your query against that data set.

Using Macro Variables Set by PROC SQL

PROC SQL sets up macro variables with certain values after it executes each statement. These macro variables can be tested inside a macro to determine whether to continue executing the PROC SQL step. SAS/AF software users can also test them in a program after an SQL SUBMIT block of code, using the SYMGET function.

After each PROC SQL statement has executed, the following macro variables are updated with these values:

SQLOBS

contains the number of rows executed by an SQL procedure statement. For example, it contains the number of rows formatted and displayed in SAS output by a SELECT statement or the number of rows deleted by a DELETE statement.

SQLRC

contains the following status values that indicate the success of the SQL procedure statement:

0

PROC SQL statement completed successfully with no errors.

4

PROC SQL statement encountered a situation for which it issued a warning. The statement continued to execute.

8

PROC SQL statement encountered an error. The statement stopped execution at this point.

12

PROC SQL statement encountered an internal error, indicating a bug in PROC SQL that should be reported to SAS Technical Support. These errors can occur only during compile time.

16

PROC SQL statement encountered a user error. This error code is used, for example, when a subquery (that can only return a single value) evaluates to more than one row. These errors can only be detected during run time.

24

PROC SQL statement encountered a system error. This error is used, for example, if the system cannot write to a PROC SQL table because the disk is full. These errors can occur only during run time.

28

PROC SQL statement encountered an internal error, indicating a bug in PROC SQL that should be reported to SAS Technical Support. These errors can occur only during run time.

SQLOOPS

contains the number of iterations that the inner loop of PROC SQL executes. The number of iterations increases proportionally with the complexity of the query. See also the description of LOOPS= on page 1122.

SQLXRC

contains the DBMS-specific return code that is returned by the Pass-Through Facility.

SQLXMSG

contains descriptive information and the DBMS-specific return code for the error that is returned by the Pass-Through Facility.

Note: Because the value of the SQLXMSG macro variable can contain special characters (such as &, %, /, *, and ;), use the %SUPERQ macro function when printing the value:

```
%put %superq(sqlxmsg);
```

See *SAS Macro Language: Reference* for information about the %SUPERQ function. △

This example retrieves the data but does not display them in SAS output because of the NOPRINT option in the PROC SQL statement. The %PUT macro statement displays the macro variables values.

```
proc sql noprint;
  select *
    from proclib.payroll;

%put sqlobs=**&sqlobs**
      sqloops=**&sqloops**
      sqlrc=**&sqlrc**;
```

The message in Output 40.3 on page 1203 appears in the SAS log and gives you the macros' values.

Output 40.3 PROC SQL Macro Variable Values

```
40  options ls=80;
41
42  proc sql noprint;
43    select *
44      from proclib.payroll;
45
46  %put sqlobs=**&sqlobs**
47        sqloops=**&sqloops**
48        sqlrc=**&sqlrc**;
```

sqlobs=**1** sqloops=**11** sqlrc=**0**

Updating PROC SQL and SAS/ACCESS Views

You can update PROC SQL and SAS/ACCESS views using the INSERT, DELETE, and UPDATE statements, under the following conditions.

- If the view accesses a DBMS table, then you must have been granted the appropriate authorization by the external database management system (for example, DB2). You must have installed the SAS/ACCESS software for your DBMS. See the SAS/ACCESS interface guide for your DBMS for more information on SAS/ACCESS views.
- You can update only a single table through a view. The table cannot be joined to another table or linked to another table with a set-operator. The view cannot contain a subquery.
- You can update a column in a view using the column's alias, but you cannot update a derived column, that is, a column produced by an expression. In the following example, you can update the column SS, but not WeeklySalary.

```
create view EmployeeSalaries as
  select Employee, SSNumber as SS,
         Salary/52 as WeeklySalary
```

from employees;

- You cannot update a view containing an ORDER BY.

Note: Starting in Version 9, PROC SQL views, the Pass-Through Facility, and the SAS/ACCESS LIBNAME statement are the preferred ways to access relational DBMS data; SAS/ACCESS views are no longer recommended. You can convert existing SAS/ACCESS views to PROC SQL views by using the CV2VIEW procedure. See The CV2VIEW Procedure in *SAS/ACCESS for Relational Databases: Reference* for more information. \triangle

PROC SQL and the ANSI Standard

PROC SQL follows most of the guidelines set by the American National Standards Institute (ANSI) in its implementation of SQL. However, it is not fully compliant with the current ANSI Standard for SQL.*

The SQL research project at SAS has focused primarily on the expressive power of SQL as a query language. Consequently, some of the database features of SQL have not yet been implemented in PROC SQL.

This section describes

- enhancements to SQL that SAS has made through PROC SQL
- the ways in which PROC SQL differs from the current ANSI Standard for SQL.

SQL Procedure Enhancements

Most of the enhancements described here are required by the current ANSI Standard.

Reserved Words

PROC SQL reserves very few keywords and then only in certain contexts. The ANSI Standard reserves all SQL keywords in all contexts. For example, according to the Standard you cannot name a column GROUP because of the keywords GROUP BY.

The following words are reserved in PROC SQL:

- The keyword CASE is always reserved; its use in the CASE expression (an SQL2 feature) precludes its use as a column name.

If you have a column named CASE in a table and you want to specify it in a PROC SQL step, then you can use the SAS data set option RENAME= to rename that column for the duration of the query. You can also surround CASE in double quotation marks ("CASE") and set the PROC SQL option DQUOTE=ANSI.

- The keywords AS, ON, FULL, JOIN, LEFT, FROM, WHEN, WHERE, ORDER, GROUP, RIGHT, INNER, OUTER, UNION, EXCEPT, HAVING, and INTERSECT cannot normally be used for table aliases. These keywords all introduce clauses that appear after a table name. Since the alias is optional, PROC SQL deals with this ambiguity by assuming that any one of these words introduces the corresponding clause and is not the alias. If you want to use one of these keywords as an alias, then use the PROC SQL option DQUOTE=ANSI.
- The keyword USER is reserved for the current userid. If you specify USER on a SELECT statement in conjunction with a CREATE TABLE statement, then the

* International Organization for Standardization (ISO): *Database SQL*. Document ISO/IEC 9075:1992. Also available as American National Standards Institute (ANSI) Document ANSI X3.135-1992.

column is created in the table with a temporary column name that is similar to _TEMA001. If you specify USER in a SELECT statement without using the CREATE TABLE statement, then the column is written to the output without a column heading. In either case, the value for the column varies by operating environment, but is typically the userid of the user who is submitting the program or the value of the &SYSJOBID automatic macro variable.

If you have a column named USER in a table and you want to specify it in a PROC SQL step, then you can use the SAS data set option RENAME= to rename that column for the duration of the query. You can also enclose USER with double quotation marks ("USER") and set the PROC SQL option DQUOTE=ANSI.

Column Modifiers

PROC SQL supports the SAS INFORMAT=, FORMAT=, and LABEL= modifiers for expressions within the SELECT clause. These modifiers control the format in which output data are displayed and labeled.

Alternate Collating Sequences

PROC SQL allows you to specify an alternate collating (sorting) sequence to be used when you specify the ORDER BY clause. See the description of the SORTSEQ= option in "PROC SQL Statement" on page 1119 for more information.

ORDER BY Clause in a View Definition

PROC SQL permits you to specify an ORDER BY clause in a CREATE VIEW statement. When the view is queried, its data are always sorted according to the specified order unless a query against that view includes a different ORDER BY clause. See "CREATE VIEW Statement" on page 1133 for more information.

In-Line Views

The ability to code nested query-expressions in the FROM clause is a requirement of the ANSI Standard. PROC SQL supports such nested coding.

Outer Joins

The ability to include columns that both match and do not match in a join-expression is a requirement of the ANSI Standard. PROC SQL supports this ability.

Arithmetic Operators

PROC SQL supports the SAS exponentiation (**) operator. PROC SQL uses the notation <> to mean not equal.

Orthogonal Expressions

PROC SQL permits the combination of comparison, Boolean, and algebraic expressions. For example, (X=3)*7 yields a value of 7 if X=3 is true because true is defined to be 1. If X=3 is false, then it resolves to 0 and the entire expression yields a value of 0.

PROC SQL permits a subquery in any expression. This feature is required by the ANSI Standard. Therefore, you can have a subquery on the left side of a comparison operator in the WHERE expression.

PROC SQL permits you to order and group data by any kind of mathematical expression (except those including summary functions) using ORDER BY and GROUP BY clauses. You can also group by an expression that appears on the SELECT clause by using the integer that represents the expression's ordinal position in the SELECT clause. You are not required to select the expression by which you are grouping or ordering. See ORDER BY Clause on page 1151 and GROUP BY Clause on page 1149 for more information.

Set Operators

The set operators UNION, INTERSECT, and EXCEPT are required by the ANSI Standard. PROC SQL provides these operators plus the OUTER UNION operator.

The ANSI Standard also requires that the tables being operated upon all have the same number of columns with matching data types. The SQL procedure works on tables that have the same number of columns, as well as on those that do not, by creating virtual columns so that a query can evaluate correctly. See “query-expression” on page 1176 for more information.

Statistical Functions

PROC SQL supports many more summary functions than required by the ANSI Standard for SQL.

PROC SQL supports the remerging of summary function results into the table's original data. For example, computing the percentage of total is achieved with $100 * x / \text{SUM}(x)$ in PROC SQL. See “summary-function” on page 1190 for more information on the available summary functions and remerging data.

SAS DATA Step Functions

PROC SQL supports all the functions available to the SAS DATA step, except for LAG, DIF, and SOUND. Other SQL databases support their own set of functions.

SQL Procedure Omissions

PROC SQL differs from the ANSI Standard for SQL in the following ways.

COMMIT Statement

The COMMIT statement is not supported.

ROLLBACK Statement

The ROLLBACK statement is not supported. The UNDO_POLICY= option in the PROC SQL statement addresses rollback. See the description of the UNDO_POLICY= option in “PROC SQL Statement” on page 1119 for more information.

Identifiers and Naming Conventions

In SAS, table names, column names, and aliases are limited to 32 characters and can contain mixed case. For more information on SAS naming conventions, see *SAS Language Reference: Dictionary*. The ANSI Standard for SQL allows longer names.

Granting User Privileges

The GRANT statement, PRIVILEGES keyword, and authorization-identifier features of SQL are not supported. You might want to use operating environment-specific means of security instead.

Three-Valued Logic

ANSI-compatible SQL has three-valued logic, that is, special cases for handling comparisons involving NULL values. Any value compared with a NULL value evaluates to NULL.

PROC SQL follows the SAS convention for handling missing values: when numeric NULL values are compared to non-NULL numbers, the NULL values are less than or smaller than all the non-NULL values; when character NULL values are compared to non-NULL characters, the character NULL values are treated as a string of blanks.

Embedded SQL

Currently there is no provision for embedding PROC SQL statements in other SAS programming environments, such as the DATA step or SAS/IML software.

Examples: SQL Procedure

Example 1: Creating a Table and Inserting Data into It

Procedure features:

- CREATE TABLE statement
 - column-modifier
- INSERT statement
 - VALUES clause
- SELECT clause
- FROM clause

Table: PROCLIB.PAYLIST

This example creates the table PROCLIB.PAYLIST and inserts data into it.

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Create the PROCLIB.PAYLIST table. The CREATE TABLE statement creates PROCLIB.PAYLIST with six empty columns. Each column definition indicates whether the column is character or numeric. The number in parentheses specifies the width of the column. INFORMAT= and FORMAT= assign date informats and formats to the Birth and Hired columns.

```
proc sql;
  create table proclib.paylist
    (IdNum char(4),
     Gender char(1),
     Jobcode char(3),
     Salary num,
     Birth num informat=date7.
           format=date7.,
     Hired num informat=date7.
           format=date7.);
```

Insert values into the PROCLIB.PAYLIST table. The INSERT statement inserts data values into PROCLIB.PAYLIST according to the position in the VALUES clause. Therefore, in the first VALUES clause, 1639 is inserted into the first column, F into the second column, and so forth. Dates in SAS are stored as integers with 0 equal to January 1, 1960. Suffixing the date with a d is one way to use the internal value for dates.

```
insert into proclib.paylist
  values('1639','F','TA1',42260,'26JUN70'd,'28JAN91'd)
  values('1065','M','ME3',38090,'26JAN54'd,'07JAN92'd)
  values('1400','M','ME1',29769,'05NOV67'd,'16OCT90'd)
```

Include missing values in the data. The value null represents a missing value for the character column Jobcode. The period represents a missing value for the numeric column Salary.

```
values('1561','M',null,36514,'30NOV63'd,'07OCT87'd)
values('1221','F','FA3',., '22SEP63'd,'04OCT94'd);
```

Specify the title.

```
title 'PROCLIB.PAYLIST Table';
```

Display the entire PROCLIB.PAYLIST table. The SELECT clause selects columns from PROCLIB.PAYLIST. The asterisk (*) selects all columns. The FROM clause specifies PROCLIB.PAYLIST as the table to select from.

```
select *
  from proclib.paylist;
```

Output Table

PROCLIB.PAYLIST

PROCLIB.PAYLIST Table						
Id Num	Gender	Jobcode	Salary	Birth	Hired	
1639	F	TA1	42260	26JUN70	28JAN91	
1065	M	ME3	38090	26JAN54	07JAN92	
1400	M	ME1	29769	05NOV67	16OCT90	
1561	M		36514	30NOV63	07OCT87	
1221	F	FA3	.	22SEP63	04OCT94	

Example 2: Creating a Table from a Query's Result

Procedure features:

CREATE TABLE statement
 AS query-expression
 SELECT clause
 column alias
 FORMAT= column-modifier
object-item

Other features:

data set option
 OBS=

Tables:

PROCLIB.PAYROLL, PROCLIB.BONUS

This example builds a column with an arithmetic expression and creates the PROCLIB.BONUS table from the query's result.

Input Table

PROCLIB.PAYROLL (Partial Listing)

PROCLIB.PAYROLL First 10 Rows Only						
Id Number	Gender	Jobcode	Salary	Birth	Hired	
1919	M	TA2	34376	12SEP60	04JUN87	
1653	F	ME2	35108	15OCT64	09AUG90	
1400	M	ME1	29769	05NOV67	16OCT90	
1350	F	FA3	32886	31AUG65	29JUL90	
1401	M	TA3	38822	13DEC50	17NOV85	
1499	M	ME3	43025	26APR54	07JUN80	
1101	M	SCP	18723	06JUN62	01OCT90	
1333	M	PT2	88606	30MAR61	10FEB81	
1402	M	TA2	32615	17JAN63	02DEC90	
1479	F	TA3	38785	22DEC68	05OCT89	

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Create the PROCLIB.BONUS table. The CREATE TABLE statement creates the table PROCLIB.BONUS from the result of the subsequent query.

```
proc sql;
  create table proclib.bonus as
```

Select the columns to include. The SELECT clause specifies that three columns will be in the new table: IdNumber, Salary, and Bonus. FORMAT= assigns the DOLLAR8. format to Salary. The Bonus column is built with the SQL expression **salary*.025**.

```
  select IdNumber, Salary format=dollar8.,
         salary*.025 as Bonus format=dollar8.
  from proclib.payroll;
```

Specify the title.

```
  title 'BONUS Information';
```

Display the first 10 rows of the PROCLIB.BONUS table. The SELECT clause selects columns from PROCLIB.BONUS. The asterisk (*) selects all columns. The FROM clause specifies PROCLIB.BONUS as the table to select from. The OBS= data set option limits the printing of the output to 10 rows.

```
select *
  from proclib.bonus(obs=10);
```

Output

PROCLIB.BONUS

BONUS Information		
Id		
Number	Salary	Bonus

1919	\$34,376	\$859
1653	\$35,108	\$878
1400	\$29,769	\$744
1350	\$32,886	\$822
1401	\$38,822	\$971
1499	\$43,025	\$1,076
1101	\$18,723	\$468
1333	\$88,606	\$2,215
1402	\$32,615	\$815
1479	\$38,785	\$970

Example 3: Updating Data in a PROC SQL Table

Procedure features:

ALTER TABLE statement

DROP clause

MODIFY clause

UPDATE statement

SET clause

CASE expression

Table: EMPLOYEES

This example updates data values in the EMPLOYEES table and drops a column.

Input

```
data Employees;
  input IdNum $4. +2 LName $11. FName $11. JobCode $3.
```

```

          +1 Salary 5. +1 Phone $12.;
    datalines;
1876  CHIN          JACK          TA1 42400 212/588-5634
1114  GREENWALD    JANICE        ME3 38000 212/588-1092
1556  PENNINGTON   MICHAEL       ME1 29860 718/383-5681
1354  PARKER       MARY          FA3 65800 914/455-2337
1130  WOOD         DEBORAH       PT2 36514 212/587-0013
;

```

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=40;
```

Display the entire EMPLOYEES table. The SELECT clause displays the table before the updates. The asterisk (*) selects all columns for display. The FROM clause specifies EMPLOYEES as the table to select from.

```
proc sql;
    title 'Employees Table';
    select * from Employees;
```

Update the values in the Salary column. The UPDATE statement updates the values in EMPLOYEES. The SET clause specifies that the data in the Salary column be multiplied by 1.04 when the job code ends with a 1 and 1.025 for all other job codes. (The two underscores represent any character.) The CASE expression returns a value for each row that completes the SET clause.

```
update employees
    set salary=salary*
    case when jobcode like '__1' then 1.04
         else 1.025
    end;
```

Modify the format of the Salary column and delete the Phone column. The ALTER TABLE statement specifies EMPLOYEES as the table to alter. The MODIFY clause permanently modifies the format of the Salary column. The DROP clause permanently drops the Phone column.

```
alter table employees
    modify salary num format=dollar8.
    drop phone;
```

Specify the title.

```
title 'Updated Employees Table';
```

Display the entire updated EMPLOYEES table. The SELECT clause displays the EMPLOYEES table after the updates. The asterisk (*) selects all columns.

```
select * from employees;
```

Output

Employees Table							1
Id Num	LName	FName	Job Code	Salary	Phone		
1876	CHIN	JACK	TA1	42400	212/588-5634		
1114	GREENWALD	JANICE	ME3	38000	212/588-1092		
1556	PENNINGTON	MICHAEL	ME1	29860	718/383-5681		
1354	PARKER	MARY	FA3	65800	914/455-2337		
1130	WOOD	DEBORAH	PT2	36514	212/587-0013		

Updated Employees Table							2
Id Num	LName	FName	Job Code	Salary			
1876	CHIN	JACK	TA1	\$44,096			
1114	GREENWALD	JANICE	ME3	\$38,950			
1556	PENNINGTON	MICHAEL	ME1	\$31,054			
1354	PARKER	MARY	FA3	\$67,445			
1130	WOOD	DEBORAH	PT2	\$37,427			

Example 4: Joining Two Tables

Procedure features:

FROM clause

table alias

inner join

joined-table component

PROC SQL statement option

NUMBER

WHERE clause

IN condition

Tables: PROCLIB.STAFF, PROCLIB.PAYROLL

This example joins two tables in order to get more information about data that are common to both tables.

Input Tables

PROCLIB.STAFF (Partial Listing)

PROCLIB.STAFF First 10 Rows Only					
Id Num	Lname	Fname	City	State	Hphone
1919	ADAMS	GERALD	STAMFORD	CT	203/781-1255
1653	ALIBRANDI	MARIA	BRIDGEPORT	CT	203/675-7715
1400	ALHERTANI	ABDULLAH	NEW YORK	NY	212/586-0808
1350	ALVAREZ	MERCEDES	NEW YORK	NY	718/383-1549
1401	ALVAREZ	CARLOS	PATERSON	NJ	201/732-8787
1499	BAREFOOT	JOSEPH	PRINCETON	NJ	201/812-5665
1101	BAUCOM	WALTER	NEW YORK	NY	212/586-8060
1333	BANADYGA	JUSTIN	STAMFORD	CT	203/781-1777
1402	BLALOCK	RALPH	NEW YORK	NY	718/384-2849
1479	BALLETTI	MARIE	NEW YORK	NY	718/384-8816

PROCLIB.PAYROLL (Partial Listing)

PROCLIB.PAYROLL First 10 Rows Only						
Id Number	Gender	Jobcode	Salary	Birth	Hired	
1919	M	TA2	34376	12SEP60	04JUN87	
1653	F	ME2	35108	15OCT64	09AUG90	
1400	M	ME1	29769	05NOV67	16OCT90	
1350	F	FA3	32886	31AUG65	29JUL90	
1401	M	TA3	38822	13DEC50	17NOV85	
1499	M	ME3	43025	26APR54	07JUN80	
1101	M	SCP	18723	06JUN62	01OCT90	
1333	M	PT2	88606	30MAR61	10FEB81	
1402	M	TA2	32615	17JAN63	02DEC90	
1479	F	TA3	38785	22DEC68	05OCT89	

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.


```
options nodate pageno=1 linesize=120 pagesize=40;
```

Add row numbers to PROC SQL output. NUMBER adds a column that contains the row number.

```
proc sql number;
```

Specify the title.

```
title 'Information for Certain Employees Only';
```

Select the columns to display. The SELECT clause selects the columns to show in the output.

```
select Lname, Fname, City, State,
       IdNumber, Salary, Jobcode
```

Specify the tables from which to obtain the data. The FROM clause lists the tables to select from.

```
from proclib.staff, proclib.payroll
```

Specify the join criterion and subset the query. The WHERE clause specifies that the tables are joined on the ID number from each table. WHERE also further subsets the query with the IN condition, which returns rows for only four employees.

```
where idnumber=idnum and idnum in
      ('1919', '1400', '1350', '1333');
```

Output

Information for Certain Employees Only						
Row	Lname		Fname	City	State	Id Number
	Salary	Jobcode				
1	ADAMS		GERALD	STAMFORD	CT	1919
	34376	TA2				
2	ALHERTANI		ABDULLAH	NEW YORK	NY	1400
	29769	ME1				
3	ALVAREZ		MERCEDES	NEW YORK	NY	1350
	32886	FA3				
4	BANADYGA		JUSTIN	STAMFORD	CT	1333
	88606	PT2				

Example 5: Combining Two Tables

Procedure features:

DELETE statement

IS condition

RESET statement option

DOUBLE

UNION set operator

Tables: PROCLIB.NEWPAY, PROCLIB.PAYLIST, PROCLIB.PAYLIST2

This example creates a new table, PROCLIB.NEWPAY, by concatenating two other tables: PROCLIB.PAYLIST and PROCLIB.PAYLIST2.

Input Tables

PROCLIB.PAYLIST

Information for Certain Employees Only						
Id						
Num	Gender	Jobcode	Salary	Birth	Hired	
1639	F	TA1	42260	26JUN70	28JAN91	
1065	M	ME3	38090	26JAN54	07JAN92	
1400	M	ME1	29769	05NOV67	16OCT90	
1561	M		36514	30NOV63	07OCT87	
1221	F	FA3	.	22SEP63	04OCT94	

PROCLIB.PAYLIST2

PROCLIB.PAYLIST2 Table						
Id						
Num	Gender	Jobcode	Salary	Birth	Hired	
1919	M	TA2	34376	12SEP66	04JUN87	
1653	F	ME2	31896	15OCT64	09AUG92	
1350	F	FA3	36886	31AUG55	29JUL91	
1401	M	TA3	38822	13DEC55	17NOV93	
1499	M	ME1	23025	26APR74	07JUN92	

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the PROCLIB.NEWPAY table. The SELECT clauses select all the columns from the tables that are listed in the FROM clauses. The UNION set operator concatenates the query results that are produced by the two SELECT clauses.

```
proc sql;
  create table proclib.newpay as
    select * from proclib.paylist
  union
    select * from proclib.paylist2;
```

Delete rows with missing Jobcode or Salary values. The DELETE statement deletes rows from PROCLIB.NEWPAY that satisfy the WHERE expression. The IS condition specifies rows that contain missing values in the Jobcode or Salary column.

```
delete
  from proclib.newpay
  where jobcode is missing or salary is missing;
```

Reset the PROC SQL environment and double-space the output. RESET changes the procedure environment without stopping and restarting PROC SQL. The DOUBLE option double-spaces the output. (The DOUBLE option has no effect on ODS output.)

```
reset double;
```

Specify the title.

```
title 'Personnel Data';
```

Display the entire PROCLIB.NEWPAY table. The SELECT clause selects all columns from the newly created table, PROCLIB.NEWPAY.

```
select *
  from proclib.newpay;
```

Output

Personnel Data						
Id						
Num	Gender	Jobcode	Salary	Birth	Hired	
1065	M	ME3	38090	26JAN54	07JAN92	
1350	F	FA3	36886	31AUG55	29JUL91	
1400	M	ME1	29769	05NOV67	16OCT90	
1401	M	TA3	38822	13DEC55	17NOV93	
1499	M	ME1	23025	26APR74	07JUN92	
1639	F	TA1	42260	26JUN70	28JAN91	
1653	F	ME2	31896	15OCT64	09AUG92	
1919	M	TA2	34376	12SEP66	04JUN87	

Example 6: Reporting from DICTIONARY Tables

Procedure features:

DESCRIBE TABLE statement

DICTIONARY.*table-name* component

Table: DICTIONARY.MEMBERS

This example uses DICTIONARY tables to show a list of the SAS files in a SAS data library. If you do not know the names of the columns in the DICTIONARY table that you are querying, then use a DESCRIBE TABLE statement with the table.

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. SOURCE writes the programming statements to the SAS log.

```
options nodate pageno=1 source linesize=80 pagesize=60;
```

List the column names from the DICTONARY.MEMBERS table. DESCRIBE TABLE writes the column names from DICTONARY.MEMBERS to the SAS log.

```
proc sql;
    describe table dictionary.members;
```

Specify the title.

```
title 'SAS Files in the PROCLIB Library';
```

Display a list of files in the PROCLIB library. The SELECT clause selects the MEMNAME and MEMTYPE columns. The FROM clause specifies DICTONARY.MEMBERS as the table to select from. The WHERE clause subsets the output to include only those rows that have a libref of *PROCLIB* in the LIBNAME column.

```
select memname, memtype
    from dictionary.members
    where libname='PROCLIB';
```

Log

```
277 options nodate pageno=1 source linesize=80 pagesize=60;
278
279 proc sql;
280     describe table dictionary.members;
NOTE: SQL table DICTONARY.MEMBERS was created like:

create table DICTONARY.MEMBERS
(
    libname char(8) label='Library Name',
    memname char(32) label='Member Name',
    memtype char(8) label='Member Type',
    engine char(8) label='Engine Name',
    index char(32) label='Indexes',
    path char(1024) label='Path Name'
);

281     title 'SAS Files in the PROCLIB Library';
282
283     select memname, memtype
284         from dictionary.members
285         where libname='PROCLIB';
```

Output

SAS Files in the PROCLIB Library	
Member Name	Member Type

ALL	DATA
BONUS	DATA
BONUS95	DATA
DELAY	DATA
HOUSES	DATA
INTERNAT	DATA
MARCH	DATA
NEWPAY	DATA
PAYLIST	DATA
PAYLIST2	DATA
PAYROLL	DATA
PAYROLL2	DATA
SCHEDULE	DATA
SCHEDULE2	DATA
STAFF	DATA
STAFF2	DATA
SUPERV	DATA
SUPERV2	DATA

Example 7: Performing an Outer Join

Procedure features:

- joined-table component
- left outer join
- SELECT clause
 - COALESCE function
- WHERE clause
 - CONTAINS condition

Tables: PROCLIB.PAYROLL, PROCLIB.PAYROLL2

This example illustrates a left outer join of the PROCLIB.PAYROLL and PROCLIB.PAYROLL2 tables.

Input Tables

PROCLIB.PAYROLL (Partial Listing)

PROCLIB.PAYROLL					
First 10 Rows Only					
Id					
Number	Gender	Jobcode	Salary	Birth	Hired

1009	M	TA1	28880	02MAR59	26MAR92
1017	M	TA3	40858	28DEC57	16OCT81
1036	F	TA3	39392	19MAY65	23OCT84
1037	F	TA1	28558	10APR64	13SEP92
1038	F	TA1	26533	09NOV69	23NOV91
1050	M	ME2	35167	14JUL63	24AUG86
1065	M	ME2	35090	26JAN44	07JAN87
1076	M	PT1	66558	14OCT55	03OCT91
1094	M	FA1	22268	02APR70	17APR91
1100	M	BCK	25004	01DEC60	07MAY88

PROCLIB.PAYROLL2

PROCLIB.PAYROLL2						
Id						
Num	Sex	Jobcode	Salary	Birth	Hired	

1036	F	TA3	42465	19MAY65	23OCT84	
1065	M	ME3	38090	26JAN44	07JAN87	
1076	M	PT1	69742	14OCT55	03OCT91	
1106	M	PT3	94039	06NOV57	16AUG84	
1129	F	ME3	36758	08DEC61	17AUG91	
1221	F	FA3	29896	22SEP67	04OCT91	
1350	F	FA3	36098	31AUG65	29JUL90	
1369	M	TA3	36598	28DEC61	13MAR87	
1447	F	FA1	22123	07AUG72	29OCT92	
1561	M	TA3	36514	30NOV63	07OCT87	
1639	F	TA3	42260	26JUN57	28JAN84	
1998	M	SCP	23100	10SEP70	02NOV92	

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Limit the number of output rows. OUTOBS= limits the output to 10 rows.

```
proc sql outobs=10;
```

Specify the title for the first query.

```
title 'Most Current Jobcode and Salary Information';
```

Select the columns. The SELECT clause lists the columns to select. Some column names are prefixed with a table alias because they are in both tables. LABEL= and FORMAT= are column modifiers.

```
select p.IdNumber, p.Jobcode, p.Salary,
       p2.jobcode label='New Jobcode',
       p2.salary label='New Salary' format=dollar8.
```

Specify the type of join. The FROM clause lists the tables to join and assigns table aliases. The keywords LEFT JOIN specify the type of join. The order of the tables in the FROM clause is important. PROCLIB.PAYROLL is listed first and is considered the “left” table. PROCLIB.PAYROLL2 is the “right” table.

```
from proclib.payroll as p left join proclib.payroll2 as p2
```

Specify the join criterion. The ON clause specifies that the join be performed based on the values of the ID numbers from each table.

```
on p.IdNumber=p2.idnum;
```

Output

As the output shows, all rows from the left table, PROCLIB.PAYROLL, are returned. PROC SQL assigns missing values for rows in the left table, PAYROLL, that have no matching values for IdNum in PAYROLL2.

Most Current Jobcode and Salary Information				
Id Number	Jobcode	Salary	New Jobcode	New Salary
1009	TA1	28880		.
1017	TA3	40858		.
1036	TA3	39392	TA3	\$42,465
1037	TA1	28558		.
1038	TA1	26533		.
1050	ME2	35167		.
1065	ME2	35090	ME3	\$38,090
1076	PT1	66558	PT1	\$69,742
1094	FA1	22268		.
1100	BCK	25004		.

Specify the title for the second query.

```
title 'Most Current Jobcode and Salary Information';
```

Select the columns and coalesce the Jobcode columns. The SELECT clause lists the columns to select. COALESCE overlays the like-named columns. For each row, COALESCE returns the first nonmissing value of either P2.JOBCODE or P.JOBCODE. Because P2.JOBCODE is the first argument, if there is a nonmissing value for P2.JOBCODE, COALESCE returns that value. Thus, the output contains the most recent job code information for every employee. LABEL= assigns a column label.

```
select p.idnumber, coalesce(p2.jobcode,p.jobcode)
       label='Current Jobcode',
```

Coalesce the Salary columns. For each row, COALESCE returns the first nonmissing value of either P2.SALARY or P.SALARY. Because P2.SALARY is the first argument, if there is a nonmissing value for P2.SALARY, then COALESCE returns that value. Thus, the output contains the most recent salary information for every employee.

```
       coalesce(p2.salary,p.salary) label='Current Salary'
       format=dollar8.
```

Specify the type of join and the join criterion. The FROM clause lists the tables to join and assigns table aliases. The keywords LEFT JOIN specify the type of join. The ON clause specifies that the join is based on the ID numbers from each table.

```
from proclib.payroll1 p left join proclib.payroll2 p2
on p.IdNumber=p2.idnum;
```

Output

Most Current Jobcode and Salary Information

Id Number	Current Jobcode	Current Salary

1009	TA1	\$28,880
1017	TA3	\$40,858
1036	TA3	\$42,465
1037	TA1	\$28,558
1038	TA1	\$26,533
1050	ME2	\$35,167
1065	ME3	\$38,090
1076	PT1	\$69,742
1094	FA1	\$22,268
1100	BCK	\$25,004

Subset the query. The WHERE clause subsets the left join to include only those rows containing the value **TA**.

```

title 'Most Current Information for Ticket Agents';
select p.IdNumber,
       coalesce(p2.jobcode,p.jobcode) label='Current Jobcode',
       coalesce(p2.salary,p.salary) label='Current Salary'
  from proclib.payroll p left join proclib.payroll2 p2
    on p.IdNumber=p2.idnum
 where p2.jobcode contains 'TA';

```

Output

Most Current Information for Ticket Agents			
	Id	Current	Current
	Number	Jobcode	Salary

	1036	TA3	42465
	1369	TA3	36598
	1561	TA3	36514
	1639	TA3	42260

Example 8: Creating a View from a Query's Result

Procedure features:

CREATE VIEW statement

GROUP BY clause

SELECT clause

COUNT function

HAVING clause

Other features:

AVG summary function

data set option

PW=

Tables: PROCLIB.PAYROLL, PROCLIB.JOBS

This example creates the PROC SQL view PROCLIB.JOBS from the result of a query-expression.

Input Table

PROCLIB.PAYROLL (Partial Listing)

PROCLIB.PAYROLL						
First 10 Rows Only						
Id						
Number	Gender	Jobcode	Salary	Birth	Hired	
1009	M	TA1	28880	02MAR59	26MAR92	
1017	M	TA3	40858	28DEC57	16OCT81	
1036	F	TA3	39392	19MAY65	23OCT84	
1037	F	TA1	28558	10APR64	13SEP92	
1038	F	TA1	26533	09NOV69	23NOV91	
1050	M	ME2	35167	14JUL63	24AUG86	
1065	M	ME2	35090	26JAN44	07JAN87	
1076	M	PT1	66558	14OCT55	03OCT91	
1094	M	FA1	22268	02APR70	17APR91	
1100	M	BCK	25004	01DEC60	07MAY88	

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the PROCLIB.JOBS view. CREATE VIEW creates the PROC SQL view PROCLIB.JOBS. The PW= data set option assigns password protection to the data that is generated by this view.

```
proc sql;
  create view proclib.jobs(pw=red) as
```

Select the columns. The SELECT clause specifies four columns for the view: Jobcode and three columns, Number, AVGAGE, and AVGSAL, whose values are the products functions. COUNT returns the number of nonmissing values for each job code because the data is grouped by Jobcode. LABEL= assigns a label to the column.

```
  select Jobcode,
         count(jobcode) as number label='Number',
```

Calculate the Avgage and Avgsal columns. The AVG summary function calculates the average age and average salary for each job code.

```

avg(int((today()-birth)/365.25)) as avgage
    format=2. label='Average Age',
avg(salary) as avgsal
    format=dollar8. label='Average Salary'

```

Specify the table from which the data is obtained. The FROM clause specifies PAYROLL as the table to select from. PROC SQL assumes the libref of PAYROLL to be PROCLIB because PROCLIB is used in the CREATE VIEW statement.

```

from payroll

```

Organize the data into groups and specify the groups to include in the output. The GROUP BY clause groups the data by the values of Jobcode. Thus, any summary statistics are calculated for each grouping of rows by value of Jobcode. The HAVING clause subsets the grouped data and returns rows for job codes that contain an average age of greater than or equal to 30.

```

group by jobcode
having avgage ge 30;

```

Specify the titles.

```

title 'Current Summary Information for Each Job Category';
title2 'Average Age Greater Than or Equal to 30';

```

Display the entire PROCLIB.JOBS view. The SELECT statement selects all columns from PROCLIB.JOBS. PW=RED is necessary because the view is password protected.

```

select * from proclib.jobs(pw=red);

```

Output

Current Summary Information for Each Job Category
Average Age Greater Than Or Equal to 30

Jobcode	Number	Average Age	Average Salary
BCK	9	36	\$25,794
FA1	11	33	\$23,039
FA2	16	37	\$27,987
FA3	7	39	\$32,934
ME1	8	34	\$28,500
ME2	14	39	\$35,577
ME3	7	42	\$42,411
NA1	5	30	\$42,032
NA2	3	42	\$52,383
PT1	8	38	\$67,908
PT2	10	43	\$87,925
PT3	2	54	\$10,505
SCP	7	37	\$18,309
TA1	9	36	\$27,721
TA2	20	36	\$33,575
TA3	12	40	\$39,680

Example 9: Joining Three Tables

Procedure features:

FROM clause

joined-table component

WHERE clause

Tables: PROCLIB.STAFF2, PROCLIB.SCHEDULE2, PROCLIB.SUPERV2

This example joins three tables and produces a report that contains columns from each table.

Input Tables

PROCLIB.STAFF2

PROCLIB.STAFF2					
Id Num	Lname	Fname	City	State	Hphone
1106	MARSHBURN	JASPER	STAMFORD	CT	203/781-1457
1430	DABROWSKI	SANDRA	BRIDGEPORT	CT	203/675-1647
1118	DENNIS	ROGER	NEW YORK	NY	718/383-1122
1126	KIMANI	ANNE	NEW YORK	NY	212/586-1229
1402	BLALOCK	RALPH	NEW YORK	NY	718/384-2849
1882	TUCKER	ALAN	NEW YORK	NY	718/384-0216
1479	BALLETTI	MARIE	NEW YORK	NY	718/384-8816
1420	ROUSE	JEREMY	PATERSON	NJ	201/732-9834
1403	BOWDEN	EARL	BRIDGEPORT	CT	203/675-3434
1616	FUENTAS	CARLA	NEW YORK	NY	718/384-3329

PROCLIB.SCHEDULE2

PROCLIB.SCHEDULE2				
Flight	Date	Dest	Id Num	
132	01MAR94	BOS	1118	
132	01MAR94	BOS	1402	
219	02MAR94	PAR	1616	
219	02MAR94	PAR	1478	
622	03MAR94	LON	1430	
622	03MAR94	LON	1882	
271	04MAR94	NYC	1430	
271	04MAR94	NYC	1118	
579	05MAR94	RDU	1126	
579	05MAR94	RDU	1106	

PROCLIB.SUPERV2

PROCLIB.SUPERV2		
Supervisor Id	State	Job Category
1417	NJ	NA
1352	NY	NA
1106	CT	PT
1442	NJ	PT
1118	NY	PT
1405	NJ	SC
1564	NY	SC
1639	CT	TA
1126	NY	TA
1882	NY	ME

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Select the columns. The SELECT clause specifies the columns to select. IdNum is prefixed with a table alias because it appears in two tables.

```
proc sql;
  title 'All Flights for Each Supervisor';
  select s.IdNum, Lname, City 'Hometown', Jobcat,
         Flight, Date
```

Specify the tables to include in the join. The FROM clause lists the three tables for the join and assigns an alias to each table.

```
  from proclib.schedule2 s, proclib.staff2 t, proclib.superv2 v
```

Specify the join criteria. The WHERE clause specifies the columns that join the tables. The STAFF2 and SCHEDULE2 tables have an IdNum column, which has related values in both tables. The STAFF2 and SUPERV2 tables have the IdNum and SUPID columns, which have related values in both tables.

```
  where s.idnum=t.idnum and t.idnum=v.supid;
```

Output

All Flights for Each Supervisor					
Id Num	Lname	Hometown	Job Category	Flight	Date
1106	MARSHBURN	STAMFORD	PT	579	05MAR94
1118	DENNIS	NEW YORK	PT	132	01MAR94
1118	DENNIS	NEW YORK	PT	271	04MAR94
1126	KIMANI	NEW YORK	TA	579	05MAR94
1882	TUCKER	NEW YORK	ME	622	03MAR94

Example 10: Querying an In-Line View

Procedure features:

FROM clause
in-line view

Tables: PROCLIB.STAFF, PROCLIB.SCHEDULE, PROCLIB.SUPERV

This example uses the query explained in Example 9 on page 1227 as an in-line view. The example also shows how to rename columns with an in-line view.

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Select the columns. The SELECT clause selects all columns that are returned by the query in the FROM clause.

```
proc sql outobs=10;
  title 'All Flights for Each Supervisor';
  select *
```

Specify the join as an in-line query. Instead of including the name of a table or view, the FROM clause includes the query that joins the three tables. In the in-line query, the SELECT clause lists the columns to select. IdNum is prefixed with a table alias because it appears in two tables. The FROM clause lists the three tables for the join and assigns an alias to each table. The WHERE clause specifies the columns that join the tables. The STAFF2 and SCHEDULE2 tables have an IdNum column, which has related values in both tables. The STAFF2 and SUPERV2 tables have the IdNum and SUPID columns, which have related values in both tables.

```
    from (select lname, s.idnum, city, jobcat,
               flight, date
          from proclib.schedule2 s, proclib.staff2 t,
               proclib.superv2 v
          where s.idnum=t.idnum and t.idnum=v.supid)
```


Specify an alias for the query and names for the columns. The alias **THREE** refers to the entire query. The names in parentheses become the names for the columns in the output. The label **Job Category** appears in the output instead of the name **Jobtype** because PROC SQL prints a column's label if the column has a label.

```
as three (Surname, Emp_ID, Hometown,
         Jobtype, FlightNumber, FlightDate);
```

Output

All Flights for Each Supervisor					
Surname	Emp_ID	Hometown	Job Category	FlightNumber	FlightDate
MARSHBURN	1106	STAMFORD	PT	579	05MAR94
DENNIS	1118	NEW YORK	PT	132	01MAR94
DENNIS	1118	NEW YORK	PT	271	04MAR94
KIMANI	1126	NEW YORK	TA	579	05MAR94
TUCKER	1882	NEW YORK	ME	622	03MAR94

Example 11: Retrieving Values with the SOUNDS-LIKE Operator

Procedure features:

ORDER BY clause

SOUNDS-LIKE operator

Table: PROCLIB.STAFF

This example returns rows based on the functionality of the SOUNDS-LIKE operator in a WHERE clause.

Note: The SOUNDS-LIKE operator is based on the SOUNDEX algorithm for identifying words that sound alike. The SOUNDEX algorithm is English-biased and is less useful for languages other than English. For more information on the SOUNDEX algorithm, see *SAS Language Reference: Dictionary*. Δ

Input Table

PROCLIB.STAFF

PROCLIB.STAFF First 10 Rows Only					
Id Num	Lname	Fname	City	State	Hphone
1919	ADAMS	GERALD	STAMFORD	CT	203/781-1255
1653	ALIBRANDI	MARIA	BRIDGEPORT	CT	203/675-7715
1400	ALHERTANI	ABDULLAH	NEW YORK	NY	212/586-0808
1350	ALVAREZ	MERCEDES	NEW YORK	NY	718/383-1549
1401	ALVAREZ	CARLOS	PATERSON	NJ	201/732-8787
1499	BAREFOOT	JOSEPH	PRINCETON	NJ	201/812-5665
1101	BAUCOM	WALTER	NEW YORK	NY	212/586-8060
1333	BANADYGA	JUSTIN	STAMFORD	CT	203/781-1777
1402	BLALOCK	RALPH	NEW YORK	NY	718/384-2849
1479	BALLETTI	MARIE	NEW YORK	NY	718/384-8816

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-library';
```

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Select the columns and the table from which the data is obtained. The SELECT clause selects all columns from the table in the FROM clause, PROCLIB.STAFF.

```
proc sql;
  title "Employees Whose Last Name Sounds Like 'Johnson'";
  select idnum, upcase(lname), fname
  from proclib.staff
```

Subset the query and sort the output. The WHERE clause uses the SOUNDS-LIKE operator to subset the table by those employees whose last name sounds like **Johnson**. The ORDER BY clause orders the output by the second column.

```
  where lname="Johnson"
  order by 2;
```

Output

Employees Whose Last Name Sounds Like 'Johnson'			1
Id		Fname	
Num			

1411	JOHNSEN	JACK	
1113	JOHNSON	LESLIE	
1369	JONSON	ANTHONY	

SOUNDS-LIKE is useful, but there might be instances where it does not return every row that seems to satisfy the condition. PROCLIB.STAFF has an employee with the last name **SANDERS** and an employee with the last name **SANYERS**. The algorithm does not find **SANYERS**, but it does find **SANDERS** and **SANDERSON**.

```
title "Employees Whose Last Name Sounds Like 'Sanders'";
select *
  from proclib.staff
 where lname="Sanders"
 order by 2;
```

Employees Whose Last Name Sounds Like 'Sanders'						2
Id						
	Num	Lname	Fname	City	State	Hphone

1561	SANDERS	RAYMOND	NEW YORK	NY	212/588-6615	
1414	SANDERSON	NATHAN	BRIDGEPORT	CT	203/675-1715	
1434	SANDERSON	EDITH	STAMFORD	CT	203/781-1333	

Example 12: Joining Two Tables and Calculating a New Value

Procedure features:

GROUP BY clause

HAVING clause

SELECT clause

ABS function

FORMAT= column-modifier

LABEL= column-modifier

MIN summary function

** operator, exponentiation

SQRT function

Tables: STORES, HOUSES

This example joins two tables in order to compare and analyze values that are unique to each table yet have a relationship with a column that is common to both tables.

```
options ls=80 ps=60 nodate pageno=1 ;
data stores;
  input Store $ x y;
  datalines;
store1 5 1
store2 5 3
store3 3 5
store4 7 5
;
data houses;
  input House $ x y;
  datalines;
house1 1 1
house2 3 3
house3 2 3
house4 7 7
;
```

Input Tables

STORES and HOUSES

The tables contain X and Y coordinates that represent the location of the stores and houses.

STORES Table			1
Coordinates of Stores			
Store	x	y	

store1	6	1	
store2	5	2	
store3	3	5	
store4	7	5	

HOUSES Table			2
Coordinates of Houses			
House	x	y	

house1	1	1	
house2	3	3	
house3	2	3	
house4	7	7	

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Specify the query. The SELECT clause specifies three columns: HOUSE, STORE, and DIST. The arithmetic expression uses the square root function (SQRT) to create the values of DIST, which contain the distance from HOUSE to STORE for each row. The double asterisk (**) represents exponentiation. LABEL= assigns a label to STORE and to DIST.

```
proc sql;
  title 'Each House and the Closest Store';
  select house, store label='Closest Store',
         sqrt((abs(s.x-h.x)**2)+(abs(h.y-s.y)**2)) as dist
         label='Distance' format=4.2
  from stores s, houses h
```

Organize the data into groups and subset the query. The minimum distance from each house to all the stores is calculated because the data are grouped by house. The HAVING clause specifies that each row be evaluated to determine if its value of DIST is the same as the minimum distance from that house to any store.

```
group by house
having dist=min(dist);
```

Output

Note that two stores are tied for shortest distance from house2.

Each House and the Closest Store			1
House	Closest Store	Distance	
house1	store1	4.00	
house2	store2	2.00	
house2	store3	2.00	
house3	store3	2.24	
house4	store4	2.00	

Example 13: Producing All the Possible Combinations of the Values in a Column

Procedure features:

CASE expression
 joined-table component
 SELECT clause
 DISTINCT keyword

Tables: PROCLIB.MARCH, FLIGHTS

This example joins a table with itself to get all the possible combinations of the values in a column.

Input Table

PROCLIB.MARCH (Partial Listing)

PROCLIB.MARCH							
First 10 Rows Only							
Flight	Date	Depart	Orig	Dest	Miles	Boarded	Capacity
114	01MAR94	7:10	LGA	LAX	2475	172	210
202	01MAR94	10:43	LGA	ORD	740	151	210
219	01MAR94	9:31	LGA	LON	3442	198	250
622	01MAR94	12:19	LGA	FRA	3857	207	250
132	01MAR94	15:35	LGA	YYZ	366	115	178
271	01MAR94	13:17	LGA	PAR	3635	138	250
302	01MAR94	20:22	LGA	WAS	229	105	180
114	02MAR94	7:10	LGA	LAX	2475	119	210
202	02MAR94	10:43	LGA	ORD	740	120	210
219	02MAR94	9:31	LGA	LON	3442	147	250

Program

Declare the PROCLIB library. The PROCLIB library is used in these examples to store created tables.

```
libname proclib 'SAS-data-library';
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the FLIGHTS table. The CREATE TABLE statement creates the table FLIGHTS from the output of the query. The SELECT clause selects the unique values of Dest. DISTINCT specifies that only one row for each value of city be returned by the query and stored in the table FLIGHTS. The FROM clause specifies PROCLIB.MARCH as the table to select from.

```
proc sql;
  create table flights as
    select distinct dest
      from proclib.march;
```

Specify the title.

```
title 'Cities Serviced by the Airline';
```

Display the entire FLIGHTS table.

```
select * from flights;
```

Output

FLIGHTS Table

Cities Serviced by the Airline		1
	Dest	

	FRA	
	LAX	
	LON	
	ORD	
	PAR	
	WAS	
	YYZ	

Specify the title.

```
title 'All Possible Connections';
```

Select the columns. The SELECT clause specifies three columns for the output. The prefixes on DEST are table aliases to specify which table to take the values of Dest from. The CASE expression creates a column that contains the character string **to and from**.

```
select f1.Dest, case
                  when f1.dest ne ' ' then 'to and from'
                end,
       f2.Dest
```

Specify the type of join. The FROM clause joins FLIGHTS with itself and creates a table that contains every possible combination of rows. The table contains two rows for each possible route, for example, **PAR <-> WAS** and **WAS <-> PAR**.

```
from flights as f1, flights as f2
```

Specify the join criterion. The WHERE clause subsets the internal table by choosing only those rows where the name in F1.Dest sorts before the name in F2.Dest. Thus, there is only one row for each possible route.

```
where f1.dest < f2.dest
```

Sort the output. ORDER BY sorts the result by the values of F1.Dest.

```
order by f1.dest;
```

Output

All Possible Connections				2
Dest		Dest		
-----		-----		
FRA	to and from	LAX		
FRA	to and from	LON		
FRA	to and from	WAS		
FRA	to and from	ORD		
FRA	to and from	PAR		
FRA	to and from	YYZ		
LAX	to and from	LON		
LAX	to and from	PAR		
LAX	to and from	WAS		
LAX	to and from	ORD		
LAX	to and from	YYZ		
LON	to and from	ORD		
LON	to and from	WAS		
LON	to and from	PAR		
LON	to and from	YYZ		
ORD	to and from	WAS		
ORD	to and from	PAR		
ORD	to and from	YYZ		
PAR	to and from	WAS		
PAR	to and from	YYZ		
WAS	to and from	YYZ		

Example 14: Matching Case Rows and Control Rows

Procedure features:

joined-table component

Tables: MATCH_11 on page 1641, MATCH

This example uses a table that contains data for a case-control study. Each row contains information for a case or a control. To perform statistical analysis, you need a

table with one row for each case-control pair. PROC SQL joins the table with itself in order to match the cases with their appropriate controls. After the rows are matched, differencing can be performed on the appropriate columns.

The input table MATCH_11 contains one row for each case and one row for each control. Pair contains a number that associates the case with its control. Low is 0 for the controls and 1 for the cases. The remaining columns contain information about the cases and controls.

Input Table

MATCH_11 Table										
First 10 Rows Only										
Pair	Low	Age	Lwt	Race	Smoke	Ptd	Ht	UI	racel	race2
1	0	14	135	1	0	0	0	0	0	0
1	1	14	101	3	1	1	0	0	0	1
2	0	15	98	2	0	0	0	0	1	0
2	1	15	115	3	0	0	0	1	0	1
3	0	16	95	3	0	0	0	0	0	1
3	1	16	130	3	0	0	0	0	0	1
4	0	17	103	3	0	0	0	0	0	1
4	1	17	130	3	1	1	0	1	0	1
5	0	17	122	1	1	0	0	0	0	0
5	1	17	110	1	1	0	0	0	0	0

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the MATCH table. The SELECT clause specifies the columns for the table MATCH. SQL expressions in the SELECT clause calculate the differences for the appropriate columns and create new columns.

```
proc sql;
  create table match as
  select
    one.Low,
    one.Pair,
    (one.lwt - two.lwt) as Lwt_d,
    (one.smoke - two.smoke) as Smoke_d,
    (one.ptd - two.ptd) as Ptd_d,
    (one.ht - two.ht) as Ht_d,
    (one.ui - two.ui) as UI_d
```

Specify the type of join and the join criterion. The FROM clause lists the table MATCH_11 twice. Thus, the table is joined with itself. The WHERE clause returns only the rows for each pair that show the difference when the values for control are subtracted from the values for case.

```
from match_11 one, match_11 two
where (one.pair=two.pair and one.low>two.low);
```

Specify the title.

```
title 'Differences for Cases and Controls';
```

Display the first five rows of the MATCH table. The SELECT clause selects all the columns from MATCH. The OBS= data set option limits the printing of the output to five rows.

```
select *
from match(obs=5);
```

Output

MATCH Table

Differences for Cases and Controls							1
Low	Pair	Lwt_d	Smoke_d	Ptd_d	Ht_d	UI_d	
1	1	-34	1	1	0	0	
1	2	17	0	0	0	1	
1	3	35	0	0	0	0	
1	4	27	1	1	0	1	
1	5	-12	0	0	0	0	

Example 15: Counting Missing Values with a SAS Macro

Procedure feature:

COUNT function

Table: SURVEY

This example uses a SAS macro to create columns. The SAS macro is not explained here. See the *SAS Macro Language: Reference* for information on SAS macros.

Input Table

SURVEY contains data from a questionnaire about diet and exercise habits. SAS enables you to use a special notation for missing values. In the EDUC column, the **.x** notation indicates that the respondent gave an answer that is not valid, and **.n** indicates that the respondent did not answer the question. A period as a missing value indicates a data entry error.

```
data survey;
  input id $ diet $ exer $ hours xwk educ;
  datalines;
1001 yes yes 1 3 1
1002 no yes 1 4 2
1003 no no . . .n
1004 yes yes 2 3 .x
1005 no yes 2 3 .x
1006 yes yes 2 4 .x
1007 no yes .5 3 .
1008 no no . . .
;
```

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Count the nonmissing responses. The COUNTM macro uses the COUNT function to perform various counts for a column. Each COUNT function uses a CASE expression to select the rows to be counted. The first COUNT function uses only the column as an argument to return the number of nonmissing rows.

```
%macro countm(col);
  count(&col) "Valid Responses for &col",
```

Count missing or invalid responses. The NMSS function returns the number of rows for which the column has any type of missing value: **.n**, **.x**, or a period.

```
nmss(&col) "Missing or NOT VALID Responses for &col",
```

Count the occurrences of various sources of missing or invalid responses. The last three COUNT functions use CASE expressions to count the occurrences of the three notations for missing values. The “count me” character string gives the COUNT function a nonmissing value to count.

```
count(case
  when &col=.n then "count me"
end) "Coded as NO ANSWER for &col",
count(case
  when &col=.x then "count me"
end) "Coded as NOT VALID answers for &col",
```

```

count(case
    when &col=. then "count me"
    end) "Data Entry Errors for &col"
%mend;

```

Use the COUNTM macro to create the columns. The SELECT clause specifies the columns that are in the output. COUNT(*) returns the total number of rows in the table. The COUNTM macro uses the values of the EDUC column to create the columns that are defined in the macro.

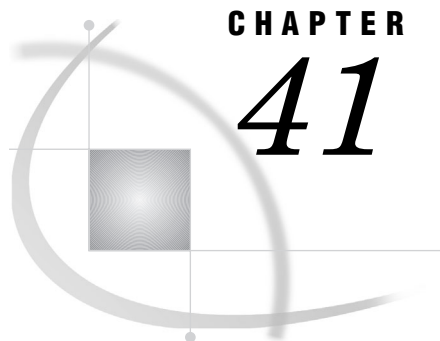
```

proc sql;
    title 'Counts for Each Type of Missing Response';
    select count(*) "Total No. of Rows",
           %countm(educ)
    from survey;

```

Output

Counts for Each Type of Missing Response						1
Total No. of Rows	Valid Responses for educ	Missing or NOT VALID Responses for educ	Coded as NO ANSWER for educ	Coded as NOT VALID answers for educ	Data Entry Errors for educ	
8	2	6	1	3	2	



CHAPTER

41

The STANDARD Procedure

Overview: <i>STANDARD</i> Procedure	1243
Syntax: <i>STANDARD</i> Procedure	1245
<i>PROC STANDARD</i> Statement	1246
<i>BY</i> Statement	1248
<i>FREQ</i> Statement	1248
<i>VAR</i> Statement	1249
<i>WEIGHT</i> Statement	1249
Results: <i>STANDARD</i> Procedure	1250
Missing Values	1250
Output Data Set	1250
Statistical Computations: <i>STANDARD</i> Procedure	1250
Examples: <i>STANDARD</i> Procedure	1251
Example 1: Standardizing to a Given Mean and Standard Deviation	1251
Example 2: Standardizing <i>BY</i> Groups and Replacing Missing Values	1253

Overview: STANDARD Procedure

The *STANDARD* procedure standardizes variables in a SAS data set to a given mean and standard deviation, and it creates a new SAS data set containing the standardized values.

Output 41.1 on page 1243 shows a simple standardization where the output data set contains standardized student exam scores. The statements that produce the output follow:

```
proc standard data=score mean=75 std=5
               out=stndtest;

run;

proc print data=stndtest;
run;
```

Output 41.1 Standardized Test Scores Using PROC STANDARD

The SAS System			1
Obs	Student	Test1	
1	Capalleti	80.5388	
2	Dubose	64.3918	
3	Engles	80.9143	
4	Grant	68.8980	
5	Krupski	75.2816	
6	Lundsford	79.7877	
7	Mcbane	73.4041	
8	Mullen	78.6612	
9	Nguyen	74.9061	
10	Patel	71.9020	
11	Si	73.4041	
12	Tanaka	77.9102	

Output 41.2 on page 1244 shows a more complex example that uses BY-group processing. PROC STANDARD computes Z scores separately for two BY groups by standardizing life-expectancy data to a mean of 0 and a standard deviation of 1. The data are 1950 and 1993 life expectancies at birth for 16 countries. The birth rates for each country, classified as stable or rapid, form the two BY groups. The statements that produce the analysis also

- ☐ print statistics for each variable to standardize
- ☐ replace missing values with the given mean
- ☐ calculate standardized values using a given mean and standard deviation
- ☐ print the data set with the standardized values.

For an explanation of the program that produces this output, see Example 2 on page 1253.

Output 41.2 Z Scores for Each BY Group Using PROC STANDARD

Life Expectancies by Birth Rate				2
----- PopulationRate=Stable -----				
The STANDARD Procedure				
Name Label	Mean	Standard Deviation	N	
Life50 1950 life expectancy	67.400000	1.854724	5	
Life93 1993 life expectancy	74.500000	4.888763	6	
----- PopulationRate=Rapid -----				
Name Label	Mean	Standard Deviation	N	
Life50 1950 life expectancy	42.000000	5.033223	8	
Life93 1993 life expectancy	59.100000	8.225300	10	

Standardized Life Expectancies at Birth by a Country's Birth Rate				3
Population Rate	Country	Life50	Life93	
Stable	France	-0.21567	0.51138	
Stable	Germany	0.32350	0.10228	
Stable	Japan	-1.83316	0.92048	
Stable	Russia	0.00000	-1.94323	
Stable	United Kingdom	0.86266	0.30683	
Stable	United States	0.86266	0.10228	
Rapid	Bangladesh	0.00000	-0.74161	
Rapid	Brazil	1.78812	0.96045	
Rapid	China	-0.19868	1.32518	
Rapid	Egypt	0.00000	0.10942	
Rapid	Ethiopia	-1.78812	-1.59265	
Rapid	India	-0.59604	-0.01216	
Rapid	Indonesia	-0.79472	-0.01216	
Rapid	Mozambique	0.00000	-1.47107	
Rapid	Philippines	1.19208	0.59572	
Rapid	Turkey	0.39736	0.83888	

Syntax: STANDARD Procedure

Tip: Supports the Output Delivery System. See “Output Delivery System” on page 32 for details.

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, “Statements with the Same Function in Multiple Procedures,” on page 53 for details. You can also use any global statements as well. See “Global Statements” on page 18 for a list.

```

PROC STANDARD <option(s)>;
  BY <DESCENDING> variable-1 <...<DESCENDING> variable-n>
    <NOTSORTED>;
  FREQ variable;
  VAR variable(s);
  WEIGHT variable;

```

To do this	Use this statement
Calculate separate standardized values for each BY group	BY
Identify a variable whose values represent the frequency of each observation	FREQ
Select the variables to standardize and determine the order in which they appear in the printed output	VAR
Identify a variable whose values weight each observation in the statistical calculations	WEIGHT

PROC STANDARD Statement

PROC STANDARD <option(s)>;

To do this	Use this option
Specify the input data set	DATA=
Specify the output data set	OUT=
Computational options	
Exclude observations with nonpositive weights	EXCLNPWGT
Specify the mean value	MEAN=
Replace missing values with a variable mean or MEAN= value	REPLACE
Specify the standard deviation value	STD=
Specify the divisor for variance calculations	VARDEF=
Control printed output	
Print statistics for each variable to standardize	PRINT

Without Options

If you do not specify MEAN=, REPLACE, or STD=, the output data set is an identical copy of the input data set.

Options

DATA=SAS-data-set

identifies the input SAS data set.

Main discussion: “Input Data Sets” on page 19

Restriction: You cannot use PROC STANDARD with an engine that supports concurrent access if another user is updating the data set at the same time.

EXCLNPWGT

excludes observations with nonpositive weight values (zero or negative). The procedure does not use the observation to calculate the mean and standard deviation, but the observation is still standardized. By default, the procedure treats observations with negative weights like those with zero weights and counts them in the total number of observations.

MEAN=mean-value

standardizes variables to a mean of *mean-value*.

Alias: M=

Default: mean of the input values

Featured in: Example 1 on page 1251

OUT=SAS-data-set

identifies the output data set. If *SAS-data-set* does not exist, PROC STANDARD creates it. If you omit OUT=, the data set is named DATA n , where n is the smallest integer that makes the name unique.

Default: DATA n

Featured in: Example 1 on page 1251

PRINT

prints the original frequency, mean, and standard deviation for each variable to standardize.

Featured in: Example 2 on page 1253

REPLACE

replaces missing values with the variable mean.

Interaction: If you use MEAN=, PROC STANDARD replaces missing values with the given mean.

Featured in: Example 2 on page 1253

STD=std-value

standardizes variables to a standard deviation of *std-value*.

Alias: S=

Default: standard deviation of the input values

Featured in: Example 1 on page 1251

VARDEF=divisor

specifies the divisor to use in the calculation of variances and standard deviation. Table 41.1 on page 1247 shows the possible values for *divisor* and the associated divisors.

Table 41.1 Possible Values for VARDEF=

Value	Divisor	Formula for Divisor
DF	degrees of freedom	$n - 1$
N	number of observations	n
WDF	sum of weights minus one	$(\sum_i w_i) - 1$
WEIGHT WGT	sum of weights	$\sum_i w_i$

The procedure computes the variance as $CSS/divisor$, where CSS is the corrected sums of squares and equals $\sum_i (x_i - \bar{x})^2$. When you weight the analysis variables, CSS equals $\sum w_i (x_i - \bar{x}_w)^2$ where \bar{x}_w is the weighted mean.

Default: DF

Tip: When you use the WEIGHT statement and VARDEF=DF, the variance is an estimate of σ^2 , where the variance of the i th observation is $var(x_i) = \sigma^2/w_i$ and w_i is the weight for the i th observation. This yields an estimate of the variance of an observation with unit weight.

Tip: When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large n) an estimate of σ^2/\bar{w} , where \bar{w} is the average weight. This yields an asymptotic estimate of the variance of an observation with average weight.

See also: “WEIGHT” on page 59

Main discussion: “Keywords and Formulas” on page 1578

BY Statement

Calculates standardized values separately for each BY group.

Main discussion: “BY” on page 54

Featured in: Example 2 on page 1253

BY <DESCENDING> *variable-1* <...<DESCENDING> *variable-n*><NOTSORTED>;

Required Arguments

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, the observations in the data set must either be sorted by all the variables that you specify, or they must be indexed appropriately. These variables are called *BY variables*.

Options

DESCENDING

specifies that the data set is sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The data are grouped in another way, such as chronological order.

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, the procedure treats each contiguous set as a separate BY group.

FREQ Statement

Specifies a numeric variable whose values represent the frequency of the observation.

Tip: The effects of the FREQ and WEIGHT statements are similar except when calculating degrees of freedom.

See also: For an example that uses the FREQ statement, see “FREQ” on page 56

FREQ *variable*;

Required Arguments

variable

specifies a numeric variable whose value represents the frequency of the observation. If you use the FREQ statement, the procedure assumes that each observation represents n observations, where n is the value of *variable*. If n is not an integer, the SAS System truncates it. If n is less than 1 or is missing, the procedure does not use that observation to calculate statistics but the observation is still standardized.

The sum of the frequency variable represents the total number of observations.

VAR Statement

Specifies the variables to standardize and their order in the printed output.

Default: If you omit the VAR statement, PROC STANDARD standardizes all numeric variables not listed in the other statements.

Featured in: Example 1 on page 1251

VAR *variable(s)*;

Required Arguments

variable(s)

identifies one or more variables to standardize.

WEIGHT Statement

Specifies weights for analysis variables in the statistical calculations.

See also: For information about calculating weighted statistics and for an example that uses the WEIGHT statement, see “WEIGHT” on page 59

WEIGHT *variable*;

Required Arguments

variable

specifies a numeric variable whose values weight the values of the analysis variables. The values of the variable do not have to be integers. If the value of the weight variable is

Weight value...	PROC STANDARD...
0	counts the observation in the total number of observations
less than 0	converts the weight value to zero and counts the observation in the total number of observations
missing	excludes the observation from the calculation of mean and standard deviation

To exclude observations that contain negative and zero weights from the calculation of mean and standard deviation, use EXCLNPWGT. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default.

Tip: When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See VARDEF= on page 1247 and the calculation of weighted statistics in “Keywords and Formulas” on page 1578 for more information.

Note: Prior to Version 7 of the SAS System, the procedure did not exclude the observations with missing weights from the count of observations. Δ

Results: STANDARD Procedure

Missing Values

By default, PROC STANDARD excludes missing values for the analysis variables from the standardization process, and the values remain missing in the output data set. When you specify the REPLACE option, the procedure replaces missing values with the variable's mean or the MEAN= value.

If the value of the WEIGHT variable or the FREQ variable is missing then the procedure does not use the observation to calculate the mean and the standard deviation. However, the observation is standardized.

Output Data Set

PROC STANDARD always creates an output data set that stores the standardized values in the VAR statement variables, regardless of whether you specify the OUT= option. The output data set contains all the input data set variables, including those not standardized. PROC STANDARD does not print the output data set. Use PROC PRINT, PROC REPORT, or another SAS reporting tool to print the output data set.

Statistical Computations: STANDARD Procedure

Standardizing values removes the location and scale attributes from a set of data. The formula to compute standardized values is

$$x'_i = \frac{S * (x_i - \bar{x})}{s_x} + M$$

where

x'_i	is a new standardized value
S	is the value of STD=
M	is the value of MEAN=
x_i	is an observation's value
\bar{x}	is a variable's mean
s_x	is a variable's standard deviation.

PROC STANDARD calculates the mean (\bar{x}) and standard deviation (s_x) from the input data set. The resulting standardized variable has a mean of M and a standard deviation of S .

If the data are normally distributed, standardizing is also studentizing since the resulting data have a Student's t distribution.

Examples: STANDARD Procedure

Example 1: Standardizing to a Given Mean and Standard Deviation

Procedure features:

PROC STANDARD statement options:

MEAN=

OUT=

STD=

VAR statement

Other features:

PRINT procedure

This example

- ☐ standardizes two variables to a mean of 75 and a standard deviation of 5
- ☐ specifies the output data set
- ☐ combines standardized variables with original variables
- ☐ prints the output data set.

Program

Set the SAS system options. The NODATE option specifies to omit the date and time when the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines for a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the SCORE data set. This data set contains test scores for students who took two tests and a final exam. The FORMAT statement assigns the *Zw.d* format to StudentNumber. This format pads right-justified output with 0s instead of blanks. The LENGTH statement specifies the number of bytes to use to store values of Student.

```
data score;
    length Student $ 9;
    input Student $ StudentNumber Section $
           Test1 Test2 Final @@;
    format studentnumber z4.;
    datalines;
Capalleti 0545 1 94 91 87  Dubose      1252 2 51 65 91
Engles    1167 1 95 97 97  Grant       1230 2 63 75 80
Krupski   2527 2 80 69 71  Lundsford  4860 1 92 40 86
Mcbane    0674 1 75 78 72  Mullen    6445 2 89 82 93
Nguyen    0886 1 79 76 80  Patel      9164 2 71 77 83
Si        4915 1 75 71 73  Tanaka     8534 2 87 73 76
;
```

Generate the standardized data and create the output data set STNDTEST. PROC STANDARD uses a mean of 75 and a standard deviation of 5 to standardize the values. OUT= identifies STNDTEST as the data set to contain the standardized values.

```
proc standard data=score mean=75 std=5 out=stndtest;
```

Specify the variables to standardize. The VAR statement specifies the variables to standardize and their order in the output.

```
    var test1 test2;
run;
```

Create a data set that combines the original values with the standardized values. PROC SQL joins SCORE and STNDTEST to create the COMBINED data set (table) that contains standardized and original test scores for each student. Using AS to rename the standardized variables NEW.TEST1 to StdTest1 and NEW.TEST2 to StdTest2 makes the variable names unique.

```
proc sql;
    create table combined as
    select old.student, old.studentnumber,
           old.section,
           old.test1, new.test1 as StdTest1,
           old.test2, new.test2 as StdTest2,
           old.final
    from score as old, stndtest as new
    where old.student=new.student;
```

Print the data set. PROC PRINT prints the COMBINED data set. ROUND rounds the standardized values to two decimal places. The TITLE statement specifies a title.

```
proc print data=combined noobs round;
    title 'Standardized Test Scores for a College Course';
run;
```

Output

The data set contains variables with both standardized and original values. StdTest1 and StdTest2 store the standardized test scores that PROC STANDARD computes.

Standardized Test Scores for a College Course							1
Student	Student Number	Section	Test1	Std Test1	Test2	Std Test2	Final
Capalleti	0545	1	94	80.54	91	80.86	87
Dubose	1252	2	51	64.39	65	71.63	91
Engles	1167	1	95	80.91	97	82.99	97
Grant	1230	2	63	68.90	75	75.18	80
Krupski	2527	2	80	75.28	69	73.05	71
Lundsford	4860	1	92	79.79	40	62.75	86
Mcbane	0674	1	75	73.40	78	76.24	72
Mullen	6445	2	89	78.66	82	77.66	93
Nguyen	0886	1	79	74.91	76	75.53	80
Patel	9164	2	71	71.90	77	75.89	83
Si	4915	1	75	73.40	71	73.76	73
Tanaka	8534	2	87	77.91	73	74.47	76

Example 2: Standardizing BY Groups and Replacing Missing Values

Procedure features:

PROC STANDARD statement options:

PRINT
REPLACE

BY statement

Other features:

FORMAT procedure

PRINT procedure

SORT procedure

This example

- ☐ calculates Z scores separately for each BY group using a mean of 1 and standard deviation of 0

- replaces missing values with the given mean
- prints the mean and standard deviation for the variables to standardize
- prints the output data set.

Program

Set the SAS system options. The NODATE option specifies to omit the date and time when the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines for a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Assign a character string format to a numeric value. PROC FORMAT creates the format POPFMT to identify birth rates with a character value.

```
proc format;
    value popfmt 1='Stable'
                2='Rapid';
run;
```

Create the LIFEEXP data set. Each observation in this data set contains information on 1950 and 1993 life expectancies at birth for 16 nations.* The birth rate for each nation is classified as stable (1) or rapid (2). The nations with missing data obtained independent status after 1950.

```
data lifexp;
    input PopulationRate Country $char14. Life50 Life93 @@;
    label life50='1950 life expectancy'
          life93='1993 life expectancy';
    datalines;
2 Bangladesh      .  53 2 Brazil          51 67
2 China            41 70 2 Egypt          42 60
2 Ethiopia         33 46 1 France         67 77
1 Germany          68 75 2 India           39 59
2 Indonesia        38 59 1 Japan           64 79
2 Mozambique       .  47 2 Philippines    48 64
1 Russia           .  65 2 Turkey          44 66
1 United Kingdom  69 76 1 United States    69 75
;
```

Sort the LIFEEXP data set. PROC SORT sorts the observations by the birth rate.

```
proc sort data=lifexp;
    by populationrate;
```

* Data are from *Vital Signs 1994: The Trends That Are Shaping Our Future*, Lester R. Brown, Hal Kane, and David Malin Roodman, eds. Copyright © 1994 by Worldwatch Institute. Reprinted by permission of W.W. Norton & Company, Inc.


```
run;
```

Generate the standardized data for all numeric variables and create the output data set ZSCORE. PROC STANDARD standardizes all numeric variables to a mean of 1 and a standard deviation of 0. REPLACE replaces missing values. PRINT prints statistics.

```
proc standard data=lifexp mean=0 std=1 replace
      print out=zscore;
```

Create the standardized values for each BY group. The BY statement standardizes the values separately by birth rate.

```
      by populationrate;
```

Assign a format to a variable and specify a title for the report. The FORMAT statement assigns a format to PopulationRate. The output data set contains formatted values. The TITLE statement specifies a title.

```
      format populationrate popfmt.;
      title1 'Life Expectancies by Birth Rate';
run;
```

Print the data set. PROC PRINT prints the ZSCORE data set with the standardized values. The TITLE statements specify two titles to print.

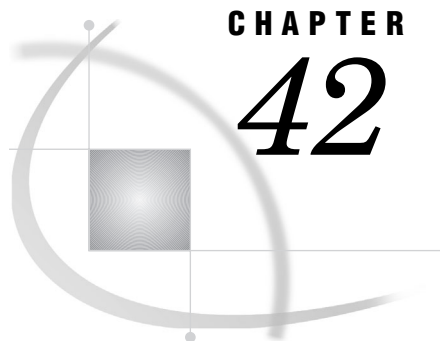
```
proc print data=zscore noobs;
      title 'Standardized Life Expectancies at Birth';
      title2 'by a Country''s Birth Rate';
run;
```

Output

PROC STANDARD prints the variable name, mean, standard deviation, input frequency, and label of each variable to standardize for each BY group.

Life expectancies for Bangladesh, Mozambique, and Russia are no longer missing. The missing values are replaced with the given mean (0).

Life Expectancies by Birth Rate					1
----- PopulationRate=Stable -----					
Name	Mean	Standard Deviation	N	Label	
Life50	67.400000	1.854724	5	1950 life expectancy	
Life93	74.500000	4.888763	6	1993 life expectancy	
----- PopulationRate=Rapid -----					
Name	Mean	Standard Deviation	N	Label	
Life50	42.000000	5.033223	8	1950 life expectancy	
Life93	59.100000	8.225300	10	1993 life expectancy	
Standardized Life Expectancies at Birth by a Country's Birth Rate					2
Population Rate	Country	Life50	Life93		
Stable	France	-0.21567	0.51138		
Stable	Germany	0.32350	0.10228		
Stable	Japan	-1.83316	0.92048		
Stable	Russia	0.00000	-1.94323		
Stable	United Kingdom	0.86266	0.30683		
Stable	United States	0.86266	0.10228		
Rapid	Bangladesh	0.00000	-0.74161		
Rapid	Brazil	1.78812	0.96045		
Rapid	China	-0.19868	1.32518		
Rapid	Egypt	0.00000	0.10942		
Rapid	Ethiopia	-1.78812	-1.59265		
Rapid	India	-0.59604	-0.01216		
Rapid	Indonesia	-0.79472	-0.01216		
Rapid	Mozambique	0.00000	-1.47107		
Rapid	Philippines	1.19208	0.59572		
Rapid	Turkey	0.39736	0.83888		



CHAPTER

42

The SUMMARY Procedure

Overview: SUMMARY Procedure	1257
Syntax: SUMMARY Procedure	1257
PROC SUMMARY Statement	1258
VAR Statement	1258

Overview: SUMMARY Procedure

The SUMMARY procedure provides data summarization tools that compute descriptive statistics for variables across all observations or within groups of observations. The SUMMARY procedure is very similar to the MEANS procedure; for full syntax details, see Chapter 26, “The MEANS Procedure,” on page 649. Except for the differences that are discussed here, all the PROC MEANS information also applies to PROC SUMMARY.

Syntax: SUMMARY Procedure

Tip: Supports the Output Delivery System. See “Output Delivery System” on page 32 for details.

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, “Statements with the Same Function in Multiple Procedures,” on page 53 for details. You can also use any global statements as well. See “Global Statements” on page 18 for a list.

Reminder: Full syntax descriptions are in “Syntax: MEANS Procedure” on page 652.

```

PROC SUMMARY <option(s)> <statistic-keyword(s)>;
  BY <DESCENDING> variable-1<...<DESCENDING> variable-n>
    <NOTSORTED>;
  CLASS variable(s) </ option(s)>;
  FREQ variable;
  ID variable(s);
  OUTPUT <OUT=SAS-data-set><output-statistic-specification(s)>
    <id-group-specification(s)> <maximum-id-specification(s)>
    <minimum-id-specification(s)></ option(s)> ;
  TYPES request(s);
  VAR variable(s)</ WEIGHT=weight-variable>;

```

WAYS *list*;
WEIGHT *variable*;

PROC SUMMARY Statement

PRINT | NOPRINT

specifies whether PROC SUMMARY displays the descriptive statistics. By default, PROC SUMMARY produces no display output, but PROC MEANS does produce display output.

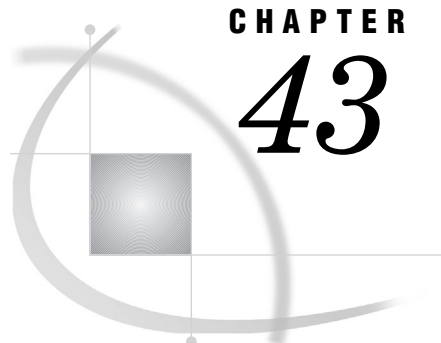
Default: NOPRINT

VAR Statement

Identifies the analysis variables and their order in the results.

Default: If you omit the VAR statement, then PROC SUMMARY produces a simple count of observations, whereas PROC MEANS tries to analyze all the numeric variables that are not listed in the other statements.

Interaction: If you specify statistics on the PROC SUMMARY statement and the VAR statement is omitted, then PROC SUMMARY stops processing and an error message is written to the SAS log.



CHAPTER

43

The TABULATE Procedure

<i>Overview: TABULATE Procedure</i>	1260
<i>Terminology Used with PROC TABULATE</i>	1263
<i>Syntax: TABULATE Procedure</i>	1266
<i>PROC TABULATE Statement</i>	1267
<i>BY Statement</i>	1275
<i>CLASS Statement</i>	1276
<i>CLASSLEV Statement</i>	1279
<i>FREQ Statement</i>	1280
<i>KEYLABEL Statement</i>	1281
<i>KEYWORD Statement</i>	1281
<i>TABLE Statement</i>	1282
<i>VAR Statement</i>	1289
<i>WEIGHT Statement</i>	1291
<i>Concepts: TABULATE Procedure</i>	1291
<i>Statistics That Are Available in PROC TABULATE</i>	1291
<i>Formatting Class Variables</i>	1292
<i>Formatting Values in Tables</i>	1293
<i>How Using BY-Group Processing Differs from Using the Page Dimension</i>	1294
<i>Calculating Percentages</i>	1294
<i>Specifying a Denominator for the PCTN Statistic</i>	1295
<i>Specifying a Denominator for the PCTSUM Statistic</i>	1296
<i>Using Style Elements in PROC TABULATE</i>	1298
<i>Results: TABULATE Procedure</i>	1299
<i>Missing Values</i>	1299
<i>No Missing Values</i>	1301
<i>A Missing Class Variable</i>	1302
<i>Including Observations with Missing Class Variables</i>	1303
<i>Formatting Headings for Observations with Missing Class Variables</i>	1304
<i>Providing Headings for All Categories</i>	1305
<i>Providing Text for Cells That Contain Missing Values</i>	1306
<i>Providing Headings for All Values of a Format</i>	1307
<i>Understanding the Order of Headings with ORDER=DATA</i>	1308
<i>Examples: TABULATE Procedure</i>	1310
<i>Example 1: Creating a Basic Two-Dimensional Table</i>	1310
<i>Example 2: Specifying Class Variable Combinations to Appear in a Table</i>	1312
<i>Example 3: Using Preloaded Formats with Class Variables</i>	1314
<i>Example 4: Using Multilabel Formats</i>	1320
<i>Example 5: Customizing Row and Column Headings</i>	1322
<i>Example 6: Summarizing Information with the Universal Class Variable ALL</i>	1324
<i>Example 7: Eliminating Row Headings</i>	1326
<i>Example 8: Indenting Row Headings and Eliminating Horizontal Separators</i>	1328

<i>Example 9: Creating Multipage Tables</i>	1330
<i>Example 10: Reporting on Multiple-Response Survey Data</i>	1333
<i>Example 11: Reporting on Multiple-Choice Survey Data</i>	1338
<i>Example 12: Calculating Various Percentage Statistics</i>	1344
<i>Example 13: Using Denominator Definitions to Display Basic Frequency Counts and Percentages</i>	1347
<i>Example 14: Specifying Style Elements for ODS Output</i>	1357
<i>References</i>	1361

Overview: TABULATE Procedure

The TABULATE procedure displays descriptive statistics in tabular format, using some or all of the variables in a data set. You can create a variety of tables ranging from simple to highly customized.

PROC TABULATE computes many of the same statistics that are computed by other descriptive statistical procedures such as MEANS, FREQ, and REPORT. PROC TABULATE provides

- simple but powerful methods to create tabular reports
- flexibility in classifying the values of variables and establishing hierarchical relationships between the variables
- mechanisms for labeling and formatting variables and procedure-generated statistics.

Output 43.1 on page 1261 shows a simple table that was produced by PROC TABULATE. The data set on page 1310 contains data on expenditures of energy by two types of customers, residential and business, in individual states in the Northeast (1) and West (4) regions of the United States. The table sums expenditures for states within a geographic division. (The RTS option provides enough space to display the column headers without hyphenating them.)

```
options nodate pageno=1 linesize=64
      pagesize=40;

proc tabulate data=energy;
  class region division type;
  var expenditures;
  table region*division, type*expenditures /
      rts=20;
run;
```

Output 43.1 Simple Table Produced by PROC TABULATE

The SAS System				1
		Type		
		1	2	
		Expenditures	Expenditures	
		Sum	Sum	
Region	Division			
1	1	7477.00	5129.00	
	2	19379.00	15078.00	
4	3	5476.00	4729.00	
	4	13959.00	12619.00	

Output 43.2 on page 1262 is a more complicated table using the same data set that was used to create Output 43.1 on page 1261. The statements that create this report

- customize column and row headers
- apply a format to all table cells
- sum expenditures for residential and business customers
- compute subtotals for each division
- compute totals for all regions.

For an explanation of the program that produces this report, see Example 6 on page 1324.

Output 43.2 Complex Table Produced by PROC TABULATE

Energy Expenditures for Each Region (millions of dollars)					2
		Customer Base			
		Residential Customers	Business Customers	All Customers	
Region	Division				
Northeast	New England	7,477	5,129	12,606	
	Middle Atlantic	19,379	15,078	34,457	
	Subtotal	26,856	20,207	47,063	
West	Division				
	Mountain	5,476	4,729	10,205	
	Pacific	13,959	12,619	26,578	
	Subtotal	19,435	17,348	36,783	
Total for All Regions		\$46,291	\$37,555	\$83,846	

Display 43.1 on page 1263 shows a table that is created in Hypertext Markup Language (HTML). You can use the Output Delivery System with PROC TABULATE to create customized output in HTML, Rich Text Format (RTF), Portable Document Format (PDF), and other output formats. For an explanation of the program that produces this table, see Example 14 on page 1357.

Display 43.1 HTML Table Produced by PROC TABULATE

<i>Energy Expenditures (millions of dollars)</i>				
<i>Region by Division by Type</i>		Type		Total
		Residential Customers	Business Customers	
		Expenditures	Expenditures	Expenditures
		Sum	Sum	Sum
Region	Division			
Northeast	New England	\$7,477	\$5,129	\$12,606
	Middle Atlantic	\$19,379	\$15,078	\$34,457
	Total	\$26,856	\$20,207	\$47,063
West	Division			
	Mountain	\$5,476	\$4,729	\$10,205
	Pacific	\$13,959	\$12,619	\$26,578
	Total	\$19,435	\$17,348	\$36,783
Total	Division			
	New England	\$7,477	\$5,129	\$12,606
	Middle Atlantic	\$19,379	\$15,078	\$34,457
	Mountain	\$5,476	\$4,729	\$10,205
	Pacific	\$13,959	\$12,619	\$26,578
	Total	\$46,291	\$37,555	\$83,846

Terminology Used with PROC TABULATE

The following figure illustrates some of the terms that are commonly used in discussions of PROC TABULATE.

Figure 43.1 Illustration of Terms That Are Used to Discuss PROC TABULATE

Region	Division	Type
Northeast	New England	Residential Customers
		Business Customers
	Middle Atlantic	
West	Mountain	
	Pacific	

In addition, the following terms frequently appear in discussions of PROC TABULATE:

category

the combination of unique values of class variables. The TABULATE procedure creates a separate category for each unique combination of values that exists in the observations of the data set. Each category that is created by PROC TABULATE is represented by one or more cells in the table where the pages, rows, and columns that describe the category intersect.

The table in Figure 43.1 on page 1264 contains three class variables: Region, Division, and Type. These class variables form the eight categories listed in Table 43.1 on page 1264. (For convenience, the categories are described in terms of their formatted values.)

Table 43.1 Categories Created from Three Class Variables

Region	Division	Type
Northeast	New England	Residential Customers
Northeast	New England	Business Customers
Northeast	Middle Atlantic	Residential Customers
Northeast	Middle Atlantic	Business Customers
West	Mountain	Residential Customers
West	Mountain	Business Customers

Region	Division	Type
West	Pacific	Residential Customers
West	Pacific	Business Customers

column dimension

the combination of variables, variable values, and statistics that determine the number and arrangement of columns in the table.

continuation message

the text that appears below the table if it spans multiple physical pages.

dimension

the page, row, or column portion of a table. Each dimension is defined by a *dimension expression*.

dimension expression

a part of the TABLE statement that defines the content and appearance of the rows, columns, or pages of the table.

nested variable

a variable whose values appear in the table with each value of another variable.

In Figure 43.1 on page 1264, Division is nested under Region.

page dimension

the combination of variables, variable values, and statistics that determine the number and arrangement of the pages in the table.

page dimension text

the text that appears above the table if the table has a page dimension. However, if you specify BOX=_PAGE_ in the TABLE statement, then the text that would appear above the table appears in the box.

Page dimension text has a style. The default style is *Beforecaption*. For more information about using styles, see STYLE= on page 1273 in the PROC TABULATE statement and “Output Delivery System” on page 32.

row dimension

the combination of variables, variable values, and statistics that determine the number and arrangement of rows in the table.

subtable

the group of cells that is produced by crossing a single element from each dimension of the TABLE statement when one or more dimensions contain concatenated elements.

Figure 43.1 on page 1264 contains no subtables. For an illustration of a table that is composed of multiple subtables, see Figure 43.17 on page 1352.

Syntax: TABULATE Procedure

Requirements: At least one TABLE statement is required.

Requirements: Depending on the variables that appear in the TABLE statement, a CLASS statement, a VAR statement, or both are required.

Tip: Supports the Output Delivery System. See “Output Delivery System” on page 32 for details.

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, “Statements with the Same Function in Multiple Procedures,” on page 53 for details. You can also use any global statements as well. See “Global Statements” on page 18 for a list.

```

PROC TABULATE <option(s)>;
  BY <DESCENDING> variable-1
      <...<DESCENDING> variable-n>
      <NOTSORTED>;
  CLASS variable(s) </ options>;
  CLASSLEV variable(s) / style =<style-element-name | <PARENT>>
      <[style-attribute-specification(s)]>;
  FREQ variable;
  KEYLABEL keyword-1='description-1'
      <...keyword-n='description-n'>;
  KEYWORD keyword(s) / style =<style-element-name | <PARENT>>
      <[style-attribute-specification(s)] >;
  TABLE <<page-expression,> row-expression,> column-expression</ table-option(s)>;
  VAR analysis-variable(s)</ options>;
  WEIGHT variable;

```

To do this	Use this statement
Create a separate table for each BY group	BY
Identify variables in the input data set as class variables	CLASS
Specify a style for class variable level value headings	CLASSLEV
Identify a variable in the input data set whose values represent the frequency of each observation	FREQ
Specify a label for a keyword	KEYLABEL
Specify a style for keyword headings	KEYWORD
Describe the table to create	TABLE
Identify variables in the input data set as analysis variables	VAR
Identify a variable in the input data set whose values weight each observation in the statistical calculations	WEIGHT

PROC TABULATE Statement

PROC TABULATE *<option(s)>*;

To do this	Use this option
Customize the HTML contents link to the output	CONTENTS=
Specify the input data set	DATA=
Specify the output data set	OUT=
Enable floating point exception recovery	TRAP
Identify categories of data that are of interest	
Specify a secondary data set that contains the combinations of values of class variables to include in tables and output data sets	CLASSDATA=
Exclude from tables and output data sets all combinations of class variable values that are not in the CLASSDATA= data set	EXCLUSIVE
Consider missing values as valid values for class variables	MISSING
Control the statistical analysis	
Specify the confidence level for the confidence limits	ALPHA=
Exclude observations with nonpositive weights	EXCLNPWGTS
Specify the sample size to use for the P^2 quantile estimation method	QMARKERS=
Specify the quantile estimation method	QMETHOD=
Specify the mathematical definition to calculate quantiles	QNTLDEF=
Specify the variance divisor	VARDEF=
Customize the appearance of the table	
Specify a default format for each cell in the table	FORMAT=
Define the characters to use to construct the table outlines and dividers	FORMCHAR=
Eliminate horizontal separator lines from the row titles and the body of the table	NOSEPS
Order the values of a class variable according to the specified order	ORDER=
Specify the default style element or style elements (for the Output Delivery System) to use for each cell of the table	STYLE=

Options

ALPHA=*value*

specifies the confidence level to compute the confidence limits for the mean. The percentage for the confidence limits is $(1 - \text{value}) \times 100$. For example, ALPHA=.05 results in a 95% confidence limit.

Default: .05

Range: between 0 and 1

Interaction: To compute confidence limits specify the *statistic-keyword* CLM, LCLM, or UCLM.

CLASSDATA=SAS-*data-set*

specifies a data set that contains the combinations of values of the class variables that must be present in the output. Any combinations of values of the class variables that occur in the CLASSDATA= data set but not in the input data set appear in each table or output data set and have a frequency of zero.

Restriction: The CLASSDATA= data set must contain all class variables. Their data type and format must match the corresponding class variables in the input data set.

Interaction: If you use the EXCLUSIVE option, then PROC TABULATE excludes any observations in the input data set whose combinations of values of class variables are not in the CLASSDATA= data set.

Tip: Use the CLASSDATA= data set to filter or supplement the input data set.

Featured in: Example 2 on page 1312

CONTENTS=*link-name*

enables you to name the link in the HTML table of contents that points to the ODS output of the first table that was produced by using the TABULATE procedure.

Note: CONTENTS= affects only the contents file of ODS HTML output. It has no effect on the actual TABULATE procedure reports. \triangle

DATA=SAS-*data-set*

specifies the input data set.

Main Discussion: “Input Data Sets” on page 19

EXCLNPWGTS

excludes observations with nonpositive weight values (zero or negative) from the analysis. By default, PROC TABULATE treats observations with negative weights like those with zero weights and counts them in the total number of observations.

Alias: EXCLNPWGT

See also: WEIGHT= on page 1290 and “WEIGHT Statement” on page 1291

EXCLUSIVE

excludes from the tables and the output data sets all combinations of the class variable that are not found in the CLASSDATA= data set.

Requirement: If a CLASSDATA= data set is not specified, then this option is ignored.

Featured in: Example 2 on page 1312

FORMAT=*format-name*

specifies a default format for the value in each table cell. You can use any SAS or user-defined format.

Alias: F=

Default: If you omit `FORMAT=`, then PROC TABULATE uses BEST12.2 as the default format.

Interaction: Formats that are specified in a TABLE statement override the format that is specified with `FORMAT=`.

Tip: This option is especially useful for controlling the number of print positions that are used to print a table.

Featured in: Example 1 on page 1310 and Example 6 on page 1324

FORMCHAR *<(position(s))>='formatting-character(s)'*

defines the characters to use for constructing the table outlines and dividers.

position(s)

identifies the position of one or more characters in the SAS formatting-character string. A space or a comma separates the positions.

Default: Omitting *position(s)* is the same as specifying all 20 possible SAS formatting characters, in order.

Range: PROC TABULATE uses 11 of the 20 formatting characters that SAS provides. Table 43.2 on page 1270 shows the formatting characters that PROC TABULATE uses. Figure 43.2 on page 1270 illustrates the use of each formatting character in the output from PROC TABULATE.

formatting-character(s)

lists the characters to use for the specified positions. PROC TABULATE assigns characters in *formatting-character(s)* to *position(s)*, in the order that they are listed. For example, the following option assigns the asterisk (*) to the third formatting character, the pound sign (#) to the seventh character, and does not alter the remaining characters:

```
formchar(3,7)='*#'
```

Interaction: The SAS system option `FORMCHAR=` specifies the default formatting characters. The system option defines the entire string of formatting characters. The `FORMCHAR=` option in a procedure can redefine selected characters.

Restriction: The `FORMCHAR=` option affects only the traditional SAS monospace output destination.

Tip: You can use any character in *formatting-characters*, including hexadecimal characters. If you use hexadecimal characters, then you must put an **x** after the closing quotation mark. For instance, the following option assigns the hexadecimal character 2D to the third formatting character, assigns the hexadecimal character 7C to the seventh character, and does not alter the remaining characters:

```
formchar(3,7)='2D7C'x
```

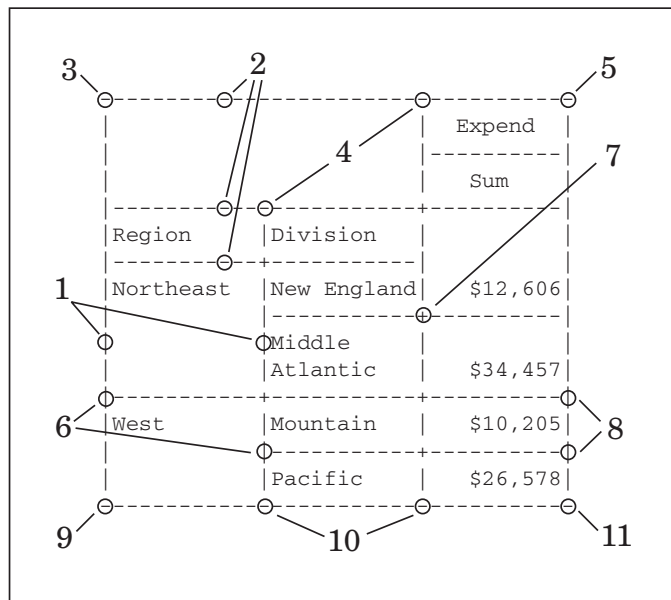
Tip: Specifying all blanks for *formatting-character(s)* produces tables with no outlines or dividers.

```
formchar(1,2,3,4,5,6,7,8,9,10,11)
      = '                ' (11 blanks)
```

See also: For more information about formatting output, see Chapter 5, “Controlling the Table’s Appearance,” in the *SAS Guide to TABULATE Processing*. For information about which hexadecimal codes to use for which characters, consult the documentation for your hardware.

Table 43.2 Formatting Characters Used by PROC TABULATE

Position	Default	Used to draw
1		the right and left borders and the vertical separators between columns
2	-	the top and bottom borders and the horizontal separators between rows
3	-	the top character in the left border
4	-	the top character in a line of characters that separate columns
5	-	the top character in the right border
6		the leftmost character in a row of horizontal separators
7	+	the intersection of a column of vertical characters and a row of horizontal characters
8		the rightmost character in a row of horizontal separators
9	-	the bottom character in the left border
10	-	the bottom character in a line of characters that separate columns
11	-	the bottom character in the right border

Figure 43.2 Formatting Characters in PROC TABULATE Output**MISSING**

considers missing values as valid values to create the combinations of class variables. Special missing values that are used to represent numeric values (the letters A through Z and the underscore (_) character) are each considered as a separate value. A heading for each missing value appears in the table.

Default: If you omit MISSING, then PROC TABULATE does not include observations with a missing value for any class variable in the report.

Main Discussion: “Including Observations with Missing Class Variables” on page 1303

See also: *SAS Language Reference: Concepts* for a discussion of missing values that have special meaning.

NOSEPS

eliminates horizontal separator lines from the row titles and the body of the table. Horizontal separator lines remain between nested column headers.

Restriction: The NOSEPS option affects only the traditional SAS monospace output destination.

Tip: If you want to replace the separator lines with blanks rather than remove them, then use the FORMCHAR= option on page 1269.

Featured in: Example 8 on page 1328

NOTRAP

See TRAP | NOTRAP on page 1274.

ORDER=DATA | FORMATTED | FREQ | UNFORMATTED

specifies the sort order to create the unique combinations of the values of the class variables, which form the headings of the table, according to the specified order.

DATA

orders values according to their order in the input data set.

Interaction: If you use PRELOADFMT in the CLASS statement, then the order for the values of each class variable matches the order that PROC FORMAT uses to store the values of the associated user-defined format. If you use the CLASSDATA= option, then PROC TABULATE uses the order of the unique values of each class variable in the CLASSDATA= data set to order the output levels. If you use both options, then PROC TABULATE first uses the user-defined formats to order the output. If you omit EXCLUSIVE, then PROC TABULATE appends after the user-defined format and the CLASSDATA= values the unique values of the class variables in the input data set in the same order in which they are encountered.

Tip: By default, PROC FORMAT stores a format definition in sorted order. Use the NOTSORTED option to store the values or ranges of a user defined format in the order that you define them.

FORMATTED

orders values by their ascending formatted values. If no format has been assigned to a numeric class variable, then the default format, BEST12., is used. This order depends on your operating environment.

Alias: FMT | EXTERNAL

FREQ

orders values by descending frequency count.

Interaction: Use the ASCENDING option in the CLASS statement to order values by ascending frequency count.

UNFORMATTED

orders values by their unformatted values, which yields the same order as PROC SORT. This order depends on your operating environment. This sort sequence is particularly useful for displaying dates chronologically.

Alias: UNFMT | INTERNAL

Default: UNFORMATTED

Interaction: If you use the PRELOADFMT option in the CLASS statement, then PROC TABULATE orders the levels by the order of the values in the user-defined format.

Featured in: “Understanding the Order of Headings with ORDER=DATA” on page 1308

OUT=SAS-*data-set*

names the output data set. If *SAS-data-set* does not exist, then PROC TABULATE creates it.

The number of observations in the output data set depends on the number of categories of data that are used in the tables and the number of subtables that are generated. The output data set contains these variables (in this order):

by variables

variables listed in the BY statement.

class variables

variables listed in the CLASS statement.

TYPE

a character variable that shows which combination of class variables produced the summary statistics in that observation. Each position in _TYPE_ represents one variable in the CLASS statement. If that variable is in the category that produced the statistic, then the position contains a 1; if it is not, then the position contains a 0. In simple PROC TABULATE steps that do not use the universal class variable ALL, all values of _TYPE_ contain only 1's because the only categories that are being considered involve all class variables. If you use the variable ALL, then your tables will contain data for categories that do not include all the class variables, and positions of _TYPE_ will, therefore, include both 1's and 0's.

PAGE

The logical page that contains the observation.

TABLE

The number of the table that contains the observation.

statistics

statistics that are calculated for each observation in the data set.

Featured in: Example 3 on page 1314

QMARKERS=*number*

specifies the default number of markers to use for the P^2 quantile estimation method. The number of markers controls the size of fixed memory space.

Default: The default value depends on which quantiles you request. For the median (P50), *number* is 7. For the quartiles (P25 and P75), *number* is 25. For the quantiles P1, P5, P10, P90, P95, or P99, *number* is 105. If you request several quantiles, then PROC TABULATE uses the largest default value of *number*.

Range: an odd integer greater than 3

Tip: Increase the number of markers above the default settings to improve the accuracy of the estimates; reduce the number of markers to conserve memory and computing time.

Main Discussion: “Quantiles” on page 680

QMETHOD=OS|P2|HIST

specifies the method PROC TABULATE uses to process the input data when it computes quantiles. If the number of observations is less than or equal to the QMARKERS= value and QNTLDEF=5, then both methods produce the same results.

OS

uses order statistics. This is the technique that PROC UNIVARIATE uses.

Note: This technique can be very memory-intensive. Δ

P2|HIST

uses the P^2 method to approximate the quantile.

Default: OS

Restriction: When QMETHOD=P2, PROC TABULATE does not compute weighted quantiles.

Tip: When QMETHOD=P2, reliable estimates of some quantiles (P1, P5, P95, P99) may not be possible for some types of data.

Main Discussion: “Quantiles” on page 680

QNTLDEF=1|2|3|4|5

specifies the mathematical definition that the procedure uses to calculate quantiles when QMETHOD=OS is specified. When QMETHOD=P2, you must use QNTLDEF=5.

Default: 5

Alias: PCTLDEF=

Main discussion: “Percentile and Related Statistics” on page 1583

STYLE=<style-element-name | <PARENT>><[style-attribute-specification(s)]>

specifies the style element to use for the data cells of a table when it is used in the PROC TABULATE statement. For example, the following statement specifies that the background color for data cells be red:

```
proc tabulate data=one style=[background=red];
```

Note: This option can be used in other statements, or in dimension expressions, to specify style elements for other parts of a table. Δ

Note: You can use braces ({ and }) instead of square brackets ([and]). Δ

style-element-name

is the name of a style element that is part of a style definition that is registered with the Output Delivery System. SAS provides some style definitions. You can create your own style definitions with PROC TEMPLATE.

Default: If you do not specify a style element, then PROC TABULATE uses Data.

See also: See *SAS Output Delivery System User's Guide* for information about PROC TEMPLATE and the default style definitions.

PARENT

specifies that the data cell use the style element of its parent heading. The parent style element of a data cell is one of the following:

- ☐ the style element of the leaf heading above the column that contains the data cell, if the table specifies no row dimension, or if the table specifies the style element in the column dimension expression.
- ☐ the style element of the leaf heading above the row that contains the cell, if the table specifies the style element in the row dimension expression.
- ☐ the Beforecaption style element, if the table specifies the style element in the page dimension expression.
- ☐ undefined, otherwise.

Note: The parent of a heading (not applicable to STYLE= in the PROC TABULATE statement) is the heading under which the current heading is nested. Δ

style-attribute-specification(s)

describes the attribute to change. Each *style-attribute-specification* has this general form:

style-attribute-name=style-attribute-value

The following table shows attributes that you can set or change with the STYLE= option in the PROC TABULATE statement (or in any other statement that uses STYLE=, except for the TABLE statement). Note that not all attributes are valid in all destinations. See *SAS Output Delivery System User's Guide* for more information about these style attributes, their valid values, and their applicable destinations.

ASIS=	FONT_WIDTH=
BACKGROUND=	HREFTARGET=
BACKGROUNDIMAGE=	HTMLCLASS=
BORDERCOLOR=	JUST=
BORDERCOLORDARK=	NOBREAKSPACE=
BORDERCOLORLIGHT=	POSTHTML=
BORDERWIDTH=	POSTIMAGE=
CELLHEIGHT=	POSTTEXT=
CELLWIDTH=	PREHTML=
FLYOVER=	PREIMAGE=
FONT=	PRETEXT=
FONT_FACE=	PROTECTSPECIALCHARS=
FONT_SIZE=	TAGATTR=
FONT_STYLE=	URL=
FONT_WEIGHT=	VJUST=

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: To specify a style element for data cells with missing values, use STYLE= in the TABLE statement MISSTEXT= option.

See also: “Using Style Elements in PROC TABULATE” on page 1298

Featured in: Example 14 on page 1357

TRAP | NOTRAP

enables or disables floating point exception (FPE) recovery during data processing beyond that provided by normal SAS FPE handling, which terminates PROC TABULATE in the case of math exceptions. Note that with NOTRAP, normal SAS FPE handling is still in effect so that PROC TABULATE terminates in the case of math exceptions.

Default: NOTRAP

VARDEF=*divisor*

specifies the divisor to use in the calculation of the variance and standard deviation. Table 43.3 on page 1275 shows the possible values for *divisor* and the associated divisors.

Table 43.3 Possible Values for VARDEF=

Value	Divisor	Formula for Divisor
DF	degrees of freedom	$n - 1$
N	number of observations	n
WDF	sum of weights minus one	$(\sum_i w_i) - 1$
WEIGHT WGT	sum of weights	$\sum_i w_i$

The procedure computes the variance as $CSS/divisor$, where CSS is the corrected sums of squares and equals $\sum (x_i - \bar{x})^2$. When you weight the analysis variables, CSS equals $\sum w_i (x_i - \bar{x}_w)^2$ where \bar{x}_w is the weighted mean.

Default: DF

Requirement: To compute standard error of the mean, use the default value of VARDEF=.

Tip: When you use the WEIGHT statement and VARDEF=DF, the variance is an estimate of σ^2 , where the variance of the i th observation is $var(x_i) = \sigma^2/w_i$, and w_i is the weight for the i th observation. This yields an estimate of the variance of an observation with unit weight.

Tip: When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large n) an estimate of σ^2/\bar{w} , where \bar{w} is the average weight. This yields an asymptotic estimate of the variance of an observation with average weight.

See also: “Weighted Statistics Example” on page 60

BY Statement

Creates a separate table on a separate page for each BY group.

Main discussion: “BY” on page 54

```
BY <DESCENDING> variable-1
   <...<DESCENDING> variable-n>
   <NOTSORTED>;
```

Required Arguments

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, then the observations in the data set must either be sorted by all the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Options

DESCENDING

specifies that the observations are sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The observations are grouped in another way, for example, chronological order.

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, then the procedure treats each contiguous set as a separate BY group.

CLASS Statement

Identifies class variables for the table. Class variables determine the categories that PROC TABULATE uses to calculate statistics.

Tip: You can use multiple CLASS statements.

Tip: Some CLASS statement options are also available in the PROC TABULATE statement. They affect all CLASS variables rather than just the one(s) that you specify in a CLASS statement.

CLASS *variable(s)* *</option(s)>*;

Required Arguments***variable(s)***

specifies one or more variables that the procedure uses to group the data. Variables in a CLASS statement are referred to as *class variables*. Class variables can be numeric or character. Class variables can have continuous values, but they typically have a few discrete values that define the classifications of the variable. You do not have to sort the data by class variables.

Options**ASCENDING**

specifies to sort the class variable values in ascending order.

Alias: ASCEND

Interaction: PROC TABULATE issues a warning message if you specify both ASCENDING and DESCENDING and ignores both options.

DESCENDING

specifies to sort the class variable values in descending order.

Alias: DESCEND

Default: ASCENDING

Interaction: PROC TABULATE issues a warning message if you specify both ASCENDING and DESCENDING and ignores both options.

EXCLUSIVE

excludes from tables and output data sets all combinations of class variables that are not found in the preloaded range of user-defined formats.

Requirement: You must specify the PRELOADFMT option in the CLASS statement to preload the class variable formats.

Featured in: Example 3 on page 1314

GROUPINTERNAL

specifies not to apply formats to the class variables when PROC TABULATE groups the values to create combinations of class variables.

Interaction: If you specify the PRELOADFMT option in the CLASS statement, then PROC TABULATE ignores the GROUPINTERNAL option and uses the formatted values.

Interaction: If you specify the ORDER=FORMATTED option, then PROC TABULATE ignores the GROUPINTERNAL option and uses the formatted values.

Tip: This option saves computer resources when the class variables contain discrete numeric values.

MISSING

considers missing values as valid class variable levels. Special missing values that represent numeric values (the letters A through Z and the underscore (_) character) are each considered as a separate value.

Default: If you omit MISSING, then PROC TABULATE excludes the observations with any missing CLASS variable values from tables and output data sets.

See also: *SAS Language Reference: Concepts* for a discussion of missing values with special meanings.

MLF

enables PROC TABULATE to use the format label or labels for a given range or overlapping ranges to create subgroup combinations when a multilabel format is assigned to a class variable.

Requirement: You must use PROC FORMAT and the MULTILABEL option in the VALUE statement to create a multilabel format.

Interaction: Using MLF with ORDER=FREQ may not produce the order that you expect for the formatted values.

Interaction: When you specify MLF, the formatted values of the class variable become internal values. Therefore, specifying ORDER=FORMATTED produces the same results as specifying ORDER=UNFORMATTED.

Tip: If you omit MLF, then PROC TABULATE uses the primary format labels, which correspond to the first external format value, to determine the subgroup combinations.

See also: The MULTILABEL option on page 460 in the VALUE statement of the FORMAT procedure.

Featured in: Example 4 on page 1320

Note: When the formatted values overlap, one internal class variable value maps to more than one class variable subgroup combination. Therefore, the sum of the N statistics for all subgroups is greater than the number of observations in the data set (the overall N statistic). Δ

ORDER=DATA | FORMATTED | FREQ | UNFORMATTED

specifies the order to group the levels of the class variables in the output, where

DATA

orders values according to their order in the input data set.

Interaction: If you use PRELOADFMT, then the order for the values of each class variable matches the order that PROC FORMAT uses to store the values of the associated user-defined format. If you use the CLASSDATA= option in the PROC statement, then PROC TABULATE uses the order of the unique values of each class variable in the CLASSDATA= data set to order the output levels. If you use both options, then PROC TABULATE first uses the user-defined formats to order the output. If you omit EXCLUSIVE in the PROC statement, then PROC TABULATE places, in the order in which they are encountered, the unique values of the class variables that are in the input data set after the user-defined format and the CLASSDATA= values.

Tip: By default, PROC FORMAT stores a format definition in sorted order. Use the NOTSORTED option to store the values or ranges of a user-defined format in the order that you define them.

FORMATTED

orders values by their ascending formatted values. This order depends on your operating environment.

Alias: FMT | EXTERNAL

FREQ

orders values by descending frequency count.

Interaction: Use the ASCENDING option to order values by ascending frequency count.

UNFORMATTED

orders values by their unformatted values, which yields the same order as PROC SORT. This order depends on your operating environment. This sort sequence is particularly useful for displaying dates chronologically.

Alias: UNFMT | INTERNAL

Default: UNFORMATTED

Interaction: If you use the PRELOADFMT option in the CLASS statement, then PROC TABULATE orders the levels by the order of the values in the user-defined format.

Tip: By default, all orders except FREQ are ascending. For descending orders, use the DESCENDING option.

Featured in: “Understanding the Order of Headings with ORDER=DATA” on page 1308

PRELOADFMT

specifies that all formats are preloaded for the class variables.

Requirement: PRELOADFMT has no effect unless you specify EXCLUSIVE, ORDER=DATA, or PRINTMISS and you assign formats to the class variables.

Note: If you specify PRELOADFMT without also specifying EXCLUSIVE, ORDER=DATA, or PRINTMISS, then SAS writes a warning message to the SAS log. \triangle

Interaction: To limit PROC TABULATE output to the combinations of formatted class variable values present in the input data set, use the EXCLUSIVE option in the CLASS statement.

Interaction: To include all ranges and values of the user-defined formats in the output, use the PRINTMISS option in the TABLE statement.

Note: Use care when you use PRELOADFMT with PRINTMISS. This feature creates all possible combinations of formatted class variables. Some of these combinations may not make sense. Δ

Featured in: Example 3 on page 1314

STYLE=<*style-element-name* | <PARENT>><[*style-attribute-specification(s)*]>
specifies the style element to use for page dimension text and class variable name headings. For information about the arguments of this option, and how it is used, see STYLE= on page 1273 in the PROC TABULATE statement.

Note: When you use STYLE= in the CLASS statement, it differs slightly from its use in the PROC TABULATE statement. In the CLASS statement, the parent of the heading is the page dimension text or heading under which the current heading is nested. Δ

Note: If a page dimension expression contains multiple nested elements, then the Beforecaption style element is the style element of the first element in the nesting. Δ

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: To override a style element that is specified for page dimension text in the CLASS statement, you can specify a style element in the TABLE statement page dimension expression.

Tip: To override a style element that is specified for a class variable name heading in the CLASS statement, you can specify a style element in the related TABLE statement dimension expression.

Featured in: Example 14 on page 1357

How PROC TABULATE Handles Missing Values for Class Variables

By default, if an observation contains a missing value for any class variable, then PROC TABULATE excludes that observation from all tables that it creates. CLASS statements apply to all TABLE statements in the PROC TABULATE step. Therefore, if you define a variable as a class variable, then PROC TABULATE omits observations that have missing values for that variable from every table even if the variable does not appear in the TABLE statement for one or more tables.

If you specify the MISSING option in the PROC TABULATE statement, then the procedure considers missing values as valid levels for all class variables. If you specify the MISSING option in a CLASS statement, then PROC TABULATE considers missing values as valid levels for the class variable(s) that are specified in that CLASS statement.

CLASSLEV Statement

Specifies a style element for class variable level value headings.

Restriction: This statement affects only the HTML, RTF, and Printer destinations.

CLASSLEV *variable(s)* / style =<*style-element-name* | <PARENT>>
 <[*style-attribute-specification(s)*] >;

Required Arguments

variable(s)

specifies one or more class variables from the CLASS statement for which you want to specify a style element.

Options

STYLE=<*style-element-name* | <PARENT>><[*style-attribute-specification(s)*]>

specifies a style element for class variable level value headings. For information on the arguments of this option and how it is used, see STYLE= on page 1273 in the PROC TABULATE statement.

Note: When you use STYLE= in the CLASSLEV statement, it differs slightly from its use in the PROC TABULATE statement. In the CLASSLEV statement, the parent of the heading is the heading under which the current heading is nested. \triangle

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: To override a style element that is specified in the CLASSLEV statement, you can specify a style element in the related TABLE statement dimension expression.

Featured in: Example 14 on page 1357

FREQ Statement

Specifies a numeric variable that contains the frequency of each observation.

Tip: The effects of the FREQ and WEIGHT statements are similar except when calculating degrees of freedom.

See also: For an example that uses the FREQ statement, see “FREQ” on page 56.

FREQ *variable*;

Required Arguments

variable

specifies a numeric variable whose value represents the frequency of the observation. If you use the FREQ statement, then the procedure assumes that each observation represents n observations, where n is the value of *variable*. If n is not an integer, then SAS truncates it. If n is less than 1 or is missing, then the procedure does not use that observation to calculate statistics.

The sum of the frequency variable represents the total number of observations.

KEYLABEL Statement

Labels a keyword for the duration of the PROC TABULATE step. PROC TABULATE uses the label anywhere that the specified keyword would otherwise appear.

```
KEYLABEL keyword-1='description-1'  
      <...keyword-n='description-n'>;
```

Required Arguments

keyword

is one of the keywords for statistics that is discussed in “Statistics That Are Available in PROC TABULATE” on page 1291 or is the universal class variable ALL (see “Elements That You Can Use in a Dimension Expression” on page 1287).

description

is up to 256 characters to use as a label. As the syntax shows, you must enclose *description* in quotation marks.

Restriction: Each keyword can have only one label in a particular PROC TABULATE step; if you request multiple labels for the same keyword, then PROC TABULATE uses the last one that is specified in the step.

KEYWORD Statement

Specifies a style element for keyword headings.

Restriction: This statement affects only the HTML, RTF, and Printer output.

```
KEYWORD keyword(s) / style =<style-element-name | <PARENT>>  
      <[style-attribute-specification(s)] >;
```

Required Arguments

keyword

is one of the keywords for statistics that is discussed in “Statistics That Are Available in PROC TABULATE” on page 1291 or is the universal class variable ALL (see “Elements That You Can Use in a Dimension Expression” on page 1287).

Options

```
STYLE=<style-element-name | <PARENT>><[style-attribute-specification(s)]>
```

specifies a style element for the keyword headings. For information on the arguments of this option and how it is used, see STYLE= on page 1273 in the PROC TABULATE statement.

Note: When you use STYLE= in the KEYWORD statement, it differs slightly from its use in the PROC TABULATE statement. In the KEYWORD statement, the parent of the heading is the heading under which the current heading is nested. △

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: To override a style element that is specified in the KEYWORD statement, you can specify a style element in the related TABLE statement dimension expression.

Featured in: Example 14 on page 1357

TABLE Statement

Describes a table to print.

Requirement: All variables in the TABLE statement must appear in either the VAR statement or the CLASS statement.

Tip: Use multiple TABLE statements to create several tables.

TABLE <<*page-expression*,> *row-expression*,>
 column-expression </ *table-option(s)*>;

Required Arguments

column-expression

defines the columns in the table. For information on constructing dimension expressions, see “Constructing Dimension Expressions” on page 1286.

Restriction: A column dimension is the last dimension in a TABLE statement. A row dimension or a row dimension and a page dimension may precede a column dimension.

Options

To do this	Use this option
Add dimensions	
Define the pages in a table	<i>page-expression</i>
Define the rows in a table	<i>row-expression</i>
Customize the HTML contents entry link to the output	CONTENTS=
Specify a style element for various parts of the table	STYLE=
Customize text in the table	
Specify the text to place in the empty box above row titles	BOX=
Supply up to 256 characters to print in table cells that contain missing values	MISSTEXT=

To do this	Use this option
Suppress the continuation message for tables that span multiple physical pages	NOCONTINUED
Modify the layout of the table	
Print as many complete logical pages as possible on a single printed page or, if possible, print multiple pages of tables that are too wide to fit on a page one below the other on a single page, instead of on separate pages.	CONDENSE
Create the same row and column headings for all logical pages of the table	PRINTMISS
Customize row headings	
Specify the number of spaces to indent nested row headings	INDENT=
Control allocation of space for row titles within the available space	ROW=
Specify the number of print positions available for row titles	RTSPACE=

BOX=*value*

BOX={<*label*=*value*>

<*style*=<*style-element-name*><[*style-attribute-specification(s)*]>> }

specifies text and a style element for the empty box above the row titles.

Value can be one of the following:

PAGE

writes the page-dimension text in the box. If the page-dimension text does not fit, then it is placed in its default position above the box, and the box remains empty.

'*string*'

writes the quoted string in the box. Any string that does not fit in the box is truncated.

variable

writes the name (or label, if the variable has one) of a variable in the box. Any name or label that does not fit in the box is truncated.

For details about the arguments of the STYLE= option and how it is used, see STYLE= on page 1273 in the PROC TABULATE statement.

Featured in: Example 9 on page 1330 and Example 14 on page 1357

CONDENSE

prints as many complete logical pages as possible on a single printed page or, if possible, prints multiple pages of tables that are too wide to fit on a page one below the other on a single page, instead of on separate pages. A *logical page* is all the rows and columns that fall within one of the following:

- ☐ a page-dimension category (with no BY-group processing)
- ☐ a BY group with no page dimension
- ☐ a page-dimension category within a single BY group.

Restrictions: CONDENSE has no effect on the pages that are generated by the BY statement. The first table for a BY group always begins on a new page.

Featured in: Example 9 on page 1330

CONTENTS=link-name

enables you to name the link in the HTML table of contents that points to the ODS output of the table that is produced by using the TABLE statement.

Note: CONTENTS= affects only the contents file of ODS HTML output. It has no effect on the actual TABULATE procedure reports. \triangle

FUZZ=number

supplies a numeric value against which analysis variable values and table cell values other than frequency counts are compared to eliminate trivial values (absolute values less than the FUZZ= value) from computation and printing. A number whose absolute value is less than the FUZZ= value is treated as zero in computations and printing. The default value is the smallest representable floating-point number on the computer that you are using.

INDENT=number-of-spaces

specifies the number of spaces to indent nested row headings, and suppresses the row headings for class variables.

Tip: When there are no crossings in the row dimension, there is nothing to indent, so the value of *number-of-spaces* has no effect. However, in such cases INDENT= still suppresses the row headings for class variables.

Restriction: In the HTML, RTF, and Printer destinations, the INDENT= option suppresses the row headings for class variables but does not indent nested row headings.

Featured in: Example 8 on page 1328 (with crossings) and Example 9 on page 1330 (without crossings)

page-expression

defines the pages in a table. For information on constructing dimension expressions, see “Constructing Dimension Expressions” on page 1286.

Restriction: A page dimension is the first dimension in a table statement. Both a row dimension and a column dimension must follow a page dimension.

Featured in: Example 9 on page 1330

MISSTEXT='text'**MISSTEXT={<label= 'text'**

><style=<style-element-name><[style-attribute-specification(s)]>> }

supplies up to 256 characters of text to print and specifies a style element for table cells that contain missing values. For details on the arguments of the STYLE= option and how it is used, see STYLE= on page 1273 in the PROC TABULATE statement.

Interaction: A style element that is specified in a dimension expression overrides a style element that is specified in the MISSTEXT= option for any given cell(s).

Featured in: “Providing Text for Cells That Contain Missing Values” on page 1306 and Example 14 on page 1357

NOCONTINUED

suppresses the continuation message, **continued**, that is displayed at the bottom of tables that span multiple pages. The text is rendered with the Aftercaption style element.

Note: Because HTML browsers do not break pages, NOCONTINUED has no effect on the HTML destination. \triangle

PRINTMISS

prints all values that occur for a class variable each time headings for that variable are printed, even if there are no data for some of the cells that these headings create. Consequently, PRINTMISS creates row and column headings that are the same for all logical pages of the table, within a single BY group.

Default: If you omit PRINTMISS, then PROC TABULATE suppresses a row or column for which there are no data, unless you use the CLASSDATA= option in the PROC TABULATE statement.

Restrictions: If an entire logical page contains only missing values, then that page does not print regardless of the PRINTMISS option.

See also: CLASSDATA= option on page 1268

Featured in: “Providing Headings for All Categories” on page 1305

ROW=*spacing*

specifies whether all title elements in a row crossing are allotted space even when they are blank. The possible values for *spacing* are as follows:

CONSTANT

allots space to all row titles even if the title has been blanked out (for example, N=' ').

Alias: CONST

FLOAT

divides the row title space equally among the nonblank row titles in the crossing.

Default: CONSTANT

Featured in: Example 7 on page 1326

row-expression

defines the rows in the table. For information on constructing dimension expressions, see “Constructing Dimension Expressions” on page 1286.

Restriction: A row dimension is the next to last dimension in a table statement. A column dimension must follow a row dimension. A page dimension may precede a row dimension.

RTSPACE=*number*

specifies the number of print positions to allot to all of the headings in the row dimension, including spaces that are used to print outlining characters for the row headings. PROC TABULATE divides this space equally among all levels of row headings.

Alias: RTS=

Default: one-fourth of the value of the SAS system option LINESIZE=

Restriction: The RTSPACE= option affects only the traditional SAS monospace output destination.

Interaction: By default, PROC TABULATE allots space to row titles that are blank. Use ROW=FLOAT in the TABLE statement to divide the space among only nonblank titles.

See also: For more information about controlling the space for row titles, see Chapter 5, “Controlling the Table’s Appearance,” in *SAS Guide to TABULATE Processing*.

Featured in: Example 1 on page 1310

STYLE=<*style-element-name*><[*style-attribute-specification(s)*]>

specifies a style element to use for parts of the table other than table cells. For information about the arguments of this option and how it is used, see STYLE= on page 1273 in the PROC TABULATE statement.

Note: The list of attributes that you can set or change with the STYLE= option in the TABLE statement differs from that of the PROC TABULATE statement. Δ

The following table shows the attributes that you can set or change with the STYLE= option in the TABLE statement. Most of these attributes apply to parts of

the table other than cells (for example, table borders and the lines between columns and rows). Attributes that you apply in the PROC TABULATE statement and in other locations in the PROC TABULATE step apply to cells within the table. Note that not all attributes are valid in all destinations. See *SAS Output Delivery System User's Guide* for more information about these style attributes, their valid values, and their applicable destinations.

BACKGROUND=	FONT_WIDTH=*
BACKGROUNDIMAGE=	FOREGROUND=*
BORDERCOLOR=	FRAME=
BORDERCOLORDARK=	HTMLCLASS=
BORDERCOLORLIGHT=	JUST=
BORDERWIDTH=	OUTPUTWIDTH=
CELLPADDING=	POSTHTML=
CELLSPACING=	POSTIMAGE=
FONT=*	POSTTEXT=
FONT_FACE=*	PREHTML=
FONT_SIZE=*	PREIMAGE=
FONT_STYLE=*	PRETEXT=
FONT_WEIGHT=*	RULES=

* When you use these attributes in this location, they affect only the text that is specified with the PRETEXT=, POSTTEXT=, PREHTML=, and POSTHTML= attributes. To alter the foreground color or the font for the text that appears in the table, you must set the corresponding attribute in a location that affects the cells rather than the table.

Note: You can use braces ({ and }) instead of square brackets ([and]). △

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: To override a style element specification that is made as an option in the TABLE statement, specify STYLE= in a dimension expression of the TABLE statement.

Featured in: Example 14 on page 1357

Constructing Dimension Expressions

What Are Dimension Expressions?

A *dimension expression* defines the content and appearance of a dimension (the columns, rows, or pages in the table) by specifying the combination of variables, variable values, and statistics that make up that dimension. A TABLE statement consists of from one to three dimension expressions separated by commas. Options can follow the dimension expressions.

If all three dimensions are specified, then the leftmost dimension expression defines pages, the middle dimension expression defines rows, and the rightmost dimension expression defines columns. If two dimensions are specified, then the left dimension

expression defines rows, and the right dimension expression defines columns. If a single dimension is specified, then the dimension expression defines columns.

A dimension expression is composed of one or more elements and operators.

Elements That You Can Use in a Dimension Expression

analysis variables

(see “VAR Statement” on page 1289).

class variables

(see “CLASS Statement” on page 1276).

the universal class variable ALL

summarizes all of the categories for class variables in the same parenthetical group or dimension (if the variable ALL is not contained in a parenthetical group).

Featured in: Example 6 on page 1324, Example 9 on page 1330, and Example 13 on page 1347

Note: If the input data set contains a variable named ALL, then enclose the name of the universal class variable in quotation marks. Δ

keywords for statistics

See “Statistics That Are Available in PROC TABULATE” on page 1291 for a list of available statistics. Use the asterisk (*) operator to associate a statistic keyword with a variable. The N statistic (number of nonmissing values) can be specified in a dimension expression without associating it with a variable.

Restriction: Statistic keywords other than N must be associated with an analysis variable.

Default: For analysis variables, the default statistic is SUM. Otherwise, the default statistic is N.

Examples:

```
n
Region*n
Sales*max
```

Featured in: Example 10 on page 1333 and Example 13 on page 1347

format modifiers

define how to format values in cells. Use the asterisk (*) operator to associate a format modifier with the element (an analysis variable or a statistic) that produces the cells that you want to format. Format modifiers have the form

```
f=format
```

Example:

```
Sales*f=dollar8.2
```

Tip: Format modifiers have no effect on CLASS variables.

See also: For more information on specifying formats in tables, see “Formatting Values in Tables” on page 1293.

Featured in: Example 6 on page 1324

labels

temporarily replace the names of variables and statistics. Labels affect only the variable or statistic that immediately precedes the label. Labels have the form

```
statistic-keyword-or-variable-name='label-text'
```

Tip: PROC TABULATE eliminates the space for blank column headings from a table but by default does not eliminate the space for blank row headings unless all row headings are blank. Use ROW=FLOAT in the TABLE statement to remove the space for blank row headings.

Examples:

```
Region='Geographical Region'
Sales*max='Largest Sale'
```

Featured in: Example 5 on page 1322 and Example 7 on page 1326

style-element specifications

specify style elements for page dimension text, headings, or data cells. For details, see “Specifying Style Elements in Dimension Expressions” on page 1288.

Operators That You Can Use in a Dimension Expression

asterisk *

creates categories from the combination of values of the class variables and constructs the appropriate headers for the dimension. If one of the elements is an analysis variable, then the statistics for the analysis variable are calculated for the categories that are created by the class variables. This process is called *crossing*.

Examples:

```
Region*Division
Quarter*Sales*f=dollar8.2
```

Featured in: Example 1 on page 1310

(blank)

places the output for each element immediately after the output for the preceding element. This process is called *concatenation*.

Example:

```
n Region*Sales ALL
```

Featured in: Example 6 on page 1324

parentheses ()

group elements and associate an operator with each concatenated element in the group.

Examples:

```
Division*(Sales*max Sales*min)
(Region ALL)*Sales
```

Featured in: Example 6 on page 1324

angle brackets <>

specify denominator definitions, which determine the value of the denominator in the calculation of a percentage. For a discussion of how to construct denominator definitions, see “Calculating Percentages” on page 1294.

Featured in: Example 10 on page 1333 and Example 13 on page 1347

Specifying Style Elements in Dimension Expressions

You can specify a style element in a dimension expression to control the appearance in HTML, RTF, and Printer output of the following table elements:

- analysis variable name headings
- class variable name headings

class variable level value headings
 data cells
 keyword headings
 page dimension text

Specifying a style element in a dimension expression is useful when you want to override a style element that you have specified in another statement, such as the PROC TABULATE, CLASS, CLASSLEV, KEYWORD, TABLE, or VAR statements.

The syntax for specifying a style element in a dimension expression is

```
[STYLE<(CLASSLEV)>=<style-element-name |
  <PARENT >><[style-attribute-specification(s)]>]
```

Some examples of style elements in dimension expressions are

```
dept={label='Department'
      style=[foreground=red]}, N
dept*[style=MyDataStyle], N
dept*[format=12.2 style=MyDataStyle], N
```

Note: When used in a dimension expression, the STYLE= option must be enclosed within square brackets ([and]) or braces ({ and }). Δ

With the exception of (CLASSLEV), all arguments are described in STYLE= on page 1273 in the PROC TABULATE statement.

(CLASSLEV)

assigns a style element to a class variable level value heading. For example, the following TABLE statement specifies that the level value heading for the class variable, DEPT, has a foreground color of yellow:

```
table dept=[style(classlev)=
             [foreground=yellow]]*sales;
```

Note: This option is used only in dimension expressions. Δ

For an example that shows how to specify style elements within dimension expressions, see Example 14 on page 1357.

VAR Statement

Identifies numeric variables to use as analysis variables.

Alias: VARIABLES

Tip: You can use multiple VAR statements.

VAR *analysis-variable(s)* </option(s)>;

Required Arguments

analysis-variable(s);

identifies the analysis variables in the table. Analysis variables are numeric variables for which PROC TABULATE calculates statistics. The values of an analysis variable can be continuous or discrete.

If an observation contains a missing value for an analysis variable, then PROC TABULATE omits that value from calculations of all statistics except N (the number of observations with nonmissing variable values) and NMISS (the number of observations with missing variable values). For example, the missing value does not increase the SUM, and it is not counted when you are calculating statistics such as the MEAN.

Options

STYLE=<style-element-name | <PARENT>><[style-attribute-specification(s)]>

specifies a style element for analysis variable name headings. For information on the arguments of this option and how it is used, see STYLE= on page 1273 in the PROC TABULATE statement.

Note: When you use STYLE= in the VAR statement, it differs slightly from its use in the PROC TABULATE statement. In the VAR statement, the parent of the heading is the heading under which the current heading is nested. △

Alias: S=

Restriction: This option affects only the HTML, RTF, and Printer destinations.

Tip: To override a style element that is specified in the VAR statement, you can specify a style element in the related TABLE statement dimension expression.

Featured in: Example 14 on page 1357

WEIGHT=weight-variable

specifies a numeric variable whose values weight the values of the variables that are specified in the VAR statement. The variable does not have to be an integer. If the value of the weight variable is

Weight value...	PROC TABULATE...
0	counts the observation in the total number of observations
less than 0	converts the value to zero and counts the observation in the total number of observations
missing	excludes the observation

To exclude observations that contain negative and zero weights from the analysis, use EXCLNPWGT. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default.

Restriction: To compute weighted quantiles, use QMETHOD=OS in the PROC statement.

Tip: When you use the WEIGHT= option, consider which value of the VARDEF= option is appropriate (see the discussion of VARDEF= on page 1274).

Tip: Use the WEIGHT option in multiple VAR statements to specify different weights for the analysis variables.

Note: Prior to Version 7 of SAS, the procedure did not exclude the observations with missing weights from the count of observations. △

WEIGHT Statement

Specifies weights for analysis variables in the statistical calculations.

See also: For information on calculating weighted statistics and for an example that uses the WEIGHT statement, see “Calculating Weighted Statistics” on page 60

WEIGHT *variable*;

Required Arguments

variable

specifies a numeric variable whose values weight the values of the analysis variables. The values of the variable do not have to be integers. PROC TABULATE responds to weight values in accordance with the following table.

Weight value	PROC TABULATE response
0	counts the observation in the total number of observations
less than 0	converts the value to zero and counts the observation in the total number of observations
missing	excludes the observation

To exclude observations that contain negative and zero weights from the analysis, use EXCLNPWGT. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default.

Restriction: To compute weighted quantiles, use QMETHOD=OS in the PROC statement.

Interaction: If you use the WEIGHT= option in a VAR statement to specify a weight variable, then PROC TABULATE uses this variable instead to weight those VAR statement variables.

Tip: When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See the discussion of VARDEF= on page 1274 and the calculation of weighted statistics in “Keywords and Formulas” on page 1578 for more information.

Note: Prior to Version 7 of SAS, the procedure did not exclude the observations with missing weights from the count of observations. Δ

Concepts: TABULATE Procedure

Statistics That Are Available in PROC TABULATE

Use the following keywords to request statistics in the TABLE statement or to specify statistic keywords in the KEYWORD or KEYLABEL statement. If a variable

name (class or analysis) and a statistic name are the same, then enclose the statistic name in single quotation marks — for example, '**MAX**'.

Descriptive statistic keywords

COLPCTN	PCTSUM
COLPCTSUM	RANGE
CSS	REPPCTN
CV	REPPCTSUM
KURTOSIS KURT	ROWPCTN
LCLM	ROWPCTSUM
MAX	SKEWNESS SKEW
MEAN	STDDEV STD
MIN	STDERR
N	SUM
NMISS	SUMWGT
PAGEPCTN	UCLM
PAGEPCTSUM	USS
PCTN	VAR

Quantile statistic keywords

MEDIAN P50	Q3 P75
P1	P90
P5	P95
P10	P99
Q1 P25	QRANGE

Hypothesis testing keywords

PROBT	T
-------	---

These statistics, the formulas that are used to calculate them, and their data requirements are discussed in “Keywords and Formulas” on page 1578.

To compute standard error of the mean (STDERR) or Student’s *t*-test, you must use the default value of the VARDEF= option, which is DF. The VARDEF= option is specified in the PROC TABULATE statement.

To compute weighted quantiles, you must use QMETHOD=OS in the PROC TABULATE statement.

Use both LCLM and UCLM to compute a two-sided confidence limit for the mean. Use only LCLM or UCLM to compute a one-sided confidence limit. Use the ALPHA= option in the PROC TABULATE statement to specify a confidence level.

Formatting Class Variables

Use the FORMAT statement to assign a format to a class variable for the duration of a PROC TABULATE step. When you assign a format to a class variable, PROC

TABULATE uses the formatted values to create categories, and it uses the formatted values in headings. If you do not specify a format for a class variable, and the variable does not have any other format assigned to it, then the default format, BEST12., is used, unless the GROUPINTERNAL option is specified.

User-defined formats are particularly useful for grouping values into fewer categories. For example, if you have a class variable, Age, with values ranging from 1 to 99, then you could create a user-defined format that groups the ages so that your tables contain a manageable number of categories. The following PROC FORMAT step creates a format that condenses all possible values of age into six groups of values.

```
proc format;
  value agefmt  0-29='Under 30'
               30-39='30-39'
               40-49='40-49'
               50-59='50-59'
               60-69='60-69'
               other='70 or over';
run;
```

For information on creating user-defined formats, see Chapter 21, “The FORMAT Procedure,” on page 441.

By default, PROC TABULATE includes in a table only those formats for which the frequency count is not zero and for which values are not missing. To include missing values for all class variables in the output, use the MISSING option in the PROC TABULATE statement, and to include missing values for selected class variables, use the MISSING option in a CLASS statement. To include formats for which the frequency count is zero, use the PRELOADFMT option in a CLASS statement and the PRINTMISS option in the TABLE statement, or use the CLASSDATA= option in the PROC TABULATE statement.

Formatting Values in Tables

The formats for data in table cells serve two purposes. They determine how PROC TABULATE displays the values, and they determine the width of the columns. The default format for values in table cells is 12.2. You can modify the format for printing values in table cells by

- changing the default format with the FORMAT= option in the PROC TABULATE statement
- crossing elements in the TABLE statement with the F= format modifier.

PROC TABULATE determines the format to use for a particular cell from the following order of precedence for formats:

- 1 If no other formats are specified, then PROC TABULATE uses the default format (12.2).
- 2 The FORMAT= option in the PROC TABULATE statement changes the default format. If no format modifiers affect a cell, then PROC TABULATE uses this format for the value in that cell.
- 3 A format modifier in the page dimension applies to the values in all the table cells on the page unless you specify another format modifier for a cell in the row or column dimension.
- 4 A format modifier in the row dimension applies to the values in all the table cells in the row unless you specify another format modifier for a cell in the column dimension.

- 5 A format modifier in the column dimension applies to the values in all the table cells in the column.

For more information about formatting table cells, see “Formatting Values in Table Cells” in Chapter 5, “Controlling the Table’s Appearance,” in *SAS Guide to TABULATE Processing*.

How Using BY-Group Processing Differs from Using the Page Dimension

Using the page-dimension expression in a TABLE statement can have an effect similar to using a BY statement.

Table 43.4 on page 1294 contrasts the two methods.

Table 43.4 Contrasting the BY Statement and the Page Dimension

Issue	PROC TABULATE with a BY statement	PROC TABULATE with a page dimension in the TABLE statement
Order of observations in the input data set	The observations in the input data set must be sorted by the BY variables. ¹	Sorting is unnecessary.
One report summarizing all BY groups	You cannot create one report for all the BY groups.	Use ALL in the page dimension to create a report for all classes. (See Example 6 on page 1324.)
Percentages	The percentages in the tables are percentages of the total for that BY group. You cannot calculate percentages for a BY group compared to the totals for all BY groups because PROC TABULATE prepares the individual reports separately. Data for the report for one BY group are not available to the report for another BY group.	You can use denominator definitions to control the meaning of PCTN (see “Calculating Percentages” on page 1294.)
Titles	You can use the #BYVAL, #BYVAR, and #BYLINE specifications in TITLE statements to customize the titles for each BY group (see “Creating Titles That Contain BY-Group Information” on page 19).	The BOX= option in the TABLE statement customizes the page headers, but you must use the same title on each page.
Ordering class variables	ORDER=DATA and ORDER=FREQ order each BY group independently.	The order of class variables is the same on every page.
Obtaining uniform headings	You may need to insert dummy observations into BY groups that do not have all classes represented.	The PRINTMISS option ensures that each page of the table has uniform headings.
Multiple ranges with the same format	PROC TABULATE produces a table for each range.	PROC TABULATE combines observations from the two ranges.

¹ You can use the BY statement without sorting the data set if the data set has an index for the BY variable.

Calculating Percentages

The following statistics print the percentage of the value in a single table cell in relation to the total of the values in a group of cells. No denominator definitions are

required; however, an analysis variable may be used as a denominator definition for percentage sum statistics.

REPPCTN and REPPCTSUM statistics—print the percentage of the value in a single table cell in relation to the total of the values in the report.

COLPCTN and COLPCTSUM statistics—print the percentage of the value in a single table cell in relation to the total of the values in the column.

ROWPCTN and ROWPCTSUM statistics—print the percentage of the value in a single table cell in relation to the total of the values in the row.

PAGEPCTN and PAGEPCTSUM statistics—print the percentage of the value in a single table cell in relation to the total of the values in the page.

These statistics calculate the most commonly used percentages. See Example 12 on page 1344 for an example.

PCTN and PCTSUM statistics can be used to calculate these same percentages. They allow you to manually define denominators. PCTN and PCTSUM statistics print the percentage of the value in a single table cell in relation to the value (used in the denominator of the calculation of the percentage) in another table cell or to the total of the values in a group of cells. By default, PROC TABULATE summarizes the values in all N cells (for PCTN) or all SUM cells (for PCTSUM) and uses the summarized value for the denominator. You can control the value that PROC TABULATE uses for the denominator with a denominator definition.

You place a denominator definition in angle brackets (< and >) next to the PCTN or PCTSUM statistic. The denominator definition specifies which categories to sum for the denominator.

This section illustrates how to specify denominator definitions in a simple table. Example 13 on page 1347 illustrates how to specify denominator definitions in a table that is composed of multiple subtables. For more examples of denominator definitions, see “How Percentages Are Calculated” in Chapter 3, “Details of TABULATE Processing,” in *SAS Guide to TABULATE Processing*.

Specifying a Denominator for the PCTN Statistic

The following PROC TABULATE step calculates the N statistic and three different versions of PCTN using the data set ENERGY on page 1310.

```
proc tabulate data=energy;
  class division type;
  table division*
    (n='Number of customers'
     pctn<type>='% of row' ①
     pctn<division>='% of column' ②
     pctn='% of all customers'), ③
    type/rtts=50;
  title 'Number of Users in Each Division';
run;
```

The TABLE statement creates a row for each value of Division and a column for each value of Type. Within each row, the TABLE statement nests four statistics: N and three different calculations of PCTN (see Figure 43.3 on page 1296). Each occurrence of PCTN uses a different denominator definition.

Figure 43.3 Three Different Uses of the PCTN Statistic with Frequency Counts Highlighted

Number of Users in Each Division			
1			
		Type	
		1	2
Division			
1	Number of customers	6.00	6.00
	% of row ❶	50.00	50.00
	% of column ❷	27.27	27.27
	% of all customers ❸	13.64	13.64
2	Number of customers	3.00	3.00
	% of row	50.00	50.00
	% of column	13.64	13.64
	% of all customers	6.82	6.82
3	Number of customers	8.00	8.00
	% of row	50.00	50.00
	% of column	36.36	36.36
	% of all customers	18.18	18.18
4	Number of customers	5.00	5.00
	% of row	50.00	50.00
	% of column	22.73	22.73
	% of all customers	11.36	11.36

- ❶ **<type>** sums the frequency counts for all occurrences of Type within the same value of Division. Thus, for Division=1, the denominator is 6 + 6, or 12.
- ❷ **<division>** sums the frequency counts for all occurrences of Division within the same value of Type. Thus, for Type=1, the denominator is 6 + 3 + 8 + 5, or 22.
- ❸ The third use of PCTN has no denominator definition. Omitting a denominator definition is the same as including all class variables in the denominator definition. Thus, for all cells, the denominator is 6 + 3 + 8 + 5 + 6 + 3 + 8 + 5, or 44.

Specifying a Denominator for the PCTSUM Statistic

The following PROC TABULATE step sums expenditures for each combination of Type and Division and calculates three different versions of PCTSUM.

```
proc tabulate data=energy format=8.2;
  class division type;
  var expenditures;
  table division*
    (sum='Expenditures'*f=dollar10.2
     pctsum<type>='% of row' ❶
```

```

pctsum<division>='% of column' ❷
pctsum='% of all customers'), ❸
type*expenditures/rt=40;
title 'Expenditures in Each Division';
run;

```

The TABLE statement creates a row for each value of Division and a column for each value of Type. Because Type is crossed with Expenditures, the value in each cell is the sum of the values of Expenditures for all observations that contribute to the cell. Within each row, the TABLE statement nests four statistics: SUM and three different calculations of PCTSUM (see Figure 43.4 on page 1297). Each occurrence of PCTSUM uses a different denominator definition.

Figure 43.4 Three Different Uses of the PCTSUM Statistic with Sums Highlighted

Expenditures in Each Division				1
Division		Type		
		1	2	
		Expend	Expend	
1	Expenditures	\$7,477.00	\$5,129.00	
	% of row ❶	59.31	40.69	
	% of column ❷	16.15	13.66	
	% of all customers ❸	8.92	6.12	
2	Expenditures	\$19,379.00	\$15,078.00	
	% of row	56.24	43.76	
	% of column	41.86	40.15	
	% of all customers	23.11	17.98	
3	Expenditures	\$5,476.00	\$4,729.00	
	% of row	53.66	46.34	
	% of column	11.83	12.59	
	% of all customers	6.53	5.64	
4	Expenditures	\$13,959.00	\$12,619.00	
	% of row	52.52	47.48	
	% of column	30.15	33.60	
	% of all customers	16.65	15.05	

- ❶ **<type>** sums the values of Expenditures for all occurrences of Type within the same value of Division. Thus, for Division=1, the denominator is \$7,477 + \$5,129.
- ❷ **<division>** sums the frequency counts for all occurrences of Division within the same value of Type. Thus, for Type=1, the denominator is \$7,477 + \$19,379 + \$5,476 + \$13,959.
- ❸ The third use of PCTN has no denominator definition. Omitting a denominator definition is the same as including all class variables in the denominator

definition. Thus, for all cells, the denominator is \$7,477 + \$19,379 + \$5,476 + \$13,959 + \$5,129 + \$15,078 + \$4,729 + \$12,619.

Using Style Elements in PROC TABULATE

If you use the Output Delivery System to create HTML, RTF, or Printer output from PROC TABULATE, then you can set the style element that the procedure uses for various parts of the table. Style elements determine presentation attributes, such as font face, font weight, color, and so forth. See “Output Delivery System” on page 32 for more information. The following table lists the default styles for various regions of a table.

Table 43.5 Default Styles for Table Regions

Region	Style
column headings	Header
box	Header
page dimension text	Beforecaption
row headings	Rowheader
data cells	Data
table	Table

You specify style elements for PROC TABULATE with the STYLE= option. The following table shows where you can use this option. Specifications in the TABLE statement override the same specification in the PROC TABULATE statement. However, any style attributes that you specify in the PROC TABULATE statement and that you do not override in the TABLE statement are inherited. For instance, if you specify a blue background and a white foreground for all data cells in the PROC TABULATE statement, and you specify a gray background for the data cells of a particular crossing in the TABLE statement, then the background for those data cells is gray, and the foreground is white (as specified in the PROC TABULATE statement).

Detailed information on STYLE= is provided in the documentation for individual statements.

Table 43.6 Using the STYLE= Option in PROC TABULATE

To set the style element for	Use STYLE in this statement
data cells	PROC TABULATE
page dimension text and class variable name headings	CLASS
class level value headings	CLASSLEV
keyword headings	KEYWORD
table borders, rules, and other parts that are not specified elsewhere	TABLE
box text	TABLE statement, BOX= option
missing values	TABLE statement, MISSING= option
analysis variable name headings	VAR

You can use a format to assign a style attribute value to any cell whose content is determined by value(s) of a class or analysis variable. For example, the following code assigns a red background to cells whose values are less than 10,000, yellow to cells whose values are between 10,000 and 20,000, and green to cells whose values are greater than 20,000:

```
proc format;
  value expfmt low-<10000='red'
                10000-<20000='yellow'
                20000-high='green';
run;

ods html body='external-HTML-file';
proc tabulate data=energy style=[background=expfmt.];
  class region division type;
  var expenditures;
  table (region all)*(division all),
        type*expenditures;
run;
ods html close;
```

Results: TABULATE Procedure

Missing Values

How a missing value for a variable in the input data set affects your output depends on how you use the variable in the PROC TABULATE step. Table 43.7 on page 1299 summarizes how the procedure treats missing values.

Table 43.7 Summary of How PROC TABULATE Treats Missing Values

If ...	PROC TABULATE, by default, ...	To override the default ...
an observation contains a missing value for an analysis variable	excludes that observation from the calculation of statistics (except N and NMISS) for that particular variable	no alternative
an observation contains a missing value for a class variable	excludes that observation from the table ¹	use MISSING in the PROC TABULATE statement, or MISSING in the CLASS statement
there are no data for a category	does not show the category in the table	use PRINTMISS in the TABLE statement, or use CLASSDATA= in the PROC TABULATE statement
every observation that contributes to a table cell contains a missing value for an analysis variable	displays a missing value for any statistics (except N and NMISS) in that cell	use MISSTEXT= in the TABLE statement

If ...	PROC TABULATE, by default, ...	To override the default ...
there are no data for a formatted value	does not display that formatted value in the table	use PRELOADFMT in the CLASS statement with PRINTMISS in the TABLE statement, or use CLASSDATA= in the PROC TABULATE statement, or add dummy observations to the input data set so that it contains data for each formatted value
a FREQ variable value is missing or is less than 1	does not use that observation to calculate statistics	no alternative
a WEIGHT variable value is missing or 0	uses a value of 0	no alternative

- 1 The CLASS statement applies to all TABLE statements in a PROC TABULATE step. Therefore, if you define a variable as a class variable, PROC TABULATE omits observations that have missing values for that variable even if you do not use the variable in a TABLE statement.

This section presents a series of PROC TABULATE steps that illustrate how PROC TABULATE treats missing values. The following program creates the data set and formats that are used in this section and prints the data set. The data set COMPREV contains no missing values (see Figure 43.5 on page 1301).

```
proc format;
    value centryfmt 1='United States'
                  2='Japan';
    value compfmt 1='Supercomputer'
                 2='Mainframe'
                 3='Midrange'
                 4='Workstation'
                 5='Personal Computer'
                 6='Laptop';
run;

data comprev;
    input Country Computer Rev90 Rev91 Rev92;
    datalines;
1 1 788.8 877.6 944.9
1 2 12538.1 9855.6 8527.9
1 3 9815.8 6340.3 8680.3
1 4 3147.2 3474.1 3722.4
1 5 18660.9 18428.0 23531.1
2 1 469.9 495.6 448.4
2 2 5697.6 6242.4 5382.3
2 3 5392.1 5668.3 4845.9
2 4 1511.6 1875.5 1924.5
2 5 4746.0 4600.8 4363.7
;

proc print data=comprev noobs;
    format country centryfmt. computer compfmt.;
    title 'The Data Set COMPREV';
run;
```

Figure 43.5 The Data Set COMPREV

The Data Set COMPREV					1
Country	Computer	Rev90	Rev91	Rev92	
United States	Supercomputer	788.8	877.6	944.9	
United States	Mainframe	12538.1	9855.6	8527.9	
United States	Midrange	9815.8	6340.3	8680.3	
United States	Workstation	3147.2	3474.1	3722.4	
United States	Personal Computer	18660.9	18428.0	23531.1	
Japan	Supercomputer	469.9	495.6	448.4	
Japan	Mainframe	5697.6	6242.4	5382.3	
Japan	Midrange	5392.1	5668.3	4845.9	
Japan	Workstation	1511.6	1875.5	1924.5	
Japan	Personal Computer	4746.0	4600.8	4363.7	

No Missing Values

The following PROC TABULATE step produces Figure 43.6 on page 1302:

```
proc tabulate data=comprev;
  class country computer;
  var rev90 rev91 rev92;
  table computer*country,rev90 rev91 rev92 /
    rts=32;
  format country centryfmt. computer compfmt.;
  title 'Revenues from Computer Sales';
  title2 'for 1990 to 1992';
run;
```

Figure 43.6 Computer Sales Data: No Missing Values

Because the data set contains no missing values, the table includes all observations. All headers and cells contain nonmissing values.

Revenues from Computer Sales for 1990 to 1992					1
		Rev90	Rev91	Rev92	
		Sum	Sum	Sum	
Computer	Country				
Supercomputer	United States	788.80	877.60	944.90	
	Japan	469.90	495.60	448.40	
Mainframe	United States	12538.10	9855.60	8527.90	
	Japan	5697.60	6242.40	5382.30	
Midrange	United States	9815.80	6340.30	8680.30	
	Japan	5392.10	5668.30	4845.90	
Workstation	United States	3147.20	3474.10	3722.40	
	Japan	1511.60	1875.50	1924.50	
Personal Computer	United States	18660.90	18428.00	23531.10	
	Japan	4746.00	4600.80	4363.70	

A Missing Class Variable

The next program copies COMPREV and alters the data so that the eighth observation has a missing value for Computer. Except for specifying this new data set, the program that produces Figure 43.7 on page 1303 is the same as the program that produces Figure 43.6 on page 1302. By default, PROC TABULATE ignores observations with missing values for a class variable.

```
data compmiss;
  set comprev;
  if _n_=8 then computer=.;
run;

proc tabulate data=compmiss;
  class country computer;
  var rev90 rev91 rev92;
  table computer*country, rev90 rev91 rev92 /
    rts=32;
  format country cnyfmt. computer compfmt.;
  title 'Revenues from Computer Sales';
  title2 'for 1990 to 1992';
run;
```


Figure 43.7 Computer Sales Data: Midrange, Japan, Deleted

The observation with a missing value for Computer was the category **Midrange, Japan**. This category no longer exists. By default, PROC TABULATE ignores observations with missing values for a class variable, so this table contains one fewer row than Figure 43.6 on page 1302.

Revenues from Computer Sales for 1990 to 1992					1
		Rev90	Rev91	Rev92	
		Sum	Sum	Sum	
Computer	Country				
Supercomputer	United States	788.80	877.60	944.90	
	Japan	469.90	495.60	448.40	
Mainframe	United States	12538.10	9855.60	8527.90	
	Japan	5697.60	6242.40	5382.30	
Midrange	United States	9815.80	6340.30	8680.30	
Workstation	United States	3147.20	3474.10	3722.40	
	Japan	1511.60	1875.50	1924.50	
Personal Computer	United States	18660.90	18428.00	23531.10	
	Japan	4746.00	4600.80	4363.70	

Including Observations with Missing Class Variables

This program adds the MISSING option to the previous program. MISSING is available either in the PROC TABULATE statement or in the CLASS statement. If you want MISSING to apply only to selected class variables, but not to others, then specify MISSING in a separate CLASS statement with the selected variable(s). The MISSING option includes observations with missing values of a class variable in the report (see Figure 43.8 on page 1304).

```
proc tabulate data=compmiss missing;
  class country computer;
  var rev90 rev91 rev92;
  table computer*country,rev90 rev91 rev92 /
    rts=32;
  format country centryfmt. computer compfmt.;
  title 'Revenues from Computer Sales';
  title2 'for 1990 to 1992';
run;
```

Figure 43.8 Computer Sales Data: Missing Values for Computer

This table includes a category with missing values of Computer. This category makes up the first row of data in the table.

Revenues from Computer Sales for 1990 to 1992					1
		Rev90	Rev91	Rev92	
		Sum	Sum	Sum	
Computer	Country				
.	Japan	5392.10	5668.30	4845.90	
Supercomputer	United States	788.80	877.60	944.90	
	Japan	469.90	495.60	448.40	
Mainframe	United States	12538.10	9855.60	8527.90	
	Japan	5697.60	6242.40	5382.30	
Midrange	United States	9815.80	6340.30	8680.30	
Workstation	United States	3147.20	3474.10	3722.40	
	Japan	1511.60	1875.50	1924.50	
Personal Computer	United States	18660.90	18428.00	23531.10	
	Japan	4746.00	4600.80	4363.70	

Formatting Headings for Observations with Missing Class Variables

By default, as shown in Figure 43.8 on page 1304, PROC TABULATE displays missing values of a class variable as one of the standard SAS characters for missing values (a period, a blank, an underscore, or one of the letters A through Z). If you want to display something else instead, then you must assign a format to the class variable that has missing values, as shown in the following program (see Figure 43.9 on page 1305):

```
proc format;
    value misscomp 1='Supercomputer'
                  2='Mainframe'
                  3='Midrange'
                  4='Workstation'
                  5='Personal Computer'
                  6='Laptop'
                  .='No type given';
run;

proc tabulate data=compmiss missing;
    class country computer;
    var rev90 rev91 rev92;
    table computer*country, rev90 rev91 rev92 /
        rts=32;
    format country centryfmt. computer misscomp.;
```

```

title 'Revenues for Computer Sales';
title2 'for 1990 to 1992';
run;

```

Figure 43.9 Computer Sales Data: Text Supplied for Missing Computer Value

In this table, the missing value appears as the text that the MISSCOMP. format specifies.

Revenues for Computer Sales for 1990 to 1992					1
		Rev90	Rev91	Rev92	
		Sum	Sum	Sum	
Computer	Country				
No type given	Japan	5392.10	5668.30	4845.90	
Supercomputer	United States	788.80	877.60	944.90	
	Japan	469.90	495.60	448.40	
Mainframe	United States	12538.10	9855.60	8527.90	
	Japan	5697.60	6242.40	5382.30	
Midrange	United States	9815.80	6340.30	8680.30	
Workstation	United States	3147.20	3474.10	3722.40	
	Japan	1511.60	1875.50	1924.50	
Personal Computer	United States	18660.90	18428.00	23531.10	
	Japan	4746.00	4600.80	4363.70	

Providing Headings for All Categories

By default, PROC TABULATE evaluates each page that it prints and omits columns and rows for categories that do not exist. For example, Figure 43.9 on page 1305 does not include a row for **No type given** and for **United States** or for **Midrange** and for **Japan** because there are no data in these categories. If you want the table to represent all possible categories, then use the PRINTMISS option in the TABLE statement, as shown in the following program (see Figure 43.10 on page 1306):

```

proc tabulate data=compmiss missing;
  class country computer;
  var rev90 rev91 rev92;
  table computer*country, rev90 rev91 rev92 /
    rts=32 printmiss;
  format country cnyrfmt. computer misscomp.;
  title 'Revenues for Computer Sales';
  title2 'for 1990 to 1992';
run;

```

Figure 43.10 Computer Sales Data: Missing Statistics Values

This table contains a row for the categories **No type given**, **United States** and **Midrange**, **Japan**. Because there are no data in these categories, the values for the statistics are all missing.

Revenues for Computer Sales for 1990 to 1992					1
		Rev90	Rev91	Rev92	
		Sum	Sum	Sum	
Computer	Country				
No type given	United States	.	.	.	
	Japan	5392.10	5668.30	4845.90	
Supercomputer	United States	788.80	877.60	944.90	
	Japan	469.90	495.60	448.40	
Mainframe	United States	12538.10	9855.60	8527.90	
	Japan	5697.60	6242.40	5382.30	
Midrange	United States	9815.80	6340.30	8680.30	
	Japan	.	.	.	
Workstation	United States	3147.20	3474.10	3722.40	
	Japan	1511.60	1875.50	1924.50	
Personal Computer	United States	18660.90	18428.00	23531.10	
	Japan	4746.00	4600.80	4363.70	

Providing Text for Cells That Contain Missing Values

If some observations in a category contain missing values for analysis variables, then PROC TABULATE does not use those observations to calculate statistics (except N and NMISS). However, if each observation in a category contains a missing value, then PROC TABULATE displays a missing value for the value of the statistic. To replace missing values for analysis variables with text, use the MISSTEXT= option in the TABLE statement to specify the text to use, as shown in the following program (see Figure 43.11 on page 1307).

```
proc tabulate data=compmiss missing;
  class country computer;
  var rev90 rev91 rev92;
  table computer*country, rev90 rev91 rev92 /
    rts=32 printmiss misstext='NO DATA!';
  format country centryfmt. computer misscomp.;
  title 'Revenues for Computer Sales';
  title2 'for 1990 to 1992';
run;
```

Figure 43.11 Computer Sales Data: Text Supplied for Missing Statistics Values

This table replaces the period normally used to display missing values with the text of the MISSTEXT= option.

Revenues for Computer Sales for 1990 to 1992					1
		Rev90	Rev91	Rev92	
		Sum	Sum	Sum	
Computer	Country				
No type given	United States	NO DATA!	NO DATA!	NO DATA!	
	Japan	5392.10	5668.30	4845.90	
Supercomputer	United States	788.80	877.60	944.90	
	Japan	469.90	495.60	448.40	
Mainframe	United States	12538.10	9855.60	8527.90	
	Japan	5697.60	6242.40	5382.30	
Midrange	United States	9815.80	6340.30	8680.30	
	Japan	NO DATA!	NO DATA!	NO DATA!	
Workstation	United States	3147.20	3474.10	3722.40	
	Japan	1511.60	1875.50	1924.50	
Personal Computer	United States	18660.90	18428.00	23531.10	
	Japan	4746.00	4600.80	4363.70	

Providing Headings for All Values of a Format

PROC TABULATE prints headings only for values that appear in the input data set. For example, the format COMPFMT. provides for six possible values of Computer. Only five of these values occur in the data set COMPREV. The data set contains no data for laptop computers.

If you want to include headings for all possible values of Computer (perhaps to make it easier to compare the output with tables that are created later when you do have data for laptops), then you have three different ways to create such a table:

- Use the PRELOADFMT option in the CLASS statement with the PRINTMISS option in the TABLE statement. See Example 3 on page 1314 for another example that uses PRELOADFMT.
- Use the CLASSDATA= option in the PROC TABULATE statement. See Example 2 on page 1312 for an example that uses the CLASSDATA= option.
- Add dummy values to the input data set so that each value that the format handles appears at least once in the data set.

The following program adds the PRELOADFMT option to a CLASS statement that contains the relevant variable.

The results are shown in Figure 43.12 on page 1308.

```
proc tabulate data=compmiss missing;
  class country;
```

```

class computer / preloadfmt;
var rev90 rev91 rev92;
table computer*country,rev90 rev91 rev92 /
    rts=32 printmiss misstext='NO DATA!';
format country cntryfmt. computer compfmt.;
title 'Revenues for Computer Sales';
title2 'for 1990 to 1992';
run;

```

Figure 43.12 Computer Sales Data: All Possible Computer Values Included

This table contains a heading for each possible value of Computer.

Revenues for Computer Sales for 1990 to 1992					1
		Rev90	Rev91	Rev92	
		Sum	Sum	Sum	
Computer	Country				
.	United States	NO DATA!	NO DATA!	NO DATA!	
	Japan	5392.10	5668.30	4845.90	
Supercomputer	United States	788.80	877.60	944.90	
	Japan	469.90	495.60	448.40	
Mainframe	United States	12538.10	9855.60	8527.90	
	Japan	5697.60	6242.40	5382.30	
Midrange	United States	9815.80	6340.30	8680.30	
	Japan	NO DATA!	NO DATA!	NO DATA!	
Workstation	United States	3147.20	3474.10	3722.40	
	Japan	1511.60	1875.50	1924.50	
Personal Computer	United States	18660.90	18428.00	23531.10	
	Japan	4746.00	4600.80	4363.70	
Laptop	United States	NO DATA!	NO DATA!	NO DATA!	
	Japan	NO DATA!	NO DATA!	NO DATA!	

Understanding the Order of Headings with ORDER=DATA

The ORDER= option applies to all class variables. Occasionally, you want to order the headings for different variables differently. One method for doing this is to group the data as you want them to appear and to specify ORDER=DATA.

For this technique to work, the first value of the first class variable must occur in the data with all possible values of all the other class variables. If this criterion is not met, then the order of the headings might surprise you.

Examples: TABULATE Procedure

Example 1: Creating a Basic Two-Dimensional Table

Procedure features:

PROC TABULATE statement options:

FORMAT=

TABLE statement

crossing (*) operator

TABLE statement options:

RTS=

Other features: FORMAT statement

This example

- ☐ creates a category for each type of user (residential or business) in each division of each region
- ☐ applies the same format to all cells in the table
- ☐ applies a format to each class variable
- ☐ extends the space for row headings.

Program

Create the ENERGY data set. ENERGY contains data on expenditures of energy for business and residential customers in individual states in the Northeast and West regions of the United States. A DATA step on page 1625 creates the data set.

```
data energy;
    length State $2;
    input Region Division state $ Type Expenditures;
    datalines;
1 1 ME 1 708
1 1 ME 2 379

. . . more data lines . . .

4 4 HI 1 273
4 4 HI 2 298
;
```

Create the REGFMT., DIVFMT., and USETYPE. formats. PROC FORMAT creates formats for Region, Division, and Type.


```

proc format;
  value regfmt 1='Northeast'
              2='South'
              3='Midwest'
              4='West';
  value divfmt 1='New England'
              2='Middle Atlantic'
              3='Mountain'
              4='Pacific';
  value usetype 1='Residential Customers'
               2='Business Customers';
run;

```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Specify the table options. The FORMAT= option specifies DOLLAR12. as the default format for the value in each table cell.

```
proc tabulate data=energy format=dollar12.;
```

Specify subgroups for the analysis. The CLASS statement separates the analysis by values of Region, Division, and Type.

```
class region division type;
```

Specify the analysis variable. The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;
```

Define the table rows and columns. The TABLE statement creates a row for each formatted value of Region. Nested within each row are rows for each formatted value of Division. The TABLE statement also creates a column for each formatted value of Type. Each cell that is created by these rows and columns contains the sum of the analysis variable Expenditures for all observations that contribute to that cell.

```
table region*division,
      type*expenditures
```

Specify the row title space. RTS= provides 25 characters per line for row headings.

```
/ rts=25;
```

Format the output. The FORMAT statement assigns formats to the variables Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

Specify the titles.

```
title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
run;
```

Output

Energy Expenditures for Each Region (millions of dollars)				1
		Type		
		Residential Customers	Business Customers	
		Expenditures	Expenditures	
		Sum	Sum	
Region	Division			
Northeast	New England	\$7,477	\$5,129	
	Middle Atlantic	\$19,379	\$15,078	
West	Mountain	\$5,476	\$4,729	
	Pacific	\$13,959	\$12,619	

Example 2: Specifying Class Variable Combinations to Appear in a Table

Procedure features:

PROC TABULATE Statement options:

CLASSDATA=
EXCLUSIVE

Data set: ENERGY on page 1310

Formats: REGFMT., DIVFMT., and USETYPE. on page 1311

This example

- uses the CLASSDATA= option to specify combinations of class variables to appear in a table
- uses the EXCLUSIVE option to restrict the output to only the combinations specified in the CLASSDATA= data set. Without the EXCLUSIVE option, the output would be the same as in Example 1 on page 1310.

Program

Create the CLASSES data set. CLASSES contains the combinations of class variable values that PROC TABULATE uses to create the table.

```
data classes;
    input region division type;
    datalines;
1 1 1
1 1 2
4 4 1
4 4 2
;
```

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Specify the table options. CLASSDATA= and EXCLUSIVE restrict the class level combinations to those that are specified in the CLASSES data set.

```
proc tabulate data=energy format=dollar12.
    classdata=classes exclusive;
```

Specify subgroups for the analysis. The CLASS statement separates the analysis by values of Region, Division, and Type.

```
class region division type;
```

Specify the analysis variable. The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;
```

Define the table rows and columns. The TABLE statement creates a row for each formatted value of Region. Nested within each row are rows for each formatted value of Division. The TABLE statement also creates a column for each formatted value of Type. Each cell that is created by these rows and columns contains the sum of the analysis variable Expenditures for all observations that contribute to that cell.

```
table region*division,
      type*expenditures
```

Specify the row title space. RTS= provides 25 characters per line for row headings.

```
/ rts=25;
```

Format the output. The FORMAT statement assigns formats to the variables Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

Specify the titles.

```
title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
run;
```

Output

Energy Expenditures for Each Region (millions of dollars)				1
		Type		
		Residential Customers	Business Customers	
		Expenditures	Expenditures	
		Sum	Sum	
Region	Division			
Northeast	New England	\$7,477	\$5,129	
West	Pacific	\$13,959	\$12,619	

Example 3: Using Preloaded Formats with Class Variables

Procedure features:

PROC TABULATE statement option:

OUT=

CLASS statement options:

EXCLUSIVE

PRELOADFMT

TABLE statement option:

PRINTMISS

Other features: PRINT procedure

Data set: ENERGY on page 1310

Formats: REGFMT., DIVFMT., and USETYPE. on page 1311

This example

- ☐ creates a table that includes all possible combinations of formatted class variable values (PRELOADFMT with PRINTMISS), even if those combinations have a zero frequency and even if they do not make sense
- ☐ uses only the preloaded range of user-defined formats as the levels of class variables (PRELOADFMT with EXCLUSIVE).
- ☐ writes the output to an output data set, and prints that data set.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Specify the table options. The FORMAT= option specifies DOLLAR12. as the default format for the value in each table cell.

```
proc tabulate data=energy format=dollar12.;
```

Specify subgroups for the analysis. The CLASS statement separates the analysis by values of Region, Division, and Type. PRELOADFMT specifies that PROC TABULATE use the preloaded values of the user-defined formats for the class variables.

```
class region division type / preloadfmt;
```

Specify the analysis variable. The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;
```

Define the table rows and columns, and specify row and column options. PRINTMISS specifies that all possible combinations of user-defined formats be used as the levels of the class variables.

```
table region*division,
      type*expenditures / rts=25 printmiss;
```

Format the output. The FORMAT statement assigns formats to the variables Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

Specify the titles.

```
title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
run;
```

Specify the table options and the output data set. The OUT= option specifies the name of the output data set to which PROC TABULATE writes the data.

```
proc tabulate data=energy format=dollar12. out=tabdata;
```

Specify subgroups for the analysis. The EXCLUSIVE option, when used with PRELOADFMT, uses only the preloaded range of user-defined formats as the levels of class variables.

```
class region division type / preloadfmt exclusive;
```

Specify the analysis variable. The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;
```

Define the table rows and columns, and specify row and column options. The PRINTMISS option is not specified in this case. If it were, then it would override the EXCLUSIVE option in the CLASS statement.

```
table region*division,
      type*expenditures / rts=25;
```

Format the output. The FORMAT statement assigns formats to the variables Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

Specify the titles.

```
title 'Energy Expenditures for Each Region';  
title2 '(millions of dollars)';  
run;
```

Print the output data set WORK.TABDATA.

```
proc print data=tabdata;  
run;
```

Output

This output, created with the PRELOADFMT and PRINTMISS options, contains all possible combinations of preloaded user-defined formats for the class variable values. It includes combinations with zero frequencies, and combinations that make no sense, such as **Northeast** and **Pacific**.

Energy Expenditures for Each Region (millions of dollars)				1
		Type		
		Residential Customers	Business Customers	
		Expenditures	Expenditures	
		Sum	Sum	
Region	Division			
Northeast	New England	\$7,477	\$5,129	
	Middle Atlantic	\$19,379	\$15,078	
	Mountain	.	.	
	Pacific	.	.	
South	New England	.	.	
	Middle Atlantic	.	.	
	Mountain	.	.	
	Pacific	.	.	
Midwest	New England	.	.	
	Middle Atlantic	.	.	
	Mountain	.	.	
	Pacific	.	.	
West	New England	.	.	
	Middle Atlantic	.	.	
	Mountain	\$5,476	\$4,729	
	Pacific	\$13,959	\$12,619	

This output, created with the PRELOADFMT and EXCLUSIVE options, contains only those combinations of preloaded user-defined formats for the class variable values that appear in the input data set. This output is identical to the output from Example 1 on page 1310.

Energy Expenditures for Each Region (millions of dollars)				1
		Type		
		Residential Customers	Business Customers	
		Expenditures	Expenditures	
		Sum	Sum	
Region	Division			
Northeast	New England	\$7,477	\$5,129	
	Middle Atlantic	\$19,379	\$15,078	
West	Mountain	\$5,476	\$4,729	
	Pacific	\$13,959	\$12,619	

This output is a listing of the output data set TABDATA, which was created by the OUT= option in the PROC TABULATE statement. TABDATA contains the data that is created by having the PRELOADFMT and EXCLUSIVE options specified.

Energy Expenditures for Each Region (millions of dollars)							
O b s e r v a t i o n	R e g i o n	D i v i s i o n	T y p e	T Y P E	P A G E	T A B L E	E x p e n d i t u r e s S u m
1	Northeast	New England	Residential Customers	111	1	1	7477
2	Northeast	New England	Business Customers	111	1	1	5129
3	Northeast	Middle Atlantic	Residential Customers	111	1	1	19379
4	Northeast	Middle Atlantic	Business Customers	111	1	1	15078
5	West	Mountain	Residential Customers	111	1	1	5476
6	West	Mountain	Business Customers	111	1	1	4729
7	West	Pacific	Residential Customers	111	1	1	13959
8	West	Pacific	Business Customers	111	1	1	12619

Example 4: Using Multilabel Formats

Procedure features:

CLASS statement options:

MLF

PROC TABULATE statement options:

FORMAT=

TABLE statement

ALL class variable

concatenation (blank) operator

crossing (*) operator

grouping elements (parentheses) operator

label

variable list

Other features:

FORMAT procedure

FORMAT statement

VALUE statement options:

MULTILABEL

This example

- ☐ shows how to specify a multilabel format in the VALUE statement of PROC FORMAT
- ☐ shows how to activate multilabel format processing using the MLF option with the CLASS statement
- ☐ demonstrates the behavior of the N statistic when multilabel format processing is activated.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=64;
```

Create the CARSURVEY data set. CARSURVEY contains data from a survey that was distributed by a car manufacturer to a focus group of potential customers who were brought together to evaluate new car names. Each observation in the data set contains an identification number, the participant's age, and the participant's ratings of four car names. A DATA step creates the data set.

```
data carsurvey;
    input Rater Age Progressa Remark Jupiter Dynamo;
    datalines;
1   38   94   98   84   80
2   49   96   84   80   77
3   16   64   78   76   73
```

```

4    27  89  73  90  92

. . . more data lines . . .

77    61  92  88  77  85
78    24  87  88  88  91
79    18  54  50  62  74
80    62  90  91  90  86
;

```

Create the AGEFMT. format. The FORMAT procedure creates a multilabel format for ages by using the MULTILABEL option on page 460. A multilabel format is one in which multiple labels can be assigned to the same value, in this case because of overlapping ranges. Each value is represented in the table for each range in which it occurs. The NOTSORTED option stores the ranges in the order in which they are defined.

```

proc format;
  value agefmt (multilabel notsorted)
    15 - 29 = 'Below 30 years'
    30 - 50 = 'Between 30 and 50'
    51 - high = 'Over 50 years'
    15 - 19 = '15 to 19'
    20 - 25 = '20 to 25'
    25 - 39 = '25 to 39'
    40 - 55 = '40 to 55'
    56 - high = '56 and above';
run;

```

Specify the table options. The FORMAT= option specifies up to 10 digits as the default format for the value in each table cell.

```
proc tabulate data=carsurvey format=10.;
```

Specify subgroups for the analysis. The CLASS statement identifies Age as the class variable and uses the MLF option to activate multilabel format processing.

```
class age / mlf;
```

Specify the analysis variables. The VAR statement specifies that PROC TABULATE calculate statistics on the Progressa, Remark, Jupiter, and Dynamo variables.

```
var progressa remark jupiter dynamo;
```

Define the table rows and columns. The row dimension of the TABLE statement creates a row for each formatted value of Age. Multilabel formatting allows an observation to be included in multiple rows or age categories. The row dimension uses the ALL class variable to summarize information for all rows. The column dimension uses the N statistic to calculate the number of observations for each age group. Notice that the result of the N statistic crossed with the ALL class variable in the row dimension is the total number of observations instead of the sum of the N statistics for the rows. The column dimension uses the ALL class variable at the beginning of a crossing to assign a label, **Potential Car Names**. The four nested columns calculate the mean ratings of the car names for each age group.

```

table age all, n all='Potential Car Names'*(progressa remark
jupiter dynamo)*mean;

```

Specify the titles.

```

title1 "Rating Four Potential Car Names";
title2 "Rating Scale 0-100 (100 is the highest rating)";

```

Format the output. The FORMAT statement assigns the user-defined format AGEFMT. to Age for this analysis.

```

format age agefmt.;
run;

```

Output**Output 43.3**

Rating Four Potential Car Names Rating Scale 0-100 (100 is the highest rating)						1
	N	Potential Car Names				
		Progressa	Remark	Jupiter	Dynamo	
		Mean	Mean	Mean	Mean	
Age						
15 to 19	14	75	78	81	73	
20 to 25	11	89	88	84	89	
25 to 39	26	84	90	82	72	
40 to 55	14	85	87	80	68	
56 and above	15	84	82	81	75	
Below 30 years	36	82	84	82	75	
Between 30 and 50	25	86	89	81	73	
Over 50 years	19	82	84	80	76	
All	80	83	86	81	74	

Example 5: Customizing Row and Column Headings

Procedure features:

TABLE statement

labels

Data set: ENERGY on page 1310

Formats: REGFMT., DIVFMT., and USETYPE. on page 1311

This example shows how to customize row and column headings. A label specifies text for a heading. A blank label creates a blank heading. PROC TABULATE removes the space for blank column headings from the table.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Specify the table options. The FORMAT= option specifies DOLLAR12. as the default format for the value in each table cell.

```
proc tabulate data=energy format=dollar12.;
```

Specify subgroups for the analysis. The CLASS statement identifies Region, Division, and Type as class variables.

```
class region division type;
```

Specify the analysis variable. The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;
```

Define the table rows and columns. The TABLE statement creates a row for each formatted value of Region. Nested within each row are rows for each formatted value of Division. The TABLE statement also creates a column for each formatted value of Type. Each cell that is created by these rows and columns contains the sum of the analysis variable Expenditures for all observations that contribute to that cell. Text in quotation marks specifies headings for the corresponding variable or statistic. Although Sum is the default statistic, it is specified here so that you can specify a blank for its heading.

```
table region*division,
      type='Customer Base'*expenditures=' '*sum=' '
```

Specify the row title space. RTS= provides 25 characters per line for row headings.

```
/ rts=25;
```

Format the output. The FORMAT statement assigns formats to Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

Specify the titles.

```

title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
run;

```

Output

The heading for Type contains text that is specified in the TABLE statement. The TABLE statement eliminated the headings for Expenditures and Sum.

Energy Expenditures for Each Region (millions of dollars)				1
		Customer Base		
		Residential Customers	Business Customers	
Region	Division			
Northeast	New England	\$7,477	\$5,129	
	Middle Atlantic	\$19,379	\$15,078	
West	Mountain	\$5,476	\$4,729	
	Pacific	\$13,959	\$12,619	

Example 6: Summarizing Information with the Universal Class Variable ALL

Procedure features:

PROC TABULATE statement options:

FORMAT=

TABLE statement:

ALL class variable

concatenation (blank operator)

format modifiers

grouping elements (parentheses operator)

Data set: ENERGY on page 1310

Formats: REGFMT., DIVFMT., and USETYPE. on page 1311

This example shows how to use the universal class variable ALL to summarize information from multiple categories.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=64 pagesize=60;
```

Specify the table options. The FORMAT= option specifies COMMA12. as the default format for the value in each table cell.

```
proc tabulate data=energy format=comma12.;
```

Specify subgroups for the analysis. The CLASS statement identifies Region, Division, and Type as class variables.

```
class region division type;
```

Specify the analysis variable. The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;
```

Define the table rows. The row dimension of the TABLE statement creates a row for each formatted value of Region. Nested within each row are rows for each formatted value of Division and a row (labeled **Subtotal**) that summarizes all divisions in the region. The last row of the report (labeled **Total for All Regions**) summarizes all regions. The format modifier f=DOLLAR12. assigns the DOLLAR12. format to the cells in this row.

```
table region*(division all='Subtotal')
          all='Total for All Regions'*f=dollar12.,
```

Define the table columns. The column dimension of the TABLE statement creates a column for each formatted value of Type and a column that is labeled **All customers** that shows expenditures for all customers in a row of the table. Each cell that is created by these rows and columns contains the sum of the analysis variable Expenditures for all observations that contribute to that cell. Text in quotation marks specifies headings for the corresponding variable or statistic. Although Sum is the default statistic, it is specified here so that you can specify a blank for its heading.

```
type='Customer Base'*expenditures=' '*sum=' '
all='All Customers'*expenditures=' '*sum=' '
```

Specify the row title space. RTS= provides 25 characters per line for row headings.

```
/ rts=25;
```

Format the output. The FORMAT statement assigns formats to the variables Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

Specify the titles.

```
title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
run;
```

Output

The universal class variable ALL provides subtotals and totals in this table.

Energy Expenditures for Each Region (millions of dollars)					1
		Customer Base		All	
		Residential	Business	Customers	
		Customers	Customers	Customers	
Region	Division				
Northeast	New England	7,477	5,129	12,606	
	Middle Atlantic	19,379	15,078	34,457	
	Subtotal	26,856	20,207	47,063	
West	Division				
	Mountain	5,476	4,729	10,205	
	Pacific	13,959	12,619	26,578	
	Subtotal	19,435	17,348	36,783	
Total for All Regions		\$46,291	\$37,555	\$83,846	

Example 7: Eliminating Row Headings

Procedure features:

TABLE statement:

labels

ROW=FLOAT

Data set: ENERGY on page 1310

Formats: REGFMT., DIVFMT., and USETYPE. on page 1311

This example shows how to eliminate blank row headings from a table. To do so, you must both provide blank labels for the row headings and specify ROW=FLOAT in the TABLE statement.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Specify the table options. The FORMAT= option specifies DOLLAR12. as the default format for the value in each table cell.

```
proc tabulate data=energy format=dollar12.;
```

Specify subgroups for the analysis. The CLASS statement identifies Region, Division, and Type as class variables.

```
class region division type;
```

Specify the analysis variable. The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;
```

Define the table rows. The row dimension of the TABLE statement creates a row for each formatted value of Region. Nested within these rows is a row for each formatted value of Division. The analysis variable Expenditures and the Sum statistic are also included in the row dimension, so PROC TABULATE creates row headings for them as well. The text in quotation marks specifies the headings for the corresponding variable or statistic. Although Sum is the default statistic, it is specified here so that you can specify a blank for its heading.

```
table region*division*expenditures=' '*sum=' ',
```

Define the table columns. The column dimension of the TABLE statement creates a column for each formatted value of Type.

```
type='Customer Base'
```

Specify the row title space and eliminate blank row headings. RTS= provides 25 characters per line for row headings. ROW=FLOAT eliminates blank row headings.

```
/ rts=25 row=float;
```

Format the output. The FORMAT statement assigns formats to the variables Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

Specify the titles.

```
title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
run;
```

Output

Compare this table with the output in Example 5 on page 1322. The two tables are identical, but the program that creates this table uses Expenditures and Sum in the row dimension. PROC TABULATE automatically eliminates blank headings from the column dimension, whereas you must specify ROW=FLOAT to eliminate blank headings from the row dimension.

Energy Expenditures for Each Region (millions of dollars)				1
		Customer Base		
		Residential Customers	Business Customers	
Region	Division			
Northeast	New England	\$7,477	\$5,129	
	Middle Atlantic	\$19,379	\$15,078	
West	Mountain	\$5,476	\$4,729	
	Pacific	\$13,959	\$12,619	

Example 8: Indenting Row Headings and Eliminating Horizontal Separators

Procedure features:

PROC TABULATE statement options:

NOSEPS

TABLE statement options:

INDENT=

Data set: ENERGY on page 1310

Formats: REGFMT., DIVFMT., and USETYPE. on page 1311

This example shows how to condense the structure of a table by

- ☐ removing row headings for class variables
- ☐ indenting nested rows underneath parent rows instead of placing them next to each other
- ☐ eliminating horizontal separator lines from the row titles and the body of the table.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Specify the table options. The FORMAT= option specifies DOLLAR12. as the default format for the value in each table cell. NOSEPS eliminates horizontal separator lines from row titles and from the body of the table.

```
proc tabulate data=energy format=dollar12. noseps;
```

Specify subgroups for the analysis. The CLASS statement identifies Region, Division, and Type as class variables.

```
class region division type;
```

Specify the analysis variable. The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;
```

Define the table rows and columns. The TABLE statement creates a row for each formatted value of Region. Nested within each row are rows for each formatted value of Division. The TABLE statement also creates a column for each formatted value of Type. Each cell that is created by these rows and columns contains the sum of the analysis variable Expenditures for all observations that contribute to that cell. Text in quotation marks in all dimensions specifies headings for the corresponding variable or statistic. Although Sum is the default statistic, it is specified here so that you can specify a blank for its heading.

```
table region*division,
       type='Customer Base'*expenditures=' '*sum=' '
```

Specify the row title space and indentation value. RTS= provides 25 characters per line for row headings. INDENT= removes row headings for class variables, places values for Division beneath values for Region rather than beside them, and indents values for Division four spaces.

```
/ rts=25 indent=4;
```

Format the output. The FORMAT statement assigns formats to the variables Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

Specify the titles.

```
title 'Energy Expenditures for Each Region';
title2 '(millions of dollars)';
run;
```

Output

NOSEPS removes the separator lines from the row titles and the body of the table. INDENT= eliminates the row headings for Region and Division and indents values for Division underneath values for Region.

Energy Expenditures for Each Region (millions of dollars)			1
	Customer Base		
	Residential Customers	Business Customers	
Northeast			
New England	\$7,477	\$5,129	
Middle Atlantic	\$19,379	\$15,078	
West			
Mountain	\$5,476	\$4,729	
Pacific	\$13,959	\$12,619	

Example 9: Creating Multipage Tables

Procedure features:
TABLE statement

```

ALL class variable
BOX=
CONDENSE
INDENT=
page expression

```

Data set: ENERGY on page 1310

Formats: REGFMT., DIVFMT., and USETYPE. on page 1311

This example creates a separate table for each region and one table for all regions. By default, PROC TABULATE creates each table on a separate page, but the CONDENSE option places them all on the same page.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Specify the table options. The FORMAT= option specifies DOLLAR12. as the default format for the value in each table cell.

```
proc tabulate data=energy format=dollar12.;
```

Specify subgroups for the analysis. The CLASS statement identifies Region, Division, and Type as class variables.

```
class region division type;
```

Specify the analysis variable. The VAR statement specifies that PROC TABULATE calculate statistics on the Expenditures variable.

```
var expenditures;
```

Define the table pages. The page dimension of the TABLE statement creates one table for each formatted value of Region and one table for all regions. Text in quotation marks provides the heading for each page.

```
table region='Region: ' all='All Regions',
```

Define the table rows. The row dimension creates a row for each formatted value of Division and a row for all divisions. Text in quotation marks provides the row headings.

```
division all='All Divisions',
```

Define the table columns. The column dimension of the TABLE statement creates a column for each formatted value of Type. Each cell that is created by these pages, rows, and columns contains the sum of the analysis variable Expenditures for all observations that contribute to that cell. Text in quotation marks specifies headings for the corresponding variable or statistic. Although Sum is the default statistic, it is specified here so that you can specify a blank for its heading.

```
type='Customer Base'*expenditures=' '*sum=' '
```

Specify additional table options. RTS= provides 25 characters per line for row headings. BOX= places the page heading inside the box above the row headings. CONDENSE places as many tables as possible on one physical page. INDENT= eliminates the row heading for Division. (Because there is no nesting in the row dimension, there is nothing to indent.)

```
/ rts=25 box=_page_ condense indent=1;
```

Format the output. The FORMAT statement assigns formats to the variables Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

Specify the titles.

```
title 'Energy Expenditures for Each Region and All Regions';
title2 '(millions of dollars)';
run;
```

Output

Energy Expenditures for Each Region and All Regions
(millions of dollars)

1

Region: Northeast	Customer Base	
	Residential Customers	Business Customers
New England	\$7,477	\$5,129
Middle Atlantic	\$19,379	\$15,078
All Divisions	\$26,856	\$20,207

Region: West	Customer Base	
	Residential Customers	Business Customers
Mountain	\$5,476	\$4,729
Pacific	\$13,959	\$12,619
All Divisions	\$19,435	\$17,348

All Regions	Customer Base	
	Residential Customers	Business Customers
New England	\$7,477	\$5,129
Middle Atlantic	\$19,379	\$15,078
Mountain	\$5,476	\$4,729
Pacific	\$13,959	\$12,619
All Divisions	\$46,291	\$37,555

Example 10: Reporting on Multiple-Response Survey Data

Procedure features:

TABLE statement:

denominator definition (angle bracket operators)

N statistic

PCTN statistic

variable list

Other features:

FORMAT procedure

SAS system options:

FORMDLIM=

NONUMBER

SYMPUT routine

The two tables in this example show

- ☐ which factors most influenced customers' decisions to buy products
- ☐ where customers heard of the company.

The reports appear on one physical page with only one page number. By default, they would appear on separate pages.

In addition to showing how to create these tables, this example shows how to

- ☐ use a DATA step to count the number of observations in a data set
- ☐ store that value in a macro variable
- ☐ access that value later in the SAS session.

Collecting the Data

Figure 43.14 on page 1334 shows the survey form that is used to collect data.

Figure 43.14 Completed Survey Form

Customer Questionnaire

ID#: _____

Please place a check beside all answers that apply.

Why do you buy our products?

☐ Cost
 ☐ Performance
 ☐ Reliability
 ☐ Sales staff

How did you find out about our company?

☐ T.V. / Radio
 ☐ Newspaper / Magazine
 ☐ Word of mouth

What makes a sales person effective?

☐ Product knowledge
 ☐ Personality
 ☐ Appearance

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page. The FORMDLIM= option replaces the character that delimits page breaks with a single blank. By default, a new physical page starts whenever a page break occurs.


```
options nodate pageno=1 linesize=80 pagesize=18 formdlm=' ';
```

Create the CUSTOMER_RESPONSE data set. CUSTOMER_RESPONSE contains data from a customer survey. Each observation in the data set contains information about factors that influence one respondent's decisions to buy products. A DATA step on page 1619 creates the data set. Using missing values rather than 0's is crucial for calculating frequency counts in PROC TABULATE.

```
data customer_response;
    input Customer Factor1-Factor4 Source1-Source3
           Quality1-Quality3;
    datalines;
1 . . 1 1 1 1 . 1 . .
2 1 1 . 1 1 1 . 1 1 .
3 . . 1 1 1 1 . . . .

. . . more data lines . . .

119 . . . 1 . . . 1 . .
120 1 1 . 1 . . . . 1 .
;
```

Store the number of observations in a macro variable. The SET statement reads the descriptor portion of CUSTOMER_RESPONSE at compile time and stores the number of observations (the number of respondents) in COUNT. The SYMPUT routine stores the value of COUNT in the macro variable NUM. This variable is available for use by other procedures and DATA steps for the remainder of the SAS session. The IF 0 condition, which is always false, ensures that the SET statement, which reads the observations, never executes. (Reading observations is unnecessary.) The STOP statement ensures that the DATA step executes only once.

```
data _null_;
    if 0 then set customer_response nobs=count;
    call symput('num',left(put(count,4.)));
    stop;
run;
```

Create the PCTFMT. format. The FORMAT procedure creates a format for percentages. The PCTFMT. format writes all values with at least one digit to the left of the decimal point and with one digit to the right of the decimal point. A blank and a percent sign follow the digits.

```
proc format;
    picture pctfmt low-high='009.9 %';
run;
```

Create the report and use the default table options.

```
proc tabulate data=customer_response;
```

Specify the analysis variables. The VAR statement specifies that PROC TABULATE calculate statistics on the Factor1, Factor2, Factor3, Factor4, and Customer variables. The variable Customer must be listed because it is used to calculate the Percent column that is defined in the TABLE statement.

```
var factor1-factor4 customer;
```

Define the table rows and columns. The TABLE statement creates a row for each factor, a column for frequency counts, and a column for the percentages. Text in quotation marks supplies headers for the corresponding row or column. The format modifiers F=7. and F=PCTFMT9. provide formats for values in the associated cells and extend the column widths to accommodate the column headers.

```
table factor1='Cost'
      factor2='Performance'
      factor3='Reliability'
      factor4='Sales Staff',
      (n='Count'*f=7. pctl<customer>='Percent'*f=pctfmt9.) ;
```

Specify the titles.

```
title 'Customer Survey Results: Spring 1996';
title3 'Factors Influencing the Decision to Buy';
run;
```

Suppress page numbers. The SAS system option NONUMBER suppresses page numbers for subsequent pages.

```
options nonumber;
```

Create the report and use the default table options.

```
proc tabulate data=customer_response;
```

Specify the analysis variables. The VAR statement specifies that PROC TABULATE calculate statistics on the Source1, Source2, Source3, and Customer variables. The variable Customer must be in the variable list because it appears in the denominator definition.

```
var source1-source3 customer;
```

Define the table rows and columns. The TABLE statement creates a row for each source of the company name, a column for frequency counts, and a column for the percentages. Text in quotation marks supplies a heading for the corresponding row or column.

```
table source1='TV/Radio'
      source2='Newspaper'
      source3='Word of Mouth',
      (n='Count'*f=7. pctl<customer>='Percent'*f=pctfmt9.) ;
```

Specify the title and footnote. The macro variable NUM resolves to the number of respondents. The FOOTNOTE statement uses double rather than single quotation marks so that the macro variable will resolve.

```
title 'Source of Company Name';
footnote "Number of Respondents: &num";
run;
```

Reset the SAS system options. The FORMDLIM= option resets the page delimiter to a page eject. The NUMBER option resumes the display of page numbers on subsequent pages.

```
options formdlim='' number;
```

Output

Customer Survey Results: Spring 1996 1

Factors Influencing the Decision to Buy

	Count	Percent
Cost	87	72.5 %
Performance	62	51.6 %
Reliability	30	25.0 %
Sales Staff	120	100.0 %

Source of Company Name

	Count	Percent
TV/Radio	92	76.6 %
Newspaper	69	57.5 %
Word of Mouth	26	21.6 %

Number of Respondents: 120

Example 11: Reporting on Multiple-Choice Survey Data

Procedure features:

TABLE statement:

N statistic

Other features:

FORMAT procedure

TRANSPOSE procedure

Data set options:

RENAME=

This report of listener preferences shows how many listeners select each type of programming during each of seven time periods on a typical weekday. The data was collected by a survey, and the results were stored in a SAS data set. Although this data set contains all the information needed for this report, the information is not arranged in a way that PROC TABULATE can use.

To make this crosstabulation of time of day and choice of radio programming, you must have a data set that contains a variable for time of day and a variable for programming preference. PROC TRANSPOSE reshapes the data into a new data set that contains these variables. Once the data are in the appropriate form, PROC TABULATE creates the report.

Collecting the Data

Figure 43.15 on page 1339 shows the survey form that is used to collect data.

Figure 43.15 Completed Survey Form

phone_ _ _

LISTENER SURVEY

1. _____ What is your age?

2. _____ What is your gender?

3. _____ On the average WEEKDAY, how many hours do you listen to the radio?

4. _____ On the average WEEKEND-DAY, how many hours do you listen to the radio?

Use codes 1-8 for question 5. Use codes 0-8 for 6-19.

0 Do not listen at that time

1 Rock	5 Classical
2 Top 40	6 Easy Listening
3 Country	7 News/Information/Talk
4 Jazz	8 Other

5. _____ What style of music or radio programming do you most often listen to?

On a typical WEEKDAY, what kind of radio programming do you listen to	On a typical WEEKEND-DAY, what kind of radio programming do you listen to
6. _____ from 6-9 a.m.?	13. _____ from 6-9 a.m.?
7. _____ from 9 a.m. to noon?	14. _____ from 9 a.m. to noon?
8. _____ from noon to 1 p.m.?	15. _____ from noon to 1 p.m.?
9. _____ from 1-4 p.m.?	16. _____ from 1-4 p.m.?
10. _____ from 4-6 p.m.?	17. _____ from 4-6 p.m.?
11. _____ from 6-10 p.m.?	18. _____ from 6-10 p.m.?
12. _____ from 10 p.m. to 2 a.m.?	19. _____ from 10 p.m. to 2 a.m.?

An external file on page 1658 contains the raw data for the survey. Several lines from that file appear here.

```

967 32 f 5 3 5
7 5 5 5 7 0 0 0 8 7 0 0 8 0
781 30 f 2 3 5
5 0 0 0 5 0 0 0 4 7 5 0 0 0
859 39 f 1 0 5
1 0 0 0 1 0 0 0 0 0 0 0 0 0

. . . more data lines . . .

859 32 m .25 .25 1
1 0 0 0 0 0 0 0 1 0 0 0 0 0

```

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=132 pagesize=40;
```

Create the RADIO data set and specify the input file. RADIO contains data from a survey of 336 listeners. The data set contains information about listeners and their preferences in radio programming. The INFILE statement specifies the external file that contains the data. MISSEVER prevents the input pointer from going to the next record if it fails to find values in the current line for all variables that are listed in the INPUT statement.

```
data radio;
  infile 'input-file' missover;
```

Read the appropriate data line, assign a unique number to each respondent, and write an observation to RADIO. Each raw-data record contains two lines of information about each listener. The INPUT statement reads only the information that this example needs. The / line control skips the first line of information in each record. The rest of the INPUT statement reads Time1-Time7 from the beginning of the second line. These variables represent the listener's radio programming preference for each of seven time periods on weekdays (see Figure 43.15 on page 1339). The `listener=_n_` statement assigns a unique identifier to each listener. An observation is automatically written to RADIO at the end of each iteration.

```
  input /(Time1-Time7) ($1. +1);
  listener=_n_;
run;
```

Create the \$TIMEFMT. and \$PGMFMT. formats. PROC FORMAT creates formats for the time of day and the choice of programming.

```
proc format;
  value $timefmt 'Time1'='6-9 a.m.'
                'Time2'='9 a.m. to noon'
                'Time3'='noon to 1 p.m.'
                'Time4'='1-4 p.m.'
                'Time5'='4-6 p.m.'
                'Time6'='6-10 p.m.'
                'Time7'='10 p.m. to 2 a.m.'
                other='*** Data Entry Error ***';
  value $pgmfmt  '0'='Don't Listen'
                '1','2'='Rock and Top 40'
                '3'='Country'
                '4','5','6'='Jazz, Classical, and Easy Listening'
                '7'='News/ Information /Talk'
                '8'='Other'
                other='*** Data Entry Error ***';
run;
```

Reshape the data by transposing the RADIO data set. PROC TRANSPOSE creates RADIO_TRANSPOSED. This data set contains the variable Listener from the original data set. It also contains two transposed variables: Timespan and Choice. Timespan contains the names of the variables (Time1-Time7) from the input data set that are transposed to form observations in the output data set. Choice contains the values of these variables. (See “A Closer Look” on page 1342 for a complete explanation of the PROC TRANSPOSE step.)

```
proc transpose data=radio
               out=radio_transposed(rename=(coll=Choice))
               name=Timespan;
  by listener;
  var time1-time7;
```

Format the transposed variables. The FORMAT statement permanently associates these formats with the variables in the output data set.

```
format timespan $timefmt. choice $pgmfmt.;
run;
```

Create the report and specify the table options. The FORMAT= option specifies the default format for the values in each table cell.

```
proc tabulate data=radio_transposed format=12.;
```

Specify subgroups for the analysis. The CLASS statement identifies Timespan and Choice as class variables.

```
class timespan choice;
```

Define the table rows and columns. The TABLE statement creates a row for each formatted value of Timespan and a column for each formatted value of Choice. In each column are values for the N statistic. Text in quotation marks supplies headings for the corresponding rows or columns.

```
table timespan='Time of Day',
      choice='Choice of Radio Program'*n='Number of Listeners';
```

Specify the title.

```
title 'Listening Preferences on Weekdays';
run;
```

Output

Listening Preferences on Weekdays							1
Time of Day	Choice of Radio Program						
	Don't Listen	Rock and Top 40	Country	Jazz, Classical, and Easy Listening	News/Information/Talk	Other	
	Number of Listeners	Number of Listeners	Number of Listeners	Number of Listeners	Number of Listeners	Number of Listeners	
6-9 a.m.	34	143	7	39	96	17	
9 a.m. to noon	214	59	5	51	3	4	
noon to 1 p.m.	238	55	3	27	9	4	
1-4 p.m.	216	60	5	50	2	3	
4-6 p.m.	56	130	6	57	69	18	
6-10 p.m.	202	54	9	44	20	7	
10 p.m. to 2 a.m.	264	29	3	36	2	2	

A Closer Look

Reshape the data

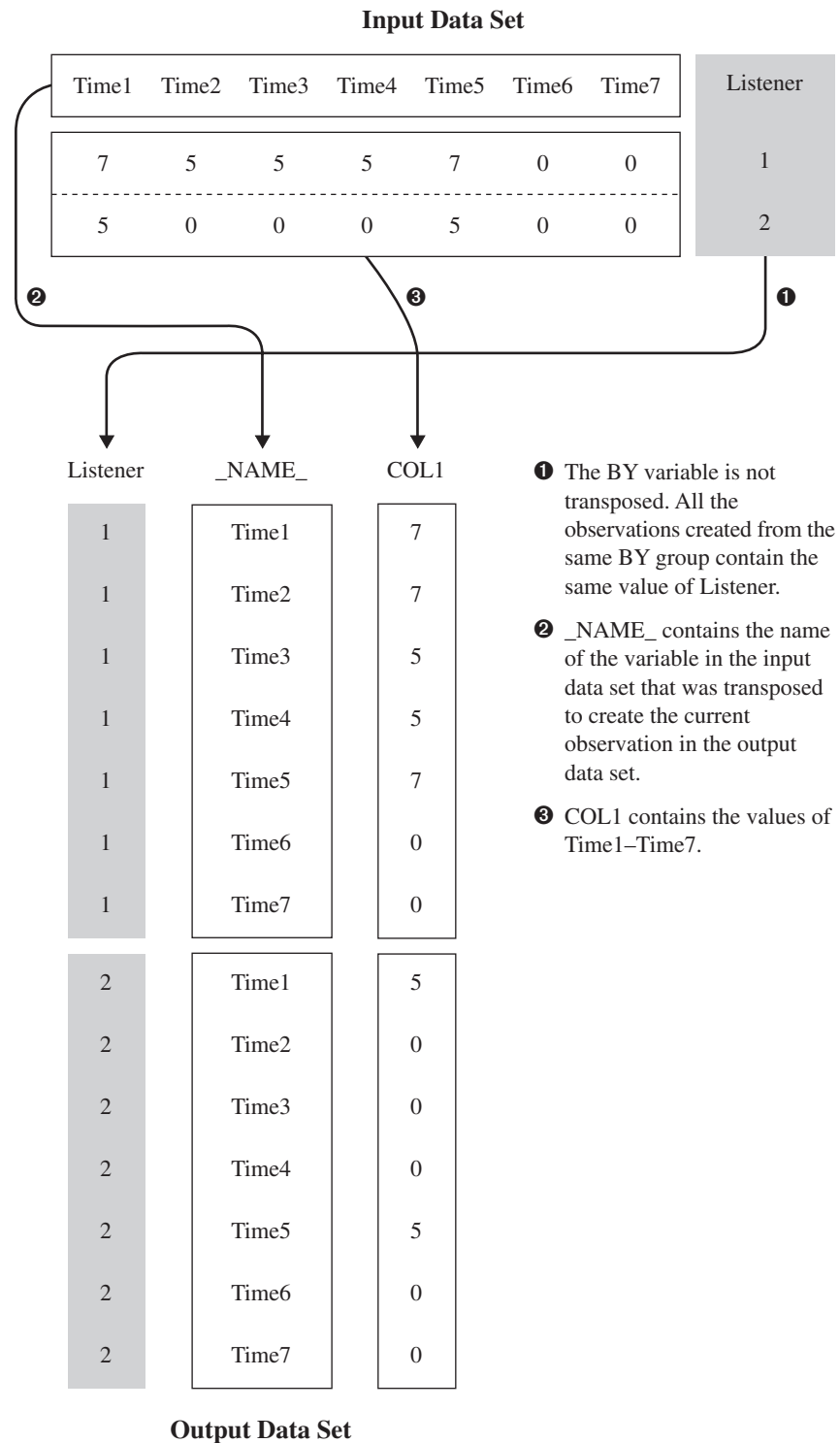
The original input data set has all the information that you need to make the crosstabular report, but PROC TABULATE cannot use the information in that form. PROC TRANSPOSE rearranges the data so that each observation in the new data set contains the variable Listener, a variable for time of day, and a variable for programming preference. Figure 43.16 on page 1343 illustrates the transposition. PROC TABULATE uses this new data set to create the crosstabular report.

PROC TRANSPOSE restructures data so that values that were stored in one observation are written to one variable. You can specify which variables you want to transpose. This section illustrates how PROC TRANSPOSE reshapes the data. The following section explains the PROC TRANSPOSE step in this example.

When you transpose with BY processing, as this example does, you create from each BY group one observation for each variable that you transpose. In this example, Listener is the BY variable. Each observation in the input data set is a BY group because the value of Listener is unique for each observation.

This example transposes seven variables, Time1 through Time7. Therefore, the output data set has seven observations from each BY group (each observation) in the input data set.

Figure 43.16 Transposing Two Observations



Understanding the PROC TRANSPOSE Step

Here is a detailed explanation of the PROC TRANSPOSE step that reshapes the data:

```
proc transpose data=radio ❶
    out=radio_transposed(rename=(col1=Choice)) ❷
```

```

                                name=Timespan;    ❸
    by listener;                ❹
    var time1-time7;            ❺
    format timespan $timefmt. choice $pgmfmt.;    ❻
run;

```

- ❶ The DATA= option specifies the input data set.
- ❷ The OUT= option specifies the output data set. The RENAME= data set option renames the transposed variable from COL1 (the default name) to Choice.
- ❸ The NAME= option specifies the name for the variable in the output data set that contains the name of the variable that is being transposed to create the current observation. By default, the name of this variable is _NAME_.
- ❹ The BY statement identifies Listener as the BY variable.
- ❺ The VAR statement identifies Time1 through Time7 as the variables to transpose.
- ❻ The FORMAT statement assigns formats to Timespan and Choice. The PROC TABULATE step that creates the report does not need to format Timespan and Choice because the formats are stored with these variables.

Example 12: Calculating Various Percentage Statistics

Procedure features:

PROC TABULATE statement options:

FORMAT=

TABLE statement:

ALL class variable
 COLPCTSUM statistic
 concatenation (blank) operator
 crossing (*) operator
 format modifiers
 grouping elements (parentheses) operator
 labels
 REPPCTSUM statistic
 ROWPCTSUM statistic
 variable list

TABLE statement options:

ROW=FLOAT
 RTS=

Other features: FORMAT procedure

This example shows how to use three percentage sum statistics: COLPCTSUM, REPPCTSUM, and ROWPCTSUM.

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=105 pagesize=60;
```

Create the FUNDRAIS data set. FUNDRAIS contains data on student sales during a school fund-raiser. A DATA step creates the data set.

```
data fundrais;
  length name $ 8 classrm $ 1;
  input @1 team $ @8 classrm $ @10 name $
        @19 pencils @23 tablets;
  sales=pencils + tablets;
  datalines;
BLUE   A ANN           4    8
RED     A MARY          5   10
GREEN  A JOHN           6    4
RED     A BOB           2    3
BLUE   B FRED           6    8
GREEN  B LOUISE        12    2
BLUE   B ANNETTE        .    9
RED     B HENRY         8   10
GREEN  A ANDREW         3    5
RED     A SAMUEL        12   10
BLUE   A LINDA          7   12
GREEN  A SARA           4    .
BLUE   B MARTIN         9   13
RED     B MATTHEW        7    6
GREEN  B BETH          15   10
RED     B LAURA         4    3
;
```

Create the PCTFMT. format. The FORMAT procedure creates a format for percentages. The PCTFMT. format writes all values with at least one digit, a blank, and a percent sign.

```
proc format;
  picture pctfmt low-high='009 %';
run;
```

Specify the title.

```
title "Fundraiser Sales";
```

Create the report and specify the table options. The FORMAT= option specifies up to seven digits as the default format for the value in each table cell.

```
proc tabulate format=7.;
```

Specify subgroups for the analysis. The CLASS statement identifies Team and Classrm as class variables.

```
class team classrm;
```

Specify the analysis variable. The VAR statement specifies that PROC TABULATE calculate statistics on the Sales variable.

```
var sales;
```

Define the table rows. The row dimension of the TABLE statement creates a row for each formatted value of Team. The last row of the report summarizes sales for all teams.

```
table (team all),
```

Define the table columns. The column dimension of the TABLE statement creates a column for each formatted value of Classrm. Crossed within each value of Classrm is the analysis variable (**sales**) with a blank label. Nested within each column are columns that summarize sales for the class.

- The first nested column, labeled **sum**, is the sum of sales for the row for the classroom.
- The second nested column, labeled **ColPctSum**, is the percentage of the sum of sales for the row for the classroom in relation to the sum of sales for all teams in the classroom.
- The third nested column, labeled **RowPctSum**, is the percentage of the sum of sales for the row for the classroom in relation to the sum of sales for the row for all classrooms.
- The fourth nested column, labeled **RepPctSum**, is the percentage of the sum of sales for the row for the classroom in relation to the sum of sales for all teams for all classrooms.

The last column of the report summarizes sales for the row for all classrooms.

```
classrm='Classroom'*sales=' '(sum
colpctsum*f=pctfmt9.
rowpctsum*f=pctfmt9.
reppctsum*f=pctfmt9.)
all*sales*sum=' '
```

Specify the row title space and eliminate blank row headings. RTS= provides 20 characters per line for row headings.

```
/rts=20;
```

```
run;
```

Output

Fundraiser Sales										1
team	Classroom									
	A				B				All	
	Sum	ColPctSum	RowPctSum	RepPctSum	Sum	ColPctSum	RowPctSum	RepPctSum	Sum	
BLUE	31	34 %	46 %	15 %	36	31 %	53 %	17 %	67	
GREEN	18	19 %	31 %	8 %	39	34 %	68 %	19 %	57	
RED	42	46 %	52 %	20 %	38	33 %	47 %	18 %	80	
All	91	100 %	44 %	44 %	113	100 %	55 %	55 %	204	

A Closer Look

Here are the percentage sum statistic calculations used to produce the output for the Blue Team in Classroom A:

COLPCTSUM=31/91*100=34%

ROWPCTSUM=31/67*100=46%

REPPCTSUM=31/204*100=15%

Similar calculations were used to produce the output for the remaining teams and classrooms.

Example 13: Using Denominator Definitions to Display Basic Frequency Counts and Percentages

Procedure features:

TABLE statement:

ALL class variable

denominator definitions (angle bracket operators)

N statistic

PCTN statistic

Other features:

FORMAT procedure

Crosstabulation tables (also called *contingency tables* and *stub-and-banner reports*) show combined frequency distributions for two or more variables. This table shows frequency counts for females and males within each of four job classes. The table also shows the percentage that each frequency count represents of

- ☐ the total women and men in that job class (row percentage)
- ☐ the total for that gender in all job classes (column percentage)
- ☐ the total for all employees.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= specifies the output line length, and PAGESIZE= specifies the number of lines on an output page.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the JOBCLASS data set. JOBCLASS contains encoded information about the gender and job class of employees at a fictitious company.

```
data jobclass;
  input Gender Occupation @@;
  datalines;
1 1  1 1  1 1  1 1  1 1  1 1
1 2  1 2  1 2  1 2  1 2  1 2
1 3  1 3  1 3  1 3  1 3  1 3
```

```

1 1 1 1 1 1 1 2 1 2 1 2 1 2
1 2 1 2 1 3 1 3 1 4 1 4 1 4
1 4 1 4 1 4 1 1 1 1 1 1 1 1
1 1 1 2 1 2 1 2 1 2 1 2 1 2
1 2 1 3 1 3 1 3 1 3 1 4 1 4
1 4 1 4 1 4 1 1 1 3 2 1 2 1
2 1 2 1 2 1 2 1 2 1 2 2 2 2
2 2 2 2 2 2 2 3 2 3 2 3 2 4
2 4 2 4 2 4 2 4 2 4 2 1 2 3
2 3 2 3 2 3 2 3 2 4 2 4 2 4
2 4 2 4 2 1 2 1 2 1 2 1 2 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 3 2 3 2 4 2 4 2 4 2 1 2 1
2 1 2 1 2 1 2 2 2 2 2 2 2 3
2 3 2 3 2 3 2 4
;

```

Create the GENDFMT. and OCCUPFMT. formats. PROC FORMAT creates formats for the variables Gender and Occupation.

```

proc format;
  value gendfmt 1='Female'
               2='Male'
               other='*** Data Entry Error ***';
  value occupfmt 1='Technical'
                2='Manager/Supervisor'
                3='Clerical'
                4='Administrative'
                other='*** Data Entry Error ***';
run;

```

Create the report and specify the table options. The FORMAT= option specifies the 8.2 format as the default format for the value in each table cell.

```
proc tabulate data=jobclass format=8.2;
```

Specify subgroups for the analysis. The CLASS statement identifies Gender and Occupation as class variables.

```
class gender occupation;
```

Define the table rows. The TABLE statement creates a set of rows for each formatted value of Occupation and for all jobs together. Text in quotation marks supplies a header for the corresponding row.

The asterisk in the row dimension indicates that the statistics that follow in parentheses are nested within the values of Occupation and All to form sets of rows. Each set of rows includes four statistics:

- N, the frequency count. The format modifier (F=9.) writes the values of N without the decimal places that the default format would use. It also extends the column width to nine characters so that the word **Employees** fits on one line.
- the percentage of the row total (row percent).
- the percentage of the column total (column percent).
- the overall percent. Text in quotation marks supplies the heading for the corresponding row. A comma separates the row definition from the column definition.

For detailed explanations of the structure of this table and of the use of denominator definitions, see “A Closer Look” on page 1350.

```
table (occupation='Job Class' all='All Jobs')
      *(n='Number of employees'*f=9.
      pctn<gender all>='Percent of row total'
      pctn<occupation all>='Percent of column total'
      pctn='Percent of total'),
```

Define the table columns and specify the amount of space for row headings. The column dimension creates a column for each formatted value of Gender and for all employees. Text in quotation marks supplies the heading for the corresponding column. The RTS= option provides 50 characters per line for row headings.

```
gender='Gender' all='All Employees' / rts=50;
```

Format the output. The FORMAT statement assigns formats to the variables Gender and Occupation.

```
format gender gendfmt. occupation occupfmt.;
```

Specify the titles.

```
title 'Gender Distribution';
title2 'within Job Classes';
run;
```

Output

Gender Distribution within Job Classes					1
		Gender		All	
		Female	Male	Employees	
Job Class					
Technical	Number of employees	16	18	34	
	Percent of row total	47.06	52.94	100.00	
	Percent of column total	26.23	29.03	27.64	
	Percent of total	13.01	14.63	27.64	
Manager/Supervisor	Number of employees	20	15	35	
	Percent of row total	57.14	42.86	100.00	
	Percent of column total	32.79	24.19	28.46	
	Percent of total	16.26	12.20	28.46	
Clerical	Number of employees	14	14	28	
	Percent of row total	50.00	50.00	100.00	
	Percent of column total	22.95	22.58	22.76	
	Percent of total	11.38	11.38	22.76	
Administrative	Number of employees	11	15	26	
	Percent of row total	42.31	57.69	100.00	
	Percent of column total	18.03	24.19	21.14	
	Percent of total	8.94	12.20	21.14	
All Jobs	Number of employees	61	62	123	
	Percent of row total	49.59	50.41	100.00	
	Percent of column total	100.00	100.00	100.00	
	Percent of total	49.59	50.41	100.00	

A Closer Look

The part of the TABLE statement that defines the rows of the table uses the PCTN statistic to calculate three different percentages.

In all calculations of PCTN, the numerator is N, the frequency count for one cell of the table. The denominator for each occurrence of PCTN is determined by the *denominator definition*. The denominator definition appears in angle brackets after the keyword PCTN. It is a list of one or more expressions. The list tells PROC TABULATE which frequency counts to sum for the denominator.

Analyzing the Structure of the Table

Taking a close look at the structure of the table helps you understand how PROC TABULATE uses the denominator definitions. The following simplified version of the TABLE statement clarifies the basic structure of the table:

```
table occupation='Job Class' all='All Jobs',
      gender='Gender' all='All Employees';
```

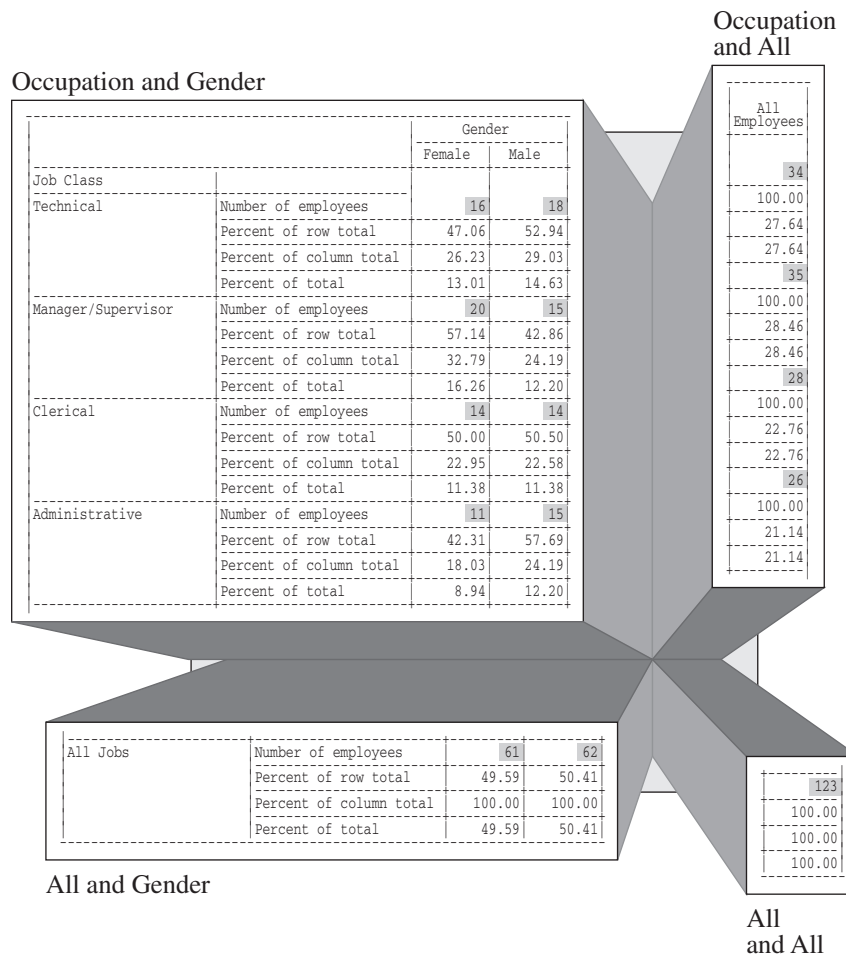
The table is a concatenation of four subtables. In this report, each subtable is a crossing of one class variable in the row dimension and one class variable in the column dimension. Each crossing establishes one or more categories. A *category* is a combination of unique values of class variables, such as **female**, **technical** or **all**, **clerical**. Table 43.8 on page 1351 describes each subtable.

Table 43.8 Contents of Subtables

Class variables contributing to the subtable	Description of frequency counts	Number of categories
Occupation and Gender	number of females in each job or number of males in each job	8
All and Gender	number of females or number of males	2
Occupation and All	number of people in each job	4
All and All	number of people in all jobs	1

Figure 43.17 on page 1352 highlights these subtables and the frequency counts for each category.

Figure 43.17 Illustration of the Four Subtables



Interpreting Denominator Definitions

The following fragment of the TABLE statement defines the denominator definitions for this report. The PCTN keyword and the denominator definitions are highlighted.

```
table (occupation='Job Class' all='All Jobs')
      *(n='Number of employees'*f=5.
        pctn<gender all>='Row percent'
        pctn<occupation all>='Column percent'
        pctn='Percent of total'),
```

Each use of PCTN nests a row of statistics within each value of Occupation and All. Each denominator definition tells PROC TABULATE which frequency counts to sum for the denominators in that row. This section explains how PROC TABULATE interprets these denominator definitions.

Row Percentages

The part of the TABLE statement that calculates the row percentages and that labels the row is

```
pctn<gender all>='Row percent'
```

Consider how PROC TABULATE interprets this denominator definition for each subtable.

Subtable 1: Occupation and Gender

Gender Distribution within Job Classes				
Job Class		Gender		Total
		Female	Male	
Technical	Number of employees	16	18	34
	Percent of job total	47.62	52.38	100.00
	Percent of female total	100.00	22.50	122.50
	Percent of male total	13.33	100.00	113.33
Manager/Supervisor	Number of employees	15	20	35
	Percent of job total	41.67	58.33	100.00
	Percent of female total	100.00	25.00	125.00
	Percent of male total	12.50	100.00	112.50
Craftsman	Number of employees	12	10	22
	Percent of job total	54.55	45.45	100.00
	Percent of female total	100.00	16.67	116.67
	Percent of male total	18.18	100.00	118.18
Administrative	Number of employees	10	15	25
	Percent of job total	40.00	60.00	100.00
	Percent of female total	100.00	20.00	120.00
	Percent of male total	16.67	100.00	116.67
All jobs		53	63	116
		Percent of female total	Percent of male total	
		47.62	52.38	100.00
		13.33	18.75	32.08
		22.50	29.17	51.67
		16.67	22.22	38.89
		20.00	27.78	47.78

PROC TABULATE looks at the first element in the denominator definition, Gender, and asks if Gender contributes to the subtable. Because Gender does contribute to the subtable, PROC TABULATE uses it as the denominator definition. This denominator definition tells PROC TABULATE to sum the frequency counts for all occurrences of Gender within the same value of Occupation.

For example, the denominator for the category **female, technical** is the sum of all frequency counts for all categories in this subtable for which the value of Occupation is **technical**. There are two such categories: **female, technical** and **male, technical**. The corresponding frequency counts are 16 and 18. Therefore, the denominator for this category is 16+18, or 34.

Subtable 2: All and Gender

Gender Distribution within Job Classes				
Job Class		Gender		Total
		Female	Male	
Technical	Number of employees	16	18	34
	Percent of job total	47.62	52.38	100.00
	Percent of female total	100.00	22.50	122.50
	Percent of male total	13.33	100.00	113.33
Manager/Supervisor	Number of employees	15	20	35
	Percent of job total	41.67	58.33	100.00
	Percent of female total	100.00	25.00	125.00
	Percent of male total	12.50	100.00	112.50
Craftsman	Number of employees	12	10	22
	Percent of job total	54.55	45.45	100.00
	Percent of female total	100.00	16.67	116.67
	Percent of male total	18.18	100.00	118.18
Administrative	Number of employees	10	15	25
	Percent of job total	40.00	60.00	100.00
	Percent of female total	100.00	20.00	120.00
	Percent of male total	16.67	100.00	116.67
All jobs		53	63	116
		Percent of female total	Percent of male total	
		47.62	52.38	100.00
		13.33	18.75	32.08
		22.50	29.17	51.67
		16.67	22.22	38.89
		20.00	27.78	47.78

PROC TABULATE looks at the first element in the denominator definition, Gender, and asks if Gender contributes to the subtable. Because Gender does contribute to the subtable, PROC TABULATE uses it as the denominator definition. This denominator definition tells PROC TABULATE to sum the frequency counts for all occurrences of Gender in the subtable.

For example, the denominator for the category **all, female** is the sum of the frequency counts for **all, female** and **all, male**. The corresponding frequency counts are 61 and 62. Therefore, the denominator for cells in this subtable is 61+62, or 123.

Subtable 3: Occupation and All

Occupation	Gender	Percent	N
Professional	Male	100	28
	Female	100	28
Managerial	Male	100	28
	Female	100	28
Clerical	Male	100	28
	Female	100	28
Laborer	Male	100	28
	Female	100	28

PROC TABULATE looks at the first element in the denominator definition, Gender, and asks if Gender contributes to the subtable. Because Gender does not contribute to the subtable, PROC TABULATE looks at the next element in the denominator definition, which is All. The variable All does contribute to this subtable, so PROC TABULATE uses it as the denominator definition. All is a reserved class variable with only one category. Therefore, this denominator definition tells PROC TABULATE to use the frequency count of All as the denominator.

For example, the denominator for the category **clerical**, **all** is the frequency count for that category, 28.

Note: In these table cells, because the numerator and the denominator are the same, the row percentages in this subtable are all 100. △

Subtable 4: All and All

Occupation	Gender	Percent	N
Professional	Male	100	28
	Female	100	28
Managerial	Male	100	28
	Female	100	28
Clerical	Male	100	28
	Female	100	28
Laborer	Male	100	28
	Female	100	28

PROC TABULATE looks at the first element in the denominator definition, Gender, and asks if Gender contributes to the subtable. Because Gender does not contribute to the subtable, PROC TABULATE looks at the next element in the denominator definition, which is All. The variable All does contribute to this subtable, so PROC TABULATE uses it as the denominator definition. All is a reserved class variable with only one category. Therefore, this denominator definition tells PROC TABULATE to use the frequency count of All as the denominator.

There is only one category in this subtable: **all**, **all**. The denominator for this category is 123.

Note: In this table cell, because the numerator and denominator are the same, the row percentage in this subtable is 100. △

Column Percentages

The part of the TABLE statement that calculates the column percentages and labels the row is

```
pctn<occupation all>='Column percent'
```

Consider how PROC TABULATE interprets this denominator definition for each subtable.

Subtable 1: Occupation and Gender

Gender: Occupation within Job Category				
		Gender		All
		Female	Male	Both
Job Class	Technical	18	48	66
	Manager/Supervisor	15	35	50
	Clerical	14	30	44
	Administrative	15	35	50
Total				
		62	128	190
		32.6%	67.4%	100%

PROC TABULATE looks at the first element in the denominator definition, Occupation, and asks if Occupation contributes to the subtable. Because Occupation does contribute to the subtable, PROC TABULATE uses it as the denominator definition. This denominator definition tells PROC TABULATE to sum the frequency counts for all occurrences of Occupation within the same value of Gender.

For example, the denominator for the category **manager/supervisor, male** is the sum of all frequency counts for all categories in this subtable for which the value of Gender is **male**. There are four such categories: **technical, male**; **manager/supervisor, male**; **clerical, male**; and **administrative, male**. The corresponding frequency counts are 18, 15, 14, and 15. Therefore, the denominator for this category is 18+15+14+15, or 62.

Subtable 2: All and Gender

Gender: Occupation within Job Category				
		Gender		All
		Female	Male	Both
Job Class	Technical	18	48	66
	Manager/Supervisor	15	35	50
	Clerical	14	30	44
	Administrative	15	35	50
Total				
		62	128	190
		32.6%	67.4%	100%

PROC TABULATE looks at the first element in the denominator definition, Occupation, and asks if Occupation contributes to the subtable. Because Occupation does not contribute to the subtable, PROC TABULATE looks at the next element in the denominator definition, which is All. Because the variable All does contribute to this subtable, PROC TABULATE uses it as the denominator definition. All is a reserved class variable with only one category. Therefore, this denominator definition tells PROC TABULATE to use the frequency count for All as the denominator.

For example, the denominator for the category **all, female** is the frequency count for that category, 61.

Note: In these table cells, because the numerator and denominator are the same, the column percentages in this subtable are all 100. Δ

Subtable 3: Occupation and All

Table Distribution within the Tables				
Job Class	Occupation	Frequency		Percent of Total
		Count	Percent	
Technical	Percent of all occupations	34	28.33	28.33
	Percent of new hires	47	39.17	39.17
	Percent of current hires	35	29.17	29.17
	Percent of total	116	96.67	96.67
Manager/Supervisor	Percent of all occupations	35	29.17	29.17
	Percent of new hires	51	42.50	42.50
	Percent of current hires	30	25.00	25.00
	Percent of total	116	96.67	96.67
Clerical	Percent of all occupations	28	23.33	23.33
	Percent of new hires	42	35.00	35.00
	Percent of current hires	25	20.83	20.83
	Percent of total	95	79.17	79.17
Administrative	Percent of all occupations	26	21.67	21.67
	Percent of new hires	40	33.33	33.33
	Percent of current hires	34	28.33	28.33
	Percent of total	100	83.33	83.33
All Data		123	100.00	100.00

PROC TABULATE looks at the first element in the denominator definition, Occupation, and asks if Occupation contributes to the subtable. Because Occupation does contribute to the subtable, PROC TABULATE uses it as the denominator definition. This denominator definition tells PROC TABULATE to sum the frequency counts for all occurrences of Occupation in the subtable.

For example, the denominator for the category **technical, all** is the sum of the frequency counts for **technical, all**; **manager/supervisor, all**; **clerical, all**; and **administrative, all**. The corresponding frequency counts are 34, 35, 28, and 26. Therefore, the denominator for this category is 34+35+28+26, or 123.

Subtable 4: All and All

Table Distribution within the Tables				
Job Class	All	Frequency		Percent of Total
		Count	Percent	
Technical	Percent of all occupations	34	28.33	28.33
	Percent of new hires	47	39.17	39.17
	Percent of current hires	35	29.17	29.17
	Percent of total	116	96.67	96.67
Manager/Supervisor	Percent of all occupations	35	29.17	29.17
	Percent of new hires	51	42.50	42.50
	Percent of current hires	30	25.00	25.00
	Percent of total	116	96.67	96.67
Clerical	Percent of all occupations	28	23.33	23.33
	Percent of new hires	42	35.00	35.00
	Percent of current hires	25	20.83	20.83
	Percent of total	95	79.17	79.17
Administrative	Percent of all occupations	26	21.67	21.67
	Percent of new hires	40	33.33	33.33
	Percent of current hires	34	28.33	28.33
	Percent of total	100	83.33	83.33
All Data		123	100.00	100.00

PROC TABULATE looks at the first element in the denominator definition, Occupation, and asks if Occupation contributes to the subtable. Because Occupation does not contribute to the subtable, PROC TABULATE looks at the next element in the denominator definition, which is All. Because the variable All does contribute to this subtable, PROC TABULATE uses it as the denominator definition. All is a reserved class variable with only one category. Therefore, this denominator definition tells PROC TABULATE to use the frequency count of All as the denominator.

There is only one category in this subtable: **all, all**. The frequency count for this category is 123.

Note: In this calculation, because the numerator and denominator are the same, the column percentage in this subtable is 100. △

Total Percentages

The part of the TABLE statement that calculates the total percentages and labels the row is

```
pctn='Total percent'
```

If you do not specify a denominator definition, then PROC TABULATE obtains the denominator for a cell by totaling all the frequency counts in the subtable. Table 43.9 on page 1357 summarizes the process for all subtables in this example.

Table 43.9 Denominators for Total Percentages

Class variables contributing to the subtable	Frequency counts	Total
Occupat and Gender	16, 18, 20, 15 14, 14, 11, 15	123
Occupat and All	34, 35, 28, 26	123
Gender and All	61, 62	123
All and All	123	123

Consequently, the denominator for total percentages is always 123.

Example 14: Specifying Style Elements for ODS Output

Procedure features:

STYLE= option in
 PROC TABULATE statement
 CLASSLEV statement
 KEYWORD statement
 TABLE statement
 VAR statement

Other features:

ODS HTML statement
 ODS PDF statement
 ODS RTF statement

Data set: ENERGY on page 1310

Formats: REGFMT, DIVFMT, and USETYPE. on page 1311

This example creates HTML, RTF, and PDF files and specifies style elements for various table regions.

Program

Set the SAS system options. The NODATE option suppresses the display of the date and time in the output. PAGENO= specifies the starting page number. LINESIZE= and PAGESIZE= are not set for this example because they have no effect on HTML, RTF, and Printer output.

```
options nodate pageno=1;
```

Specify the ODS output filenames. By opening multiple ODS destinations, you can produce multiple output files in a single execution. The ODS HTML statement produces output that is written in HTML. The ODS PDF statement produces output in Portable Document Format (PDF). The ODS RTF statement produces output in Rich Text Format (RTF). The output from PROC TABULATE goes to each of these files.

```
ods html body='external-HTML-file';
ods pdf file='external-PDF-file';
```

```
ods rtf file='external-RTF-file';
```

Specify the table options. The STYLE= option in the PROC TABULATE statement specifies the style element for the data cells of the table.

```
proc tabulate data=energy style=[font_weight=bold];
```

Specify subgroups for the analysis. The STYLE= option in the CLASS statement specifies the style element for the class variable name headings.

```
class region division type / style=[just=center];
```

Specify the style attributes for the class variable value headings. The STYLE= option in the CLASSLEV statement specifies the style element for the class variable level value headings.

```
classlev region division type / style=[just=left];
```

Specify the analysis variable and its style attributes. The STYLE= option in the VAR statement specifies a style element for the variable name headings.

```
var expenditures / style=[font_size=3];
```

Specify the style attributes for keywords, and label the “all” keyword. The STYLE= option in the KEYWORD statement specifies a style element for keywords. The KEYLABEL statement assigns a label to the keyword.

```
keyword all sum / style=[font_width=wide];
keylabel all="Total";
```

Define the table rows and columns and their style attributes. The STYLE= option in the dimension expression overrides any other STYLE= specifications in PROC TABULATE that specify attributes for table cells. The STYLE= option after the slash (/) specifies attributes for parts of the table other than table cells.

```
table (region all)*(division all*[style=[background=yellow]]),
      (type all)*(expenditures*f=dollar10.) /
      style=[bordercolor=blue]
```

Specify the style attributes for cells with missing values. The STYLE= option in the MISSTEXT option of the TABLE statement specifies a style element to use for the text in table cells that contain missing values.

```
misstext=[label="Missing" style=[font_weight=light]]
```

Specify the style attributes for the box above the row titles. The STYLE= option in the BOX option of the TABLE statement specifies a style element to use for text in the box above the row titles.

```
box=[label="Region by Division by Type"
      style=[font_style=italic]];
```

Format the class variable values. The FORMAT statement assigns formats to Region, Division, and Type.

```
format region regfmt. division divfmt. type usetype.;
```

Specify the titles.


```

title 'Energy Expenditures';
title2 '(millions of dollars)';
run;

```

Close the ODS destinations.

```

ods html close;
ods pdf close;
ods rtf close;

```

HTML Output

<i>Energy Expenditures (millions of dollars)</i>				
<i>Region by Division by Type</i>		Type		Total
		Residential Customers	Business Customers	
		Expenditures	Expenditures	Expenditures
		Sum	Sum	Sum
Region	Division			
Northeast	New England	\$7,477	\$5,129	\$12,606
	Middle Atlantic	\$19,379	\$15,078	\$34,457
	Total	\$26,856	\$20,207	\$47,063
West	Division			
	Mountain	\$5,476	\$4,729	\$10,205
	Pacific	\$13,959	\$12,619	\$26,578
	Total	\$19,435	\$17,348	\$36,783
Total	Division			
	New England	\$7,477	\$5,129	\$12,606
	Middle Atlantic	\$19,379	\$15,078	\$34,457
	Mountain	\$5,476	\$4,729	\$10,205
	Pacific	\$13,959	\$12,619	\$26,578
	Total	\$46,291	\$37,555	\$83,846

PDF Output

Energy Expenditures (millions of dollars)

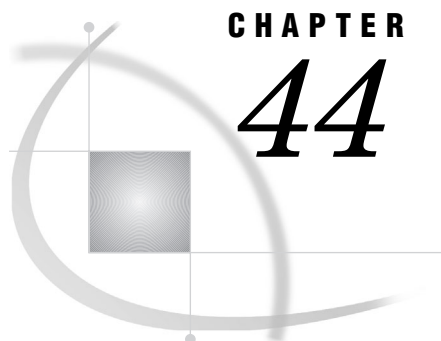
<i>Region by Division by Type</i>		Type		Total
		Residential Customers	Business Customers	
		Expenditures	Expenditures	
		Sum	Sum	
Region	Division			
Northeast	New England	\$7,477	\$5,129	\$12,606
	Middle Atlantic	\$19,379	\$15,078	\$34,457
	Total	\$26,856	\$20,207	\$47,063
West	Division			
	Mountain	\$5,476	\$4,729	\$10,205
	Pacific	\$13,959	\$12,619	\$26,578
	Total	\$19,435	\$17,348	\$36,783
Total	Division			
	New England	\$7,477	\$5,129	\$12,606
	Middle Atlantic	\$19,379	\$15,078	\$34,457
	Mountain	\$5,476	\$4,729	\$10,205
	Pacific	\$13,959	\$12,619	\$26,578
	Total	\$46,291	\$37,555	\$83,846

RTF Output*Energy Expenditures
(millions of dollars)*

<i>Region by Division by Type</i>		Type		Total
		Residential Customers	Business Customers	
		Expenditures	Expenditures	
		Sum	Sum	
Region	Division			
Northeast	New England	\$7,477	\$5,129	\$12,606
	Middle Atlantic	\$19,379	\$15,078	\$34,457
	Total	\$26,856	\$20,207	\$47,063
West	Division			
	Mountain	\$5,476	\$4,729	\$10,205
	Pacific	\$13,959	\$12,619	\$26,578
	Total	\$19,435	\$17,348	\$36,783
Total	Division			
	New England	\$7,477	\$5,129	\$12,606
	Middle Atlantic	\$19,379	\$15,078	\$34,457
	Mountain	\$5,476	\$4,729	\$10,205
	Pacific	\$13,959	\$12,619	\$26,578
	Total	\$46,291	\$37,555	\$83,846

References

Jain, Raj and Chlamtac, Imrich (1985), "The P² Algorithm for Dynamic Calculation of Quantiles and Histograms without Storing Observations," *Communications of the Association of Computing Machinery*, 28:10.

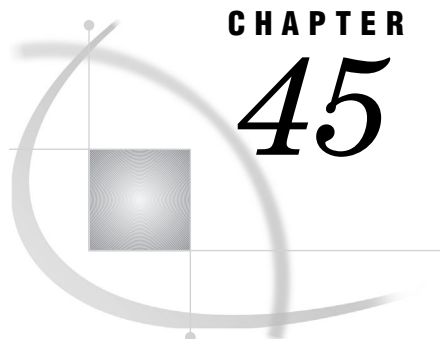


The TEMPLATE Procedure

Information about the TEMPLATE Procedure 1363

Information about the TEMPLATE Procedure

See: For complete documentation of the TEMPLATE procedure, see *SAS Output Delivery System User's Guide*.



CHAPTER 45

The TIMEPLOT Procedure

<i>Overview: TIMEPLOT Procedure</i>	1365
<i>Syntax: TIMEPLOT Procedure</i>	1367
<i>PROC TIMEPLOT Statement</i>	1368
<i>BY Statement</i>	1369
<i>CLASS Statement</i>	1369
<i>ID Statement</i>	1370
<i>PLOT Statement</i>	1371
<i>Results: TIMEPLOT Procedure</i>	1375
<i>Data Considerations</i>	1375
<i>Procedure Output</i>	1375
<i>Page Layout</i>	1375
<i>Contents of the Listing</i>	1376
<i>Missing Values</i>	1376
<i>Examples: TIMEPLOT Procedure</i>	1376
<i>Example 1: Plotting a Single Variable</i>	1376
<i>Example 2: Customizing an Axis and a Plotting Symbol</i>	1378
<i>Example 3: Using a Variable for a Plotting Symbol</i>	1380
<i>Example 4: Superimposing Two Plots</i>	1382
<i>Example 5: Showing Multiple Observations on One Line of a Plot</i>	1384

Overview: TIMEPLOT Procedure

The TIMEPLOT procedure plots one or more variables over time intervals. A listing of variable values accompanies the plot. Although the plot and the listing are similar to those produced by the PLOT and PRINT procedures, PROC TIMEPLOT output has these distinctive features:

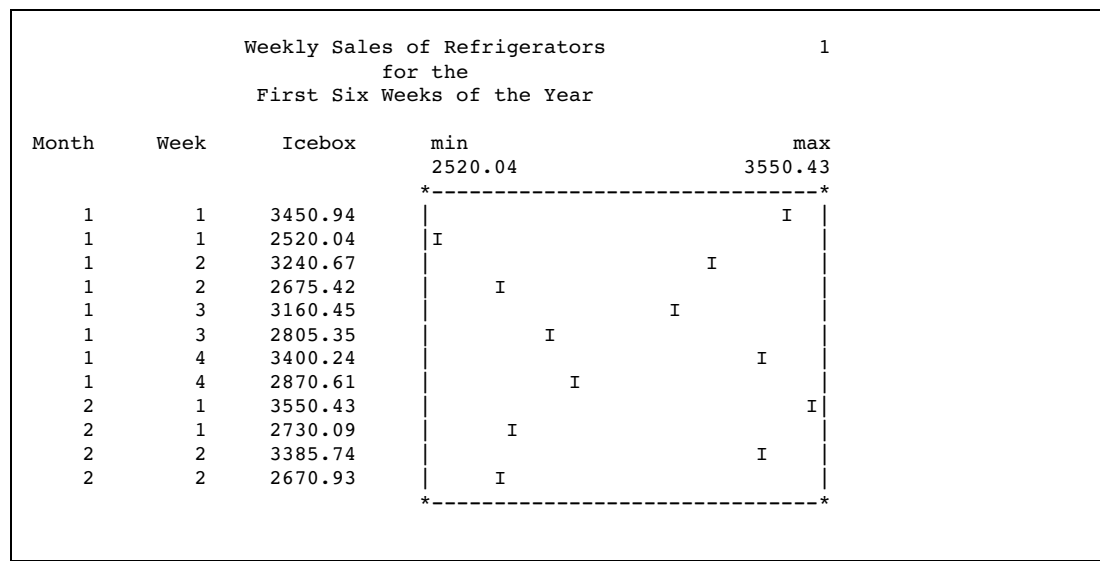
- The vertical axis always represents the sequence of observations in the data set; thus, if the observations are in order of date or time, the vertical axis represents the passage of time.
- The horizontal axis represents the values of the variable that you are examining. Like PROC PLOT, PROC TIMEPLOT can overlay multiple plots on one set of axes so that each line of the plot can contain values for more than one variable.
- A plot produced by PROC TIMEPLOT may occupy more than one page.
- Each observation appears sequentially on a separate line of the plot; PROC TIMEPLOT does not hide observations as PROC PLOT sometimes does.
- The listing of the plotted values may include variables that do not appear in the plot.

Output 45.1 on page 1366 illustrates a simple report that you can produce with PROC TIMEPLOT. This report shows sales of refrigerators for two sales representatives during the first six weeks of the year. The statements that produce the output follow. A DATA step on page 1377 creates the data set SALES.

```
options linesize=64 pagesize=60 nodate
        pageno=1;

proc timeplot data=sales;
  plot icebox;
  id month week;
  title 'Weekly Sales of Refrigerators';
  title2 'for the';
  title3 'First Six Weeks of the Year';
run;
```

Output 45.1 Simple Report Created with PROC TIMEPLOT



Output 45.2 on page 1366 is a more complicated report of the same data set that is used to create Output 45.1 on page 1366. The statements that create this report

- ☐ create one plot for the sale of refrigerators and one for the sale of stoves
- ☐ plot sales for both sales representatives on the same line
- ☐ identify points on the plots by the first letter of the sales representative's last name
- ☐ control the size of the horizontal axis
- ☐ control formats and labels.

For an explanation of the program that produces this report, see Example 5 on page 1384.

Output 45.2 More Complex Report Created with PROC TIMEPLOT

Weekly Appliance Sales for the First Quarter						1
Month	Week	Seller :Kreitz Stove	Seller :LeGrange Stove	min \$184.24	max \$2,910.37	

January	1	\$1,312.61	\$728.13	L	K	
January	2	\$222.35	\$184.24	!		
January	3	\$2,263.33	\$267.35	L		K
January	4	\$1,787.45	\$274.51	L	K	
February	1	\$2,910.37	\$397.98	L		K
February	2	\$819.69	\$2,242.24		K	L

Weekly Appliance Sales for the First Quarter						2
Month	Week	Kreitz Icebox	LeGrange Icebox	min \$2,520.04	max \$3,550.43	

January	1	\$3,450.94	\$2,520.04	L		K
January	2	\$3,240.67	\$2,675.42	L		K
January	3	\$3,160.45	\$2,805.35		L	K
January	4	\$3,400.24	\$2,870.61		L	K
February	1	\$3,550.43	\$2,730.09	L		K
February	2	\$3,385.74	\$2,670.93	L		K

Syntax: TIMEPLOT Procedure**Requirements:** At least one PLOT statement**Tip:** Supports the Output Delivery System. See “Output Delivery System” on page 32 for details.**Reminder:** You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, “Statements with the Same Function in Multiple Procedures,” on page 53 for details. You can also use any global statements as well. See “Global Statements” on page 18 for a list.**PROC TIMEPLOT** *<option(s)>*;
BY *<DESCENDING> variable-1*
<...<DESCENDING> variable-n>
<NOTSORTED>;
CLASS *variable(s)*;**ID** *variable(s)*;**PLOT** *plot-request(s)/option(s)*;

To do this	Use this statement
Produce a separate plot for each BY group	BY
Group data according to the values of the class variables	CLASS
Print in the listing the values of the variables that you identify	ID
Specify the plots to produce	PLOT

PROC TIMEPLOT Statement

PROC TIMEPLOT *<option(s)>*;

Options

DATA=SAS-*data-set*

identifies the input data set.

MAXDEC=number

specifies the maximum number of decimal places to print in the listing.

Interaction: A decimal specification in a format overrides a MAXDEC= specification.

Default: 2

Range: 0-12

Featured in: Example 4 on page 1382

SPLIT='split-character'

specifies a split character, which controls line breaks in column headings. It also specifies that labels be used as column headings. PROC TIMEPLOT breaks a column heading when it reaches the split character and continues the heading on the next line. Unless the split character is a blank, it is not part of the column heading. Each occurrence of the split character counts toward the 256-character maximum for a label.

Alias: S=

Default: blank (' ')

Note: Column headings can occupy up to three lines. If the column label can be split into more lines than this fixed number, then the split character is used only as a recommendation on how to split the label. \triangle

UNIFORM

uniformly scales the horizontal axis across all BY groups. By default, PROC TIMEPLOT separately determines the scale of the axis for each BY group.

Interaction: UNIFORM also affects the calculation of means for reference lines (see REF= on page 1374).

BY Statement

Produces a separate plot for each BY group.

Main discussion: “BY” on page 54

```
BY <DESCENDING> variable-1
      <...<DESCENDING> variable-n>
      <NOTSORTED>;
```

Required Arguments

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, the observations in the data set must either be sorted by all the variables that you specify, or they must be indexed appropriately. These variables are called *BY variables*.

Options

DESCENDING

specifies that the data set is sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The data are grouped in another way, for example, chronological order.

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, the procedure treats each contiguous set as a separate BY group.

CLASS Statement

Groups data according to the values of the class variables.

Tip: PROC TIMEPLOT uses the formatted values of the CLASS variables to form classes. Thus, if a format groups the values, the procedure uses those groups.

Featured in: Example 5 on page 1384

```
CLASS variable(s);
```

Required Arguments

variable(s)

specifies one or more variables that the procedure uses to group the data. Variables in a CLASS statement are called *class variables*. Class variables can be numeric or character. Class variables can have continuous values, but they typically have a few discrete values that define the classifications of the variable. You do not have to sort the data by class variables.

The values of the class variables appear in the listing. PROC TIMEPLOT prints and plots one line each time the combination of values of the class variables changes. Therefore, the output typically is more meaningful if you sort or group the data according to values of the class variables.

Using Multiple CLASS Statements

You can use any number of CLASS statements. If you use more than one CLASS statement, PROC TIMEPLOT simply concatenates all variables from all of the CLASS statements. The following form of the CLASS statement includes three variables:

```
CLASS variable-1 variable-2 variable-3;
```

It has the same effect as this form:

```
CLASS variable-1;
```

```
CLASS variable-2;
```

```
CLASS variable-3;
```

Using a Symbol Variable

Normally, you use the CLASS statement with a symbol variable (see the discussion of plot requests on page 1372). In this case, the listing of the plot variable contains a column for each value of the symbol variable, and each row of the plot contains a point for each value of the symbol variable. The plotting symbol is the first character of the formatted value of the symbol variable. If more than one observation within a class has the same value of a symbol variable, PROC TIMEPLOT plots and prints only the first occurrence of that value and writes a warning message to the SAS log.

ID Statement

Prints in the listing the values of the variables that you identify.

Featured in: Example 1 on page 1376

ID *variable(s)*;

Required Arguments

variable(s)

identifies one or more *ID variables* to print in the listing.

PLOT Statement

Specifies the plots to produce.

Tip: Each PLOT statement produces a separate plot.

PLOT *plot-request(s)/option(s);*

Table 45.1 on page 1371 summarizes the options that are available in the PLOT statement.

Table 45.1 Summary of Options for the PLOT Statement

To do this	Use this option
Customize the axis	
Specify the range of values to plot on the horizontal axis, as well as the interval represented by each print position on the horizontal axis	AXIS=
Order the values on the horizontal axis with the largest value in the leftmost position	REVERSE
Control the appearance of the plot	
Connect the leftmost plotting symbol to the rightmost plotting symbol with a line of hyphens (-)	HILOC
Connect the leftmost and rightmost symbols on each line of the plot with a line of hyphens (-) regardless of whether the symbols are reference symbols or plotting symbols	JOINREF
Suppress the name of the symbol variable in column headings when you use a CLASS statement	NOSYMNAM
Suppress the listing of the values of the variables that appear in the PLOT statement	NPP
Specify the number of print positions to use for the horizontal axis	POS=
Create and customize a reference line	
Draw lines on the plot that are perpendicular to the specified values on the horizontal axis	REF=
Specify the character for drawing reference lines	REFCHAR=
Display multiple plots on the same set of axes	
Plot all requests in one PLOT statement on one set of axes	OVERLAY
Specify the character to print if multiple plotting symbols coincide	OVPCHAR=

Required Arguments

plot-request(s)

specifies the variable or variables to plot and, optionally, the plotting symbol to use. By default, each plot request produces a separate plot.

A plot request can have the following forms. You can mix different forms of requests in one PLOT statement (see Example 4 on page 1382).

variable(s)

identifies one or more numeric variables to plot. PROC TIMEPLOT uses the first character of the variable name as the plotting symbol.

Featured in: Example 1 on page 1376

(variable(s))='plotting-symbol'

identifies one or more numeric variables to plot and specifies the plotting symbol to use for all variables in the list. You can omit the parentheses if you use only one variable.

Featured in: Example 2 on page 1378

(variable(s))=symbol-variable

identifies one or more numeric variables to plot and specifies a *symbol variable*. PROC TIMEPLOT uses the first nonblank character of the formatted value of the symbol variable as the plotting symbol for all variables in the list. The plotting symbol changes from one observation to the next if the value of the symbol variable changes. You can omit the parentheses if you use only one variable.

Featured in: Example 3 on page 1380

Options

AXIS=axis-specification

specifies the range of values to plot on the horizontal axis, as well as the interval represented by each print position on the axis. PROC TIMEPLOT labels the first and last ends of the axis, if space permits.

- For numeric values, *axis-specification* can be one of the following or a combination of both:

n < . . n >

n TO n <BY increment>

The values must be in either ascending or descending order. Use a negative value for *increment* to specify descending order. The specified values are spaced evenly along the horizontal axis even if the values are not uniformly distributed. Numeric values can be specified in the following ways:

Specification	Comments
axis=1 2 10	Values are 1, 2, and 10.
axis=10 to 100 by 5	Values appear in increments of 5, starting at 10 and ending at 100.
axis=12 10 to 100 by 5	A combination of the two previous forms of specification.

- For axis variables that contain datetime values, *axis-specification* is either an explicit list of values or a starting and an ending value with an increment specified:

'date-time-value'i < . . 'date-time-value'i >

HILOC

connects the leftmost plotting symbol to the rightmost plotting symbol with a line of hyphens (-).

Interactions: If you specify JOINREF, PROC TIMEPLOT ignores HILOC.

JOINREF

connects the leftmost and rightmost symbols on each line of the plot with a line of hyphens (-), regardless of whether the symbols are reference symbols or plotting symbols. However, if a line contains only reference symbols, PROC TIMEPLOT does not connect the symbols.

Featured in: Example 3 on page 1380

NOSYMNAM

suppresses the name of the symbol variable in column headings when you use a CLASS statement. If you use NOSYMNAM, only the value of the symbol variable appears in the column heading.

Featured in: Example 5 on page 1384

NPP

suppresses the listing of the values of the variables that appear in the PLOT statement.

Featured in: Example 3 on page 1380

OVERLAY

plots all requests in one PLOT statement on one set of axes. Otherwise, PROC TIMEPLOT produces a separate plot for each plot request.

Featured in: Example 4 on page 1382

OVPCHAR=*'character'*

specifies the character to print if multiple plotting symbols coincide. If a plotting symbol and a character in a reference line coincide, PROC TIMEPLOT prints the plotting symbol.

Default: at sign (@)

Featured in: Example 5 on page 1384

POS=*print-positions-for-plot*

specifies the number of print positions to use for the horizontal axis.

Default: If you omit both POS= and AXIS=, PROC TIMEPLOT initially assumes that POS=20. However, if space permits, this value increases so that the plot fills the available space.

Interaction: If you specify POS=0 and AXIS=, the plot fills the available space. POS= overrides an interval set with AXIS= (see the discussion of AXIS= on page 1372).

See also: "Page Layout" on page 1375

Featured in: Example 1 on page 1376

REF=*reference-value(s)*

draws lines on the plot that are perpendicular to the specified values on the horizontal axis. The values for *reference-value(s)* may be constants, or you may use the form

MEAN(*variable(s)*)

If you use this form of REF=, PROC TIMEPLOT evaluates the mean for each variable that you list and draws a reference line for each mean.

Interaction: If you use the UNIFORM option in the PROC TIMEPLOT statement, the procedure calculates the mean values for the variables over all observations for

all BY groups. If you do not use UNIFORM, the procedure calculates the mean for each variable for each BY group.

Interaction: If a plotting symbol and a reference character coincide, PROC TIMEPLOT prints the plotting symbol.

Featured in: Example 3 on page 1380 and Example 4 on page 1382

REFCHAR='character'

specifies the character for drawing reference lines.

Default: vertical bar (|)

Interaction: If you are using the JOINREF or HILOC option, do not specify a value for REFCHAR= that is the same as a plotting symbol because PROC TIMEPLOT will interpret the plotting symbols as reference characters and will not connect the symbols as you expect.

Featured in: Example 3 on page 1380

REVERSE

orders the values on the horizontal axis with the largest value in the leftmost position.

Featured in: Example 4 on page 1382

Results: TIMEPLOT Procedure

Data Considerations

The input data set usually contains a date variable to use as either a class or an ID variable. Although PROC TIMEPLOT does not require an input data set sorted by date, the output is usually more meaningful if the observations are in chronological order. In addition, if you use a CLASS statement, the output is more meaningful if the input data set groups observations according to combinations of class variable values. (For more information see “CLASS Statement” on page 1369.)

Procedure Output

Page Layout

For each plot request, PROC TIMEPLOT prints a listing and a plot. PROC TIMEPLOT determines the arrangement of the page as follows:

- ☐ If you use POS=, the procedure
 - ☐ determines the size of the plot from the POS= value
 - ☐ determines the space for the listing from the width of the columns of printed values, equally spaced and with a maximum of five positions between columns
 - ☐ centers the output on the page.
- ☐ If you omit POS=, the procedure
 - ☐ determines the width of the plot from the value of the AXIS= option
 - ☐ expands the listing to fill the rest of the page.

If there is not enough room to print the listing and the plot for a particular plot request, PROC TIMEPLOT produces no output and writes the following error message to the SAS log:

```
ERROR: Too many variables/symbol values
       to print.
```

The error does not affect other plot requests.

Contents of the Listing

The listing in the output contains different information depending on whether or not you use a CLASS statement. If you do not use a CLASS statement (see Example 1 on page 1376), PROC TIMEPLOT prints (and plots) each observation on a separate line. If you do use a CLASS statement, the form of the output varies depending on whether or not you specify a symbol variable (see “Using a Symbol Variable” on page 1370).

Missing Values

Four types of variables can appear in the listing from PROC TIMEPLOT: plot variables, ID variables, class variables, and symbol variables (as part of some column headers). Plot variables and symbol variables can also appear in the plot.

Observations with missing values of a class variable form a class of observations.

In the listing, missing values appear as a period (.), a blank, or a special missing value (the letters A through Z and the underscore (_) character).

In the plot, PROC TIMEPLOT handles different variables in different ways:

- An observation or class of observations with a missing value of the plot variable does not appear in the plot.
- If you use a symbol variable (see the discussion of plot requests on page 1372), PROC TIMEPLOT uses a period (.) as the symbol variable on the plot for all observations with a missing value of the symbol variable.

Examples: TIMEPLOT Procedure

Example 1: Plotting a Single Variable

Procedure features:

ID statement

PLOT statement arguments:

simple plot request

POS=

This example

- uses a single PLOT statement to plot sales of refrigerators
- specifies the number of print positions to use for the horizontal axis of the plot
- provides context for the points in the plot by printing in the listing the values of two variables that are not in the plot.

Program

```
options nodate pageno=1 linesize=80 pagesize=60;
```

The data set SALES contains weekly information on the sales of refrigerators and stoves by two sales representatives.

```
data sales;
  input Month Week Seller $ Icebox Stove;
  datalines;
1 1 Kreitz 3450.94 1312.61
1 1 LeGrange 2520.04 728.13
1 2 Kreitz 3240.67 222.35
1 2 LeGrange 2675.42 184.24
1 3 Kreitz 3160.45 2263.33
1 3 LeGrange 2805.35 267.35
1 4 Kreitz 3400.24 1787.45
1 4 LeGrange 2870.61 274.51
2 1 Kreitz 3550.43 2910.37
2 1 LeGrange 2730.09 397.98
2 2 Kreitz 3385.74 819.69
2 2 LeGrange 2670.93 2242.24
;
```

The plot variable, Icebox, appears in both the listing and the output. POS= provides 50 print positions for the horizontal axis.

```
proc timeplot data=sales;
  plot icebox / pos=50;
```

The values of the ID variables, Month and Week, appear in the listing.

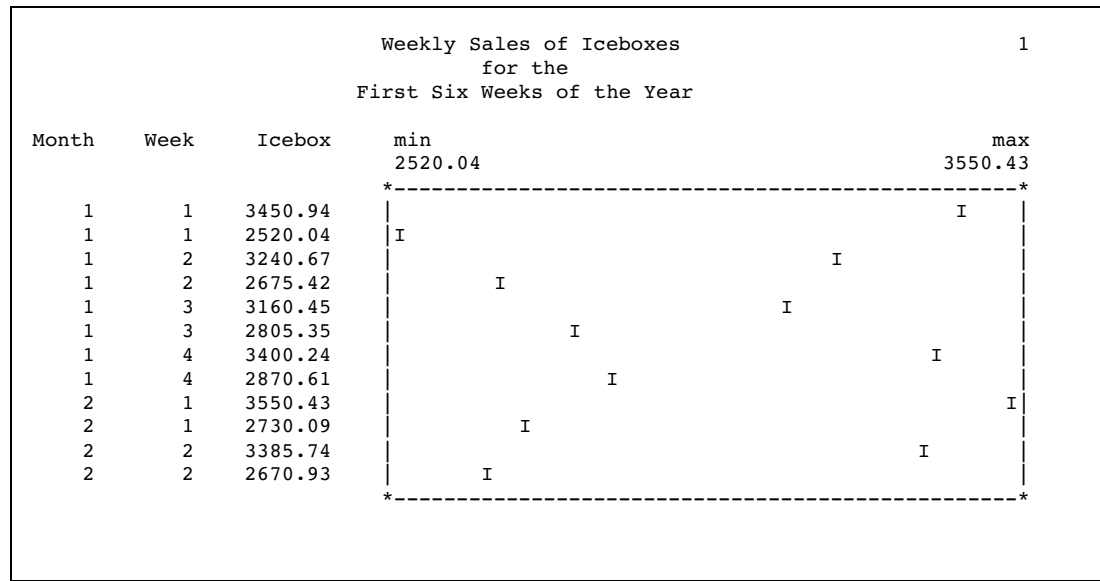
```
id month week;
```

The TITLE statements specify titles for the report.

```
title 'Weekly Sales of Iceboxes';
title2 'for the';
title3 'First Six Weeks of the Year';
run;
```

Output

The column headers in the listing are the variables' names. The plot uses the default plotting symbol, which is the first character of the plot variable's name.



Example 2: Customizing an Axis and a Plotting Symbol

Procedure features:

ID statement

PLOT statement arguments:

using a plotting symbol

AXIS=

Other features:

LABEL statement

PROC FORMAT

SAS system options:

FMTSEARCH=

Data set: SALES on page 1377

This example

- ☐ specifies the character to use as the plotting symbol
- ☐ specifies the minimum and maximum values for the horizontal axis as well as the interval represented by each print position
- ☐ provides context for the points in the plot by printing in the listing the values of two variables that are not in the plot
- ☐ uses a variable's label as a column header in the listing
- ☐ creates and uses a permanent format.

Program

```
libname proclib 'SAS-data-library';
```

The SAS system option FMTSEARCH= adds the SAS data library PROCLIB to the search path that is used to locate formats.

```
options nodate pageno=1 linesize=80 pagesize=60
      fmtsearch=(proclib);
```

PROC FORMAT creates a permanent format for Month. The LIBRARY= option specifies a permanent storage location so that the formats are available in subsequent SAS sessions. This format is used for examples throughout this chapter.

```
proc format library=proclib;
  value monthfmt 1='January'
                2='February';
run;
```

The plot variable, Icebox, appears in both the listing and the output. The plotting symbol is 'R'. AXIS= sets the minimum value of the axis to 2500 and the maximum value to 3600. BY 25 specifies that each print position on the axis represents 25 units (in this case, dollars).

```
proc timeplot data=sales;
  plot icebox='R' / axis=2500 to 3600 by 25;
```

The values of the ID variables, Month and Week, appear in the listing.

```
id month week;
```

The LABEL statement associates a label with the variable Icebox for the duration of the PROC TIMEPLOT step. PROC TIMEPLOT uses the label as the column header in the listing.

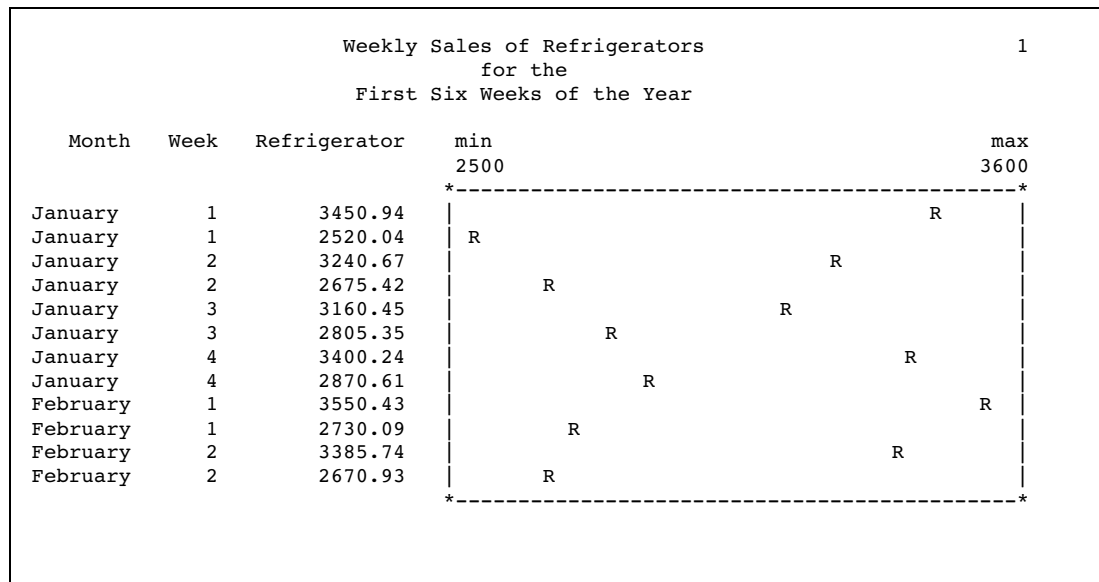
```
label icebox='Refrigerator';
```

The FORMAT statement assigns a format to use for Month in the report. The TITLE statements specify titles.

```
format month monthfmt.;
title 'Weekly Sales of Refrigerators';
title2 'for the';
title3 'First Six Weeks of the Year';
run;
```

Output

The column headers in the listing are the variables' names (for Month and Week, which have no labels) and the variable's label (for Icebox, which has a label). The plotting symbol is **R** (for Refrigerator).



Example 3: Using a Variable for a Plotting Symbol

Procedure features:

ID statement

PLOT statement arguments:

using a variable as the plotting symbol

JOINREF

NPP

REF=

REFCHAR=

Data set: SALES on page 1377

Formats: MONTHFMT. on page 1379

This example

- specifies a variable to use as the plotting symbol to distinguish between points for each of two sales representatives
- suppresses the printing of the values of the plot variable in the listing
- draws a reference line to a specified value on the axis and specifies the character to use to draw the line
- connects the leftmost and rightmost symbols on each line of the plot.

Program

```
libname proclib 'SAS-data-library';
```

The SAS system option FMTSEARCH= adds the SAS data library PROCLIB to the search path that is used to locate formats.

```
options nodate pageno=1 linesize=80 pagesize=60
      fmtsearch=(proclib);
```

The PLOT statement specifies both the plotting variable, Stove, and a symbol variable, Seller. The plotting symbol is the first letter of the formatted value of the Seller (in this case, **L** or **K**).

```
proc timeplot data=sales;
  plot stove=seller /
```

NPP suppresses the appearance of the plotting variable, Stove, in the listing.

```
npp
```

REF= and REFCHAR= draw a line of colons at the sales target of \$1500.

```
ref=1500 refchar=':'
```

JOINREF connects the leftmost and rightmost symbols on each line of the plot.

```
joinref
```

AXIS= sets the minimum value of the horizontal axis to 100 and the maximum value to 3000. BY 50 specifies that each print position on the axis represents 50 units (in this case, dollars).

```
axis=100 to 3000 by 50;
```

The ID statement writes the values of the ID variables, Month and Week, in the listing.

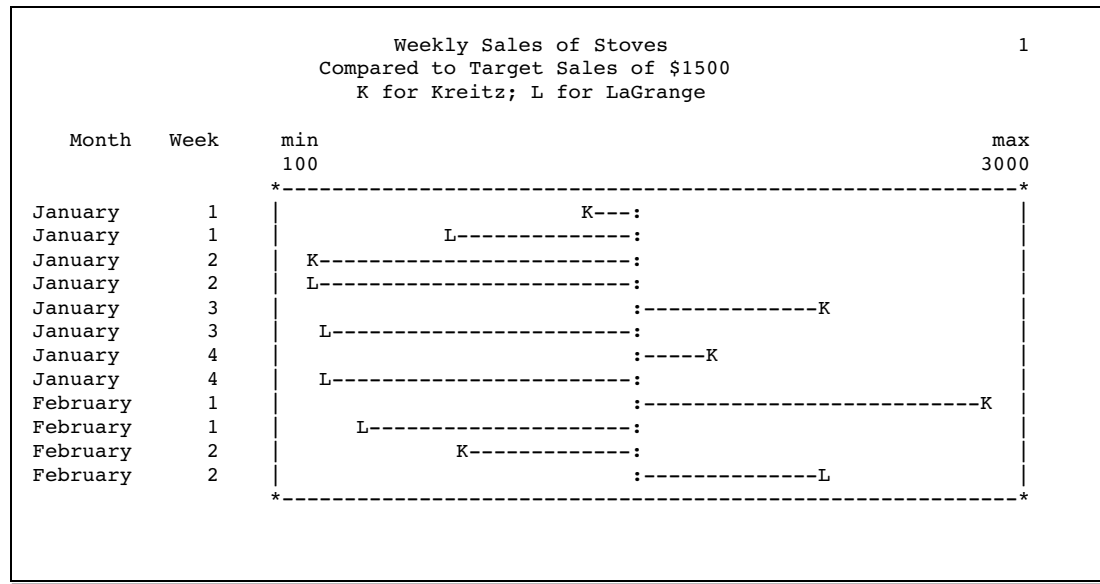
```
id month week;
```

The FORMAT statement assigns a format to use for Month in the report. The TITLE statements specify titles.

```
format month monthfmt.;
title 'Weekly Sales of Stoves';
title2 'Compared to Target Sales of $1500';
title3 'K for Kreitz; L for LaGrange';
run;
```

Output

The plot uses the first letter of the value of Seller as the plotting symbol.



Example 4: Superimposing Two Plots

Procedure features:

PROC TIMEPLOT statement options:

MAXDEC=

PLOT statement arguments:

using two types of plot requests

OVERLAY

REF=MEAN(*variable(s)*)

REVERSE

Data set: SALES on page 1377

This example

- superimposes two plots on one set of axes
- specifies a variable to use as the plotting symbol for one plot and a character to use as the plotting symbol for the other plot
- draws a reference line to the mean value of each of the two variables plotted
- reverses the labeling of the axis so that the largest value is at the far left of the plot.

Program


```
options nodate pageno=1 linesize=80 pagesize=60;
```

MAXDEC= specifies the number of decimal places to display in the listing.

```
proc timeplot data=sales maxdec=0;
```

The PLOT statement requests two plots. One plot uses the first letter of the formatted value of Seller to plot the values of Stove. The other uses the letter **R** (to match the label Refrigerators) to plot the value of Icebox.

```
plot stove=seller icebox='R' /
```

OVERLAY places the two plots on the same set of axes.

```
overlay
```

REF= draws two reference lines: one perpendicular to the mean of Stove, the other perpendicular to the mean of Icebox.

```
ref=mean(stove icebox)
```

REVERSE orders the values on the horizontal axis from largest to smallest.

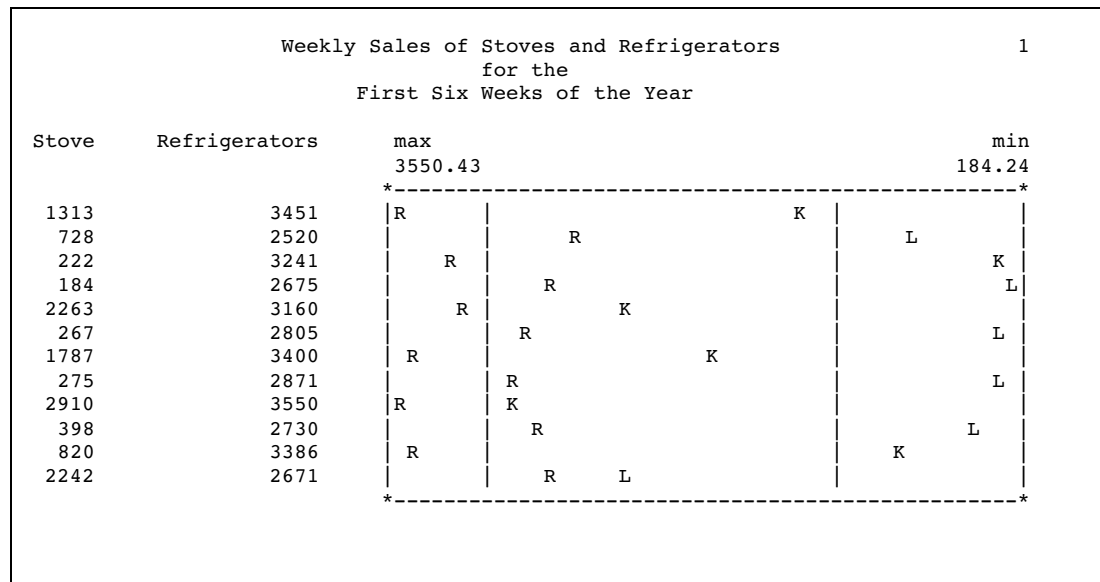
```
reverse;
```

The LABEL statement associates a label with the variable Icebox for the duration of the PROC TIMEPLOT step. PROC TIMEPLOT uses the label as the column header in the listing. The TITLE statements specify titles.

```
label icebox='Refrigerators';
title 'Weekly Sales of Stoves and Refrigerators';
title2 'for the';
title3 'First Six Weeks of the Year';
run;
```

Output

The column header for the variable Icebox in the listing is the variable's label (Refrigerators). One plot uses the first letter of the value of Seller as the plotting symbol. The other plot uses the letter **R**.



Example 5: Showing Multiple Observations on One Line of a Plot

Procedure features:

CLASS statement

PLOT statement arguments:

creating multiple plots

NOSYMNAME

OVPCHAR=

Data set: SALES on page 1377

Formats: MONTHFMT. on page 1379

This example

- ☐ groups observations for the same month and week so that sales for the two sales representatives for the same week appear on the same line of the plot
- ☐ specifies a variable to use as the plotting symbol
- ☐ suppresses the name of the plotting variable from one plot
- ☐ specifies a size for the plots so that they both occupy the same amount of space.

Program

The SAS system option FMTSEARCH= adds the SAS data library PROCLIB to the search path that is used to locate formats.

```
libname proclib 'SAS-data-library';

options nodate pageno=1 linesize=80 pagesize=60
      fmtsearch=(proclib);
```

The CLASS statement groups all observations with the same values of Month and Week into one line in the output. Using the CLASS statement with a symbol variable produces in the listing one column of the plot variable for each value of the symbol variable.

```
proc timeplot data=sales;
  class month week;
```

Each PLOT statement produces a separate plot. The plotting symbol is the first character of the formatted value of the symbol variable: **K** for Kreitz; **L** for LaGrange. POS= specifies that each plot uses 25 print positions for the horizontal axis. OVPCCHAR= designates the exclamation point as the plotting symbol when the plotting symbols coincide. NOSYMNNAME suppresses the name of the symbol variable Seller from the second listing.

```
plot stove=seller / pos=25 ovpcchar='!';
plot icebox=seller / pos=25 ovpcchar='!' nosymname;
```

The FORMAT statement assigns formats to use for Stove, Icebox, and Month in the report. The TITLE statement specifies a title.

```
format stove icebox dollar10.2 month monthfmt.;
title 'Weekly Appliance Sales for the First Quarter';
run;
```

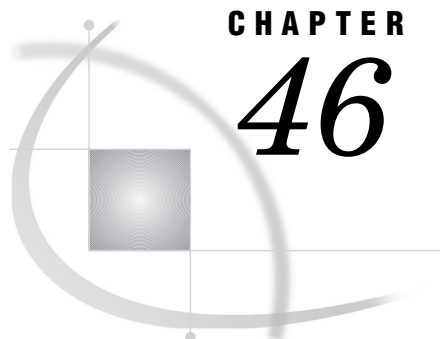
Output

Weekly Appliance Sales for the First Quarter					1
Month	Week	Seller :Kreitz Stove	Seller :LeGrange Stove	min \$184.24	max \$2,910.37

January	1	\$1,312.61	\$728.13	L K	
January	2	\$222.35	\$184.24	!	
January	3	\$2,263.33	\$267.35	L K	
January	4	\$1,787.45	\$274.51	L K	
February	1	\$2,910.37	\$397.98	L K	
February	2	\$819.69	\$2,242.24	K L	

Weekly Appliance Sales for the First Quarter					2
Month	Week	Kreitz Icebox	LeGrange Icebox	min \$2,520.04	max \$3,550.43

January	1	\$3,450.94	\$2,520.04	L	K
January	2	\$3,240.67	\$2,675.42	L K	
January	3	\$3,160.45	\$2,805.35	L K	
January	4	\$3,400.24	\$2,870.61	L K	
February	1	\$3,550.43	\$2,730.09	L K	
February	2	\$3,385.74	\$2,670.93	L K	



CHAPTER

46

The TRANSPOSE Procedure

<i>Overview: TRANSPOSE Procedure</i>	1387
<i>Syntax: TRANSPOSE Procedure</i>	1389
<i>PROC TRANSPOSE Statement</i>	1390
<i>BY Statement</i>	1391
<i>COPY Statement</i>	1393
<i>ID Statement</i>	1393
<i>IDLABEL Statement</i>	1394
<i>VAR Statement</i>	1395
<i>Results: TRANSPOSE Procedure</i>	1395
<i>Output Data Set</i>	1395
<i>Attributes of Transposed Variables</i>	1396
<i>Names of Transposed Variables</i>	1396
<i>Examples: TRANSPOSE Procedure</i>	1396
<i>Example 1: Performing a Simple Transposition</i>	1396
<i>Example 2: Naming Transposed Variables</i>	1398
<i>Example 3: Labeling Transposed Variables</i>	1399
<i>Example 4: Transposing BY Groups</i>	1400
<i>Example 5: Naming Transposed Variables When the ID Variable Has Duplicate Values</i>	1402
<i>Example 6: Transposing Data for Statistical Analysis</i>	1404

Overview: TRANSPOSE Procedure

The TRANSPOSE procedure creates an output data set by restructuring the values in a SAS data set, transposing selected variables into observations. The TRANSPOSE procedure can often eliminate the need to write a lengthy DATA step to achieve the same result. Further, the output data set can be used in subsequent DATA or PROC steps for analysis, reporting, or further data manipulation.

PROC TRANSPOSE does not produce printed output. To print the output data set from the PROC TRANSPOSE step, use PROC PRINT, PROC REPORT, or another SAS reporting tool.

A *transposed variable* is a variable that the procedure creates by transposing the values of an observation in the input data set into values of a variable in the output data set.

The following example illustrates a simple transposition. In the input data set, each *variable* represents the scores from one tester. In the output data set, each *observation* now represents the scores from one tester. Each value of `_NAME_` is the name of a variable in the input data set that the procedure transposed. Thus, the value of `_NAME_` identifies the source of each observation in the output data set. For example, the values in the first observation in the output data set come from the values of the variable Tester1 in the input data set. The statements that produce the output follow.

```

proc print data=proclib.product noobs;
    title 'The Input Data Set';
run;

proc transpose data=proclib.product
               out=proclib.product_transposed;
run;

proc print data=proclib.product_transposed noobs;
    title 'The Output Data Set';
run;

```

Output 46.1 A Simple Transposition

The Input Data Set									1
	Tester1	Tester2	Tester3	Tester4					
	22	25	21	21					
	15	19	18	17					
	17	19	19	19					
	20	19	16	19					
	14	15	13	13					
	15	17	18	19					
	10	11	9	10					
	22	24	23	21					

The Output Data Set									2
NAME	COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8	
Tester1	22	15	17	20	14	15	10	22	
Tester2	25	19	19	19	15	17	11	24	
Tester3	21	18	19	16	13	18	9	23	
Tester4	21	17	19	19	13	19	10	21	

The next example, which uses BY groups, is more complex. The input data set represents measurements of the weight and length of fish at two lakes. The statements that create the output data set do the following:

- ☐ transpose only the variables that contain the length measurements
- ☐ create six BY groups, one for each lake and date
- ☐ use a data set option to name the transposed variable.

Output 46.2 A Transposition with BY Groups

Input Data Set										1
L o c a t i o n		D a t e	L e n g t h	W e i g h t	L e n g t h	W e i g h t	L e n g t h	W e i g h t	L e n g t h	W e i g h t
			1	1	2	2	3	3	4	4
Cole Pond	02JUN95	31	0.25	32	0.30	32	0.25	33	0.30	
Cole Pond	03JUL95	33	0.32	34	0.41	37	0.48	32	0.28	
Cole Pond	04AUG95	29	0.23	30	0.25	34	0.47	32	0.30	
Eagle Lake	02JUN95	32	0.35	32	0.25	33	0.30	.	.	
Eagle Lake	03JUL95	30	0.20	36	0.45	
Eagle Lake	04AUG95	33	0.30	33	0.28	34	0.42	.	.	

Fish Length Data for Each Location and Date					2
Location	Date	_NAME_	Measurement		
Cole Pond	02JUN95	Length1	31		
Cole Pond	02JUN95	Length2	32		
Cole Pond	02JUN95	Length3	32		
Cole Pond	02JUN95	Length4	33		
Cole Pond	03JUL95	Length1	33		
Cole Pond	03JUL95	Length2	34		
Cole Pond	03JUL95	Length3	37		
Cole Pond	03JUL95	Length4	32		
Cole Pond	04AUG95	Length1	29		
Cole Pond	04AUG95	Length2	30		
Cole Pond	04AUG95	Length3	34		
Cole Pond	04AUG95	Length4	32		
Eagle Lake	02JUN95	Length1	32		
Eagle Lake	02JUN95	Length2	32		
Eagle Lake	02JUN95	Length3	33		
Eagle Lake	02JUN95	Length4	.		
Eagle Lake	03JUL95	Length1	30		
Eagle Lake	03JUL95	Length2	36		
Eagle Lake	03JUL95	Length3	.		
Eagle Lake	03JUL95	Length4	.		
Eagle Lake	04AUG95	Length1	33		
Eagle Lake	04AUG95	Length2	33		
Eagle Lake	04AUG95	Length3	34		
Eagle Lake	04AUG95	Length4	.		

For a complete explanation of the SAS program that produces these results, see Example 4 on page 1400.

Syntax: TRANSPOSE Procedure

Tip: Does not support the Output Delivery System

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, “Statements with the Same Function in Multiple Procedures,” on page 53 for details. You can also use any global statements as well. See “Global Statements” on page 18 for a list.

```

PROC TRANSPOSE <DATA=input-data-set> <LABEL=label> <LET>
    <NAME=name> <OUT=output-data-set> <PREFIX=prefix>;
BY <DESCENDING> variable-1
    <...<DESCENDING> variable-n>
    <NOTSORTED>;
COPY variable(s);
ID variable;
    IDLABEL variable;
VAR variable(s);

```

To do this	Use this statement
Transpose each BY group	BY
Copy variables directly without transposing them	COPY
Specify a variable whose values name the transposed variables	ID
Create labels for the transposed variables	IDLABEL
List the variables to transpose	VAR

PROC TRANSPOSE Statement

Reminder: You can use data set options with the DATA= and OUT= options. See “Data Set Options” on page 17 for a list.

```

PROC TRANSPOSE <DATA=input-data-set> <LABEL=label> <LET>
    <NAME=name> <OUT=output-data-set> <PREFIX=prefix>;

```

Options

DATA= *input-data-set*

names the SAS data set to transpose.

Default: most recently created SAS data set

LABEL= *label*

specifies a name for the variable in the output data set that contains the label of the variable that is being transposed to create the current observation.

Default: _LABEL_

LET

allows duplicate values of an ID variable. PROC TRANSPOSE transposes the observation containing the last occurrence of a particular ID value within the data set or BY group.

Featured in: Example 5 on page 1402

NAME= *name*

specifies the name for the variable in the output data set that contains the name of the variable that is being transposed to create the current observation.

Default: `_NAME_`

Featured in: Example 2 on page 1398

OUT= *output-data-set*

names the output data set. If *output-data-set* does not exist, PROC TRANSPOSE creates it by using the `DATA n` naming convention.

Default: `DATA n`

Featured in: Example 1 on page 1396

PREFIX= *prefix*

specifies a prefix to use in constructing names for transposed variables in the output data set. For example, if `PREFIX=VAR`, the names of the variables are `VAR1`, `VAR2`, ..., `VAR n` .

Interaction: when you use `PREFIX=` with an ID statement, the value prefixes to the ID value.

Featured in: Example 2 on page 1398

BY Statement

Defines BY groups.

Main discussion: “BY” on page 54

Featured in: Example 4 on page 1400

Restriction: You cannot use PROC TRANSPOSE with a BY statement or an ID statement with an engine that supports concurrent access if another user is updating the data set at the same time.

Required Arguments

variable

specifies the variable that PROC TRANSPOSE uses to form BY groups. You can specify more than one variable. If you do not use the `NOTSORTED` option in the BY statement, either the observations must be sorted by all the variables that you specify, or they must be indexed appropriately. Variables in a BY statement are called *BY variables*.

Options

DESCENDING

specifies that the data set is sorted in descending order by the variable that immediately follows the word `DESCENDING` in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The data are grouped in another way, such as chronological order.

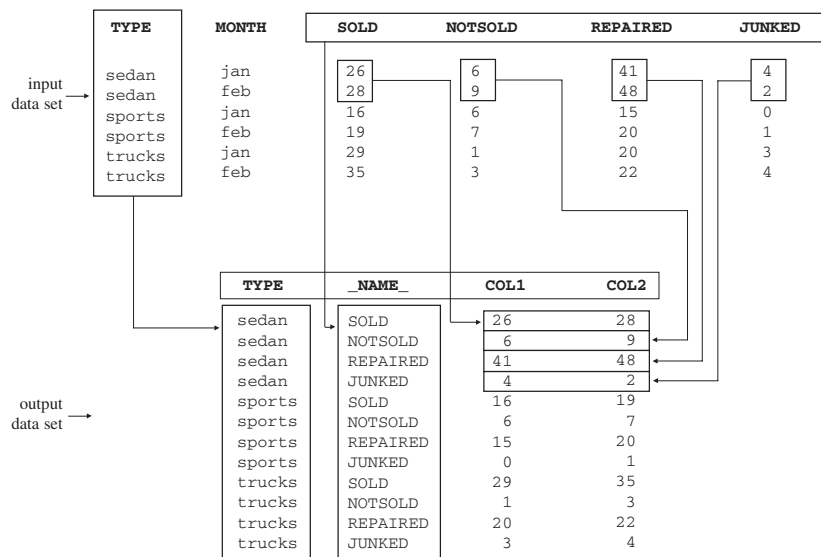
The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, the procedure treats each contiguous set as a separate BY group.

Transpositions with BY Groups

PROC TRANSPOSE does not transpose BY groups. Instead, for each BY group, PROC TRANSPOSE creates one observation for each variable that it transposes.

Figure 46.1 on page 1392 shows what happens when you transpose a data set with BY groups. TYPE is the BY variable, and SOLD, NOTSOLD, REPAIRED, and JUNKED are the variables to transpose.

Figure 46.1 Transposition with BY Groups



- The number of observations in the output data set (12) is the number of BY groups (3) multiplied by the number of variables that are transposed (4).
- The BY variable is not transposed.
- _NAME_ contains the name of the variable in the input data set that was transposed to create the current observation in the output data set. You can use the NAME= option to specify another name for the _NAME_ variable.
- The maximum number of observations in any BY group in the input data set is two; therefore, the output data set contains two variables, COL1 and COL2. COL1 and COL2 contain the values of SOLD, NOTSOLD, REPAIRED, and JUNKED.

Note: If a BY group in the input data set has more observations than other BY groups, PROC TRANSPOSE assigns missing values in the output data set to the variables that have no corresponding input observations. \triangle

COPY Statement

Copies variables directly from the input data set to the output data set without transposing them.

Featured in: Example 6 on page 1404

COPY *variable(s)*;

Required Argument

variable(s)

names one or more variables that the COPY statement copies directly from the input data set to the output data set without transposing them.

Details

Because the COPY statement copies variables directly to the output data set, the number of observations in the output data set is equal to the number of observations in the input data set.

The procedure pads the output data set with missing values if the number of observations in the input data set is not equal to the number of variables that it transposes.

ID Statement

Specifies a variable in the input data set whose formatted values name the transposed variables in the output data set.

Featured in: Example 2 on page 1398

Restriction: You cannot use PROC TRANSPOSE with an ID statement or a BY statement with an engine that supports concurrent access if another user is updating the data set at the same time.

ID *variable*;

Required Argument

variable

names the variable whose formatted values name the transposed variables.

Duplicate ID Values

Typically, each formatted ID value occurs only once in the input data set or, if you use a BY statement, only once within a BY group. Duplicate values cause PROC TRANSPOSE to issue a warning message and stop. However, if you use the LET option

in the PROC TRANSPOSE statement, the procedure issues a warning message about duplicate ID values and transposes the observation that contains the last occurrence of the duplicate ID value.

Making Variable Names out of Numeric Values

When you use a numeric variable as an ID variable, PROC TRANSPOSE changes the formatted ID value into a valid SAS name.

However, SAS variable names cannot begin with a number. Thus, when the first character of the formatted value is numeric, the procedure prefixes an underscore to the value, truncating the last character of a 32-character value. Any remaining invalid characters are replaced by underscores. The procedure truncates to 32 characters any ID value that is longer than 32 characters when it uses that value to name a transposed variable.

If the formatted value looks like a numeric constant, PROC TRANSPOSE changes the characters '+', '−', and '.' to 'P', 'N', and 'D', respectively. If the formatted value has characters that are not numerics, PROC TRANSPOSE changes the characters '+', '−', and '.' to underscores.

Note: If the value of the VALIDVARNAME system option is V6, PROC TRANSPOSE truncates transposed variable names to eight characters. △

Missing Values

If you use an ID variable that contains a missing value, PROC TRANSPOSE writes an error message to the log. The procedure does not transpose observations that have a missing value for the ID variable.

IDLABEL Statement

Creates labels for the transposed variables.

Restriction: Must appear after an ID statement.

Featured in: Example 3 on page 1399

IDLABEL *variable*;

Required Argument

variable

names the variable whose values the procedure uses to label the variables that the ID statement names. *variable* can be character or numeric.

Note: To see the effect of the IDLABEL statement, print the output data set with the PRINT procedure by using the LABEL option, or print the contents of the output data set by using the CONTENTS statement in the DATASETS procedure. △

VAR Statement

Lists the variables to transpose.

Featured in: Example 4 on page 1400 and Example 6 on page 1404

VAR *variable(s)*;

Required Argument

variable(s)

names one or more variables to transpose.

Details

- ☐ If you omit the VAR statement, the TRANSPOSE procedure transposes all numeric variables in the input data set that are not listed in another statement.
- ☐ You must list character variables in a VAR statement if you want to transpose them.

Results: TRANSPOSE Procedure

Output Data Set

The TRANSPOSE procedure always produces an output data set, regardless of whether you specify the OUT= option in the PROC TRANSPOSE statement. PROC TRANSPOSE does not print the output data set. Use PROC PRINT, PROC REPORT, or some other SAS reporting tool to print the output data set.

The output data set contains the following variables:

- ☐ variables that result from transposing the values of each variable into an observation.
- ☐ a variable that PROC TRANSPOSE creates to identify the source of the values in each observation in the output data set. This variable is a character variable whose values are the names of the variables that are transposed from the input data set. By default, PROC TRANSPOSE names this variable `_NAME_`. To override the default name, use the NAME= option. The label for the `_NAME_` variable is NAME OF FORMER VARIABLE.
- ☐ variables that PROC TRANSPOSE copies from the input data set when you use either the BY or COPY statement. These variables have the same names and values as they do in the input data set.
- ☐ a character variable whose values are the variable labels of the variables that are being transposed (if any of the variables that the procedure is transposing have labels). Specify the name of the variable by using the LABEL= option. The default is `_LABEL_`.

Note: If the value of the LABEL= option or the NAME= option is the same as a variable that appears in a BY or COPY statement, the output data set does not contain a variable whose values are the names or labels of the transposed variables. \triangle

Attributes of Transposed Variables

- All transposed variables are the same type and length.
- If all variables that the procedure is transposing are numeric, the transposed variables are numeric. Thus, if the numeric variable has a character string as a formatted value, its unformatted numeric value is transposed.
- If any variable that the procedure is transposing is character, all transposed variables are character. Thus, if you are transposing a numeric variable that has a character string as a formatted value, the formatted value is transposed.
- The length of the transposed variables is equal to the length of the longest variable that is being transposed.

Names of Transposed Variables

PROC TRANSPOSE names transposed variables by using the following rules:

- 1 An ID statement specifies a variable in the input data set whose formatted values become names for the transposed variables.
- 2 The PREFIX= option specifies a prefix to use in constructing the names of transposed variables.
- 3 If you do not use an ID statement or the PREFIX= option, then PROC TRANSPOSE looks for an input variable called _NAME_ from which to get the names of the transposed variables.
- 4 If you do not use an ID statement or the PREFIX= option, and if the input data set does not contain a variable named _NAME_, then PROC TRANSPOSE assigns the names COL1, COL2, ..., COL n to the transposed variables.

Examples: TRANSPOSE Procedure

Example 1: Performing a Simple Transposition

Procedure features:

PROC TRANSPOSE statement option:
OUT=

This example performs a default transposition and uses no subordinate statements.

Program

```
options nodate pageno=1 linesize=80 pagesize=40;
```

The data set SCORE contains students' names, their identification numbers, and their grades on two tests and a final exam.

```
data score;
  input Student $9. +1 StudentID $ Section $ Test1 Test2 Final;
  datalines;
Capalleti 0545 1 94 91 87
Dubose    1252 2 51 65 91
Engles    1167 1 95 97 97
Grant     1230 2 63 75 80
Krupski   2527 2 80 76 71
Lundsford 4860 1 92 40 86
Mcbane    0674 1 75 78 72
;
```

PROC TRANSPOSE transposes only the numeric variables, Test1, Test2, and Final, because no VAR statement appears and none of the numeric variables appear in another statement. OUT= puts the result of the transposition in the data set SCORE_TRANSPOSED.

```
proc transpose data=score out=score_transposed;
run;
```

PROC PRINT prints the output data set.

```
proc print data=score_transposed noobs;
  title 'Student Test Scores in Variables';
run;
```

Output

In the output data set SCORE_TRANSPOSED, the variables COL1 through COL7 contain the individual scores for the students. Each observation contains all the scores for one test. The variable _NAME_ contains the names of the variables from the input data set that were transposed.

Student Test Scores in Variables							1
NAME	COL1	COL2	COL3	COL4	COL5	COL6	COL7
Test1	94	51	95	63	80	92	75
Test2	91	65	97	75	76	40	78
Final	87	91	97	80	71	86	72

Example 2: Naming Transposed Variables

Procedure features:

PROC TRANSPOSE statement options:

NAME=

PREFIX=

ID statement

Data set: SCORE on page 1397

This example uses the values of a variable and a user-supplied value to name transposed variables.

Program

```
options nodate pageno=1 linesize=80 pagesize=40;
```

PROC TRANSPOSE transposes only the numeric variables, Test1, Test2, and Final, because no VAR statement appears. OUT= puts the result of the transposition in the IDNUMBER data set. NAME= specifies Test as the name for the variable that contains the names of the variables in the input data set that the procedure transposes. The procedure names the transposed variables by using the value from PREFIX=, sn, and the value of the ID variable StudentID.

```
proc transpose data=score out=idnumber name=Test
    prefix=sn;
    id studentid;
run;
```

PROC PRINT prints the data set.

```
proc print data=idnumber noobs;
    title 'Student Test Scores';
run;
```

Output

This is the output data set, IDNUMBER.

Student Test Scores							1
Test	sn0545	sn1252	sn1167	sn1230	sn2527	sn4860	sn0674
Test1	94	51	95	63	80	92	75
Test2	91	65	97	75	76	40	78
Final	87	91	97	80	71	86	72

Example 3: Labeling Transposed Variables

Procedure features:

PROC TRANSPOSE statement option:

PREFIX=

IDLABEL statement

Data set: SCORE on page 1397

This example uses the values of the variable in the IDLABEL statement to label transposed variables.

Program

```
options nodate pageno=1 linesize=80 pagesize=40;
```

PROC TRANSPOSE transposes only the numeric variables, Test1, Test2, and Final, because no VAR statement appears. OUT= puts the result of the transposition in the IDLABEL data set. NAME= specifies Test as the name for the variable that contains the names of the variables in the input data set that the procedure transposes. The procedure names the transposed variables by using the value from PREFIX=, sn, and the value of the ID variable StudentID.

```
proc transpose data=score out=idlabel name=Test
  prefix=sn;
  id studentid;
```

PROC TRANSPOSE uses the values of the variable Student to label the transposed variables. The procedure provides the following as the label for the _NAME_ variable:

```
NAME OF FORMER VARIABLE
```

```
      idlabel student;
run;
```

PROC PRINT prints the output data set and uses the variable labels as column headers. The LABEL option causes PROC PRINT to print variable labels for column headers.

```
proc print data=idlabel label noobs;
  title 'Student Test Scores';
run;
```

Output

This is the output data set, IDLABEL.

Student Test Scores								1
NAME OF FORMER VARIABLE	Capalleti	Dubose	Engles	Grant	Krupski	Lundsford	Mcbane	
Test1	94	51	95	63	80	92	75	
Test2	91	65	97	75	76	40	78	
Final	87	91	97	80	71	86	72	

Example 4: Transposing BY Groups

Procedure features:

BY statement

VAR statement

Other features: Data set option:

RENAME=

This example illustrates transposing BY groups and selecting variables to transpose.

Program

```
options nodate pageno=1 linesize=80 pagesize=40;
```

The input data represent length and weight measurements of fish that were caught at two ponds on three separate days. The data are sorted by Location and Date.

```
data fishdata;
  infile datalines missover;
  input Location & $10. Date date7.
        Length1 Weight1 Length2 Weight2 Length3 Weight3
        Length4 Weight4;
  format date date7.;
  datalines;
Cole Pond  2JUN95 31 .25 32 .3  32 .25 33 .3
Cole Pond  3JUL95 33 .32 34 .41 37 .48 32 .28
Cole Pond  4AUG95 29 .23 30 .25 34 .47 32 .3
Eagle Lake 2JUN95 32 .35 32 .25 33 .30
Eagle Lake 3JUL95 30 .20 36 .45
Eagle Lake 4AUG95 33 .30 33 .28 34 .42
;
```

OUT= puts the result of the transposition in the FISHLENGTH data set. RENAME= renames COL1 in the output data set to Measurement.

```
proc transpose data=fishdata  
    out=fishlength(rename=(coll=Measurement));
```

PROC TRANSPOSE transposes only the Length1-Length4 variables because they appear in the VAR statement.

```
var length1-length4;
```

The BY statement creates BY groups for each unique combination of values of Location and Date. The procedure does not transpose the BY variables.

```
by location date;  
run;
```

PROC PRINT prints the output data set.

```
proc print data=fishlength noobs;  
    title 'Fish Length Data for Each Location and Date';  
run;
```

Output

This is the output data set, FISHLENGTH. For each BY group in the original data set, PROC TRANSPOSE creates four observations, one for each variable that it is transposing. Missing values appear for the variable Measurement (renamed from COL1) when the variables that are being transposed have no value in the input data set for that BY group. Several observations have a missing value for Measurement. For example, in the last observation, a missing value appears because the input data contained no value for Length4 on 04AUG95 at Eagle Lake.

Fish Length Data for Each Location and Date				1
Location	Date	_NAME_	Measurement	
Cole Pond	02JUN95	Length1	31	
Cole Pond	02JUN95	Length2	32	
Cole Pond	02JUN95	Length3	32	
Cole Pond	02JUN95	Length4	33	
Cole Pond	03JUL95	Length1	33	
Cole Pond	03JUL95	Length2	34	
Cole Pond	03JUL95	Length3	37	
Cole Pond	03JUL95	Length4	32	
Cole Pond	04AUG95	Length1	29	
Cole Pond	04AUG95	Length2	30	
Cole Pond	04AUG95	Length3	34	
Cole Pond	04AUG95	Length4	32	
Eagle Lake	02JUN95	Length1	32	
Eagle Lake	02JUN95	Length2	32	
Eagle Lake	02JUN95	Length3	33	
Eagle Lake	02JUN95	Length4	.	
Eagle Lake	03JUL95	Length1	30	
Eagle Lake	03JUL95	Length2	36	
Eagle Lake	03JUL95	Length3	.	
Eagle Lake	03JUL95	Length4	.	
Eagle Lake	04AUG95	Length1	33	
Eagle Lake	04AUG95	Length2	33	
Eagle Lake	04AUG95	Length3	34	
Eagle Lake	04AUG95	Length4	.	

Example 5: Naming Transposed Variables When the ID Variable Has Duplicate Values

Procedure features:

PROC TRANSPOSE statement option:

LET

This example shows how to use values of a variable (ID) to name transposed variables even when the ID variable has duplicate values.

Program

```
options nodate pageno=1 linesize=64 pagesize=40;
```

STOCKS contains stock prices for two competing kite manufacturers. The prices are recorded three times a day – at opening, at noon, and at closing – on two days. Notice that the input data set contains duplicate values for the Date variable.

```
data stocks;
    input Company $14. Date $ Time $ Price;
    datalines;
Horizon Kites jun11 opening 29
Horizon Kites jun11 noon    27
Horizon Kites jun11 closing 27
Horizon Kites jun12 opening 27
Horizon Kites jun12 noon    28
Horizon Kites jun12 closing 30
SkyHi Kites   jun11 opening 43
SkyHi Kites   jun11 noon    43
SkyHi Kites   jun11 closing 44
SkyHi Kites   jun12 opening 44
SkyHi Kites   jun12 noon    45
SkyHi Kites   jun12 closing 45
    ;
```

LET transposes only the last observation for each BY group. PROC TRANSPOSE transposes only the Price variable. OUT= puts the result of the transposition in the CLOSE data set.

```
proc transpose data=stocks out=close let;
```

The BY statement creates two BY groups, one for each company.

```
    by company;
```

The values of Date are used as names for the transposed variables.

```
    id date;
run;
```

PROC PRINT prints the output data set.

```
proc print data=close noobs;
    title 'Closing Prices for Horizon Kites and SkyHi Kites';
run;
```

Output

This is the output data set, CLOSE.

Closing Prices for Horizon Kites and SkyHi Kites				1
Company	_NAME_	jun11	jun12	
Horizon Kites	Price	27	30	
SkyHi Kites	Price	44	45	

Example 6: Transposing Data for Statistical Analysis

Procedure features:

COPY statement

VAR statement

This example arranges data to make them suitable for either a multivariate or a univariate repeated-measures analysis.

The data are from Chapter 8, “Repeated-Measures Analysis of Variance,” in *SAS System for Linear Models, Third Edition*.

Program 1

```
options nodate pageno=1 linesize=80 pagesize=40;
```

The data represent the results of an exercise therapy study of three weight-lifting programs: CONT is control, RI is a program in which the number of repetitions is increased, and WI is a program in which the weight is increased.

```
data weights;
  input Program $ s1-s7;
  datalines;
CONT 85 85 86 85 87 86 87
CONT 80 79 79 78 78 79 78
CONT 78 77 77 77 76 76 77
CONT 84 84 85 84 83 84 85
CONT 80 81 80 80 79 79 80
RI 79 79 79 80 80 78 80
RI 83 83 85 85 86 87 87
RI 81 83 82 82 83 83 82
RI 81 81 81 82 82 83 81
RI 80 81 82 82 82 84 86
WI 84 85 84 83 83 83 84
WI 74 75 75 76 75 76 76
WI 83 84 82 81 83 83 82
```

```

WI      86 87 87 87 87 87 86
WI      82 83 84 85 84 85 86
;

```

The DATA step rearranges WEIGHTS to create the data set SPLIT. The DATA step transposes the strength values and creates two new variables: Time and Subject. SPLIT contains one observation for each repeated measure. SPLIT can be used in a PROC GLM step for a univariate repeated-measures analysis.

```

data split;
  set weights;
  array s{7} s1-s7;
  Subject + 1;
  do Time=1 to 7;
    Strength=s{time};
    output;
  end;
  drop s1-s7;
run;

```

PROC PRINT prints the data set. The OBS= data set option limits the printing to the first 15 observations. SPLIT has 105 observations.

```

proc print data=split(obs=15) noobs;
  title 'SPLIT Data Set';
  title2 'First 15 Observations Only';
run;

```

Output 1

SPLIT Data Set				1
First 15 Observations Only				
Program	Subject	Time	Strength	
CONT	1	1	85	
CONT	1	2	85	
CONT	1	3	86	
CONT	1	4	85	
CONT	1	5	87	
CONT	1	6	86	
CONT	1	7	87	
CONT	2	1	80	
CONT	2	2	79	
CONT	2	3	79	
CONT	2	4	78	
CONT	2	5	78	
CONT	2	6	79	
CONT	2	7	78	
CONT	3	1	78	

Program 2

```
options nodate pageno=1 linesize=80 pagesize=40;
```

PROC TRANSPOSE transposes SPLIT to create TOTSPLIT. The TOTSPLIT data set contains the same variables as SPLIT and a variable for each strength measurement (Str1-Str7). TOTSPLIT can be used for either a multivariate repeated-measures analysis or a univariate repeated-measures analysis.

```
proc transpose data=split out=totsplit prefix=Str;
```

The variables in the BY and COPY statements are not transposed. TOTSPLIT contains the variables Program, Subject, Time, and Strength with the same values that are in SPLIT. The BY statement creates the first observation in each BY group, which contains the transposed values of Strength. The COPY statement creates the other observations in each BY group by copying the values of Time and Strength without transposing them.

```
by program subject;  
copy time strength;
```

The VAR statement specifies the Strength variable as the only variable to be transposed.

```
var strength;  
run;
```

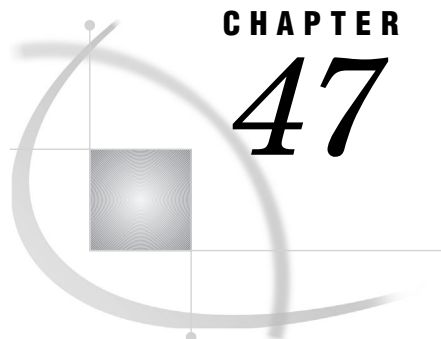
PROC PRINT prints the output data set.

```
proc print data=totsplit(obs=15) noobs;  
  title  'TOTSPLIT Data Set';  
  title2 'First 15 Observations Only';  
run;
```

Output 2

The variables in TOTSPLIT with missing values are used only in a multivariate repeated-measures analysis. The missing values do not preclude this data set from being used in a repeated-measures analysis because the MODEL statement in PROC GLM ignores observations with missing values.

TOTSPLIT Data Set												1
First 15 Observations Only												
Program	Subject	Time	Strength	_NAME_	Str1	Str2	Str3	Str4	Str5	Str6	Str7	
CONT	1	1	85	Strength	85	85	86	85	87	86	87	
CONT	1	2	85		
CONT	1	3	86		
CONT	1	4	85		
CONT	1	5	87		
CONT	1	6	86		
CONT	1	7	87		
CONT	2	1	80	Strength	80	79	79	78	78	79	78	
CONT	2	2	79		
CONT	2	3	79		
CONT	2	4	78		
CONT	2	5	78		
CONT	2	6	79		
CONT	2	7	78		
CONT	3	1	78	Strength	78	77	77	77	76	76	77	



CHAPTER

47

The TRANTAB Procedure

<i>Overview: TRANTAB Procedure</i>	1409
<i>Concepts: TRANTAB Procedure</i>	1410
<i>Understanding Translation Tables and Character Sets for PROC TRANTAB</i>	1410
<i>Storing Translation Tables with PROC TRANTAB</i>	1410
<i>Modifying SAS-supplied Translation Tables with PROC TRANTAB</i>	1411
<i>Using Translation Tables Outside PROC TRANTAB</i>	1411
<i>Using Translation Tables in the SORT Procedure</i>	1411
<i>Using Translation Tables with the CPORT and CIMPORT Procedures</i>	1411
<i>Using Translation Tables with Remote Library Services</i>	1412
<i>Using Translation Tables in SAS/GRAPH Software</i>	1412
<i>Syntax: TRANTAB Procedure</i>	1413
<i>PROC TRANTAB Statement</i>	1413
<i>CLEAR Statement</i>	1415
<i>INVERSE Statement</i>	1415
<i>LIST Statement</i>	1415
<i>LOAD Statement</i>	1416
<i>REPLACE Statement</i>	1417
<i>SAVE Statement</i>	1418
<i>SWAP Statement</i>	1418
<i>Examples: TRANTAB Procedure</i>	1419
<i>Example 1: Viewing a Translation Table</i>	1419
<i>Example 2: Creating a Translation Table</i>	1420
<i>Example 3: Editing by Specifying a Decimal Value for Starting Position</i>	1422
<i>Example 4: Editing by Using a Quoted Character for Starting Position</i>	1425
<i>Example 5: Creating the Inverse of a Table</i>	1427
<i>Example 6: Using Different Translation Tables for Sorting</i>	1429
<i>Example 7: Editing Table 1 and Table 2</i>	1431

Overview: TRANTAB Procedure

The TRANTAB procedure creates, edits, and displays customized translation tables. In addition, you can use PROC TRANTAB to view and modify translation tables that are supplied by SAS. These SAS-supplied tables are stored in the SASHELP.HOST catalog. Any translation table that you create or customize is stored in your SASUSER.PROFILE catalog. Translation tables have an entry type of TRANTAB.

Translation tables are operating environment-specific SAS catalog entries that are used to translate the values of one (coded) character set to another. A translation table has two halves: table one provides a translation, such as ASCII to EBCDIC; table two provides the inverse (or reverse) translation, such as EBCDIC to ASCII. Each half of a

translation table is an array of 256 two-digit *positions*, each of which contains a one-byte unsigned number that corresponds to a coded character.

The SAS System uses translation tables for the following purposes:

- determining the collating sequence in the SORT procedure
- performing transport-format translations when you transfer files with the CPORT and CIMPORT procedures
- performing translations between operating environments when you access remote data in SAS/CONNECT or SAS/SHARE software
- facilitating data communications between the operating environment and a graphics device when you run SAS/GRAPH software in an IBM environment
- accommodating national language character sets other than U.S. English.

PROC TRANTAB produces no output. It can display translation tables and notes in the SAS log.

Concepts: TRANTAB Procedure

Understanding Translation Tables and Character Sets for PROC TRANTAB

The k th element in a translation table corresponds to the k th element of an ordered character set. For example, position 00 (which is byte 1) in a translation table contains a coded value that corresponds to the first element of the ordered character set. To determine the position of a character in your operating environment's character set, use the SAS function RANK. The following example shows how to use RANK:

```
data _null_;
  x=rank('a');
  put "The position of a is " x ".";
```

The SAS log prints the following message: **The position of a is 97 .**

Each position in a translation table contains a hexadecimal number that is within the range of 0 ('00'x) to 255 ('FF'x). Hexadecimal values always end with an x. You can represent one or more consecutive hexadecimal values within quotation marks followed by a single x. For example, a string of three consecutive hexadecimal values can be written as '08090A'x. The SAS log displays each row of a translation table as 16 hexadecimal values enclosed in quotes followed by an x. The SAS log also lists reference numbers in the vertical and horizontal margins that correspond to the positions in the table. Example 1 on page 1419 shows how the SAS log displays a translation table.

Storing Translation Tables with PROC TRANTAB

When you use PROC TRANTAB to create a customized translation table, the procedure automatically stores the table in your SASUSER.PROFILE catalog. This enables you to use customized translation tables without affecting other users. When you specify the translation table in the SORT procedure or in a GOPTIONS statement, the software first looks in your SASUSER.PROFILE catalog to find the table. If the specified translation table is not in your SASUSER.PROFILE catalog, the software looks in the SASHELP.HOST catalog.

If you want the translation table you create to be globally accessed, have your SAS Installation Coordinator copy the table from your SASUSER.PROFILE catalog (using the CATALOG procedure) to the SASHELP.HOST catalog. If the table is not found there, the software will continue to search in SASHELP.LOCALE for the table.

Modifying SAS-supplied Translation Tables with PROC TRANTAB

If a SAS-supplied translation table does not meet your needs, you can use PROC TRANTAB to edit it and create a new table. That is, you can issue the PROC TRANTAB statement that specifies the SAS-supplied table, edit the table, and then save the table using the SAVE statement. The modified translation table is saved in your SASUSER.PROFILE catalog. If you are a SAS Installation Coordinator, you can modify a translation table with PROC TRANTAB and then use the CATALOG procedure to copy the modified table from your SASUSER.PROFILE catalog to the SASHELP.HOST catalog, as shown in the following example:

```
proc catalog c=sasuser.profile;
    copy out=sashelp.host entrytype=trantab;
run;
```

You can use PROC TRANTAB to modify translation tables stored in the SASHELP.HOST catalog only if you have update (or write) access to that data library and catalog.

Using Translation Tables Outside PROC TRANTAB

Using Translation Tables in the SORT Procedure

PROC SORT uses translation tables to determine the collating sequence to be used by the sort. You can specify an alternative translation table with the SORTSEQ= option of PROC SORT. For example, if your operating environment sorts with the EBCDIC sequence by default, and you want to sort with the ASCII sequence, you can issue the following statement to specify the ASCII translation table:

```
proc sort sortseq=ascii;
```

You can also create a customized translation table with PROC TRANTAB and specify the new table with PROC SORT. This is useful when you want to specify sorting sequences for languages other than U.S. English.

See Example 6 on page 1429 for an example that uses translation tables to sort data in different ways. For information on the tables available for sorting and the SORTSEQ= option, see Chapter 39, “The SORT Procedure,” on page 1091.

Using Translation Tables with the CPORT and CIMPORT Procedures

The CPORT and CIMPORT procedures use translation tables to translate characters in catalog entries that you export from one operating environment and import on another operating environment. You may specify the name of a SAS-supplied or a customized translation table in the TRANTAB statement of PROC CPORT. See “TRANTAB Statement” on page 315 in Chapter 13, “The CPORT Procedure,” on page 307 for more information.

Using Translation Tables with Remote Library Services

Remote Library Services (RLS) use translation tables to translate characters when you access remote data. SAS/CONNECT and SAS/SHARE software use translation tables to translate characters when you transfer or share files between two operating environments that use different encoding standards.

Using Translation Tables in SAS/GRAPH Software

In SAS/GRAPH software, translation tables are most commonly used on an IBM operating environment where tables are necessary because graphics commands must leave IBM operating environments in EBCDIC representation but must reach asynchronous graphics devices in ASCII representation. Specifically, SAS/GRAPH software builds the command stream for these devices internally in ASCII representation but must convert the commands to EBCDIC representation before they can be given to the communications software for transmission to the device. SAS/GRAPH software uses a translation table internally to make the initial conversion from ASCII to EBCDIC. The communications software then translates the command stream back to ASCII representation before it reaches the graphics device.

Translation tables are operating environment-specific. In most cases, you can simply use the default translation table, SASGTAB0, or one of the SAS-supplied graphics translation tables. However, if these tables are not able to do all of the translation correctly, you can create your own translation table with PROC TRANTAB. The SASGTAB0 table may fail to do the translation correctly when it encounters characters from languages other than U.S. English.

To specify an alternative translation table for SAS/GRAPH software, you can either use the TRANTAB= option in a GOPTIONS statement or modify the TRANTAB device parameter in the device entry. For example, the following GOPTIONS statement specifies the GTABTCAM graphics translation table:

```
goptions trantab=gtabtcam;
```

Translation tables used in SAS/GRAPH software perform both *device-to-operating environment* translation and *operating environment-to-device* translation. Therefore, a translation table is made up of 512 bytes, with the first 256 bytes used to perform device-to-operating environment translation (ASCII to EBCDIC on IBM mainframes) and the second 256 bytes used to perform operating environment-to-device translation (EBCDIC to ASCII on IBM mainframes). For PROC TRANTAB, the area of a translation table for device-to-operating environment translation is considered to be *table one*, and the area for operating environment-to-device translation is considered to be *table two*. See Example 1 on page 1419 for a listing of the ASCII translation table (a SAS-provided translation table), which shows both areas of the table.

On operating environments other than IBM mainframes, translation tables can be used to translate specific characters in the data stream that are created by the driver. For example, if the driver normally generates a vertical bar in the data stream, but you want another character to be generated in place of the vertical bar, you can create a translation table that translates the vertical bar to an alternate character.

For details on how to specify translation tables with the TRANTAB= option in SAS/GRAPH software, see *SAS/GRAPH Software: Reference, Version 6, First Edition, Volume 1* and *Volume 2*.

SAS/GRAPH software also uses key maps and device maps to map codes generated by the keyboard to specified characters and to map character codes to codes required by the graphics output device. These maps are specific to SAS/GRAPH software and are discussed in "The GKEYMAP Procedure" in *SAS/GRAPH Software: Reference*.

Syntax: TRANTAB Procedure

Tip: Supports RUN-group processing

```
PROC TRANTAB TABLE=table-name <NLS>;
  CLEAR <ONE | TWO | BOTH>;
  INVERSE;
  LIST <ONE | TWO | BOTH>;
  LOAD TABLE=table-name <NLS>;
  REPLACE position value-1<...value-n>;
  SAVE <TABLE=table-name> <ONE | TWO | BOTH>;
  SWAP;
```

To do this	Use this statement
Set all positions in the translation table to zero	CLEAR
Create an inverse of table one	INVERSE
Display a translation table in hexadecimal representation	LIST
Load a translation table into memory for editing	LOAD
Replace the characters in a translation table with specified values	REPLACE
Save the translation table in your SASUSER.PROFILE catalog	SAVE
Exchange table one with table two	SWAP

Note: PROC TRANTAB is an interactive procedure. Once you submit a PROC TRANTAB statement, you can continue to enter and execute statements without repeating the PROC TRANTAB statement. To terminate the procedure, submit a QUIT statement or submit another DATA or PROC statement. △

PROC TRANTAB Statement

Tip: If there is an incorrect table name in the PROC TRANTAB statement, use the LOAD statement to load the correct table. You do not need to reinvoke PROC TRANTAB. New tables are not stored in the catalog until you issue the SAVE statement, so you will not have unwanted tables in your catalog.

PROC TRANTAB TABLE=*table-name* <NLS>;

Required Arguments

TABLE=*table-name*

specifies the translation table to create, edit, or display. The specified table name must be a valid one-level SAS name with no more than 8 characters.

Options

NLS

specifies that the table you listed in the TABLE= argument is one of five special internal translation tables provided with every copy of the SAS System. You must use the NLS option when you specify one of the five special tables in the TABLE= argument:

SASXPT

the local-to-transport format translation table (used by the CPORT procedure)

SASLCL

the transport-to-local format translation table (used by the CIMPORT procedure)

SASUCS

the lowercase-to-uppercase translation table (used by the UPCASE function)

SASLCS

the uppercase-to-lowercase translation table (used by the LOWCASE macro)

SASCCL

the character classification table (used internally), which contains flag bytes that correspond to each character position that indicate the class or classes to which each character belongs.

NLS stands for National Language Support. This option and the associated translation tables provide a method to translate characters that exist in languages other than English. To make SAS use the modified NLS table, specify its name in the SAS system option TRANTAB= .

Note: When you load one of these special translation tables, the SAS log displays a note that states that table 2 is uninitialized. That is, table 2 is an empty table that contains all zeros. PROC TRANTAB does not use table 2 at all for translation in these special cases, so you do not need to be concerned about this note. Δ

CLEAR Statement

Sets all positions in the translation table to zero. This statement is useful when you create a new table.

CLEAR <ONE | TWO | BOTH>;

Options

ONE | TWO | BOTH

ONE
clears table one.

TWO
clears table two.

BOTH
clears both table one and table two.

Default: ONE

INVERSE Statement

Creates an inverse of table one in a translation table. That is, it creates table two.

Featured in: Example 5 on page 1427

INVERSE;

Details

INVERSE does not preserve multiple translations. Suppose table one has two (or more) different characters translated to the same value; for example, "A" and "B" are both translated to "1". For table two, INVERSE uses the last translated character for the value; that is, "1" is always translated to "B" and not "A", assuming that "A" appears before "B" in the first table.

Operating environment sort programs in the SAS System require an inverse table for proper operation.

LIST Statement

Displays a translation table in hexadecimal representation. The translation table listing appears in the SAS log.

Featured in: All examples

LIST <ONE | TWO | BOTH>;

Options

ONE | TWO | BOTH

ONE

displays table one.

TWO

displays table two.

BOTH

displays both table one and table two.

Default: ONE

LOAD Statement

Loads a translation table into memory for editing.

Tip: Use LOAD when you specify an incorrect table name in the PROC TRANTAB statement. You can specify the correct name without the need to reinvoke the procedure.

Tip: Use LOAD to edit multiple translation tables in a single PROC TRANTAB step. (Be sure to save the first table before you load another one.)

Featured in: Example 4 on page 1425

LOAD TABLE=*table-name* <NLS>;

Required Arguments

TABLE=*table-name*

specifies the name of an existing translation table to be edited. The specified table name must be a valid one-level SAS name.

Option

NLS

specifies that the table you listed in the TABLE= argument is one of five special internal translation tables provided with every copy of the SAS System. You must use the NLS option when you specify one of the five special tables in the TABLE= argument:

SASXPT

the local-to-transport format translation table

SASLCL

the transport-to-local format translation table

SASUCS

the lowercase-to-uppercase translation table

SASLCS

the uppercase-to-lowercase translation table

SASCCL

the character classification table, which contains flag bytes that correspond to each character position that indicate the class or classes to which each character belongs.

NLS stands for National Language Support. This option and the associated translation tables provide a method to map characters that exist in languages other than English to programs, displays, files, or products of the SAS System.

Note: When you load one of these special translation tables, the SAS log displays a note that states that table 2 is uninitialized. That is, table 2 is an empty table that contains all zeros. PROC TRANTAB does not use table 2 at all for translation in these special cases, so you do not need to be concerned about this note. Δ

REPLACE Statement

Replaces characters in a translation table with the values given, starting at the specified position.

Alias: REP

Tip: To save edits, you must issue the SAVE statement.

Featured in: Example 2 on page 1420, Example 3 on page 1422, and Example 4 on page 1425

REPLACE *position value-1<...value-n>;*

Required Arguments

position

specifies the position in a translation table where the replacement is to begin. The editable positions in a translation table begin at position decimal 0 and end at decimal 255. To specify the position, you can

- ☐ use a decimal or hexadecimal value to specify an actual location. If you specify a decimal value, for example, 20, PROC TRANTAB locates position 20 in the table, which is byte 21. If you specify a hexadecimal value, for example, '14'x, PROC TRANTAB locates the decimal position that is equivalent to the specified hexadecimal value, which in this case is position 20 (or byte 21) in the table.
- ☐ use a quoted character. PROC TRANTAB locates the quoted character in the table (that is, the quoted character's hexadecimal value) and uses that character's position as the starting position. For example, if you specify the following REPLACE statement, the statement replaces the first occurrence of the hexadecimal value for "a" and the next two hexadecimal values with the hexadecimal equivalent of "ABC":

```
replace 'a' 'ABC';
```

This is useful when you want to locate alphabetic and numerical characters when you do not know their actual location. If the quoted character is not found, PROC TRANTAB displays an error message and ignores the statement.

To edit positions 256 through 511 (table two), follow this procedure:

- 1 Issue the SWAP statement.
- 2 Issue the appropriate REPLACE statement.
- 3 Issue the SWAP statement again to reposition the table.

value-1 <...value-n>

is one or more decimal, hexadecimal, or character constants that give the actual value to be put into the table, starting at *position*. You can also use a mixture of the types of values. That is, you can specify a decimal, a hexadecimal, and a character value in one REPLACE statement. Example 3 on page 1422 shows a mixture of all three types of values in the REPLACE statement.

SAVE Statement

Saves the translation table in your SASUSER.PROFILE catalog.

Featured in: Example 2 on page 1420 and Example 4 on page 1425

SAVE <TABLE=*table-name*> <ONE|TWO|BOTH>;

Options

TABLE=*table-name*

specifies the table name under which the current table is to be saved. The name must be a valid one-level SAS name.

Default: If you omit the TABLE= option, the current table is saved under the name you specify in the PROC TRANTAB statement or the LOAD statement.

ONE | TWO | BOTH

ONE

saves table one.

TWO

saves table two.

BOTH

saves both table one and table two.

Default: BOTH

SWAP Statement

Exchanges table one with table two to enable you to edit positions 256 through 511.

Tip: After you edit the table, you must issue SWAP again to reposition the table.

Featured in: Example 7 on page 1431

SWAP;

Examples: TRANTAB Procedure

Note: All examples were produced in the UNIX environment. △

Example 1: Viewing a Translation Table

Procedure features:

LIST statement

This example uses PROC TRANTAB to display the SAS-supplied ASCII translation table.

Program

Set the options and specify a translation table.

```
options nodate pageno=1 linesize=80 pagesize=60;  
proc trantab table=ascii;
```

Display both halves of the translation table. The LIST BOTH statement displays both the table that provides the translation and the table that provides the inverse translation.

```
list both;
```

SAS Log

```
NOTE: Table specified is ASCII.
ASCII table 1:
      0 1 2 3 4 5 6 7 8 9 A B C D E F
00 '000102030405060708090A0B0C0D0E0F'x
10 '101112131415161718191A1B1C1D1E1F'x
20 '202122232425262728292A2B2C2D2E2F'x
30 '303132333435363738393A3B3C3D3E3F'x
40 '404142434445464748494A4B4C4D4E4F'x
50 '505152535455565758595A5B5C5D5E5F'x
60 '606162636465666768696A6B6C6D6E6F'x
70 '707172737475767778797A7B7C7D7E7F'x
80 '808182838485868788898A8B8C8D8E8F'x
90 '909192939495969798999A9B9C9D9E9F'x
A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAFAF'x
B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x

ASCII table 2:
      0 1 2 3 4 5 6 7 8 9 A B C D E F
00 '000102030405060708090A0B0C0D0E0F'x
10 '101112131415161718191A1B1C1D1E1F'x
20 '202122232425262728292A2B2C2D2E2F'x
30 '303132333435363738393A3B3C3D3E3F'x
40 '404142434445464748494A4B4C4D4E4F'x
50 '505152535455565758595A5B5C5D5E5F'x
60 '606162636465666768696A6B6C6D6E6F'x
70 '707172737475767778797A7B7C7D7E7F'x
80 '808182838485868788898A8B8C8D8E8F'x
90 '909192939495969798999A9B9C9D9E9F'x
A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAFAF'x
B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x
```

Example 2: Creating a Translation Table

Procedures features:

LIST statement
 REPLACE statement
 SAVE statement

This example uses PROC TRANTAB to create a customized translation table.

Program

Set the system options and specify the translation table to edit.

```
options nodate pageno=1 linesize=80 pagesize=60;
proc trantab table=newtable;
```

Replace characters in the translation table starting at a specified position. The REPLACE statement places the values in the table starting at position 0. You can use hexadecimal strings of any length in the REPLACE statement. This example uses strings of length 16 to match the way that translation tables appear in the SAS log.

```
replace 0
'00010203a309e57ff9ecc40b0c0d0e0f'x
'10111213a5e008e71819c6c51c1d1e1f'x
'c7fce9e2e40a171beaebe8efee050607'x
'c9e616f4f6f2fb04ffd6dca2b6a7501a'x
'20e1edf3faf1d1aababfa22e3c282b7c'x
'265facbdbcalabbb5f5f21242a293bac'x
'2d2f5fa6a6a6a62b2ba6a62c255f3e3f'x
'a62b2b2b2b2b2b2d2d603a2340273d22'x
'2b6162636465666768692d2ba6a62b2b'x
'2d6a6b6c6d6e6f7071722da62d2b2d2d'x
'2d7e737475767778787a2d2b2b2b2b2b'x
'2b2b2b5f5fa65f5f5fdf5fb65f5fb55f'x
'7b4142434445464748495f5f5f5f5f'x
'7d4a4b4c4d4e4f5051525f5f5fb15f5f'x
'5c83535455565758595a5f5ff75f5fb0'x
'30313233343536373839b75f6eb25f5f'x
;
```

Save the table. The SAVE statement saves the table under the name that is specified in the PROC TRANTAB statement. By default, the table is saved in your SASUSER.PROFILE catalog.

```
save;
```

Display both halves of the translation table in the SAS log. The LIST BOTH statement displays both the table that provides the translation and the table that provides the inverse translation.

```
list both;
```

SAS Log

Create and edit table 2. Table 2 is empty; that is, it consists entirely of 0s. To create table 2, you can use the INVERSE statement. (See Example 5 on page 1427.) To edit table 2, you can use the SWAP statement with the REPLACE statement. (See Example 7 on page 1431.)

```

NOTE: Table specified is NEWTABLE.
WARNING: Table NEWTABLE not found! New table is assumed.
NOTE: NEWTABLE table 1 is uninitialized.
NOTE: NEWTABLE table 2 is uninitialized.

NOTE: Saving table NEWTABLE.
NOTE: NEWTABLE table 2 will not be saved because it is uninitialized.
NEWTABLE table 1:
  0 1 2 3 4 5 6 7 8 9 A B C D E F
00 '00010203A309E57FF9ECC40B0C0D0E0F'x
10 '10111213A5E008E71819C6C51C1D1E1F'x
20 'C7FCE9E2E40A171BEAEBE8EFEE050607'x
30 'C9E616F4F6F2FB04FFD6DCA2B6A7501A'x
40 '20E1EDF3FAF1D1AABABFA22E3C282B7C'x
50 '265FACBDBC1ABBB5F5F21242A293BAC'x
60 '2D2F5FA6A6A6A62B2BA6A62C255F3E3F'x
70 'A62B2B2B2B2B2B2D2D2D603A2340273D22'x
80 '2B6162636465666768692D2BA6A62B2B'x
90 '2D6A6B6C6D6E6F7071722DA62D2B2D2D'x
A0 '2D7E737475767778787A2D2B2B2B2B'x
B0 '2B2B2B5F5FA65F5F5FDF5FB65F5FB55F'x
C0 '7B4142434445464748495F5F5F5F5F'x
D0 '7D4A4B4C4D4E4F5051525F5F5FB15F5F'x
E0 '5C83535455565758595A5F5FF75F5FB0'x
F0 '30313233343536373839B75F6EB25F5F'x

NOTE: NEWTABLE table 2 is uninitialized.
NEWTABLE table 2:
  0 1 2 3 4 5 6 7 8 9 A B C D E F
00 '00000000000000000000000000000000'x
10 '00000000000000000000000000000000'x
20 '00000000000000000000000000000000'x
30 '00000000000000000000000000000000'x
40 '00000000000000000000000000000000'x
50 '00000000000000000000000000000000'x
60 '00000000000000000000000000000000'x
70 '00000000000000000000000000000000'x
80 '00000000000000000000000000000000'x
90 '00000000000000000000000000000000'x
A0 '00000000000000000000000000000000'x
B0 '00000000000000000000000000000000'x
C0 '00000000000000000000000000000000'x
D0 '00000000000000000000000000000000'x
E0 '00000000000000000000000000000000'x
F0 '00000000000000000000000000000000'x

```

Example 3: Editing by Specifying a Decimal Value for Starting Position

Procedure features:

LIST statement

REPLACE statement

SAVE statement

This example edits the translation table created in Example 2 on page 1420. The decimal value specified in the REPLACE statement marks the starting position for the changes to the table.

The vertical arrow in both SAS logs marks the point at which the changes begin.

Program

Set the system options and specify which translation table to edit.

```
options nodate pageno=1 linesize=80 pagesize=60;
proc trantab table=newtable;
```

Display the original table. This LIST statement displays the original NEWTABLE translation table.

```
list one;
```

SAS Log

The Original NEWTABLE Translation Table

```
NOTE: Table specified is NEWTABLE.
NOTE: NEWTABLE table 2 is uninitialized.
NEWTABLE table 1:
      0 1 2 3 4 5 6 7 8 9 A B C D E F
      ↓
00 '00010203A309E57FF9ECC40B0C0D0E0F'x
10 '10111213A5E008E71819C6C51C1D1E1F'x
20 'C7FCE9E2E40A171BEAEBE8EFEE050607'x
30 'C9E616F4F6F2FB04FFD6DCA2B6A7501A'x
40 '20E1EDF3FAF1D1AABABFA22E3C282B7C'x
50 '265FACBDBCA1ABBB5F5F21242A293BAC'x
60 '2D2F5FA6A6A6A62B2BA6A62C255F3E3F'x
70 'A62B2B2B2B2B2B2D2D603A2340273D22'x
80 '2B6162636465666768692D2BA6A62B2B'x
90 '2D6A6B6C6D6E6F7071722DA62D2B2D2D'x
A0 '2D7E737475767778787A2D2B2B2B2B'x
B0 '2B2B2B5F5FA65F5F5FDF5FB65F5FB55F'x
C0 '7B4142434445464748495F5F5F5F5F'x
D0 '7D4A4B4C4D4E4F5051525F5F5FB15F5F'x
E0 '5C83535455565758595A5F5FF75F5FB0'x
F0 '30313233343536373839B75F6EB25F5F'x
```

Replace characters in the translation table starting at a specified position. The REPLACE statement starts at position decimal 10, which is byte 11 in the original table, and performs a byte-to-byte replacement with the given values.

```
replace 10
20 10 200 'x' 'ux' '092040'x;
```

Save your changes. The SAVE statement saves the changes that you made to the NEWTABLE translation table.

```
save;
```

Display the new table. The second LIST statement displays the edited NEWTABLE translation table.

```
list one;
```

SAS Log

The Edited NEWTABLE Translation Table

```
NOTE: Saving table NEWTABLE.
NOTE: NEWTABLE table 2 will not be saved because it is uninitialized.
NEWTABLE table 1:
      0 1 2 3 4 5 6 7 8 9 A B C D E F
      ↓
00 '00010203A309E57FF9EC140AC8787578'x
10 '09204013A5E008E71819C6C51C1D1E1F'x
20 'C7FCE9E2E40A171BEAEBE8EFEE050607'x
30 'C9E616F4F6F2FB04FFD6DCA2B6A7501A'x
40 '20E1EDF3FAF1D1AABABFA22E3C282B7C'x
50 '265FACBDBCA1ABBB5F5F21242A293BAC'x
60 '2D2F5FA6A6A6A62B2BA6A62C255F3E3F'x
70 'A62B2B2B2B2B2B2D2D603A2340273D22'x
80 '2B6162636465666768692D2BA6A62B2B'x
90 '2D6A6B6C6D6E6F7071722DA62D2B2D2D'x
A0 '2D7E737475767778787A2D2B2B2B2B2B'x
B0 '2B2B2B5F5FA65F5F5FDF5FB65F5FB55F'x
C0 '7B4142434445464748495F5F5F5F5F'x
D0 '7D4A4B4C4D4E4F5051525F5F5FB15F5F'x
E0 '5C83535455565758595A5F5FF75F5FB0'x
F0 '30313233343536373839B75F6EB25F5F'x
```

At position 10 (which is byte 11), a vertical arrow denotes the starting point for the changes to the translation table.

- At byte 11, decimal 20 (which is hexadecimal 14) replaces hexadecimal C4.
- At byte 12, decimal 10 (which is hexadecimal 0A) replaces hexadecimal 0B.
- At byte 13, decimal 200 (which is hexadecimal C8) replaces hexadecimal 0C.
- At byte 14, character 'x' (which is hexadecimal 78) replaces hexadecimal 0D.

- At bytes 15 and 16, characters 'ux' (which are hexadecimal 75 and 78, respectively) replace hexadecimal 0E and 0F.
- At bytes 17, 18, and 19, hexadecimal 092040 replaces hexadecimal 101112.

Example 4: Editing by Using a Quoted Character for Starting Position

Procedure features:

- LIST statement
- LOAD statement
- REPLACE statement
- SAVE statement

This example creates a new translation table by editing the SAS-supplied ASCII translation table. The first occurrence of the hexadecimal equivalent of the quoted character specified in the REPLACE statement is the starting position for the changes to the table. This differs from Example 3 on page 1422 in that you do not need to know the exact position at which to start the changes to the table. PROC TRANTAB finds the correct position for you.

The edited table is saved under a new name. Horizontal arrows in both SAS logs denote the edited rows in the translation table.

Program

Set the system options and specify which translation table to edit.

```
options nodate pageno=1 linesize=80 pagesize=60;
proc trantab table=ascii;
```

Display the translation table. The LIST statement displays the original translation table in the SAS log.

```
list one;
```

SAS Log

```
NOTE: Table specified is ASCII.
ASCII table 1:
      0 1 2 3 4 5 6 7 8 9 A B C D E F
00 '000102030405060708090A0B0C0D0E0F'x
10 '101112131415161718191A1B1C1D1E1F'x
20 '202122232425262728292A2B2C2D2E2F'x
30 '303132333435363738393A3B3C3D3E3F'x
40 '404142434445464748494A4B4C4D4E4F'x
50 '505152535455565758595A5B5C5D5E5F'x
60 '606162636465666768696A6B6C6D6E6F'x ←
70 '707172737475767778797A7B7C7D7E7F'x ←
80 '808182838485868788898A8B8C8D8E8F'x
90 '909192939495969798999A9B9C9D9E9F'x
A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAFAF'x
B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x
```

Replace characters in the translation table starting at a specified position. The REPLACE statement finds the first occurrence of the hexadecimal "a" (which is 61) and replaces it and the next 25 hexadecimal values with the hexadecimal values for uppercase "A" through "Z."

```
replace 'a' 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
```

Save your changes. The SAVE statement saves the changes made to the ASCII translation table under the new table name UPPER. The stored contents of the ASCII translation table remain unchanged.

```
save table=upper;
```

Load and display the translation table. The LOAD statement loads the edited translation table UPPER. The LIST statement displays the translation table UPPER in the SAS log.

```
load table=upper;
list one;
```

SAS Log

The UPPER Translation Table

The horizontal arrows in the SAS log denote the rows in which the changes are made.

```
NOTE: Table UPPER being loaded.
UPPER table 1:
      0 1 2 3 4 5 6 7 8 9 A B C D E F
00 '000102030405060708090A0B0C0D0E0F'x
10 '101112131415161718191A1B1C1D1E1F'x
20 '202122232425262728292A2B2C2D2E2F'x
30 '303132333435363738393A3B3C3D3E3F'x
40 '404142434445464748494A4B4C4D4E4F'x
50 '505152535455565758595A5B5C5D5E5F'x
60 '604142434445464748494A4B4C4D4E4F'x ←
70 '505152535455565758595A7B7C7D7E7F'x ←
80 '808182838485868788898A8B8C8D8E8F'x
90 '909192939495969798999A9B9C9D9E9F'x
A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'x
B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x
```

Example 5: Creating the Inverse of a Table

Procedure features:

INVERSE statement
LIST statement
SAVE statement

This example creates the inverse of the translation table that was created in Example 4 on page 1425. The new translation table created in this example is the operating environment-to-device translation for use in data communications.

Program

```
options nodate pageno=1 linesize=80 pagesize=60;
proc trantab table=upper;
```

Create the inverse translation table, save the tables, and display the tables. The INVERSE statement creates table 2 by inverting the original table 1 (called UPPER). The SAVE statement saves the translation tables. The LIST BOTH statement displays both the original translation table and its inverse.

```
inverse;
save;
list both;
```

SAS Log

The UPPER Translation Table and Its Inverse

The SAS log lists all the duplicate values that it encounters as it creates the inverse of table one. To conserve space, most of these messages are deleted in this example.

```
NOTE: Table specified is UPPER.
NOTE: This table cannot be mapped one to one.
duplicate of '41'x found at '61'x in table one.
duplicate of '42'x found at '62'x in table one.
duplicate of '43'x found at '63'x in table one.
.
.
.
duplicate of '58'x found at '78'x in table one.
duplicate of '59'x found at '79'x in table one.
duplicate of '5A'x found at '7A'x in table one.
NOTE: Saving table UPPER.
UPPER table 1:
      0 1 2 3 4 5 6 7 8 9 A B C D E F
00 '000102030405060708090A0B0C0D0E0F'x
10 '101112131415161718191A1B1C1D1E1F'x
20 '202122232425262728292A2B2C2D2E2F'x
30 '303132333435363738393A3B3C3D3E3F'x
40 '404142434445464748494A4B4C4D4E4F'x
50 '505152535455565758595A5B5C5D5E5F'x
60 '604142434445464748494A4B4C4D4E4F'x
70 '505152535455565758595A7B7C7D7E7F'x
80 '808182838485868788898A8B8C8D8E8F'x
90 '909192939495969798999A9B9C9D9E9F'x
A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'x
B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x

UPPER table 2:
      0 1 2 3 4 5 6 7 8 9 A B C D E F
00 '000102030405060708090A0B0C0D0E0F'x
10 '101112131415161718191A1B1C1D1E1F'x
20 '202122232425262728292A2B2C2D2E2F'x
30 '303132333435363738393A3B3C3D3E3F'x
40 '404142434445464748494A4B4C4D4E4F'x
50 '505152535455565758595A5B5C5D5E5F'x
60 '60000000000000000000000000000000'x
70 '000000000000000000000000007B7C7D7E7F'x
80 '808182838485868788898A8B8C8D8E8F'x
90 '909192939495969798999A9B9C9D9E9F'x
A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'x
B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x
```

The INVERSE statement lists in the SAS log all of the multiple translations that it encounters as it inverts the translation table. In Example 4 on page 1425, all the

lowercase letters were converted to uppercase in the translation table UPPER, which means that there are two sets of uppercase letters in UPPER. When INVERSE cannot make a translation, PROC TRANTAB fills the value with 00. Note that the inverse of the translation table UPPER has numerous 00 values.

Example 6: Using Different Translation Tables for Sorting

Procedure features:

PROC SORT statement option:

SORTSEQ=

Other features:

PRINT procedure

This example shows how to specify a different translation table to sort data in an order that is different from the default sort order. Characters that are written in a language other than U.S. English may require a sort order that is different from the default order.

Note: You can use the TRABASE program in the SAS Sample Library to create translation tables for several different languages. Δ

Program

Set the SAS system options.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the TESTSORT data set. The DATA step creates a SAS data set with four pairs of words, each pair differing only in the case of the first letter.

```
data testsort;
    input Values $10.;
    datalines;
Always
always
Forever
forever
Later
later
Yesterday
yesterday
;
```

Sort the data in an order that is different from the default sort order. PROC SORT sorts the data by using the default translation table, which sorts all lowercase words first, then all uppercase words.

```
proc sort;
  by values;
run;
```

Print the data set. PROC PRINT prints the sorted data set.

```
proc print noobs;
  title 'Default Sort Sequence';
run;
```

SAS Output

Output from Sorting Values with Default Translation Table

The default sort sequence sorts all the capitalized words in alphabetical order before it sorts any lowercase words.

Default Sort Sequence	1
Values	
Always	
Forever	
Later	
Yesterday	
always	
forever	
later	
yesterday	

Sort the data according to the translation table UPPER and print the new data set.

The SORTSEQ= option specifies that PROC SORT sort the data according to the customized translation table UPPER, which treats lowercase and uppercase letters alike. This is useful for sorting without regard for case. PROC PRINT prints the sorted data set.

```
proc sort sortseq=upper;
  by values;
run;
proc print noobs;
  title 'Customized Sort Sequence';
run;
```

SAS Output

Output from Sorting Values with Customized Translation Table

The customized sort sequence sorts all the words in alphabetical order, without regard for the case of the first letters.

Customized Sort Sequence	2
Values	
Always	
always	
Forever	
forever	
Later	
later	
Yesterday	
yesterday	

Example 7: Editing Table 1 and Table 2

Procedure features:

LIST statement
 REPLACE statement
 SAVE statement
 SWAP statement

This example shows how to edit both areas of a translation table. To edit positions 256 through 511 (table two), you must

- 1 Issue the SWAP statement to have table two change places with table one.
- 2 Issue an appropriate REPLACE statement to make changes to table two.
- 3 Issue the SWAP statement again to reposition the table.

Arrows in the SAS logs mark the rows and columns that are changed.

Program**Set the SAS system options and specify the translation table.**

```
options nodate pageno=1 linesize=80 pagesize=60;
proc trantab table=upper;
```

Display the original translation table. The LIST statement displays the original UPPER translation table.

```
list both;
```

SAS Log

The Original UPPER Translation Table

NOTE: Table specified is UPPER.

UPPER table 1:

```

      ↓
      0 1 2 3 4 5 6 7 8 9 A B C D E F
00 '000102030405060708090A0B0C0D0E0F'x ←
10 '101112131415161718191A1B1C1D1E1F'x
20 '202122232425262728292A2B2C2D2E2F'x
30 '303132333435363738393A3B3C3D3E3F'x
40 '404142434445464748494A4B4C4D4E4F'x
50 '505152535455565758595A5B5C5D5E5F'x
60 '604142434445464748494A4B4C4D4E4F'x
70 '505152535455565758595A7B7C7D7E7F'x
80 '808182838485868788898A8B8C8D8E8F'x
90 '909192939495969798999A9B9C9D9E9F'x
A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'x
B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
C0 'C0C1C2C3C4C5C6C7C8C9CACBCCDCECF'x
D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x

```

UPPER table 2:

```

      ↓
      0 1 2 3 4 5 6 7 8 9 A B C D E F
00 '000102030405060708090A0B0C0D0E0F'x ←
10 '101112131415161718191A1B1C1D1E1F'x
20 '202122232425262728292A2B2C2D2E2F'x
30 '303132333435363738393A3B3C3D3E3F'x
40 '404142434445464748494A4B4C4D4E4F'x
50 '505152535455565758595A5B5C5D5E5F'x
60 '60000000000000000000000000000000'x
70 '00000000000000000000000007B7C7D7E7F'x
80 '808182838485868788898A8B8C8D8E8F'x
90 '909192939495969798999A9B9C9D9E9F'x
A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'x
B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
C0 'C0C1C2C3C4C5C6C7C8C9CACBCCDCECF'x
D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x

```

Replace characters in the translation table starting at a specified position. The REPLACE statement starts at position 1 and replaces the current value of 01 with '0A'.

```
replace 1 '0A'x;
```

Prepare table 2 to be edited. The first SWAP statement positions table 2 so that it can be edited. The second REPLACE statement makes the same change in table 2 that was made in table 1.

```
swap;
replace 1 '0A'x;
```

Save and display the tables in their original positions. The second SWAP statement restores tables 1 and table 2 to their original positions. The SAVE statement saves both areas of the translation table by default. The LIST statement displays both areas of the table.

```
swap;
save;
list both;
```

SAS Log

The Edited UPPER Translation Table

In byte 2, in both areas of the translation table, hexadecimal value '0A' replaces hexadecimal value 01. Arrows denote the rows and columns of the table in which this change is made.

NOTE: Table specified is UPPER.
UPPER table 1:

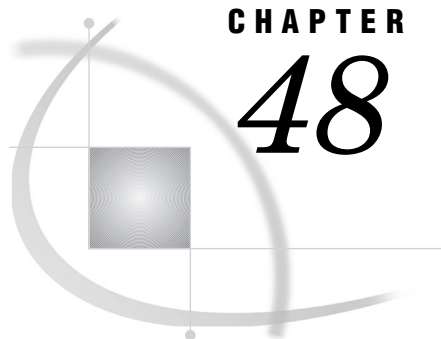
```

      ↓
      0 1 2 3 4 5 6 7 8 9 A B C D E F
00 '000A02030405060708090A0B0C0D0E0F'x ←
10 '101112131415161718191A1B1C1D1E1F'x
20 '202122232425262728292A2B2C2D2E2F'x
30 '303132333435363738393A3B3C3D3E3F'x
40 '404142434445464748494A4B4C4D4E4F'x
50 '505152535455565758595A5B5C5D5E5F'x
60 '604142434445464748494A4B4C4D4E4F'x
70 '505152535455565758595A7B7C7D7E7F'x
80 '808182838485868788898A8B8C8D8E8F'x
90 '909192939495969798999A9B9C9D9E9F'x
A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAFAF'x
B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x
```

UPPER table 2:

```

      ↓
      0 1 2 3 4 5 6 7 8 9 A B C D E F
00 '000A02030405060708090A0B0C0D0E0F'x ←
10 '101112131415161718191A1B1C1D1E1F'x
20 '202122232425262728292A2B2C2D2E2F'x
30 '303132333435363738393A3B3C3D3E3F'x
40 '404142434445464748494A4B4C4D4E4F'x
50 '505152535455565758595A5B5C5D5E5F'x
60 '60000000000000000000000000000000'x
70 '00000000000000000000000000000007B7C7D7E7F'x
80 '808182838485868788898A8B8C8D8E8F'x
90 '909192939495969798999A9B9C9D9E9F'x
A0 'A0A1A2A3A4A5A6A7A8A9AAABACADAFAF'x
B0 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'x
C0 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'x
D0 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'x
E0 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'x
F0 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'x
```

CHAPTER

48

The UNIVARIATE Procedure

<i>Overview: UNIVARIATE Procedure</i>	1436
<i>Syntax: UNIVARIATE Procedure</i>	1442
<i>PROC UNIVARIATE Statement</i>	1443
<i>BY Statement</i>	1451
<i>CLASS Statement</i>	1451
<i>FREQ Statement</i>	1454
<i>HISTOGRAM Statement</i>	1455
<i>ID Statement</i>	1473
<i>INSET Statement</i>	1473
<i>OUTPUT Statement</i>	1482
<i>PROBPLOT Statement</i>	1485
<i>QQPLOT Statement</i>	1497
<i>VAR Statement</i>	1510
<i>WEIGHT Statement</i>	1510
<i>Concepts: UNIVARIATE Procedure</i>	1511
<i>Rounding</i>	1511
<i>Generating Line Printer Plots</i>	1512
<i>Stem-and-Leaf Plot</i>	1512
<i>Box Plot</i>	1513
<i>Normal Probability Plot</i>	1513
<i>Side-by-Side Box Plots</i>	1514
<i>Generating High-Resolution Graphics</i>	1514
<i>Quantile-Quantile and Probability Plots</i>	1514
<i>Interpreting Quantile-Quantile and Probability Plots</i>	1515
<i>Determining Computer Resources</i>	1516
<i>Statistical Computations: UNIVARIATE Procedure</i>	1517
<i>Confidence Limits for Parameters of the Normal Distribution</i>	1517
<i>Tests for Location</i>	1518
<i>Student's t Test</i>	1518
<i>Sign Test</i>	1519
<i>Wilcoxon Signed Rank Test</i>	1519
<i>Goodness-of-Fit Tests</i>	1520
<i>Shapiro-Wilk Statistic</i>	1521
<i>EDF Goodness-of-Fit Tests</i>	1521
<i>Kolmogorov D Statistic</i>	1522
<i>Anderson-Darling Statistic</i>	1522
<i>Cramer-von Mises Statistic</i>	1523
<i>Probability Values of EDF Tests</i>	1523
<i>Robust Estimators</i>	1525
<i>Winsorized Means</i>	1525
<i>Trimmed Means</i>	1526

Robust Measures of Scale	1527
Calculating Percentiles	1528
Confidence Limits for Quantiles	1528
Weighted Quantiles	1529
Calculating the Mode	1530
Formulas for Fitted Continuous Distributions	1530
Beta Distribution	1530
Exponential Distribution	1531
Gamma Distribution	1532
Lognormal Distribution	1533
Normal Distribution	1533
Weibull Distribution	1534
Kernel Density Estimates	1534
Theoretical Distributions for Quantile-Quantile and Probability Plots	1536
Beta Distribution	1536
Exponential Distribution	1537
Gamma Distribution	1537
Lognormal Distribution	1537
Normal Distribution	1538
Three-Parameter Weibull Distribution	1538
Two-Parameter Weibull Distribution	1538
Shape Parameters	1539
Location and Scale Parameters	1539
Results: UNIVARIATE Procedure	1540
Missing Values	1540
Histograms	1541
Histogram Intervals	1541
Quantiles	1541
ODS Table Names	1541
Output Data Set	1542
OUTHISTOGRAM= Data Set	1543
Examples: UNIVARIATE Procedure	1543
Example 1: Univariate Analysis for Multiple Variables	1543
Example 2: Identifying Extreme Values and Creating a Histogram	1546
Example 3: Computing Robust Estimators	1549
Example 4: Performing a Sign Test Using Paired Data	1552
Example 5: Examining the Data Distribution and Saving Percentiles	1555
Example 6: Creating an Output Data Set with Multiple Analysis Variables	1560
Example 7: Fitting Density Curves	1561
Example 8: Displaying a Reference Line on a Normal Quantile-Quantile Plot	1566
Example 9: Creating a Two-Way Comparative Histogram	1568
References	1572

Overview: UNIVARIATE Procedure

k

The UNIVARIATE procedure provides a variety of descriptive measures, high-resolution graphical displays, and statistical methods to summarize, visualize, analyze, and model the statistical distributions of numeric variables. These tools are useful for a broad range of tasks and applications:

- Exploring the distributions of the variables in a data set is an important preliminary step in data analysis, data warehousing, and data mining. With the UNIVARIATE procedure you can use tables and graphical displays, such as

histograms and nonparametric density estimates, to find key features of distributions, identify outliers and extreme observations, determine the need for data transformations, and compare distributions.

- Modeling the distributions of data and validating distributional assumptions are basic steps in statistical analysis. You can use the UNIVARIATE procedure to fit parametric distributions (beta, exponential, gamma, lognormal, normal, and Weibull) and to compute probabilities and percentiles from these models. You can assess goodness of fit with hypothesis tests and with graphical displays such as probability plots and quantile-quantile plots. You can also use the UNIVARIATE procedure to validate distributional assumptions for other types of statistical analysis. When standard assumptions are not met, you can use the UNIVARIATE procedure to perform nonparametric tests and compute robust estimates of location and scale.
- Summarizing the distribution of the data is often helpful for creating effective statistical reports and presentations. You can use the UNIVARIATE procedure to create tables of summary measures, such as means and percentiles, together with graphical displays, such as histograms and comparative histograms, which facilitate the interpretation of the report.

The following examples illustrate a few of the tables and plots that you can produce with the UNIVARIATE procedure.

The following output shows a table of summary measures and a table of extreme observations for the loan-to-value ratios of 5,840 home mortgages in the data set HOMELOANS. The DATA step that creates the HomeLoans data set is shown in “HOMELOANS” on page 1627. The statements that produce this output follow:

```
ods select BasicMeasures ExtremeObs;

proc univariate data=homeloans;
    var LoanToValueRatio;
run;
```

Note: The ODS SELECT statement restricts the default output so that only the relevant tables appear in the preliminary analysis of this data. Δ

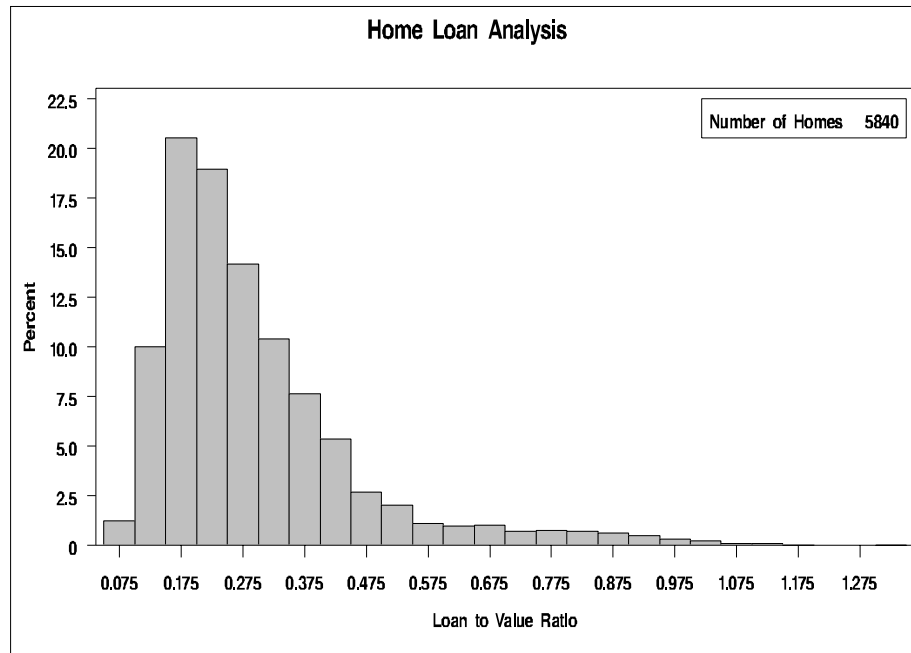
Output 48.1

The SAS System				
The UNIVARIATE Procedure				
Variable: LoanToValueRatio (Loan to Value Ratio)				
Basic Statistical Measures				
Location		Variability		
Mean	0.292512	Std Deviation	0.16476	
Median	0.248050	Variance	0.02715	
Mode	0.250000	Range	1.24780	
		Interquartile Range	0.16419	
Extreme Observations				
-----Lowest-----		-----Highest-----		
Value	Obs	Value	Obs	
0.0651786	1	1.13976	5776	
0.0690157	3	1.14209	5791	
0.0699755	59	1.14286	5801	
0.0702412	84	1.17090	5799	
0.0704787	4	1.31298	5811	

Figure 48.1 on page 1439 shows a histogram that enables you to visualize the distribution of loan-to-value ratios. The histogram reveals features of the distribution, such as its skewness and the peak at 0.175, which are not evident from the tables in the previous example. The following statements create the histogram:

```
proc univariate data=homeloans noprint;
  histogram LoanToValueRatio / cfill=ltgray;
  inset n = 'Number of Homes' / position=ne;
  title1 'Home Loan Analysis';
run;
```

Note: The NOPRINT option suppresses the display of summary statistics. The INSET statement inserts the total number of analyzed home loans in the upper-right corner of the plot. \triangle

Figure 48.1 Histogram for Loan-to-Value Ratio

The following output shows the quantiles for the distributions of loan-to-value ratios for two types of loans. Figure 48.2 on page 1441 shows a comparative histogram, which enables you to compare the two distributions more easily. The following statements produce the output and histogram:

```
ods select Quantiles;

proc univariate data=HomeLoans;
  var LoanToValueRatio;
  class LoanType;
  histogram LoanToValueRatio / cfill=ltgray
                                kernel(color=black);
  inset n='Number of Homes' median='Median Ratio'(5.3) / position=ne;
  title 'Comparison of Loan Types';
  label LoanType = 'Type of Loan';
run;
```

Note: The CLASS statement specifies LoanType as a classification variable for the comparative histogram. The KERNEL option adds a smooth nonparametric estimate of the ratio density to each histogram. △

Output 48.2

```

Comparison of Loan Types

The UNIVARIATE Procedure
Variable: LoanToValueRatio (Loan to Value Ratio)
         LoanType = Gold

Quantiles (Definition 5)

Quantile      Estimate
100% Max      1.0617647
99%           0.8974576
95%           0.6385908
90%           0.4471369
75% Q3        0.2985099
50% Median    0.2217033
25% Q1        0.1734568
10%           0.1411130
5%            0.1213079
1%            0.0942167
0% Min        0.0651786

```

```

Comparison of Loan Types

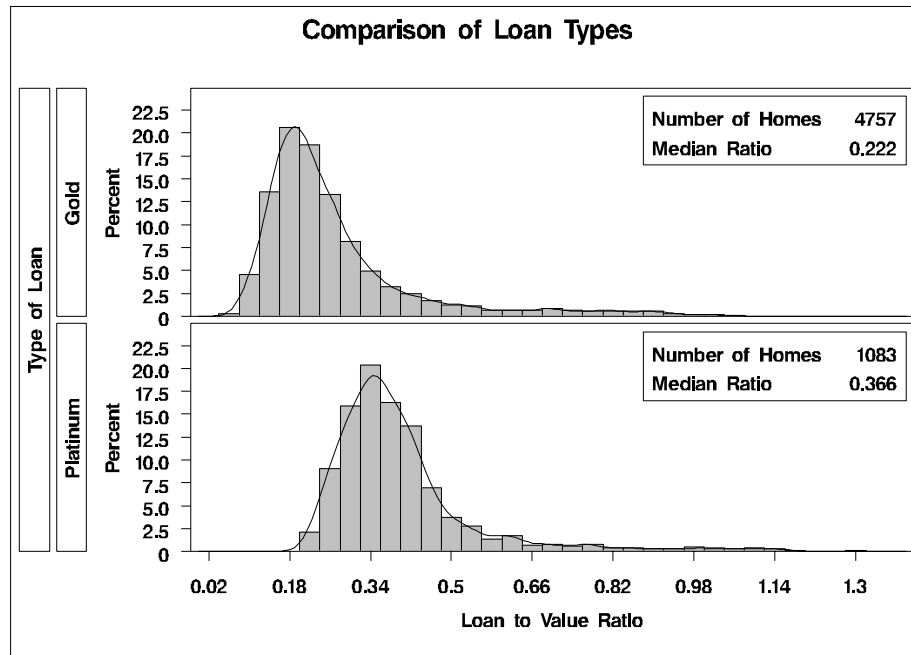
The UNIVARIATE Procedure
Variable: LoanToValueRatio (Loan to Value Ratio)
         LoanType = Platinum

Quantiles (Definition 5)

Quantile      Estimate
100% Max      1.312981
99%           1.050000
95%           0.691803
90%           0.549273
75% Q3        0.430160
50% Median    0.366168
25% Q1        0.314452
10%           0.273670
5%            0.253124
1%            0.231114
0% Min        0.215504

```

Figure 48.2 Comparative Histogram for Loan-to-Value Ratio



In addition to summarizing the distribution of population data, you can use PROC UNIVARIATE to analyze the distribution of sample data. The following output shows an analysis of the distribution for measurements of position deviation in a sample of 30 aircraft components. Because this small sample is from an unknown population, an initial question is whether the sample measurements are from a normal distribution. The DATA step that creates the Aircraft data set is shown in “AIRCRAFT” on page 1615. The goodness-of-fit tests reject the hypothesis that the measurements are normally distributed. Figure 48.3 on page 1442 shows a normal probability plot for these measurements. The curved point pattern suggests that a skewed distribution, such as the lognormal, may be more appropriate than the normal distribution, which the diagonal reference line represents. The following statements produce the output and probability plot:

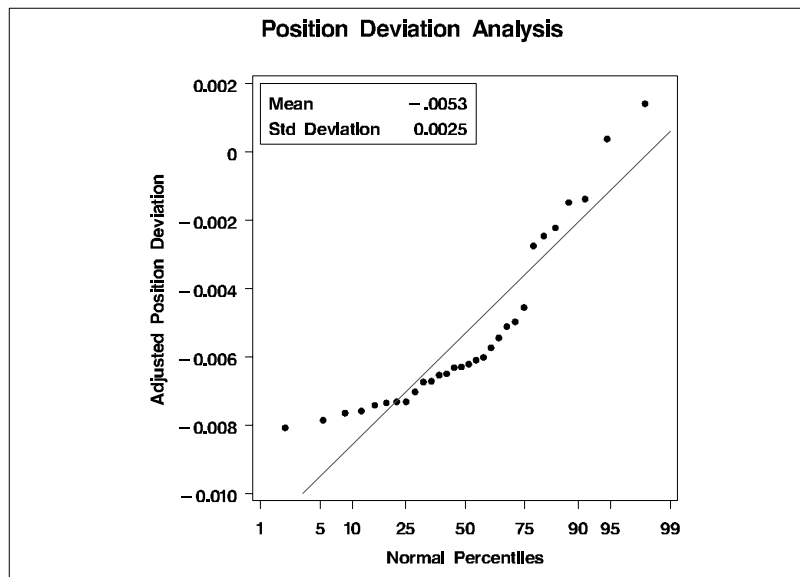
```
ods select Moments TestsForNormality;
symbol v=dot;

proc univariate data=aircraft normaltest;
  var Deviation;
  probplot Deviation / normal (mu=est sigma=est) square;
  label Deviation = 'Adjusted Position Deviation';
  inset mean std / format=6.4;
  title 'Position Deviation Analysis';
run;
```

Output 48.3

Position Deviation Analysis				1
The UNIVARIATE Procedure				
Variable: Deviation (Position Deviation - Adjusted Tolerance)				
Moments				
N	30	Sum Weights		30
Mean	-0.0053067	Sum Observations		-0.1592
Std Deviation	0.00254362	Variance		6.47002E-6
Skewness	1.2562507	Kurtosis		0.69790426
Uncorrected SS	0.00103245	Corrected SS		0.00018763
Coeff Variation	-47.932613	Std Error Mean		0.0004644
Tests for Normality				
Test	--Statistic--		-----p Value-----	
Shapiro-Wilk	W	0.845364	Pr < W	0.0005
Kolmogorov-Smirnov	D	0.208921	Pr > D	<0.0100
Cramer-von Mises	W-Sq	0.329274	Pr > W-Sq	<0.0050
Anderson-Darling	A-Sq	1.784881	Pr > A-Sq	<0.0050

Figure 48.3 Normal Probability Plot



Syntax: UNIVARIATE Procedure

Tip: Supports the Output Delivery System. See “Output Delivery System” on page 32 for details.

Reminder: You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, “Statements with the Same Function in Multiple Procedures,” on page 53 for details. You can also use any global statements as well. See “Global Statements” on page 18 for a list.

```

PROC UNIVARIATE <option(s)>;
  BY <DESCENDING> variable-1 <...<DESCENDING> variable-n>
    <NOTSORTED>;
  CLASS variable-1<(variable-option(s))> <variable-2<(variable-option(s))>>
    </ KEYLEVEL='value1'|('value1' 'value2')>;
  FREQ variable;
  HISTOGRAM <variable(s)> </ option(s)>;
  ID variable(s);
  INSET <keyword(s) DATA=SAS-data-set> </ option(s)>;
  OUTPUT <OUT=SAS-data-set> statistic-keyword-1=name(s)
    <... statistic-keyword-n=name(s)> <percentiles-specification>;
  PROBPLOT <variable(s)> </ option(s)>;
  QQPLOT <variable(s)> </ option(s)>;
  VAR variable(s);
  WEIGHT variable;

```

To do this	Use this statement
Calculate separate statistics for each BY group	BY
Specify up to two class variables to categorize the analysis	CLASS
Specify a variable that contains the frequency of each observation	FREQ
Create a high-resolution graph of a histogram	HISTOGRAM
Specify one or more variables whose values identify the extreme observations	ID
Inset a table of summary statistics in a high-resolution graph	INSET
Create an output data set that contains specified statistics	OUTPUT
Create a high-resolution graph of a probability plot	PROBPLOT
Create a high-resolution graph of a quantile-quantile plot	QQPLOT
Select the analysis variables and determine their order in the report	VAR
Identify a variable whose values weight each observation in the statistical calculations	WEIGHT

PROC UNIVARIATE Statement

```

PROC UNIVARIATE <option(s)>;

```

To do this	Use this option
Specify the input data set	DATA=
Specify the input data set that contains annotate variables	ANNOTATE=
Specify the SAS catalog to save high-resolution graphics output	GOUT=
Control the statistical analysis	
Request all statistics and tables that the FREQ, MODES, NEXTRVAL=, PLOT, and CIBASIC options generate	ALL
Specify the confidence level for the confidence limits	ALPHA=
Request confidence limits for the mean, standard deviation, and variance based on normally distributed data	CIBASIC
Request confidence limits for quantiles using a distribution-free method	CIPCTLDF
Request confidence limits for quantiles based on normally distributed data	CIPCTLNORMAL
Exclude observations with nonpositive weights from the analysis	EXCLNPWGT
Specify the value of the mean or location parameter	MU0=
Specify the number of extreme observations displayed	NEXTROBS=
Specify the number of extreme values displayed	NEXTRVAL=
Request tests for normality	NORMAL
Specify the mathematical definition used to compute quantiles	PCTLDEF=
Compute robust estimates of scale	ROBUSTSCALE
Specify the units to round the analysis variable prior to computing statistics	ROUND=
Compute trimmed means	TRIMMED=
Specify the variance divisor	VARDEF=
Compute Winsorized means	WINSORIZED=
Control the displayed output	
Request a frequency table	FREQ
Request a table that shows number of observations greater than, equal to, and less than MU0=	LOCCOUNT
Request a table of all possible modes	MODES
Suppress side-by-side plots	NOBYPLOT
Suppress tables of descriptive statistics	NOPRINT
Create low-resolution stem-and-leaf, box, and normal probability plots	PLOTS
Specify the approximate number of rows the plots use	PLOTSIZE=

Options

ALL

requests all statistics and tables that the FREQ, MODES, NEXTRVAL=5, PLOT, and CIBASIC options generate. If the analysis variables are not weighted, this option also requests the statistics and tables that the CIPCTLDF, CIPCTLNORMAL, LOCCOUNT, NORMAL, ROBUSTCALE, TRIMMED=.25, and WINSORIZED=.25 options generate. PROC UNIVARIATE also uses any values that you specify for ALPHA=, MU0=, NEXTRVAL=, CIBASIC, CIPCTLDF, CIPCTLNORMAL, TRIMMED=, or WINSORIZED= to produce the output.

ALPHA=value

specifies the default confidence level to compute confidence limits. The percentage for the confidence limits is $(1 - \text{value}) \times 100$. For example, ALPHA=.05 results in a 95 percent confidence limit.

Default: .05

Range: between 0 and 1

Main discussion: “Confidence Limits for Parameters of the Normal Distribution” on page 1517

Featured in: Example 5 on page 1555

ANNOTATE=SAS-data-set

specifies an input data set that contains annotate variables as described in *SAS/GRAPH Reference*. You can use this data set to add features to your high-resolution graphics. PROC UNIVARIATE adds the features in this data set to every high-resolution graph that is produced in the PROC step.

Alias: ANNO=

Interaction: PROC UNIVARIATE does not use the ANNOTATE= data set unless you create a high-resolution graph with the HISTOGRAM, PROBLOT, or QQPLOT statement.

Tip: Use the ANNOTATE= option in the HISTOGRAM, PROBLOT, or QQPLOT statement if you want to add a feature to a specific graphics display.

CIBASIC(<(<TYPE=keyword> <ALPHA=value>)>)

requests confidence limits for the mean, standard deviation, and variance based on the assumption that the data are normally distributed. For large sample sizes, this assumption is not required for the mean because of the Central Limit Theorem.

TYPE=keyword

specifies the type of confidence limit, where *keyword* is LOWER, UPPER, or TWOSIDED.

Default: TWOSIDED

ALPHA=value

specifies the confidence level to compute the confidence limit. The percentage for the confidence limits is $(1 - \text{value}) \times 100$. For example, ALPHA=.05 results in a 95 percent confidence limit.

Default: The value of ALPHA= in the PROC statement

Range: between 0 and 1

Requirement: You must use the default value of VARDEF=, which is DF.

Main discussion: “Confidence Limits for Parameters of the Normal Distribution” on page 1517

Featured in: Example 4 on page 1552 and Example 5 on page 1555

CIPCTLDF(<(<TYPE=keyword> <ALPHA=value>)>)

requests confidence limits for quantiles by using a method that is distribution-free. In other words, no specific parametric distribution such as the normal is assumed for

the data. PROC UNIVARIATE uses order statistics (ranks) to compute the confidence limits as described by Hahn and Meeker (1991).

TYPE=*keyword*

specifies the type of confidence limit, where *keyword* is LOWER, UPPER, SYMMETRIC, or ASYMMETRIC.

Default: SYMMETRIC

ALPHA=*value*

specifies the confidence level to compute the confidence limit. The percentage for the confidence limits is $(1 - \text{value}) \times 100$. For example, ALPHA=.05 results in a 95 percent confidence limit.

Default: The value of ALPHA= in the PROC statement

Range: between 0 and 1

Alias: CIQUANTDF

Restriction: This option is not available if you specify a WEIGHT statement.

Main discussion: “Confidence Limits for Quantiles” on page 1528

Featured in: Example 4 on page 1552

CIPCTLNORMAL <(<TYPE=*keyword*> <ALPHA=*value*>)>

requests confidence limits for quantiles based on the assumption that the data are normally distributed.

TYPE=*keyword*

specifies the type of confidence limit, where *keyword* is LOWER, UPPER, or TWOSIDED.

Default: TWOSIDED

ALPHA=*value*

specifies the confidence level to compute the confidence limit. The percentage for the confidence limits is $(1 - \text{value}) \times 100$. For example, ALPHA=.05 results in a 95 percent confidence limit.

Default: The value of ALPHA= in the PROC statement

Range: between 0 and 1

Alias: CIQUANTNORMAL

Requirement: You must use the default value of VARDEF=, which is DF.

Restriction: This option is not available if you specify a WEIGHT statement.

Main discussion: “Confidence Limits for Quantiles” on page 1528

DATA=SAS-*data-set*

specifies the input SAS data set.

Main discussion: “Input Data Sets” on page 19

EXCLNPWGT

excludes observations with nonpositive weight values (zero or negative) from the analysis. By default, PROC UNIVARIATE treats observations with negative weights like those with zero weights and counts them in the total number of observations.

Requirement: You must use a WEIGHT statement.

See also: “WEIGHT Statement” on page 1510

FREQ

requests a frequency table that consists of the variable values, frequencies, cell percentages, and cumulative percentages.

Interaction: If you specify the WEIGHT statement, PROC UNIVARIATE includes the weighted count in the table and uses this value to compute the percentages.

GOUT=graphics-catalog

specifies the SAS catalog that PROC UNIVARIATE uses to save the high-resolution graphics output.

Tip: If you omit the libref, PROC UNIVARIATE looks for the catalog in the temporary library called WORK and creates the catalog if it does not exist.

See also: For information on storing graphics output in SAS catalogs, see *SAS/GRAPH Reference*

LOCCOUNT

requests a table that shows the number of observations greater than, not equal to, and less than the value of MU0=. PROC UNIVARIATE uses these values to construct the sign test and the signed rank test.

Restriction: This option is not available if you specify a WEIGHT statement.

See also: MU0= on page 1447

Featured in: Example 4 on page 1552

MODES

requests a table of all possible modes. By default, when the data contain multiple modes, PROC UNIVARIATE displays the lowest mode in the table of basic statistical measures. When all the values are unique, PROC UNIVARIATE does not produce a table of modes.

Alias: MODE

Main discussion: “Calculating the Mode” on page 1530

Featured in: Example 4 on page 1552

MU0=value(s)

specifies the value of the mean or location parameter (μ_0) in the null hypothesis for tests of location. If you specify one value, PROC UNIVARIATE tests the same null hypothesis for all analysis variables. If you specify multiple values, a VAR statement is required, and PROC UNIVARIATE tests a different null hypothesis for each analysis variable in the corresponding order.

Alias: LOCATION=

Default: 0

Main discussion: “Tests for Location” on page 1518

Example statement: The following statement tests if the mean of the first variable equals 0 and the mean of the second variable equals 0.5.

```
proc univariate mu0=0 0.5;
```

Featured in: Example 5 on page 1555

NEXTROBS=n

specifies the number of extreme observations that PROC UNIVARIATE lists in the table of extreme observations. The table lists the n lowest observations and the n highest observations.

Default: 5

Range: an integer between 0 and the half the maximum number of observations

Tip: Use NEXTROBS=0 to suppress the table of extreme observations.

Featured in: Example 2 on page 1546

NEXTRVAL=n

specifies the number of extreme values that PROC UNIVARIATE lists in the table of extreme values. The table lists the n lowest unique values and the n highest unique values.

Default: 0

Range: an integer between 0 and half the maximum number of observations

Featured in: Example 2 on page 1546

NOBYPLOT

suppresses side-by-side box plots when you use the BY statement and the ALL option or the PLOT option in the PROC statement.

NOPRINT

suppresses all the tables of descriptive statistics that the PROC UNIVARIATE statement creates. NOPRINT does not suppress the tables that the HISTOGRAM statement creates.

Tip: Use NOPRINT when you want to create an OUT= output data set only.

Featured in: Example 6 on page 1560 and Example 7 on page 1561

NORMAL

requests tests for normality that include the Shapiro-Wilk test and a series of goodness-of-fit tests based on the empirical distribution function.

Alias: NORMALTEST

Restriction: This option is not available if you specify a WEIGHT statement.

Main discussion: “Goodness-of-Fit Tests” on page 1520

Featured in: Example 5 on page 1555

PCTLDEF=*value*

specifies the definition that PROC UNIVARIATE uses to calculate quantiles.

Alias: DEF=

Default: 5

Range: 1, 2, 3, 4, 5

Restriction: You cannot use PCTLDEF= when you compute weighted quantiles.

Main discussion: “Percentile and Related Statistics” on page 1583

PLOTS

produces a stem-and-leaf plot (or a horizontal bar chart), a box plot, and a normal probability plot. If you use a BY statement, side-by-side box plots that are labeled **Schematic Plots** appear after the univariate analysis for the last BY group.

Alias: PLOT

Main discussion: “Generating Line Printer Plots” on page 1512

Featured in: Example 5 on page 1555

PLOTSIZE=*n*

specifies the approximate number of rows that the plots use. If *n* is larger than the value of the SAS system option PAGESIZE=, PROC UNIVARIATE uses the value of PAGESIZE=. If *n* is less than eight, PROC UNIVARIATE uses eight rows to draw the plots.

Default: the value of PAGESIZE=

Range: 8 to the value of PAGESIZE=

ROBUSTSCALE

produces a table with robust estimates of scale. The statistics include the interquartile range, Gini’s mean difference, the median absolute deviation about the median (*MAD*), and two statistics proposed by Rousseeuw and Croux (1993), Q_n , and S_n .

Restriction: This option is not available if you specify a WEIGHT statement.

Main discussion: “Robust Measures of Scale” on page 1527

ROUND=unit(s)

specifies the units to use to round the analysis variables prior to computing statistics. If you specify one unit, PROC UNIVARIATE uses this unit to round all analysis variables. If you specify multiple units, a VAR statement is required, and each unit rounds the values of the corresponding analysis variable. If ROUND=0, no rounding occurs.

Default: 0

Tip: ROUND= reduces the number of unique variable values, thereby reducing the memory requirements.

Range: ≥ 0

Main discussion: “Rounding” on page 1511

Example statement: To make 1 the rounding unit for the first analysis variable and 0.5 the rounding unit for second analysis variable, submit the statement

```
proc univariate round=1 0.5;
```

TRIMMED=value(s) <(<TYPE=keyword> <ALPHA=value>)>

requests a table of trimmed means, where *value* specifies the number or the proportion of observations that PROC UNIVARIATE trims. If *value* is a proportion p between 0 and .5, the number of observations that PROC UNIVARIATE trims is the smallest integer that is greater than or equal to np , where n is the number of observations.

TYPE=*keyword*

specifies the type of confidence limit for the mean, where *keyword* is LOWER, UPPER, or TWOSIDED.

Default: TWOSIDED

ALPHA=*value*

specifies the confidence level to compute the confidence limit. The percentage for the confidence limits is $(1 - \text{value}) \times 100$. For example, ALPHA=.05 results in a 95 percent confidence limit.

Default: The value of ALPHA= in the PROC statement

Range: between 0 and 1

Alias: TRIM=

Range: between 0 and half the number of nonmissing observations. When a proportion is specified, *value* must be less than .5.

Requirement: To compute confidence limits for the mean and the Student’s t test, you must use the default value of VARDEF=, which is DF.

Restriction: This option is not available if you specify a WEIGHT statement.

Main discussion: “Trimmed Means” on page 1526

Featured in: Example 3 on page 1549

VARDEF=divisor

specifies the divisor to use in the calculation of variances and standard deviation. Table 48.1 on page 1449 shows the possible values for *divisor* and associated divisors.

Table 48.1 Possible Values for VARDEF=

Value	Divisor	Formula for Divisor
DF	degrees of freedom	$n - 1$
N	number of observations	n

Value	Divisor	Formula for Divisor
WDF	sum of weights minus one	$(\sum_i w_i) - 1$
WEIGHT WGT	sum of weights	$\sum_i w_i$

The procedure computes the variance as $CSS/divisor$, where CSS is the corrected sums of squares and equals $\sum (x_i - \bar{x})^2$. When you weight the analysis variables, CSS equals $\sum w_i (x_i - \bar{x}_w)^2$, where \bar{x}_w is the weighted mean.

Default: DF

Requirement: To compute the standard error of the mean, confidence limits, and Student's t test, use the default value of VARDEF=.

Tip: When you use the WEIGHT statement and VARDEF=DF, the variance is an estimate of σ^2 , where the variance of the i th observation is $var(x_i) = \sigma^2/w_i$ and w_i is the weight for the i th observation. This yields an estimate of the variance of an observation with unit weight.

Tip: When you use the WEIGHT statement and VARDEF=WGT, the computed variance is asymptotically (for large n) an estimate of σ^2/\bar{w} , where \bar{w} is the average weight. This yields an asymptotic estimate of the variance of an observation with average weight.

See also: “Keywords and Formulas” on page 1578 and “WEIGHT Statement” on page 1510

WINSORIZED=*value(s)* <(<TYPE=keyword> <ALPHA=value>)>

requests of a table of Winsorized means, where *value* is the number or the proportion of observations that PROC UNIVARIATE uses to compute the Winsorized mean. If *value* is a proportion p between 0 and .5, the number of observations that PROC UNIVARIATE uses is equal to the smallest integer that is greater than or equal to np , where n is the number of observations.

TYPE=*keyword*

specifies the type of confidence limit for the mean, where *keyword* is LOWER, UPPER, or TWOSIDED.

Default: TWOSIDED

ALPHA=*value*

specifies the confidence level to compute the confidence limit. The percentage for the confidence limits is $(1-value) \times 100$. For example, ALPHA=.05 results in a 95 percent confidence limit.

Default: The value of ALPHA= in the PROC statement

Range: between 0 and 1

Alias: WINSOR=

Range: between 0 and half the number of nonmissing observations. When a proportion is specified, *value* must be less than .5.

Requirement: To compute confidence limits and the Student's t test, you must use the default value of VARDEF=, which is DF.

Restriction: This option is not available if you specify a WEIGHT statement.

Main discussion “Winsorized Means” on page 1525

Featured in: Example 3 on page 1549

BY Statement

Calculates univariate statistics separately for each BY group.

Main discussion: “BY” on page 54

BY <DESCENDING> *variable-1* <...> <DESCENDING> *variable-n* <NOTSORTED>;

Required Arguments

variable

specifies the variable that the procedure uses to form BY groups. You can specify more than one variable. If you do not use the NOTSORTED option in the BY statement, the observations in the data set must either be sorted by all the variables that you specify, or they must be indexed appropriately. These variables are called *BY variables*.

Options

DESCENDING

specifies that the data set is sorted in descending order by the variable that immediately follows the word DESCENDING in the BY statement.

NOTSORTED

specifies that observations are not necessarily sorted in alphabetic or numeric order. The data are grouped in another way, for example, chronological order.

The requirement for ordering or indexing observations according to the values of BY variables is suspended for BY-group processing when you use the NOTSORTED option. In fact, the procedure does not use an index if you specify NOTSORTED. The procedure defines a BY group as a set of contiguous observations that have the same values for all BY variables. If observations with the same values for the BY variables are not contiguous, the procedure treats each contiguous set as a separate BY group.

CLASS Statement

Specifies up to two variables whose values define the classification levels for the analysis.

Interaction: When you use the HISTOGRAM, PROBPLOT, or QQPLOT statement, PROC UNIVARIATE creates comparative histograms, comparative probability plots, or comparative quantile-quantile plots.

Featured in: Example 9 on page 1568

CLASS *variable-1*<(variable-option(s))> <*variable-2*<(variable-option(s))>>
</ KEYLEVEL=*'value1'* | (*'value1'* *'value2'*)>;

Required Arguments

variable-n

specifies one or two variables that the procedure uses to group the data into classification levels. Variables in a CLASS statement are referred to as *class variables*.

Class variables can be numeric or character. Class variables can have continuous values, but they typically have a few discrete values that define levels of the variable. You do not have to sort the data by class variables. PROC UNIVARIATE uses the formatted values of the class variables to determine the classification levels.

You can use the HISTOGRAM, PROBPLOT, or QQPLOT statement with the CLASS statement to create one-way and two-way comparative plots. When you use one class variable, PROC UNIVARIATE displays an array of component plots (stacked or side-by-side), one for each level of the classification variable. When you use two class variables, PROC UNIVARIATE displays a matrix of component plots, one for each combination of levels of the classification variables. The observations in a given level are referred to collectively as a *cell*.

Restriction: The length of a character class variable cannot exceed 16.

Interaction: When you create a one-way comparative plot, the observations in the input data set are sorted by the formatted values (levels) of the variable. PROC UNIVARIATE creates a separate plot for the analysis variable values in each level, and arranges these component plots in an array to form the comparative plot with uniform horizontal and vertical axes.

When you create a two-way comparative plot, the observations in the input data set are cross-classified according to the values (levels) of these variables. PROC UNIVARIATE creates a separate plot for the analysis variable values in each cell of the cross-classification and arranges these component plots in a matrix to form the comparative plot with uniform horizontal and vertical axes. The levels of *variable-1* are the labels for the rows of the matrix, and the levels of *variable-2* are the labels for the columns of the matrix.

Interaction: If you associate a label with a variable, PROC UNIVARIATE displays the variable label in the comparative plot and this label is parallel to the column (or row) labels.

Tip: Use the MISSING option to treat missing values as valid levels.

Tip: To reduce the number of classification levels, use a FORMAT statement to combine variable values.

Options

KEYLEVEL=*value1*' | (*value1* *value2*)

specifies the *key cell* in a comparative plot. PROC UNIVARIATE first determines the bin size and midpoints for the key cell, and then extends the midpoint list to accommodate the data ranges for the remaining cells. Thus, the choice of the key cell determines the uniform horizontal axis that PROC UNIVARIATE uses for all cells.

If you specify only one class variable and use a HISTOGRAM statement, KEYLEVEL=*value*' identifies the key cell as the level for which *variable* is equal to *value*. By default, PROC UNIVARIATE sorts the levels in the order that is determined by the ORDER= option. Then, the key cell is the first occurrence of a level in this order. The cells display in order from top to bottom or left to right. Consequently, the key cell appears at the top (or left). When you specify a different key cell with the KEYLEVEL= option, this cell appears at the top (or left).

Likewise, with the PROBPLOT statement and the QQPLOT statement the key cell determines uniform axis scaling.

If you specify two class variables, use KEYLEVEL=('value1' 'value2') to identify the key cell as the level for which *variable-n* is equal to *value-n*. By default, PROC UNIVARIATE sorts the levels of the first variable in the order that is determined by its ORDER= option and, within each of these levels, it sorts the levels of the second variable in the order that is determined by its ORDER= option. Then, the default key cell is the first occurrence of a combination of levels for the two variables in this order. The cells display in the order of *variable-1* from top to bottom and in the order of *variable-2* from left to right. Consequently, the default key cell appears at the upper left corner. When you specify a different key cell with the KEYLEVEL= option, this cell appears at the upper left corner.

Restriction: The length of the KEYLEVEL= value cannot exceed 16 characters and you must specify a formatted value.

Requirement: This option is not available unless you specify a HISTOGRAM, PROBPLOT, or QQPLOT statement.

See also: the ORDER= option on page 1453

MISSING

specifies to treat the missing values for the class variable as valid classification levels. Special missing values that represent numeric values (the letters A through Z and the underscore (_) character) are each considered as a separate value.

Default: If you omit MISSING, PROC UNIVARIATE excludes the observations with a missing class variable value from the analysis.

Requirement: Enclose this option in parentheses after the class variable.

See also: *SAS Language Reference: Concepts* for a discussion of missing values that have special meaning.

ORDER=DATA | FORMATTED | FREQ | INTERNAL

specifies the display order for the class variable values, where

DATA

orders values according to their order in the input data set.

Interaction: When you use a HISTOGRAM, PROBPLOT, or QQPLOT statement, PROC UNIVARIATE displays the rows (columns) of the comparative plot from top to bottom (left to right) in the order that the class variable values first appear in the input data set.

FORMATTED

orders values by their ascending formatted values. This order depends on your operating environment.

Interaction: When you use a HISTOGRAM, PROBPLOT, or QQPLOT statement, PROC UNIVARIATE displays the rows (columns) of the comparative plot from top to bottom (left to right) in increasing order of the formatted class variable values. For example, a numeric class variable DAY (with values 1, 2, and 3) has a user-defined format that assigns **Wednesday** to the value 1, **Thursday** to the value 2, and **Friday** to the value 3. The rows of the comparative plot will appear in alphabetical order (Friday, Thursday, Wednesday) from top to bottom.

FREQ

orders values by descending frequency count so that levels with the most observations are listed first. If two or more values have the same frequency count, PROC UNIVARIATE uses the formatted values to determine the order.

Interaction: When you use a HISTOGRAM, PROBPLOT, or QQPLOT statement, PROC UNIVARIATE displays the rows (columns) of the comparative plot from

top to bottom (left to right) in order of decreasing frequency count for the class variable values.

INTERNAL

orders values by their unformatted values, which yields the same order as PROC SORT. This order depends on your operating environment.

If there are two or more distinct internal values with the same formatted value then PROC UNIVARIATE determines the order by the internal value that occurs first in the input data set.

Interaction: When you use a HISTOGRAM, PROBPLOT, or QQPLOT statement, PROC UNIVARIATE displays the rows (columns) of the comparative plot from top to bottom (left to right) in increasing order of the internal (unformatted) values of the class variable. The first class variable is used to label the rows of the comparative plots (top to bottom). The second class variable are used to label the columns of the comparative plots (left to right). For example, a numeric class variable DAY (with values 1, 2, and 3) has a user-defined format that assigns **Wednesday** to the value 1, **Thursday** to the value 2, and **Friday** to the value 3. The rows of the comparative plot will appear in day-of-the-week order (Wednesday, Thursday, Friday) from top to bottom.

Default: INTERNAL

Requirement: Enclose this option in parentheses after the class variable.

Interaction: When you use a HISTOGRAM, PROBPLOT, or QQPLOT statement and ORDER=INTERNAL, PROC UNIVARIATE constructs the levels of the class variables by using the formatted values of the variables. The formatted values of the first class variable are used to label the rows of the comparative plots (top to bottom). The formatted values of a second class variable are used to label the columns of the comparative plots (left to right).

PROC UNIVARIATE determines the layout of a two-way comparative plot by using the order for the first class variable to obtain the order of the rows from top to bottom. Then it applies the order for the second class variable to the observations that correspond to the first row to obtain the order of the columns from left to right. If any columns remain unordered (that is, the categories are unbalanced), PROC UNIVARIATE applies the order for the second class variable to the observations in the second row, and so on, until all the columns have been ordered.

Featured in: Example 9 on page 1568

FREQ Statement

Specifies a numeric variable whose values represent the frequency of the observation.

Tip: The FREQ statement affects the degrees of freedom, but the WEIGHT statement does not.

See also: For an example that uses the FREQ statement, see “FREQ” on page 56

FREQ *variable;*

Required Arguments

variable

specifies a numeric variable whose value represents the frequency of the observation. If you use the FREQ statement, the procedure assumes that each observation represents n observations, where n is the value of *variable*. If *variable* is not an integer, the SAS System truncates it. If *variable* is less than 1 or is missing, the procedure excludes that observation from the analysis.

HISTOGRAM Statement

Creates histograms using high-resolution graphics and optionally superimposes parametric and nonparametric density curve estimates.

Alias: HIST

Restriction: You can not specify the WEIGHT statement with the HISTOGRAM statement.

Tip: You can use multiple HISTOGRAM statements.

Featured in: Example 2 on page 1546, Example 7 on page 1561 and Example 9 on page 1568

HISTOGRAM *<variable(s)>* *</option(s)>*;

To do this	Use this option
Create output data set with information on histogram intervals	OUTHISTOGRAM=
Request estimated density curve	
Fit beta density with threshold parameter θ , scale parameter σ , and shape parameters α and β	BETA(<i>beta-suboptions</i>)
Fit exponential density with threshold parameter θ and scale parameter σ	EXPONENTIAL(<i>exponential-suboptions</i>)
Fit gamma density with threshold parameter θ , scale parameter σ , and shape parameter α	GAMMA(<i>gamma-suboptions</i>)
Fit nonparametric kernel density estimates	KERNEL(<i>kernel-suboptions</i>)
Fit lognormal density with threshold parameter θ , scale parameter ζ , and shape parameter σ	LOGNORMAL(<i>lognormal-suboptions</i>)
Fit normal density with mean μ and standard deviation σ	NORMAL(<i>normal-suboptions</i>)
Fit Weibull density with threshold parameter θ , scale parameter σ , and shape parameter C	WEIBULL(<i>Weibull-suboptions</i>)
Parametric density curve suboptions	
Specify shape parameter α for fitted beta or gamma curve	ALPHA=

To do this	Use this option
Specify second shape parameter β for beta fitted curve	BETA=
Specify shape parameter C for fitted Weibull curve	C=
Specify the mean μ for fitted normal curve	MU=
Specify scale parameter σ for the fitted beta curve, exponential curve, gamma curve and Weibull curve; standard deviation σ for fitted normal curve; or the scale parameter σ for the fitted lognormal curve	SIGMA=
Specify threshold parameter θ for fitted beta curve, exponential curve, gamma curve, lognormal curve, and Weibull curve	THETA=
Specify scale parameter ζ for fitted lognormal curve	ZETA=
Nonparametric density curve suboptions	
Specify standardized bandwidth parameter C for fitted kernel density estimates	C=
Specify type of kernel density curve	K=
Specify lower bounds for fitted kernel density estimates	LOWER=
Specify upper bounds for fitted kernel density estimates	UPPER=
Control appearance of fitted density curves	
Specify color of fitted curve	COLOR=
Fill area under fitted curve	FILL
Specify line type of fitted curve	L=
Display table of histogram interval midpoints	MIDPERCENTS
Suppress the table summarizing the fitted curve	NOPRINT
List percentages for calculated and estimated quantiles	PERCENTS=
Specify width of fitted density curve	W=
Control general histogram layout	
Specify width for the bars	BARWIDTH=
Force creation of a histogram	FORCEHIST
Draw reference lines in front of the histogram bars	FRONTREF
Create a grid	GRID
Specify offset for horizontal axis	HOFFSET=
Specify reference lines perpendicular to the horizontal axis	HREF=
Specify labels for HREF= lines	HREFLABELS=
Specify vertical position of labels for HREF= lines	HREFLABPOS=
Specify a line style for grid lines	LGRID=

To do this	Use this option
List percentages for histogram intervals	MIDPOINTS=
Suppress histogram bars	NOBARS
Suppress frame around plotting area	NOFRAME
Suppress label for horizontal axis	NOHLABEL
Suppress plot	NOPLOT
Suppress label for vertical axis	NOVLABEL
Suppress tick marks and tick mark labels for vertical axis	NOVTICK
Include right endpoint in interval	RTINCLUDE
Turn and vertically string out characters in labels for vertical axis	TURNVLABELS
Specify tick mark values for vertical axis	VAXIS=
Specify label for vertical axis	VAXISLABEL=
Specify length of offset at upper end of vertical axis	VOFFSET=
Specify reference lines perpendicular to the vertical axis	VREF=
Specify labels for VREF= lines	VREFLABELS=
Specify horizontal position of labels for VREF= lines	VREFLABPOS=
Specify scale for vertical axis	VSCALE=
Specify line thickness for axes and frame	WAXIS=
Specify line thickness for grid	WGRID=
Enhance the graph	
Specify annotate data set	ANNOTATE=
Specify color for axis	CAXIS=
Specify color of outlines of histogram bars	CBARLINE=
Specify color for filling under curve	CFILL=
Specify color for frame	CFRAME=
Specify color for grid lines	CGRID=
Specify color for HREF= lines	CHREF=
Specify color for text	CTEXT=
Specify color for VREF= lines	CVREF=
Specify description for plot in graphics catalog	DESCRIPTION=
Specify software font for text	FONT=
Specify height of text used outside framed areas	HEIGHT=
Specify number of horizontal minor tick marks	HMINOR=
Specify software font for text inside framed areas	INFONT=
Specify height of text inside framed areas	INHEIGHT=
Specify line style for HREF= lines	LHREF=

To do this	Use this option
Specify line style for VREF= lines	LVREF=
Specify name for plot in graphics catalog	NAME=
Specify pattern for filling under curve	PFILL=
Specify number of vertical minor tick marks	VMINOR=
Specify line thickness for bar outlines	WBARLINE=
Enhance comparative histograms	
Apply annotation requested in ANNOTATE= data set to key cell only	ANNOKEY
Specify color for filling frame for row labels	CFRAMESIDE=
Specify color for filling frame for column labels	CFRAMETOP=
Specify color for proportion of frequency bar	CPROP=
Specify distance between tiles	INTERTILE=
Specify maximum number of bins to display	MAXNBIN=
Limit the number of bins that display to within a specified number of standard deviations above and below mean of data in key cell	MAXSIGMAS=
Specify number of columns in comparative histogram	NCOLS=
Specify number of rows in comparative histogram	NROWS=

Arguments

variable(s)

identifies one or more analysis variables that the procedure uses to create histograms.

Default: If you omit *variable(s)* in the HISTOGRAM statement, then the procedure creates a histogram for each variable that you list in the VAR statement, or for each numeric variable in the DATA= data set if you omit a VAR statement.

Requirement: If you specify a VAR statement, use a subset of the *variable(s)* that you list in the VAR statement. Otherwise, *variable(s)* are any numeric variables in the DATA= data set.

Options

ALPHA=*value*

specifies the shape parameter α for fitted density curves when you request the BETA and GAMMA options.

Alias: A= if you use it as a *beta-suboption*. SHAPE= if you use it as a *gamma-suboption*

Default: a maximum likelihood estimate

Requirement: Enclose this suboption in parentheses after the BETA option or GAMMA option.

ANNOKEY

specifies to apply the annotation requested with the ANNOTATE= option to the *key cell* only. By default, PROC UNIVARIATE applies annotation to all of the cells.

Requirement: This option is not available unless you specify the CLASS statement.

Tip: Use the KEYLEVEL= option in the CLASS statement to specify the key cell.

See also: the KEYLEVEL= option on page 1452

ANNOTATE=SAS-data-set

specifies an input data set that contains annotate variables as described in *SAS/GRAPH Reference*.

Alias: ANNO=

Tip: You can also specify an ANNOTATE= data set in the PROC UNIVARIATE statement to enhance all the graphic displays that the procedure creates.

See also: ANNOTATE= on page 1445 in the PROC UNIVARIATE statement

BARWIDTH=value

specifies the width of the histogram bars in screen percent units.

BETA<(beta-suboptions)>

displays a fitted beta density curve on the histogram.

Restriction: The BETA option can occur only once in a HISTOGRAM statement.

Interaction: The beta distribution is bounded below by the parameter θ and above by the value $\theta + \sigma$. Use the THETA= and SIGMA= suboptions to specify these parameters. The default values for THETA= and SIGMA= are 0 and 1, respectively. You can specify THETA=EST and SIGMA=EST to request maximum likelihood estimates for θ and σ .

Note: Three- and four-parameter maximum likelihood estimation may not always converge. Δ

Interaction: The beta distribution has two shape parameters, α and β . If these parameters are known, you can specify their values with the ALPHA= and BETA= options. By default, PROC UNIVARIATE computes maximum likelihood estimates for α and β .

Main Discussion: See “Beta Distribution” on page 1530

See also: the ALPHA= suboption on page 1458, BETA= suboption on page 1459, SIGMA= suboption on page 1470, and THETA= suboption on page 1470

BETA=value

specifies the second shape parameter β for the fitted beta density curves when you request the BETA option.

Alias: B=

Default: a maximum likelihood estimate

Requirement: Enclose this suboption in parentheses after the BETA option.

C=value

specifies the shape parameter c for the fitted Weibull density curve when you request the WEIBULL option.

Default: a maximum likelihood estimate

Requirement: Enclose this suboption in parentheses after the WEIBULL option.

C=value(s) | MISE

specifies the standardized bandwidth parameter c for kernel density estimates when you request the KERNEL option.

Default: the bandwidth that minimizes the approximate MISE.

Restriction: You can specify up to five values to request multiple estimates.

Requirement: Enclose this suboption in parentheses after the KERNEL option.

Interaction: You can also use the C= suboption with the K= suboption, which specifies the kernel function, to compute multiple estimates. If you specify more kernel functions than bandwidths, PROC UNIVARIATE repeats the last bandwidth in the list for the remaining estimates. Likewise, if you specify more bandwidths than kernel functions, then PROC UNIVARIATE repeats the last kernel function for the remaining estimates. For example, the following statements compute three density estimates:

```
proc univariate;
  var length;
  histogram length / kernel(c=1 2 3 k=normal quadratic);
run;
```

The first uses a normal kernel and a bandwidth of 1, the second uses a quadratic kernel and a bandwidth of 2, and the third uses a quadratic kernel and a bandwidth of 3.

Tip: To estimate a bandwidth that minimizes the approximate mean integrated square error (MISE) use the C=MISE suboption. For example, the following statements compute three density estimates:

```
proc univariate;
  var length;
  histogram length / kernel(c=0.5 1.0 mise);
run;
```

The first two estimates have standardized bandwidths of 0.5 and 1.0, respectively, and the third has a bandwidth that minimizes the approximate MISE.

CAXIS=*color*

specifies the color for the axes and tick marks.

Alias: CAXES= and CA=

Default: the first color in the device color list

CBARLINE=*color*

specifies the color for the outline of the histogram bars.

Default: the first color in the device color list

Featured in: Example 7 on page 1561

CFILL=*color*

specifies the color to fill the bars of the histogram (or the area under a fitted density curve if you also specify the FILL option).

See also: FILL option on page 1463 and PFILL=option on page 1470

Featured in: Example 2 on page 1546, Example 7 on page 1561, and Example 9 on page 1568

CFRAME=*color*

specifies the color for the area that is enclosed by the axes and frame.

Default: The area is not filled.

CFRAMESIDE=*color*

specifies the color to fill the frame area for the row labels that display along the left side of the comparative histogram. This color also fills the frame area for the label of the corresponding class variable (if you associate a label with the variable).

Default: These areas are not filled.

Requirement: This option is not available unless you specify the CLASS statement.

CFRAMETOP=*color*

specifies the color to fill the frame area for the column labels that display across the top of the comparative histogram. This color also fills the frame area for the label of the corresponding class variable (if you associate a label with the variable).

Default: These areas are not filled.

Requirement: This option is not available unless you specify the CLASS statement.

CGRID=*color*

specifies the color for grid lines when a grid displays on the histogram.

Default: the first color in the device color list

Interaction: This option automatically invokes the GRID= option.

CHREF=*color*

specifies the color for horizontal axis reference lines when you specify the HREF= option.

Alias: CH=

Default: the first color in the device color list

COLOR=*color*

specifies the color of the density curve.

Requirement: You must enclose this suboption in parentheses after the density curve option or the KERNEL option.

Interaction: You can specify as a KERNEL suboption a list of up to five colors in parentheses for multiple kernel density estimates. If there are more estimates than colors, the remaining estimates use the last color that you specify.

CPROP=*color* | **EMPTY**

specifies the color for a horizontal bar whose length (relative to the width of the tile) indicates the proportion of the total frequency that is represented by the corresponding cell in a comparative histogram.

Default: bars do not display

Requirement: This option is not available unless you specify the CLASS statement.

Tip: Use the keyword EMPTY to display empty bars.

CTEXT=*color*

specifies the color for tick mark values and axis labels.

Alias: CT=

Default: The color that you specify for the CTEXT= option in the GOPTIONS statement. If you omit the GOPTIONS statement, the default is the first color in the device color list.

CTEXTSIDE=*color*

specifies the color for the row labels that display along the left side of the comparative histogram.

Default: The color for CTEXT=. If you omit this option, the color that you specify for the CTEXT= option in the GOPTIONS statement. If you omit the GOPTIONS statement, the default is the first color in the device color list.

Requirement: This option is not available unless you specify the CLASS statement.

Tip: Use CFRAMESSIDE= to change the background color for the row labels.

CTEXTTOP=*color*

specifies the color for the row labels that display along the left side of the comparative histogram.

Default: The color for CTEXT=. If you omit this option, the color that you specify for the CTEXT= option in the GOPTIONS statement. If you omit the GOPTIONS statement, the default is the first color in the device color list.

Requirement: This option is not available unless you specify the CLASS statement.

Tip: Use CFRAMETOP= to change the background color for the column labels.

CVREF=*color*

specifies the color for the reference lines that you request with the VREF= option.

Alias: CV=

Default: the first color in the device color list

DESCRIPTION=*'string'*

specifies a description, up to 40 characters long, that appears in the PROC GREPLAY master menu.

Alias: DES=

Default: the variable name

ENDPOINTS<=*value(s)* | KEY | UNIFORM>

uses the endpoints as the tick mark values for the horizontal axis and determines how to compute the bin width of the histogram bars, where

value(s)

specifies values for both the left and right endpoint of each histogram interval. The width of the histogram bars is the difference between consecutive endpoints. PROC UNIVARIATE uses the same *value(s)* for all variables.

Range: The range of endpoints must cover the range of the data. For example, if you specify

```
endpoints=2 to 10 by 2
```

then all of the observations fall in the intervals [2,4) [4,6) [6,8) [8,10].

Requirement: You must use evenly spaced endpoints which you list in increasing order.

KEY

determines the endpoints for the data in the key cell. The initial number of endpoints is based on the number of observations in the key cell using the method of Terrell and Scott (1985). PROC UNIVARIATE extends the endpoint list for the key cell in either direction as necessary until it spans the data in the remaining cells.

Requirement: This option is not available unless you specify the CLASS statement.

UNIFORM

determines the endpoints by using all the observations as if there were no cells. In other words, the number of endpoints is based on the total sample size by using the method of Terrell and Scott (1985).

Requirement: This option does not apply unless you specify the CLASS statement.

Default: If you omit ENDPOINTS, then PROC UNIVARIATE uses the midpoints. If you specify ENDPOINTS, PROC UNIVARIATE computes the endpoints by using an algorithm (Terrell and Scott, 1985) that is primarily applicable to continuous data that are approximately normally distributed.

Interaction: If you specify both MIDPOINTS= and ENDPOINTS, then PROC UNIVARIATE issues a warning message and uses the endpoints.

Interaction: If you specify RTINCLUDE, then PROC UNIVARIATE includes the right endpoint of each histogram interval in that interval instead of including the left endpoint.

Interaction: If you use a CLASS statement and specify ENDPOINTS, then PROC UNIVARIATE use ENDPOINTS=KEY as the default. However if the key cell is empty, then PROC UNIVARIATE use ENDPOINTS=UNIFORM.

See also: MIDPOINTS= option on page 1467 and RTINCLUDE option on page 1470

EXPONENTIAL<(exponential-suboptions)>

displays a fitted exponential density curve on the histogram.

Alias EXP

Restriction: The EXPONENTIAL option can occur only once in a HISTOGRAM statement.

Interaction: The parameter θ must be less than or equal to the minimum data value. Use the THETA= suboption to specify θ . The default value for θ is zero. Specify THETA=EST to request the maximum likelihood estimate for θ .

Interaction: Use the SIGMA= suboption to specify σ . By default, PROC UNIVARIATE computes a maximum likelihood estimate for σ . For example, the following statements fit an exponential curve with $\theta = 10$ and with a maximum likelihood estimate for σ :

```
proc univariate;
  var length;
  histogram / exponential(theta=10 l=2 color=red);
run;
```

Main discussion: See “Exponential Distribution” on page 1531

See also: the SIGMA= suboption on page 1470 and THETA= suboption on page 1470

Featured in: Example 7 on page 1561

FILL

fills areas under the fitted density curve or the kernel density estimate with colors and patterns.

Restriction: The FILL suboption can occur with only one fitted curve.

Requirement: Enclose the FILL suboption in parentheses after a density curve option or the KERNEL option.

Interaction: The CFILL= and PFILL= options specify the color and pattern for the area under the curve.

See also: For a list of available colors and patterns, see *SAS/GRAPH Reference*

Featured in: Example 7 on page 1561

FONT=*font*

specifies a software font for the axis labels.

Default: hardware characters

Interaction: The FONT= *font* takes precedence over the FTEXT= *font* that you specify in the GOPTIONS statement.

Tip: Reference line labels use the font that you specify in the GOPTIONS statement.

FRONTREF

draws reference lines in front of the histogram bars instead of behind them.

Tip: Use FRONTREF when a vertical reference line intersects a histogram bar that is close to the top of the chart. Otherwise, the reference line is almost completely obscured because it is drawn behind the bar. The reference line is visible when it is drawn in front of the bar.

See also: HREF= option on page 1464 and VREF= option on page 1471

FORCEHIST

forces PROC UNIVARIATE to create a histogram when there is only one unique observation. By default, if the standard deviation of the data is zero then PROC UNIVARIATE does not create a histogram.

GAMMA<(gamma-suboptions)>

displays a fitted gamma density curve on the histogram.

Restriction: The GAMMA option can occur only once in a HISTOGRAM statement.

Interaction: The parameter θ must be less than the minimum data value. Use the THETA= suboption to specify θ . The default value for θ is zero. Specify THETA=EST to request the maximum likelihood estimate for θ .

Interaction: Use the ALPHA= and the SIGMA= suboptions to specify the shape parameter α and the scale parameter σ . By default, PROC UNIVARIATE computes maximum likelihood estimates for α and σ . For example, the following statements fit a gamma curve with $\theta = 4$ and with a maximum likelihood estimate for α and σ :

```
proc univariate;
  var length;
  histogram length/ gamma(theta=4);
run;
```

PROC UNIVARIATE calculates the maximum likelihood estimate of α iteratively using the Newton-Raphson approximation.

Main discussion: See “Gamma Distribution” on page 1532

See also: the SIGMA= suboption on page 1470, ALPHA= suboption on page 1458, and the THETA= suboption on page 1470

GRID

specifies to display a grid on the histogram. Grid lines are horizontal lines that are positioned at major tick marks on the vertical axis.

See also: the CGRID= option on page 1461

HEIGHT=*value*

specifies the height in percentage screen units of text for axis labels, tick mark labels, and legends. This option takes precedence over the HTEXT= option in the GOPTIONS statement.

HMINOR=*n*

specifies the number of minor tick marks between each major tick mark on the horizontal axis. PROC UNIVARIATE does not label minor tick marks.

Alias: HM=

Default: 0

HOFFSET=*value*

specifies the offset in percentage screen units at both ends of the horizontal axis.

Tip: Use HOFFSET=0 to eliminate the default offset.

HREF=*value(s)*

draws reference lines that are perpendicular to the horizontal axis at the values that you specify.

Tip: If a reference line is almost completely obscured, then use the FRONTREF option to draw the reference lines in front of the histogram bars.

See also: CHREF= option on page 1461, FRONTREF on page 1463, and LHREF= option on page 1466.

HREFLABELS=*'label1' ... 'labeln'*

specifies labels for the reference lines that you request with the HREF= option.

Alias: HREFLABEL= and HREFLAB=

Restriction: The number of labels must equal the number of reference lines. Labels can have up to 16 characters.

HREFLABPOS=*n*

specifies the vertical position of HREFLABELS= labels, where *n* is

- 1 positions the labels along the top of the histogram
- 2 staggers the labels from top to bottom
- 3 positions the labels along the bottom.

Default: 1

INFONT=*font*

specifies a software font to use for text inside the framed areas of the histogram. The INFONT= option takes precedence over the FTEXT= option in the GOPTIONS statement.

See also: For a list of fonts, see *SAS/GRAPH Reference*.

INHEIGHT=*value*

specifies the height, in percentage screen units of text, to use inside the framed areas of the histogram.

Default: the height that you specify with the HEIGHT= option. If you do not specify the HEIGHT= option, the default height is the height that you specify with the HTEXT= option in the GOPTIONS statement.

INTERTILE=*value*

specifies the distance in horizontal percentage screen units between the framed areas, which are called *tiles*.

Default: .75 in percentage screen units.

Requirement: This option is not available unless you specify the CLASS statement.

Tip: Use INTERTILE=0 to create contiguous tiles.

Featured in: Example 9 on page 1568

K=NORMAL | QUADRATIC | TRIANGULAR

specifies the kernel function (normal, quadratic, or triangular) that PROC UNIVARIATE uses to compute a kernel density estimate.

Default: normal kernel

Restriction: You can specify up to five values to request multiple estimates.

Requirement: You must enclose this suboption in parentheses after the KERNEL option.

Interaction: You can also use the K= suboption with the C= suboption, which specifies standardized bandwidths. If you specify more kernel functions than bandwidths, PROC UNIVARIATE repeats the last bandwidth in the list for the remaining estimates. Likewise, if you specify more bandwidths than kernel functions, PROC UNIVARIATE repeats the last kernel function for the remaining estimates. For example, the following statements compute three estimates with bandwidths of 0.5, 1.0, and 1.5:

```
proc univariate;
  var length;
  histogram length / kernel(c=0.5 1.0 1.5 k=normal quadratic);
run;
```

The first estimate uses a normal kernel, and the last two estimates use a quadratic kernel.

KERNEL<(kernel-suboptions)>

superimposes up to five kernel density estimates on the histogram. By default, PROC UNIVARIATE uses the AMISE method to compute kernel density estimates.

Tip: To request multiple kernel density estimates on the same histogram, specify a list of values for either the C= suboption or K= suboption.

Main discussion: “Kernel Density Estimates” on page 1534

See also: C= suboption on page 1459, K= suboption on page 1465, LOWER= suboption on page 1466, and UPPER= suboption on page 1471

L=linetype

specifies the line type for a fitted density curve or kernel density estimate curve.

Default: 1, which produces a solid line.

Requirement: You must enclose the L= suboption in parentheses after a density curve option or the KERNEL option.

Interaction: If you use the L= suboption with the KERNEL option, you can specify a single line type or a list of line types.

See also: For a list of available line types, see *SAS/GRAPH Reference*

Featured in: Example 7 on page 1561

LGRID=linetype

specifies the line type for the grid when a grid displays on the histogram.

Default: 1, which produces a solid line

Interaction: This option automatically invokes the GRID= option.

LHREF=linetype

specifies the line type for the reference lines that you request with the HREF= option.

Alias: LH=

Default: 2, which produces a dashed line

LOGNORMAL<(lognormal-suboptions)>

displays a fitted lognormal density curve on the histogram.

Restriction: The LOGNORMAL option can occur only once in a HISTOGRAM statement.

Interaction: The parameter θ must be less than the minimum data value. Use the THETA= suboption to specify θ . The default value for θ is zero. Specify THETA=EST to request the maximum likelihood estimate for θ .

Interaction: Use the SIGMA= and ZETA= suboptions to specify σ and ζ . By default, PROC UNIVARIATE computes a maximum likelihood estimate for σ and ζ . For example, the following statements fit a lognormal distribution function with a default value of $\theta = 0$ and with maximum likelihood estimates for σ and ζ :

```
proc univariate;
  var length;
  histogram length/ lognormal;
run;
```

Main discussion: See “Lognormal Distribution” on page 1533

See also: the ZETA= suboption on page 1473, SIGMA= suboption on page 1470, and THETA= suboption on page 1470

LOWER=value(s)

specifies the lower bounds for fitted kernel density curves when you request the KERNEL option.

Default: a missing value, no lower bounds for fitted kernel density curves.

Requirement: Enclose this suboption in parentheses after the KERNEL option.

Interaction: If you specify more kernel curves than lower bounds, PROC UNIVARIATE repeats the last lower bound in the list for the remaining density curves.

LVREF=linetype

specifies the line type for the reference lines that you request with the VREF= option.

Alias: LV=

Default: 2, which produces a dashed line

MAXNBIN=n

specifies the maximum number of bins in the comparative histogram that display. This option is useful when the scales or ranges of the data distributions differ greatly from cell to cell.

By default, PROC UNIVARIATE determines the bin size and midpoints for the key cell, and then extends the midpoint list to accommodate the data ranges for the remaining cells. However, if the cell scales differ considerably, the resulting number of bins may be so great that each cell histogram is scaled into a narrow region. By using MAXNBIN= to limit the number of bins, you can narrow the window about the data distribution in the key cell.

Requirement: This option is not available unless you specify the CLASS statement.

Tip: MAXNBIN= provides an alternative to the MAXSIGMAS= option.

MAXSIGMAS=value

specifies to limit the number of bins in the comparative histogram that display to a range of *value* standard deviations (of the data in the key cell) above and below the mean of the data in the key cell. This option is useful when the scales or ranges of the data distributions differ greatly from cell to cell.

By default, PROC UNIVARIATE determines the bin size and midpoints for the key cell, and then extends the midpoint list to accommodate the data ranges for the remaining cells. However, if the cell scales differ considerably, the resulting number of bins may be so great that each cell histogram is scaled into a narrow region. By using MAXSIGMAS= to limit the number of bins, you can narrow the window that surrounds the data distribution in the key cell.

Requirement: This option is not available unless you specify the CLASS statement.

MIDPERCENTS

requests a table that lists the midpoints and percentage of observations in each histogram interval.

Interaction: If you specify MIDPERCENTS in parentheses after a density estimate option, PROC UNIVARIATE displays a table that lists the midpoints, the observed percentage of observations, and the estimated percentage of the population in each interval (estimated from the fitted distribution).

MIDPOINTS=value(s) | KEY | UNIFORM

specifies how to determine the midpoints for the histogram intervals, where

value(s)

determines the width of the histogram bars as the difference between consecutive midpoints. PROC UNIVARIATE uses the same *value(s)* for all variables.

Range: The range of midpoints, extended at each end by half of the bar width, must cover the range of the data. For example, if you specify

```
midpoints=2 to 10 by 0.5
```

then all of the observations should fall between 1.75 and 10.25.

Requirement: You must use evenly spaced midpoints which you list in increasing order.

KEY

determines the midpoints for the data in the key cell. The initial number of midpoints is based on the number of observations in the key cell that use the method of Terrell and Scott (1985). PROC UNIVARIATE extends the midpoint list for the key cell in either direction as necessary until it spans the data in the remaining cells.

Requirement: This option is not available unless you specify the CLASS statement.

UNIFORM

determines the midpoints by using all the observations as if there were no cells. In other words, the number of midpoints is based on the total sample size by using the method of Terrell and Scott (1985).

Requirement: This option does not apply unless you specify the CLASS statement.

Default: If you use a CLASS statement, MIDPOINTS=KEY; however, if the key cell is empty then MIDPOINTS=UNIFORM. Otherwise, PROC UNIVARIATE computes the midpoints by using an algorithm (Terrell and Scott, 1985) that is primarily applicable to continuous data that are approximately normally distributed.

Featured in: Example 2 on page 1546, Example 7 on page 1561, and Example 9 on page 1568

MU=*value*

specifies the parameter μ for normal density curves.

Default: the sample mean

Requirement: You must enclose this suboption in parentheses after the NORMAL option.

NAME=*'string'*

specifies a name for the plot, up to eight characters long, that appears in the PROC GREPLAY master menu.

Default: UNIVAR

NCOLS=*n*

specifies the number of columns in the comparative histogram.

Alias: NCOL=

Default: NCOLS=1, if you specify only one class variable, and NCOLS=2, if you specify two class variables.

Requirement: This option is not available unless you specify the CLASS statement.

Interaction: If you specify two class variables, you can use the NCOLS= option with the NROWS= option.

Featured in: Example 9 on page 1568

NOBARS

suppresses drawing of histogram bars.

Tip: Use this option to display only the fitted curves.

NOFRAME

suppresses the frame that surrounds the subplot area.

NOHLABEL

suppresses the label for the horizontal axis.

Tip: Use this option to reduce clutter.

NOPLOT

suppresses the creation of a plot.

Alias: NOCHART

Tip: Use NOPLOT when you want to display only descriptive statistics for a fitted density or create an OUTHISTOGRAM= data set.

NOPRINT

suppresses the table of statistics that summarizes the fitted density curve.

Requirement: Enclose this option in the parentheses that follow the density curve option.

Featured in: Example 7 on page 1561

NORMAL<(normal-suboptions)>

displays a fitted normal density curve on the histogram.

Restriction: The NORMAL option can occur only once in a HISTOGRAM statement.

Interaction: Use the MU= and SIGMA= suboptions to specify μ and σ . By default, PROC UNIVARIATE uses the sample mean and sample standard deviation for μ and σ .

Main discussion: See “Normal Distribution” on page 1533

See also: the MU= suboption on page 1468 and the SIGMA= suboption on page 1470

Featured in: Example 7 on page 1561

NOVLABEL

suppresses the label for the vertical axis.

NOVTICK

suppresses the tick marks and tick mark labels for the vertical axis.

Interaction: This option automatically invokes the NOVLABEL option.

NROWS=*n*

specifies the number of rows in the comparative histogram.

Alias: NROW=

Default: 2

Requirement: This option is not available unless you specify the CLASS statement.

Interaction: If you specify two class variables, you can use the NCOLS= option with the NROWS= option.

Featured in: Example 9 on page 1568

OUTHISTOGRAM=SAS-data-set

creates a SAS data set that contains information about histogram intervals.

Specifically, the data set contains the midpoints of the histogram intervals, the observed percentage of observations in each interval, and the estimated percentage of observations in each interval (estimated from each of the specified fitted curves).

Alias: OUTHIST=

See also: “OUTHISTOGRAM= Data Set” on page 1543

PERCENTS=value(s)

specifies a list of percentages that PROC UNIVARIATE uses to calculate quantiles from the data and to estimate quantiles from the fitted density curve.

Alias: PERCENT=

Default: 1, 5, 10, 25, 50, 75, 90, 95, and 99 percent

Range: between 0 and 100

Requirement: You must enclose this suboption in parentheses after the curve option.

PFILL=pattern

specifies a pattern to fill the bars of the histograms (or the areas that are under a fitted density curve if you also specify the FILL option).

Default: The bars and curve areas are not filled.

See also: CFILL= option on page 1460 and FILL option on page 1463

See also: *SAS/GRAPH Reference*

Featured in: Example 2 on page 1546

RTINCLUDE

includes the right endpoint of each histogram interval in that interval. By default, PROC UNIVARIATE includes the left endpoint in the histogram interval.

SCALE=value

is an alias for the SIGMA= suboption when you request density curves with the BETA, EXPONENTIAL, GAMMA, and WEIBULL options and an alias for the ZETA= suboption when you request density curves with the LOGNORMAL option.

See also: SIGMA= suboption on page 1470 and ZETA= suboption on page 1473

SHAPE=value

is an alias for the ALPHA= suboption when you request gamma curves with the GAMMA option, the SIGMA= suboption when you request lognormal curves with the LOGNORMAL option, and the C= suboption when you request Weibull curves with the WEIBULL option.

See also: ALPHA suboption on page 1458, SIGMA suboption on page 1470, and C= suboption on page 1459

SIGMA=value | EST

specifies the parameter σ for the fitted density curve when you request the BETA, EXPONENTIAL, GAMMA, LOGNORMAL, NORMAL, and WEIBULL options. See Table 48.2 on page 1470 for a summary of how to use the SIGMA= suboption.

Default: see Table 48.2 on page 1470

Requirement: You must enclose this suboption in parentheses after the density curve option.

Tip: As a BETA suboption, you can specify SIGMA=EST to request a maximum likelihood estimate for σ .

Table 48.2 Uses of the SIGMA Suboption

Distribution Keyword	SIGMA= Specifies	Default Value	Alias
BETA	scale parameter σ	1	SCALE=
EXPONENTIAL	scale parameter σ	maximum likelihood estimate	SCALE=
GAMMA	scale parameter σ	maximum likelihood estimate	SCALE=
WEIBULL	scale parameter σ	maximum likelihood estimate	SCALE=
LOGNORMAL	shape parameter σ	maximum likelihood estimate	SCALE=
NORMAL	scale parameter σ	standard deviation	SHAPE=

THETA=value | EST

specifies the lower threshold parameter θ for the fitted density curve when you request the BETA, EXPONENTIAL, GAMMA, LOGNORMAL, and WEIBULL options.

Default: 0

Requirement: You must enclose this suboption in parentheses after the curve option.

Tip: To compute a maximum likelihood estimate for θ , specify THETA=EST.

THRESHOLD= *value*

is an alias for the THETA= option. See the THETA= suboption on page 1470.

TURNVLABELS

specifies that PROC UNIVARIATE turn the characters in the vertical axis labels so that they display vertically. This happens by default when you use a hardware font.

Alias: TURNVLABEL

UPPER=*value(s)*

specifies the upper bounds for fitted kernel density curves when you request the KERNEL option.

Default: a missing value, no upper bounds for fitted kernel density curves.

Requirement: Enclose this suboption in parentheses after the KERNEL option.

Interaction: If you specify more kernel curves than upper bounds, PROC UNIVARIATE repeats the last upper bound in the list for the remaining density curves.

VAXIS=*value(s)*

specifies tick mark values for the vertical axis.

Requirement: Use evenly spaced values which you list in increasing order. The first value must be zero and the last value must be greater than or equal to the height of the largest bar. You must scale the values in the same units as the bars.

See also: the VSCALE= option on page 1472

Featured in: Example 9 on page 1568

VAXISLABEL=*'label'*

specifies a label for the vertical axis.

Requirement: Labels can have up to 40 characters.

Featured in: Example 9 on page 1568

VMINOR=*n*

specifies the number of minor tick marks between each major tick mark on the vertical axis. PROC UNIVARIATE does not label minor tick marks.

Alias: VM=

Default: 0

VOFFSET=*value*

specifies the offset in percentage screen units at the upper end of the vertical axis.

VREF=*value(s)*

draws reference lines that are perpendicular to the vertical axis at the *value(s)* that you specify.

Tip: If a reference line is almost completely obscured, then use the FRONTREF option to draw the reference lines in front of the histogram bars.

See also: CVREF= option on page 1462, FRONTREF on page 1463, and LVREF= option on page 1467.

VREFLABELS=*'label1'... 'labeln'*

specifies labels for the reference lines that you request with the VREF= option.

Alias: VREFLABEL= and VREFLAB=

Restriction: The number of labels must equal the number of reference lines. Labels can have up to 16 characters.

VREFLABPOS=*n*

specifies the horizontal position of VREFLABELS= labels, where *n* is

- 1 positions the labels at the left of the histogram.
- 2 positions the labels at the right of the histogram.

Default: 1

VSCALE=*scale*

specifies the scale of the vertical axis, where *scale* is

COUNT

scales the data in units of the number of observations per data unit.

PERCENT

scales the data in units of percentage of observations per data unit.

PROPORTION

scales the data in units of proportion of observations per data unit.

Default: PERCENT

Featured in: Example 9 on page 1568

W=*n*

specifies the width in pixels of the fitted density curve or the kernel density estimate curve.

Default: 1

Requirement: You must enclose this suboption in parentheses after the density curve option or the KERNEL option.

Interaction: As a KERNEL suboption, you can specify a list of up to five W= values.

WAXIS=*n*

specifies the line thickness (in pixels) for the axes and frame.

Default: 1

WBARLINE=*n*

specifies the line thickness for the histogram bar outlines.

Default: 1

WEIBULL<(Weibull-suboptions)>

displays a fitted Weibull density curve on the histogram.

Restriction: The WEIBULL option can occur only once in a HISTOGRAM statement.

Interaction: The parameter θ must be less than the minimum data value. Use the THETA= suboption to specify θ . The default value for θ is zero. Specify THETA=EST to request the maximum likelihood estimate for θ .

Interaction: Use ALPHA= and the SIGMA= suboptions to specify the shape parameter c and the scale parameter σ . By default, PROC UNIVARIATE computes the maximum likelihood estimates for c and σ . For example, the following statements fit a Weibull curve with $\theta = 15$ and with a maximum likelihood estimate for c and σ :

```
proc univariate;
  var length;
  histogram length/ weibull(theta=4);
```

run;

PROC UNIVARIATE calculates the maximum likelihood estimate of α iteratively by using the Newton-Raphson approximation.

Main discussion: See “Weibull Distribution” on page 1534

See also: the C= suboption on page 1459, SIGMA= suboption on page 1470, and THETA= suboption on page 1470

WGRID=*n*

specifies the line thickness for the grid.

ZETA= *value*

specifies a value for the scale parameter ζ for the lognormal density curve when you request the LOGNORMAL option.

Default: a maximum likelihood estimate

Requirement: You must enclose this suboption in parentheses after the LOGNORMAL option.

ID Statement

Identifies the extreme observations in the table of extreme observations.

Featured in: Example 2 on page 1546

ID *variable(s)*;

Required Arguments

variable(s)

specifies one or more variables to include in the table of extreme observations. The corresponding values of the ID variables appear beside the n largest and n smallest observations, where n is the value of NEXTROBS= option.

See also: NEXTROBS= on page 1447

INSET Statement

Places a box or table of summary statistics, called an *inset*, directly in the high-resolution graph.

Requirement: The INSET statement must follow the HISTOGRAM, PROBPLOT, or QQPLOT statement that creates the plot that you want to augment. The inset appears in all the graphs that the preceding plot statement produces.

Tip: You can use multiple INSET statements.

Featured in: Example 8 on page 1566 and Example 9 on page 1568

INSET *<keyword(s) DATA=SAS-data-set> </option(s)>*;

Arguments

keyword(s)

specifies one or more keywords that identify the information to display in the inset. PROC UNIVARIATE displays the information in the order that you request the keywords.

You can specify statistical keywords, primary keywords, and secondary keywords. The available statistical keywords are

Descriptive statistic keywords

CSS	CV	KURTOSIS
MAX	MEAN	N
MIN	MODE	RANGE
NMISS	NOBS	STDMEAN
SKEWNESS	STD	USS
SUM	SUMWGT	VAR

Quantile statistic keywords

MEDIAN	P1	P5
P10	P90	P95
P99	Q1	Q3
QRANGE		

Robust statistic keywords

GINI	MAD	QN
SN	STD_GINI	STD_MAD
STD_QN	STD_QRANGE	STD_SN

Hypothesis testing keywords

MSIGN	PROBM	PROBT
NORMALTEST	PROBN	SIGNRANK
PNORMAL	PROBS	T

A *primary keyword* allows you to specify *secondary keywords* in parentheses immediately after the primary keyword. Primary keywords are BETA, EXPONENTIAL, GAMMA, LOGNORMAL, NORMAL, WEIBULL, WEIBULL2, KERNEL, and KERNEL n . If you specify a primary keyword but omit a secondary keyword, the inset displays a colored line and the distribution name as a key for the density curve. For a list of the secondary keywords, see Table 48.3 on page 1475.

By default, PROC UNIVARIATE identifies inset statistics with appropriate labels and prints numeric values using appropriate formats. To customize the label, specify the keyword followed by an equal sign (=) and the desired label in quotes. To customize the format, specify a numeric format in parentheses after the keyword. Labels can have up to 24 characters. If you specify both a label and a format for a statistic, the label must appear before the format. For example,

```
inset n='Sample Size' std='Std Dev' (5.2);
```

requests customized labels for two statistics and displays the standard deviation with field width of 5 and two decimal places.

Table 48.3 Available Secondary Keywords

Keyword	Alias	Description
For BETA primary keyword		
ALPHA	SHAPE1	first shape parameter α
BETA	SHAPE2	second shape parameter β
SIGMA	SCALE	scale parameter σ
THETA	THRESHOLD	lower threshold parameter θ
For EXP primary keyword		
SIGMA	SCALE	scale parameter σ
THETA	THRESHOLD	threshold parameter θ
For GAMMA primary keyword		
ALPHA	SHAPE	shape parameter α
SIGMA	SCALE	scale parameter σ
THETA	THRESHOLD	threshold parameter θ
For LOGNORMAL primary keyword		
SIGMA	SHAPE	shape parameter σ
THETA	THRESHOLD	threshold parameter θ
ZETA	SCALE	scale parameter ζ
For NORMAL primary keyword		
MU	MEAN	mean parameter μ
SIGMA	STD	shape parameter σ
For WEIBULL primary keyword		
C	SHAPE	shape parameter c
SIGMA	SCALE	scale parameter σ
THETA	THRESHOLD	threshold parameter θ
For WEIBULL2 primary keyword		
C	SHAPE	shape parameter c
SIGMA	SCALE	scale parameter σ
THETA	THRESHOLD	known lower threshold parameter θ_0
For any parametric distribution primary keyword*		
AD		Anderson-Darling EDF test statistic
ADPVAL		Anderson-Darling EDF test p -value
CVM		Cramer-von Mises EDF test statistic

Keyword	Alias	Description
CVMPVAL		Cramer-von Mises EDF test p -value
KSD		Kolmogorov-Smirnov EDF test statistic
KSDPVAL		Kolmogorov-Smirnov EDF test p -value
For KERNEL or KERNEL n primary keyword*		
TYPE		kernel type: normal, quadratic, or triangular
BANDWIDTH	BWIDTH	bandwidth λ for the density estimate
C		standardized bandwidth c for the density estimate: $c = \frac{\lambda}{Q} n^{\frac{1}{5}}$ where n =sample size, λ =bandwidth, and Q =interquartile range
AMISE		approximate mean integrated square error (MISE) for the kernel density

* Available with only the HISTOGRAM statement and a BETA, EXPONENTIAL, LOGNORMAL, NORMAL, or WEIBULL distribution.

Requirement: Some inset statistics are not available unless you request a plot statement and options that calculate these statistics. For example:

```
proc univariate data=score;
    histogram final / normal;
    inset mean std normal(ad adpval);
run;
```

The MEAN and STD keywords display the sample mean and standard deviation of FINAL. The NORMAL keyword with the *secondary keywords* AD and ADPVAL display the Anderson-Darling goodness-of-fit test statistic and p -value. The statistics that are specified with the NORMAL keyword are available only because the NORMAL option is requested in the HISTOGRAM statement.

The KERNEL or KERNEL n keyword is available only if you request a kernel density estimate in a HISTOGRAM statement. The WEIBULL2 keyword is available only if you request a two-parameter Weibull distribution in the PROBLOT or QQPLOT statement.

Tip: To specify the same format for all the statistics in the INSET statement, use the FORMAT= option.

Tip: To create a completely customized inset, use a DATA= data set. The data set contains the label and the value that you want to display in the inset.

Tip: If you specify multiple kernel density estimates, you can request inset statistics for all the estimates with the KERNEL keyword. Alternatively, you can display inset statistics for individual curves with KERNEL n keyword, where n is the curve number between 1 and 5.

Featured in: Example 8 on page 1566 and Example 9 on page 1568

DATA=SAS-data-set

requests that PROC UNIVARIATE display customized statistics from a SAS data set in the inset table. The data set must contain two variables:

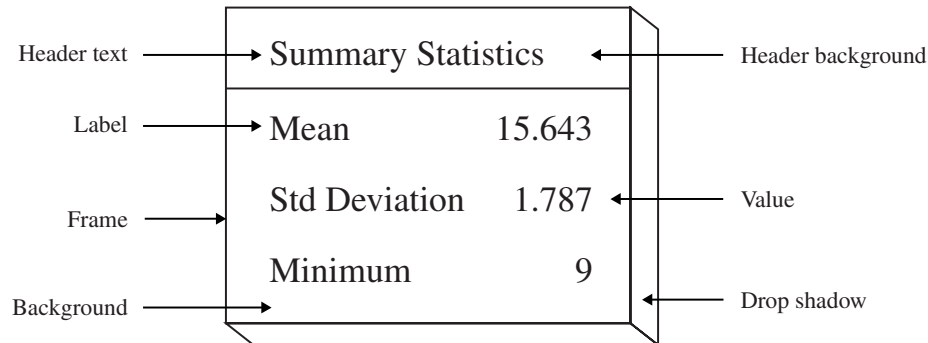
<u>_LABEL_</u>	a character variable whose values provide labels for inset entries.
<u>_VALUE_</u>	a variable that is either character or numeric and whose values provide values for inset entries.

The label and value from each observation in the data set occupy one line in the inset. The position of the DATA= keyword in the keyword list determines the position of its lines in the inset.

Options

Figure 48.4 on page 1477 illustrates the meaning of terms that are used in this section.

Figure 48.4 The Inset



CFILL=*color* | **BLANK**

specifies the color of the background which, if you omit the CFILLH= option, includes the header background.

Default The background is empty which causes items that overlap the inset (such as curves, histogram bars, or specification limits) to show through the inset.

Tip: Specify a value for CFILL= so that items that overlap no longer show through the inset. Use CFILL=BLANK to leave the background uncolored.

Featured in: Example 8 on page 1566

CFILLH=*color*

specifies the color of the header background.

Default: the CFILL= *color*

CFRAME=*color*

specifies the color of the frame.

Default: the same color as the axis of the plot

CHEADER=*color*

specifies the color of the header text.

Default: the CTEXT=*color*

CSHADOW=*color*

specifies the color of the drop shadow.

Default: A drop shadow is not displayed.

CTEXT=*color*

specifies the color of the text.

Default: the same color as the other text on the plot

DATA

specifies how to use data coordinates to position the inset with the POSITION= option.

Requirement: The DATA option is available only when you specify POSITION=(x,y). You must place DATA immediately after the coordinates (x,y).

Main Discussion: “Positioning the Inset Using Coordinates” on page 1480

See also: POSITION= option on page 1478

FONT=*font*

specifies the font of the text.

Default: If you locate the inset in the interior of the plot then the font is SIMPLEX. If you locate the inset in the exterior of the plot then the font is the same as the other text on the plot.

Featured in: Example 9 on page 1568

FORMAT=*format*

specifies a format for all the values in the inset.

Interaction: If you specify a format for a particular statistic, then this format overrides FORMAT=*format*.

See also: For more information about SAS formats, see *SAS Language Reference: Dictionary*

Featured in: Example 8 on page 1566

HEADER=*string*

specifies the header text where *string* cannot exceed 40 characters.

Default: No header line appears in the inset.

Interaction: If all the keywords that you list in the INSET statement are secondary keywords that correspond to a fitted curve on a histogram, PROC UNIVARIATE displays a default header that indicates the distribution and identifies the curve.

Featured in: Example 8 on page 1566

HEIGHT=*value*

specifies the height of the text.

Featured in: Example 9 on page 1568

NOFRAME

suppresses the frame drawn around the text.

Featured in: Example 9 on page 1568

POSITION=*position*

determines the position of the inset. The *position* is a compass point keyword, a margin keyword, or a pair of coordinates (x,y).

Alias: POS=

Default: NW, which positions the inset in the upper left (northwest) corner of the display.

Requirement: You must specify coordinates in axis percentage units or axis data units.

Main discussion: “Positioning the Inset Using Compass Point” on page 1479, “Positioning the Inset in the Margins” on page 1480, and “Positioning the Inset Using Coordinates” on page 1480

Featured in: Example 8 on page 1566 and Example 9 on page 1568

REFPOINT=BR | BL | TR | TL

specifies the reference point for an inset that PROC UNIVARIATE positions by a pair of coordinates with the POSITION= option. The REFPOINT= option specifies which

corner of the inset frame that you want to position at coordinates (x,y) . The reference points are

BL	bottom left
BR	bottom right
TL	top left
TR	top right

Default: BL

Requirement: You must use REFPOINT= with POSITION= (x,y) coordinates.

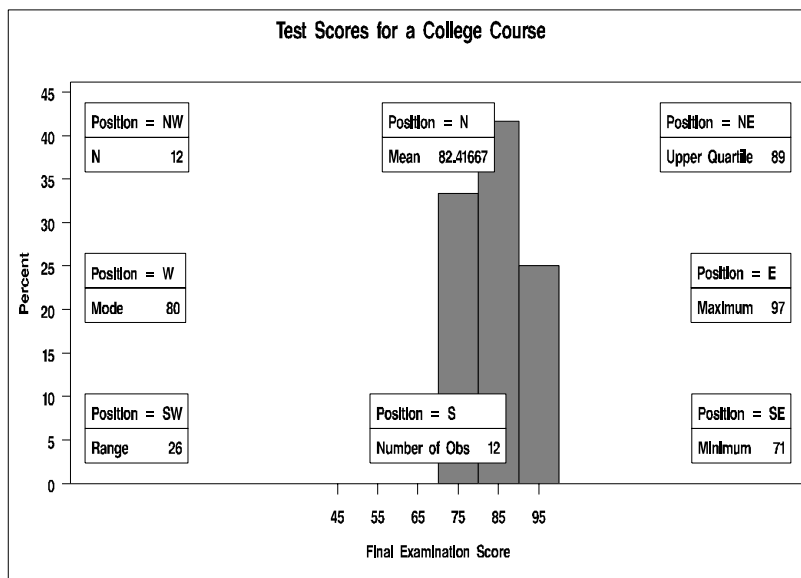
Featured in: Example 8 on page 1566

Positioning the Inset Using Compass Point

To position the inset by using a compass point position, use the keyword N, NE, E, SE, S, SW, W, or NW in the POSITION= option. The default position of the inset is NW.

The following statements produce a histogram to show the position of the inset for the eight compass points:

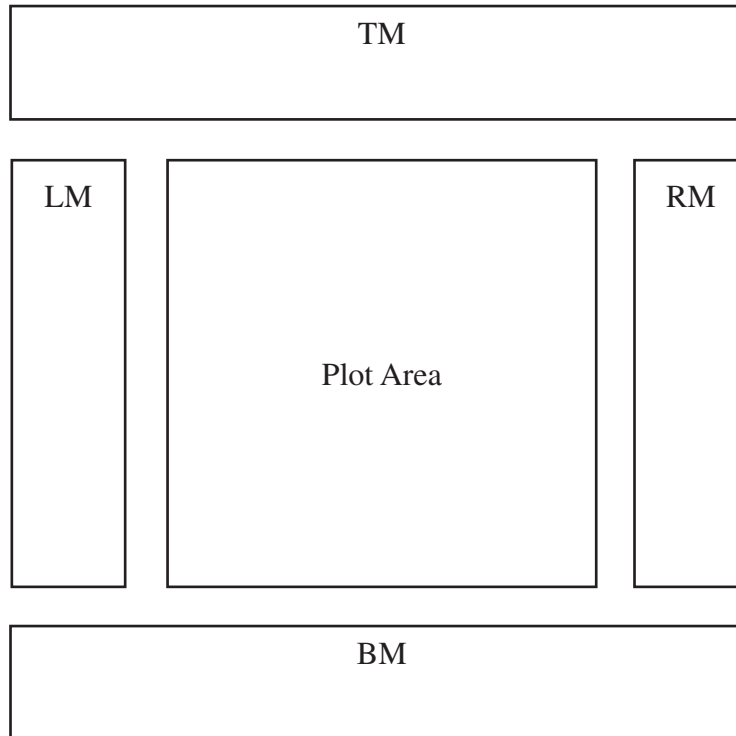
```
proc univariate data=score noprint;
  histogram final / cfill=gray midpoints=45 to 95 by 10 barwidth=5;
  inset n      / cfill=blank header='Position = NW' pos=nw;
  inset mean   / cfill=blank header='Position = N ' pos=n ;
  inset sum    / cfill=blank header='Position = NE' pos=ne;
  inset max    / cfill=blank header='Position = E ' pos=e ;
  inset min    / cfill=blank header='Position = SE' pos=se;
  inset nobs   / cfill=blank header='Position = S ' pos=s ;
  inset range  / cfill=blank header='Position = SW' pos=sw;
  inset mode   / cfill=blank header='Position = W ' pos=w ;
  label final='Final Examination Score';
  title 'Test Scores for a College Course';
run;
```



Positioning the Inset in the Margins

To position the inset in one of the four margins that surround the plot area use the margin keywords LM, RM, TM, or BM in the POSITION= option. Figure 48.5 on page 1480 shows the location of the inset in the margin.

Figure 48.5 Locating the Inset in the Margins



Margin positions are recommended if you list a large number of statistics in the INSET statement. If you attempt to display a lengthy inset in the interior of the plot, it is most likely that the inset will collide with the data display.

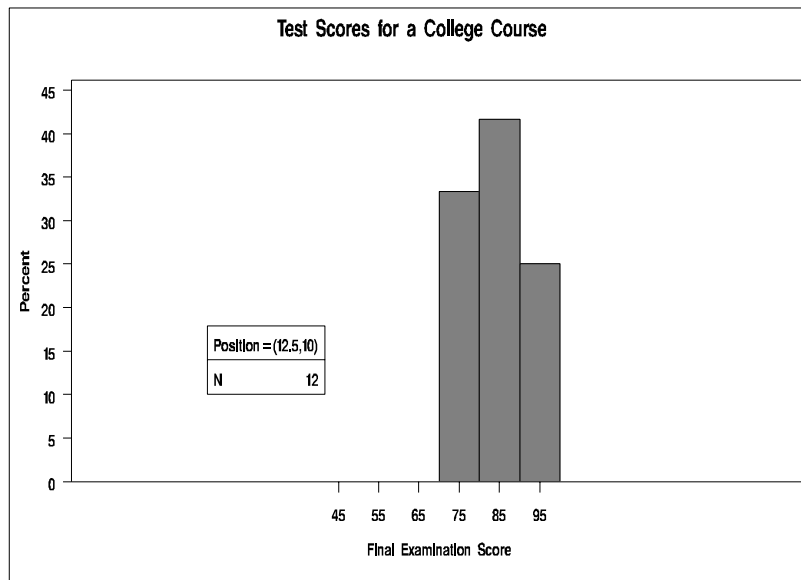
Positioning the Inset Using Coordinates

To position the inset with coordinates, use POSITION=(x,y). You specify the coordinates in axis data units or in axis percentage units (the default).

data unit

If you specify the DATA option immediately following the coordinates, PROC UNIVARIATE positions the inset by using axis data units. For example, the following statements place the bottom left corner of the inset at 12.5 on the horizontal axis and 10 on the vertical axis:

```
proc univariate data=score;
  histogram final / midpoints 45 to 95 by 10 barwidth=5
                  cfill=gray ;
  inset n / header   = 'Position=(12.5,10)'
                position = (12.5,10) data;
run;
```



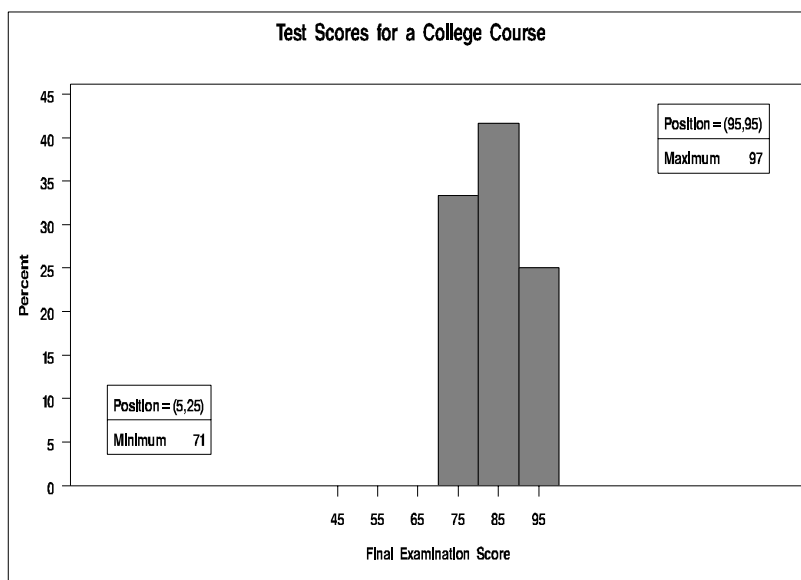
By default, the specified coordinates determine the position of the bottom left corner of the inset. To change this reference point, use the REFPOINT= option (see the next example).

axis percent unit

If you omit the DATA option, PROC UNIVARIATE positions the inset by using axis percentage units. The coordinates in axis percentage units must be *between* 0 and 100. The coordinates of the bottom left corner of the display are (0,0), while the upper right corner is (100,100). For example, the following statements create a histogram and use coordinates in axis percentage units to position the two insets:

```
proc univariate data=score;
  histogram final / midpoints 45 to 95 by 10 barwidth=5
                  cfill=gray;
  inset min / position = (5,25)
             header   = 'Position=(5,25)'
             refpoint = tl;
  inset max / position = (95,95)
             header   = 'Position=(95,95)'
             refpoint = tr;
run;
```

The REFPOINT= option determines which corner of the inset to place at the coordinates that are specified with the POSITION= option. The first inset uses REFPOINT=TL, so that the top left corner of the inset is positioned 5% of the way across the horizontal axis and 25% of the way up the vertical axis. The second inset uses REFPOINT=TR, so that the top right corner of the inset is positioned 95% of the way across the horizontal axis and 95% of the way up the vertical axis.



OUTPUT Statement

Saves statistics and BY variables in an output data set.

Tip: You can save percentiles that are not automatically computed.

Tip: You can use multiple OUTPUT statements to create several OUT= data sets.

Main discussion: “Output Data Set” on page 1542

Featured in: Example 5 on page 1555 and Example 6 on page 1560

OUTPUT <OUT=SAS-data-set> *statistic-keyword-1=name(s)*
 <...*statistic-keyword-n=name(s)*> <percentiles-specification> ;

Options

OUT=SAS-data-set

identifies the output data set. If *SAS-data-set* does not exist, PROC UNIVARIATE creates it. If you omit OUT=, the data set is named DATA*n*, where *n* is the smallest integer that makes the name unique.

Default: DATA*n*

statistic-keyword=name(s)

specifies a statistic to store in the OUT= data set and names the new variable that will contain the statistic. The available statistical keywords are

Descriptive statistic keywords

CSS	CV	KURTOSIS
MAX	MEAN	N
MIN	MODE	RANGE

NMISS	NOBS	STDMEAN
SKEWNESS	STD	USS
SUM	SUMWGT	VAR
Quantile statistic keywords		
MEDIAN	P1	P5
P10	P90	P95
P99	Q1	Q3
QRANGE		
Robust statistic keywords		
GINI	MAD	QN
SN	STD_GINI	STD_MAD
STD_QN	STD_QRANGE	STD_SN
Hypothesis testing keywords		
NORMAL	PROBN	MSIGN
PROBM	SIGNRANK	PROBS
T	PROBT	

See Appendix 1, “SAS Elementary Statistics Procedures,” on page 1577 and “Statistical Computations: UNIVARIATE Procedure” on page 1517 for the keyword definitions and statistical formulas.

To store the same statistic for several analysis variables, specify a list of names. The order of the names corresponds to the order of the analysis variables in the VAR statement. PROC UNIVARIATE uses the first name to create a variable that contains the statistic for the first analysis variable, the next name to create a variable that contains the statistic for the second analysis variable, and so on. If you do not want to output statistics for all the analysis variables, specify fewer names than the number of analysis variables.

percentiles-specification

specifies one or more percentiles to store in the OUT= data set and names the new variables that contain the percentiles. The form of *percentiles-specification* is

PCTLPTS=*percentile(s)* PCTLPRE=*prefix-name(s)* <PCTLNAME=*suffix-name(s)*>

PCTLPTS=*percentile(s)*

specifies one or more percentiles to compute. You can specify percentiles with the expression *start* TO *stop* BY *increment* where *start* is a starting number, *stop* is an ending number, and *increment* is a number to increment by.

Range: any decimal numbers between 0 and 100, inclusive

Example statement: To compute the 50th, 95th, 97.5th, and 100th percentiles, submit the statement

```
output pctlpre=P_ pctlpts=50,95 to 100 by 2.5;
```

PCTLPRE=*prefix-name(s)*

specifies one or more prefixes to create the variable names for the variables that contain the PCTLPTS= percentiles. To save the same percentiles for more than one analysis variable, specify a list of prefixes. The order of the prefixes corresponds to the order of the analysis variables in the VAR statement.

Interaction: PROC UNIVARIATE creates a variable name by combining the PCTLPRE= value and either *suffix-name* or (if you omit PCTLNAME= or if you specify too few *suffix-name(s)*) the PCTLPTS= value.

PCTLNAME=*suffix-name(s)*

specifies one or more suffixes to create the names for the variables that contain the PCTLPTS= percentiles. PROC UNIVARIATE creates a variable name by combining the PCTLPRE= value and *suffix-name*. Because the suffix names are associated with the percentiles that are requested, list the suffix names in the same order as the PCTLPTS= percentiles.

Requirement: You must specify PCTLPRE= to supply prefix names for the variables that contain the PCTLPTS= percentiles.

Interaction: If the number of PCTLNAME= values is fewer than the number of *percentile(s)* or if you omit PCTLNAME=, PROC UNIVARIATE uses *percentile* as the suffix to create the name of the variable that contains the percentile. For an integer percentile, PROC UNIVARIATE uses *percentile*. For a noninteger percentile, PROC UNIVARIATE truncates decimal values of *percentile* to two decimal places and replaces the decimal point with an underscore.

Interaction: If either the prefix and suffix name combination or the prefix and percentile name combination is longer than 32 characters, PROC UNIVARIATE truncates the prefix name so that the variable name is 32 characters.

Saving Percentiles Not Automatically Computed

You can use PCTLPTS= to output percentiles that are not in the list of quantile statistics. PROC UNIVARIATE computes the requested percentiles based on the method that you specify with the PCTLDEF= option in the PROC UNIVARIATE statement. You must use PCTLPRE=, and optionally PCTLNAME=, to specify variable names for the percentiles. For example, the following statements create an output data set that is named PCTLS that contains the 20th and 40th percentiles of the analysis variables Test1 and Test2:

```
proc univariate data=score;
  var Test1 Test2;
  output out=pctls pctlpts=20 40 pctlpre=Test1_ Test2_
        pctlname=P20 P40;
run;
```

PROC UNIVARIATE saves the 20th and 40th percentiles for Test1 and Test2 in the variables Test1_P20, Test2_P20, Test1_P40, and Test2_P40.

Using the BY Statement with the OUTPUT Statement

When you use a BY statement, the number of observations in the OUT= data set corresponds to the number of BY groups. Otherwise, the OUT= data set contains only one observation.

PROBLOT Statement

Creates a probability plot by using high-resolution graphs, which compare ordered variable values with the percentiles of a specified theoretical distribution.

Alias: PROB

Default: Normal probability plot

Restriction: You can not specify the WEIGHT statement with the PROBLOT statement.

Restriction: You can specify only one theoretical distribution.

Tip: You can use multiple PROBLOT statements.

Main discussion:

Featured in: Example 5 on page 1555

PROBLOT *<variable(s)>* *</ option(s)>*;

To do this:	Use this option:
Request a distribution	
Specify beta probability plot with required shape parameters α , β .	BETA(<i>beta-suboptions</i>)
Specify exponential probability plot	EXPONENTIAL(<i>exponential-suboptions</i>)
Specify gamma probability plot with a required shape parameter α	GAMMA(<i>gamma-suboptions</i>)
Specify lognormal probability plot with a required shape parameter σ	LOGNORMAL(<i>lognormal-suboptions</i>)
Specify normal probability plot	NORMAL(<i>normal-suboptions</i>)
Specify three-parameter Weibull probability plot with a required shape parameter c	WEIBULL(<i>Weibull-suboptions</i>)
Specify two-parameter Weibull probability plot	WEIBULL2(<i>Weibull2-suboptions</i>)
Distribution suboptions	
Specify shape parameter α for the beta or gamma distribution	ALPHA=
Specify shape parameter β for the beta distribution	BETA=
Specify shape parameter c for the Weibull distribution or c_0 for distribution reference line of the Weibull2 distribution	C=
Specify μ_0 for distribution reference line for the normal distribution	MU=
Specify σ_0 for distribution reference line for the beta, exponential, gamma, normal, Weibull, or Weibull2 distribution or the required shape parameter σ for the lognormal option	SIGMA=

To do this:	Use this option:
Specify slope of distribution reference line for the lognormal or Weibull2 distribution	SLOPE=
Specify θ_0 for distribution reference line for the beta, exponential, gamma, lognormal, or Weibull distribution, or the lower known threshold θ_0 for the Weibull2 distribution	THETA=
Specify ζ_0 for distribution reference line for the lognormal distribution	ZETA=
Control appearance of distribution reference line	
Specify color of distribution reference line	COLOR=
Specify line type of distribution reference line	L=
Specify width of distribution reference line	W=
Control general plot layout	
Create a grid	GRID
Specify reference lines perpendicular to the horizontal axis	HREF=
Specify labels for HREF lines	HREFLABELS=
Specify a line style for grid lines	LGRID=
Adjust sample size when computing percentiles	NADJ=
Suppress frame around plotting area	NOFRAME
Suppress label for horizontal axis	NOHLABEL
Suppress label for vertical axis	NOVLABEL
Suppress tick marks and tick mark labels for vertical axis	NOVTICK
Request minor tick marks for percentile axis	PCTLMINOR
Specify tick mark labels for percentile axis	PCTLORDER=
Adjust ranks when computing percentiles	RANKADJ=
Display plot in square format	SQUARE
Specify label for vertical axis	VAXISLABEL=
Specify reference lines perpendicular to the vertical axis	VREF=
Specify labels for VREF lines	VREFLABELS=
Specify horizontal position of labels for VREF= lines	VREFLABPOS=
Specify line thickness for axes and frame	WAXIS=
Enhance the probability plot	
Specify annotate data set	ANNOTATE=
Specify color for axis	CAXIS=
Specify color for frame	CFRAME=
Specify color for grid lines	CGRID=

To do this:	Use this option:
Specify color for HREF= lines	CHREF=
Specify color for text	CTEXT=
Specify color for VREF= lines	CVREF=
Specify description for plot in graphics catalog	DESCRIPTION=
Specify software font for text	FONT=
Specify height of text used outside framed areas	HEIGHT=
Specify number of horizontal minor tick marks	HMINOR=
Specify software font for text inside framed areas	INFONT=
Specify height of text inside framed areas	INHEIGHT=
Specify line style for HREF= lines	LHREF=
Specify line style for VREF= lines	LVREF=
Specify name for plot in graphics catalog	NAME=
Specify number of vertical minor tick marks	VMINOR=
Enhance the comparative probability plot	
Apply annotation requested in ANNOTATE= data set to key cell only	ANNOKEY
Specify color for filling frame for row labels	CFRAMESIDE=
Specify color for filling frame for column labels	CFRAMETOP=
Specify distance between tiles	INTERTILE=
Specify number of columns in comparative probability plot	NCOLS=
Specify number of rows in comparative probability plot	NROWS=

Arguments

variable(s)

identifies one or more variables that the procedure uses to create probability plots.

Default: If you omit *variable(s)* in the PROBLOT statement then the procedure creates a probability plot for each variable that you list in the VAR statement, or for each numeric variable in the DATA= data set if you omit a VAR statement.

Requirement: If you specify a VAR statement, use a subset of the *variable(s)* that you list in the VAR statement. Otherwise, *variable(s)* are any numeric variables in the DATA= data set.

Options

ALPHA=*value* | EST

specifies the required shape parameter α ($\alpha > 0$) for probability plots when you request the BETA or GAMMA options. The PROBPLOT statement creates a plot for each value that you specify.

Requirement: Enclose this suboption in parentheses following the BETA or GAMMA options.

Tip: To compute a maximum likelihood estimate for α , specify ALPHA=EST.

ANNOKEY

specifies to apply the annotation requested with the ANNOTATE= option to the *key cell* only. By default, PROC UNIVARIATE applies annotation to all of the cells.

Requirement: This option is not available unless you specify the CLASS statement.

Tip: Use the KEYLEVEL= option in the CLASS statement to specify the key cell.

See also: the KEYLEVEL= option on page 1452

ANNOTATE=SAS-data-set

specifies an input data set that contains annotate variables as described in *SAS/GRAPH Reference*.

Alias: ANNO=

Tip: The ANNOTATE = data set that you specify in the PROBPLOT statement is used by all plots that this statement creates. You can also specify an ANNOTATE= data set in the PROC UNIVARIATE statement to enhance all the graphics displays that the procedure creates.

See also: the ANNOTATE= option on page 1445 in the PROC UNIVARIATE statement

BETA(ALPHA=value | EST BETA=value | EST <beta-suboptions>)

displays a beta probability plot for each combination of the required shape parameters α and β .

Requirement: You must specify the shape parameters with the ALPHA= and BETA= suboptions.

Interaction: To create a plot that is based on maximum likelihood estimates for α and β , specify ALPHA=EST and BETA=EST.

Tip: To obtain graphical estimates of α and β , specify lists of values in the ALPHA= and BETA= suboptions. Then select the combination of α and β that most nearly linearizes the point pattern.

To assess the point pattern, add a diagonal distribution reference line that corresponds to the lower threshold parameter θ_0 and the scale parameter σ_0 with the THETA= and SIGMA= suboptions. Alternatively, you can add a line that corresponds to estimated values of θ_0 and σ_0 with THETA=EST and SIGMA=EST.

Agreement between the reference line and the point pattern indicates that the beta distribution with parameters α , β , θ_0 , and σ_0 is a good fit.

Main discussion: “Beta Distribution” on page 1530

See also: the ALPHA= on page 1487 suboption and BETA= suboption on page 1488

BETA=value | EST

specifies the shape parameter β ($\beta > 0$) for probability plots when you request the BETA distribution option. PROC UNIVARIATE creates a plot for each value that you specify.

Alias: B=

Requirement: Enclose this suboption in parentheses after the BETA option.

Tip: To compute a maximum likelihood estimate for β , specify BETA=EST.

C=value | EST

specifies the shape parameter c ($c > 0$) for probability plots when you request the WEIBULL option or WEIBULL2 option. C= is a required suboption in the WEIBULL option.

Requirement: Enclose this suboption in parentheses after the WEIBULL option or WEIBULL2 option.

Interaction: To request a distribution reference line in the WEIBULL2 option, you must specify both the C= and SIGMA= suboptions.

Tip: To compute a maximum likelihood estimate for c , specify C=EST.

CAXIS=*color*

specifies the color for the axes.

Alias: CAXES=

Default: the first color in the device color list

Interaction: This option overrides any COLOR= specification.

CFRAME=*color*

specifies the color for the area that is enclosed by the axes and frame.

Default: the area is not filled

CFRAMESIDE=*color*

specifies the color to fill the frame area for the row labels that display along the left side of the comparative probability plot. This color also fills the frame area for the label of the corresponding class variable (if you associate a label with the variable).

Default: These areas are not filled.

Requirement: This option is not available unless you specify the CLASS statement.

CFRAMETOP=*color*

specifies the color to fill the frame area for the column labels that display across the top of the comparative probability plot. This color also fills the frame area for the label of the corresponding class variable (if you associate a label with the variable).

Default: These areas are not filled.

Requirement: This option does not apply unless you specify the CLASS statement.

CGRID=*color*

specifies the color for grid lines when a grid displays on the plot.

Default: the first color in the device color list

Interaction: This option automatically invokes the GRID= option.

CHREF=*color*

specifies the color for horizontal axis reference lines when you specify the HREF= option.

Alias: CH=

Default: the first color in the device color list

COLOR=*color*

specifies the color of the diagonal distribution reference line.

Default: the first color in the device color list

Requirement: You must enclose this suboption in parentheses after a distribution option keyword.

CTEXT=*color*

specifies the color for tick mark values and axis labels.

Default: the color that you specify for the CTEXT= option in the GOPTIONS statement. If you omit the GOPTIONS statement, the default is the first color in the device color list.

CVREF=*color*

specifies the color for the reference lines that you request with the VREF= option.

Alias: CV=

Default: the first color in the device color list

DESCRIPTION=*'string'*

specifies a description, up to 40 characters long, that appears in the PROC GREPLAY master menu.

Alias: DES=

Default: the variable name

EXPONENTIAL<(exponential-options)>

displays an exponential probability plot.

Alias: EXP

Tip: To assess the point pattern, add a diagonal distribution reference line that corresponds to θ_0 and σ_0 with the THETA= and SIGMA= suboptions.

Alternatively, you can add a line that corresponds to estimated values of the threshold parameter θ_0 and the scale parameter σ_0 with the THETA=EST and SIGMA=EST suboptions.

Agreement between the reference line and the point pattern indicates that the exponential distribution with parameters θ_0 and σ_0 is a good fit.

Main discussion: “Exponential Distribution” on page 1537

See also: the SIGMA= suboption on page 1494 and the THETA= suboption on page 1495

FONT=*font*

specifies a software font for the reference lines and the axis labels.

Default: hardware characters

Interaction: FONT=*font* takes precedence over the FTEXT=*font* that you specify in the GOPTIONS statement.

GAMMA(ALPHA=*value* | EST <gamma-suboptions>)

displays a gamma probability plot for each value of the required shape parameter α .

Requirement: You must specify the shape parameter with the ALPHA= suboption.

Interaction: To create a plot that is based on a maximum likelihood estimate for α , specify ALPHA=EST.

Tip: To obtain a graphical estimate of α , specify a list of values in the ALPHA= suboption. Then select the value that most nearly linearizes the point pattern.

To assess the point pattern, add a diagonal distribution reference line that corresponds to the threshold parameter θ_0 and the scale parameter σ_0 with the THETA= and SIGMA= suboptions. Alternatively, you can add a line that corresponds to estimated values of θ_0 and σ_0 with THETA=EST and SIGMA=EST.

Agreement between the reference line and the point pattern indicates that the exponential distribution with parameters α , θ_0 , and σ_0 is a good fit.

Main discussion: “Gamma Distribution” on page 1537

See also: the ALPHA= on page 1487 suboption, SIGMA suboption on page 1494, and THETA suboption on page 1495

GRID

displays a grid, drawing reference lines that are perpendicular to the percentile axis at major tick marks.

Default: 1

See also: the CGRID= option on page 1489

HEIGHT=*value*

specifies the height in percentage screen units of text for axis labels, tick mark labels, and legends. This option takes precedence over the HTEXT= option in the GOPTIONS statement.

HMINOR=*n*

specifies the number of minor tick marks between each major tick mark on the horizontal axis. PROC UNIVARIATE does not label minor tick marks.

Alias: HM=

Default: 0

HREF=*value(s)*

draws reference lines that are perpendicular to the horizontal axis at the values you specify.

See also: CHREF= option on page 1489

HREFLABELS=*'label1' ... 'labeln'*

specifies labels for the reference lines that you request with the HREF= option.

Alias: HREFLABEL= and HREFLAB=

Restriction: The number of labels must equal the number of reference lines. Labels can have up to 16 characters.

HREFLABPOS=*n*

specifies the vertical position of HREFLABELS= labels, where *n* is

- | | |
|---|--|
| 1 | positions the labels along the top of the plot |
| 2 | staggers the labels from top to bottom |
| 3 | positions the labels along the bottom. |

Default: 1

INFONT=*font*

specifies a software font to use for text inside the framed areas of the plot. The INFONT= option takes precedence over the FTEXT= option in the GOPTIONS statement.

See also For a list of fonts, see *SAS/GRAPH Reference*.

INHEIGHT=*value*

specifies the height, in percentage screen units of text, to use inside the framed areas of the plot.

Default: the height that you specify with the HEIGHT= option. If you do not specify the HEIGHT= option, the default height is the height that you specify with the HTEXT= option in the GOPTIONS statement.

INTERTILE=*value*

specifies the distance in horizontal percentage screen units between the framed areas, which are called *tiles*.

Default: The tiles are contiguous.

Requirement: This option is not available unless you specify the CLASS statement.

L=*linetype*

specifies the line type for a diagonal distribution reference line.

Default: 1, which produces a solid line

Requirement: You must enclose this suboption in parentheses after a distribution option.

LGRID=linetype

specifies the line type for the grid that you request with the GRID= option.

Default: 1, which produces solid lines

LHREF=linetype

specifies the line type for the reference lines that you request with the HREF= option.

Alias: LH=

Default: 2, which produces a dashed line

LOGNORMAL(SIGMA=value | EST <lognormal-suboptions>)

displays a lognormal probability plot for each value of the required shape parameter σ .

Alias: LNORM

Requirement: You must specify the shape parameter with the SIGMA= suboption.

Interaction: To compute a maximum likelihood estimate for σ , specify SIGMA=EST.

Tip: To obtain a graphical estimate of σ , specify a list of values for the SIGMA= suboption, and select the value that most nearly linearizes the point pattern.

To assess the point pattern, add a diagonal distribution reference line that corresponds to the threshold parameter θ_0 and the scale parameter ζ_0 with the THETA= and ZETA= suboptions. Alternatively, you can add a line that corresponds to estimated values of θ_0 and ζ_0 with THETA=EST and ZETA=EST.

Agreement between the reference line and the point pattern indicates that the lognormal distribution with parameters σ , θ_0 , and ζ_0 is a good fit.

Main discussion: “Lognormal Distribution” on page 1537

See also: the SIGMA= suboption on page 1494, SLOPE= suboption on page 1494, THETA= suboption on page 1495, and ZETA= suboption on page 1496

LVREF=linetype

specifies the line type for the reference lines that you request with the VREF= option.

Default: 2, which produces a dashed line

MU=value | EST

specifies the mean μ_0 for a normal probability plot requested with the NORMAL option.

Default: the sample mean

Requirement: You must enclose this suboption in parentheses after the NORMAL option.

Tip: Specify the MU= and SIGMA= suboptions together to request a distribution reference line. Specify MU=EST to request a distribution reference line with μ_0 equal to the sample mean.

NADJ=value

specifies the adjustment value that is added to the sample size in the calculation of theoretical percentiles. For additional information, see Chambers et al. (1983)

Default: $\frac{1}{4}$ as recommended by Blom (1958)

NAME='string'

specifies a name for the plot, up to eight characters long, that appears in the PROC GREPLAY master menu.

Default: UNIVAR

NCOLS=n

specifies the number of columns in the comparative probability plot.

Alias: NCOL=

Default: NCOLS=1, if you specify only one class variable, and NCOLS=2, if you specify two class variables.

Requirement: This option is not available unless you specify the CLASS statement.

Interaction: If you specify two class variables, you can use the NCOLS= option with the NROWS= option.

NOFRAME

suppresses the frame around the area that is bounded by the axes.

NOHLABEL

suppresses the label for the horizontal axis.

Tip: Use this option to reduce clutter.

NORMAL<(normal-suboptions)>

displays a normal probability plot. This is the default if you omit a distribution option.

Tip: To assess the point pattern, add a diagonal distribution reference line that corresponds to μ_0 and σ_0 with the MU= and SIGMA= suboptions. Alternatively, you can add a line that corresponds to estimated values of μ_0 and σ_0 with the THETA=EST and SIGMA=EST; the estimates of the mean μ_0 and the standard deviation σ_0 are the sample mean and sample standard deviation.

Agreement between the reference line and the point pattern indicates that the normal distribution with parameters μ_0 and σ_0 is a good fit.

Main discussion: “Normal Distribution” on page 1538

See also: the MU= suboption on page 1492 and SIGMA= suboption on page 1494

NOVLABEL

suppresses the label for the vertical axis.

NOVTICK

suppresses the tick marks and tick mark labels for the vertical axis.

Interaction: This option automatically invokes the NOVLABEL option.

NROWS=*n*

specifies the number of rows in the comparative probability plot.

Alias: NROW=

Default: 2

Requirement: This option is not available unless you specify the CLASS statement.

Interaction: If you specify two class variables, you can use the NCOLS= option with the NROWS= option.

PCTLMINOR

requests minor tick marks for the percentile axis.

Interaction: The HMINOR option overrides the minor tick marks that PCTLMINOR determines.

Featured in: Example 5 on page 1555

PCTLORDER=*value(s)*

specifies the tick marks that are labeled on the theoretical percentile axis.

Default: 1, 5, 10, 25, 50, 75, 90, 95, and 99

Range: $0 \leq \text{value} \leq 100$

Restriction: The values that you specify must be in increasing order and cover the plotted percentile range. Otherwise, PROC UNIVARIATE uses the default.

RANKADJ=*value*

specifies the adjustment value that PROC UNIVARIATE adds to the ranks in the calculation of theoretical percentiles. For additional information, see Chambers et al. (1983).

Default: $-\frac{3}{8}$ as recommended by Blom (1958)

SCALE=value

is an alias for the SIGMA= option when you request probability plots with the BETA, EXPONENTIAL, GAMMA, and WEIBULL options and for the ZETA= option when you request the LOGNORMAL option.

See also: the SIGMA= suboption on page 1494 and ZETA= suboption on page 1496

SHAPE=value | EST

is an alias for the ALPHA= option when you request gamma plots with the GAMMA option, for the SIGMA= option when you request lognormal plots with the LOGNORMAL option, and for the C= option when you request Weibull plots with the WEIBULL and WEIBULL2 options.

See also: the ALPHA= suboption on page 1487, SIGMA= suboption on page 1494, and C= suboption on page 1488

SIGMA=value | EST

specifies the parameter σ , where $\sigma > 0$. The interpretation and use of the SIGMA= option depend on which distribution you specify, as shown Table 48.4 on page 1494.

Table 48.4 Uses of the SIGMA Suboption

Distribution Option	Uses of the SIGMA= Option
BETA, EXPONENTIAL GAMMA, WEIBULL	THETA= θ_0 and SIGMA= σ_0 request a distribution reference line that corresponds to θ_0 and σ_0 .
LOGNORMAL	SIGMA= $\sigma_1 \dots \sigma_n$ requests n probability plots with shape parameters $\sigma_1 \dots \sigma_n$. The SIGMA= option is required.
NORMAL	MU= μ_0 and SIGMA= σ_0 request a distribution reference line that corresponds to μ_0 and σ_0 . SIGMA=EST requests a line with σ_0 equal to the sample standard deviation.
WEIBULL2	SIGMA= σ_0 and C= c_0 request a distribution reference line that corresponds to σ_0 and c_0 .

Requirement: You must enclose this suboption in parentheses after the distribution option.

Tip: To compute a maximum likelihood estimate for σ_0 , specify SIGMA=EST.

SLOPE=value | EST

specifies the slope for a distribution reference when you request the LOGNORMAL option or WEIBULL2 option.

Requirement: You must enclose this suboption in parentheses after the distribution option.

Tip: When you use the LOGNORMAL option and SLOPE= to request the line, you must also specify a threshold parameter value θ_0 with the THETA= suboption. SLOPE= is an alternative to the ZETA= suboption for specifying ζ_0 , because the slope is equal to $\exp(\zeta_0)$.

When you use the WEIBULL2 option and SLOPE= option to request the line, you must also specify a scale parameter value σ_0 with the SIGMA= suboption. SLOPE= is an alternative to the C= suboption for specifying c_0 , because the slope is equal to $\frac{1}{c_0}$.

For example, the first and second PROBLOT statements produce the same probability plots as the third and fourth PROBLOT statements:

```
proc univariate data=measures;
  probplot width /lognormal(sigma=2 theta=0 zeta=0);
  probplot width /lognormal(sigma=2 theta=0 slope=1);
  probplot width /weibull2(sigma=2 theta=0 c=.25);
  probplot width /weibull2(sigma=2 theta=0 slope=4);
```

Main Discussion: “Three-Parameter Weibull Distribution” on page 1538

SQUARE

displays the probability plot in a square frame.

Default: rectangular frame

Featured in: Example 5 on page 1555

THETA=*value* | EST

specifies the lower threshold parameter θ for probability plots when you request the BETA, EXPONENTIAL, GAMMA, LOGNORMAL, WEIBULL, or WEIBULL2 option.

Default: 0

Requirement: You must enclose this suboption in parentheses after the distribution option.

Interaction: When you use the WEIBULL2 option, the THETA= suboption specifies the known lower threshold θ_0 , which by default is 0.

When you use the THETA= suboption with another distribution option, THETA= specifies θ_0 for a distribution reference line. To compute a maximum likelihood estimate for θ_0 , specify THETA=EST. To request the line, you must also specify a scale parameter.

THRESHOLD= *value*

is an alias for the THETA= option. See the THETA= suboption on page 1495.

VAXISLABEL=*'label'*

specifies a label for the vertical axis.

Requirement: Labels can have up to 40 characters.

Featured in:

VMINOR=*n*

specifies the number of minor tick marks between each major tick mark on the vertical axis. PROBLOT does not label minor tick marks.

Alias: VM=

Default: 0

VREF=*value(s)*

draws reference lines that are perpendicular to the vertical axis at the *value(s)* that you specify.

See also: CVREF= option on page 1490 and LVREF= option on page 1492.

VREFLABELS=*'label1'... 'labeln'*

specifies labels for the reference lines that you request with the VREF= option.

Alias: VREFLABEL= and VREFLAB=

Restriction: The number of labels must equal the number of reference lines. Labels can have up to 16 characters.

VREFLABPOS=*n*

specifies the horizontal position of VREFLABELS= labels, where *n* is

1 positions the labels at the left of the plot.

2 positions the labels at the right of the plot.

Default: 1

W=*n*

specifies the width in pixels for a diagonal distribution line.

Default: 1

Requirement: You must enclose this suboption in parentheses after the distribution option.

WAXIS=*n*

specifies the line thickness (in pixels) for the axes and frame.

Default: 1

WEIBULL(C=*value* | EST <Weibull-suboptions>)

creates a three-parameter Weibull probability plot for each value of the required shape parameter c .

Alias: WEIB

Requirement: You must specify the shape parameter with the C= suboption.

Interaction: To create a plot that is based on a maximum likelihood estimate for c , specify C=EST.

Tip: To obtain a graphical estimate of c , specify a list of values in the C= suboption. Then select the value that most nearly linearizes the point pattern.

To assess the point pattern, add a diagonal distribution reference line that corresponds to θ_0 and σ_0 with the THETA= and SIGMA= suboptions.

Alternatively, you can add a line that corresponds to estimated values of θ_0 and σ_0 with THETA=EST and SIGMA=EST.

Agreement between the reference line and the point pattern indicates that the Weibull distribution with parameters c , θ_0 , and σ_0 is a good fit.

Main discussion: “Three-Parameter Weibull Distribution” on page 1538

See also: the C= suboption on page 1488, SIGMA= suboption on page 1494, and THETA= suboption on page 1495

WEIBULL2<(Weibull-suboptions)>

creates a two-parameter Weibull probability plot. Use this distribution when your data have a *known* lower threshold θ_0 , which by default is 0. To specify the threshold value θ_0 , use the THETA= suboption.

Alias: W2

Tip: An advantage of the two-parameter Weibull plot over the three-parameter Weibull plot is that the parameters c and σ can be estimated from the slope and intercept of the point pattern. A disadvantage is that the two-parameter Weibull distribution applies only in situations where the threshold parameter is known.

Tip: To obtain a graphical estimate of θ_0 , specify a list of values for the THETA= suboption. Then select the value that most nearly linearizes the point pattern.

To assess the point pattern, add a diagonal distribution reference line that corresponds to σ_0 and c_0 with the SIGMA= and C= suboptions. Alternatively, you can add a distribution reference line that corresponds to estimated values of σ_0 and c_0 with SIGMA=EST and C=EST.

Agreement between the reference line and the point pattern indicates that the Weibull2 distribution with parameters c_0 , θ_0 , and σ_0 is a good fit.

Main discussion: “Two-Parameter Weibull Distribution” on page 1538

See also: the C= suboption on page 1488, SIGMA= suboption on page 1494, SLOPE= suboption on page 1494, and THETA= suboption on page 1495

ZETA= *value* | EST

specifies a value for the scale parameter ζ for the lognormal probability plots when you request the LOGNORMAL option.

Requirement: You must enclose this suboption in parentheses after the LOGNORMAL option.

Interaction: To request a distribution reference line with intercept θ_0 and slope $99\exp(\zeta_0)$, specify THETA= θ_0 and ZETA= ζ_0 .

QQPLOT Statement

Creates a quantile-quantile plot (Q-Q plot) (using high-resolution graphics) compares ordered variable values with quantiles of a specified theoretical distribution.

Alias: QQ

Default: Normal Q-Q plot

Restriction: You can not specify the WEIGHT statement with the QQPLOT statement.

Restriction: You can specify only one theoretical distribution.

Tip: You can use multiple QQPLOT statements.

Main Discussion: “Quantile-Quantile and Probability Plots” on page 1514

QQPLOT <variable(s)> </ option(s)>;

To do this:	Use this option:
Request a distribution	
Specify beta probability plot with required shape parameters α , β .	BETA(beta-suboptions)
Specify exponential probability plot	EXPONENTIAL(exponential-suboptions)
Specify gamma probability plot with a required shape parameter α	GAMMA(gamma-suboptions)
Specify lognormal probability plot with a required shape parameter σ	LOGNORMAL(lognormal-suboptions)
Specify normal probability plot	NORMAL(normal-suboptions)
Specify three-parameter Weibull probability plot with a required shape parameter C	WEIBULL(Weibull-suboptions)
Specify two-parameter Weibull probability plot	WEIBULL2(Weibull2-suboptions)
Distribution suboptions	
Specify shape parameter α for the beta or gamma distribution	ALPHA=
Specify shape parameter β for the beta distribution	BETA=
Specify shape parameter C for the Weibull distribution or C_0 for distribution reference line of the Weibull2 distribution	C=

To do this:	Use this option:
Specify μ_0 for distribution reference line of the normal distribution	MU=
Specify σ_0 for distribution reference line for the beta, exponential, gamma, normal, Weibull, or Weibull2 distribution or the required shape parameter σ for the lognormal option	SIGMA=
Specify slope of distribution reference line for the lognormal or Weibull2 distribution	SLOPE=
Specify θ_0 for distribution reference line for the beta, exponential, gamma, lognormal, or Weibull distribution, or the lower known threshold θ_0 for the Weibull2 distribution	THETA=
Specify ζ_0 for distribution reference line for the lognormal distribution	ZETA=
Control appearance of distribution reference line	
Specify color of distribution reference line	COLOR=
Specify line type of distribution reference line	L=
Specify width of distribution reference line	W=
Control general plot layout	
Create a grid	GRID
Specify reference lines perpendicular to the horizontal axis	HREF=
Specify labels for HREF lines	HREFLABELS=
Specify vertical position of labels for HREF= lines	HREFLABPOS=
Specify a line style for grid lines	LGRID=
Adjust sample size when computing quantiles	NADJ=
Suppress frame around plotting area	NOFRAME
Suppress label for horizontal axis	NOHLABEL
Suppress label for vertical axis	NOVLABEL
Suppress tick marks and tick mark labels for vertical axis	NOVTICK
Display a nonlinear percentile axis	PCTLAXIS<(axis-options)>
Request minor tick marks for percentile axis	PCTLMINOR
Replace theoretical quantiles with percentiles	PCTLSCALE
Adjust ranks when computing quantiles	RANKADJ=
Display Q-Q plot in square format	SQUARE
Specify label for vertical axis	VAXISLABEL=
Specify reference lines perpendicular to the vertical axis	VREF=
Specify labels for VREF lines	VREFLABELS=

To do this:	Use this option:
Specify horizontal position of labels for VREF= lines	VREFLABPOS=
Specify line thickness for axes and frame	WAXIS=
Enhance the Q-Q plot	
Specify annotate data set	ANNOTATE=
Specify color for axis	CAXIS=
Specify color for frame	CFRAME=
Specify color for grid lines	CGRID=
Specify color for HREF= lines	CHREF=
Specify color for text	CTEXT=
Specify color for VREF= lines	CVREF=
Specify description for plot in graphics catalog	DESCRIPTION=
Specify software font for text	FONT=
Specify height of text used outside framed areas	HEIGHT=
Specify number of minor tick marks on horizontal axis	HMINOR=
Specify software font for text inside framed areas	INFONT=
Specify height of text inside framed areas	INHEIGHT=
Specify line style for HREF= lines	LHREF=
Specify line style for VREF= lines	LVREF=
Specify name for plot in graphics catalog	NAME=
Specify number of minor tick marks on vertical axis	VMINOR=
Enhance the comparative Q-Q plot	
Apply annotation requested in ANNOTATE= data set to key cell only	ANNOKEY
Specify color for filling frame for row labels	CFRAMESIDE=
Specify color for filling frame for column labels	CFRAMETOP=
Specify distance between tiles	INTERTILE=
Specify number of columns in comparative Q-Q plot	NCOLS=
Specify number of rows in comparative Q-Q plot	NROWS=

Arguments

variable(s)

identifies one or more variables that the procedure uses to create Q-Q plots.

Default: If you omit *variable(s)* in the QQPLOT statement, then the procedure creates a Q-Q plot for each variable that you list in the VAR statement, or for each numeric variable in the DATA= data set if you omit a VAR statement.

Requirement: If you specify a VAR statement, use the *variable(s)* that you list in the VAR statement. Otherwise, *variable(s)* are any numeric variables in the DATA= data set.

Options

ALPHA=*value* | EST

specifies the required shape parameter α ($\alpha > 0$) for quantile plots when you request the BETA or GAMMA options. The QQPLOT statement creates a plot for each value that you specify.

Requirement: Enclose this suboption in parentheses when it follows the BETA or GAMMA options.

Tip: To compute a maximum likelihood estimate for α , specify ALPHA=EST.

ANNOKEY

specifies to apply the annotation that you requested with the ANNOTATE= option to the *key cell* only. By default, PROC UNIVARIATE applies annotation to all of the cells.

Requirement: This option is not available unless you specify the CLASS statement.

Tip: Use the KEYLEVEL= option in the CLASS statement to specify the key cell.

See also: the KEYLEVEL= option on page 1452

ANNOTATE=SAS-*data-set*

specifies an input data set that contains annotate variables as described in *SAS/GRAPH Reference*.

Alias: ANNO=

Tip: The ANNOTATE = data set that you specify in the QQPLOT statement is used by all plots that this statement creates. You can also specify an ANNOTATE= data set in the PROC UNIVARIATE statement to enhance all the graphic displays that the procedure creates.

See also: ANNOTATE= on page 1445 in the PROC UNIVARIATE statement

BETA(ALPHA=*value* | EST BETA=*value* | EST <*beta-suboptions*>)

displays a beta Q-Q plot for each combination of the required shape parameters α and β .

Requirement: You must specify the shape parameters with the ALPHA= and BETA= suboptions

Interaction: To create a plot that is based on maximum likelihood estimates for α and β , specify ALPHA=EST and BETA=EST.

Tip: To obtain graphical estimates of α and β , specify lists of values in the ALPHA= and BETA= suboptions. Then select the combination of α and β that most nearly linearizes the point pattern.

To assess the point pattern, add a diagonal distribution reference line that corresponds to the lower threshold parameter θ_0 and the scale parameter σ_0 with the THETA= and SIGMA= suboptions. Alternatively, you can add a line that corresponds to estimated values of lower threshold parameter θ_0 and σ_0 with THETA=EST and SIGMA=EST.

Agreement between the reference line and the point pattern indicates that the beta distribution with parameters α , β , θ_0 , and σ_0 is a good fit.

Main discussion: “Beta Distribution” on page 1536

See also: the ALPHA= suboption on page 1500, BETA= suboption on page 1501, SIGMA= suboption on page 1506, and THETA= suboption on page 1507.

BETA=*value* | EST

specifies the shape parameter β ($\beta > 0$) for Q-Q plots when you request the BETA distribution option. PROC UNIVARIATE creates a plot for each value that you specify.

Alias: B=

Requirement: You must enclose this suboption in parentheses after the BETA option.

Tip: To compute a maximum likelihood estimate for β , specify BETA=EST.

C=*value* | EST

specifies the shape parameter c ($c > 0$) for Q-Q plots when you request the WEIBULL option or WEIBULL2 option. C= is a required suboption in the WEIBULL option.

Requirement: Enclose this suboption in parentheses after the WEIBULL option or WEIBULL2 option.

Interaction: To request a distribution reference line in the WEIBULL2 option, you must specify both the C= and SIGMA= suboptions.

Tip: To compute a maximum likelihood estimate for c , specify C=EST.

CAXIS=*color*

specifies the color for the axes.

Alias: CAXES=

Default: the first color in the device color list

Interaction: This option overrides any COLOR= specification.

CFRAME=*color*

specifies the color for the area that is enclosed by the axes and frame.

Default: the area is not filled

Featured in: Example 8 on page 1566

CFRAMESIDE=*color*

specifies the color to fill the frame area for the row labels that display along the left side of the comparative probability plot. This color also fills the frame area for the label of the corresponding class variable (if you associate a label with the variable).

Default: These areas are not filled.

Requirement: This option is not available unless you specify the CLASS statement.

CFRAMETOP=*color*

specifies the color to fill the frame area for the column labels that display across the top of the comparative probability plot. This color also fills the frame area for the label of the corresponding class variable (if you associate a label with the variable).

Default: These areas are not filled.

Requirement: This option is not available unless you specify the CLASS statement.

CGRID=*color*

specifies the color for grid lines when a grid displays on the plot.

Default: the first color in the device color list

Interaction: This option automatically invokes the GRID= option.

CHREF=*color*

specifies the color for horizontal axis reference lines when you specify the HREF= option.

Alias: CH=

Default: the first color in the device color list

COLOR=*color*

specifies the color for a distribution reference line.

Default: the fourth color in the device color list

Requirement: You must enclose this suboption in parentheses after a distribution option keyword.

CTEXT=*color*

specifies the color for tick mark values and axis labels.

Default: the color that you specify for the CTEXT= option in the GOPTIONS statement. If you omit the GOPTIONS statement, the default is the first color in the device color list.

CVREF=*color*

specifies the color for the reference lines that you request with the VREF= option.

Alias: CV=

Default: the first color in the device color list.

DESCRIPTION='*string*'

specifies a description, up to 40 characters long, that appears in the PROC GREPLAY master menu.

Alias: DES=

Default: the variable name

EXPONENTIAL<(exponential-suboptions)>

displays an exponential Q-Q plot.

Alias: EXP

Tip: To assess the point pattern, add a diagonal distribution reference line that corresponds to θ_0 and σ_0 with the THETA= and SIGMA= suboptions. Alternatively, you can add a line that corresponds to estimated values of the threshold parameter θ_0 and the scale parameter σ_0 with the THETA=EST and SIGMA=EST suboptions.

Agreement between the reference line and the point pattern indicates that the exponential distribution with parameters θ_0 and σ_0 is a good fit.

Main discussion: "Exponential Distribution" on page 1537

See also: the SIGMA suboption on page 1506 and THETA suboption on page 1507

FONT=*font*

specifies a software font for the reference lines and the axis labels.

Default: hardware characters

Interaction: FONT=*font* takes precedence over FTEXT=*font* that you specify in the GOPTIONS statement.

GAMMA(ALPHA=*value* | EST <gamma-suboptions>)

displays a gamma Q-Q plot for each value of the required shape parameter α .

Requirement: You must specify the shape parameter with the ALPHA= suboption.

Interaction: To create a plot that is based on a maximum likelihood estimate for α , specify ALPHA=EST.

Tip: To obtain a graphical estimate of α , specify a list of values in the ALPHA= suboption. Then select the value that most nearly linearizes the point pattern.

To assess the point pattern, add a diagonal distribution reference line that corresponds to θ_0 and σ_0 with the THETA= and SIGMA= suboptions. Alternatively, you can add a line that corresponds to estimated values of the threshold parameter θ_0 and the scale parameter σ_0 with THETA=EST and SIGMA=EST.

Agreement between the reference line and the point pattern indicates that the exponential distribution with parameters α , θ_0 , and σ_0 is a good fit.

Main discussion: “Gamma Distribution” on page 1537

See also: the ALPHA= suboption on page 1500, SIGMA= suboption on page 1506, and THETA= suboption on page 1507

GRID

specifies to display a grid on the plot. Grid lines are horizontal lines that are positioned at major tick marks on the vertical axis.

See also: the CGRID= option

HEIGHT=*value*

specifies the height in percentage screen units of text for axis labels, tick mark labels, and legends. This option takes precedence over the HTEXT= option in the GOPTIONS statement.

HMINOR=*n*

specifies the number of minor tick marks between each major tick mark on the horizontal axis. PROC UNIVARIATE does not label minor tick marks.

Alias: HM=

Default: 0

HREF=*value(s)*

draws reference lines that are perpendicular to the horizontal axis at the values you specify.

Restriction: When you use the PCTLAXIS option, then HREF=values must be in quantile units.

See also: CHREF= option on page 1501

HREFLABELS=*'label1' ... 'labeln'*

specifies labels for the reference lines that you request with the HREF= option.

Alias: HREFLABEL= and HREFLAB=

Restriction: The number of labels must equal the number of reference lines. Labels can have up to 16 characters.

HREFLABPOS=*n*

specifies the vertical position of HREFLABELS= labels, where *n* is

- | | |
|---|--|
| 1 | positions the labels along the top of the plot |
| 2 | staggers the labels from top to bottom |
| 3 | positions the labels along the bottom. |

Default: 1

INFONT=*font*

specifies a software font to use for text inside the framed areas of the plot. The INFONT= option takes precedence over the FTEXT= option in the GOPTIONS statement.

See also: For a list of fonts, see *SAS/GRAPH Reference*.

INHEIGHT=*value*

specifies the height, in percentage screen units of text, to use inside the framed areas of the plot.

Default: the height that you specify with the HEIGHT= option. If you do not specify the HEIGHT= option, the default height is the height that you specify with the HTEXT= option in the GOPTIONS statement.

INTERTILE=*value*

specifies the distance in horizontal percentage screen units between the framed areas, which are called *tiles*.

Default: .75 in percentage screen units.

Requirement: This option is not available unless you specify the CLASS statement.

L=*linetype*

specifies the line type for a diagonal distribution reference line.

Default: 1, which produces a solid line

Requirement: You must enclose this suboption in parentheses after a distribution option keyword.

LGRID=*linetype*

specifies the line type for the grid when a grid displays on the plot.

Default: 1, which produces a solid line

Interaction: This option automatically invokes the GRID= option.

LHREF=*linetype*

specifies the line type for the reference lines that you request with the HREF= option.

Alias: LH=

Default: 2, which produces a dashed line

LOGNORMAL(SIGMA=*value* | **EST** <*lognormal-suboptions*>)

displays a lognormal Q-Q plot for each value of the required shape parameter σ .

Alias: LNORM

Requirement: You must specify the shape parameter with the SIGMA= suboption.

Tip: To obtain a graphical estimate of σ , specify a list of values for the SIGMA= suboption, and select the value that most nearly linearizes the point pattern.

To assess the point pattern, add a diagonal distribution reference line that corresponds to the threshold parameter θ_0 and the scale parameter ζ_0 with the THETA= and ZETA= suboptions. Alternatively, you can add a line that corresponds to estimated values of θ_0 and ζ_0 with THETA=EST and ZETA=EST. This line has intercept θ_0 , and slope $\exp(\zeta_0)$.

Agreement between the reference line and the point pattern indicates that the lognormal distribution with parameters σ , θ_0 and ζ_0 is a good fit.

Main discussion: “Lognormal Distribution” on page 1537

See also: the SIGMA= suboption on page 1506, SLOPE= suboption on page 1507, THETA= suboption on page 1507, and ZETA= suboption on page 1509

LVREF=*linetype*

specifies the line type for the reference lines that you request with the VREF= option.

Alias: LV=

Default: 2, which produces a dashed line

MU=*value* | **EST**

specifies the mean μ for a normal Q-Q plot requested with the NORMAL option.

Default: the sample mean

Requirement: You must enclose this suboption in parentheses after the NORMAL option.

Tip: Specify the MU= and SIGMA= suboptions together to request a distribution reference line. Specify MU=EST to request a distribution reference line with μ_0 equal to the sample mean.

Featured in: Example 8 on page 1566

NADJ=value

specifies the adjustment value that is added to the sample size in the calculation of theoretical quantiles. For additional information, see Chambers et al. (1983).

Default: $\frac{1}{4}$ as recommended by Blom (1958)

NAME='string'

specifies a name for the plot, up to eight characters long, that appears in the PROC GREPLAY master menu.

Default: UNIVAR

NCOLS=n

specifies the number of columns in the comparative Q-Q plot.

Alias: NCOL=

Default: NCOLS=1, if you specify only one class variable, and NCOLS=2, if you specify two class variables.

Requirement: This option is not available unless you specify the CLASS statement.

Interaction: If you specify two class variables, you can use the NCOLS= option with the NROWS= option.

NOFRAME

suppresses the frame around the area that is bounded by the axes.

Restriction: If you use the PCTLAXIS option, then you cannot use the NOFRAME option.

NOHLABEL

suppresses the label for the horizontal axis.

Tip: Use this option to reduce clutter.

NORMAL<(normal-suboptions)>

displays a normal Q-Q plot. This is the default if you omit a distribution option.

Tip: To assess the point pattern, add a diagonal distribution reference line that corresponds to μ_0 and σ_0 with the MU= and SIGMA= suboptions. Alternatively, you can add a line that corresponds to estimated values of μ_0 and σ_0 with the THETA=EST and SIGMA=EST; the estimates of the mean μ_0 and the standard deviation σ_0 are the sample mean and sample standard deviation.

Agreement between the reference line and the point pattern indicates that the normal distribution with parameters μ_0 and σ_0 is a good fit.

Main discussion: "Normal Distribution" on page 1538

See also: the MU= suboption on page 1504 and SIGMA= suboption on page 1506

Featured in: Example 8 on page 1566

NOVLABEL

suppresses the label for the vertical axis.

NOVTICK

suppresses the tick marks and tick mark labels for the vertical axis.

Interaction: This option automatically invokes the NOVLABEL option.

NROWS=n

specifies the number of rows in the comparative Q-Q plot.

Alias: NROW=

Default: 2

Requirement: This option is not available unless you specify the CLASS statement.

Interaction: If you specify two class variables, you can use the NCOLS= option with the NROWS= option.

PCTLAXIS<(axis-options) >

adds a nonlinear percentile axis along the frame of the Q-Q plot opposite the theoretical quantile axis. The added axis is identical to the axis for probability plots produced with the PROBPLOT statement. You can specify the following axis options:

GRID

draws vertical grid lines at major percentiles.

GRIDCHAR=*character*

specifies grid line plotting character on line printer.

LABEL=*string*

specifies label for percentile axis.

LGRID=*linetype*

specifies label for percentile axis.

Restriction: When you use the PCTLAXIS option, then you must specify HREF=values in quantile units, and you cannot use the NOFRAME option.

Featured in: Example 8 on page 1566

PCTLMINOR

requests minor tick marks for the percentile axis when you specify PCTLAXIS.

Interaction: The HMINOR option overrides the minor tick marks that PCTLMINOR determines.

PCTLSCALE

requests scale labels for the theoretical quantile axis in percentile units, resulting in a nonlinear axis scale.

Tip: Tick marks are drawn uniformly across the axis based on the quantile scale. In all other respects, the plot remains the same, and you must specify HREF= values in quantile units. For a true nonlinear axis, use the PROBPLOT statement.

RANKADJ=*value*

specifies the adjustment value that PROC UNIVARIATE adds to the ranks in the calculation of theoretical quantiles. For additional information, see Chambers et al. (1983).

Default: $-\frac{3}{8}$ as recommended by Blom (1958)

SCALE=*value*

is an alias for the SIGMA= option when you request Q-Q plots with the BETA, EXPONENTIAL, GAMMA, WEIBULL, and WEIBULL2 options and for the ZETA= option when you request the LOGNORMAL option.

See also: the SIGMA= suboption on page 1506 and the ZETA= suboption on page 1509

SHAPE=*value* | **EST**

is an alias for the ALPHA=option when you request gamma plots with the GAMMA option, for the SIGMA= option when you request lognormal plots with the LOGNORMAL option, and for the C= option when you request Weibull plots with the WEIBULL, and WEIBULL2 options.

See also: the ALPHA= suboption on page 1500, the SIGMA= suboption on page 1506, and the C= suboption on page 1501

SIGMA=*value* | **EST**

specifies the distribution parameter σ , where $\sigma > 0$ for the quantile plot. The interpretation and use of the SIGMA= option depend on which distribution you specify, as shown in Table 48.5 on page 1507.

Table 48.5 Uses of the SIGMA Suboption

Distribution Option	Uses of the SIGMA= Option
BETA, EXPONENTIAL GAMMA, WEIBULL	THETA= θ_0 and SIGMA= σ_0 request a distribution reference line with intercept θ_0 and slope σ_0 .
LOGNORMAL	SIGMA= $\sigma_1 \dots \sigma_n$ requests n Q-Q plots with shape parameters $\sigma_1 \dots \sigma_n$. The SIGMA= option is required.
NORMAL	MU= μ_0 and SIGMA= σ_0 request a distribution reference line with intercept μ_0 and slope σ_0 . SIGMA=EST requests a slope σ_0 equal to the sample standard deviation.
WEIBULL2	SIGMA= σ_0 and C= c_0 request a distribution reference line with intercept $\log(\sigma_0)$ and slope $\frac{1}{c_0}$.

Requirement: Enclose this suboption in parentheses after the distribution option.

Tip: To compute a maximum likelihood estimate for σ_0 , specify SIGMA=EST.

Featured in: Example 8 on page 1566

SLOPE=value|EST

specifies the slope for a distribution reference when you request the LOGNORMAL option or WEIBULL2 option.

Requirement: Enclose this suboption in parentheses after the distribution option.

Tip: When you use the LOGNORMAL option and SLOPE= to request the line, you must also specify a threshold parameter value θ_0 with the THETA= suboption.

SLOPE= is an alternative to the ZETA= suboption for specifying ζ_0 , because the slope is equal to $\exp(\zeta_0)$.

When you use the WEIBULL2 option and SLOPE= option to request the line, you must also specify a scale parameter value σ_0 with the SIGMA= suboption.

SLOPE= is an alternative to the C= suboption for specifying c_0 , because the slope is equal to $\frac{1}{c_0}$.

For example, the first and second QQPLOT statements produce the same quantile-quantile plots as the third and fourth QQPLOT statements:

```
proc univariate data=measures;
  qqplot width /lognormal(sigma=2 theta=0 zeta=0);
  qqplot width /lognormal(sigma=2 theta=0 slope=1);
  qqplot width /weibull2(sigma=2 theta=0 c=.25);
  qqplot width /weibull2(sigma=2 theta=0 slope=4);
```

Main Discussion: “Shape Parameters” on page 1539

SQUARE

displays the Q-Q plot in a square frame.

Default: rectangular frame

THETA=value|EST

specifies the lower threshold parameter θ for Q-Q plots when you request BETA, EXPONENTIAL, GAMMA, LOGNORMAL, WEIBULL, or WEIBULL2 option.

Default: 0

Requirement: You must enclose this suboption in parentheses after the distribution option.

Interaction: When you use the WEIBULL2 option, the THETA= suboption specifies the known lower threshold θ_0 , which by default is 0.

When you use the THETA= suboption with another distribution option, THETA= specifies θ_0 for a distribution reference line. To compute a maximum likelihood estimate for θ_0 , specify THETA=EST. To request the line, you must also specify a scale parameter.

THRESHOLD= *value* | EST

is an alias for the THETA= option. See the THETA= suboption on page 1507.

VAXISLABEL= '*label*'

specifies a label for the vertical axis.

Requirement: Labels can have up to 40 characters.

Featured in:

VMINOR=*n*

specifies the number of minor tick marks between each major tick mark on the vertical axis. QQPLOT does not label minor tick marks.

Alias: VM=

Default: 0

VREF=*value(s)*

draws reference lines that are perpendicular to the vertical axis at the *value(s)* you specify.

See also: CVREF= option on page 1502 and LVREF= option on page 1504

VREFLABELS= '*label1*' ... '*labeln*'

specifies labels for the reference lines that you request with the VREF= option.

Alias: VREFLABEL= and VREFLAB=

Restriction: The number of labels must equal the number of reference lines. Labels can have up to 16 characters.

VREFLABPOS=*n*

specifies the horizontal position of VREFLABELS= labels, where *n* is

- 1 positions the labels at the left of the plot.
- 2 positions the labels at the right of the plot.

Default: 1

W=*n*

specifies the width in pixels for a distribution reference line.

Default: 1

Requirement: You must enclose this suboption in parentheses after the distribution option.

WAXIS=*n*

specifies the line thickness (in pixels) for the axes and frame.

Default: 1

WEIBULL(C=*value* | EST <*Weibull-suboptions*>)

creates a three-parameter Weibull Q-Q plot for each value of the required shape parameter *c*.

Alias: WEIB

Requirement: You must specify the shape parameter with the C= suboption.

Interaction: To create a plot that is based on a maximum likelihood estimate for *c*, specify C=EST.

To specify the threshold value θ_0 , use the THETA= suboption.

Tip: To obtain a graphical estimate of c , specify a list of values in the C= suboption. Then select the value that most nearly linearizes the point pattern.

To assess the point pattern, add a diagonal distribution reference line with intercept θ_0 and slope σ_0 with the THETA= and SIGMA= suboptions. Alternatively, you can add a line that corresponds to estimated values of θ_0 and σ_0 with THETA=EST and SIGMA=EST.

Agreement between the reference line and the point pattern indicates that the Weibull distribution with parameters c , θ_0 , and σ_0 is a good fit.

Main discussion: “Three-Parameter Weibull Distribution” on page 1538

See also: the C= suboption on page 1501, SIGMA= suboption on page 1506, and THETA= suboption on page 1507

WEIBULL2<(Weibull-suboptions)>

creates a two-parameter Weibull Q-Q plot. Use this distribution when your data have a *known* lower threshold θ_0 , which by default is 0. To specify the threshold value θ_0 , use the THETA= suboption.

Note: The C= shape parameter option is not required with the Weibull2 option. Δ

Alias: W2

Default: 0

Interaction: To specify the threshold value θ_0 , use the THETA= suboption.

Tip: An advantage of the two-parameter Weibull plot over the three-parameter Weibull plot is that the parameters c and σ can be estimated from the slope and intercept of the point pattern. A disadvantage is that the two-parameter Weibull distribution applies only in situations where the threshold parameter is known.

Tip: To obtain a graphical estimate of θ_0 , specify a list of values for the THETA= suboption. Then select the value that most nearly linearizes the point pattern. To assess the point pattern, add a diagonal distribution reference line that corresponds to σ_0 and c_0 with the SIGMA= and C= suboptions. Alternatively, you can add a distribution reference line that corresponds to estimated values of σ_0 and c_0 with SIGMA=EST and C=EST.

Agreement between the reference line and the point pattern indicates that the Weibull2 distribution with parameters c_0 , θ_0 , and σ_0 is a good fit.

Main discussion: “Two-Parameter Weibull Distribution” on page 1538

See also: the C= suboption on page 1501, SIGMA= suboption on page 1506, SLOPE= suboption on page 1507, and THETA= suboption on page 1507

ZETA= value | EST

specifies a value for the scale parameter ζ for the lognormal Q-Q plots when you request the LOGNORMAL option.

Requirement: You must enclose this suboption in parentheses after the LOGNORMAL option.

Interaction: To request a distribution reference line with intercept θ_0 and slope $\exp(\zeta_0)$, specify THETA= θ_0 and ZETA= ζ_0 .

Theoretical Percentiles of Quantile-Quantile Plots

To estimate percentiles from a Q-Q plot

- ☐ Specify the PCTLAXIS option, which adds a percentile axis opposite the theoretical quantile axis. The scale for the percentile axis ranges between 0 and 100 with tick marks at percentile values such as 1, 5, 10, 25, 50, 75, 90, 95, and 99.
- ☐ Specify the PCTLSCALE option, which relabels the horizontal axis tick marks with their percentile equivalents but does not alter their spacing. For example, on

a normal Q-Q plot, the tick mark labeled 0 is relabeled as 50 because the 50th percentile corresponds to the zero quantile.

You can also use the PROBPLOT statement to estimate percentiles.

VAR Statement

Specifies the analysis variables and their order in the results.

Default: If you omit the VAR statement, PROC UNIVARIATE analyzes all numeric variables that are not listed in the other statements.

Featured in: Example 1 on page 1543 and Example 6 on page 1560

VAR *variable(s)*;

Required Arguments

variable(s)

identifies one or more analysis variables.

Using the Output Statement with the VAR Statement

Use a VAR statement when you use an OUTPUT statement. To store the same statistic for several analysis variables in the OUT= data set, you specify a list of names in the OUTPUT statement. PROC UNIVARIATE makes a one-to-one correspondence between the order of the analysis variables in the VAR statement and the list of names that follow a statistic keyword.

WEIGHT Statement

Specifies weights for analysis variables in the statistical calculations.

See also: For information about how to calculate weighted statistics and for an example that uses the WEIGHT statement, see “Calculating Weighted Statistics” on page 60

WEIGHT *variable*;

Required Arguments

variable

specifies a numeric variable whose values weight the values of the analysis variables. The values of the variable do not have to be integers. If the value of the weight variable is

Weight value...	PROC UNIVARIATE...
0	counts the observation in the total number of observations
less than 0	converts the weight value to zero and counts the observation in the total number of observations
missing	excludes the observation

To exclude observations that contain negative and zero weights from the analysis, use EXCLNPWGT. Note that most SAS/STAT procedures, such as PROC GLM, exclude negative and zero weights by default.

The weight variable does not change how the procedure determines the range, mode, extreme values, extreme observations, or number of missing values. The Student's t test is the only test of location that PROC UNIVARIATE computes when you weight the analysis variables.

Restriction: The CIPCTLDF, CIPCTLNORMAL, LOCCOUNT, NORMAL, ROBUSTSCALE, TRIMMED=, and WINSORIZED= options are not available with the WEIGHT statement.

Restriction: To compute weighted skewness or kurtosis, use VARDEF=DF or VARDEF=N in the PROC statement.

Restriction: You can not specify the HISTOGRAM, PROBPLOT, or QQPLOT statements with the WEIGHT statement.

Tip: When you use the WEIGHT statement, consider which value of the VARDEF= option is appropriate. See VARDEF= on page 1449 and the calculation of weighted statistics in “Keywords and Formulas” on page 1578 for more information.

Note: Prior to Version 7 of the SAS System, the procedure did not exclude the observations with missing weights from the count of observations. Δ

Concepts: UNIVARIATE Procedure

Rounding

When you specify ROUND= u , PROC UNIVARIATE rounds a variable by using the rounding unit to divide the number line into intervals with midpoints $u*i$, where u is the nonnegative rounding unit and i equals the integers (... , -4, -3, -2, -1, 0, 1, 2, 3, 4,...). The interval width is u . Any variable value that falls in an interval rounds to the midpoint of that interval. A variable value that is midway between two midpoints, and is therefore on the boundary of two intervals, rounds to the even midpoint. Even midpoints occur when i is an even integer (0, ± 2 , ± 4 , ...).

When ROUND=1 and the analysis variable values are between -2.5 and 2.5, the intervals are as follows:

i	Interval	Midpoint	Left endpt rounds to	Right endpt rounds to
-2	[-2.5,-1.5]	-2	-2	-2
-1	[-1.5,-0.5]	-1	-2	0
0	[-0.5,0.5]	0	0	0

i	Interval	Midpoint	Left endpt rounds to	Right endpt rounds to
1	[0.5,1.5]	1	0	2
2	[1.5,2.5]	2	2	2

When ROUND=.5 and the analysis variable values are between -1.25 and 1.25, the intervals are as follows:

i	Interval	Midpoint	Left endpt rounds to	Right endpt rounds to
-2	[-1.25,-0.75]	-1.0	-1	-1
-1	[-0.75,-0.25]	-0.5	-1	0
0	[-0.25,0.25]	0.0	0	0
1	[0.25,0.75]	0.5	0	1
2	[0.75,1.25]	1.0	1	1

As the rounding unit increases, the interval width also increases. This reduces the number of unique values and decreases the amount of memory that PROC UNIVARIATE needs.

Generating Line Printer Plots

The PLOTS option in the PROC UNIVARIATE statement provides up to four diagnostic line printer plots to examine the data distribution. These plots are the stem-and-leaf plot or horizontal bar chart, the box plot, the normal probability plot, and the side-by-side box plots. If you specify the WEIGHT statement, PROC UNIVARIATE provides a weighted histogram, a weighted box plot based on the weighted quantiles, and a weighted normal probability plot.

Stem-and-Leaf Plot

The first plot in the output is either a stem-and-leaf plot (Tukey 1977) or a horizontal bar chart. If any single interval contains more than 49 observations, the horizontal bar chart appears. Otherwise, the stem-and-leaf plot appears. The stem-and-leaf plot is like a horizontal bar chart in that both plots provide a method to visualize the overall distribution of the data. The stem-and-leaf plot provides more detail because each point in the plot represents an individual data value.

To change the number of stems that the plot displays, use PLOTSIZE= to increase or decrease the number of rows. Instructions that appear below the plot explain how to determine the values of the variable. If no instructions appear, you multiply *Stem.Leaf* by 1 to determine the values of the variable. For example, if the stem value is 10 and the leaf value is 1, then the variable value is approximately 10.1.

For the stem-and-leaf plot, the procedure rounds a variable value to the nearest leaf. If the variable value is exactly halfway between two leaves, the value rounds to the nearest leaf with an even integer value. For example, a variable value of 3.15 has a stem value of 3 and a leaf value of 2.

Box Plot

The box plot, also known as a schematic plot, appears beside the stem-and-leaf plot. Both plots use the same vertical scale. The box plot provides a visual summary of the data and identifies outliers. The bottom and top edges of the box correspond to the sample 25th (Q1) and 75th (Q3) percentiles. The box length is one *interquartile range* (Q3 - Q1). The center horizontal line with asterisk endpoints corresponds to the sample median. The central plus sign (+) corresponds to the sample mean. If the mean and median are equal, the plus sign falls on the line inside the box. The vertical lines that project out from the box, called *whiskers*, extend as far as the data extend, up to a distance of 1.5 interquartile ranges. Values farther away are potential outliers. The procedure identifies the extreme values with a zero or an asterisk (*). If zero appears, the value is between 1.5 and 3 interquartile ranges from the top or bottom edge of the box. If an asterisk appears, the value is more extreme.

To generate box plot using high-resolution graphics, use the BOXPLOT procedure in SAS/STAT software.

Normal Probability Plot

The normal probability plot is a quantile-quantile plot of the data. The procedure plots the empirical quantiles against the quantiles of a standard normal distribution. Asterisks (*) indicate the data values. The plus signs (+) provide a straight reference line that is drawn by using the sample mean and standard deviation. If the data are from a normal distribution, the asterisks tend to fall along the reference line. The vertical coordinate is the data value, and the horizontal coordinate is $\Phi^{-1}(v_i)$ where

$$\Phi^{-1}((r_i - 3/8) / (n + 1/4))$$

and where

v_i is $(r_i - \frac{3}{8}) / (n + \frac{1}{4})$.

Φ^{-1} is the inverse of the standard normal distribution function.

r_i is the rank of the i th data value when ordered from smallest to largest.

n is the number of nonmissing data values.

For weighted normal probability plot, the i th ordered observation is plotted against the normal quantile $\Phi^{-1}(v_i)$, where Φ^{-1} is the inverse standard cumulative normal distribution and

$$v_i = \frac{\sum_{j=1}^i w_{(j)} (1 - \frac{3}{8i})}{W (1 + \frac{1}{4n})}$$

where $w_{(j)}$ is weight that is associated with $y_{(j)}$ for the j th ordered observation and

$W = \sum_{i=1}^n w_i$ is the sum of the individual weights.

When each observation has an identical weight, $w_{(j)} = w$, the formula for v_i reduces to the expression for v_i in the unweighted normal probability plot

$$v_i = \frac{i - \frac{3}{8}}{n + \frac{1}{4}}$$

When the value of VARDEF= is WDF or WEIGHT, PROC UNIVARIATE draws a reference line with intercept $\hat{\mu}$ and slope $\hat{\sigma}$ and when the value of VARDEF= is DF or N, the slope is $\hat{\sigma}/\sqrt{\bar{w}}$ where $\bar{w} = W/n$ is the average weight.

When each observation has an identical weight and the value of VARDEF= is DF, N, or WEIGHT, the reference line reduces to the usual reference line with intercept $\hat{\mu}$ and slope $\hat{\sigma}$ in the unweighted normal probability plot.

If the data are normally distributed with mean μ , standard deviation σ , and each observation has an identical weight w , then, as in the unweighted normal probability plot, the points on the plot should lie approximately on a straight line. The intercept is μ and slope is σ when VARDEF= is WDF or WEIGHT, and the slope is σ/\sqrt{w} when VARDEF= is DF or N.

Side-by-Side Box Plots

When you use a BY statement with the PLOT option, PROC UNIVARIATE produces full-page side-by-side box plots, one for each BY group. The box plots (also known as schematic plots) use a common scale that allows you to compare the data distribution across BY groups. This plot appears after the univariate analyses of all BY groups. Use the NOBYPLOT option to suppress this plot.

For more information on how to interpret these plots see *SAS System for Elementary Statistical Analysis* and *SAS System for Statistical Graphics*.

Generating High-Resolution Graphics

If your site licenses SAS/GRAPH software, you can use the HISTOGRAM statement, PROBLOT statement, and QQPLOT statement to create high-resolution graphs.

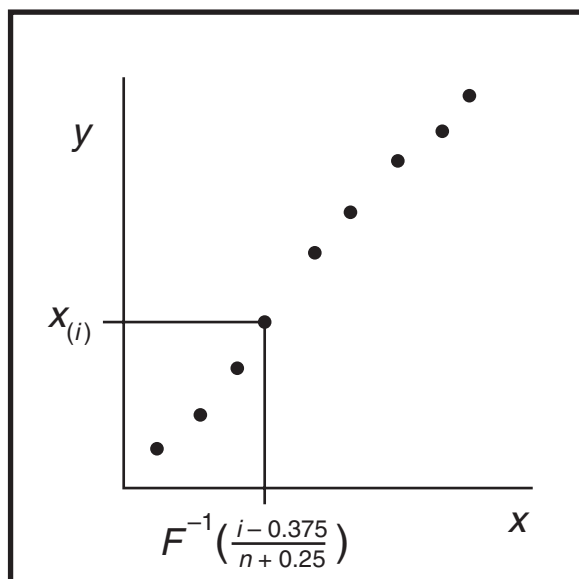
The HISTOGRAM statement generates histograms and comparative histograms that allow you to examine the data distribution. You can optionally fit families of density curves and superimpose kernel density estimates on the histograms. For additional information about the fitted distributions and kernel density estimates, see “Formulas for Fitted Continuous Distributions” on page 1530.

The PROBLOT statement generates a probability plot, which compares ordered values of a variable with percentiles of a specified theoretical distribution. The QQPLOT statement generates a quantile-quantile plot, which compares ordered values of a variable with quantiles of a specified theoretical distribution. Thus, you can use these plots to determine how well a theoretical distribution models a set of measures.

Quantile-Quantile and Probability Plots

The following figure illustrates how to construct a Q-Q plot for a specified theoretical distribution $F(x)$ with the QQPLOT statement.

Figure 48.6 Construction of a Q-Q Plot



First, the n nonmissing values of the variable are ordered from smallest to largest: $x_{(1)} \leq x_{(2)} \leq \dots \leq x_{(n)}$. Then, the i^{th} ordered value $x_{(i)}$ is represented on the plot by a point whose y -coordinate is $x_{(i)}$ and whose x -coordinate is $F^{-1}\left(\frac{i-0.375}{n+0.25}\right)$, where $F(\cdot)$ is the theoretical distribution with a zero location parameter and a unit scale parameter. For additional information about the theoretical distributions that you can request, see “Theoretical Distributions for Quantile-Quantile and Probability Plots” on page 1536.

You can modify the adjustment constants -0.375 and 0.25 with the RANKADJ= and NADJ= options. The default combination is recommended by Blom (1958). For additional information, see Chambers et al. (1983). Since $x_{(i)}$ is a quantile of the empirical cumulative distribution function (ecdf), a Q-Q plot compares quantiles of the ecdf with quantiles of a theoretical distribution. Probability plots are constructed the same way, except that the x -axis is scaled nonlinearly in percentiles.

Interpreting Quantile-Quantile and Probability Plots

If the data distribution matches the theoretical distribution, the points on the plot form a linear pattern. Thus, you can use a Q-Q plot or a probability plot to determine how well a theoretical distribution models a set of measurements. The following properties of these plots make them useful diagnostics to test how well a specified theoretical distribution fits a set of measurements:

- If the quantiles of the theoretical and data distributions agree, the plotted points fall on or near the line $y = x$.
- If the theoretical and data distributions differ only in their location or scale, the points on the plot fall on or near the line $y = ax + b$. The slope a and intercept b are visual estimates of the scale and location parameters of the theoretical distribution.

Q-Q plots are more convenient than probability plots for graphical estimation of the location and scale parameters because the x -axis of a Q-Q plot is scaled linearly. On the other hand, probability plots are more convenient for estimating percentiles or probabilities. There are many reasons why the point pattern in a Q-Q plot may not be linear. Chambers et al. (1983) and Fowlkes (1987) discuss the interpretations of

commonly encountered departures from linearity, and these are summarized in the following table.

Table 48.6 Quantile-Quantile Plot Diagnostics

Description of Point Pattern	Possible Interpretation
All but a few points fall on a line	Outliers in the data
Left end of pattern is below the line; right end of pattern is above the line	Long tails at both ends of the data distribution
Left end of pattern is above the line; right end of pattern is below the line	Short tails at both ends of the distribution
Curved pattern with slope increasing from left to right	Data distribution is skewed to the right
Curved pattern with slope decreasing from left to right	Data distribution is skewed to the left
Staircase pattern (plateaus and gaps)	Data have been rounded or are discrete

In some applications, a nonlinear pattern may be more revealing than a linear pattern. However as noted by Chambers et al. (1983), departures from linearity can also be due to chance variation.

Determining Computer Resources

Because PROC UNIVARIATE computes quantile statistics, it requires additional memory to store a copy of the data in memory. By default, the report procedures PROC MEANS, PROC SUMMARY, and PROC TABULATE require less memory because they do not automatically compute quantiles. These procedures also provide an option to use a new fixed-memory quantiles estimation method that is usually less memory intense. For more information, see “Quantiles” on page 680.

The only factor that limits the number of variables that you can analyze is the computer resources that are available. The amount of temporary storage and CPU time that PROC UNIVARIATE requires depends on the statements and the options that you specify. To calculate the computer resources the procedure needs, let

- N be the number of observations in the data set
- V be the number of variables in the VAR statement
- U_i be the number of unique values for the i th variable.

Then the minimum memory requirement in bytes to process all variables is

$$M = 24 \sum U_i$$

If M bytes are not available, PROC UNIVARIATE must process the data multiple times to compute all the statistics. This reduces the minimum memory requirement to

$$M = 24 \max (U_i)$$

ROUND= reduces the number of unique values (U_i), thereby reducing memory requirements. ROBUSTSCALE requires $40U_i$ bytes of temporary storage.

Several factors affect the CPU time requirement:

- The time to create V tree structures to internally store the observations is proportional to $NV \log(N)$.
- The time to compute moments and quantiles for the i th variable is proportional to U_i .
- The time to compute the NORMAL option test statistics is proportional to N .
- The time to compute the ROBUSTSCALE option test statistics is proportional to $U_i \log(U_i)$.
- The time to compute the exact significance level of the sign rank statistic may increase when the number of nonzero values is less than or equal to 20.

Each of these factors has a different constant of proportionality. For additional information on how to optimize CPU performance and memory usage, see the SAS documentation for your operating environment.

Statistical Computations: UNIVARIATE Procedure

PROC UNIVARIATE uses standard algorithms to compute the moment statistics (such as the mean, variance, skewness, and kurtosis). See Appendix 1, “SAS Elementary Statistics Procedures,” on page 1577 for the statistical formulas. The computational details for confidence limits, hypothesis test statistics, and quantile statistics follow.

Confidence Limits for Parameters of the Normal Distribution

The two-sided $100(1 - \alpha)$ percent confidence interval for the mean has upper and lower limits

$$\bar{x} \pm t_{(1-\alpha/2;n-1)} \frac{s}{\sqrt{n}}$$

where s is $\sqrt{\frac{1}{n-1} \sum (x_i - \bar{x})^2}$ and $t_{(1-\alpha/2;n-1)}$ is the $(1 - \alpha/2)$ percentile of the t distribution with $n - 1$ degrees of freedom.

The one-sided $100(1 - \alpha)$ percent confidence limit is computed as

$$\begin{aligned} \bar{x} + t_{(1-\alpha;n-1)} \frac{s}{\sqrt{n}} & \quad (\text{upper}) \\ \bar{x} - t_{(1-\alpha;n-1)} \frac{s}{\sqrt{n}} & \quad (\text{lower}) \end{aligned}$$

The two-sided $100(1 - \alpha)$ percent confidence interval for the standard deviation has lower and upper limits

$$s \sqrt{\frac{n-1}{\chi^2_{(1-\alpha/2; n-1)}}}, s \sqrt{\frac{n-1}{\chi^2_{(\alpha/2; n-1)}}}$$

where $\chi^2_{(1-\alpha/2; n-1)}$ and $\chi^2_{(\alpha/2; n-1)}$ are the $(1 - \alpha/2)$ and $\alpha/2$ percentiles of the chi-square distribution with $n - 1$ degrees of freedom. A one-sided $100(1 - \alpha)$ percent confidence limit is computed by replacing $\alpha/2$ with α .

A $100(1 - \alpha)$ percent confidence interval for the variance has upper and lower limits equal to the squares of the corresponding upper and lower limits for the standard deviation.

When you use the WEIGHT statement and specify VARDEF=DF in the PROC statement, the $100(1 - \alpha)$ percent confidence interval for the weighted mean is

$$\bar{x}_w \pm t_{(1-\alpha/2)} \frac{s_w}{\sqrt{\sum_{i=1}^n w_i}}$$

where \bar{x}_w is the weighted mean, s_w is the weighted standard deviation, w_i is the weight for i th observation, and $t_{(1-\alpha/2)}$ is the $(1 - \alpha/2)$ critical percentage for the t distribution with $n - 1$ degrees of freedom.

Tests for Location

PROC UNIVARIATE computes tests for location that include Student's t test, the sign test, and the Wilcoxon signed rank test. All three tests produce a test statistic for the null hypothesis that the mean or median is equal to a given value μ_0 against the two-sided alternative that the mean or median is not equal to μ_0 . By default, PROC UNIVARIATE sets the value of μ_0 to zero. Use the MU0= option in the PROC UNIVARIATE statement to test that the mean or median is equal to another value.

The Student's t test is appropriate when the data are from an approximately normal population; otherwise, use nonparametric tests such as the sign test or the signed rank test. For large sample situations, the t test is asymptotically equivalent to a z test.

If you use the WEIGHT statement, PROC UNIVARIATE computes only one weighted test for location, the t test. You must use the default value for the VARDEF= option in the PROC statement.

You can also compare means or medians of *paired data*. Data are said to be paired when subjects or units are matched in pairs according to one or more variables, such as pairs of subjects with the same age and gender. Paired data also occur when each subject or unit is measured at two times or under two conditions. To compare the means or medians of the two times, create an analysis variable that is the difference between the two measures. The test that the mean or the median difference of the variables equals zero is equivalent to the test that the means or medians of the two original variables are equal. See Example 4 on page 1552.

Student's t Test

PROC UNIVARIATE calculates the t statistic as

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

where \bar{x} is the sample mean, n is the number of nonmissing values for a variable, and s is the sample standard deviation. Under the null hypothesis, the population mean equals μ_0 . When the data values are approximately normally distributed, the probability under the null hypothesis of a t statistic that is as extreme, or more extreme, than the observed value (the p -value) is obtained from the t distribution with $n - 1$ degrees of freedom. For large n , the t statistic is asymptotically equivalent to a z test.

When you use the WEIGHT statement and the default value of VARDEF=, which is DF, the t statistic is calculated as

$$t_w = \frac{\bar{x}_w - \mu_0}{s_w / \sqrt{\sum_{i=1}^n w_i}}$$

where \bar{x}_w is the weighted mean, s_w is the weighted standard deviation, and w_i is the weight for i th observation. The t_w statistic is treated as having a Student's t distribution with $n - 1$ degrees of freedom. If you specify the EXCLNPWGT option in the PROC statement, n is the number of nonmissing observations when the value of the WEIGHT variable is positive. By default, n is the number of nonmissing observations for the WEIGHT variable.

Sign Test

PROC UNIVARIATE calculates the sign test statistic as

$$M = (n^+ - n^-) / 2$$

where n^+ is the number of values that is greater than u_0 and n^- is the number of values that is less than u_0 . Values equal to u_0 are discarded.

Under the null hypothesis that the population median is equal to u_0 , the p -value for the observed statistic M is

$$\text{Prob} \{ |M| \geq |M| \} = 0.5^{(n_t-1)} \sum_{j=0}^{\min(n^+, n^-)} \binom{n_t}{j}$$

where $n_t = n^+ + n^-$ is the number of x_i values not equal to u_0 .

Wilcoxon Signed Rank Test

PROC UNIVARIATE calculates the Wilcoxon signed rank test statistic as

$$S = \sum r_i^+ - n_t(n_t + 1) / 4$$

where r_i^+ is the rank of $|x_i - \mu_0|$ after discarding values of x_i equal to u_0 , n_t is the number of x_i values not equal to u_0 , and the sum is calculated for values of $x_i - \mu_0$ greater than 0. Average ranks are used for tied values.

The p -value is the probability of obtaining a signed rank statistic greater in absolute value than the absolute value of the observed statistic S . If $n_t \leq 20$, the significance level of S is computed from the exact distribution of S , which can be enumerated under the null hypothesis that the distribution is symmetric about u_0 . When $n_t > 20$, the significance of level S is computed by treating

$$\frac{S\sqrt{n_t - 1}}{\sqrt{n_t V - S^2}}$$

as a Student's t variate with $n_t - 1$ degrees of freedom. V is computed as

$$V = \frac{1}{24} n_t (n_t + 1) (2n_t + 1) - 0.5 \sum t_i (t_i + 1) (t_i - 1)$$

where the sum is calculated over groups that are tied in absolute value, and t_i is the number of tied values in the i th group (Iman 1974; Conover 1998).

The Wilcoxon signed rank test assumes that the distribution is symmetric. If the assumption is not valid, you can use the sign test to test that the median is u_0 . See Lehman (1998) for more details.

Goodness-of-Fit Tests

When you specify the NORMAL option in the PROC UNIVARIATE statement or you request a fitted parametric distribution in the HISTOGRAM statement, the procedure computes test statistics for the null hypothesis that the values of the analysis variable are a random sample from the specified theoretical distribution. When you specify the normal distribution, the test statistics depend on the sample size. If the sample size is less than or equal to 2000, PROC UNIVARIATE calculates the Shapiro-Wilk W statistic. For a specified distribution, the procedure attempts to calculate three goodness-of-fit tests that are based on the empirical distribution function (EDF): the Kolmogorov-Smirnov D statistic, the Anderson-Darling statistic, and the Cramer-von Mises statistic. However, some of the EDF tests are currently not supported when the parameters of a specified distribution are estimated. See Table 48.7 on page 1524 for more information.

You determine whether to reject the null hypothesis by examining the probability that is associated with a test statistic. When the p -value is less than the predetermined critical value (alpha value), you reject the null hypothesis and conclude that the data did not come from the theoretical distribution.

If you want to test the normality assumptions that underlie analysis of variance methods, beware of using a statistical test for normality alone. A test's ability to reject the null hypothesis (known as the *power* of the test) increases with the sample size. As the sample size becomes larger, increasingly smaller departures from normality can be detected. Since small deviations from normality do not severely affect the validity of analysis of variance tests, it is important to examine other statistics and plots to make a final assessment of normality. The skewness and kurtosis measures and the plots that are provided by the PLOTS option, the HISTOGRAM statement, PROBLOT statement, and QQPLOT statement can be very helpful. For small sample sizes, power is low for detecting larger departures from normality that may be important. To increase the test's ability to detect such deviations, you may want to declare significance at higher levels, such as 0.15 or 0.20, rather than the often-used 0.05 level.

Again, consulting plots and additional statistics will help you assess the severity of the deviations from normality.

Shapiro-Wilk Statistic

If the sample size is less than or equal to 2000 and you specify the NORMAL option, PROC UNIVARIATE computes the Shapiro-Wilk statistic, W . The W statistic is the ratio of the best estimator of the variance (based on the square of a linear combination of the order statistics) to the usual corrected sum of squares estimator of the variance (Shapiro, 1965). W must be greater than zero and less than or equal to one. Small values of W lead to the rejection of the null hypothesis of normality. The distribution of W is highly skewed. Seemingly large values of W (such as 0.90) may be considered small and lead you to reject the null hypothesis. When the sample size is greater than three, the coefficients to compute the linear combination of the order statistics are approximated by the method of Royston (1992).

$$Z_n = (-\log(\gamma - \log(1 - W_n)) - \mu) / \sigma$$

when $4 \leq n \leq 11$ and

$$Z_n = (\log(1 - W_n) - \mu) / \sigma$$

when $12 \leq n \leq 2000$, where γ , μ , and σ are functions of n , obtained from simulation results, and Z_n is a standard normal variate. Large values of Z_n indicate departure from normality.

EDF Goodness-of-Fit Tests

When you fit a parametric distribution, PROC UNIVARIATE provides a series of goodness-of-fit tests that are based on the empirical distribution function (EDF). The empirical distribution function is defined for a set of n independent observations X_1, \dots, X_n with a common distribution function $F(x)$. The observations that are ordered from smallest to largest as $X_{(1)}, \dots, X_{(n)}$. The empirical distribution function, $F_n(x)$, is defined as

$$\begin{aligned} F_n(x) &= 0, & x < X_{(1)} \\ F_n(x) &= \frac{i}{n}, & X_{(i)} \leq x < X_{(i+1)} \quad i = 1, \dots, n-1 \\ F_n(x) &= 1, & X_{(n)} \leq x \end{aligned}$$

Note that $F_n(x)$ is a step function that takes a step of height $\frac{1}{n}$ at each observation. This function estimates the distribution function $F(x)$. At any value x , $F_n(x)$ is the proportion of observations that is less than or equal to x while $F(x)$ is the theoretical probability of an observation that is less than or equal to x . EDF statistics measure the discrepancy between $F_n(x)$ and $F(x)$.

The computational formulas for the EDF statistics use the probability integral transformation $U = F(X)$. If $F(X)$ is the distribution function of X , the random variable U is uniformly distributed between 0 and 1.

Given n observations $X_{(1)}, \dots, X_{(n)}$, PROC UNIVARIATE computes the values $U_{(i)} = F(X_{(i)})$ by applying the transformation, as follows.

When you specify the NORMAL option in the PROC UNIVARIATE statement or use the HISTOGRAM statement to fit a parametric distribution, PROC UNIVARIATE provides a series of goodness-of-fit tests that are based on the empirical distribution function (EDF):

- ☐ Kolmogorov-Smirnov
- ☐ Anderson-Darling
- ☐ Cramer-von Mises

These tests are based on various measures of the discrepancy between the empirical distribution function $F_n(x)$ and the proposed cumulative distribution function $F(x)$.

Once the EDF test statistics are computed, the associated p -values are calculated. PROC UNIVARIATE uses internal tables of probability levels that are similar to those given by D'Agostino and Stephens (1986). If the value lies between two probability levels, then linear interpolation is used to estimate the probability value.

Note: PROC UNIVARIATE does not support some of the EDF tests when you use the HISTOGRAM statement and you estimate the parameters of the specified distribution. See Table 48.7 on page 1524 for more information. Δ

Kolmogorov D Statistic

The Kolmogorov-Smirnov statistic (D) is defined as

$$D = \sup_x |F_n(x) - F(x)|$$

The Kolmogorov-Smirnov statistic belongs to the supremum class of EDF statistics. This class of statistics is based on the largest vertical difference between $F(x)$ and $F_n(x)$.

The Kolmogorov-Smirnov statistic is computed as the maximum of D^+ and D^- . D^+ is the largest vertical distance between the EDF and the distribution function when the EDF is greater than the distribution function. D^- is the largest vertical distance when the EDF is less than the distribution function.

$$\begin{aligned} D^+ &= \max_i \left(\frac{i}{n} - U_{(i)} \right) \\ D^- &= \max_i \left(U_{(i)} - \frac{i-1}{n} \right) \\ D &= \max(D^+, D^-) \end{aligned}$$

PROC UNIVARIATE uses a modified Kolmogorov D statistic to test the data against a normal distribution with mean and variance equal to the sample mean and variance.

Anderson-Darling Statistic

The Anderson-Darling statistic and the Cramer-von Mises statistic belong to the quadratic class of EDF statistics. This class of statistics is based on the squared difference $(F_n(x) - F(x))^2$. Quadratic statistics have the following general form:

$$Q = n \int_{-\infty}^{+\infty} (F_n(x) - F(x))^2 \psi(x) dF(x)$$

The function $\psi(x)$ weights the squared difference $(F_n(x) - F(x))^2$.

The Anderson-Darling statistic (A^2) is defined as

$$A^2 = n \int_{-\infty}^{+\infty} (F_n(x) - F(x))^2 [F(x)(1 - F(x))]^{-1} dF(x)$$

where the weight function is $\psi(x) = [F(x)(1 - F(x))]^{-1}$.

The Anderson-Darling statistic is computed as

$$A^2 = -n - \frac{1}{n} \sum_{i=1}^n [(2i-1)(\log U_{(i)} + \log(1 - U_{(n+1-i)}))]$$

Cramer-von Mises Statistic

The Cramer-von Mises statistic (W^2) is defined as

$$W^2 = n \int_{-\infty}^{+\infty} (F_n(x) - F(x))^2 dF(x)$$

where the weight function is $\psi(x) = 1$.

The Cramer-von Mises statistic is computed as

$$W^2 = \sum_{i=1}^n \left(U_{(i)} - \frac{2i-1}{2n} \right)^2 + \frac{1}{12n}$$

Probability Values of EDF Tests

Once the EDF test statistics are computed, PROC UNIVARIATE computes the associated probability values.

The probability value depends upon the parameters that are known and the parameters that PROC UNIVARIATE estimates for the fitted distribution. Table 48.7 on page 1524 summarizes different combinations of estimated parameters for which EDF tests are available.

Note: PROC UNIVARIATE assumes that the threshold (THETA=) parameter for the beta, exponential, gamma, lognormal, and Weibull distributions is known. If you omit its value, PROC UNIVARIATE assumes that it is zero and that it is known. Likewise, PROC UNIVARIATE assumes that the SIGMA= parameter, which determines the

upper threshold (SIGMA) for the beta distribution, is known. If you omit its value, PROC UNIVARIATE assumes that the value is one. These parameters are not listed in Table 48.7 on page 1524 because they are assumed to be known in all cases, and they do not affect which EDF statistics PROC UNIVARIATE computes. \triangle

Table 48.7 Availability of EDF Tests

Distribution	Parameters			Tests Available
	Threshold	Scale	Shape	
Beta	θ known	σ known	α, β known	all
	θ known	σ known	$\alpha, \beta < 5$ unknown	all
Exponential	θ known	σ known		all
	θ known	σ unknown		all
	θ unknown	σ known		all
	θ unknown	σ unknown		all
Gamma	θ known	σ known	α known	all
	θ known	σ unknown	α known	all
	θ known	σ known	α unknown	all
	θ known	σ unknown	α unknown	all
	θ unknown	σ known	$\alpha > 1$ known	all
	θ unknown	σ unknown	$\alpha > 1$ known	all
	θ unknown	σ known	$\alpha > 1$ unknown	all
	θ unknown	σ unknown	$\alpha > 1$ unknown	all
Lognormal	θ known	ζ known	σ known	all
	θ known	ζ known	σ unknown	A^2 and W^2
	θ known	ζ unknown	σ known	A^2 and W^2
	θ known	ζ unknown	σ unknown	all
	θ unknown	ζ known	$\sigma < 3$ known	all
	θ unknown	ζ known	$\sigma < 3$ unknown	all
	θ unknown	ζ unknown	$\sigma < 3$ known	all
	θ unknown	ζ unknown	$\sigma < 3$ unknown	all

Distribution	Parameters			Tests Available
	Threshold	Scale	Shape	
Normal	θ known	σ known		all
	θ known	σ unknown		A^2 and W^2
	θ unknown	σ known		A^2 and W^2
	θ unknown	σ unknown		all
Weibull	θ known	σ known	c known	all
	θ known	σ unknown	c known	A^2 and W^2
	θ known	σ known	c unknown	A^2 and W^2
	θ known	σ unknown	c unknown	A^2 and W^2
	θ unknown	σ known	$c > 2$ known	all
	θ unknown	σ unknown	$c > 2$ known	all
	θ unknown	σ known	$c > 2$ unknown	all
	θ unknown	σ unknown	$c > 2$ unknown	all

Robust Estimators

A statistical method is robust if the method is insensitive to slight departures from the assumptions that justify the method. PROC UNIVARIATE provides several methods for robust estimation of location and scale.

Winsorized Means

When outliers are present in the data, the Winsorized mean is a robust estimator of the location that is relatively insensitive to the outlying values. The k -times Winsorized mean is calculated as

$$\bar{x}_{wk} = \frac{1}{n} \left((k+1) x_{(k+1)} + \sum_{i=k+2}^{n-k-1} x_{(i)} + (k+1) x_{(n-k)} \right)$$

The Winsorized mean is computed after the k smallest observations are replaced by the $(k+1)$ smallest observation, and the k largest observations are replaced by the $(k+1)$ largest observation.

For a symmetric distribution, the symmetrically Winsorized mean is an unbiased estimate of the population mean. But the Winsorized mean does not have a normal distribution even if the data are from a normal population.

The Winsorized sum of squared deviations is defined as

$$s_{wk}^2 = (k+1) (x_{(k+1)} - \bar{x}_{wk})^2 + \sum_{i=k+2}^{n-k-1} (x_{(i)} - \bar{x}_{wk})^2 + (k+1) (x_{(n-k)} - \bar{x}_{wk})^2$$

A Winsorized t test is given by

$$t_{wk} = \frac{(\bar{x}_{wk} - \mu_0)}{STDERR(\bar{x}_{wk})}$$

where the standard error of the Winsorized mean is

$$STDERR(\bar{x}_{wk}) = \frac{n-1}{n-2k-1} \frac{s_{wk}}{\sqrt{n(n-1)}}$$

When the data are from a symmetric distribution, the distribution of the Winsorized t statistic t_{wk} is approximated by a Student's t distribution with $n-2k-1$ degrees of freedom (Tukey and McLaughlin 1963, Dixon and Tukey 1968).

A $100(1-\alpha)$ percent confidence interval for the Winsorized mean has upper and lower limits

$$\bar{x}_{wk} \pm t_{(1-\alpha/2)} STDERR(\bar{x}_{wk})$$

and the $(1-\alpha/2)$ critical value of the Student's t statistics has $n-2k-1$ degrees of freedom.

Trimmed Means

When outliers are present in the data, the trimmed mean is a robust estimator of the location that is relatively insensitive to the outlying values. The k -times trimmed mean is calculated as

$$\bar{x}_{tk} = \frac{1}{n-2k} \sum_{i=k+1}^{n-k} x_{(i)}$$

The trimmed mean is computed after the k smallest and k largest observations are deleted from the sample. In other words, the observations are trimmed at each end.

For a symmetric distribution, the symmetrically trimmed mean is an unbiased estimate of the population mean. But the trimmed mean does not have a normal distribution even if the data are from a normal population.

A robust estimate of the variance of the trimmed mean t_{tk} can be based on the Winsorized sum of squared deviations (Tukey and McLaughlin 1963). The resulting trimmed t test is given by

$$t_{tk} = \frac{(\bar{x}_{tk} - \mu_0)}{STDERR(\bar{x}_{tk})}$$

where the standard error of the trimmed mean is

$$STDERR(\bar{x}_{tk}) = \frac{s_{wk}}{\sqrt{(n-2k)(n-2k-1)}}$$

and s_{wk} is the square root of the Winsorized sum of squared deviations

When the data are from a symmetric distribution, the distribution of the trimmed t statistic t_{tk} is approximated by a Student's t distribution with $n - 2k - 1$ degrees of freedom (Tukey and McLaughlin 1963, Dixon and Tukey 1968).

A $100(1 - \alpha)$ percent confidence interval for the trimmed mean has upper and lower limits

$$\bar{x}_{tk} \pm t_{(1-\alpha/2)} STDERR(\bar{x}_{tk})$$

and the $(1 - \alpha/2)$ critical value of the Student's t statistics has $n - 2k - 1$ degrees of freedom.

Robust Measures of Scale

The sample standard deviation is a commonly used estimator of the population scale. However, it is sensitive to outliers and may not remain bounded when a single data point is replaced by an arbitrary number. With robust scale estimators, the estimates remain bounded even when a portion of the data points are replaced by arbitrary numbers.

PROC UNIVARIATE computes robust measures of scale that include statistics of interquartile range, Gini's mean difference G , MAD , Q_n , and S_n , with their corresponding estimates of σ .

The interquartile range is a simple robust scale estimator, which is the difference between the upper and lower quartiles. For a normal population, the standard deviation σ can be estimated by dividing the interquartile range by 1.34898.

Gini's mean difference is also a robust estimator of the standard deviation σ . For a normal population, Gini's mean difference has expected value $2\sigma/\sqrt{\pi}$. Thus, multiplying Gini's mean difference by $\sqrt{\pi}/2$ yields a robust estimator of the standard deviation when the data are from a normal sample. The constructed estimator has high efficiency for the normal distribution relative to the usual sample standard deviation. It is also less sensitive to the presence of outliers than the sample standard deviation.

Gini's mean difference is computed as

$$G = \frac{1}{\binom{n}{2}} \sum_{i < j} |x_i - x_j|$$

If the observations are from a normal distribution, then $\sqrt{\pi} G/2$ is an unbiased estimator of the standard deviation σ .

A very robust scale estimator is the MAD , the median absolute deviation about the median (Hampel, 1974.)

$$MAD = \text{med}_i (|x_i - \text{med}_j(x_j)|)$$

where the inner median, $\text{med}_j(x_j)$, is the median of the n observations and the outer median, med_i , is the median of the n absolute values of the deviations about the median.

For a normal distribution, $1.4826 \cdot MAD$ can be used to estimate the standard deviation σ .

The *MAD* statistic has low efficiency for normal distributions, and it may not be appropriate for symmetric distributions. Rousseeuw and Croux (1993) proposed two new statistics as alternatives to the *MAD* statistic.

The first statistic is

$$S_n = 1.1926 \operatorname{med}_i(\operatorname{med}_j(|x_i - x_j|))$$

where the outer median, med_i , is the median of the n medians of $(|x_i - x_j|); j = 1, 2, \dots, n$.

To reduce the small-sample bias, $c_{sn} S_n$ is used to estimate the standard deviation σ , where c_{sn} is a the correction factor (Croux and Rousseeuw, 1992.)

The second statistic is

$$Q_n = 2.219 \{ |x_i - x_j|; i < j \}_{(k)}$$

where $k = \binom{h}{2}$, $h = [n/2] + 1$, and $[n/2]$ is the integer part of $n/2$. That is, Q_n is 2.2219 times the k th order statistic of the $\binom{n}{2}$ distances between data points.

The bias-corrected statistic, $c_{qn} Q_n$, is used to estimate the standard deviation σ , where c_{qn} is a correction factor.

Calculating Percentiles

The UNIVARIATE procedure automatically computes the minimum, 1st, 5th, 10th, 25th, 50th, 75th, 90th, 95th, 99th, and maximum percentiles. You use the PCTLDEF= option in the PROC UNIVARIATE statement to specify one of five methods to compute quantile statistics. See “Percentile and Related Statistics” on page 1583 for more information.

To compute the quantile that each observation falls in, use PROC RANK with the GROUP= option. To calculate percentiles other than the default percentiles, use PCTLPTS= and PCTLPRE= in the OUTPUT statement.

Confidence Limits for Quantiles

The CIPCTLDF option and CIPCTLNORMAL option compute confidence limits for quantiles using methods described in Hahn and Meeker (1991).

When $0.0 < p < 0.5$, the two-sided $100(1 - \alpha)$ percent confidence interval for quantiles that are based on normal data has lower and upper limits

$$\bar{x} - g'_{(\alpha/2; 1-p, n)} s, \bar{x} - g'_{(1-\alpha/2; 1-p, n)} s$$

where p is the percentile $100 \times p$.

When $0.5 \leq p < 1.0$, the lower and upper limits are

$$\bar{x} + g'_{(\alpha/2; p, n)} s, \bar{x} + g'_{(1-\alpha/2; p, n)} s$$

A one-sided $100(1 - \alpha)$ percent confidence limit is computed by replacing $\alpha/2$ with α . The factor $g'_{(\gamma, p, n)}$ is described in Owen and Hua (1977) and Odeh and Owen (1980).

The two-sided distribution-free $100(1 - \alpha)\%$ confidence interval for quantiles from a sample of size n is

$$x_{(l)}, x_{(u)}$$

where $x_{(j)}$ is j th order statistic. The lower rank l and upper rank u are integers that are symmetric or nearly symmetric around $i = [np] + 1$, where $[np]$ is the integral part of np .

The l and u are chosen so that the order statistics $x_{(l)}$ and $x_{(u)}$

- ☐ are approximately symmetric about $x_{((n+1)p)}$
- ☐ are as close to $x_{((n+1)p)}$ as possible
- ☐ satisfy the coverage probability requirement.

$$Q_b(u - 1; n, p) - Q_b(l - 1; n, p) \geq 1 - \alpha$$

where Q_b is the cumulative binomial probability, $0 < l < u \leq n$, and $0 < p < 1$.

The coverage probability is sometimes less than $1 - \alpha$. This can occur in the tails of the distribution when the sample size is small. To avoid this problem, you can specify the option TYPE=ASYMMETRIC, which causes PROC UNIVARIATE to use asymmetric values of l and u . However, PROC UNIVARIATE first attempts to compute confidence limits that satisfy all three conditions. If the last condition is not satisfied, then the first condition is relaxed. Thus, some of the confidence limits may be symmetric while others, especially in the extremes, are not.

A one-sided distribution-free lower $100(1 - \alpha)$ percent confidence limit is computed as $x_{(l)}$ when l is the largest integer that satisfies the inequality

$$1 - Q_b(l - 1; n, p) \geq 1 - \alpha$$

where $0 < l \leq n$, and $0 < p < 1$. Likewise, a one-sided distribution-free upper $100(1 - \alpha)\%$ confidence limit is computed as $x_{(u)}$ when u is the smallest integer that satisfies the inequality

$$Q_b(u - 1; n, p) \geq 1 - \alpha$$

where $0 < u \leq n$, and $0 < p < 1$.

Weighted Quantiles

When you use the WEIGHT statement the percentiles are computed as follows. Let x_i be the i th ordered nonmissing value, $x_1 \leq x_2 \leq \dots \leq x_n$. Then, for a given value of p between 0 and 1, the p th weighted quantile (or 100 p th weighted percentile), y , is computed from the empirical distribution function with averaging

$$y = \begin{cases} \frac{1}{2}(x_i + x_{i+1}) & \text{if } \sum_{j=1}^i w_j = pW \\ x_{i+1} & \text{if } \sum_{j=1}^i w_j < pW < \sum_{j=1}^{i+1} w_j \end{cases}$$

where w_j is the weight associated with x_i , $W = \sum_{i=1}^n w_i$ is the sum of the weights and w_i is the weight for i th observation.

When the observations have identical weights, the weighted percentiles are the same as the unweighted percentiles with PCTLDEF=5.

Calculating the Mode

The mode is the value that occurs most often in the data. PROC UNIVARIATE counts repetitions of the actual values or, if you specify the ROUND= option, the rounded values. If a tie occurs for the most frequent value, the procedure reports the lowest value. To list all possible modes, use the MODES option in the PROC UNIVARIATE statement. When no repetitions occur in the data (as with truly continuous data), the procedure does not report the mode.

The WEIGHT statement has no effect on the mode.

Formulas for Fitted Continuous Distributions

The following sections provide information about the families of parametric distributions that you can fit with the HISTOGRAM statement. Properties of the parametric curves are discussed by Johnson, et al. (1994).

Beta Distribution

The fitted density function is

$$p(x) = \begin{cases} \frac{(x-\theta)^{\alpha-1}(\sigma+\theta-x)^{\beta-1}}{B(\alpha,\beta)\sigma^{(\alpha+\beta-1)}}h \times 100\% & \text{for } \theta < x < \theta + \sigma \\ 0 & \text{for } x \leq \theta \text{ or } x \geq \theta + \sigma \end{cases}$$

where $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ and

- θ = lower threshold parameter (lower endpoint parameter)
- σ = scale parameter ($\sigma > 0$)
- α = shape parameter ($\alpha > 0$)
- β = shape parameter ($\beta > 0$)
- h = width of histogram interval

This notation is consistent with that of other distributions that you can fit with the HISTOGRAM statement. However, many texts, including Johnson, et al. (1994), write the beta density function as:

$$p(x) = \begin{cases} \frac{(x-a)^{p-1}(b-x)^{q-1}}{B(p,q)(b-a)^{p+q-1}} & \text{for } a < x < b \\ 0 & \text{for } x \leq a \text{ or } x \geq b \end{cases}$$

The two notations are related as follows:

$$\begin{aligned}\sigma &= b - a \\ \theta &= a \\ \alpha &= p \\ \beta &= q\end{aligned}$$

The range of the beta distribution is bounded below by a threshold parameter $\theta = a$ and above by $\theta + \sigma = b$. If you specify a fitted beta curve using the BETA option, θ must be less than the minimum data value, and $\theta + \sigma$ must be greater than the maximum data value. You can specify θ and σ with the THETA= and SIGMA= *value* in parentheses after the keyword BETA. By default, $\sigma = 1$ and $\theta = 0$. If you specify THETA=EST and SIGMA=EST, maximum likelihood estimates are computed for θ and σ .

Note: However, three- and four-parameter maximum likelihood estimation may not always converge. Δ

In addition, you can specify α and β with the ALPHA= and BETA= *beta-options*, respectively. By default, the procedure calculates maximum likelihood estimates for α and β . For example, to fit a beta density curve to a set of data bounded below by 32 and above by 212 with maximum likelihood estimates for α and β , use the following statement:

```
histogram length / beta(theta=32 sigma=180);
```

The beta distributions are also referred to as Pearson Type I or II distributions. These include the *power-function* distribution ($\beta = 1$), the *arc-sine* distribution ($\alpha = \beta = \frac{1}{2}$), and the generalized *arc-sine* distributions ($\alpha + \beta = 1, \beta \neq \frac{1}{2}$). You can use the DATA step function BETAINV to compute beta quantiles and the DATA step function PROBBETA to compute beta probabilities.

Exponential Distribution

The fitted density function is

$$p(x) = \begin{cases} \frac{h \times 100\%}{\sigma} \exp\left(-\left(\frac{x-\theta}{\sigma}\right)\right) & \text{for } x \geq \theta \\ 0 & \text{for } x < \theta \end{cases}$$

where

$$\begin{aligned}\theta &= \text{threshold parameter} \\ \sigma &= \text{scale parameter } (\sigma > 0) \\ h &= \text{width of histogram interval}\end{aligned}$$

The threshold parameter θ must be less than or equal to the minimum data value. You can specify θ with the THRESHOLD= *exponential-option*. By default, $\theta = 0$. If you specify THETA=EST, a maximum likelihood estimate is computed for θ . In addition, you can specify σ with the SCALE= *exponential-option*. By default, the procedure calculates a maximum likelihood estimate for σ . Note that some authors define the scale parameter as $\frac{1}{\sigma}$.

The exponential distribution is a special case of both the gamma distribution (with $\alpha = 1$ and the Weibull distribution (with $c = 1$). A related distribution is the *extreme value* distribution. If $Y = \exp(-X)$ has an exponential distribution, then X has an extreme value distribution.

Gamma Distribution

The fitted density function is

$$p(x) = \begin{cases} \frac{h \times 100\%}{\Gamma(\alpha)\sigma} \left(\frac{x-\theta}{\sigma}\right)^{\alpha-1} \exp\left(-\left(\frac{x-\theta}{\sigma}\right)\right) & \text{for } x > \theta \\ 0 & \text{for } x \leq \theta \end{cases}$$

where

θ = threshold parameter

σ = scale parameter ($\sigma > 0$)

α = shape parameter ($\alpha > 0$)

h = width of histogram interval

The threshold parameter θ must be less than the minimum data value. You can specify θ with the THRESHOLD= *gamma-option*. By default, $\theta = 0$. If you specify THETA=EST, a maximum likelihood estimate is computed for θ . In addition, you can specify σ and α with the SCALE= and ALPHA= *gamma-options*. By default, the procedure calculates maximum likelihood estimates for σ and α .

The gamma distributions are also referred to as Pearson Type III distributions, and they include the chi-square, exponential, and Erlang distributions. The probability density function for the chi-square distribution is

$$p(x) = \begin{cases} \frac{1}{2\Gamma(\frac{\nu}{2})} \left(\frac{x}{2}\right)^{\frac{\nu}{2}-1} \exp\left(-\frac{x}{2}\right) & \text{for } x > 0 \\ 0 & \text{for } x \leq 0 \end{cases}$$

Notice that this is a gamma distribution with $\alpha = \frac{\nu}{2}$, and $\theta = 0$. The exponential distribution is a gamma distribution with $\alpha = 1$, and the Erlang distribution is a gamma distribution with α being a positive integer. A related distribution is the Rayleigh distribution. If $R = \frac{\max(X_1, \dots, X_n)}{\min(X_1, \dots, X_n)}$ where the X_i 's are independent χ_ν^2 variables, then $\log R$ is distributed with a χ_ν distribution having a probability density function of

$$p(x) = \begin{cases} \left[2^{\frac{\nu}{2}-1} \Gamma\left(\frac{\nu}{2}\right)\right]^{-1} x^{\nu-1} \exp\left(-\frac{x^2}{2}\right) & \text{for } x > 0 \\ 0 & \text{for } x \leq 0 \end{cases}$$

If $\nu = 2$, the preceding distribution is referred to as the Rayleigh distribution. You can use the DATA step function GAMINV to compute gamma quantiles and the DATA step function PROBGAM to compute gamma probabilities.

Lognormal Distribution

The fitted density function is

$$p(x) = \begin{cases} \frac{h \times 100\%}{\sigma \sqrt{2\pi}(x-\theta)} \exp\left(-\frac{(\log(x-\theta)-\zeta)^2}{2\sigma^2}\right) & \text{for } x > \theta \\ 0 & \text{for } x \leq \theta \end{cases}$$

where

θ = threshold parameter
 ζ = scale parameter ($-\infty < \zeta < \infty$)
 σ = shape parameter ($\sigma > 0$)
 h = width of histogram interval

The threshold parameter θ must be less than the minimum data value. You can specify θ with the THRESHOLD= *lognormal-option*. By default, $\theta = 0$. If you specify THETA=EST, a maximum likelihood estimate is computed for θ . You can specify ζ and σ with the SCALE= and SHAPE= *lognormal-options*, respectively. By default, the procedure calculates maximum likelihood estimates for these parameters.

Note: σ denotes the shape parameter of the lognormal distribution, whereas σ denotes the scale parameter of the beta, exponential, gamma, normal, and Weibull distributions. The use of σ to denote the lognormal shape parameter is based on the fact that $\frac{1}{\sigma}(\log(X - \theta) - \zeta)$ has a standard normal distribution if X is lognormally distributed. Δ

Normal Distribution

The fitted density function is

$$p(x) = \frac{h \times 100\%}{\sigma \sqrt{2\pi}} \exp\left(-\frac{1}{2} \left(\frac{x - \mu}{\sigma}\right)^2\right) \quad \text{for } -\infty < x < \infty$$

where

μ = mean
 σ = standard deviation ($\sigma > 0$)
 h = width of histogram interval

You can specify μ and σ with the MU= and SIGMA= *normal-options*, respectively. By default, the procedure estimates μ with the sample mean and σ with the sample standard deviation. You can use the DATA step function PROBIT to compute normal quantiles and the DATA step function PROBNORM to compute probabilities.

Weibull Distribution

The fitted density function is

$$p(x) = \begin{cases} \frac{ch \times 100\%}{\sigma} \left(\frac{x-\theta}{\sigma}\right)^{c-1} \exp\left(-\left(\frac{x-\theta}{\sigma}\right)^c\right) & \text{for } x > \theta \\ 0 & \text{for } x \leq \theta \end{cases}$$

where

θ = threshold parameter

σ = scale parameter ($\sigma > 0$)

c = shape parameter ($\alpha > 0$)

h = width of histogram interval

The threshold parameter θ must be less than the minimum data value. You can specify θ with the THRESHOLD= *Weibull-option*. By default, $\theta = 0$. If you specify THETA=EST, a maximum likelihood estimate is computed for θ . You can specify σ and c with the SCALE= and SHAPE= *Weibull-options*, respectively. By default, the procedure calculates maximum likelihood estimates for σ and c .

The exponential distribution is a special case of the Weibull distribution where $c = 1$.

Kernel Density Estimates

You can use the KERNEL option to superimpose kernel density estimates on histograms. Smoothing the data distribution with a kernel density estimate can be more effective than using a histogram to visualize features that might be obscured by the choice of histogram bins or sampling variation. For example, a kernel density estimate can also be more effective when the data distribution is multimodal. The general form of the kernel density estimator is

$$\hat{f}_\lambda(x) = \frac{1}{n_\lambda} \sum_{i=1}^n K_0\left(\frac{x - x_i}{\lambda}\right)$$

where $K_0(\cdot)$ is a kernel function, λ is the bandwidth, n is the sample size, and x_i is the i^{th} observation.

The KERNEL option provides three kernel functions (K_0): normal, quadratic, and triangular. You can specify the function with the K=*kernel-function* in parentheses after the KERNEL option. Values for the K= option are NORMAL, QUADRATIC, and TRIANGULAR (with aliases of N, Q, and T, respectively). By default, a normal kernel is used. The formulas for the kernel functions are

$$\text{Normal} \quad K_0(t) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}t^2\right) \text{ for } -\infty < t < \infty$$

$$\text{Quadratic} \quad K_0(t) = \frac{3}{4}(1 - t^2) \text{ for } |t| \leq 1$$

Triangular $K_0(t) = 1 - |t|$ for $|t| \leq 1$

The value of λ , referred to as the bandwidth parameter, determines the degree of smoothness in the estimated density function. You specify λ indirectly by specifying a standardized bandwidth c with the `C=kernel-option`. If Q is the interquartile range, and n is the sample size, then c is related to λ by the formula

$$\lambda = cQn^{-\frac{1}{5}}$$

For a specific kernel function, the discrepancy between the density estimator $\hat{f}_\lambda(x)$ and the true density $f(x)$ is measured by the mean integrated square error (MISE):

$$\text{MISE}(\lambda) = \int_x \left\{ E\left(\hat{f}_\lambda(x)\right) - f(x) \right\}^2 dx + \int_x \text{var}\left(\hat{f}_\lambda(x)\right) dx$$

The MISE is the sum of the integrated squared bias and the variance. An approximate mean integrated square error (AMISE) is

$$\text{AMISE}(\lambda) = \frac{1}{4}\lambda^4 \left(\int_t t^2 K(t) dt \right)^2 \int_x (f''(x))^2 dx + \frac{1}{n\lambda} \int_t K(t)^2 dt$$

A bandwidth that minimizes AMISE can be derived by treating $f(x)$ as the normal density having parameters μ and σ estimated by the sample mean and standard deviation. If you do not specify a bandwidth parameter or if you specify `C=MISE`, the bandwidth that minimizes AMISE is used. The value of AMISE can be used to compare different density estimates. For each estimate, the bandwidth parameter c , the kernel function type, and the value of AMISE are reported in the SAS log.

The general kernel density estimates assume that the domain of the density to estimate can take on all values on a real line. However, sometimes the domain of a density is an interval that is bounded on one or both sides. For example, if a variable Y is a measurement of only positive values, then kernel density curve should be bounded so that it is zero for negative Y values.

PROC UNIVARIATE uses a reflection technique to create the bounded kernel density curve, as described in Silverman (1986, pp. 30-31). It adds the reflections of kernel density that are outside the boundary to the bounded kernel estimates. The general form of the bounded kernel density estimator is computed by replacing $K_0\left(\frac{(x-x_i)}{\lambda}\right)$ in the original equation with

$$\left\{ K_0\left(\frac{x-x_i}{\lambda}\right) + K_0\left(\frac{(x-x_l)+(x_i-x_l)}{\lambda}\right) + K_0\left(\frac{(x_u-x)+(x_u-x_i)}{\lambda}\right) \right\}$$

where x_l is the lower bound and x_u is the upper bound.

Without a lower bound, $x_l = \infty$ and $K_0\left(\frac{(x-x_l)+(x_i-x_l)}{\lambda}\right)$ equals zero. Similarly, without an upper bound, $x_u = \infty$ and $K_0\left(\frac{(x_u-x)+(x_u-x_i)}{\lambda}\right)$ equals zero.

When $c=MISE$ is used with a bounded kernel density, PROC UNIVARIATE uses a bandwidth that minimizes the AMISE for its corresponding unbounded kernel.

Theoretical Distributions for Quantile-Quantile and Probability Plots

You can use the PROBLOT and QQPLOT statements to request probability and Q-Q plots that are based on the theoretical distributions that are summarized in the following table:

Table 48.8 Distributions and Parameters

Distribution	Density Function $p(x)$	Range	Parameters		
			Location	Scale	Shape
Beta	$\frac{(x-\theta)^{\alpha-1}(\theta+\sigma-x)^{\beta-1}}{\beta(\alpha,\beta)\sigma^{(\alpha+\beta-1)}}$	$\theta < x < \theta + \sigma$	θ	σ	α, β
Exponential	$\frac{1}{\sigma} \exp\left(-\frac{x-\theta}{\sigma}\right)$	$x \geq \theta$	θ	σ	
Gamma	$\frac{1}{\sigma\Gamma(\alpha)} \left(\frac{x-\theta}{\sigma}\right)^{\alpha-1} \exp\left(-\frac{x-\theta}{\sigma}\right)$	$x > \theta$	θ	σ	α
Lognormal (3-parameter)	$\frac{1}{\sigma\sqrt{2\pi}(x-\theta)} \exp\left(-\frac{(\log(x-\theta)-\zeta)^2}{2\sigma^2}\right)$	$x > \theta$	θ	ζ	σ
Normal	$\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$	<i>all</i> x	μ	σ	
Weibull (3-parameter)	$\frac{c}{\sigma} \left(\frac{x-\theta}{\sigma}\right)^{c-1} \exp\left(-\left(\frac{x-\theta}{\sigma}\right)^c\right)$	$x > \theta$	θ	σ	c
Weibull2 (2-parameter)	$\frac{c}{\sigma} \left(\frac{x-\theta_0}{\sigma}\right)^{c-1} \exp\left(-\left(\frac{x-\theta_0}{\sigma}\right)^c\right)$	$x > \theta_0$	θ_0 (known)	σ	c

You can request these distributions with the BETA, EXPONENTIAL, GAMMA, LOGNORMAL, NORMAL, WEIBULL, and WEIBULL2 options, respectively. If you omit a distribution option, the PROBLOT statement creates a normal probability plot and the QQPLOT statement creates a normal Q-Q plot.

The following sections provide the details for constructing Q-Q plots that are based on these distributions. Probability plots are constructed similarly except that the horizontal axis is scaled in percentile units.

Beta Distribution

To create a plot that is based on the beta distribution, PROC UNIVARIATE orders the observations from smallest to largest, and plots the i^{th} ordered observation against the quantile $B_{\alpha\beta}^{-1}\left(\frac{i-0.375}{n+0.25}\right)$ where $B_{\alpha\beta}^{-1}(\cdot)$ is the inverse normalized incomplete beta function, n is the number of nonmissing observations, and α and β are the shape parameters of the beta distribution.

The point pattern on the plot for $ALPHA=\alpha$ and $BETA=\beta$ tends to be linear with intercept θ and slope σ if the data are beta distributed with the specific density function

$$p(x) = \begin{cases} \frac{(x-\theta)^{\alpha-1}(\theta+\sigma-x)^{\beta-1}}{B(\alpha,\beta)\sigma^{(\alpha+\beta-1)}} & \text{for } \theta < x < \theta + \sigma \\ 0 & \text{for } x \leq \theta \text{ or } x \geq \theta + \sigma \end{cases}$$

where $B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ and θ is the lower threshold parameter, σ is the scale parameter ($\sigma > 0$), α the first shape parameter ($\alpha > 0$) and β is the second shape parameter ($\beta > 0$).

Exponential Distribution

To create a plot that is based on the exponential distribution, PROC UNIVARIATE orders the observations from smallest to largest, and plots the i^{th} ordered observation against the quantile $-\log\left(1 - \frac{i-0.375}{n+0.25}\right)$ where n is the number of nonmissing observations.

The point pattern on the plot tends to be linear with intercept θ and slope σ if the data are exponentially distributed with the specific density function

$$p(x) = \begin{cases} \frac{1}{\sigma} \exp\left(-\frac{x-\theta}{\sigma}\right) & \text{for } x \geq \theta \\ 0 & \text{for } x < \theta \end{cases}$$

where θ is a threshold parameter and σ is a positive scale parameter.

Gamma Distribution

To create a plot that is based on the gamma distribution, PROC UNIVARIATE orders the observations from smallest to largest, and plots the i^{th} ordered observation against the quantile $G_{\alpha}^{-1}\left(\frac{i-0.375}{n+0.25}\right)$ where G_{α}^{-1} is the inverse normalized incomplete gamma function, n is the number of nonmissing observations, and α is the shape parameter of the gamma distribution.

The point pattern on the plot tends to be linear with intercept θ and slope σ if the data are gamma distributed with the specific density function

$$p(x) = \begin{cases} \frac{1}{\sigma\Gamma(\alpha)} \left(\frac{x-\theta}{\sigma}\right)^{\alpha-1} \exp\left(-\frac{x-\theta}{\sigma}\right) & \text{for } x > \theta \\ 0 & \text{for } x \leq \theta \end{cases}$$

where θ is the threshold parameter, σ is the scale parameter ($\sigma > 0$), and α is the shape parameter ($\alpha > 0$).

Lognormal Distribution

To create a plot that is based on the lognormal distribution, PROC UNIVARIATE orders the observations from smallest to largest, and plots the i^{th} ordered observation against the quantile $\exp\left(\sigma\Phi^{-1}\left(\frac{i-0.375}{n+0.25}\right)\right)$ where $\Phi^{-1}(\cdot)$ is the inverse standard cumulative normal distribution, n is the number of nonmissing observations, and σ is the shape parameter of the lognormal distribution.

The point pattern on the plot for SIGMA= σ tends to be linear with intercept θ and slope $\exp(\zeta)$ if the data are lognormally distributed with the specific density function

$$p(x) = \begin{cases} \frac{1}{\sigma\sqrt{2\pi}(x-\theta)} \exp\left(-\frac{(\log(x-\theta)-\zeta)^2}{2\sigma^2}\right) & \text{for } x > \theta \\ 0 & \text{for } x \leq \theta \end{cases}$$

where θ is the threshold parameter, ζ is the scale parameter, and σ is the shape parameter ($\sigma > 0$).

Normal Distribution

To create a plot that is based on the normal distribution, PROC UNIVARIATE orders the observations from smallest to largest, and plots the i^{th} ordered observation against the quantile $\Phi^{-1}\left(\frac{i-0.375}{n+0.25}\right)$ where $\Phi^{-1}(\cdot)$ is the inverse cumulative standard normal distribution and n is the number of nonmissing observations.

The point pattern on the plot tends to be linear with intercept μ and slope σ if the data are normally distributed with the specific density function

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \quad \text{for all } x$$

where μ is the mean and σ is the standard deviation ($\sigma > 0$).

Three-Parameter Weibull Distribution

To create a plot that is based on a three-parameter Weibull distribution, PROC UNIVARIATE orders the observations from smallest to largest, and plots the i^{th} ordered observation against the quantile $\left(-\log\left(1 - \frac{i-0.375}{n+0.25}\right)\right)^{\frac{1}{c}}$ where n is the number of nonmissing observations, and α and c are the Weibull distribution shape parameters.

The point pattern on the plot for $C=c$ tends to be linear with intercept θ and slope σ if the data are Weibull distributed with the specific density function

$$p(x) = \begin{cases} \frac{c}{\sigma} \left(\frac{x-\theta}{\sigma}\right)^{c-1} \exp\left(-\left(\frac{x-\theta}{\sigma}\right)^c\right) & \text{for } x > \theta \\ 0 & \text{for } x \leq \theta \end{cases}$$

where θ is the threshold parameter, σ is the scale parameter ($\sigma > 0$), and c is the shape parameter ($c > 0$).

Two-Parameter Weibull Distribution

To create a plot that is based on a two-parameter Weibull distribution, PROC UNIVARIATE orders the observations from smallest to largest, and plots the log of the shifted i^{th} ordered observation $x_{(i)}$, denoted by $\log(x_{(i)} - \theta_0)$, against the quantile $\left(-\log\left(1 - \frac{i-0.375}{n+0.25}\right)\right)$ where n is the number of nonmissing observations.

Unlike the three-parameter Weibull quantile, the preceding expression is free of distribution parameters. This is why the $C=$ shape parameter is not required in the WEIBULL2 option.

The point pattern on the plot for $\text{THETA}=\theta_0$ tends to be linear with intercept $\log(\sigma)$ and slope $\frac{1}{c}$ if the data are Weibull distributed with the specific density function

$$p(x) = \begin{cases} \frac{c}{\sigma} \left(\frac{x-\theta_0}{\sigma}\right)^{c-1} \exp\left(-\left(\frac{x-\theta_0}{\sigma}\right)^c\right) & \text{for } x > \theta_0 \\ 0 & \text{for } x \leq \theta_0 \end{cases}$$

where θ_0 is the known lower threshold, σ is the scale parameter ($\sigma > 0$), and c is the shape parameter ($c > 0$).

Shape Parameters

Some distribution options in the PROBPLOT and QQPLOT statements require that you specify one or two shape parameters in parentheses after the distribution keyword. These are summarized in the following table:

Table 48.9 Shape Parameter Options

Distribution Keyword	Required Shape Parameter Option	Range
BETA	ALPHA= α , BETA= β	$\alpha > 0$, $\beta > 0$
EXPONENTIAL	None	
GAMMA	ALPHA= α	$\alpha > 0$
LOGNORMAL	SIGMA= σ	$\sigma > 0$
NORMAL	None	
WEIBULL	C= c	$c > 0$
WEIBULL2	None	

You can visually estimate the value of a shape parameter by specifying a list of values for the shape parameter option. PROC UNIVARIATE produces a separate plot for each value. Then you can use the value of the shape parameter that produces the most nearly linear point pattern. Alternatively, you can request that PROC UNIVARIATE use an estimated shape parameter to create the plot.

Note: For Q-Q plots that are requested with the WEIBULL2 option, you can estimate the shape parameter c from a linear pattern by using the fact that the slope of the pattern is $\frac{1}{c}$. Δ

Location and Scale Parameters

When you use the PROBPLOT statement to specify or estimate the location and scale parameters for a distribution, a diagonal distribution reference line appears on the probability plot. (An exception is the two-parameter Weibull distribution, where the line appears when you specify or estimate the scale and shape parameters.) Agreement between this line and the point pattern indicates that the distribution with these parameters is a good fit.

Note: Close visual agreement may not necessarily mean that the distribution is a good fit based on the criteria that are used by formal goodness-of-fit tests. Δ

When the point pattern on a Q-Q plot is linear, its intercept and slope provide estimates of the location and scale parameters. (An exception to this rule is the two-parameter Weibull distribution, for which the intercept and slope are related to the scale and shape parameters.) When you use the QQPLOT statement to specify or estimate the slope and intercept of the line, a diagonal distribution reference line appears on the Q-Q plot. This line allows you to check the linearity of the point pattern.

The following table shows which parameters to specify to determine the intercept and slope of the line:

Table 48.10 Intercept and Slope of Distribution Reference Line

Distribution	Parameters			Linear Pattern	
	Location	Scale	Shape	Intercept	Slope
BETA	θ	σ	α, β	θ	σ
EXPONENTIAL	θ	σ		θ	σ
GAMMA	θ	σ	α	θ	σ
LOGNORMAL	θ	ζ	σ	θ	$\exp(\zeta)$
NORMAL	μ	σ		μ	σ
WEIBULL (3-parameter)	θ	σ	c	θ	σ
WEIBULL2 (2-parameter)	θ_0 (known)	σ	c	$\log(\sigma)$	$\frac{1}{c}$

For the LOGNORMAL and WEIBULL2 options, you can specify the slope directly with the SLOPE= option. That is, for the LOGNORMAL option, when you specify THETA= θ_0 and SLOPE= $\exp(\zeta_0)$, PROC UNIVARIATE displays the same line as that which is specified by THETA= θ_0 and ZETA= ζ_0 . For the WEIBULL2 option, when you specify SIGMA= σ_0 and SLOPE= $\frac{1}{c_0}$, PROC UNIVARIATE displays the same line when you specify SIGMA= σ_0 and C= c_0 . Alternatively, you can request to use the estimated values of the parameters to determine the reference line.

Results: UNIVARIATE Procedure

By default, PROC UNIVARIATE produces tables of moments, basic statistical measures, tests for location, quantiles, and extreme observations. You must specify options in the PROC UNIVARIATE statement to produce other statistics and tables.

The CIBASIC option produces the table of the basic confidence measures that includes the confidence limits for the mean, standard deviation, and variance. The CIPCTLDF option and CIPCTLNORMAL option produce tables of confidence limits for the quantiles. The LOCCOUNT option produces the table that shows the number of values greater than, not equal to, and less than the value of MU0=. The FREQ option produces the table of frequencies counts. The NEXTRVAL= option produces the table with the frequencies of the extreme values. The NORMAL option produces the table with the tests for normality. The TRIMMED=, WINSORIZED=, and ROBUSTCALE options produce tables with robust estimators.

The table of trimmed or Winsorized means includes the percentage and the number of observations that are trimmed or Winsorized at each end, the mean and standard error, confidence limits, and the Student's t test. The table with robust measures of scale includes interquartile range, Gini's mean difference G , MAD , Q_n , and S_n , with their corresponding estimates of σ .

Missing Values

PROC UNIVARIATE excludes missing values for the analysis variable before calculating statistics. Each analysis variable is treated individually; a missing value for

an observation in one variable does not affect the calculations for other variables. The statements handle missing values as follows:

- If a BY or an ID variable value is missing, PROC UNIVARIATE treats it like any other BY or ID variable value. The missing values form a separate BY group.
- If the FREQ variable value is missing or nonpositive, PROC UNIVARIATE excludes the observation from the analysis.
- If the WEIGHT variable value is missing, PROC UNIVARIATE excludes the observation from the analysis.

PROC UNIVARIATE tabulates the number of the missing values and reports this information in the procedure output. Before the number of missing values is tabulated, PROC UNIVARIATE excludes observations when

- you use the FREQ statement and the frequencies are nonpositive
- you use the WEIGHT statement and the weights are missing or nonpositive (you must specify the EXCLNPWGT option).

Histograms

If you request a fitted parametric distribution with a HISTOGRAM statement, PROC UNIVARIATE creates a report that summarizes the fit in addition to the graphical display. The report includes information about

- parameters for the fitted curve, estimated mean, and estimated standard deviation
- EDF goodness-of-fit tests
- histogram intervals
- quantiles.

Histogram Intervals

If you specify the MIDPERCENTS suboption in parentheses after a density estimate option, PROC UNIVARIATE includes a table that lists the interval midpoints along with the observed and estimated percentages of the observations that lie in the interval. The estimated percentages are based on the fitted distribution. You can also specify the MIDPERCENTS suboption to request a table of interval midpoints with the observed percentage of observations that lie in the interval.

Quantiles

By default, PROC UNIVARIATE displays a table that lists observed and estimated quantiles for the 1, 5, 10, 25, 50, 75, 90, 95, and 99 percent of a fitted parametric distribution. You can use the PERCENTS= suboption to request that the quantiles for specific percentiles appear in the table.

ODS Table Names

PROC UNIVARIATE assigns a name to each table that it creates. You can use these names to reference the table when using the Output Delivery System (ODS) to select tables and create output data sets. For more information, see *SAS Output Delivery System User's Guide*.

Table 48.11 ODS Tables Produced with the PROC UNIVARIATE Statement

Table Name	Description	Option
BasicIntervals	confidence intervals for mean, standard deviation, variance	CIBASIC
BasicMeasures	measures of location and variability	default
ExtremeObs	extreme observations	default
ExtremeValues	extreme values	NEXTRVAL=
Frequencies	frequencies	FREQ
LocationCounts	counts used for sign test and signed rank test	LOCCOUNT
MissingValues	missing values	default
Modes	modes	MODES
Moments	sample moments	default
Plots	line printer plots	PLOTS
Quantiles	quantiles	default
RobustScale	robust measures of scale	ROBUSTSCALE
SSPlots	line printer side-by-side box plot	PLOTS (with BY statement)
TestsForLocation	tests for location	default
TestsForNormality	tests for normality	NORMALTEST
TrimmedMeans	trimmed means	TRIMMED=
WinsorizedMeans	Winsorized means	WINSORIZED=

Table 48.12 ODS Tables Produced with the HISTOGRAM Statement

Table Name	Description	Option
Bins	histogram bins	MIDPERCENTS suboption
FitQuantiles	quantiles of fitted distribution	any distribution option
GoodnessOfFit	goodness-of-fit tests for fitted distribution	any distribution option
ParameterEstimates	parameter estimates for fitted distribution	any distribution option

Output Data Set

PROC UNIVARIATE can create one or more output SAS data sets. When you specify an OUTPUT statement and no BY statement, PROC UNIVARIATE creates an output data set that contains one observation. If you use a BY statement, the corresponding output data set contains an observation with statistics for each BY group. The procedure does not print the output data set. Use PROC PRINT, PROC REPORT, or another SAS reporting tool to print the output data set.

The output data set includes

- BY statement variables
- variables that contain statistics
- variables that contain percentiles.

The BY variables indicate which BY group each observation summarizes. When you omit a BY statement, the procedure computes statistics and percentiles by using all the observations in the input data set. When you use a BY statement, the procedure computes statistics and percentiles by using the observations within each BY group.

OUTHISTOGRAM= Data Set

You can create a OUTHISTOGRAM= data in the HISTOGRAM statement that contains information about histogram intervals. Because you can specify multiple HISTOGRAM statements with the UNIVARIATE procedure, you can create multiple OUTHISTOGRAM= data sets.

The data set contains a group of observations for each variable that the HISTOGRAM statement plots. The group contains an observation for each interval of the histogram, beginning with the leftmost interval that contains a value of the variable and ending with the rightmost interval that contains a value of the variable. These intervals will not necessarily coincide with the intervals displayed in the histogram since the histogram may be padded with empty intervals at either end. If you superimpose one or more fitted curves on the histogram, the OUTHISTOGRAM= data set contains multiple groups of observations for each variable (one group for each curve). If you use a BY statement, the OUTHISTOGRAM= data set contains groups of observations for each BY group. ID variables are not saved in the OUTHISTOGRAM= data set.

The variables in OUTHISTOGRAM= data set are

CURVE	name of fitted distribution (if requested in HISTOGRAM statement)
EXPPCT	estimated percent of population in histogram interval determined from optional fitted distribution
MIDPT	midpoint of fitted distribution
OBSPCT	percent of variable values in histogram interval
VAR	variable name

Examples: UNIVARIATE Procedure

Example 1: Univariate Analysis for Multiple Variables

Procedure features:

VAR statement

Other features:

ODS SELECT statement

This example computes the univariate statistics for two variables.

Program

Set the SAS system options. The NODATE option specifies to omit the date and time when the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines on a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=72;
```

Create the STATEPOP data set. This data set contains information from the 1990 and 2000 U.S. Census estimates of the population in metropolitan and nonmetropolitan areas. The 50 states and the District of Columbia are divided into four geographic regions. The data are organized by state within each region. The metropolitan and nonmetropolitan population counts are stored in one observation for each year. A DATA step "STATEPOP" on page 1670 creates the data set.

```
data statepop;
  input State $ CityPop_1990 CityPop_2000 NonCityPop_1990 NonCityPop_2000 Region @@;
  label citypop_1990= '1990 metropolitan pop in millions'
        noncitypop_1990='1990 nonmetropolitan pop in millions'
        citypop_2000= '2000 metropolitan pop in millions'
        noncitypop_2000='2000 nonmetropolitan pop in million'
        region='Geographic region';
  datalines;
ME    .443    .467    .785    .808    1    NH    .659    .740    .450    .496    1
VT    .152    .169    .411    .439    1    MA    5.788    6.088    .229    .261    1
RI    .938    .986    .065    .062    1    CT    3.148    3.257    .140    .149    1
    ...more lines of data...
WA    4.036    4.899    .830    .995    4    OR    2.056    2.502    .787    .919    4
CA    28.797    32.750    .961    1.121    4    AK    .226    .260    .324    .367    4
HI    .836    .876    .272    .335    4
;
```

Select output objects by name. The ODS SELECT statement specifies two output objects to send to all open destinations: a table of moments and a table of quantiles.

```
ods select Moments Quantiles;
```

Generate the statistical tables for the analysis variables. The PROC UNIVARIATE statement calculates univariate statistics for numeric variables in the STATEPOP data set. The VAR statement specifies the analysis variables and their order in the output.

```
proc univariate data=statepop;
  var citypop_1990 citypop_2000;
```

Specify the title.

```

    title 'United States Census of Population and Housing';
run;

```

Output

Univariate statistics for both analysis variables appear on separate pages. Because each population value is unique, the mode is missing.

United States Census of Population and Housing				1
The UNIVARIATE Procedure				
Variable: CityPop_1990 (1990 metropolitan pop in millions)				
Moments				
N	51	Sum Weights	51	
Mean	3.89037255	Sum Observations	198.409	
Std Deviation	5.15898276	Variance	26.6151031	
Skewness	2.87381702	Kurtosis	10.5609336	
Uncorrected SS	2102.64008	Corrected SS	1330.75516	
Coeff Variation	132.608965	Std Error Mean	0.72240208	
Quantiles (Definition 5)				
	Quantile	Estimate		
	100% Max	28.797		
	99%	28.797		
	95%	14.166		
	90%	9.574		
	75% Q3	4.380		
	50% Median	2.422		
	25% Q1	0.787		
	10%	0.270		
	5%	0.221		
	1%	0.134		
	0% Min	0.134		

United States Census of Population and Housing				2
The UNIVARIATE Procedure				
Variable: CityPop_2000 (2000 metropolitan pop in millions)				
Moments				
N	51	Sum Weights	51	
Mean	4.43072549	Sum Observations	225.967	
Std Deviation	5.8469492	Variance	34.186815	
Skewness	2.90620484	Kurtosis	10.7563067	
Uncorrected SS	2710.5385	Corrected SS	1709.34075	
Coeff Variation	131.963698	Std Error Mean	0.81873665	
Quantiles (Definition 5)				
	Quantile	Estimate		
	100% Max	32.750		
	99%	32.750		
	95%	17.473		
	90%	10.392		
	75% Q3	5.437		
	50% Median	2.807		
	25% Q1	0.876		
	10%	0.306		
	5%	0.260		
	1%	0.148		
	0% Min	0.148		

Example 2: Identifying Extreme Values and Creating a Histogram

Procedure features:

PROC UNIVARIATE statement options:

NEXTROBS=

NEXTRVAL=

HISTOGRAM statement

CFILL=

MIDPOINTS=

PFILL=

ID statement

Other features:

GOPTIONS statement

ODS SELECT statement

Data set: STATEPOP on page 1544

This example

- ☐ identifies extreme observations
- ☐ creates a histogram of the data distribution.

Program

Set the graphics environment. The GOPTIONS statement sets the graphics environment to control the appearance of graphic elements. HTITLE= and HTEXT= specify the text height in GUNIT= units. FTEXT= and FTITLE= specify the font.*

```
options htitle=4 htext=3 gunit=pct ftext=swiss ftitle=swiss;
```

Set the SAS system options. The NODATE option specifies to omit the date and time when the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines on a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=68;
```

Select output objects by name. The ODS SELECT statement specifies four output objects to send to all open destinations: a table of basic measures, a table of extreme observations, a table of extreme values, and a histogram plot.

```
ods select BasicMeasures ExtremeObs ExtremeValues Univar;
```

Generate the default statistics and the extreme values. The PROC UNIVARIATE statement calculates univariate statistics and two tables that list four extreme observations and extreme values.

```
proc univariate data=statepop nextrobs=2 next rval=4;
```

Specify the analysis variable. The VAR statement specifies that CityPop_2000 is the analysis variable.

```
var citypop_2000;
```

Request the variables that have their extreme values listed. The ID statement identifies these variables.

```
id region state;
```

Create a histogram. The HISTOGRAM statement creates a histogram for all the analysis variables. The MIDPOINTS= option specifies a list of values to use as bin midpoints. The PFILL= option specifies a crosshatched fill pattern for the bars and the CFILL= option specifies blue as the fill color.

```
histogram / midpoints=0 to 35 by 5 pfill=x cfill=blue;
```

Specify the title.

* For additional information about the GOPTIONS statement, see *SAS/GRAPH Reference*.

```

title 'United States Census of Population and Housing';
run;

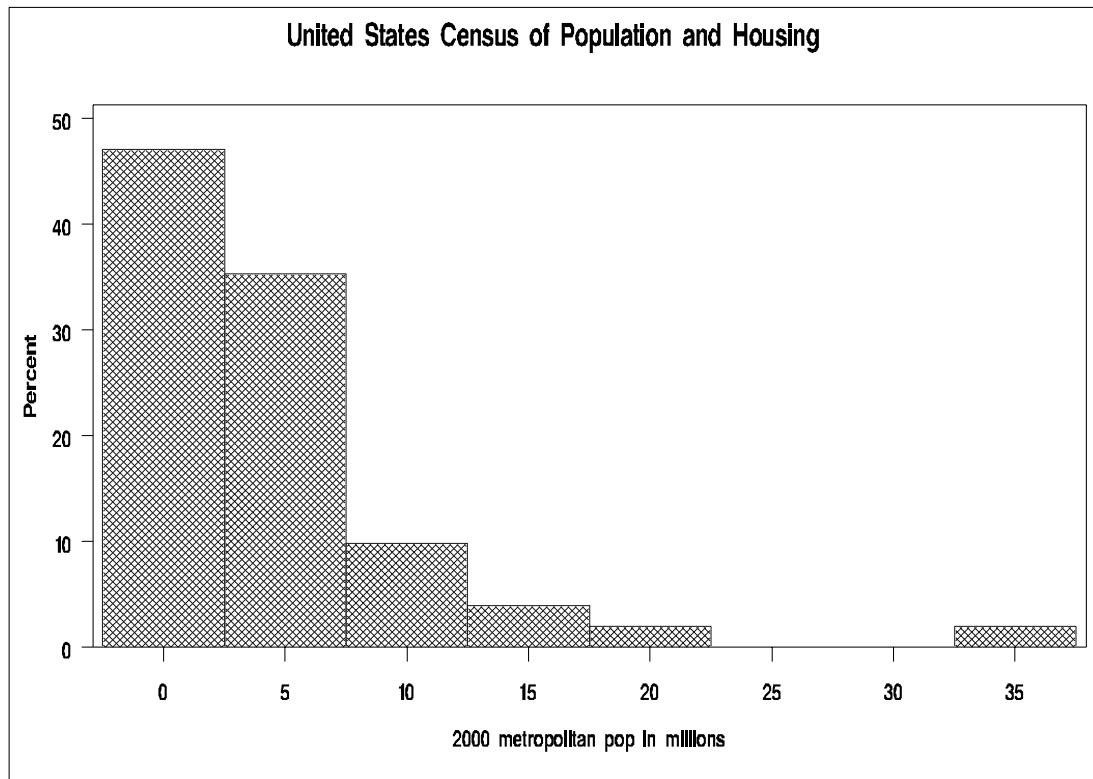
```

Output

Because each value of population is unique, the mode is missing.

The Extreme Observations table lists values of the ID variables, Region and State. Regions 4 and 1 report the lowest metropolitan populations, while regions 2 and 4 report the highest populations. The states with the four most extreme observations are WY, VT, TX, and CA. The Extreme Values table lists the four lowest unique values and the four highest unique values.

United States Census of Population and Housing						1	
The UNIVARIATE Procedure							
Variable: CityPop_2000 (2000 metropolitan pop in millions)							
Basic Statistical Measures							
Location				Variability			
Mean	4.430725	Std Deviation		5.84695			
Median	2.807000	Variance		34.18682			
Mode	.	Range		32.60200			
		Interquartile Range		4.56100			
Extreme Observations							
-----Lowest-----				-----Highest-----			
Value	Region	State	Obs	Value	Region	State	Obs
0.148	4	WY	41	17.692	2	TX	26
0.169	1	VT	3	32.750	4	CA	49
Extreme Values							
-----Lowest-----				-----Highest-----			
Order	Value	Order	Value	Order	Value	Order	Value
1	0.148	48	14.837	48	14.837	48	14.837
2	0.169	49	17.473	49	17.473	49	17.473
3	0.260	50	17.692	50	17.692	50	17.692
4	0.261	51	32.750	51	32.750	51	32.750



Example 3: Computing Robust Estimators

Procedure features:

PROC UNIVARIATE statement options:

TRIMMED=

WINSORIZED=

Other features:

ODS EXCLUDE statement

Data set: STATEPOP on page 1544

This example

- ☐ computes two trimmed means
- ☐ computes a Winsorized mean.

Program

Set the SAS system options. The NODATE option specifies to omit the date and time when the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines on a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=72;
```

Exclude output objects by name. The ODS EXCLUDE statement specifies three output objects to exclude from the open destinations: a table of measures of location and variability, a table of extreme observations, and a table of quantiles.

```
ods exclude TestsForLocation ExtremeObs Quantiles;
```

Generate the default statistics and the robust statistics. The PROC UNIVARIATE statement calculates basic statistical measures. TRIMMED= computes two trimmed means after removing 6 observations and 25 percent of the observations. WINSORIZED= computes a Winsorized mean that replaces 10 percent of the observations.

```
proc univariate data=statepop trimmed=6 .25 winsorized=.1;
```

Specify the analysis variable. The VAR statement specifies that CityPop_2000 is the analysis variable.

```
var citypop_2000;
```

Specify the title.

```
title 'United States 2000 Estimate of Population and Housing';  
run;
```


Output

Because each value of population is unique, the mode is missing.

Both the trimmed and Winsorized means are smaller than the arithmetic mean. This may be due to the positive skewness of the data. PROC UNIVARIATE trims 6 observations or 11.76 percent of the data from the tails. When you request to trim 25 percent of the data, PROC UNIVARIATE trims 13 observations or 25.49 percent of the data from the tails. This is because the number of observations trimmed is the smallest integer greater than or equal to 12.75 ($.25 \times 51$). Likewise, when you compute a Winsorized mean for 10 percent of the data ($.1 \times 51 = 5.1$), PROC UNIVARIATE uses 6 observations or 11.76 percent of the data from the tails.

United States 2000 Estimate of Population and Housing

1

The UNIVARIATE Procedure

Variable: CityPop_2000 (2000 metropolitan pop in millions)

Moments

N	51	Sum Weights	51
Mean	4.43072549	Sum Observations	225.967
Std Deviation	5.8469492	Variance	34.186815
Skewness	2.90620484	Kurtosis	10.7563067
Uncorrected SS	2710.5385	Corrected SS	1709.34075
Coeff Variation	131.963698	Std Error Mean	0.81873665

Basic Statistical Measures

Location		Variability	
Mean	4.430725	Std Deviation	5.84695
Median	2.807000	Variance	34.18682
Mode	.	Range	32.60200
		Interquartile Range	4.56100

Trimmed Means

Percent Trimmed in Tail	Number Trimmed in Tail	Trimmed Mean	Std Error Trimmed Mean	95% Confidence Limits		DF
11.76	6	3.098795	0.558886	1.967390	4.230200	38
25.49	13	2.711200	0.494141	1.691342	3.731058	24

Trimmed Means

Percent Trimmed in Tail	t for H0: Mu0=0.00	Pr > t
11.76	5.544595	<.0001
25.49	5.486689	<.0001

Winsorized Means

Percent Winsorized in Tail	Number Winsorized in Tail	Winsorized Mean	Std Error Winsorized Mean	95% Confidence Limits		DF
11.76	6	3.508608	0.560613	2.373706	4.643510	38

Winsorized Means

Percent Winsorized in Tail	t for H0: Mu0=0.00	Pr > t
11.76	6.258517	<.0001

Example 4: Performing a Sign Test Using Paired Data

Procedure features:

PROC UNIVARIATE statement option:

CIBASIC
LOCCOUNT
MODES

Other features:

LABEL statement
ODS EXCLUDE statement

This example

- ☐ computes difference scores for paired data
- ☐ lists all values of the mode
- ☐ examines the tests for location to determine if the median difference between scores is zero
- ☐ lists the number of observations less than, greater than, and equal to zero
- ☐ specifies the confidence levels for the confidence limits

Program

Set the SAS system options. The NODATE option specifies to omit the date and time when the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines on a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Create the SCORE data set. This data set contains test scores for college students who took two tests and a final exam. ScoreChange contains the difference in the scores between the first test and the second test.

```
data score;
    input Student $ Test1 Test2 Final @@;
    ScoreChange=test2-test1;
    datalines;
Capalleti 94 91 87 Dubose 51 65 91
Engles 95 97 97 Grant 63 75 80
Krupski 80 75 71 Lundsford 92 55 86
Mcbane 75 78 72 Mullen 89 82 93
Nguyen 79 76 80 Patel 71 77 83
Si 75 70 73 Tanaka 87 73 76
;
```

Exclude output objects by name. The ODS EXCLUDE statement specifies three output objects to exclude from the open destinations: a table of moments, a table of extreme observations, and a table of quantiles.

```
ods exclude Moments ExtremeObs Quantiles;
```

Generate the default statistics, the location counts, and the modes. Compute confidence limits. The PROC UNIVARIATE statement calculates basic statistical measures, tests for location, quantiles, and extreme observations. LOCCOUNT produces a Location Counts table. MODES produces a Modes table. CIBASIC(ALPHA=.05) specifies a 95 percent confidence limit for the basic measures.

```
proc univariate data=score loccount modes cibasic(alpha=.05);
```

Specify the analysis variable. The VAR statement specifies that ScoreChange is the analysis variable.

```
var scorechange;
```

Specify a label for the report. The LABEL statement associates a label with the analysis variable for the duration of the PROC step. The TITLE statement specifies a title.

```
label scorechange='Change in Test Scores';  
title 'Test Scores for a College Course';  
run;
```

Output

PROC UNIVARIATE includes the variable label in the report. The report also provides a message to indicate that the lowest mode is shown in the Basic Statistical Measures table. The Modes table reports all the mode values.

The mean of -3.08 indicates an average decrease in test scores from Test1 to Test2. The 95 percent confidence limits (-11.56, 5.39), which include 0, and the tests for location indicate that the decrease is not statistically significant.

The Tests for Location table includes three hypothesis tests. The Student's *t* statistic assumes that the data are approximately normally distributed. The sign test and signed rank test are nonparametric tests. The signed rank test requires a symmetric distribution. If the distribution is symmetric, then you expect a skewness value that is close to zero. Because the value -1.42 indicates some distribution skewness, examine the sign test to determine if the difference in test scores is zero. The large *p*-value (.7744) provides insufficient evidence of a difference in test score medians.

Test Scores for a College Course

1

The UNIVARIATE Procedure

Variable: ScoreChange (Change in Test Scores)

Basic Statistical Measures

Location

Variability

Mean

Median

Mode

-3.08333

-3.00000

-5.00000

Std Deviation

Variance

Range

Interquartile Range

13.33797

177.90152

51.00000

10.50000

NOTE: The mode displayed is the smallest of 2 modes with a count of 2.

Modes

Mode

Count

-5

-3

2

2

Basic Confidence Limits Assuming Normality

Parameter

Estimate

95% Confidence Limits

Mean

Std Deviation

Variance

-3.08333

13.33797

177.90152

-11.55788

9.44856

89.27519

5.39121

22.64625

512.85267

Tests for Location: Mu0=0

Test

-Statistic-

-----p Value-----

Student's t

Sign

Signed Rank

t

M

S

-0.80079

-1

-8.5

Pr > |t|

Pr >= |M|

Pr >= |S|

0.4402

0.7744

0.5278

Location Counts: Mu0=0.00

Count

Value

Num Obs > Mu0

Num Obs ^= Mu0

Num Obs < Mu0

5

12

7

Example 5: Examining the Data Distribution and Saving Percentiles

Procedure features:

PROC UNIVARIATE statement options:

ALPHA=
CIBASIC
MU0=
NORMAL
PLOTS

OUTPUT statement

PROBPLOT statement options:

PCTLMINOR
SQUARE

Other features:

GOPTIONS statement
ODS EXCLUDE statement
PRINT procedure
SYMBOL statement

Data set: SCORE on page 1552

This example

- ☐ specifies the confidence level for the confidence limits
- ☐ computes a lower confidence limit for the parameters
- ☐ specifies the null hypothesis mean for the tests for locations
- ☐ tests the hypothesis that the data are normally distributed
- ☐ produces a stem-and-leaf plot, box plot, and two normal probability plots
- ☐ requests graphical enhancements that change symbol type and text font for the high-resolution normal probability plot
- ☐ displays minor tick marks between major tick marks on the percentile axis for the high-resolution normal probability plot
- ☐ computes additional percentiles
- ☐ creates an output data set with percentiles
- ☐ prints the output data set.

Program

Set the graphics environment. The GOPTIONS statement sets the graphics environment to control the appearance of graphic elements. HTITLE= and HTEXT= specify the text height in GUNIT= units. FTEXT= and FTITLE= specify the font.*

* For additional information about the GOPTIONS statement, see *SAS/GRAPH Reference*.

Specify the plot symbol character. The SYMBOL statement defines the characteristics of the symbol that appears in the plot. VALUE= specifies a dot as the plot symbol. By default, the plot symbol is the plus sign (+).

```
symbol value=dot;
```

Set the SAS system options. The NODATE option specifies to omit the date and time when the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines on a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Exclude output objects by name. The ODS EXCLUDE statement specifies three output objects to exclude from the open destinations: a table of measures of location and variability, a table of extreme observations, and a table of quantiles.

```
ods exclude Moments ExtremeObs Quantiles;
```

Generate the default statistics and various confidence limits. Specify the mean for the test of location. Create plots of the data distribution. The PROC UNIVARIATE statement calculates basic statistical measures, and tests for location. MU0= requests a test that the population mean equals 80. ALPHA= specifies a 90 percent confidence limit for all statistics. CIBASIC computes lower confidence limits for the basic measures. NORMAL computes tests for normality. PLOTS requests plots of the data distribution.

```
proc univariate data=score mu0=80 alpha=.1 cibasic(type=lower)
      normal plots;
```

Specify the analysis variable. The VAR statement specifies that Final is the analysis variable.

```
var final;
```

Create a normal probability plot. The PROBPLOT statement creates a normal probability plot for the analysis variables. SQUARE displays the probability plot in a square frame instead of a rectangular frame. PCTLMINOR specifies that minor tick marks appear between the major tick marks on the horizontal axis.

```
probplot /square pctlminor;
```

Create the output data set. The OUTPUT statement creates the PCTSCORE data set with five variables. MEDIAN= saves the median. PCTLPTS= saves four percentiles. PCTLPRE= specifies a prefix name. PCTLNAME= specifies suffix names for the variables that contain the first three percentiles. The name of the variable that contains the 70th percentile uses the default suffix.

```
output out=pctscore median=Median pctlpts=98 50 20 70
      pctlpre=Pctl_ pctlname=Top Mid Low;
```

Specify the title.

```
title 'Examining the Distribution of Final Exam Scores';  
run;
```

Print the data set. PROC PRINT prints the PCTSCORE data set. The TITLE statement specifies a title.

```
proc print data=pctscore noobs;  
  title1 'Quantile Statistics for Final Exam Scores';  
  title2 'Output Data Set from PROC UNIVARIATE';  
run;
```

Output

The estimate of the mean test score is 82.4, with a standard deviation of 8.6. The 90 percent lower confidence limit for the mean is 79.

The Tests for Location table includes three hypothesis tests. To determine whether the Student's *t* statistic is appropriate, you must determine if the data are approximately normally distributed.

PROC UNIVARIATE calculates the Shapiro-Wilk *W* statistic because the sample size is below 2000. All *p*-values from the tests for normality are >0.15, which provides insufficient evidence to reject the assumption of normality. The probability plot also supports the assumption that the data are normal. Therefore, the *t* statistic appears appropriate. The *p*-value of .35 for this test provides insufficient evidence to reject the null hypothesis that the mean test score is 80. Examination of the box plot, which is nonsymmetric, and the small sample size, which causes low power, make the sign test a more appropriate test of location. The *p*-value of .75 for this test provides insufficient evidence to reject the null hypothesis that the mean test score is 80.

Examining the Distribution of Final Exam Scores

1

The UNIVARIATE Procedure
Variable: Final

Basic Statistical Measures

Location		Variability	
Mean	82.41667	Std Deviation	8.59660
Median	81.50000	Variance	73.90152
Mode	80.00000	Range	26.00000
		Interquartile Range	14.50000

Basic Confidence Limits Assuming Normality

Parameter	Estimate	Lower 90% CL
Mean	82.41667	79.03314
Std Deviation	8.59660	6.85984
Variance	73.90152	47.05738

Tests for Location: Mu0=80

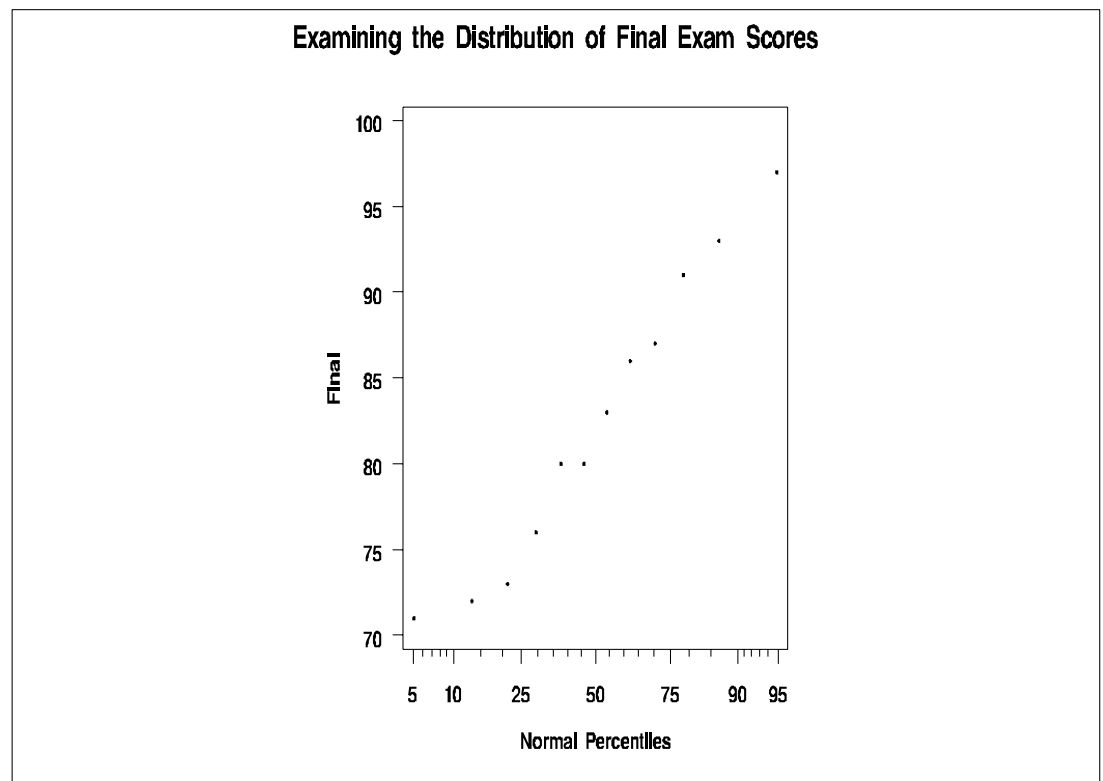
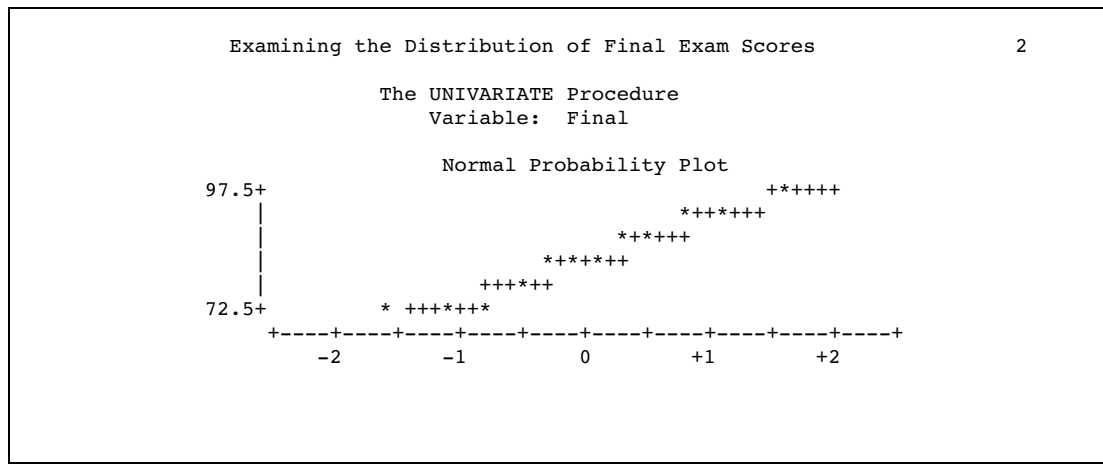
Test	-Statistic-	-----p Value-----	
Student's t	t 0.973825	Pr > t	0.3511
Sign	M 1	Pr >= M	0.7539
Signed Rank	S 8	Pr >= S	0.4434

Tests for Normality

Test	--Statistic--	-----p Value-----	
Shapiro-Wilk	W 0.952903	Pr < W	0.6797
Kolmogorov-Smirnov	D 0.113328	Pr > D	>0.1500
Cramer-von Mises	W-Sq 0.028104	Pr > W-Sq	>0.2500
Anderson-Darling	A-Sq 0.212693	Pr > A-Sq	>0.2500

Stem Leaf	#	Boxplot
9 7	1	
9 13	2	
8 67	2	
8 003	3	
7 6	1	
7 123	3	-----+

-----+-----+-----+
Multiply Stem.Leaf by 10***1



The PCTSCORE data set contains one observation. The median value in Median is equivalent to the 50th percentile in PCTL_MID.

Quantile Statistics for Final Exam Scores					4
Output Data Set from PROC UNIVARIATE					
Median	Pctl_Top	Pctl_Mid	Pctl_Low	Pctl_70	
81.5	97	81.5	73	87	

Example 6: Creating an Output Data Set with Multiple Analysis Variables

Procedure features:

PROC UNIVARIATE statement option:

 NOPRINT

OUTPUT statement

VAR statement

Other features:

PRINT procedure

Data set: SCORE on page 1552

This example

- ☐ suppresses the reporting of univariate statistics
- ☐ computes additional percentiles for two variables
- ☐ creates an output data set with descriptive statistics and percentiles
- ☐ prints the output data set.

Program

Set the SAS system options. The NODATE option specifies to omit the date and time when the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines on a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Suppress the printing of the statistics tables. NOPRINT suppresses all the tables of statistics that the PROC UNIVARIATE statement creates.

```
proc univariate data=score noprint;
```

Specify the analysis variables. The VAR statement specifies the analysis variables and their order in the output.

```
var test1 test2;
```

Create the output data set. The OUTPUT statement creates the TESTSTAT data set with nine variables. MEAN= saves the mean for Test1 and Test2. STD= saves the standard deviation for Test1. PCTLPTS= calculates three percentiles and PCTLPRE= specifies prefix names for the analysis variables. PCTLNAME= specifies a suffix name for the 33.3 percentile.

```
output out=teststat mean=MeanTest1 MeanTest2  
std=StdDeviationTest1
```

```

        pctlpts=33.3 66 99.9
        pctlpre=Test1_
        Test2_ pctlname=Low ;
run;

```

Print the data set. PROC PRINT prints the TESTSTAT data set. The TITLE statements specify the two titles that are printed.

```

proc print data=teststat noobs;
    title1 'Univariate Statistics for Two College Tests';
    title2 'Output Data Set from PROC UNIVARIATE';
run;

```

Output

The TESTSTAT data set contains one observation with the mean for the two analysis variables and the standard deviation for the first analysis variable. The remaining six variables contain computed percentiles.

Univariate Statistics for Two College Tests								1
Output Data Set from PROC UNIVARIATE								
Mean	Mean	Std	Test1_		Test1_	Test2_	Test2_	
Test1	Test2	Deviation	Test1_	Test1_66	99_9	Low	Test2_66	99_9
Test1	Test2	Test1	Low	Test1_66	99_9	Low	Test2_66	99_9
79.25	76.1667	13.3152	75	87	95	73	77	97

Example 7: Fitting Density Curves

Procedure features:

PROC UNIVARIATE statement options:

NOPRINT

HISTOGRAM statement options:

CBARLINE=

CFILL=

EXP

FILL

L=

MIDPOINTS=

NOPRINT

NORMAL

VAR statement

Other features:

GOPTIONS statement

RANNOR function

RANEXP function

This example

- creates a sample of 100 observations from a normal distribution and an exponential distribution
- suppresses the tables of descriptive statistics
- creates histograms with superimposed density curves for the normal and exponential distributions
- requests goodness-of-fit tests for a fitted exponential distribution
- specifies the midpoints for histogram intervals
- requests graphical enhancements that change plot colors and line types.

Program

Set the SAS system options. The NODATE option specifies to omit the date and time when the SAS job began. The PAGENO= option specifies the page number for the next page of output that SAS produces. The LINESIZE= option specifies the line size. The PAGESIZE= option specifies the number of lines on a page of SAS output.

```
options nodate pageno=1 linesize=80 pagesize=60;
```

Set the graphics environment. The GOPTIONS statement sets the graphics environment to control the appearance of graphics elements. HTITLE= and HTEXT= specify the text height in GUNIT= units. FTEXT= and FTITLE= specify the font.*

```
goptions htitle=4 htext=3 gunit=pct ftext=swissb ftitle=swissb;
```

Create the data set. The data set DISTRDATA contains two variables and 100 observations. The RANNOR function creates a random variate from a normal distribution with a mean of 50 and a standard deviation of 10 that is stored in the Normal_x variable. The RANEXP function creates a random variate from an exponential distribution that is stored in the Exponential_x variable.

```
data distrdata;
  drop n;
  label Normal_x='Normal Random Variable'
        Exponential_x='Exponential Random Variable';
  do n=1 to 100;
    Normal_x=10*rannor(53124)+50;
    Exponential_x=ranexp(18746363);
    output;
  end;
run;
```

Suppress the printing of the statistics tables for the analysis variable. NOPRINT suppresses the tables of statistics that the PROC UNIVARIATE statement creates. The VAR statement specifies that Normal_x is the analysis variable.

* For additional information about the GOPTIONS statement, see *SAS/GRAPH Reference*.

```
proc univariate data=distrdata noprint;
  var Normal_x;
```

Create a histogram with a normal distribution but suppress the printing of the statistics. The HISTOGRAM statement creates a histogram for the analysis variable Normal_x. The NORMAL option superimposes the fitted density curve for a normal distribution. NOPRINT suppresses the tables of statistics that summarize the fitted density curve. The CBARLINE= option specifies the color to outline the histogram bars.

```
  histogram Normal_x /normal(noprint) cbarline=grey ;
```

Specify the title.

```
  title '100 Obs Sampled from a Normal Distribution';
run;
```

Suppress the printing of the statistics tables for the analysis variable. Another PROC step will execute so that output displays a new customized title. The VAR statement specifies that Exponential_x is the analysis variable.

```
proc univariate data=distrdata noprint;
  var Exponential_x;
```

Create a histogram with an exponential distribution. The HISTOGRAM statement creates a histogram for the analysis variable Exponential_x. The EXP option superimposes a fitted density curve for an exponential distribution. The FILL option specifies to fill the area under the exponential density curve with the CFILL= color. The L= option specifies a distinct line type for the density curve. The MIDPOINTS= option specifies a list of values to use as bin midpoints.

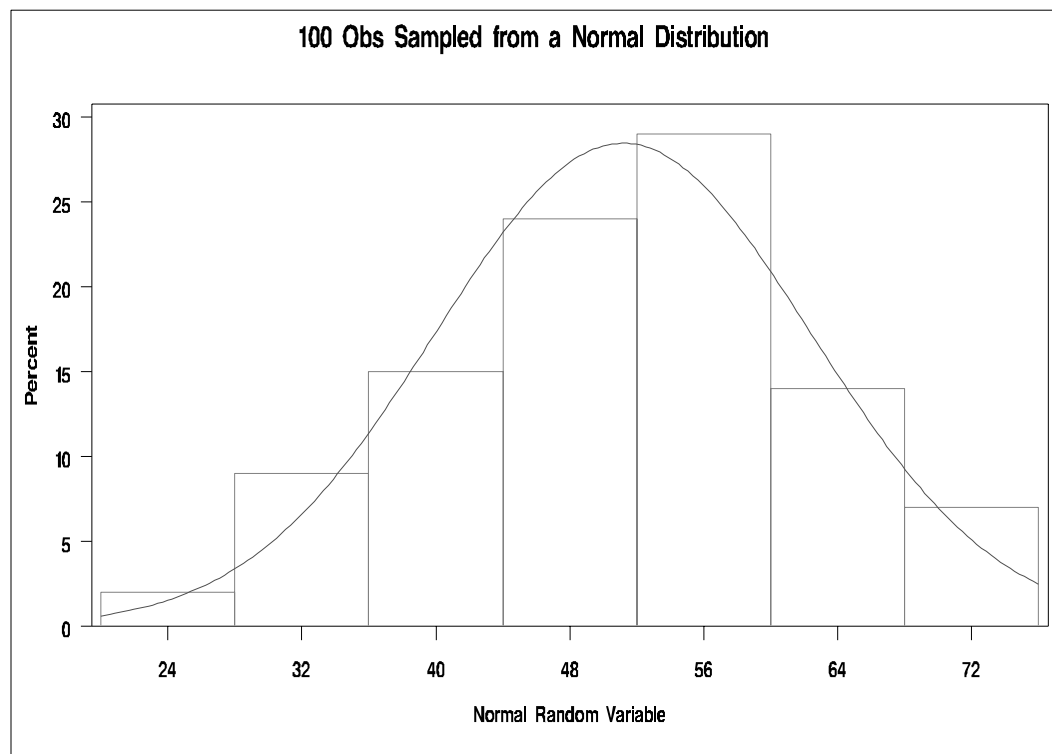
```
  histogram /exp(fill l=3) cfill=yellow midpoints=.05 to 5.55 by .25;
```

Specify the title.

```
  title '100 Obs Sampled from an Exponential Distribution';
run;
```

Output

Figure 48.7 A Histogram Superimposed with Normal Curve



The output includes parameters estimates for the exponential curve. The exponential parameter threshold parameter θ is 0 because the THETA= option was omitted. A maximum likelihood estimate is computed for the scale parameter σ .

PROC UNIVARIATE provides three goodness-of-fit tests for the exponential distribution that are based on the empirical distribution function. The **p**-values for the exponential distribution are larger than the usual cutoff values of 0.05 and 0.10, which indicates not to reject the null hypothesis that the data are exponentially distributed.

100 Obs Sampled from an Exponential Distribution

1

The UNIVARIATE Procedure
Fitted Distribution for Exponential_x

Parameters for Exponential Distribution

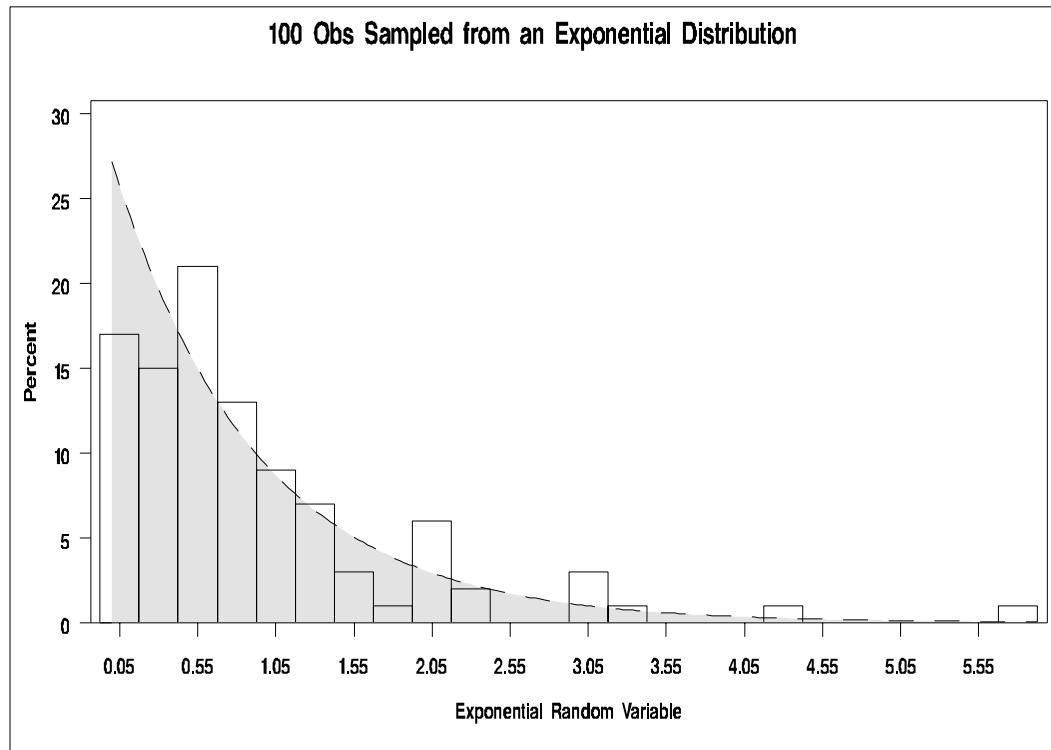
Parameter	Symbol	Estimate
Threshold	Theta	0
Scale	Sigma	0.919698
Mean		0.919698
Std Dev		0.919698

Goodness-of-Fit Tests for Exponential Distribution

Test		---Statistic---	-----p Value-----
Kolmogorov-Smirnov	D	0.05860511	Pr > D >0.500
Cramer-von Mises	W-Sq	0.05537161	Pr > W-Sq >0.500
Anderson-Darling	A-Sq	0.33426909	Pr > A-Sq >0.500

Quantiles for Exponential Distribution

Percent	-----Quantile-----	
	Observed	Estimated
1.0	0.00560	0.00924
5.0	0.05600	0.04717
10.0	0.06979	0.09690
25.0	0.30030	0.26458
50.0	0.62936	0.63749
75.0	1.20484	1.27497
90.0	2.08322	2.11768
95.0	3.00117	2.75517
99.0	5.07829	4.23536

Figure 48.8 A Histogram Superimposed with an Exponential Curve

Example 8: Displaying a Reference Line on a Normal Quantile-Quantile Plot

Procedure features:

PROC UNIVARIATE statement options:

NOPRINT

INSET statement options:

CFILL=

FORMAT=

HEADER=

POSITION=

REFPOINT=

statistical-keyword

QQPLOT statement options:

CFRAME=

MU=

NORMAL

PCTLAXIS

SIGMA=

VAR statement

Other features:

GOPTIONS statement

SYMBOL statement

Data Set: DISTRDATA on page 1562

This example

- suppresses the tables of descriptive statistics
- creates a normal quantile-quantile plot
- requests a diagonal reference line that corresponds to the normal distribution with estimated parameters μ and σ
- enhances the plot by inseting a table of summary statistics
- specifies background colors for the plot and the table of statistics
- adds a nonlinear percentile axis that is opposite the theoretical quantile axis
- requests graphical enhancements that change symbol type and text font.

Program

Set the graphics environment. The GOPTIONS statement sets the graphics environment to control the appearance of graphics elements. HTITLE= and HTEXT= specify the text height in GUNIT= units. FTEXT= and FTITLE= specify the font.*

```
goptions htitle=4 htext=3 gunit=pct ftext=swissb ftitle=swissb;
```

Specify the plot symbol character. The SYMBOL statement defines the characteristics of the symbol that appears in the plot. VALUE= specifies a star as the plot symbol. By default, the plot symbol is the plus sign (+).

```
symbol value=star;
```

Suppress the printing of the statistics tables for the analysis variable. NOPRINT suppresses the tables of statistics that the PROC UNIVARIATE statement creates. The VAR statement specifies that Normal_x is the analysis variable.

```
proc univariate data=distrdata noprint;
  var Normal_x;
```

Create a normal quantile-quantile plot. The QQPLOT statement creates a normal Q-Q plot for the analysis variable Normal_x. The NORMAL option superimposes a reference line that corresponds to the normal distribution by using estimated parameters for MU= and SIGMA=. CFRAME= specifies light gray as the background color inside the frame of the plot. PCTLAXIS adds a nonlinear percentile axis along the top of the Q-Q plot frame. GRID draws vertical grid lines by using the LGRID= linetype at the major percentiles. LABEL= specifies the label for the percentile axis.

```
qqplot normal_x /normal(mu=est sigma=est) cframe=ligr
  pctlaxis(grid lgrid=35 label='Normal Percentiles');
```

Add a table with statistical values on the plot. The INSET statement insets a table on the plot. The keywords MEAN and STD request that the mean and standard deviation be displayed. CFILL= specifies the background color for the table. FORMAT= specifies to use a format of field width 3. HEADER= displays a heading at the top of the inset. POSITION= specifies to use axis percentage coordinates to position the inset. REFPOINT= specifies to place the bottom-right corner of the inset 95% of the way across the horizontal axis and 10% of the way up the vertical axis.

* For additional information about the GOPTIONS statement, see *SAS/GRAPH Reference*.

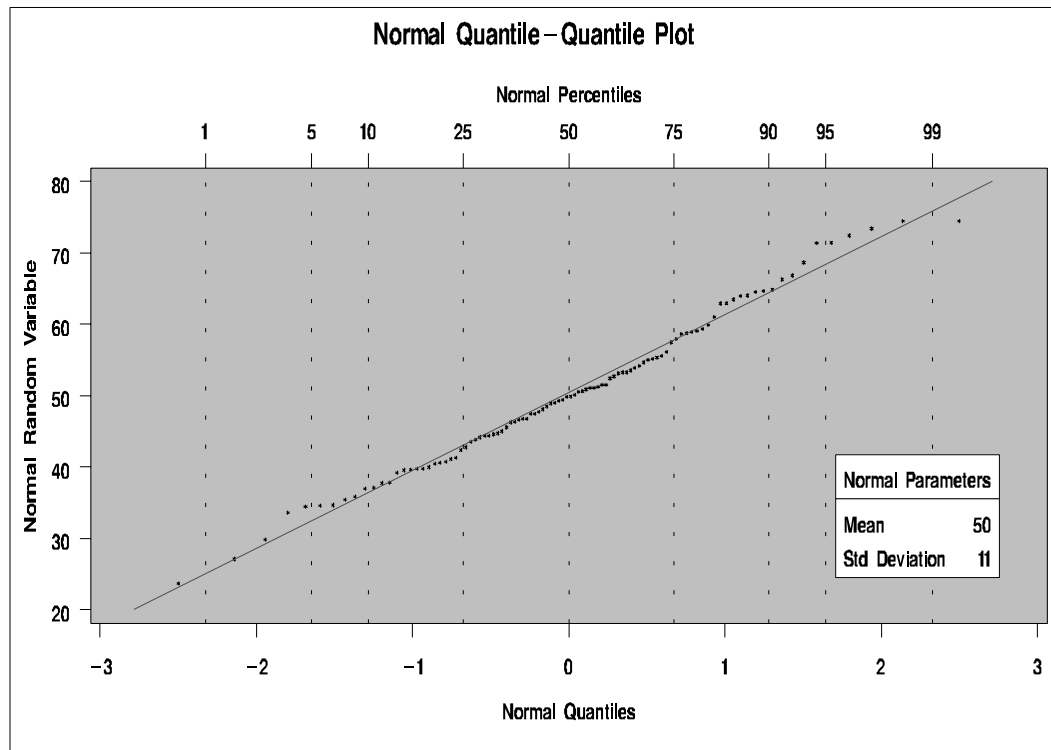
```
inset mean std / cfill=white format=3.0 header='Normal Parameters'
position=(95,10) refpoint=br;
```

Specify the title.

```
title1 'Normal Quantile-Quantile Plot';
run;
```

Output

Figure 48.9 Normal Quantile-Quantile Plot with a Normal Reference Line and a Customized Inset



Example 9: Creating a Two-Way Comparative Histogram

Procedure features:

PROC UNIVARIATE statement options:

NO PRINT

CLASS statement options:

ORDER=

HISTOGRAM statement options:

CFILL=

INTERTILE=

MIDPOINTS=

NCOLS=

NROWS=
 VAXIS=
 VAXISLABEL=
 VSCALE=

INSET statement options:

FONT=
 HEIGHT=
 NOFRAME
 POSITION=
statistical-keyword

VAR statement

Other features:

FORMAT statement
 FORMAT procedure
 GOPTIONS statement
 SORT procedure

Data set: STATEPOP on page 1544

- ☐ creates a data set with observations that are separated by census year
- ☐ sorts the data set by geographic region and census year
- ☐ suppresses the tables of descriptive statistics
- ☐ specifies two classification variables
- ☐ specifies the order of the component histograms
- ☐ creates a two-way comparative histogram with a specified number of rows and columns
- ☐
- ☐ specifies the distance between the component histogram tiles
- ☐ specifies the scale, values, and labels of the vertical axis
- ☐ specifies the midpoints for histogram intervals
- ☐ enhances the component histograms by inseting a table of summary statistics
- ☐ requests graphical enhancements that change fill color and font types.

Program

Set the graphics environment. The GOPTIONS statement sets the graphics environment to control the appearance of graphic elements. HTITLE= and HTEXT= specify the text height in GUNIT= units. FTEXT= and FTITLE= specify the font.*

```
goptions htitle=4 htext=3 gunit=pct ftext=swiss fttitle=swiss;
```

* For additional information about the GOPTIONS statement, see *SAS/GRAPH Reference*.

Assign a character string format to a numeric value. PROC FORMAT creates a format to identify regions with a character value.

```
proc format;
    value Regnfmt 1='Northeast'
                  2='South'
                  3='Midwest'
                  4='West';
run;
```

Create the METROPOP data set. This data set contains one variable, Populationcount, with the metropolitan and nonmetropolitan population counts. YEAR indicates the year for the observation. The OUTPUT statements create two observations for each state and year combination.

```
data metropop;
    set statepop;
    keep Region Year Populationcount;
    label PopulationCount='US Population (millions)'
           Year='Count year';
    year=1990;
    populationcount=sum(citypop_1990,noncitypop_1990);
    output;
    year=2000;
    populationcount=sum(citypop_2000,noncitypop_2000);
    output;
```

Sort the data set. PROC SORT sorts observations in the METROPOP data set by Region and Year.

```
proc sort data=metropop;
    by region year;
run;
```

Suppress the printing of the statistics tables for the analysis variable. NOPRINT suppresses the tables of statistics that the PROC UNIVARIATE statement creates. The VAR statement specifies that PopulationCount is the analysis variable.

```
proc univariate data=metropop noprint;
    var populationcount;
```

Specify the variables to categorize the data. The CLASS statement specifies Region and Year as the classification variables. PROC UNIVARIATE produces a component histogram for each level (distinct combination of values) of these variables. ORDER= orders the classification levels by the frequency of Year so that the year with the greatest population count is displayed first.

```
class region year(order=freq);
```

Create a two-way comparative histogram. The HISTOGRAM statement creates a two-way comparative histogram for the analysis variable PopulationCount. NROWS= and NCOLS= specify a 4 \times 2 arrangement for the tiles. INTERTILE= inserts a space of one percentage screen unit between the tiles. CFILL= specifies a fill color for the histogram bars. VSCALE= requests the vertical axis scale in units of the number of observations per data unit. VAXIS= specifies the tick mark labels and VAXISLABEL= specifies a label for the vertical axis. MIDPOINTS= specifies a list of values to use as bin midpoints. FONT= requests a software font for the text.

```
histogram /nrows=4 ncols=2 intertile=1 cfill=cyan vscale=count
          vaxis=0 4 8 12 vaxislabel='No. of States'
          midpoints=0 to 35 by 5;
```

Add a table with statistical values on the histogram. The INSET statement insets a table directly on each component histogram with the sum of PopulationCount. SUM= requests a customized label and a field width of five and two decimal places for the sum statistic. NOFRAME suppresses the frame around the inset table. POSITION= specifies to use a compass point to position the inset. HEIGHT= specifies the height of the text. FONT= requests a software font for the text.

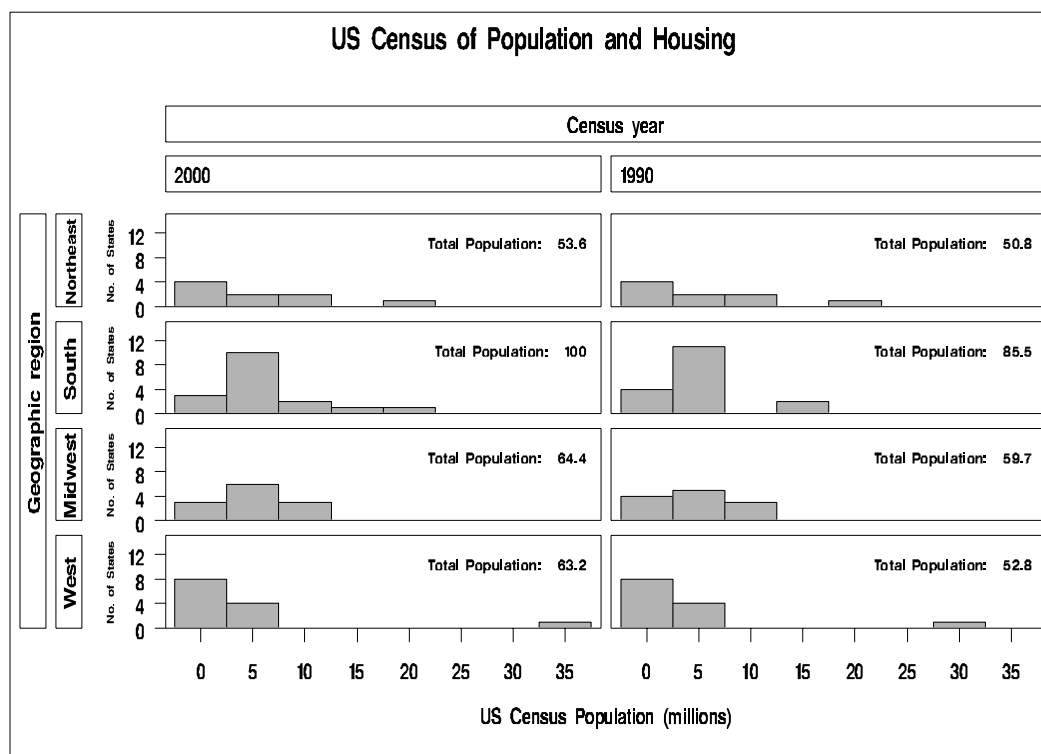
```
inset sum='Total Population:' (4.1) / noframe position=ne
          height=2 font=swissxb;
```

Assign a format to a variable and a title to graph. The FORMAT statement assigns a format to Region. The TITLE statement specifies a title.

```
format region regnfmt.;
title 'US Census of Population and Housing';
run;
```

Output

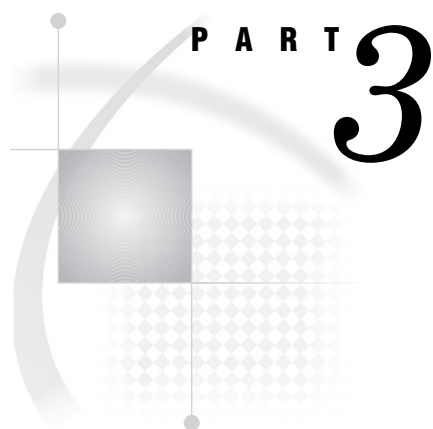
Figure 48.10 Two-way Comparative Histogram



References

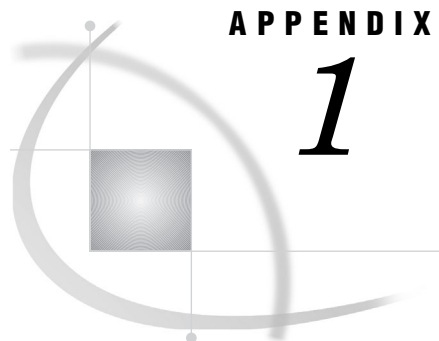
- Blom, G. (1958), *Statistical Estimates and Transformed Beta Variables*, New York: John Wiley & Sons, Inc.
- Chambers, J. M., Cleveland, W. S., Kleiner, B., and Tukey, P. A. (1983), *Graphical Methods for Data Analysis*, Pacific Grove, Wadsworth International Group.
- Conover, W.J. (1998), *Practical Nonparametric Statistics, Third Edition*, New York: John Wiley & Sons, Inc.
- Croux, C. and Rousseeuw, P.J. (1992), "Time-Efficient Algorithms for Two Highly Robust Estimators of Scale," *Computational Statistics*, Volume 1, 411-428.
- D'Agostino, R.B. and Stephens, M.A. (1986), *Goodness-of-Fit Techniques*, New York: Marcel Dekker, Inc.
- David, H.A. (1981), *Order Statistics, Second Edition*, New York: John Wiley & Sons, Inc.
- Dixon, W.J. and Tukey, J.W. (1968), "Approximate Behavior of the Distribution of Winsorized t (Trimming/Winsorization 2)," *Technometrics*, 10, 83-98.
- Frigge, M., Hoaglin, D.C., and Iglewicz, B. (1989), "Some Implementations of the Boxplot," *The American Statistician*, 43:1, 50-54.
- Friendly, M. (1991) *SAS System for Statistical Graphics, First Edition*, Cary, NC: SAS Institute Inc.

- Hahn, G.J. and Meeker, W. Q. (1991) *Statistical Intervals: A Guide for Practitioners*, New York: John Wiley & Sons, Inc.
- Hampel, F.R. (1974), "The Influence Curve and Its Role in Robust Estimation," *Journal of the American Statistical Association*, 69, 383-393.
- Iman, R.L. (1974), "Use of a t-statistic as an Approximation to the Exact Distribution of the Wilcoxon Signed Ranks Test Statistic," *Communications in Statistics*, 3, 795-806.
- Johnson, N.L., Kotz, S., and Balakrishnan, N. (1994), *Continuous Univariate Distributions, Volume 1*, New York: John Wiley & Sons, Inc.
- Johnson, N.L., Kotz, S., and Balakrishnan, N. (1995), *Continuous Univariate Distributions, Volume 2*, New York: John Wiley & Sons, Inc.
- Lehman, E.L. (1998), *Nonparametrics: Statistical Methods Based on Ranks*, New Jersey: Prentice Hall .
- Mood, A.M., Graybill, F.A., and Boes, D.C. (1974), *Introduction to the Theory of Statistics, Third Edition*, New York: McGraw-Hill.
- Odeh, R.E. and Owen, D.B. (1980), *Tables for Normal Tolerance Limits, Sampling Plans, and Screening*, New York: Marcel Dekker, Inc.
- Owen, D.B. and Hua, T.A. (1977), "Tables of Confidence Limits on the Tail Area of the Normal Distribution," *Communication and Statistics, Part B — Simulation and Computation*, 6, 285-311.
- Parzen, E. (1979), "Nonparametric Statistical Data Modeling," *Journal of the American Statistical Association*, 74, 105-121.
- Rousseeuw, P.J. and Croux, C. (1993), "Alternatives to the Median Absolute Deviation," *Journal of the American Statistical Association*, 88, 1273-1283.
- Royston, J.P. (1992), "Approximating the Shapiro-Wilk's W-Test for Non-normality," *Statistics and Computing*, 2, 117-119.
- Royston, J.P. (1982), "An Extension of Shapiro and Wilk's W Test for Normality to Large Samples," *Applied Statistics*, 31, 115-124.
- Shapiro, S.S. and Wilk, M.B. (1965), "An Analysis of Variance Test for Normality (complete samples)," *Biometrika*, 52, 591-611.
- Schlotzhauer, S.D. and Littell, R.C. (1997) *SAS System for Elementary Statistical Analysis, Second Edition*, Cary, NC: SAS Institute Inc.
- Silverman, B.W. (1986), *Density Estimation for Statistics and Data Analysis*, New York: Chapman and Hall.
- Sprent, P. (2000), *Applied Nonparametric Statistical Methods, Third Edition*, New York: Chapman and Hall.
- Stephens, M.A. (1974), "EDF Statistics for Goodness of Fit and Some Comparisons," *Journal of the American Statistical Association*, 69, 730-737.
- Terrell, G.R. and Scott, D.W. (1985), "Oversmoothed Nonparametric Density Estimates," *Journal of the American Statistical Association*, 80, 209-214.
- Tukey, J.W. (1977), *Exploratory Data Analysis*, Reading, Massachusetts: Addison-Wesley.
- Tukey, J.W. and McLaughlin, D.H. (1963), "Less Vulnerable Confidence and Significance Procedures for Location Based on a Single Sample: Trimming/Winsorization 1," *Sankhya A*, 25, 331-352.
- U.S. Bureau of the Census (2000), *Statistical Abstract of the United States: 2000*, Washington, D.C.: U.S. Government Printing Office.



Appendices

<i>Appendix 1</i>	SAS Elementary Statistics Procedures	1577
<i>Appendix 2</i>	Operating Environment-Specific Procedures	1613
<i>Appendix 3</i>	Raw Data and DATA Steps	1615
<i>Appendix 4</i>	Recommended Reading	1673



APPENDIX

1

SAS Elementary Statistics Procedures

<i>Overview</i>	1577
<i>Keywords and Formulas</i>	1578
<i>Descriptive Statistics</i>	1580
<i>Percentile and Related Statistics</i>	1583
<i>Hypothesis Testing Statistics</i>	1585
<i>Confidence Limits for the Mean</i>	1585
<i>Using Weights</i>	1586
<i>Data Requirements for Summarization Procedures</i>	1586
<i>Statistical Background</i>	1586
<i>Populations and Parameters</i>	1586
<i>Samples and Statistics</i>	1587
<i>Measures of Location</i>	1588
<i>The Mean</i>	1588
<i>The Median</i>	1588
<i>The Mode</i>	1588
<i>Percentiles</i>	1588
<i>Measures of Variability</i>	1592
<i>The Range</i>	1592
<i>The Interquartile Range</i>	1593
<i>The Variance</i>	1593
<i>The Standard Deviation</i>	1593
<i>Coefficient of Variation</i>	1593
<i>Measures of Shape</i>	1593
<i>Skewness</i>	1593
<i>Kurtosis</i>	1594
<i>The Normal Distribution</i>	1594
<i>Sampling Distribution of the Mean</i>	1597
<i>Testing Hypotheses</i>	1607
<i>Significance and Power</i>	1608
<i>Student's <i>t</i> Distribution</i>	1609
<i>Probability Values</i>	1610
<i>References</i>	1611

Overview

This appendix provides a brief description of some of the statistical concepts necessary for you to interpret the output of base SAS procedures for elementary statistics. In addition, this appendix lists statistical notation, formulas, and standard keywords used for common statistics in base SAS procedures. Brief examples illustrate the statistical concepts.

Table A1.1 on page 1579 lists the most common statistics and the procedures that compute them.

Keywords and Formulas

The base SAS procedures use a standardized set of keywords to refer to statistics. You specify these keywords in SAS statements to request the statistics to be displayed or stored in an output data set.

In the following notation, summation is over observations that contain nonmissing values of the analyzed variable and, except where shown, over nonmissing weights and frequencies of one or more:

x_i
is the nonmissing value of the analyzed variable for observation i .

f_i
is the frequency that is associated with x_i if you use a FREQ statement. If you omit the FREQ statement, then $f_i = 1$ for all i .

w_i
is the weight that is associated with x_i if you use a WEIGHT statement. The base procedures automatically exclude the values of x_i with missing weights from the analysis.

By default, the base procedures treat a negative weight as if it is equal to zero. However, if you use the EXCLNPWGT option in the PROC statement, then the procedure also excludes those values of x_i with nonpositive weights. Note that most SAS/STAT procedures, such as PROC TTEST and PROC GLM, exclude values with nonpositive weights by default.

If you omit the WEIGHT statement, then $w_i = 1$ for all i .

n
is the number of nonmissing values of x_i , $\sum f_i$. If you use the EXCLNPWGT option and the WEIGHT statement, then n is the number of nonmissing values with positive weights.

\bar{x}
is the mean

$$\sum w_i x_i / \sum w_i$$

s^2
is the variance

$$\frac{1}{d} \sum w_i (x_i - \bar{x})^2$$

where d is the variance divisor (the VARDEF= option) that you specify in the PROC statement. Valid values are as follows:

When VARDEF=	d equals . . .
N	n
DF	$n - 1$
WEIGHT	$\sum w_i$
WDF	$\sum w_i - 1$

The default is DF.

z_i
is the standardized variable

$$(x_i - \bar{x}) / s$$

The standard keywords and formulas for each statistic follow. Some formulas use keywords to designate the corresponding statistic.

Table A1.1 The Most Common Simple Statistics

Statistic	PROC MEANS and SUMMARY	PROC UNIVARIATE	PROC TABULATE	PROC REPORT	PROC CORR	PROC SQL
Number of missing values	X	X	X	X		X
Number of nonmissing values	X	X	X	X	X	X
Number of observations	X	X				X
Sum of weights	X	X	X	X	X	X
Mean	X	X	X	X	X	X
Sum	X	X	X	X	X	X
Extreme values	X	X				
Minimum	X	X	X	X	X	X
Maximum	X	X	X	X	X	X
Range	X	X	X	X		X
Uncorrected sum of squares	X	X	X	X	X	X
Corrected sum of squares	X	X	X	X	X	X
Variance	X	X	X	X	X	X
Covariance					X	
Standard deviation	X	X	X	X	X	X

Statistic	PROC MEANS and SUMMARY	PROC UNIVARIATE	PROC TABULATE	PROC REPORT	PROC CORR	PROC SQL
Standard error of the mean	X	X	X	X		X
Coefficient of variation	X	X	X	X		X
Skewness	X	X	X			
Kurtosis	X	X	X			
Confidence Limits						
of the mean	X	X	X			
of the variance		X				
of quantiles		X				
Median	X	X	X	X	X	
Mode		X				
Percentiles/Deciles/ Quartiles	X	X	X	X		
t test						
for mean= μ_0	X	X	X	X		X
for mean= μ_0		X				
Nonparametric tests for location		X				
Tests for normality		X				
Correlation coefficients					X	
Cronbach's alpha					X	

Descriptive Statistics

The keywords for descriptive statistics are

CSS

is the sum of squares corrected for the mean, computed as

$$\sum w_i (x_i - \bar{x})^2$$

CV

is the percent coefficient of variation, computed as

$$(100s) / \bar{x}$$

KURTOSIS | KURT

is the kurtosis, which measures heaviness of tails. When VARDEF=DF, the kurtosis is computed as

$$c_{4_n} \sum z_i^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$$

where c_{4_n} is $\frac{n(n+1)}{(n-1)(n-2)(n-3)}$. The weighted kurtosis is computed as

$$\begin{aligned} &= c_{4_n} \sum ((x_i - \bar{x}) / \hat{\sigma}_i)^4 - \frac{3(n-1)^2}{(n-2)(n-3)} \\ &= c_{4_n} \sum w_i^2 ((x_i - \bar{x}) / \hat{\sigma})^4 - \frac{3(n-1)^2}{(n-2)(n-3)} \end{aligned}$$

When VARDEF=N, the kurtosis is computed as

$$= \frac{1}{n} \sum z_i^4 - 3$$

and the weighted kurtosis is computed as

$$\begin{aligned} &= \frac{1}{n} \sum ((x_i - \bar{x}) / \hat{\sigma}_i)^4 - 3 \\ &= \frac{1}{n} \sum w_i^2 ((x_i - \bar{x}) / \hat{\sigma})^4 - 3 \end{aligned}$$

where σ_i^2 is σ^2 / w_i . The formula is invariant under the transformation $w_i^* = zw_i$, $z > 0$. When you use VARDEF=WDF or VARDEF=WEIGHT, the kurtosis is set to missing.

Note: PROC MEANS and PROC TABULATE do not compute weighted kurtosis. Δ

MAX

is the maximum value of x_i .

MEAN

is the arithmetic mean \bar{x} .

MIN

is the minimum value of x_i .

MODE

is the most frequent value of x_i .

N

is the number of x_i values that are not missing. Observations with f_i less than one and w_i equal to missing or $w_i \leq 0$ (when you use the EXCLNPWGT option) are excluded from the analysis and are not included in the calculation of N.

NMISS

is the number of x_i values that are missing. Observations with f_i less than one and w_i equal to missing or $w_i \leq 0$ (when you use the EXCLNPWGT option) are excluded from the analysis and are not included in the calculation of NMISS.

NOBS

is the total number of observations and is calculated as the sum of N and NMISS. However, if you use the WEIGHT statement, then NOBS is calculated as the sum of N, NMISS, and the number of observations excluded because of missing or nonpositive weights.

RANGE

is the range and is calculated as the difference between maximum value and minimum value.

SKEWNESS | SKEW

is skewness, which measures the tendency of the deviations to be larger in one direction than in the other. When VARDEF=DF, the skewness is computed as

$$c_{3_n} \sum z_i^3$$

where c_{3_n} is $\frac{n}{(n-1)(n-2)}$. The weighted skewness is computed as

$$\begin{aligned} &= c_{3_n} \sum ((x_i - \bar{x}) / \hat{\sigma}_j)^3 \\ &= c_{3_n} \sum w_i^{3/2} ((x_i - \bar{x}) / \hat{\sigma})^3 \end{aligned}$$

When VARDEF=N, the skewness is computed as

$$= \frac{1}{n} \sum z_i^3$$

and the weighted skewness is computed as

$$\begin{aligned} &= \frac{1}{n} \sum ((x_i - \bar{x}) / \hat{\sigma}_j)^3 \\ &= \frac{1}{n} \sum w_i^{3/2} ((x_i - \bar{x}) / \hat{\sigma})^3 \end{aligned}$$

The formula is invariant under the transformation $w_i^* = zw_i$, $z > 0$. When you use VARDEF=WDF or VARDEF=WEIGHT, the skewness is set to missing.

Note: PROC MEANS and PROC TABULATE do not compute weighted skewness. \triangle

STDDEV | STD

is the standard deviation s and is computed as the square root of the variance, s^2 .

STDERR | STDMEAN

is the standard error of the mean, computed as

$$s / \sqrt{\sum w_i}$$

when VARDEF=DF, which is the default. Otherwise, STDERR is set to missing.

SUM

is the sum, computed as

$$\sum w_i x_i$$

SUMWGTis the sum of the weights, W , computed as

$$\sum w_i$$

USS

is the uncorrected sum of squares, computed as

$$\sum w_i x_i^2$$

VARis the variance s^2 .

Percentile and Related Statistics

The keywords for percentiles and related statistics are

MEDIAN

is the middle value.

P1is the 1st percentile.**P5**is the 5th percentile.**P10**is the 10th percentile.**P90**is the 90th percentile.**P95**is the 95th percentile.**P99**is the 99th percentile.**Q1**is the lower quartile (25th percentile).**Q3**is the upper quartile (75th percentile).**QRANGE**

is interquartile range and is calculated as

$$Q_3 - Q_1$$

You use the PCTLDEF= option to specify the method that the procedure uses to compute percentiles. Let n be the number of nonmissing values for a variable, and let x_1, x_2, \dots, x_n represent the ordered values of the variable such that x_1 is the smallest value, x_2 is next smallest value, and x_n is the largest value. For the t th percentile between 0 and 1, let $p = t/100$. Then define j as the integer part of np and g as the fractional part of np or $(n+1)p$, so that

$$\begin{aligned} np &= j + g && \text{when PCTLDEF} = 1, 2, 3, \text{ or } 5 \\ (n+1)p &= j + g && \text{when PCTLDEF} = 4 \end{aligned}$$

Here, PCTLDEF= specifies the method that the procedure uses to compute the t th percentile, as shown in the table that follows.

When you use the WEIGHT statement, the t th percentile is computed as

$$y = \begin{cases} \frac{1}{2}(x_i + x_{i+1}) & \text{if } \sum_{j=1}^i w_j = pW \\ x_{i+1} & \text{if } \sum_{j=1}^i w_j < pW < \sum_{j=1}^{i+1} w_j \end{cases}$$

where w_j is the weight associated with x_i and $W = \sum_{i=1}^n w_i$ is the sum of the weights.

When the observations have identical weights, the weighted percentiles where the same as the unweighted percentiles with PCTLDEF=5.

Table A1.2 Methods for Computing Percentile Statistics

PCTLDEF=	Description	Formula
1	weighted average at x_{np}	$y = (1 - g)x_j + gx_{j+1}$ where x_0 is taken to be x_1
2	observation numbered closest to np	$y = x_i$ if $g \neq \frac{1}{2}$ $y = x_j$ if $g = \frac{1}{2}$ and j is even $y = x_{j+1}$ if $g = \frac{1}{2}$ and j is odd where i is the integer part of $np + \frac{1}{2}$
3	empirical distribution function	$y = x_j$ if $g = 0$ $y = x_{j+1}$ if $g > 0$
4	weighted average aimed at $x_{(n+1)p}$	$y = (1 - g)x_j + gx_{j+1}$ where x_{n+1} is taken to be x_n

PCTLDEF=	Description	Formula	
5	empirical distribution function with averaging	$y = \frac{1}{2} (x_j + x_{j+1})$	if $g = 0$
		$y = x_{j+1}$	if $g > 0$

Hypothesis Testing Statistics

The keywords for hypothesis testing statistics are

T

is the Student's t statistic to test the null hypothesis that the population mean is equal to μ_0 and is calculated as

$$\frac{\bar{x} - \mu_0}{s / \sqrt{\sum w_i}}$$

By default, μ_0 is equal to zero. You can use the MU0= option in the PROC UNIVARIATE statement to specify μ_0 . You must use VARDEF=DF, which is the default variance divisor, otherwise T is set to missing.

By default, when you use a WEIGHT statement, the procedure counts the x_i values with nonpositive weights in the degrees of freedom. Use the EXCLNPWGT option in the PROC statement to exclude values with nonpositive weights. Most SAS/STAT procedures, such as PROC TTEST and PROC GLM automatically exclude values with nonpositive weights.

PROBT

is the two-tailed p -value for Student's t statistic, T, with $n - 1$ degrees of freedom. This is the probability under the null hypothesis of obtaining a more extreme value of T than is observed in this sample.

Confidence Limits for the Mean

The keywords for confidence limits are

CLM

is the two-sided confidence limit for the mean. A two-sided $100(1 - \alpha)$ percent confidence interval for the mean has upper and lower limits

$$\bar{x} \pm t_{(1-\alpha/2; n-1)} \frac{s}{\sqrt{\sum w_i}}$$

where s is $\sqrt{\frac{1}{n-1} \sum (x_i - \bar{x})^2}$, $t_{(1-\alpha/2; n-1)}$ is the $(1 - \alpha/2)$ critical value of the Student's t statistics with $n - 1$ degrees of freedom, and α is the value of the ALPHA= option which by default is 0.05. Unless you use VARDEF=DF, which is the default variance divisor, CLM is set to missing.

LCLM

is the one-sided confidence limit below the mean. The one-sided $100(1 - \alpha)$ percent confidence interval for the mean has the lower limit

$$\bar{x} - t_{(1-\alpha;n-1)} \frac{s}{\sqrt{\sum w_i}}$$

Unless you use VARDEF=DF, which is the default variance divisor, LCLM is set to missing.

UCLM

is the one-sided confidence limit above the mean. The one-sided 100(1 - α) percent confidence interval for the mean has the upper limit

$$\bar{x} + t_{(1-\alpha;n-1)} \frac{s}{\sqrt{\sum w_i}}$$

Unless you use VARDEF=DF, which is the default variance divisor, UCLM is set to missing.

Using Weights

For more information on using weights and an example, see “WEIGHT” on page 59.

Data Requirements for Summarization Procedures

The following are the minimal data requirements to compute unweighted statistics and do not describe recommended sample sizes. Statistics are reported as missing if VARDEF=DF (the default) and these requirements are not met:

- ☐ N and NMISS are computed regardless of the number of missing or nonmissing observations.
- ☐ SUM, MEAN, MAX, MIN, RANGE, USS, and CSS require at least one nonmissing observation.
- ☐ VAR, STD, STDERR, CV, T, and PRT require at least two nonmissing observations.
- ☐ SKEWNESS requires at least three nonmissing observations.
- ☐ KURTOSIS requires at least four nonmissing observations.
- ☐ SKEWNESS, KURTOSIS, T, and PROBT require that STD is greater than zero.
- ☐ CV requires that MEAN is not equal to zero.
- ☐ CLM, LCLM, UCLM, STDERR, T, and PROBT require that VARDEF=DF.

Statistical Background

The rest of this appendix provides text descriptions and SAS code examples that explain some of the statistical concepts and terminology that you may encounter when you interpret the output of SAS procedures for elementary statistics. For a more thorough discussion, consult an introductory statistics textbook such as Mendenhall and Beaver (1998); Ott and Mendenhall (1994); or Snedecor and Cochran (1989).

Populations and Parameters

Usually, there is a clearly defined set of elements in which you are interested. This set of elements is called the *universe*, and a set of values associated with these elements

is called a *population* of values. The statistical term *population* has nothing to do with people per se. A statistical population is a collection of values, not a collection of people. For example, a universe is all the students at a particular school, and there could be two populations of interest: one of height values and one of weight values. Or, a universe is the set of all widgets manufactured by a particular company, while the population of values could be the length of time each widget is used before it fails.

A population of values can be described in terms of its *cumulative distribution function*, which gives the proportion of the population less than or equal to each possible value. A discrete population can also be described by a *probability function*, which gives the proportion of the population equal to each possible value. A continuous population can often be described by a *density function*, which is the derivative of the cumulative distribution function. A density function can be approximated by a histogram that gives the proportion of the population lying within each of a series of intervals of values. A probability density function is like a histogram with an infinite number of infinitely small intervals.

In technical literature, when the term *distribution* is used without qualification, it generally refers to the cumulative distribution function. In informal writing, *distribution* sometimes means the density function instead. Often the word *distribution* is used simply to refer to an abstract population of values rather than some concrete population. Thus, the statistical literature refers to many types of abstract distributions, such as normal distributions, exponential distributions, Cauchy distributions, and so on. When a phrase such as *normal distribution* is used, it frequently does not matter whether the cumulative distribution function or the density function is intended.

It may be expedient to describe a population in terms of a few measures that summarize interesting features of the distribution. One such measure, computed from the population values, is called a *parameter*. Many different parameters can be defined to measure different aspects of a distribution.

The most commonly used parameter is the (arithmetic) *mean*. If the population contains a finite number of values, then the population mean is computed as the sum of all the values in the population divided by the number of elements in the population. For an infinite population, the concept of the mean is similar but requires more complicated mathematics.

$E(x)$ denotes the mean of a population of values symbolized by x , such as height, where E stands for *expected value*. You can also consider expected values of derived functions of the original values. For example, if x represents height, then $E(x^2)$ is the expected value of height squared, that is, the mean value of the population obtained by squaring every value in the population of heights.

Samples and Statistics

It is often impossible to measure all of the values in a population. A collection of measured values is called a *sample*. A mathematical function of a sample of values is called a *statistic*. A statistic is to a sample as a parameter is to a population. It is customary to denote statistics by Roman letters and parameters by Greek letters. For example, the population mean is often written as μ , whereas the sample mean is written as \bar{x} . The field of *statistics* is largely concerned with the study of the behavior of sample statistics.

Samples can be selected in a variety of ways. Most SAS procedures assume that the data constitute a *simple random sample*, which means that the sample was selected in such a way that all possible samples were equally likely to be selected.

Statistics from a sample can be used to make inferences, or reasonable guesses, about the parameters of a population. For example, if you take a random sample of 30 students from the high school, then the mean height for those 30 students is a reasonable guess, or *estimate*, of the mean height of all the students in the high school.

Other statistics, such as the standard error, can provide information about how good an estimate is likely to be.

For any population parameter, several statistics can estimate it. Often, however, there is one particular statistic that is customarily used to estimate a given parameter. For example, the sample mean is the usual estimator of the population mean. In the case of the mean, the formulas for the parameter and the statistic are the same. In other cases, the formula for a parameter may be different from that of the most commonly used estimator. The most commonly used estimator is not necessarily the best estimator in all applications.

Measures of Location

Measures of location include the mean, the median, and the mode. These measures describe the center of a distribution. In the definitions that follow, notice that if the entire sample changes by adding a fixed amount to each observation, then these measures of location are shifted by the same fixed amount.

The Mean

The population mean $\mu = E(x)$ is usually estimated by the sample mean \bar{x} .

The Median

The population median is the central value, lying above and below half of the population values. The sample median is the middle value when the data are arranged in ascending or descending order. For an even number of observations, the midpoint between the two middle values is usually reported as the median.

The Mode

The mode is the value at which the density of the population is at a maximum. Some densities have more than one local maximum (peak) and are said to be *multimodal*. The sample mode is the value that occurs most often in the sample. By default, PROC UNIVARIATE reports the lowest such value if there is a tie for the most-often-occurring sample value. PROC UNIVARIATE lists all possible modes when you specify the MODES option in the PROC statement. If the population is continuous, then all sample values occur once, and the sample mode has little use.

Percentiles

Percentiles, including quantiles, quartiles, and the median, are useful for a detailed study of a distribution. For a set of measurements arranged in order of magnitude, the p th percentile is the value that has p percent of the measurements below it and $(100-p)$ percent above it. The median is the 50th percentile. Because it may not be possible to divide your data so that you get exactly the desired percentile, the UNIVARIATE procedure uses a more precise definition.

The upper quartile of a distribution is the value below which 75 percent of the measurements fall (the 75th percentile). Twenty-five percent of the measurements fall below the lower quartile value.

In the following example, SAS artificially generates the data with a pseudorandom number function. The UNIVARIATE procedure computes a variety of quantiles and measures of location, and outputs the values to a SAS data set. A DATA step then uses the SYMPUT routine to assign the values of the statistics to macro variables. The

macro %FORMGEN uses these macro variables to produce value labels for the FORMAT procedure. PROC CHART uses the resulting format to display the values of the statistics on a histogram.

```

options nodate pageno=1 linesize=80 pagesize=52;

title 'Example of Quantiles and Measures of Location';

data random;
  drop n;
  do n=1 to 1000;
    X=floor(exp(rannor(314159)*.8+1.8));
    output;
  end;
run;

proc univariate data=random nextrobs=0;
  var x;
  output out=location
    mean=Mean mode=Mode median=Median
    q1=Q1 q3=Q3 p5=P5 p10=P10 p90=P90 p95=P95
    max=Max;
run;

proc print data=location noobs;
run;

data _null_;
  set location;
  call symput('MEAN',round(mean,1));
  call symput('MODE',mode);
  call symput('MEDIAN',round(median,1));
  call symput('Q1',round(q1,1));
  call symput('Q3',round(q3,1));
  call symput('P5',round(p5,1));
  call symput('P10',round(p10,1));
  call symput('P90',round(p90,1));
  call symput('P95',round(p95,1));
  call symput('MAX',min(50,max));
run;

%macro formgen;
%do i=1 %to &max;
  %let value=&i;
  %if &i=&p5      %then %let value=&value P5;
  %if &i=&p10     %then %let value=&value P10;
  %if &i=&q1      %then %let value=&value Q1;
  %if &i=&mode    %then %let value=&value Mode;
  %if &i=&median  %then %let value=&value Median;
  %if &i=&mean    %then %let value=&value Mean;
  %if &i=&q3      %then %let value=&value Q3;
  %if &i=&p90     %then %let value=&value P90;
  %if &i=&p95     %then %let value=&value P95;

```

```

        %if &i=&max      %then %let value=>=&value;
        &i="&value"
    %end;
%mend;

proc format print;
    value stat %formgen;
run;
options pagesize=42 linesize=80;

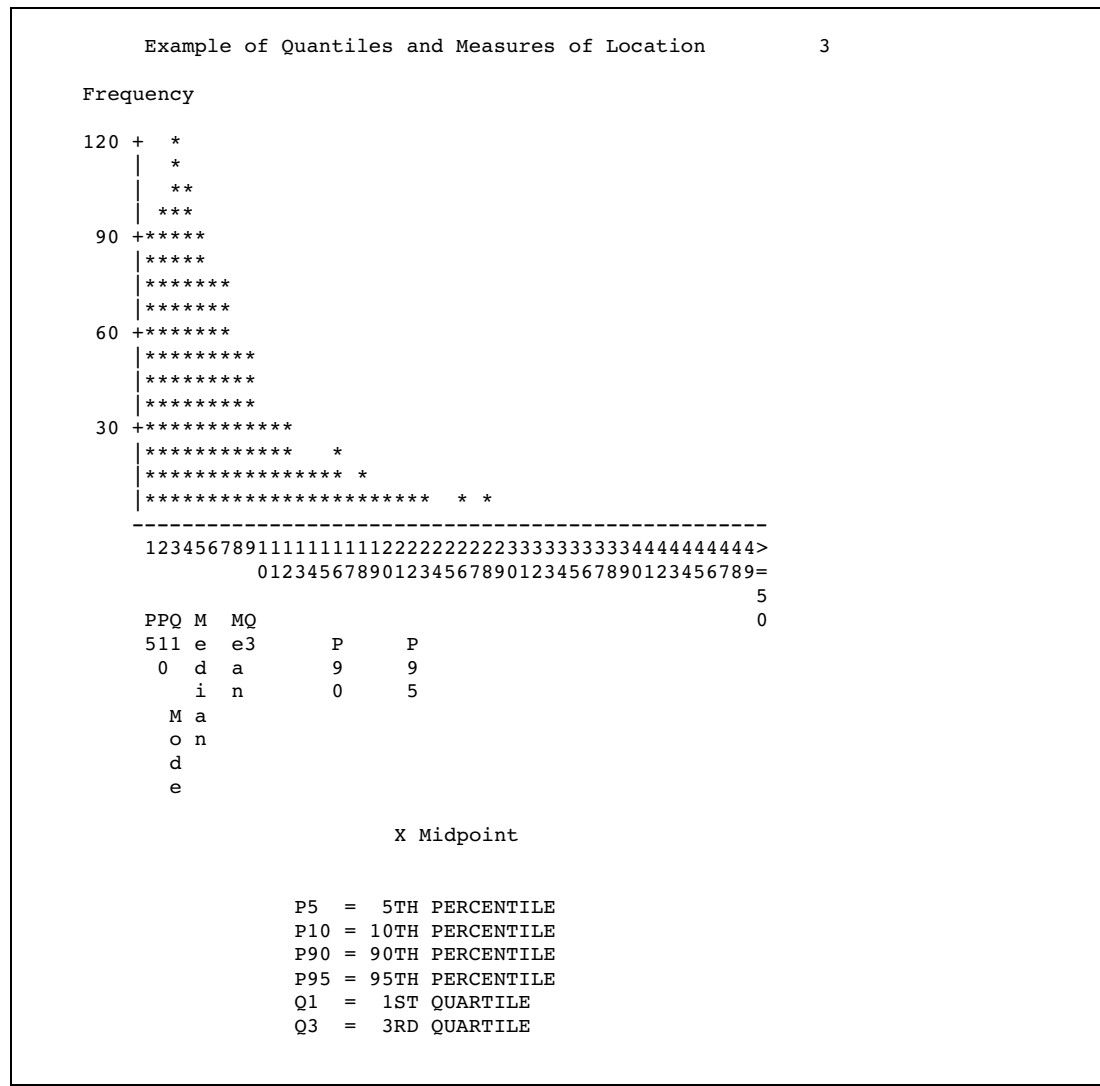
proc chart data=random;
    vbar x / midpoints=1 to &max by 1;
    format x stat.;
    footnote 'P5 = 5TH PERCENTILE';
    footnote2 'P10 = 10TH PERCENTILE';
    footnote3 'P90 = 90TH PERCENTILE';
    footnote4 'P95 = 95TH PERCENTILE';
    footnote5 'Q1 = 1ST QUARTILE  ';
    footnote6 'Q3 = 3RD QUARTILE  ';

```


run;

Example of Quantiles and Measures of Location				1
The UNIVARIATE Procedure				
Variable: X				
Moments				
N	1000	Sum Weights	1000	
Mean	7.605	Sum Observations	7605	
Std Deviation	7.38169794	Variance	54.4894645	
Skewness	2.73038523	Kurtosis	11.1870588	
Uncorrected SS	112271	Corrected SS	54434.975	
Coeff Variation	97.0637467	Std Error Mean	0.23342978	
Basic Statistical Measures				
Location		Variability		
Mean	7.605000	Std Deviation	7.38170	
Median	5.000000	Variance	54.48946	
Mode	3.000000	Range	62.00000	
		Interquartile Range	6.00000	
Tests for Location: Mu0=0				
Test	-Statistic-	-----p Value-----		
Student's t	t 32.57939	Pr > t	<.0001	
Sign	M 494.5	Pr >= M	<.0001	
Signed Rank	S 244777.5	Pr >= S	<.0001	
Quantiles (Definition 5)				
Quantile		Estimate		
100% Max		62.0		
99%		37.5		
95%		21.5		
90%		16.0		
75% Q3		9.0		
50% Median		5.0		
25% Q1		3.0		
10%		2.0		
5%		1.0		
1%		0.0		
0% Min		0.0		

Example of Quantiles and Measures of Location										2
Mean	Max	P95	P90	Q3	Median	Q1	P10	P5	Mode	
7.605	62	21.5	16	9	5	3	2	1	3	



Measures of Variability

Another group of statistics is important in studying the distribution of a population. These statistics measure the *variability*, also called the spread, of values. In the definitions given in the sections that follow, notice that if the entire sample is changed by the addition of a fixed amount to each observation, then the values of these statistics are unchanged. If each observation in the sample is multiplied by a constant, however, then the values of these statistics are appropriately rescaled.

The Range

The sample range is the difference between the largest and smallest values in the sample. For many populations, at least in statistical theory, the range is infinite, so the sample range may not tell you much about the population. The sample range tends to increase as the sample size increases. If all sample values are multiplied by a constant, then the sample range is multiplied by the same constant.

The Interquartile Range

The interquartile range is the difference between the upper and lower quartiles. If all sample values are multiplied by a constant, then the sample interquartile range is multiplied by the same constant.

The Variance

The population variance, usually denoted by σ^2 , is the expected value of the squared difference of the values from the population mean:

$$\sigma^2 = E(x - \mu)^2$$

The sample variance is denoted by s^2 . The difference between a value and the mean is called a *deviation from the mean*. Thus, the variance approximates the mean of the squared deviations.

When all the values lie close to the mean, the variance is small but never less than zero. When values are more scattered, the variance is larger. If all sample values are multiplied by a constant, then the sample variance is multiplied by the square of the constant.

Sometimes values other than $n - 1$ are used in the denominator. The VARDEF= option controls what divisor the procedure uses.

The Standard Deviation

The standard deviation is the square root of the variance, or root-mean-square deviation from the mean, in either a population or a sample. The usual symbols are σ for the population and s for a sample. The standard deviation is expressed in the same units as the observations, rather than in squared units. If all sample values are multiplied by a constant, then the sample standard deviation is multiplied by the same constant.

Coefficient of Variation

The coefficient of variation is a unitless measure of relative variability. It is defined as the ratio of the standard deviation to the mean expressed as a percentage. The coefficient of variation is meaningful only if the variable is measured on a ratio scale. If all sample values are multiplied by a constant, then the sample coefficient of variation remains unchanged.

Measures of Shape

Skewness

The variance is a measure of the overall size of the deviations from the mean. Since the formula for the variance squares the deviations, both positive and negative deviations contribute to the variance in the same way. In many distributions, positive deviations may tend to be larger in magnitude than negative deviations, or vice versa. *Skewness* is a measure of the tendency of the deviations to be larger in one direction than in the other. For example, the data in the last example are skewed to the right.

Population skewness is defined as

$$E(x - \mu)^3 / \sigma^3$$

Because the deviations are cubed rather than squared, the signs of the deviations are maintained. Cubing the deviations also emphasizes the effects of large deviations. The formula includes a divisor of σ^3 to remove the effect of scale, so multiplying all values by a constant does not change the skewness. Skewness can thus be interpreted as a tendency for one tail of the population to be heavier than the other. Skewness can be positive or negative and is unbounded.

Kurtosis

The heaviness of the tails of a distribution affects the behavior of many statistics. Hence it is useful to have a measure of tail heaviness. One such measure is *kurtosis*. The population kurtosis is usually defined as

$$\frac{E(x - \mu)^4}{\sigma^4} - 3$$

Note: Some statisticians omit the subtraction of 3. \triangle

Because the deviations are raised to the fourth power, positive and negative deviations make the same contribution, while large deviations are strongly emphasized. Because of the divisor σ^4 , multiplying each value by a constant has no effect on kurtosis.

Population kurtosis must lie between -2 and $+\infty$, inclusive. If M_3 represents population skewness and M_4 represents population kurtosis, then

$$M_4 > (M_3)^2 - 2$$

Statistical literature sometimes reports that kurtosis measures the *peakedness* of a density. However, heavy tails have much more influence on kurtosis than does the shape of the distribution near the mean (Kaplansky 1945; Ali 1974; Johnson, et al. 1980).

Sample skewness and kurtosis are rather unreliable estimators of the corresponding parameters in small samples. They are better estimators when your sample is very large. However, large values of skewness or kurtosis may merit attention even in small samples because such values indicate that statistical methods that are based on normality assumptions may be inappropriate.

The Normal Distribution

One especially important family of theoretical distributions is the *normal* or *Gaussian* distribution. A normal distribution is a smooth symmetric function often referred to as "bell-shaped." Its skewness and kurtosis are both zero. A normal distribution can be completely specified by only two parameters: the mean and the standard deviation. Approximately 68 percent of the values in a normal population are within one standard deviation of the population mean; approximately 95 percent of the values are within

two standard deviations of the mean; and about 99.7 percent are within three standard deviations. Use of the term *normal* to describe this particular kind of distribution does not imply that other kinds of distributions are necessarily abnormal or pathological.

Many statistical methods are designed under the assumption that the population being sampled is normally distributed. Nevertheless, most real-life populations do not have normal distributions. Before using any statistical method based on normality assumptions, you should consult the statistical literature to find out how sensitive the method is to nonnormality and, if necessary, check your sample for evidence of nonnormality.

In the following example, SAS generates a sample from a normal distribution with a mean of 50 and a standard deviation of 10. The UNIVARIATE procedure performs tests for location and normality. Because the data are from a normal distribution, all *p*-values from the tests for normality are greater than 0.15. The CHART procedure displays a histogram of the observations. The shape of the histogram is a belllike, normal density.

```
options nodate pageno=1 linesize=80 pagesize=52;

title '10000 Obs Sample from a Normal Distribution';
title2 'with Mean=50 and Standard Deviation=10';

data normaldat;
  drop n;
  do n=1 to 10000;
    X=10*rannor(53124)+50;
    output;
  end;
run;

proc univariate data=normaldat nextrobs=0 normal
               mu0=50 loccount;

  var x;
run;

proc format;
  picture msd
    20='20 3*Std' (noedit)
    30='30 2*Std' (noedit)
    40='40 1*Std' (noedit)
    50='50 Mean ' (noedit)
    60='60 1*Std' (noedit)
    70='70 2*Std' (noedit)
    80='80 3*Std' (noedit)
    other=' ';
run;
options linesize=80 pagesize=42;

proc chart;
  vbar x / midpoints=20 to 80 by 2;
  format x msd.;
run;
```

10000 Obs Sample from a Normal Distribution 1
with Mean=50 and Standard Deviation=10

The UNIVARIATE Procedure
Variable: X

Moments

N	10000	Sum Weights	10000
Mean	50.0323744	Sum Observations	500323.744
Std Deviation	9.92013874	Variance	98.4091525
Skewness	-0.019929	Kurtosis	-0.0163755
Uncorrected SS	26016378	Corrected SS	983993.116
Coeff Variation	19.8274395	Std Error Mean	0.09920139

Basic Statistical Measures

Location		Variability	
Mean	50.03237	Std Deviation	9.92014
Median	50.06492	Variance	98.40915
Mode	.	Range	76.51343
		Interquartile Range	13.28179

Tests for Location: Mu0=50

Test	-Statistic-	-----p Value-----
Student's t	t 0.32635	Pr > t 0.7442
Sign	M 26	Pr >= M 0.6101
Signed Rank	S 174063	Pr >= S 0.5466

Location Counts: Mu0=50.00

Count	Value
Num Obs > Mu0	5026
Num Obs ^= Mu0	10000
Num Obs < Mu0	4974

Tests for Normality

Test	--Statistic--	-----p Value-----
Kolmogorov-Smirnov	D 0.006595	Pr > D >0.1500
Cramer-von Mises	W-Sq 0.049963	Pr > W-Sq >0.2500
Anderson-Darling	A-Sq 0.371151	Pr > A-Sq >0.2500

```

10000 Obs Sample from a Normal Distribution          2
with Mean=50 and Standard Deviation=10

```

```

The UNIVARIATE Procedure
Variable:  X

```

```

Quantiles (Definition 5)

```

Quantile	Estimate
100% Max	90.2105
99%	72.6780
95%	66.2221
90%	62.6678
75% Q3	56.7280
50% Median	50.0649
25% Q1	43.4462
10%	37.1139
5%	33.5454
1%	26.9189
0% Min	13.6971

```

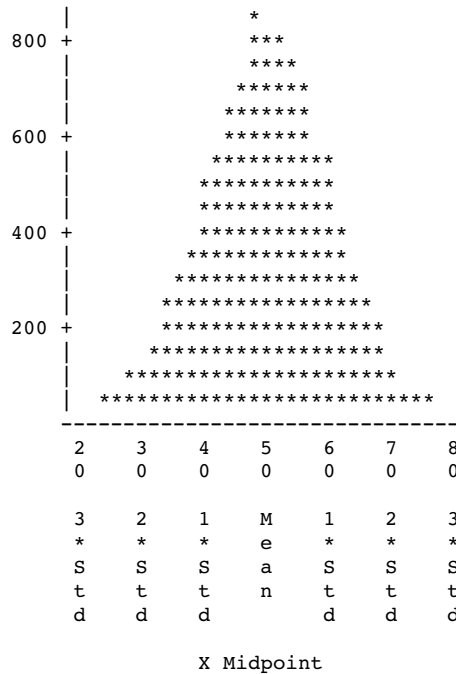
10000 Obs Sample from a Normal Distribution          3
with Mean=50 and Standard Deviation=10

```

```

Frequency

```



Sampling Distribution of the Mean

If you repeatedly draw samples of size n from a population and compute the mean of each sample, then the sample means themselves have a distribution. Consider a new population consisting of the means of all the samples that could possibly be drawn from the original population. The distribution of this new population is called a *sampling distribution*.

It can be proven mathematically that if the original population has mean μ and standard deviation σ , then the sampling distribution of the mean also has mean μ , but its standard deviation is σ/\sqrt{n} . The standard deviation of the sampling distribution of the mean is called the *standard error of the mean*. The standard error of the mean provides an indication of the accuracy of a sample mean as an estimator of the population mean.

If the original population has a normal distribution, then the sampling distribution of the mean is also normal. If the original distribution is not normal but does not have excessively long tails, then the sampling distribution of the mean can be approximated by a normal distribution for large sample sizes.

The following example consists of three separate programs that show how the sampling distribution of the mean can be approximated by a normal distribution as the sample size increases. The first DATA step uses the RANEXP function to create a sample of 1000 observations from an exponential distribution. The theoretical population mean is 1.00, while the sample mean is 1.01, to two decimal places. The population standard deviation is 1.00; the sample standard deviation is 1.04.

This is an example of a nonnormal distribution. The population skewness is 2.00, which is close to the sample skewness of 1.97. The population kurtosis is 6.00, but the sample kurtosis is only 4.80.

```
options nodate pageno=1 linesize=80 pagesize=42;

title '1000 Observation Sample';
title2 'from an Exponential Distribution';

data expodat;
  drop n;
  do n=1 to 1000;
    X=ranexp(18746363);
    output;
  end;
run;
proc format;
  value axisfmt
    .05='0.05'
    .55='0.55'
    1.05='1.05'
    1.55='1.55'
    2.05='2.05'
    2.55='2.55'
    3.05='3.05'
    3.55='3.55'
    4.05='4.05'
    4.55='4.55'
    5.05='5.05'
    5.55='5.55'
    other=' ';
run;

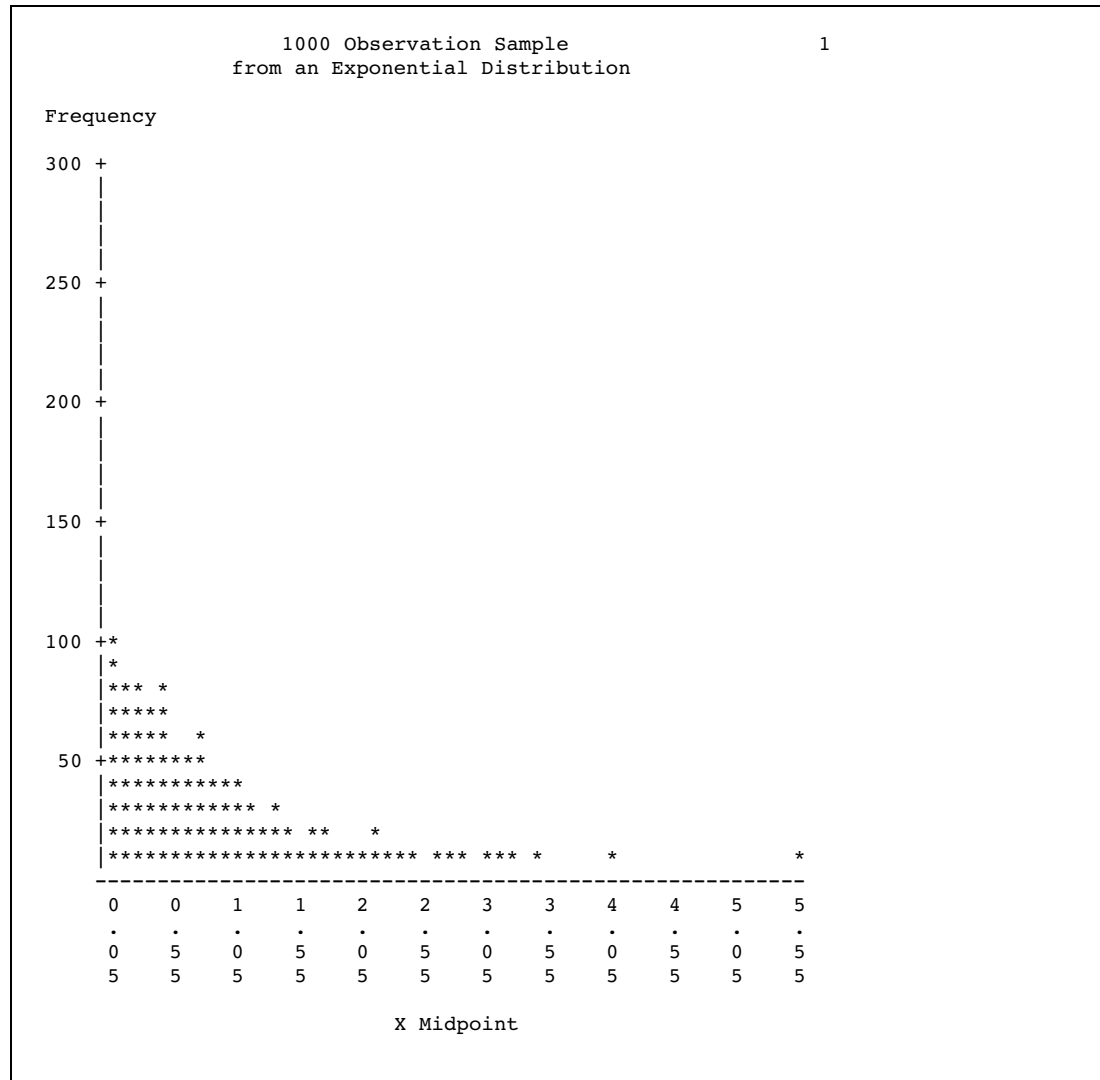
proc chart data=expodat ;
  vbar x / axis=300
    midpoints=0.05 to 5.55 by .1;
  format x axisfmt.;
run;
```



```
options pagesize=64;

proc univariate data=expodat noextrobs=0 normal
      mu0=1;
  var x;
```

run;



1000 Observation Sample			2
from an Exponential Distribution			
The UNIVARIATE Procedure			
Variable: X			
Moments			
N	1000	Sum Weights	1000
Mean	1.01176214	Sum Observations	1011.76214
Std Deviation	1.04371187	Variance	1.08933447
Skewness	1.96963112	Kurtosis	4.80150594
Uncorrected SS	2111.90777	Corrected SS	1088.24514
Coeff Variation	103.15783	Std Error Mean	0.03300507
Basic Statistical Measures			
Location		Variability	
Mean	1.011762	Std Deviation	1.04371
Median	0.689502	Variance	1.08933
Mode	.	Range	6.63851
		Interquartile Range	1.06252

Tests for Location: Mu0=1				
Test	-Statistic-		-----p Value-----	
Student's t	t	0.356374	Pr > t	0.7216
Sign	M	-140	Pr >= M	<.0001
Signed Rank	S	-50781	Pr >= S	<.0001
Tests for Normality				
Test	--Statistic---		-----p Value-----	
Shapiro-Wilk	W	0.801498	Pr < W	<0.0001
Kolmogorov-Smirnov	D	0.166308	Pr > D	<0.0100
Cramer-von Mises	W-Sq	9.507975	Pr > W-Sq	<0.0050
Anderson-Darling	A-Sq	54.5478	Pr > A-Sq	<0.0050
Quantiles (Definition 5)				
Quantile		Estimate		
100% Max		6.63906758		
99%		5.04491651		
95%		3.13482318		
90%		2.37803632		
75% Q3		1.35733401		
50% Median		0.68950221		
25% Q1		0.29481436		
10%		0.10219011		
5%		0.05192799		
1%		0.01195590		
0% Min		0.00055441		

The next DATA step generates 1000 different samples from the same exponential distribution. Each sample contains ten observations. The MEANS procedure computes the mean of each sample. In the data set that is created by PROC MEANS, each observation represents the mean of a sample of ten observations from an exponential distribution. Thus, the data set is a sample from the sampling distribution of the mean for an exponential population.

PROC UNIVARIATE displays statistics for this sample of means. Notice that the mean of the sample of means is .99, almost the same as the mean of the original population. Theoretically, the standard deviation of the sampling distribution is $\sigma/\sqrt{n} = 1.00/\sqrt{10} = .32$, whereas the standard deviation of this sample from the sampling distribution is .30. The skewness (.55) and kurtosis (-.006) are closer to zero in the sample from the sampling distribution than in the original sample from the exponential distribution. This is so because the sampling distribution is closer to a normal distribution than is the original exponential distribution. The CHART procedure displays a histogram of the 1000-sample means. The shape of the histogram is much closer to a belllike, normal density, but it is still distinctly lopsided.

```
options nodate pageno=1 linesize=80 pagesize=48;

title '1000 Sample Means with 10 Obs per Sample';
title2 'Drawn from an Exponential Distribution';

data samp10;
  drop n;
  do Sample=1 to 1000;
    do n=1 to 10;
```

```

            x=ranexp(433879);
            output;
        end;
    end;

proc means data=samp10 noprint;
    output out=mean10 mean=Mean;
    var x;
    by sample;
run;

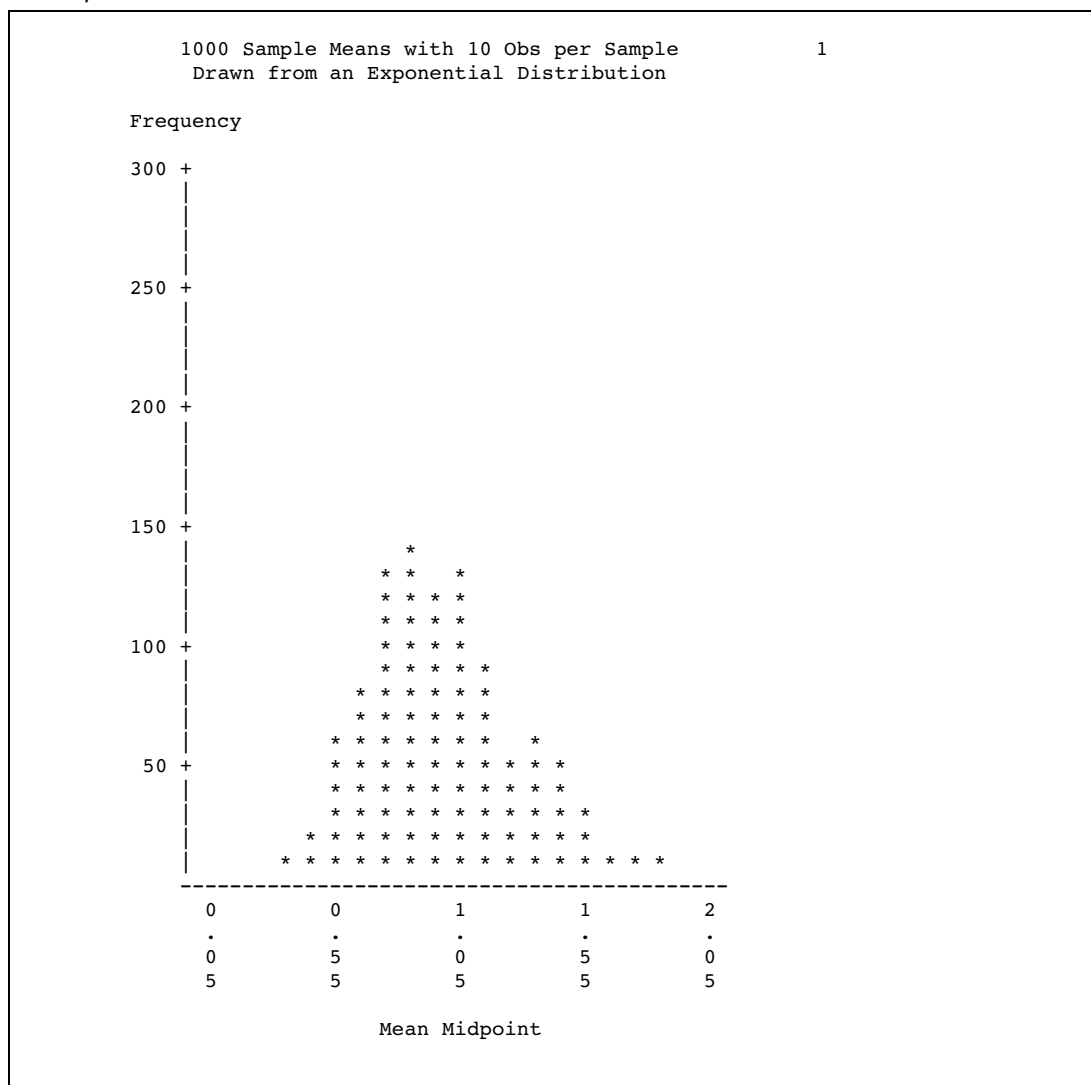
proc format;
    value axisfmt
        .05='0.05'
        .55='0.55'
        1.05='1.05'
        1.55='1.55'
        2.05='2.05'
        other=' ';
run;

proc chart data=mean10;
    vbar mean/axis=300
        midpoints=0.05 to 2.05 by .1;
    format mean axisfmt.;
run;

options pagesize=64;
proc univariate data=mean10 noextrobs=0 normal
    mu0=1;
    var mean;

```

```
run;
```



```

1000 Sample Means with 10 Obs per Sample
Drawn from an Exponential Distribution
2

The UNIVARIATE Procedure
Variable: Mean

Moments

N          1000      Sum Weights          1000
Mean      0.9906857  Sum Observations  990.685697
Std Deviation 0.30732649 Variance      0.09444957
Skewness   0.54575615 Kurtosis      -0.0060892
Uncorrected SS 1075.81327 Corrected SS  94.3551193
Coeff Variation 31.0215931 Std Error Mean 0.00971852

Basic Statistical Measures

Location          Variability

Mean      0.990686      Std Deviation      0.30733
Median    0.956152      Variance          0.09445
Mode      .             Range            1.79783
                  Interquartile Range      0.41703

```

Tests for Location: Mu0=1				
Test	-Statistic-	-----p Value-----		
Student's t	t -0.95841	Pr > t	0.3381	
Sign	M -53	Pr >= M	0.0009	
Signed Rank	S -22687	Pr >= S	0.0129	
Tests for Normality				
Test	--Statistic---	-----p Value-----		
Shapiro-Wilk	W 0.9779	Pr < W	<0.0001	
Kolmogorov-Smirnov	D 0.055498	Pr > D	<0.0100	
Cramer-von Mises	W-Sq 0.953926	Pr > W-Sq	<0.0050	
Anderson-Darling	A-Sq 5.945023	Pr > A-Sq	<0.0050	
Quantiles (Definition 5)				
Quantile	Estimate			
100% Max	2.053899			
99%	1.827503			
95%	1.557175			
90%	1.416611			
75% Q3	1.181006			
50% Median	0.956152			
25% Q1	0.763973			
10%	0.621787			
5%	0.553568			
1%	0.433820			
0% Min	0.256069			

In the following DATA step, the size of each sample from the exponential distribution is increased to 50. The standard deviation of the sampling distribution is smaller than in the previous example because the size of each sample is larger. Also, the sampling distribution is even closer to a normal distribution, as can be seen from the histogram and the skewness.

```
options nodate pageno=1 linesize=80 pagesize=48;

title '1000 Sample Means with 50 Obs per Sample';
title2 'Drawn from an Exponential Distribution';

data samp50;
  drop n;
  do sample=1 to 1000;
    do n=1 to 50;
      X=ranexp(72437213);
      output;
    end;
  end;

proc means data=samp50 noprint;
  output out=mean50 mean=Mean;
  var x;
  by sample;
run;
```

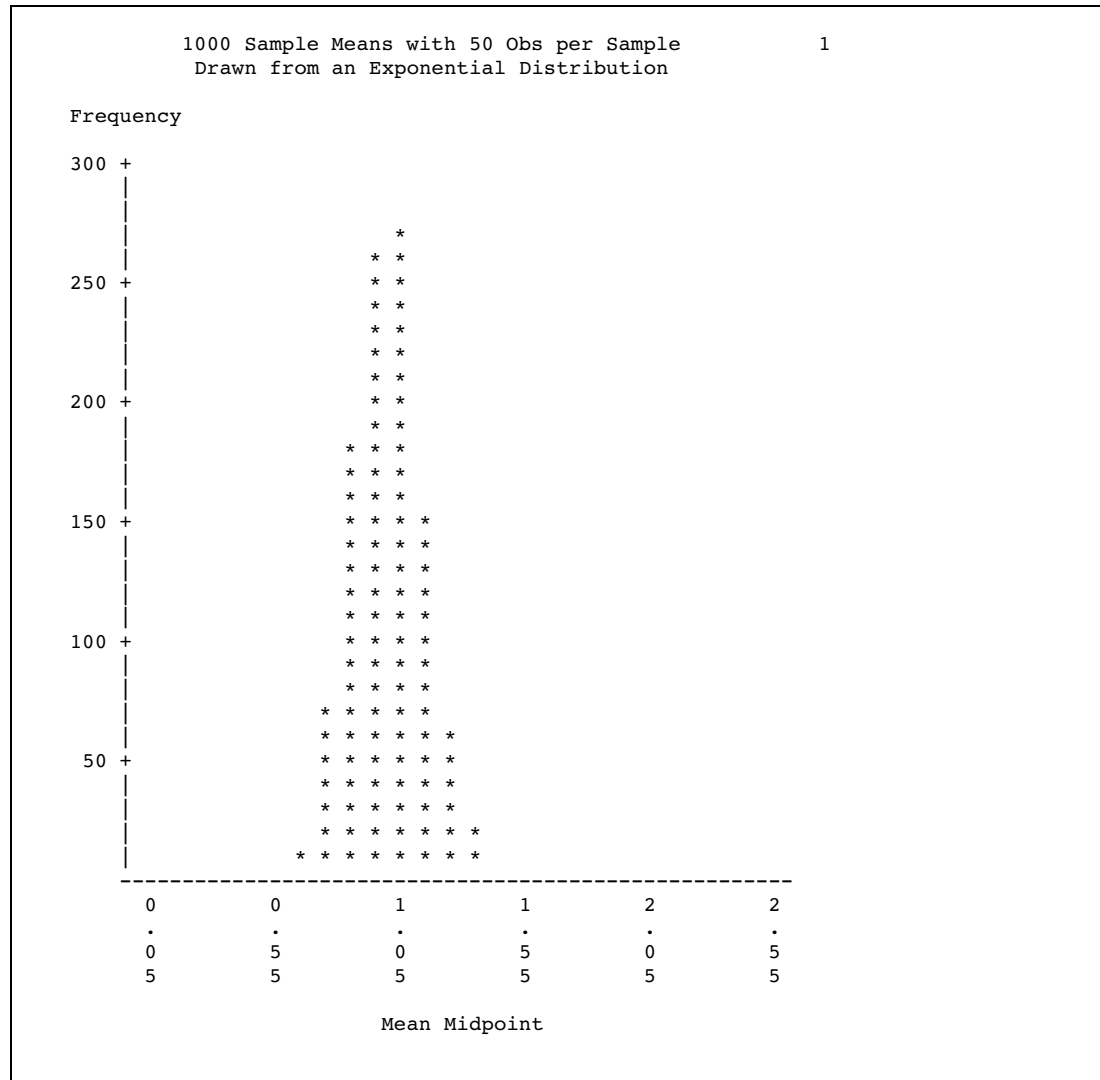
```
proc format;
  value axisfmt
    .05='0.05'
    .55='0.55'
    1.05='1.05'
    1.55='1.55'
    2.05='2.05'
    2.55='2.55'
    other=' ';
run;

proc chart data=mean50;
  vbar mean / axis=300
             midpoints=0.05 to 2.55 by .1;
  format mean axisfmt.;
run;

options pagesize=64;

proc univariate data=mean50 nextrobs=0 normal
               mu0=1;
  var mean;
```

run;



1000 Sample Means with 50 Obs per Sample Drawn from an Exponential Distribution				2
The UNIVARIATE Procedure				
Variable: Mean				
Moments				
N	1000	Sum Weights	1000	
Mean	0.99679697	Sum Observations	996.796973	
Std Deviation	0.13815404	Variance	0.01908654	
Skewness	0.19062633	Kurtosis	-0.1438604	
Uncorrected SS	1012.67166	Corrected SS	19.067451	
Coeff Variation	13.8597969	Std Error Mean	0.00436881	
Basic Statistical Measures				
Location		Variability		
Mean	0.996797	Std Deviation	0.13815	
Median	0.996023	Variance	0.01909	
Mode	.	Range	0.87040	
		Interquartile Range	0.18956	

Tests for Location: Mu0=1				
Test	-Statistic-		-----p Value-----	
Student's t	t	-0.73316	Pr > t	0.4636
Sign	M	-13	Pr >= M	0.4292
Signed Rank	S	-10767	Pr >= S	0.2388

Tests for Normality				
Test	--Statistic---		-----p Value-----	
Shapiro-Wilk	W	0.996493	Pr < W	0.0247
Kolmogorov-Smirnov	D	0.023687	Pr > D	>0.1500
Cramer-von Mises	W-Sq	0.084468	Pr > W-Sq	0.1882
Anderson-Darling	A-Sq	0.66039	Pr > A-Sq	0.0877

Quantiles (Definition 5)		
Quantile	Estimate	
100% Max	1.454957	
99%	1.337016	
95%	1.231508	
90%	1.179223	
75% Q3	1.086515	
50% Median	0.996023	
25% Q1	0.896953	
10%	0.814906	
5%	0.780783	
1%	0.706588	
0% Min	0.584558	

Testing Hypotheses

The purpose of the statistical methods that have been discussed so far is to estimate a population parameter by means of a sample statistic. Another class of statistical methods is used for testing hypotheses about population parameters or for measuring the amount of evidence against a hypothesis.

Consider the universe of students in a college. Let the variable X be the number of pounds by which a student's weight deviates from the ideal weight for a person of the same sex, height, and build. You want to find out whether the population of students is, on the average, underweight or overweight. To this end, you have taken a random sample of X values from nine students, with results as given in the following DATA step:

```

title 'Deviations from Normal Weight';

data x;
  input X @@;
  datalines;
-7 -2 1 3 6 10 15 21 30
;

```

You can define several hypotheses of interest. One hypothesis is that, on the average, the students are of exactly ideal weight. If μ represents the population mean of the X values, then you can write this hypothesis, called the *null* hypothesis, as $H_0 : \mu = 0$. The other two hypotheses, called *alternative* hypotheses, are that the students are underweight on the average, $H_1 : \mu < 0$, and that the students are overweight on the average, $H_2 : \mu > 0$.

The null hypothesis is so called because in many situations it corresponds to the assumption of “no effect” or “no difference.” However, this interpretation is not appropriate for all testing problems. The null hypothesis is like a straw man that can be toppled by statistical evidence. You decide between the alternative hypotheses according to which way the straw man falls.

A naive way to approach this problem would be to look at the sample mean \bar{x} and decide among the three hypotheses according to the following rule:

- If $\bar{x} < 0$, then decide on $H_1 : \mu < 0$.
- If $\bar{x} = 0$, then decide on $H_0 : \mu = 0$.
- If $\bar{x} > 0$, then decide on $H_2 : \mu > 0$.

The trouble with this approach is that there may be a high probability of making an incorrect decision. If H_0 is true, then you are nearly certain to make a wrong decision because the chances of \bar{x} being exactly zero are almost nil. If μ is slightly less than zero, so that H_1 is true, then there may be nearly a 50 percent chance that \bar{x} will be greater than zero in repeated sampling, so the chances of incorrectly choosing H_2 would also be nearly 50 percent. Thus, you have a high probability of making an error if \bar{x} is near zero. In such cases, there is not enough evidence to make a confident decision, so the best response may be to reserve judgment until you can obtain more evidence.

The question is, how far from zero must \bar{x} be for you to be able to make a confident decision? The answer can be obtained by considering the sampling distribution of \bar{x} . If X has a roughly normal distribution, then \bar{x} has an approximately normal sampling distribution. The mean of the sampling distribution of \bar{x} is μ . Assume temporarily that σ , the standard deviation of X , is known to be 12. Then the standard error of \bar{x} for samples of nine observations is $\sigma/\sqrt{n} = 12/\sqrt{9} = 4$.

You know that about 95 percent of the values from a normal distribution are within two standard deviations of the mean, so about 95 percent of the possible samples of nine X values have a sample mean \bar{x} between $0 - 2(4)$ and $0 + 2(4)$, or between -8 and 8 . Consider the chances of making an error with the following decision rule:

- If $\bar{x} < -8$, then decide on $H_1 : \mu < 0$.
- If $-8 \leq \bar{x} \leq 8$, then reserve judgment.
- If $\bar{x} > 8$, then decide on $H_2 : \mu > 0$.

If H_0 is true, then in about 95 percent of the possible samples \bar{x} will be between the *critical values* -8 and 8 , so you will reserve judgment. In these cases the statistical evidence is not strong enough to fell the straw man. In the other 5 percent of the samples you will make an error; in 2.5 percent of the samples you will incorrectly choose H_1 , and in 2.5 percent you will incorrectly choose H_2 .

The price you pay for controlling the chances of making an error is the necessity of reserving judgment when there is not sufficient statistical evidence to reject the null hypothesis.

Significance and Power

The probability of rejecting the null hypothesis if it is true is called the *Type I error rate* of the statistical test and is typically denoted as α . In this example, an \bar{x} value less than -8 or greater than 8 is said to be *statistically significant* at the 5 percent level. You can adjust the type I error rate according to your needs by choosing different critical values. For example, critical values of -4 and 4 would produce a significance level of about 32 percent, while -12 and 12 would give a type I error rate of about 0.3 percent.

The decision rule is a *two-tailed test* because the alternative hypotheses allow for population means either smaller or larger than the value specified in the null hypothesis. If you were interested only in the possibility of the students being overweight on the average, then you could use a *one-tailed test*:

- If $\bar{x} \leq 8$, then reserve judgment.
- If $\bar{x} > 8$, then decide on $H_2 : \mu > 0$.

For this one-tailed test, the type I error rate is 2.5 percent, half that of the two-tailed test.

The probability of rejecting the null hypothesis if it is false is called the *power* of the statistical test and is typically denoted as $1 - \beta$. β is called the *Type II error rate*, which is the probability of not rejecting a false null hypothesis. The power depends on the true value of the parameter. In the example, assume the population mean is 4. The power for detecting H_2 is the probability of getting a sample mean greater than 8. The critical value 8 is one standard error higher than the population mean 4. The chance of getting a value at least one standard deviation greater than the mean from a normal distribution is about 16 percent, so the power for detecting the alternative hypothesis H_2 is about 16 percent. If the population mean were 8, then the power for H_2 would be 50 percent, whereas a population mean of 12 would yield a power of about 84 percent.

The smaller the type I error rate is, the less the chance of making an incorrect decision, but the higher the chance of having to reserve judgment. In choosing a type I error rate, you should consider the resulting power for various alternatives of interest.

Student's t Distribution

In practice, you usually cannot use any decision rule that uses a critical value based on σ because you do not usually know the value of σ . You can, however, use s as an estimate of σ . Consider the following statistic:

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

This t statistic is the difference between the sample mean and the hypothesized mean μ_0 divided by the estimated standard error of the mean.

If the null hypothesis is true and the population is normally distributed, then the t statistic has what is called a *Student's t distribution* with $n - 1$ degrees of freedom. This distribution looks very similar to a normal distribution, but the tails of the Student's t distribution are heavier. As the sample size gets larger, the sample standard deviation becomes a better estimator of the population standard deviation, and the t distribution gets closer to a normal distribution.

You can base a decision rule on the t statistic:

- If $t < -2.3$, then decide on $H_1 : \mu < 0$.
- If $-2.3 \leq t \leq 2.3$, then reserve judgment.
- If $t > 2.3$, then decide on $H_0 : \mu > 0$.

The value 2.3 was obtained from a table of Student's t distribution to give a type I error rate of 5 percent for 8 (that is, $9 - 1 = 8$) degrees of freedom. Most common statistics texts contain a table of Student's t distribution. If you do not have a statistics text handy, then you can use the DATA step and the TINV function to print any values from the t distribution.

By default, PROC UNIVARIATE computes a t statistic for the null hypothesis that $\mu_0 = 0$, along with related statistics. Use the MU0= option in the PROC statement to specify another value for the null hypothesis.

This example uses the data on deviations from normal weight, which consist of nine observations. First, PROC MEANS computes the t statistic for the null hypothesis that $\mu = 0$. Then, the TINV function in a DATA step computes the value of Student's t distribution for a two-tailed test at the 5 percent level of significance and 8 degrees of freedom.

```

data devnorm;
    title 'Deviations from Normal Weight';
    input X @@;
    datalines;
-7 -2 1 3 6 10 15 21 30
;

proc means data=devnorm maxdec=3 n mean
           std stderr t probt;
run;

title 'Student's t Critical Value';

data _null_;
    file print;
    t=tinv(.975,8);
    put t 5.3;
run;

```

Deviations from Normal Weight						1
The MEANS Procedure						
Analysis Variable : X						
N	Mean	Std Dev	Std Error	t Value	Pr > t	
9	8.556	11.759	3.920	2.18	0.0606	

Student's t Critical Value		2
2.306		

In the current example, the value of the t statistic is 2.18, which is less than the critical t value of 2.3 (for a 5 percent significance level and 8 degrees of freedom). Thus, at a 5 percent significance level you must reserve judgment. If you had elected to use a 10 percent significance level, then the critical value of the t distribution would have been 1.86 and you could have rejected the null hypothesis. The sample size is so small, however, that the validity of your conclusion depends strongly on how close the distribution of the population is to a normal distribution.

Probability Values

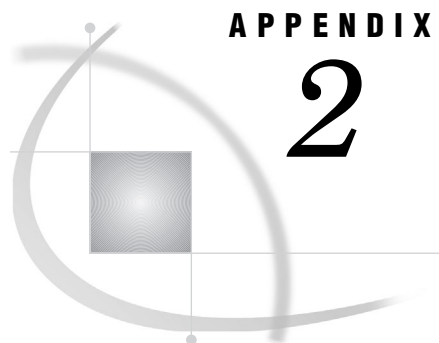
Another way to report the results of a statistical test is to compute a *probability value* or *p-value*. A p -value gives the probability in repeated sampling of obtaining a statistic as far in the direction(s) specified by the alternative hypothesis as is the value actually observed. A two-tailed p -value for a t statistic is the probability of obtaining an absolute t value that is greater than the observed absolute t value. A one-tailed p -value for a t statistic for the alternative hypothesis $\mu > \mu_0$ is the probability of obtaining a t value greater than the observed t value. Once the p -value is computed, you can perform a hypothesis test by comparing the p -value with the desired significance level. If the p -value is less than or equal to the type I error rate of the test, then the null hypothesis can be rejected. The two-tailed p -value, labeled **Pr > |t|** in the PROC MEANS output,

is .0606, so the null hypothesis could be rejected at the 10 percent significance level but not at the 5 percent level.

A p -value is a measure of the strength of the evidence against the null hypothesis. The smaller the p -value, the stronger the evidence for rejecting the null hypothesis.

References

- Ali, M.M. (1974), "Stochastic Ordering and Kurtosis Measure," *Journal of the American Statistical Association*, 69, 543–545.
- Johnson, M.E., Tietjen, G.L., and Beckman, R.J. (1980), "A New Family of Probability Distributions With Applications to Monte Carlo Studies," *Journal of the American Statistical Association*, 75, 276–279.
- Kaplansky, I. (1945), "A Common Error Concerning Kurtosis," *Journal of the American Statistical Association*, 40, 259–263.
- Mendenhall, W. and Beaver, R.. (1998), *Introduction to Probability and Statistics*, 10th Edition, Belmont, CA: Wadsworth Publishing Company.
- Ott, R. and Mendenhall, W. (1994) *Understanding Statistics*, 6th Edition, North Scituate, MA: Duxbury Press.
- Schlotzhauer, S.D. and Littell, R.C. (1997), *SAS System for Elementary Statistical Analysis*, Second Edition, Cary, NC: SAS Institute Inc.
- Snedecor, G.W. and Cochran, W.C. (1989), *Statistical Methods*, 8th Edition, Ames, IA: Iowa State University Press.



APPENDIX

2

Operating Environment-Specific Procedures

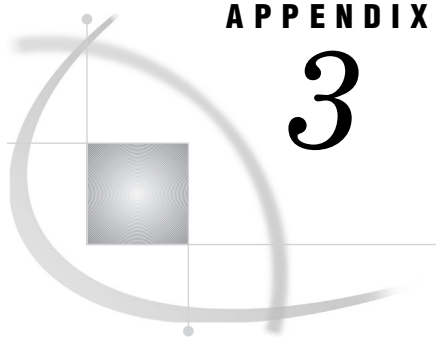
Descriptions of Operating Environment-Specific Procedures 1613

Descriptions of Operating Environment-Specific Procedures

The following table gives a brief description and the relevant releases for some common operating environment-specific procedures. All of these procedures are described in more detail in operating environment-companion documentation.

Table A2.1 Host-Specific Procedures

Procedure	Description	Releases
BMDP	Calls any BMDP program to analyze data in a SAS data set.	All
CONVERT	Converts BMDP, OSIRIS, and SPSS system files to SAS data sets.	All
C16PORT	Converts a 16-bit SAS data library or catalog created in Release 6.08 to a transport file, which you can then convert to a 32-bit format for use in the current release of SAS by using the CIMPORT procedure.	6.10 - 6.12
FSDEVICE	Creates, copies, modifies, deletes, or renames device descriptions in a catalog.	All
PDS	Lists, deletes, or renames the members of a partitioned data set.	6.09E
PDSCOPY	Copies partitioned data sets from disk to disk, disk to tape, tape to tape, or tape to disk.	6.09E
RELEASE	Releases unused space at the end of a disk data set.	6.09E
SOURCE	Provides an easy way to back up and process source library data sets.	6.09E
TAPECOPY	Copies an entire tape volume, or files from one or more tape volumes, to one output tape volume.	6.09E
TAPELABEL	Writes the label information of an IBM standard-labeled tape volume to the SAS procedure output file.	6.09E



APPENDIX

3

Raw Data and DATA Steps

<i>Overview</i>	1615
<i>AIRCRAFT</i>	1615
<i>CENSUS</i>	1616
<i>CHARITY</i>	1617
<i>CUSTOMER_RESPONSE</i>	1619
<i>DJIA</i>	1621
<i>EDUCATION</i>	1622
<i>EMPDATA</i>	1623
<i>ENERGY</i>	1625
<i>GROC</i>	1626
<i>HOMELOANS</i>	1627
<i>MATCH_11</i>	1641
<i>PROCLIB.DELAY</i>	1642
<i>PROCLIB.EMP95</i>	1643
<i>PROCLIB.EMP96</i>	1644
<i>PROCLIB.INTERNAT</i>	1645
<i>PROCLIB.LAKES</i>	1646
<i>PROCLIB.MARCH</i>	1646
<i>PROCLIB.PAYLIST2</i>	1647
<i>PROCLIB.PAYROLL</i>	1648
<i>PROCLIB.PAYROLL2</i>	1651
<i>PROCLIB.SCHEDULE</i>	1651
<i>PROCLIB.STAFF</i>	1654
<i>PROCLIB.SUPERV</i>	1657
<i>RADIO</i>	1658
<i>STATEPOP</i>	1670

Overview

The programs for examples in this document generally show you how to create the data sets that are used. Some examples show only partial data. For these examples, the complete data is shown in this appendix.

AIRCRAFT

```

data aircraft;
    input HoleSize MMC_Bonus PositionDev @@;
    label HoleSize = 'Actual Hole Size'
           MMC_Bonus = 'MMC Bonus'
           AdjustTol = 'Adjusted Tolerance'
           PositionDev = 'True Position Deviation'
           Deviation = 'Position Deviation - Adjusted Tolerance';
    AdjustTol = 0.006 + MMC_Bonus;
    Deviation = PositionDev - AdjustTol;
datalines;
0.25776 0.00176 0.00123 0.25726 0.00126 0.00867
0.25783 0.00183 0.00081 0.25806 0.00206 0.00072
0.25798 0.00198 0.00149 0.25793 0.00193 0.00192
0.25752 0.00152 0.00121 0.25759 0.00159 0.00611
0.25937 0.00337 0.00206 0.25789 0.00189 0.00025
0.25800 0.00200 0.00525 0.25808 0.00208 0.00311
0.25773 0.00173 0.00032 0.25827 0.00227 0.00154
0.25797 0.00197 0.00224 0.25834 0.00234 0.00205
0.25945 0.00345 0.00274 0.25801 0.00201 0.00555
0.25775 0.00175 0.00553 0.25843 0.00243 0.00036
0.25799 0.00199 0.00178 0.25807 0.00207 0.00022
0.25819 0.00219 0.00275 0.25823 0.00223 0.00312
0.25941 0.00341 0.00803 0.25852 0.00252 0.00243
0.25887 0.00287 0.00925 0.25915 0.00315 0.00157
0.25977 0.00377 0.00246 0.25763 0.00163 0.00308
;

```

CENSUS

```

data census;
    input Density CrimeRate State $ 14-27 PostalCode $ 29-30;
    datalines;
263.3 4575.3 Ohio OH
62.1 7017.1 Washington WA
103.4 5161.9 South Carolina SC
53.4 3438.6 Mississippi MS
180.0 8503.2 Florida FL
80.8 2190.7 West Virginia WV
428.7 5477.6 Maryland MD
71.2 4707.5 Missouri MO
43.9 4245.2 Arkansas AR
7.3 6371.4 Nevada NV
264.3 3163.2 Pennsylvania PA
11.5 4156.3 Idaho ID
44.1 6025.6 Oklahoma OK
51.2 4615.8 Minnesota MN
55.2 4271.2 Vermont VT
27.4 6969.9 Oregon OR
205.3 5416.5 Illinois IL
94.1 5792.0 Georgia GA
9.1 2678.0 South Dakota SD

```

```

9.4 2833.0   North Dakota   ND
102.4 3371.7 New Hampshire NH
54.3 7722.4   Texas          TX
76.6 4451.4   Alabama        AL
307.6 4938.8 Delaware       DE
151.4 6506.4 California    CA
111.6 4665.6 Tennessee     TN
120.4 4649.9 North Carolina NC
;

```

CHARITY

```

data Charity;
    input School $ 1-7 Year 9-12 Name $ 14-20 MoneyRaised 22-26
           HoursVolunteered 28-29;
    datalines;
Monroe 1992 Allison 31.65 19
Monroe 1992 Barry   23.76 16
Monroe 1992 Candace 21.11  5
Monroe 1992 Danny   6.89 23
Monroe 1992 Edward  53.76 31
Monroe 1992 Fiona   48.55 13
Monroe 1992 Gert    24.00 16
Monroe 1992 Harold  27.55 17
Monroe 1992 Ima     15.98  9
Monroe 1992 Jack    20.00 23
Monroe 1992 Katie   22.11  2
Monroe 1992 Lisa    18.34 17
Monroe 1992 Tonya   55.16 40
Monroe 1992 Max     26.77 34
Monroe 1992 Ned     28.43 22
Monroe 1992 Opal    32.66 14
Monroe 1993 Patsy   18.33 18
Monroe 1993 Quentin 16.89 15
Monroe 1993 Randall 12.98 17
Monroe 1993 Sam     15.88  5
Monroe 1993 Tyra    21.88 23
Monroe 1993 Myrtle  47.33 26
Monroe 1993 Frank   41.11 22
Monroe 1993 Cameron 65.44 14
Monroe 1993 Vern    17.89 11
Monroe 1993 Wendell 23.00 10
Monroe 1993 Bob     26.88  6
Monroe 1993 Leah    28.99 23
Monroe 1994 Becky   30.33 26
Monroe 1994 Sally   35.75 27
Monroe 1994 Edgar   27.11 12
Monroe 1994 Dawson  17.24 16
Monroe 1994 Lou     5.12 16
Monroe 1994 Damien  18.74 17
Monroe 1994 Mona    27.43  7
Monroe 1994 Della   56.78 15

```

Monroe	1994	Monique	29.88	19
Monroe	1994	Carl	31.12	25
Monroe	1994	Reba	35.16	22
Monroe	1994	Dax	27.65	23
Monroe	1994	Gary	23.11	15
Monroe	1994	Suzie	26.65	11
Monroe	1994	Benito	47.44	18
Monroe	1994	Thomas	21.99	23
Monroe	1994	Annie	24.99	27
Monroe	1994	Paul	27.98	22
Monroe	1994	Alex	24.00	16
Monroe	1994	Lauren	15.00	17
Monroe	1994	Julia	12.98	15
Monroe	1994	Keith	11.89	19
Monroe	1994	Jackie	26.88	22
Monroe	1994	Pablo	13.98	28
Monroe	1994	L.T.	56.87	33
Monroe	1994	Willard	78.65	24
Monroe	1994	Kathy	32.88	11
Monroe	1994	Abby	35.88	10
Kennedy	1992	Arturo	34.98	14
Kennedy	1992	Grace	27.55	25
Kennedy	1992	Winston	23.88	22
Kennedy	1992	Vince	12.88	21
Kennedy	1992	Claude	15.62	5
Kennedy	1992	Mary	28.99	34
Kennedy	1992	Abner	25.89	22
Kennedy	1992	Jay	35.89	35
Kennedy	1992	Alicia	28.77	26
Kennedy	1992	Freddy	29.00	27
Kennedy	1992	Eloise	31.67	25
Kennedy	1992	Jenny	43.89	22
Kennedy	1992	Thelma	52.63	21
Kennedy	1992	Tina	19.67	21
Kennedy	1992	Eric	24.89	12
Kennedy	1993	Bubba	37.88	12
Kennedy	1993	G.L.	25.89	21
Kennedy	1993	Bert	28.89	21
Kennedy	1993	Clay	26.44	21
Kennedy	1993	Leeann	27.17	17
Kennedy	1993	Georgia	38.90	11
Kennedy	1993	Bill	42.23	25
Kennedy	1993	Holly	18.67	27
Kennedy	1993	Benny	19.09	25
Kennedy	1993	Cammie	28.77	28
Kennedy	1993	Amy	27.08	31
Kennedy	1993	Doris	22.22	24
Kennedy	1993	Robbie	19.80	24
Kennedy	1993	Ted	27.07	25
Kennedy	1993	Sarah	24.44	12
Kennedy	1993	Megan	28.89	11
Kennedy	1993	Jeff	31.11	12
Kennedy	1993	Taz	30.55	11
Kennedy	1993	George	27.56	11

```

Kennedy 1993 Heather 38.67 15
Kennedy 1994 Nancy 29.90 26
Kennedy 1994 Rusty 30.55 28
Kennedy 1994 Mimi 37.67 22
Kennedy 1994 J.C. 23.33 27
Kennedy 1994 Clark 27.90 25
Kennedy 1994 Rudy 27.78 23
Kennedy 1994 Samuel 34.44 18
Kennedy 1994 Forrest 28.89 26
Kennedy 1994 Luther 72.22 24
Kennedy 1994 Trey 6.78 18
Kennedy 1994 Albert 23.33 19
Kennedy 1994 Che-Min 26.66 33
Kennedy 1994 Preston 32.22 23
Kennedy 1994 Larry 40.00 26
Kennedy 1994 Anton 35.99 28
Kennedy 1994 Sid 27.45 25
Kennedy 1994 Will 28.88 21
Kennedy 1994 Morty 34.44 25
;

```

CUSTOMER_RESPONSE

```

data customer_response;
    input Customer Factor1-Factor4 Source1-Source3
           Quality1-Quality3;
    datalines;
1 . . 1 1 1 1 . 1 . .
2 1 1 . 1 1 1 . 1 1 .
3 . . 1 1 1 1 . . . .
4 1 1 . 1 . 1 . . . 1
5 . 1 . 1 1 . . . . 1
6 . 1 . 1 1 . . . . .
7 . 1 . 1 1 . . 1 . .
8 1 . . 1 1 1 . 1 1 .
9 1 1 . 1 1 . . . . 1
10 1 . . 1 1 1 . 1 1 .
11 1 1 1 1 . 1 . 1 1 1
12 1 1 . 1 1 1 . . . .
13 1 1 . 1 . 1 . 1 1 .
14 1 1 . 1 1 1 . . . .
15 1 1 . 1 . 1 . 1 1 1
16 1 . . 1 1 . . 1 . .
17 1 1 . 1 1 1 . . 1 .
18 1 1 . 1 1 1 1 . . 1
19 . 1 . 1 1 1 1 . 1 .
20 1 . . 1 1 1 . 1 1 1
21 . . . 1 1 1 . 1 . .
22 . . . 1 1 1 . 1 1 .
23 1 . . 1 . . . . . 1
24 . 1 . 1 1 . . 1 . 1
25 1 1 . 1 1 . . . 1 1

```

26 1 1 . 1 1 . . 1 . .
 27 1 . . 1 1 . . . 1 .
 28 1 1 . 1 . . . 1 1 1
 29 1 . . 1 1 1 . 1 . 1
 30 1 . 1 1 1 . . 1 1 .
 31 . . . 1 1 . . 1 1 .
 32 1 1 1 1 1 . . 1 1 1
 33 1 . . 1 1 . . 1 . 1
 34 . . 1 1 . . . 1 1 .
 35 1 1 1 1 1 . 1 1 . .
 36 1 1 1 1 . 1 . 1 . .
 37 1 1 . 1 . . . 1 . .
 38 . . . 1 1 1 . 1 . .
 39 1 1 . 1 1 . . 1 . 1
 40 1 . . 1 . . 1 1 . 1
 41 1 . . 1 1 1 1 1 . 1
 42 1 1 1 1 . . 1 1 . .
 43 1 . . 1 1 1 . 1 . .
 44 1 . 1 1 . 1 . 1 . 1
 45 . . . 1 . . 1 . . 1
 46 . . . 1 1 . . . 1 .
 47 1 1 . 1 . . 1 1 . .
 48 1 . 1 1 1 . 1 1 . .
 49 . . 1 1 1 1 . 1 . 1
 50 . 1 . 1 1 . . 1 1 .
 51 1 . 1 1 1 1
 52 1 1 1 1 1 1 . 1 . .
 53 . 1 1 1 . 1 . 1 1 1
 54 1 . . 1 1 . . 1 1 .
 55 1 1 . 1 1 1 . 1 . .
 56 1 . . 1 1 . . 1 1 .
 57 1 1 . 1 1 . 1 . . 1
 58 . 1 . 1 . 1 . . 1 1
 59 1 1 1 1 . . 1 1 1 .
 60 . 1 1 1 1 1 . . 1 1
 61 1 1 1 1 1 1 . 1 . .
 62 1 1 . 1 1 . . 1 1 .
 63 . . . 1 . . . 1 1 1
 64 1 . . 1 1 1 . 1 . .
 65 1 . . 1 1 1 . 1 . .
 66 1 . . 1 1 1 1 1 1 .
 67 1 1 . 1 1 1 . 1 1 .
 68 1 1 . 1 1 1 . 1 1 .
 69 1 1 . 1 1 . 1 . . .
 70 . . . 1 1 1 . 1 . .
 71 1 . . 1 1 . 1 . . 1
 72 1 . 1 1 1 1 . . 1 .
 73 1 1 . 1 . 1 . 1 1 .
 74 1 1 1 1 1 1 . 1 . .
 75 . 1 . 1 1 1 . . 1 .
 76 1 1 . 1 1 1 . 1 1 1
 77 . . . 1 1 1
 78 1 1 1 1 1 1 . 1 1 .
 79 1 . . 1 1 1 . 1 1 .

```

80 1 1 1 1 1 . 1 1 . 1
81 1 1 . 1 1 1 1 1 1 .
82 . . . 1 1 1 1 . . .
83 1 1 . 1 1 1 . 1 1 .
84 1 . . 1 1 . . 1 1 .
85 . . . 1 . 1 . 1 . .
86 1 . . 1 1 1 . 1 1 1
87 1 1 . 1 1 1 . 1 . .
88 . . . 1 . 1 . . . .
89 1 . . 1 . 1 . . 1 1
90 1 1 . 1 1 1 . 1 . 1
91 . . . 1 1 . . . 1 .
92 1 . . 1 1 1 . 1 1 .
93 1 . . 1 1 . . 1 1 .
94 1 . . 1 1 1 1 1 . .
95 1 . . 1 . 1 1 1 1 .
96 1 . 1 1 1 1 . . 1 .
97 1 1 . 1 1 . . . 1 .
98 1 . 1 1 1 1 1 1 . .
99 1 1 . 1 1 1 1 1 1 .
100 1 . 1 1 1 . . . 1 1
101 1 . 1 1 1 1 . . . .
102 1 . . 1 1 . 1 1 . .
103 1 1 . 1 1 1 . 1 . .
104 . . . 1 1 1 . 1 1 1
105 1 . 1 1 1 . . 1 . 1
106 1 1 1 1 1 1 1 1 1 1
107 1 1 1 1 . . . 1 . 1
108 1 . . 1 . 1 1 1 . .
109 . 1 . 1 1 . . 1 1 .
110 1 . . 1 . . . . . .
111 1 . . 1 1 1 . 1 1 .
112 1 1 . 1 1 1 . . . 1
113 1 1 . 1 1 . 1 1 1 .
114 1 1 . 1 1 . . . . .
115 1 1 . 1 1 . . 1 . .
116 . 1 . 1 1 1 1 1 . .
117 . 1 . 1 1 1 . . . .
118 . 1 1 1 1 . . 1 1 .
119 . . . 1 . . . 1 . .
120 1 1 . 1 . . . . 1 .
;

```

DJIA

```

data djia;
    input Year @7 HighDate date7. High @24 LowDate date7. Low;
    format highdate lowdate date7.;
    datalines;
1954 31DEC54 404.39 11JAN54 279.87
1955 30DEC55 488.40 17JAN55 388.20
1956 06APR56 521.05 23JAN56 462.35

```

1957	12JUL57	520.77	22OCT57	419.79
1958	31DEC58	583.65	25FEB58	436.89
1959	31DEC59	679.36	09FEB59	574.46
1960	05JAN60	685.47	25OCT60	568.05
1961	13DEC61	734.91	03JAN61	610.25
1962	03JAN62	726.01	26JUN62	535.76
1963	18DEC63	767.21	02JAN63	646.79
1964	18NOV64	891.71	02JAN64	768.08
1965	31DEC65	969.26	28JUN65	840.59
1966	09FEB66	995.15	07OCT66	744.32
1967	25SEP67	943.08	03JAN67	786.41
1968	03DEC68	985.21	21MAR68	825.13
1969	14MAY69	968.85	17DEC69	769.93
1970	29DEC70	842.00	06MAY70	631.16
1971	28APR71	950.82	23NOV71	797.97
1972	11DEC72	1036.27	26JAN72	889.15
1973	11JAN73	1051.70	05DEC73	788.31
1974	13MAR74	891.66	06DEC74	577.60
1975	15JUL75	881.81	02JAN75	632.04
1976	21SEP76	1014.79	02JAN76	858.71
1977	03JAN77	999.75	02NOV77	800.85
1978	08SEP78	907.74	28FEB78	742.12
1979	05OCT79	897.61	07NOV79	796.67
1980	20NOV80	1000.17	21APR80	759.13
1981	27APR81	1024.05	25SEP81	824.01
1982	27DEC82	1070.55	12AUG82	776.92
1983	29NOV83	1287.20	03JAN83	1027.04
1984	06JAN84	1286.64	24JUL84	1086.57
1985	16DEC85	1553.10	04JAN85	1184.96
1986	02DEC86	1955.57	22JAN86	1502.29
1987	25AUG87	2722.42	19OCT87	1738.74
1988	21OCT88	2183.50	20JAN88	1879.14
1989	09OCT89	2791.41	03JAN89	2144.64
1990	16JUL90	2999.75	11OCT90	2365.10
1991	31DEC91	3168.83	09JAN91	2470.30
1992	01JUN92	3413.21	09OCT92	3136.58
1993	29DEC93	3794.33	20JAN93	3241.95
1994	31JAN94	3978.36	04APR94	3593.35

;

EDUCATION

```
data education;
    input State $14. +1 Code $ DropoutRate Expenditures MathScore Region $;
    label dropoutrate='Dropout Percentage - 1989'
           expenditures='Expenditure Per Pupil - 1989'
           mathscore='8th Grade Math Exam - 1990';
    datalines;
Alabama      AL 22.3 3197 252 SE
Alaska       AK 35.8 7716 . W
Arizona      AZ 31.2 3902 259 W
Arkansas     AR 11.5 3273 256 SE
```



```

California    CA 32.7 4121 256 W
Colorado      CO 24.7 4408 267 W
Connecticut   CT 16.8 6857 270 NE
Delaware      DE 28.5 5422 261 NE
Florida       FL 38.5 4563 255 SE
Georgia       GA 27.9 3852 258 SE
Hawaii        HI 18.3 4121 251 W
Idaho         ID 21.8 2838 272 W
Illinois      IL 21.5 4906 260 MW
Indiana       IN 13.8 4284 267 MW
Iowa          IA 13.6 4285 278 MW
Kansas        KS 17.9 4443 .   MW
Kentucky     KY 32.7 3347 256 SE
Louisiana     LA 43.1 3317 246 SE
Maine         ME 22.5 4744 .   NE
Maryland      MD 26.0 5758 260 NE
Massachusetts MA 28.0 5979 .   NE
Michigan      MI 29.3 5116 264 MW
Minnesota     MN 11.4 4755 276 MW
Mississippi   MS 39.9 2874 .   SE
Missouri      MO 26.5 4263 .   MW
Montana       MT 15.0 4293 280 W
Nebraska      NE 13.9 4360 276 MW
Nevada        NV 28.1 3791 .   W
New Hampshire NH 25.9 4807 273 NE
New Jersey    NE 20.4 7549 269 NE
New Mexico    NM 28.5 3473 256 W
New York      NY 35.0 .   261 NE
North Carolina NC 31.2 3874 250 SE
North Dakota  ND 12.1 3952 281 MW
Ohio          OH 24.4 4649 264 MW
;

```

EMPDATA

```

data empdata;
input IdNumber $ 1-4 LastName $ 9-19 FirstName $ 20-29
      City $ 30-42 State $ 43-44 /
      Gender $ 1 JobCode $ 9-11 Salary 20-29 @30 Birth date7.
      @43 Hired date7. HomePhone $ 54-65;
format birth hired date7.;
datalines;
1919  Adams      Gerald    Stamford  CT
M      TA2        34376    15SEP48   07JUN75   203/781-1255
1653  Alexander   Susan     Bridgeport CT
F      ME2        35108    18OCT52   12AUG78   203/675-7715
1400  Apple        Troy      New York  NY
M      ME1        29769    08NOV55   19OCT78   212/586-0808
1350  Arthur       Barbara   New York  NY
F      FA3        32886    03SEP53   01AUG78   718/383-1549
1401  Avery        Jerry     Paterson  NJ
M      TA3        38822    16DEC38   20NOV73   201/732-8787

```

1499	Barefoot	Joseph	Princeton	NJ	
M	ME3	43025	29APR42	10JUN68	201/812-5665
1101	Baucom	Walter	New York	NY	
M	SCP	18723	09JUN50	04OCT78	212/586-8060
1333	Blair	Justin	Stamford	CT	
M	PT2	88606	02APR49	13FEB69	203/781-1777
1402	Blalock	Ralph	New York	NY	
M	TA2	32615	20JAN51	05DEC78	718/384-2849
1479	Bostic	Marie	New York	NY	
F	TA3	38785	25DEC56	08OCT77	718/384-8816
1403	Bowden	Earl	Bridgeport	CT	
M	ME1	28072	31JAN57	24DEC79	203/675-3434
1739	Boyce	Jonathan	New York	NY	
M	PT1	66517	28DEC52	30JAN79	212/587-1247
1658	Bradley	Jeremy	New York	NY	
M	SCP	17943	11APR55	03MAR80	212/587-3622
1428	Brady	Christine	Stamford	CT	
F	PT1	68767	07APR58	19NOV79	203/781-1212
1782	Brown	Jason	Stamford	CT	
M	ME2	35345	07DEC58	25FEB80	203/781-0019
1244	Bryant	Leonard	New York	NY	
M	ME2	36925	03SEP51	20JAN76	718/383-3334
1383	Burnette	Thomas	New York	NY	
M	BCK	25823	28JAN56	23OCT80	718/384-3569
1574	Cahill	Marshall	New York	NY	
M	FA2	28572	30APR48	23DEC80	718/383-2338
1789	Caraway	Davis	New York	NY	
M	SCP	18326	28JAN45	14APR66	212/587-9000
1404	Carter	Donald	New York	NY	
M	PT2	91376	27FEB41	04JAN68	718/384-2946
1437	Carter	Dorothy	Bridgeport	CT	
F	A3	33104	23SEP48	03SEP72	203/675-4117
1639	Carter	Karen	Stamford	CT	
F	A3	40260	29JUN45	31JAN72	203/781-8839
1269	Caston	Franklin	Stamford	CT	
M	NA1	41690	06MAY60	01DEC80	203/781-3335
1065	Chapman	Neil	New York	NY	
M	ME2	35090	29JAN32	10JAN75	718/384-5618
1876	Chin	Jack	New York	NY	
M	TA3	39675	23MAY46	30APR73	212/588-5634
1037	Chow	Jane	Stamford	CT	
F	TA1	28558	13APR52	16SEP80	203/781-8868
1129	Cook	Brenda	New York	NY	
F	ME2	34929	11DEC49	20AUG79	718/383-2313
1988	Cooper	Anthony	New York	NY	
M	FA3	32217	03DEC47	21SEP72	212/587-1228
1405	Davidson	Jason	Paterson	NJ	
M	SCP	18056	08MAR54	29JAN80	201/732-2323
1430	Dean	Sandra	Bridgeport	CT	
F	TA2	32925	03MAR50	30APR75	203/675-1647
1983	Dean	Sharon	New York	NY	
F	FA3	33419	03MAR50	30APR75	718/384-1647
1134	Delgado	Maria	Stamford	CT	
F	TA2	33462	08MAR57	24DEC76	203/781-1528

1118	Dennis	Roger	New York	NY	
M	PT3	111379	19JAN32	21DEC68	718/383-1122
1438	Donaldson	Karen	Stamford	CT	
F	TA3	39223	18MAR53	21NOV75	203/781-2229
1125	Dunlap	Donna	New York	NY	
F	FA2	28888	11NOV56	14DEC75	718/383-2094
1475	Eaton	Alicia	New York	NY	
F	FA2	27787	18DEC49	16JUL78	718/383-2828
1117	Edgerton	Joshua	New York	NY	
M	TA3	39771	08JUN51	16AUG80	212/588-1239
1935	Fernandez	Katrina	Bridgeport	CT	
F	NA2	51081	31MAR42	19OCT69	203/675-2962
1124	Fields	Diana	White Plains	NY	
F	FA1	23177	13JUL46	04OCT78	914/455-2998
1422	Fletcher	Marie	Princeton	NJ	
F	FA1	22454	07JUN52	09APR79	201/812-0902
1616	Flowers	Annette	New York	NY	
F	TA2	34137	04MAR58	07JUN81	718/384-3329
1406	Foster	Gerald	Bridgeport	CT	
M	ME2	35185	11MAR49	20FEB75	203/675-6363
1120	Garcia	Jack	New York	NY	
M	ME1	28619	14SEP60	10OCT81	718/384-4930
1094	Gomez	Alan	Bridgeport	CT	
M	FA1	22268	05APR58	20APR79	203/675-7181
1389	Gordon	Levi	New York	NY	
M	BCK	25028	18JUL47	21AUG78	718/384-9326
1905	Graham	Alvin	New York	NY	
M	PT1	65111	19APR60	01JUN80	212/586-8815
1407	Grant	Daniel	Mt. Vernon	NY	
M	PT1	68096	26MAR57	21MAR78	914/468-1616
1114	Green	Janice	New York	NY	
F	TA2	32928	21SEP57	30JUN75	212/588-1092
;					

ENERGY

```

data energy;
    length State $2;
    input Region Division state $ Type Expenditures;
    datalines;
1 1 ME 1 708
1 1 ME 2 379
1 1 NH 1 597
1 1 NH 2 301
1 1 VT 1 353
1 1 VT 2 188
1 1 MA 1 3264
1 1 MA 2 2498
1 1 RI 1 531
1 1 RI 2 358
1 1 CT 1 2024
1 1 CT 2 1405

```

```
1 2 NY 1 8786
1 2 NY 2 7825
1 2 NJ 1 4115
1 2 NJ 2 3558
1 2 PA 1 6478
1 2 PA 2 3695
4 3 MT 1 322
4 3 MT 2 232
4 3 ID 1 392
4 3 ID 2 298
4 3 WY 1 194
4 3 WY 2 184
4 3 CO 1 1215
4 3 CO 2 1173
4 3 NM 1 545
4 3 NM 2 578
4 3 AZ 1 1694
4 3 AZ 2 1448
4 3 UT 1 621
4 3 UT 2 438
4 3 NV 1 493
4 3 NV 2 378
4 4 WA 1 1680
4 4 WA 2 1122
4 4 OR 1 1014
4 4 OR 2 756
4 4 CA 1 10643
4 4 CA 2 10114
4 4 AK 1 349
4 4 AK 2 329
4 4 HI 1 273
4 4 HI 2 298
;
```

GROC

```
data groc;
  input Region $9. Manager $ Department $ Sales;
  datalines;
Southeast   Hayes      Paper      250
Southeast   Hayes      Produce    100
Southeast   Hayes      Canned     120
Southeast   Hayes      Meat       80
Southeast   Michaels   Paper       40
Southeast   Michaels   Produce    300
Southeast   Michaels   Canned     220
Southeast   Michaels   Meat       70
Northwest   Jeffreys   Paper       60
Northwest   Jeffreys   Produce    600
Northwest   Jeffreys   Canned     420
Northwest   Jeffreys   Meat       30
Northwest   Duncan    Paper       45
```

Northwest	Duncan	Produce	250
Northwest	Duncan	Canned	230
Northwest	Duncan	Meat	73
Northwest	Aikmann	Paper	45
Northwest	Aikmann	Produce	205
Northwest	Aikmann	Canned	420
Northwest	Aikmann	Meat	76
Southwest	Royster	Paper	53
Southwest	Royster	Produce	130
Southwest	Royster	Canned	120
Southwest	Royster	Meat	50
Southwest	Patel	Paper	40
Southwest	Patel	Produce	350
Southwest	Patel	Canned	225
Southwest	Patel	Meat	80
Northeast	Rice	Paper	90
Northeast	Rice	Produce	90
Northeast	Rice	Canned	420
Northeast	Rice	Meat	86
Northeast	Fuller	Paper	200
Northeast	Fuller	Produce	300
Northeast	Fuller	Canned	420
Northeast	Fuller	Meat	125

;

HOMELOANS

```
data HomeLoans (drop=i n);
  input LoanType $ n;
  label LoanToValueRatio = 'Loan to Value Ratio';
  do i = 1 to n;
    input LoanToValueRatio @@;
    output;
  end;
  datalines;
Gold 4757
.06517857 .07288330 .06901565 .07047869 .10615958 .08180662 .07577663 .07402611
.07464491 .09195804 .10319149 .09403409 .09291500 .08666949 .08220036 .09706867
.08137446 .10625967 .10988118 .10192037 .13252796 .11190476 .08284432 .08546917
.11474902 .10060089 .13741259 .10128205 .11250977 .09956752 .10565992 .11426634
.12092438 .10454545 .12624580 .10063108 .12174459 .10463925 .10384962 .10373187
.11889101 .12302905 .10689936 .10830375 .10678448 .14175434 .14000000 .09835142
.10306777 .12897700 .09069053 .09938729 .12606732 .10433429 .12735051 .10419874
.10789909 .22230769 .06997554 .12153911 .12029524 .19348849 .13105041 .10736050
.14354887 .09396996 .10714286 .10797101 .19285714 .14932459 .23832392 .11417826
.18745704 .10577867 .12645260 .10076142 .11188119 .13421053 .10140067 .10351314
.14038322 .15803569 .11037262 .07024115 .09421671 .11320407 .24090399 .09201611
.23026934 .09363636 .10837845 .10310785 .10254265 .11392259 .19275909 .11753247
.14715812 .25000000 .09727273 .16507735 .11277613 .13139946 .15729614 .15459219
.10779527 .09516673 .11557296 .10474698 .13294859 .13418444 .13371444 .13247549
.15376633 .09505782 .11561042 .11290187 .12175698 .14684362 .11671033 .20535845
.10442688 .13454515 .11740196 .09767105 .09713787 .12035137 .12866358 .13301761
```

.12565753 .13436403 .11707470 .19032626 .12227205 .15390960 .16677858 .10112291
 .12632055 .14785576 .11646998 .20606177 .13310128 .21073997 .11930765 .11873847
 .12166386 .19595923 .22040359 .13781390 .13240328 .20940635 .19105401 .11405796
 .11675491 .07318562 .21712771 .12858653 .14967177 .12565169 .16525914 .11471072
 .12470624 .13818738 .13634087 .10173440 .12500000 .15442047 .14260688 .11356331
 .11966618 .13733624 .18856813 .14757208 .10822529 .16280316 .08893247 .14760000
 .17869198 .10313635 .16097770 .22304965 .16425789 .14338498 .11989321 .18025838
 .11391919 .11228049 .12348758 .07523094 .23284231 .15422620 .12077949 .12556000
 .19883453 .12742451 .13665027 .11291223 .14559935 .11427401 .16984845 .08994279
 .11917018 .15007102 .10418889 .14130435 .13643161 .15903616 .21133163 .12840209
 .12078254 .10368464 .13988315 .14798432 .11495649 .13545154 .26146617 .10093433
 .12692308 .16004694 .14122397 .12429076 .15060678 .13610020 .13531513 .13196301
 .13138869 .17387257 .12011086 .15199691 .18807685 .24139901 .13558896 .17240594
 .22357402 .11657790 .13468614 .22405275 .12174472 .12405719 .23138185 .10608283
 .12361268 .14285714 .11046512 .10530503 .14169653 .13958474 .12354416 .12777974
 .18524590 .13051432 .18072931 .10859895 .19188666 .10211788 .17094736 .16861346
 .26935568 .16239913 .14820447 .13070928 .16535018 .15410516 .23510844 .14475322
 .16220149 .09321633 .09241887 .09248327 .15449642 .12819777 .09398391 .14088965
 .12711674 .16898513 .16741953 .07746243 .14403955 .11805921 .12661627 .11073302
 .12888780 .18281491 .11270105 .15105448 .17091051 .15081382 .15447897 .13147457
 .24065514 .10607761 .11262245 .12060081 .19748641 .12127293 .12467463 .13640055
 .12957077 .14664237 .15472457 .08989741 .14296149 .27435897 .19285714 .31315789
 .14167944 .16111111 .16864407 .15235564 .09339713 .12495369 .15332103 .14497947
 .11203203 .15364964 .22668284 .26143538 .08942167 .14805005 .15004368 .14222576
 .18056751 .14229941 .13489965 .11220302 .11326500 .19925373 .23917002 .14650181
 .11136799 .13325239 .27737321 .08696203 .13413434 .14188978 .11176809 .16366290
 .08544016 .19567503 .13230910 .21572977 .09535370 .11183310 .12885847 .13543851
 .11443243 .18668772 .17474083 .15156325 .13895942 .13094066 .13102840 .17976003
 .17447720 .14305095 .17393007 .20059031 .15805671 .12172161 .12156898 .25329611
 .19734774 .32932961 .13756567 .13996258 .14807127 .13111791 .08726659 .14077120
 .17187886 .14777938 .31579932 .17971940 .12636272 .23839394 .28848608 .26171877
 .13951615 .15978670 .22111111 .16432814 .14406074 .33562950 .14205791 .16301757
 .13455220 .18565217 .16828039 .19879249 .16696257 .14111298 .14689682 .17633011
 .18220339 .17420382 .20504184 .17270518 .30725594 .11315789 .13204912 .18790696
 .22233481 .19168625 .29998418 .16776807 .16046016 .13288220 .27302524 .19905660
 .26279461 .11425272 .15468845 .19754496 .17128563 .18067412 .19226016 .14129919
 .12592942 .14843489 .08177491 .13288695 .19260250 .14205984 .14580838 .28374043
 .14216590 .22216793 .11866953 .15030845 .19814815 .13333333 .13547009 .18333333
 .17221577 .11609112 .16241165 .13937050 .21326531 .11480094 .13427262 .16666569
 .14179302 .09047309 .21060594 .11539958 .15842648 .13804826 .13280261 .18788408
 .14535806 .16344538 .14786391 .29559595 .14864455 .27175377 .19365268 .29858826
 .17297882 .26992169 .19682710 .21380016 .11828383 .14448964 .11176094 .15197863
 .21611969 .08323094 .14358487 .08292234 .18892418 .17322303 .15648244 .14196330
 .26217139 .09652641 .17648075 .15934208 .21622408 .15231455 .12803578 .09094370
 .13374012 .18841999 .11538397 .29448420 .28938810 .16416153 .17614365 .21452585
 .17548569 .19350179 .15597549 .16054728 .25029924 .08294501 .14546265 .18086529
 .21598008 .12203923 .15849390 .20965837 .14294409 .12084873 .13811589 .20354525
 .18067015 .19356275 .15016933 .11896778 .17557180 .15949191 .21816817 .15841368
 .21201828 .14177118 .14029539 .15708276 .20272727 .26359915 .11795402 .26480080
 .20465626 .15758335 .14265387 .13975414 .16867424 .11671326 .12130791 .09187976
 .24776640 .14666670 .23815717 .30830370 .18592823 .17705531 .16744225 .19840422
 .15477529 .18186268 .18948146 .35495462 .15337363 .15762355 .18561377 .15245284
 .18501493 .31262938 .16326688 .14670850 .14594979 .20904425 .20663419 .22995020
 .19160341 .15886076 .14178522 .08420490 .16824069 .20495495 .17142091 .15633825

```

.15206140 .21633464 .22068233 .19963548 .13526502 .17977019 .16920768 .16322195
.09870286 .18685055 .15013466 .19855796 .16906554 .14969290 .09313572 .15429779
.17447955 .13962604 .15262096 .18006817 .12530707 .13874110 .31479988 .17893664
.22375674 .12282833 .14064766 .14897836 .15699133 .15852752 .18261184 .19868491
.15543154 .18523596 .18567886 .13045871 .19461316 .09339507 .27160665 .14460535
.17463353 .14811026 .12195480 .16744608 .32311381 .14369976 .14813433 .15177477
.12610090 .11682133 .16370169 .16913367 .15957487 .26035713 .23108938 .14229110
.19708286 .13942233 .18994931 .14729414 .15673371 .16916426 .18184109 .12141497
.28504832 .18322084 .22360776 .14944751 .13800235 .19132278 .13571429 .18846154
.13173793 .12438017 .22394666 .18340053 .09515919 .16275511 .17317968 .29907979
.20850093 .23058780 .15486045 .20470402 .22187323 .18429554 .31405896 .16692300
.21211209 .17222968 .21126460 .15513182 .16834012 .15830497 .14630987 .13035356
.17223152 .40195103 .20333333 .15432850 .18836459 .18325608 .19753520 .15327563
.13815751 .14250326 .21305491 .13528980 .21131830 .18181953 .25258790 .14296932
.21263859 .18735249 .30276547 .19234549 .22995356 .35293160 .18184944 .21802168
.14839916 .18888681 .18007539 .26084611 .28319960 .12745740 .21856753 .20895294
.23586258 .19017089 .14989267 .16996222 .13052220 .15587016 .18428571 .28412204
.35865211 .19394659 .12511227 .10264720 .18276836 .17355903 .12890672 .14893695
.14429987 .14283217 .17575756 .16166813 .12492965 .20101615 .13055325 .19034878
.12845297 .12207891 .21585196 .27401434 .20989767 .10847989 .20087747 .16176471
.23023146 .31950355 .17511524 .21470180 .20915030 .19711348 .15701573 .18821103
.22311334 .20083430 .20778315 .09769452 .14659503 .19050700 .14638554 .20013606
.20930966 .20540770 .21871705 .21133911 .20963815 .14470725 .24742525 .15535920
.23657434 .21867512 .21551489 .13267915 .20005260 .19384862 .12917075 .09619048
.21636652 .17935748 .24834373 .20340334 .18778801 .18042192 .21006865 .22835800
.16160072 .20778712 .35721003 .09808282 .13784274 .17551551 .14013152 .09752644
.19033882 .19868309 .18346771 .14773904 .23647486 .18225907 .13660227 .09668201
.14471711 .26682855 .19966173 .17539345 .18776041 .15871615 .15478482 .18428337
.22083588 .18322458 .18111367 .22052501 .09821131 .16011133 .15162915 .32045486
.20000000 .12368977 .31581463 .09504443 .17578137 .13620915 .09501309 .14135285
.14832256 .16236593 .09907963 .19020876 .16361811 .27429652 .18682159 .12950083
.18071895 .15989011 .18058069 .22706950 .13408732 .25297974 .18795197 .17861736
.17738854 .14523810 .28531073 .21949153 .22170330 .15217114 .16217679 .17841916
.10208333 .14267841 .24607843 .47735043 .18888889 .25161290 .11410256 .10704507
.18149244 .15266940 .14768010 .13912656 .13927455 .19652015 .10128416 .20307372
.11351384 .18808722 .12684800 .19388075 .24348348 .09680607 .23496475 .14528302
.26318361 .16798376 .17014227 .15250061 .10041455 .19675966 .19521092 .17812381
.16086596 .18819525 .23123744 .16292865 .16318936 .10601619 .19196360 .17181590
.18785385 .14176062 .10748958 .19542627 .22894737 .33927143 .13925681 .15391833
.15277078 .20606591 .19852999 .19249790 .16130268 .14928747 .19094238 .17688935
.17911392 .22052864 .23147529 .17163419 .16076121 .21796206 .19657068 .28679079
.16791226 .19037212 .15105313 .13971134 .16978157 .14630637 .20844168 .18733333
.17250529 .16948956 .16464826 .19909170 .21141165 .18562268 .25376672 .23363999
.18953425 .15651830 .15367910 .21421147 .19501528 .17159418 .15597907 .19247379
.17864708 .18852465 .25175112 .21970376 .18718326 .17450467 .19208233 .18085547
.16977289 .30924818 .18547488 .18408282 .24877676 .17675662 .24245004 .15398232
.23806850 .20008514 .24477990 .21596068 .19267467 .10850038 .21143911 .19307516
.16989039 .31923077 .14133453 .20748268 .23001337 .19708560 .20450265 .16797753
.23203252 .16905032 .24533430 .15265234 .15699874 .24488836 .23822601 .37588454
.17072018 .20939850 .21476513 .14950622 .28705161 .13523710 .16429557 .26105027
.21419090 .17247822 .23375342 .19492159 .15571246 .10287864 .18020354 .20893960
.10252205 .21962443 .22214499 .18683599 .18225698 .17920369 .14847640 .12732147
.10212918 .15457877 .23855171 .17533106 .20370561 .19081326 .17889419 .20595804
.14995890 .18497830 .10797761 .21619294 .10252307 .11067479 .18707053 .16534398

```

.20470029 .20706422 .17505844 .21502668 .20814132 .10909970 .13615276 .13439750
 .20199080 .16126015 .27365286 .09778761 .13852894 .22540440 .28086789 .17875229
 .18411651 .23703890 .10848365 .21062107 .15938814 .20212125 .17549384 .14987516
 .16370815 .17475886 .20465246 .18782366 .26820386 .16720276 .19338440 .18148604
 .21124457 .13811220 .23873200 .19862113 .21340849 .19303148 .18120832 .25819406
 .21344034 .20399178 .20837268 .35453733 .19170935 .19319496 .13707531 .22864752
 .36630100 .14771958 .22887342 .17414369 .17704132 .15685629 .15123973 .21123831
 .17864745 .20876452 .22240028 .22888506 .20711143 .13029197 .42446809 .16224490
 .15297697 .14166667 .17716763 .11662790 .12675240 .21741496 .15123227 .19688798
 .23082885 .16215220 .22073587 .15256888 .23162905 .13933067 .14936407 .24348824
 .14715341 .25979021 .20599309 .27084278 .17597603 .17819921 .21204616 .14744279
 .28252404 .16084592 .16671311 .21565928 .20551446 .19364283 .14031660 .17802325
 .17336623 .23076409 .20439182 .21462248 .15783399 .29368825 .11097126 .15319726
 .23435782 .23865680 .28718763 .15390189 .14838975 .17821834 .25321515 .18975543
 .18323975 .18007828 .21789087 .20341835 .15818852 .11158956 .28260644 .20313098
 .20059037 .10761661 .19150294 .19157385 .22032180 .19005776 .20836755 .26356618
 .17665890 .10791753 .22667845 .26049494 .44510490 .18776960 .21350035 .16080169
 .10267500 .16262608 .20453837 .16198208 .43809832 .34807039 .17565168 .17729892
 .19605835 .20643654 .10837652 .25141343 .20402075 .23775940 .30418627 .46546704
 .18001232 .10638373 .12871297 .10686353 .20225376 .23568287 .19136379 .18899297
 .14413190 .18135801 .10348489 .23156844 .21051364 .17906991 .20202189 .24713385
 .18239391 .20617361 .19180200 .35490230 .14675167 .22958368 .21460552 .36146742
 .17262286 .19711291 .15689766 .15831808 .14029908 .39779480 .28200000 .14754949
 .29658816 .25525887 .27384747 .21313902 .21008832 .10837800 .16460299 .36803476
 .24848338 .17809004 .18750147 .14921392 .23382931 .19568003 .24797959 .21925859
 .17753851 .19224578 .18764706 .23835727 .18568836 .20383604 .20704698 .18450286
 .16357459 .22223359 .19039888 .15577036 .15873606 .18570567 .14415520 .24150248
 .16538875 .22941583 .43769878 .25653924 .23005646 .20127429 .19352088 .14398496
 .13589002 .27699292 .24826934 .28026187 .17880127 .21908836 .15702948 .13381813
 .23652038 .26915689 .18691853 .15878906 .23556058 .35177770 .18678161 .14691501
 .22445867 .19698072 .13253686 .24850537 .27647686 .16290001 .10235487 .15730194
 .16029241 .16401635 .10695851 .13268770 .16284541 .20900163 .14723291 .14487741
 .21171635 .25017682 .22372170 .19308881 .26786888 .32805455 .23350027 .22465469
 .27607811 .17799044 .14898622 .17227555 .27668027 .19185236 .23750000 .13888889
 .24933555 .16857708 .14725655 .17889366 .17644889 .11854597 .24591837 .24047619
 .15702341 .16320755 .22699115 .20000000 .15105263 .25168067 .16401425 .27222222
 .17121212 .20000000 .15801080 .15540184 .17252900 .24059720 .15350985 .22792011
 .15505673 .16922030 .20797996 .15451141 .18964546 .19550614 .20770815 .19686961
 .14578241 .23789403 .16816406 .23220148 .21127098 .23454968 .21311015 .40653250
 .30208333 .17855109 .21636647 .18362045 .17236931 .23093729 .21627046 .15707586
 .13361435 .20173555 .15637924 .39865293 .40130505 .21677463 .21233113 .19680754
 .13853796 .21718971 .18789518 .12488981 .16888752 .15696781 .10254634 .23845774
 .19921722 .10305986 .14755551 .18425330 .18866473 .21042078 .16042823 .15934643
 .19435095 .13964492 .16592072 .19331488 .11841201 .21844008 .14030704 .16821808
 .20957654 .29341975 .13524854 .24097303 .11912442 .30416899 .15155050 .11915824
 .18659078 .25570282 .26709969 .31579078 .14902026 .12808978 .15307982 .20609137
 .11257317 .26586241 .17711691 .12722681 .28659156 .10288884 .38813264 .26407731
 .24697234 .22880317 .13446119 .13565902 .35437664 .23984338 .44472847 .19144756
 .17831524 .15560197 .26898705 .11904170 .25194840 .14681217 .23764543 .15511887
 .23112267 .17485766 .18205155 .29026448 .18804528 .18383298 .21927119 .17399318
 .20645926 .19929473 .17370359 .44255095 .23777509 .18933787 .39720293 .36887755
 .22129389 .18951016 .16428260 .18432549 .18865047 .21241718 .16062335 .11344347
 .17182623 .14891352 .35509213 .38261179 .21333727 .21956680 .22454183 .14087696
 .17977115 .16755267 .26044193 .18370686 .23557794 .10499740 .18955431 .12196813

.31260287	.20190661	.35679293	.16804071	.15232363	.20376420	.26630276	.37921171
.17901784	.11310716	.22269278	.17452323	.15459906	.18547967	.17185761	.24586071
.24251467	.19319540	.20381448	.20299362	.11930650	.14787903	.28733438	.16167433
.18849366	.11093149	.25347821	.23265361	.21328197	.16320121	.17401539	.10620518
.21272565	.24598536	.18472691	.19712305	.31496098	.25469847	.30037654	.16195564
.19669478	.17157015	.18112604	.16225805	.16575016	.37599837	.17588513	.19202181
.19827586	.27323360	.20389571	.28996428	.22279486	.16455669	.22740250	.12185789
.15960627	.16287571	.18172809	.17804733	.20271333	.16626335	.17569669	.17799905
.35214311	.12286324	.24974606	.15558366	.27303676	.25561046	.16726526	.21538886
.19205171	.24497597	.22627526	.23076341	.23029350	.39054011	.26919102	.33952585
.17708344	.49856975	.21868877	.16811310	.18429336	.17212306	.22948364	.29459538
.18777754	.23546259	.23011777	.35457804	.16594410	.16111681	.31756128	.25072880
.32253669	.20627442	.18113828	.56130777	.39904014	.34399792	.13915896	.20199701
.23044278	.55876644	.16363636	.27525254	.11172986	.22644584	.17172399	.27057077
.23698526	.20102083	.18283848	.23053775	.22432035	.21054315	.32539522	.17616413
.19987358	.24868655	.26873800	.17293890	.22827980	.15626131	.23097172	.24107909
.27000537	.12187742	.24091974	.19948529	.21655731	.19391313	.21220002	.47844623
.15498020	.39341910	.16691348	.16545223	.21401590	.43286394	.19947683	.14498521
.15649201	.18850992	.15920437	.23446346	.17227861	.20083812	.23323850	.35093902
.28087071	.24025276	.12307130	.23397610	.27359705	.17818042	.16454556	.16485320
.54012382	.16559588	.21633941	.22021168	.19421409	.22308468	.25851624	.28584069
.24267544	.24681570	.12431522	.24408476	.18084787	.16054646	.25450522	.15333207
.20769715	.24319214	.16739249	.32922484	.19557306	.18742744	.21352631	.16288203
.20274485	.23842190	.16425745	.44805133	.16598719	.19386036	.24470405	.18140093
.16758046	.22355085	.19432946	.20381635	.25042460	.23301611	.25187822	.16748734
.17948274	.15705704	.14963100	.26496815	.16553471	.14272169	.20764766	.17149028
.14370770	.14919250	.11726223	.14730010	.17069256	.21966915	.21108972	.11385126
.43422421	.23914108	.16807298	.18029566	.17253798	.25029750	.22891441	.22918786
.20539483	.23518519	.20542857	.17264959	.31145801	.18342097	.27983455	.12448586
.15375422	.29040586	.21737432	.31746381	.14414957	.17494300	.17611850	.17158756
.25285477	.19363297	.17974714	.18967050	.16388954	.25566865	.34900260	.20255615
.20700206	.21163475	.17835287	.38353621	.16061054	.17042262	.39127992	.16380628
.22463575	.22326639	.21313602	.17175397	.26066876	.19445382	.17016384	.19891304
.21899393	.36039834	.19669771	.15992425	.20432616	.17690005	.15866142	.27543126
.36069185	.15162826	.33807014	.37031939	.47649195	.18574127	.19425037	.12470645
.15606899	.22090207	.36298195	.23695360	.22102851	.20790430	.14608206	.31127820
.20306179	.21363144	.19706194	.12774528	.18799265	.22375000	.18229905	.39001125
.17170670	.16841780	.22157952	.19801090	.12156353	.18707952	.17851331	.19371678
.25391397	.26855002	.22080784	.55037798	.25112573	.24891811	.17388065	.27878434
.17240009	.29507659	.28969730	.18777493	.20819209	.34090909	.27969647	.16200000
.33865979	.28333333	.26760080	.19372094	.18244783	.25371933	.20768785	.26216943
.14840047	.15401759	.31443615	.20484806	.25560829	.23262593	.19346034	.16982969
.18001143	.25808430	.22616852	.16037099	.18679627	.39843206	.49835970	.16684550
.24466455	.27886983	.20968692	.21427010	.15993605	.23449945	.24022613	.16480221
.19285714	.25278798	.19300000	.33718319	.22480594	.27584058	.16255500	.21009135
.22694078	.19300000	.42631579	.39181905	.17179542	.17613344	.39153332	.54115576
.19012464	.16594062	.25394193	.23586877	.21135220	.16799751	.22349398	.26039098
.16048876	.25286833	.25656415	.39296330	.34710937	.26395459	.21208185	.20871441
.18547841	.27526751	.18618695	.20585934	.26008097	.19086849	.32676873	.15133352
.17109897	.18934046	.28330652	.21359780	.27951391	.52346939	.20820713	.33035576
.23053238	.26323529	.16167591	.21208724	.23484524	.46213086	.21011838	.31370828
.26974053	.26864143	.33914899	.24722534	.22571097	.27954892	.23162818	.21590909
.11188748	.34648282	.36068457	.18713942	.18909663	.32832320	.23739090	.20362923
.17320155	.20240720	.21325618	.26431383	.16683264	.38261681	.26473647	.39385161

.33459141 .20224957 .17740510 .16895129 .42593985 .19508488 .36919745 .22940613
 .26598272 .21265380 .20597644 .17800975 .18151067 .27185331 .20433869 .19002555
 .26131084 .17741047 .27336925 .26318890 .22356225 .39042553 .19964611 .30084746
 .20855840 .29390646 .28876359 .34058923 .29331700 .21951094 .20882044 .18804940
 .43169908 .18522033 .22614375 .17691112 .21792327 .23306917 .24057120 .20235953
 .19778227 .17624230 .38812917 .18045115 .22199982 .52255082 .25599743 .19416751
 .19975777 .15651683 .26262933 .18083259 .15283380 .23317720 .22832992 .19424427
 .23328761 .25711128 .31096856 .20399778 .16165117 .18869754 .31173443 .22170330
 .21666667 .14169825 .20860428 .14615385 .35737705 .26428571 .20090543 .14740260
 .13527572 .20957447 .22985612 .22045455 .18111888 .28076923 .14433962 .19285714
 .16538462 .18452915 .25270270 .20789474 .22357093 .14740260 .17679628 .16053795
 .30597270 .18043478 .12396450 .18516558 .47492918 .15677145 .41667644 .28090422
 .27212679 .18438091 .21201853 .27052338 .23907642 .18991624 .19098804 .25768432
 .13562906 .21141446 .33363199 .18770336 .19199336 .28739526 .23714523 .31061443
 .18142320 .27892315 .30228476 .22334405 .20128441 .27049268 .15542558 .32929344
 .38072694 .14216702 .24749922 .29092925 .51765152 .18859950 .20390260 .27844776
 .30292023 .22600741 .19343547 .25802540 .13231344 .18272791 .16601105 .29243971
 .21955959 .27651744 .25399946 .43378993 .13539422 .28929848 .34867170 .29448698
 .17779445 .30835401 .20979143 .21521739 .17466680 .20151515 .20885956 .23001161
 .27748702 .20919231 .18304740 .23703319 .21224642 .39412181 .14091936 .22605836
 .36812039 .20356820 .21572790 .20783284 .20315929 .17633352 .32833364 .21109842
 .20000000 .18600000 .28538462 .18509576 .14771488 .19845289 .20925390 .18015517
 .21084438 .19986189 .33768051 .20488809 .31383404 .40812930 .19636386 .18544892
 .37532004 .16896118 .27181572 .27106427 .31953706 .28489575 .21891151 .40537915
 .17920005 .21887446 .40080526 .27078537 .17183544 .25684746 .32787301 .22870405
 .31930610 .40749106 .15611542 .36899909 .32096456 .27728950 .16281917 .13635146
 .24174388 .14774246 .29617633 .21715375 .44552924 .19507539 .18449141 .22476013
 .18400306 .21538096 .17989625 .22573696 .34343834 .40606809 .27116949 .29495859
 .13930320 .16568373 .43997635 .17671474 .28340365 .19580529 .18839480 .30159261
 .18235144 .16338610 .37762785 .19663170 .19603460 .21092096 .19126468 .17813457
 .21875264 .17636183 .20119356 .28430108 .16023333 .22731097 .21832490 .20833476
 .13631401 .21165235 .34734848 .26745755 .28901592 .17321360 .20498213 .17842746
 .60677708 .21796474 .23690476 .50780603 .12685454 .20236062 .33777242 .16920128
 .19127980 .30890501 .28100465 .21849109 .22624211 .17793455 .19442281 .15956852
 .31261115 .22382312 .24815391 .56982234 .19133391 .40791954 .15022583 .16100183
 .21183882 .45218913 .19216432 .16435106 .20652243 .21359664 .22606615 .17302038
 .24637819 .21348999 .41081297 .31232339 .21490111 .18865392 .27231233 .15162991
 .21235258 .20017426 .27563753 .20092262 .18961085 .36604683 .21494974 .18170538
 .28072905 .18277551 .18193488 .29582940 .28924885 .22710944 .18200389 .17242258
 .22231850 .22625150 .21489671 .19520548 .24367806 .18914779 .14279253 .18433706
 .35439943 .25739896 .35892802 .21786445 .35486643 .20495618 .14579691 .20964081
 .14942891 .19961241 .34629630 .17307692 .22937220 .15505581 .30236593 .32671129
 .14996876 .22582418 .17480499 .22600211 .52930022 .18821819 .19725914 .24019543
 .30063834 .21910995 .39677070 .22482709 .23244430 .13831679 .22821738 .23685695
 .23651932 .23372643 .20601985 .15098666 .22683563 .22172234 .19043106 .17920311
 .29988359 .23919586 .20186673 .18820810 .25281165 .32824825 .27463131 .29571525
 .59163162 .14577347 .27130888 .21173752 .22322445 .22581794 .28323447 .48277322
 .16743385 .22495545 .24630891 .20914025 .18916689 .19855571 .22002519 .23475856
 .22162108 .27040816 .19116663 .17549287 .41107520 .19332478 .18763222 .20524974
 .17346808 .14859964 .23387122 .21099879 .23160346 .17074790 .15106396 .24047174
 .28327370 .23341398 .24583814 .20913928 .25655397 .19211976 .43236881 .21533955
 .25811919 .22766607 .25719105 .36542101 .22313092 .17758079 .18220689 .14991897
 .23333650 .23513083 .22374722 .26339163 .20042421 .22858698 .36873129 .19152449
 .26442674 .23305819 .24037890 .20501531 .18411732 .24198352 .19126119 .17273061

```

.21387218 .27721639 .19643627 .32691068 .30095820 .23653764 .21949501 .17880462
.21724442 .21369697 .44590182 .43466918 .22647059 .25625000 .13549223 .20000000
.14844223 .16896779 .15423510 .30416680 .27381986 .15060976 .21679808 .16421768
.29639369 .18964589 .61568843 .26669184 .36108597 .22133362 .18426293 .18436291
.20558951 .22483991 .15115598 .18892027 .25355111 .29211309 .21563395 .37720965
.34312567 .24615489 .22387479 .42586324 .42830447 .21532053 .20894750 .23193703
.20474569 .22487277 .61827849 .17152790 .17812546 .32613347 .32178335 .19849328
.26772007 .23013937 .25085151 .22299425 .34623585 .41939547 .34575843 .24489992
.22903663 .17228340 .25072884 .23578111 .23665008 .16981002 .45115571 .22546242
.23723044 .21738736 .21300077 .20091763 .30012655 .22915796 .25255118 .32566743
.14139873 .19111362 .17843153 .15128312 .37784341 .34669423 .22480890 .32108530
.27495840 .23576940 .24147104 .22425375 .13072066 .23836380 .19649157 .25957601
.23435492 .18915191 .17306303 .19364641 .20875091 .15558416 .14886480 .21832850
.30115235 .22729387 .15311217 .15640295 .33197515 .15714668 .17727981 .27558386
.15214075 .25833844 .15706638 .31350053 .33302672 .32878450 .23315808 .23261701
.22545851 .24996471 .27972973 .25000000 .20081618 .19214047 .25000000 .13893725
.24169401 .13156255 .19602804 .16995653 .20719200 .17317769 .20266216 .17676823
.18236473 .23115174 .28900707 .24220178 .22805823 .19299452 .18256944 .28205949
.15655069 .20365124 .23752673 .23143814 .29751759 .29869833 .18067001 .19761995
.33133792 .27965659 .15113795 .17996390 .20099204 .31713882 .22799303 .24171822
.17849341 .24412435 .34741834 .19567136 .19209357 .31139818 .15117647 .26657286
.25080790 .30680074 .27470736 .33434922 .20745437 .30247336 .29620670 .21809022
.19482869 .19394510 .31028268 .23505369 .44713692 .19488115 .25894326 .31086039
.20664614 .31397302 .32089662 .20055785 .16068033 .26847020 .50550290 .19907111
.23960151 .25632819 .17577062 .21043177 .23806188 .18944191 .40234216 .29156276
.31974351 .26929547 .40531233 .22181622 .24240551 .24667684 .26156816 .44831517
.51535262 .28636808 .15450702 .31329727 .33405381 .27273140 .16215242 .17874965
.31054925 .25906175 .15342184 .13571893 .27120237 .20013201 .20193720 .25483119
.20313802 .15709849 .23527733 .19930594 .16325908 .30725836 .23564291 .14019363
.30168268 .17630819 .29002853 .20409813 .21062709 .25071569 .20346704 .18398360
.17592012 .23439686 .39555200 .26888642 .23297501 .21924379 .25272527 .23233429
.22073952 .23311207 .31631158 .44007957 .47516185 .24379845 .26346529 .30630944
.22970369 .18087837 .22394226 .25031413 .37257473 .31802711 .31666667 .20314735
.23041845 .23708184 .48112897 .24919870 .18783329 .20171255 .21592765 .24623930
.35825496 .36381866 .18265084 .23325258 .17984250 .20649314 .29686881 .36722704
.24572068 .23187984 .20913259 .25730457 .32658219 .31469976 .21428545 .23611654
.20738285 .28955319 .24790312 .24457167 .25784195 .15873416 .20389936 .14765196
.15386397 .15267651 .25142810 .35890979 .20287631 .23835182 .16289760 .27347491
.38637764 .24922536 .23944205 .15693973 .26953493 .24524433 .33154393 .23673065
.17508473 .21052804 .31312308 .18689620 .23149924 .23200570 .34148853 .26051893
.18828496 .23949164 .22947593 .38729973 .22531728 .20938654 .20632328 .40467980
.34380080 .15434783 .44130435 .32190332 .18709063 .14230769 .20126050 .19925373
.31277372 .28076923 .27511537 .20681355 .20149347 .14143554 .18255567 .25039411
.27958012 .30205848 .33641897 .23551728 .15209059 .22395817 .14856777 .22721990
.30069252 .16031944 .25124304 .26030372 .20207656 .14922266 .20128340 .18110527
.25832134 .28296522 .50565541 .37475688 .24584717 .15572800 .30277778 .16472010
.15686395 .24178486 .37122560 .16441072 .18296609 .89611808 .19654374 .25230539
.24257221 .65268892 .16663078 .13717227 .25125174 .24758555 .22247588 .24495265
.34180228 .27960954 .33000918 .22619194 .20926893 .24215423 .13685953 .35729447
.25879685 .18129197 .20437176 .35373948 .35247934 .24236429 .16953518 .23475331
.23836656 .24558805 .19658763 .24174350 .24507644 .20494998 .25610242 .20060734
.24220926 .24087731 .19690736 .18340487 .19244022 .21483615 .32735906 .35454500
.35794979 .31452364 .23084427 .24192857 .25158654 .31739514 .24654714 .26624275
.24442985 .20987003 .32875299 .30827889 .25373327 .22422095 .54282080 .24281076

```

.32205882 .24321149 .24473684 .30879555 .19513219 .36136918 .20069196 .25587291
 .21067865 .31520660 .25739717 .26424622 .18799755 .28332832 .31125797 .20040999
 .17097482 .51386353 .34226442 .24312235 .36421042 .13751706 .31719915 .31488180
 .16950348 .31785329 .24561445 .94855072 .23371459 .15775675 .20528469 .33874512
 .25746658 .27093834 .22893025 .34327813 .34249370 .15497656 .16503020 .24886213
 .19065121 .28667591 .25860990 .22003091 .27489206 .24797997 .31025357 .25887787
 .21303969 .19360754 .35821466 .20580713 .23195544 .23829940 .14432874 .16516080
 .20334796 .33842761 .25813728 .16373123 .19932249 .36028222 .21112030 .14276666
 .43226609 .21411236 .23149866 .36900349 .21067876 .21074572 .14507520 .32600508
 .24251337 .17229448 .21758885 .14192249 .16728422 .33859809 .23450185 .19825274
 .21667990 .20734397 .26517128 .21070062 .17066633 .25585272 .71343099 .29163498
 .14699222 .16296949 .14491268 .16212019 .23936563 .19315529 .27028986 .28750000
 .30099075 .17025316 .14984235 .16509153 .13755760 .16010982 .33947529 .33736065
 .38420238 .15801396 .34652399 .14666383 .21470203 .29892156 .35896150 .25530349
 .24445598 .28503489 .25120303 .16937403 .22490799 .21616903 .14609032 .30074112
 .24793814 .24885657 .19902551 .36892628 .22217385 .21170790 .30348043 .29717605
 .14857450 .26670540 .40864273 .23733135 .20201040 .24069639 .23775780 .20931980
 .15007570 .21520154 .28108791 .23193130 .23255215 .29588086 .14656449 .30492773
 .23729846 .28429952 .16233027 .16183232 .29122153 .24251570 .20268012 .30086194
 .17776245 .20897733 .18919424 .70000000 .19092141 .16424420 .26851188 .26687631
 .31872089 .32486081 .21047269 .26777778 .27737028 .36702899 .29483168 .28190052
 .22388064 .17024983 .37184809 .22402827 .22377957 .30938117 .19606983 .26927628
 .85549536 .18369256 .25227119 .19786903 .25674533 .28813533 .34086422 .30411982
 .21258694 .28134355 .25800941 .25808374 .17892693 .17283041 .13873233 .26747683
 .29116535 .20305455 .14991364 .27098589 .13954807 .23494768 .89193603 .33190163
 .26152661 .24493559 .29789291 .67656642 .37553143 .19285714 .13919294 .33475426
 .19372979 .16111111 .25000000 .27222222 .48010753 .22006803 .24005987 .26231423
 .32100271 .17271821 .44880359 .40477977 .16665209 .35802953 .25000000 .26451402
 .23761726 .25921596 .25356234 .24047619 .22021277 .27075055 .16299435 .20619875
 .43461538 .16111111 .20151515 .12207207 .37258065 .47062758 .39297669 .30564986
 .44163468 .26809299 .16526182 .18641355 .80451767 .20377163 .16065252 .21834171
 .23859427 .31949480 .40704770 .26319700 .92501632 .24941070 .28016409 .47872683
 .34202383 .30135368 .30130654 .29254857 .19394466 .24646557 .14989365 .12004583
 .23161119 .16203301 .80658264 .19511494 .26825333 .17137745 .25658621 .32873987
 .16563672 .45527256 .21591103 .15255473 .35976369 .25582212 .34249928 .34692346
 .37070110 .25651018 .33815155 .16765665 .23797341 .20708427 .23907817 .41890981
 .22170794 .40057422 .15135860 .27418306 .41443280 .21117763 .70881284 .14302453
 .42751212 .21570008 .20881419 .26599644 .15416986 .37159917 .25692383 .34701240
 .22765083 .14995982 .24723484 .23024227 .25335739 .25380435 .25763993 .23650776
 .26602824 .24763227 .36536521 .19777011 .20721341 .14280805 .35866521 .36336941
 .15305081 .25930328 .27699339 .22265920 .13913043 .20471698 .19978044 .25396995
 .16041332 .40314993 .25849861 .24073014 .26001550 .25234365 .24343994 .87750864
 .41472442 .36105608 .17351527 .24463709 .19898603 .26328584 .42438209 .24431396
 .16705277 .76939934 .21370124 .13803268 .16873062 .25712842 .19691062 .24905569
 .19549697 .36846154 .31217798 .36855956 .28861947 .19087670 .26408404 .16273220
 .25544682 .25349980 .15831190 .22968210 .26183851 .35369397 .16419787 .15406192
 .25800624 .42871202 .21408573 .19148312 .21793967 .82809367 .32924711 .47413491
 .26423422 .25958446 .31502555 .20942118 .19797530 .31136564 .16552154 .40570835
 .25541550 .19464969 .64909419 .33154594 .15235465 .27662214 .42602101 .29058939
 .33314412 .24805921 .24822828 .27875531 .28775937 .16957411 .33300993 .22721777
 .31561944 .19576956 .24115238 .31234259 .20014156 .24961977 .29192434 .16095671
 .22796610 .29676851 .20671642 .36343284 .20000000 .25873507 .26485354 .17084384
 .20605841 .27962113 .15516458 .29330055 .43761114 .30285208 .30919465 .22037839
 .30202456 .27066283 .21599795 .16714283 .84773157 .15372632 .20214037 .42498894

```
.19231244 .17137451 .24515428 .37765096 .24690342 .25271756 .22832509 .34816320
.26855895 .28389490 .25827398 .27277098 .19555441 .12527247 .30296912 .27616267
.18719807 .17036074 .23579416 .18316745 .16833662 .14026648 .49807203 .19392767
.29521937 .20014909 .14243067 .29034664 .22031552 .25717829 .21966839 .28690884
.31073545 .16705869 .26190024 .23608696 .25317099 .20760441 .18120543 .24729139
.27254807 .25381535 .19511721 .22582201 .31447507 .35808655 .26541528 .20995097
.91394832 .25690922 .27562178 .16566692 .17751454 .19635408 .74789334 .18393469
.17075599 .16669498 .50372051 .12549422 .16498802 .20009774 .15718167 .24282511
.55588235 .16252479 .27238772 .45337711 .32338038 .20036437 .25717300 .32280802
.33420733 .29854917 .28676589 .47074917 .33559340 .18894249 .16644581 .36625183
.20539345 .24815241 .37759536 .27382724 .38021969 .16039502 .19656190 .25519251
.27015880 .20134953 .44843577 .36420924 .19615135 .20889831 .32674689 .32678924
.16646629 .27466559 .33730306 .45329325 .27421061 .37376945 .49870867 .45227648
.27142857 .20451658 .19739044 .35295415 .36844330 .46698693 .22207882 .28059839
.20548088 .17546689 .28226690 .14262856 .14413786 .48933012 .32211224 .22819921
.27118282 .90269499 .24000802 .27018868 .41097496 .35983073 .18521979 .22635686
.18062261 .35457988 .41493907 .16343330 .34285905 .32293461 .26446829 .20743390
.20433730 .20119089 .36761686 .24239723 .25881405 .93277975 .30042882 .33274482
.26246665 .16626364 .28780825 .23424728 .43396802 .20565253 .20372089 .17122281
.26991204 .27445544 .29877591 .35427086 .41596523 .27494888 .29719101 .32163847
.27222222 .22745084 .26168501 .23128332 .16969597 .28147661 .14485498 .30880526
.28979770 .27769142 .26633792 .22491373 .27602915 .23743904 .26721823 .32394543
.50102041 .25898345 .20621466 .17951015 .38899865 .47223921 .37882990 .31081620
.17304964 .31013288 .41450808 .18649107 .17323475 .19969757 .27424922 .24108938
.26723176 .20751607 .27926542 .25359128 .70436538 .17132938 .26285532 .26659317
.21326590 .26179197 .48191106 .16959477 .26608737 .57359709 .29537042 .20304477
.20521466 .24961867 .19909806 1.0455802 .12930694 .21905660 .26880769 .26155826
.17316964 .21523926 .26461489 .37317636 .28398166 .18352090 .18642814 .23515916
.34096199 .25124520 .23420901 .26880556 .30371799 .25817650 .22982435 .27212306
.19248095 .26315875 .44419963 .44548721 .39092293 .25162742 .42458820 .22784835
.27026648 .63468847 .30900175 .18394850 .20706368 .28509007 .30787966 .57570093
.28488882 .30865339 .26317921 .23987021 .34260680 .30522647 .20999204 .24604597
.95685583 .17444415 .18072500 .27793287 .25156435 .23759857 .17613508 .29675991
.29885483 .20603317 .45811904 .13692308 .21243451 .52410267 .54405386 .37110027
.43398804 .13545651 .31456265 .45780238 .27203446 .17832233 .29418842 .28078018
.22158363 .26377583 .46498327 .26368125 .27883526 .21858146 .30661026 .18824688
.36895910 .16485878 .21265522 .44715865 .27510070 .23633236 .25780925 .26586003
.32834070 .18274491 .32011978 .20433561 .28352521 .35404865 .20527104 .18380596
.33022910 .44473001 .28734424 .28657342 .27401038 .20471091 .14828411 .29821696
.24851762 .27049974 .43171120 .45098759 .23605483 .33182881 .29590340 .28465279
.17658998 .28778373 .31003225 .21488699 .48367927 .31366996 .18109386 .33247542
.27201970 .35833446 .19760527 .45787374 .27510179 .20296215 .21343585 .30277781
.25648358 .30564929 .54097041 .26820597 .17475591 .40341119 .19076749 .13214286
.38093525 .40537701 .49184885 .51086643 .27604423 .25073516 .27760415 .28985546
.17850753 .17592467 .22012815 .26470596 .18267856 .41141185 .17433593 .19789585
.28060797 .13658398 .14766529 .28358840 .22174880 .19586608 .21560782 .33413614
.18555836 .18136215 .26120186 .22257597 .26960001 .30900688 .18161760 .27660300
.51435618 .22154174 .27273418 .22099788 .29433539 .27284653 .28502471 .29723707
.26953279 .38934191 .19119180 .24472090 .43267183 .24564563 .25030988 .40351704
.24219288 .31051771 .24694938 .50536458 .38480076 .30162216 .18906148 .33673223
.22911176 .18096738 .27994579 .44429530 .21574391 .43594186 .31980172 .25996953
.13224001 .29178447 .40489376 .31381077 .15059802 .24881557 .21912193 .29153845
.17854521 .46201117 .21975364 .18687984 .39600047 .38579488 .17471332 .34435242
.27388555 .28210527 .18485714 .25653563 .23838254 .18911977 .27173991 .17925062
```

.33209756 .28481854 .35046401 .28313921 .35071563 .17701986 .18099206 .55787528
 .36542802 .29159268 .28868271 .27175024 .21876976 .21468283 .47665034 .29196512
 .17783603 .38148556 .26574582 .39523274 .15179641 .39927064 .47752699 .21124552
 .40375080 .95188679 .13950104 .42878788 .52951527 .22561391 .29969180 .81612386
 .22760669 .46216845 .27104660 .60065318 .28726794 .24792797 .30892422 .43377545
 .18022181 .25504637 .20750834 .40134977 .24601277 .52428955 .18293527 .32871004
 .58333333 .16709430 .21000000 .32272727 .14007995 .19652372 .52642680 .31172609
 .26327434 .15095129 .29823352 .28604772 .19934993 .26463623 .31561742 .53752857
 .39467043 .22063042 .24948891 .14074887 .40874174 .31419791 .55816849 .32071168
 .18209030 .25754539 .20294772 .18885714 .29706923 .40631021 .22706079 .38147363
 .22534871 .14267381 .21860598 .29715216 .55128932 .20863276 .45080491 .40332606
 .21827586 .14334603 .39208645 .28356435 .20756167 .18401513 .49545869 .45742720
 .89429066 .20417959 .22424713 .30292051 .21504664 .25604106 .28497236 .13415795
 .20284581 .24776782 .47946291 .21389166 .51909704 .27778196 .20432293 .30883198
 .19272565 .30504893 .30136713 .33852382 .13707984 .52459466 .24932149 .20318819
 .73930591 .19893205 .26722813 .30975672 .44870340 .19949046 .21757037 .13483022
 .32547595 .18841102 .20067774 .50720421 .23555245 .34738971 .38289756 .26468553
 .27019755 .46378850 .43900130 .16701646 .22669481 .59907344 .25378648 .28272618
 .39020138 .31606016 .37121157 .31384386 .79844450 .19695121 .21960120 .22353851
 .35000000 .19536241 .56562403 .46196342 .31622084 .21859995 .44923985 .19285714
 .25000000 .17500000 .17345679 .16061947 .15582011 .49883303 .40211268 .18513514
 .31041667 .15361449 .25833333 .22636684 .28086158 .17065637 .25491803 .15584251
 .18616558 .55481594 .27123894 .30252525 .40211268 .25408163 .27361360 .19009291
 .31766595 .59628092 .18541842 .76428571 .26371905 .28970698 .34633142 .28994570
 .54339519 .18390880 .27497490 .40070070 .56886305 .40985663 .46985180 .29906724
 .40175246 .30362758 .58335033 .21714168 .64035209 .24606005 .32122234 .38171222
 .15294944 .26813757 .38283828 .51085488 .19921292 .17430789 .24804003 .29389300
 .30101602 .39738975 .38347007 .54447638 .17143935 .42013645 .29651022 .31920776
 .32796775 .32759113 .26966144 .33459583 .16821657 .15285055 .21439246 .21476503
 .79466519 .54913194 .21113726 .34072106 .34210733 .45456697 .30008649 .30868742
 .34150838 .17053682 .28645063 .37564103 .17115488 .31368516 .15438628 .28439302
 .28989422 .60985364 .19250428 .85246719 .19252658 .26687915 .37083139 .36979734
 .27553797 .18816570 .44628887 .26640797 .14532496 .25951786 .62038048 .53730150
 .64259604 .25079531 .54255397 .36761787 .14668696 .15363324 .33600795 .24873616
 .14698513 .21732026 .29723789 .22344408 .17119491 .29850991 .17070404 .26442152
 .17173202 .22873536 .50299914 .52156829 .15246637 .54613900 .35235294 .15593482
 .41738428 .27081694 .38561867 .31176677 .22773905 .29564154 .17378791 .45658094
 .42438509 .29136512 .27178286 .19308453 .20440226 .54783884 .15812120 .21739052
 .81428234 .24060375 .17266103 .26164972 .18932746 .35173468 .33336670 .34952931
 .32345482 .19518894 .14981611 .17521250 .23919273 .40003164 .14081922 .20089084
 .30780351 .33442344 .21384455 .26961685 .52651339 .46079441 .76946412 .29059069
 .19334201 .17844362 .78998469 .28046968 .22235800 .26665145 .17522070 .38529888
 .56648395 .24182896 .26678154 .28127510 .15675490 .26200498 .38955417 .19405974
 .24754650 .15837463 .26463956 .75490485 .26372731 .19277416 .19790320 .56545381
 .33186524 .34844310 .33888083 .56513074 .22795159 .62282251 .65498712 .37922326
 .29108534 .15529561 .25623245 .44981826 .26326095 .31771047 .17421425 .17735044
 .21000000 .18986014 .15128991 .19448415 .43291392 .14296495 .15553077 .35084442
 .51086155 .39027582 .19146612 .46459370 .31412311 .14900065 .73011498 .19755093
 .25215119 .46270187 .25739906 .43970588 .27261238 .21016827 .48379332 .39795625
 .26598624 .20135135 .32402341 .25164806 .26875360 .17740989 .38948057 .21090397
 .22353862 .19416847 .40316993 .20162184 .29546307 .27216860 .28288878 .53122848
 .16435669 .20702003 .54942015 .19642648 .20468529 .18876304 .23549023 .19622116
 .21084407 .21105382 .47735724 .40757649 .17677389 .14501255 .26185773 .80995141
 .30719476 .33950880 .23495600 .33948699 .39113678 .25818822 .36441028 .22700978

```

.31440985 .41451346 .14387378 .27961136 .43213315 .80664163 .20160618 .40064935
.37711809 .37757052 .36619257 .75237507 .36362163 .51492406 .48658237 .75393159
.16420047 .39928532 .39977220 .90540229 .47870250 .62986520 .21965174 .41501266
.34012714 .20178826 .27423373 .20029119 .31142786 .25168466 .26524437 .51722551
.34410493 .34149227 .27972257 .43130089 .26649141 .34196924 .36489342 .19723648
.42229980 .77739868 .17819730 .68082007 .20396850 .28391713 .19489369 .24102463
.73744041 .27528263 .16122206 .35441062 .37135862 .23351696 .43849819 .37555937
.23348379 .18298644 .20363128 .19297152 .72390399 .28344850 .18032311 .63291116
.36882451 .32540512 .37031196 .22988425 .26103499 .20832678 .19377097 .35027743
.18626803 .36477025 .28473180 .34294387 .22474991 .22541964 .30817073 .22787402
.19669693 .28289763 .32243401 .18412680 .36577027 .32300861 .60582310 .29193408
.40138015 .20117282 .26886852 .70153475 .23027751 .20176752 .43765114 .32921698
.23429025 .27492273 .59947227 .33497330 .21406988 .20677163 .17133592 .35267198
.31134118 .18413655 .32154606 .29248008 .19459330 .20076436 .39219273 .22852344
.20145018 .37397283 .37040279 .18350753 .42570781 .31904192 .28712335 .31415094
.26952004 .18526719 .20305178 .26822361 .30349343 .56538828 .22331878 .27801882
.26800977 .33906771 .43648137 .37216053 .22571626 .18620885 .49726408 .17389233
.29065335 .19732080 .26463474 .35368296 .20159416 .29123910 1.0089320 .21966732
.24251701 .28736832 .32732048 .20817302 .20202548 .16792158 .27389531 .34692515
.47424786 .19760071 .16658792 .16709216 .18784400 .21153159 .36360424 .17223203
.43651025 .16895537 .39192341 .30463480 .26618751 .73242224 .31295152 .43908912
.26835399 .20312866 .50805207 .60427431 .26951875 .27465555 .20417208 .61735900
.42684635 .23318538 .48893979 .23412307 .20373490 .18376336 .32051738 .20992956
.16722327 .22058782 .28692004 .19826789 .69341726 .20491367 .55725659 .15140351
.20342309 .27607621 .18660556 .28221618 .29493394 .24598136 .29493809 .27106970
.20859166 .63354366 .20069241 .63102260 .23240424 .39838633 .21148835 .25419580
.15057876 .30268641 .23296895 .43913081 .25212715 .15198437 .41772955 .41261278
.15154722 .42375886 .24979415 .20889012 .15248932 .46971831 .15158804 .21414051
.31000000 .22871872 .30234258 .31144702 .30276994 .88623554 .15238908 .44473684
.35927835 .15638298 .57631579 .24590941 .83752559 .83947368 .79586183 .21170711
.15527383 .30885793 .47180493 .22623099 .81780312 .25310031 .42960682 .21594738
.43208368 .24772905 .50395299 .44594904 .43564488 .49627129 .42776646 .34245392
.30049689 .34586947 .25541764 .25673621 .21836225 .51803548 .21838828 .47560294
.29592931 .15767103 .43905064 .41856954 .52325420 .22461406 .73849894 .72699747
.15507714 .87500737 .22580310 .42543577 .15618082 .15232330 .15360818 .21475866
.48288834 .23562874 .57696891 .35010262 .33567754 .21519356 .72303751 .15724487
.53172681 .22708659 .54644374 .45848820 .23206566 .15500185 .26705813 .92024001
.55280224 .46205186 .36281033 .59227134 .16137078 .52680121 .76460877 .33902788
.35703369 .46042320 .22593361 .32719665 .33587610 .23154519 .47515643 .16235876
.15624827 .43451339 .56964553 .27178653 .71729422 .32616896 .78811838 .42234414
.46618978 .93154270 .78804142 .23177494 .24018503 .58454630 .23827767 .15960210
.26230723 .76349680 .15972511 .58277472 .75900191 .84723559 .23467168 .15776725
.46293787 .35462294 .23061207 .24131039 .15931939 .16326489 .86500000 .89493171
.31080000 .89750156 .47020586 .46988837 .80626571 .16053894 .45822931 .16192418
.37502816 .54806225 .48201933 .62455255 .76295856 .36511401 .22368421 .16415091
.26668046 .76814454 .79584826 .16115501 .23877902 .37031491 .87447601 .22025554
.74528542 .94393250 .48536635 .33455181 .84143829 .46475282 .84737224 .37081669
.81167523 .30464384 .33341515 .16828945 .31995850 .22214078 .84044894 .78838738
.24575724 .21957084 .30296312 .73909109 .47716084 .82299547 .24698537 .59494599
.93253671 .16688083 .25074512 .16875448 .37830258 .22153953 .85826756 .22348834
.98460505 .24516649 .30036496 .22475926 .25307754 .28703867 .26630094 .86176829
.22713885 .31289273 .26005994 .23245783 .22785683 .24999198 .97824086 .93830233
.90683510 .75000000 .18207547 .68702359 .31435500 .99940123 .99932442 .86807195
.31208136 .30357803 .23081341 .26719899 .81236161 .73695367 .87973086 .18174455

```

.31061003 .18411018 .18284525 .23917200 .33161522 .29620295 .69193755 .66240717
.71480570 .72509604 .69211420 .89745763 .87833215 .56857143 .23603936 .53510604
.27115291 .68513113 .32678924 .18544438 .23648233 .91247637 .27234541 .32749901
.67933095 .66445984 .27482344 .74277108 .57410901 .68423498 .53569675 .18920176
.18836352 .30695637 .28475795 .24643793 .57748239 .23771842 .23346124 .54310697
.78362004 .69857074 .27677914 .18889194 .23679705 .40185185 .32162453 .19069736
.56985973 .38709655 .31787409 .24486117 .23184848 .31697044 .31513080 .58466256
.25118837 .23385330 .59164509 .19367445 .75758149 .19436816 .81917009 .25045012
.40962952 .58162708 .23944494 .60992629 .25483515 .38736463 .25469877 .25546634
.24059415 .38877767 .25818622 .25834418 .69516129 .13602151 .24170217 .25281001
.39739669 .13627481 .59919597 .41930390 .24831920 .40598137 .24697198 .13698801
.67356971 .72666667 .24503920 .67892110 .13699965 .25545151 .66637938 .63956655
.63859084 .25159998 .41558006 .26131105 .32218543 .41492462 .31088663 .24873236
.13774402 .95000000 .31143544 .91094497 .83375221 .13856830 .43169336 .63708415
.67197753 .13898388 .66722045 .68906978 .69102564 .80929325 .66957687 .73665896
.14004387 .69194983 .69611268 .31826574 .71279775 .29482759 .69074603 .67840324
.65001963 .68338908 .31921456 .69287962 .32555110 .29379433 .71259862 .71459014
.72048318 .29142440 .14255391 .75906853 .73202939 .86034547 .93533573 .29143142
.89472580 .98405768 .69975616 .95038117 .72498844 .86042075 .70501858 .30229450
.91682427 .91191733 .33699227 .92778128 .33480993 .63690804 .34801325 .33656383
.31157483 .58522500 .31417813 .30891524 .62573292 .34505577 .52084753 .47901609
.48682297 .61813425 .36088625 .66819980 .48979599 .59932373 .35942280 .35456595
.50251002 .35280376 .85666667 .51329700 .75874701 .52581516 .75510340 .53075985
.53726554 .53460246 .53012752 .65827251 .68529980 .89745763 .25000000 .83229912
.63421568 .52160487 .63667477 .54088633 .61956507 .84031550 .53408475 .82509843
.53560059 .24809345 .65816890 .63927929 .85468874 .51826319 .60637698 .64744589
.79325015 .70402308 .85413283 .88830564 .67089915 .25219686 .65172424 .69322363
.80992327 .62220315 .86526891 .68520983 .25110706 .25272443 .25584717 .67539839
.65962742 .62845745 .82448071 .69353079 .66832025 .67768022 .68747430 .70265583
.82300344 .94030260 .25655601 .25566109 .67801192 .25761541 .88912806 .73013425
.68772935 .69990863 .71156760 .64030326 .26013554 .88823529 .81876176 .87313272
.88342767 .34898990 .84178258 .90023078 .91148429 .87961072 .33874334 .90762067
.91173743 .36511864 .35172808 .36149717 .50633079 .49055584 .48624680 .50607398
.50290358 .48796307 .51023746 .49856806 .51039098 .50839063 .51134819 .51392867
.50787110 .28611111 .28602807 .28588631 .28346225 .28733853 .28715469 .28677926
.28897827 .29435725 .29215063 .29600076 .81183073 .81409993 .81199100 .81786597
.76925459 .77851607 .78583989 .78694797 .79938858 .80175471 .78133041 .78863075
.79401933 1.0617647 1.0192081 1.0581342 .98106139 .98465840 .99409832 1.0021158
1.0357515 1.0076380 1.0099242 1.0275639 1.0608620

Platinum 1083

.22818706 .21900585 .28982036 .24216270 .23155846 .24182739 .24278990 .23212851
.26711409 .23636364 .22504472 .26342434 .22144250 .25728518 .24978355 .23269978
.25882353 .33969732 .25055718 .32376238 .26877579 .24437401 .24601479 .22566372
.24264957 .24089403 .34778325 .54090909 .29090909 .21550388 .30084034 .40689655
.41276596 .23627570 .26535948 .28587258 .25245902 .27836607 .24934464 .22453039
.22008313 .26538462 .29051880 .37078169 .25825243 .26858449 .25901639 .37499054
.23889776 .28485640 .37653449 .30000000 .24962225 .32515645 .27690533 .30752688
.25213425 .25760369 .42222222 .26855184 .44968789 .24301075 .26779661 .41430065
.26307692 .28362369 .26867060 .26818182 .29782609 .25172414 .23063934 .24253308
.28070332 .30444444 .41275672 .28201523 .27449752 .34326648 .27092199 .24868549
.30559662 .23952569 .23855243 .23787879 .74945055 .29836711 .27014886 .23174603
.39801980 .29615385 .34609204 .32961763 .26711409 .25312424 .33333333 .38650453
.26053269 .24632203 .26612357 .25875441 .33583265 .35394089 .26664774 .25868407
.30912592 .36748218 .35000000 .23594714 .74736842 .33155565 .34690393 .28281250


```

.29398987 .28510638 .26136364 .35416959 .32751553 .38333333 .59941903 .31791189
.62399013 .31339739 .24500634 .28395802 .26451613 .26739262 .27600000 .25198501
.26151455 .27138742 .32384430 .41071429 .37824773 .34095948 .26982834 .39672131
.26358963 .30290713 .30928166 .36713092 .38633540 .43210832 .30989011 .41621622
.23000000 .26282723 .33735320 .26379619 .30804217 .26408040 .28289474 .39349489
.34159292 .29228498 .30655738 .28084577 .26143667 .23111388 .29497753 .35766561
.28247270 .52475401 .36190476 .44808464 .31878529 .30823190 .28313253 .24451510
.41538462 .31023622 .35047291 .28111240 .30501050 .28626001 .27169560 .24375000
.33084112 .34044944 .48000000 .28578431 .25857741 .27232675 .32909091 .29491979
.31076923 .30746269 .34400000 .38154312 .29384409 .28297812 .29632513 .37360285
.29434572 .27366996 .28308259 .34532067 .29375000 .40374345 .30416667 .35075377
.45531915 .28708273 .34962594 .26843066 .38679950 .30285244 .28539223 .24954419
.30445828 .37857143 .28180985 .31194030 .40885547 .29375000 .43262840 .86655273
.30506856 .24742865 .31282491 .29874105 .39070322 .32698413 .28844666 .30847458
.40779221 .36666667 .31704462 .45516713 .27079646 .26908463 .41775225 .27020808
.40542213 .40343580 .33666667 .28644865 .27477477 .37568368 .33014504 .31666667
.35000000 .28822325 .37142857 .38278353 .39873743 .31556764 .41144279 .38518519
.40336258 .54400000 .32951222 .31211340 .32385399 .67927673 .30958904 .37235639
.43446034 .32941176 .31866667 .31448271 .28900000 .30593599 .25654420 .25656720
.29825026 .28286835 .33044613 .35444015 .37742730 .28653846 .36071429 .39482628
.33753056 .44781460 .98787879 .29891304 .42580645 .31794872 .30823529 .28790336
.29518936 .45050505 .35373680 .30549115 .35079365 .40808691 .33816373 .43750000
.29623481 .42068966 .33287671 .37321429 .30543478 .37786416 .31829268 .38148148
.28237232 .29432794 .47049180 .31665587 .45195327 .33079667 .33404826 .38468585
.44242424 .29009009 .41459227 .59984006 .38315018 .24429679 .33651877 .29852217
.35278839 .38416206 .33346680 .38834165 .48169014 .32500000 .37857143 .35384615
.29652510 .27968127 .26600660 .34285714 .37024174 .46525199 .27520776 .38587361
.32500000 .30224949 .31520737 .38348624 .36482611 .40576132 .31072726 .35363815
.32578616 .29090909 .32820513 .28105698 .40833333 .34534884 .59920949 .33648649
.29693238 .33762025 .31277697 .45505101 .28434051 .38727273 .33661741 .34380133
.35718092 .76451613 .38667662 .23917910 .36114180 .32825787 .34929577 .50113636
.35595622 .27229730 .41504944 .38021053 .30058301 .34778932 .33688213 .32356979
.36191904 .24722963 .47169811 .29909091 .86605561 .39282138 .29956103 .32758949
.37248138 .36437046 .35616571 .33496933 .24680851 .36176471 .33538462 .37336485
.27857143 .29312169 .35022602 .36176471 .34356666 .62045455 .32987925 .32099279
.28045977 .37197697 .33671541 .38864775 .35804196 .34302168 .49523516 .35144033
.33855422 .33855422 .36625104 .36000000 .39333333 .37221077 .37145621 .36160444
.33764706 .35000000 .32485727 .32643580 .30916627 .35063291 .29240565 .35281880
.36527778 .44498703 .37101082 .33477694 .40000000 .32917115 .32464942 .33793103
.31695678 .43121387 .29036145 .36666667 .28611410 .35000000 .34634146 .48007929
.33363029 .33186813 .40689655 .34499058 .55420172 .42234987 .25326384 .36655884
.51225806 .35297092 .31111111 .37329545 .45477443 .38467637 .32154150 .37571429
.39218750 .33085106 .34599407 .51538462 .33523022 .38931009 .31071429 .36625996
.34760499 .32314907 .39325912 .40833333 .36579789 .38518519 .31870845 .32183236
.34651163 .42105263 .31413043 .34615643 .39384615 .46800599 .37735410 .35269361
.32757517 .38823529 .30281124 .35240091 .36847109 .40806452 .28993746 .35116279
.46355269 .29023016 .34332334 .40000000 .72273915 .42413793 .41666667 .30833333
.28176101 .41666667 .26500000 .35046296 .29433072 .27482014 .57643678 .37525084
.36211038 .35000000 .37733333 .35563929 .37570513 .40944882 .42634445 .29032328
.38660354 .30229008 .44222704 .33917526 .34438503 .32443428 .32385321 .35697857
.51192661 .35454545 .34822509 .36817038 .36913580 .53906634 .37468354 .31813652
.35012907 .43421177 .28315187 .36242492 .32178987 .37188505 .68289183 .34990898
.52941176 .37948718 .28805031 .39851117 .42173646 .43500000 .36592335 .36022205
.31637341 .48944150 .39589041 .46481481 .49017410 .34661274 .28690476 .52731757

```

.36781609 .41262439 .29051557 .36470588 .35336788 .39003163 .39486294 .38483830
 .40910226 .47337841 .37647059 .34992504 .47027027 .74545455 .55650624 .31296031
 .40576132 .27226756 .41978022 .45646286 .33157895 .44183797 .51649576 .34285714
 .40491803 .32254902 .40689655 .37006803 .58560411 .44768006 .31034117 .98947368
 .71679587 .35384615 .30917825 .40066890 .39493177 .50000000 .33071895 .29677419
 .28080808 .30242404 .62857143 .38656716 .37552071 .77471264 .34285714 .45862069
 .35625000 .57500000 .36666667 .27283321 .38518519 .28233841 .40586015 .40297700
 .31538462 .31783189 .33470916 .33888889 .28469791 .34705882 .43060124 .39473318
 .39740260 .33341408 .32801725 .59760915 .34143749 .42719511 .27470361 .35910077
 .38845443 .43134328 .55307517 .45619835 .41198900 .34814532 .35424775 .39500000
 .31862216 .40204880 .39625000 .32474178 .42791609 .39036145 .46781931 .33034798
 .37610147 .44365044 .46709222 .35259117 .46666667 .35067474 .33617021 .42494025
 .38306636 .40801034 .72451539 .50961538 .39479970 .31960398 .38290824 .32278308
 .27850054 .54927343 .46164058 .45873016 .35837390 .45076923 .41410464 .42029095
 .42620690 .45884383 .34224138 .43957486 .29983844 .38965517 .35068493 .55162493
 .36129032 .45846154 .39040927 .80472985 .31489379 .30803859 .43427368 .39882353
 .36620281 .42666667 .40731707 .51264368 .35127920 .35454545 .41794872 .33385827
 .40941119 .40731707 .34782609 .44637681 .46771654 .28999518 .44735647 .44225352
 .38210526 .38836698 .43046358 .35519779 .35309085 .37766497 .32323944 .42435897
 .33156411 .35086207 .38409079 .43162688 .33169802 .33179014 .39445742 .76019779
 .31194030 1.0372093 .40696792 .32857143 .50000000 .46086957 .44774961 .32233247
 .28397202 .40371842 .43131085 .30621535 .36064399 .42180205 .36694491 .38763065
 .32372077 .46714676 .37572368 .50093192 .42493651 .44708414 .46210666 .40499216
 .42691421 .31215912 .33715019 .41609195 .44216798 .66325800 .45115612 .42237910
 .49687500 .35862414 .46388889 .29500000 .36101695 .34615385 .35499955 .60402322
 1.0189655 .41029573 .34296296 .64124371 .30604396 .32229974 .35287146 .28846329
 .35937623 .41885522 .47857143 .33068845 .35844145 .34411765 .43279448 .31445171
 .35307692 .39417476 .40025031 .37278618 .38315018 .30309278 .32121212 .41276596
 .29803922 .32987013 .35102318 .38227387 .34295926 .42909507 .32040939 .37746229
 .60000000 .30774411 .26480881 .46666667 .34485196 .29925558 .32210012 .41197668
 .30526316 .33661202 .38975332 .88339453 .42599759 .52371795 .36898816 .42788761
 .35254556 .29192050 .42027388 .66403712 .41456757 .35632745 .51550596 .34190779
 .39797964 .44653534 .48611111 .29919168 .46157697 .46317115 .33850599 .43573467
 .35412957 .49291154 .58181818 .35217391 .42105263 .36800000 .51343284 .43325299
 .41530612 .39948623 .40938272 .34812651 .30070351 .34521864 .36174870 .38581738
 .31320933 .32862461 .43511057 .68444444 .37031250 .41312581 .49055883 .35109701
 .40036430 .38032787 .34193548 .46626506 .36518519 .41862745 .42300000 .52873978
 .75368796 .61443875 .36039641 .44725275 .44485374 .32085561 .41862364 .42755752
 .33613972 .44105263 .31824481 .53176107 .41065550 .30623557 .45000000 .49487179
 .61818182 .33511231 .48438691 .41693186 .43283236 .37570054 .50347642 .47011494
 .36414283 .40088389 .51519730 .42358491 .50789133 .47054370 .76801909 .42611164
 .40518608 .83157895 .69180328 .30702359 .61551724 .34855740 .36897507 .40486986
 .44043199 .38283582 .41778392 .42820082 .35030303 .57014925 .44293005 .49751237
 .41651044 .43809524 .52913359 .39230769 .47901786 .42222222 .63859649 .30740677
 .31363636 .39481785 .39685039 .37730496 .54260655 .35151515 .32500000 .39531250
 .71334702 .38518519 .31848341 .93529412 .34925373 .30822511 .43016019 .35151515
 .36534392 .33262599 .40833333 .36198004 .54482759 .35923567 .34470775 .41889885
 .33185079 .39661117 .49230046 .77736721 .52000000 .40254927 .49304700 .47689745
 .60751605 .55358844 .47263158 .65029382 .56464104 .33684211 .50063800 .48724730
 .78888889 .45825243 .42960725 .40787602 .35759885 .69715250 .45258216 .43331980
 .36666667 .46831683 .97307867 .39821821 .38700519 .68176671 .34945652 .40139733
 .40922082 .62203093 .44814785 .91025641 .43783357 .90447430 .42954839 .42047244
 .64094488 .48383838 .45089286 .45691952 .40803776 .41061655 .37109663 .67960428
 .99413058 .34427861 .45663717 .54186047 .37363155 .80000000 .76497175 .32500000

```
.84935065 .61088435 .45998623 .77664859 .69402313 .36459234 .49941566 .63055556
.58733706 1.1000000 .93187732 .52490421 .65737154 .34883721 .34608774 .41672241
.79112061 .40121951 .41143135 .62542286 .59294118 .70557643 .40294626 .35397212
.41219512 .31075949 .26930693 .40454545 .31108137 .49200000 .27139374 .41372485
.43806452 .45517241 .63285522 .43750000 .30439560 .48616397 .84500000 .43835186
.44195906 .27629645 1.1397590 .83225973 .36765957 .99000000 1.1000000 .51007752
.37777778 .45157233 .37777778 .24705882 .29638554 .50839695 .34103448 .61407035
.62142857 1.1420930 .24882264 .24882252 .38400479 1.0287129 .53600000 .45454545
.34254784 1.1708958 .34834044 1.1428621 .34473328 .25082316 .55200000 .63172809
.53940283 .59130435 .55053902 1.0996063 .59835451 1.3129808 .37537313 .39957537
.40434783 .60085874 1.1000000 .50674847 .36949153 .36616816 .37361111 .40408163
.54482759 .41276596 .38477893 .40628515 .49311386 .37478741 .37880256 .38030140
.54842529 .53609467 .57518759 .93269690 .42526690 .42567608 1.0505882 1.0414333
1.0500000 .55906977 .54456662
;
```

MATCH_11

```
data match_11;
  input Pair Low Age Lwt Race Smoke Ptd Ht UI @@;
  select(race);
    when (1) do;
      race1=0;
      race2=0;
    end;
    when (2) do;
      race1=1;
      race2=0;
    end;
    when (3) do;
      race1=0;
      race2=1;
    end;
  end;
  datalines;
1 0 14 135 1 0 0 0 0 1 1 14 101 3 1 1 0 0
2 0 15 98 2 0 0 0 0 2 1 15 115 3 0 0 0 1
3 0 16 95 3 0 0 0 0 3 1 16 130 3 0 0 0 0
4 0 17 103 3 0 0 0 0 4 1 17 130 3 1 1 0 1
5 0 17 122 1 1 0 0 0 5 1 17 110 1 1 0 0 0
6 0 17 113 2 0 0 0 0 6 1 17 120 1 1 0 0 0
7 0 17 113 2 0 0 0 0 7 1 17 120 2 0 0 0 0
8 0 17 119 3 0 0 0 0 8 1 17 142 2 0 0 1 0
9 0 18 100 1 1 0 0 0 9 1 18 148 3 0 0 0 0
10 0 18 90 1 1 0 0 1 10 1 18 110 2 1 1 0 0
11 0 19 150 3 0 0 0 0 11 1 19 91 1 1 1 0 1
12 0 19 115 3 0 0 0 0 12 1 19 102 1 0 0 0 0
13 0 19 235 1 1 0 1 0 13 1 19 112 1 1 0 0 1
14 0 20 120 3 0 0 0 1 14 1 20 150 1 1 0 0 0
15 0 20 103 3 0 0 0 0 15 1 20 125 3 0 0 0 1
16 0 20 169 3 0 1 0 1 16 1 20 120 2 1 0 0 0
17 0 20 141 1 0 1 0 1 17 1 20 80 3 1 0 0 1
```

```

18 0 20 121 2 1 0 0 0      18 1 20 109 3 0 0 0 0
19 0 20 127 3 0 0 0 0      19 1 20 121 1 1 1 0 1
20 0 20 120 3 0 0 0 0      20 1 20 122 2 1 0 0 0
21 0 20 158 1 0 0 0 0      21 1 20 105 3 0 0 0 0
22 0 21 108 1 1 0 0 1      22 1 21 165 1 1 0 1 0
23 0 21 124 3 0 0 0 0      23 1 21 200 2 0 0 0 0
24 0 21 185 2 1 0 0 0      24 1 21 103 3 0 0 0 0
25 0 21 160 1 0 0 0 0      25 1 21 100 3 0 1 0 0
26 0 21 115 1 0 0 0 0      26 1 21 130 1 1 0 1 0
27 0 22 95 3 0 0 1 0       27 1 22 130 1 1 0 0 0
28 0 22 158 2 0 1 0 0      28 1 22 130 1 1 1 0 1
29 0 23 130 2 0 0 0 0      29 1 23 97 3 0 0 0 1
30 0 23 128 3 0 0 0 0      30 1 23 187 2 1 0 0 0
31 0 23 119 3 0 0 0 0      31 1 23 120 3 0 0 0 0
32 0 23 115 3 1 0 0 0      32 1 23 110 1 1 1 0 0
33 0 23 190 1 0 0 0 0      33 1 23 94 3 1 0 0 0
34 0 24 90 1 1 1 0 0       34 1 24 128 2 0 1 0 0
35 0 24 115 1 0 0 0 0      35 1 24 132 3 0 0 1 0
36 0 24 110 3 0 0 0 0      36 1 24 155 1 1 1 0 0
37 0 24 115 3 0 0 0 0      37 1 24 138 1 0 0 0 0
38 0 24 110 3 0 1 0 0      38 1 24 105 2 1 0 0 0
39 0 25 118 1 1 0 0 0      39 1 25 105 3 0 1 1 0
40 0 25 120 3 0 0 0 1      40 1 25 85 3 0 0 0 1
41 0 25 155 1 0 0 0 0      41 1 25 115 3 0 0 0 0
42 0 25 125 2 0 0 0 0      42 1 25 92 1 1 0 0 0
43 0 25 140 1 0 0 0 0      43 1 25 89 3 0 1 0 0
44 0 25 241 2 0 0 1 0      44 1 25 105 3 0 1 0 0
45 0 26 113 1 1 0 0 0      45 1 26 117 1 1 1 0 0
46 0 26 168 2 1 0 0 0      46 1 26 96 3 0 0 0 0
47 0 26 133 3 1 1 0 0      47 1 26 154 3 0 1 1 0
48 0 26 160 3 0 0 0 0      48 1 26 190 1 1 0 0 0
49 0 27 124 1 1 0 0 0      49 1 27 130 2 0 0 0 1
50 0 28 120 3 0 0 0 0      50 1 28 120 3 1 1 0 1
51 0 28 130 3 0 0 0 0      51 1 28 95 1 1 0 0 0
52 0 29 135 1 0 0 0 0      52 1 29 130 1 0 0 0 1
53 0 30 95 1 1 0 0 0       53 1 30 142 1 1 1 0 0
54 0 31 215 1 1 0 0 0      54 1 31 102 1 1 1 0 0
55 0 32 121 3 0 0 0 0      55 1 32 105 1 1 0 0 0
56 0 34 170 1 0 1 0 0      56 1 34 187 2 1 0 1 0
;

```

PROCLIB.DELAY

```

data proclib.delay;
    input flight $3. +5 date date7. +2 orig $3. +3 dest $3. +3
        delaycat $15. +2 destype $15. +8 delay;
    informat date date7.;
    format date date7.;
    datalines;
114    01MAR94  LGA  LAX  1-10 Minutes    Domestic      8
202    01MAR94  LGA  ORD  No Delay         Domestic     -5
219    01MAR94  LGA  LON  11+ Minutes     International  18

```

622	01MAR94	LGA	FRA	No Delay	International	-5
132	01MAR94	LGA	YYZ	11+ Minutes	International	14
271	01MAR94	LGA	PAR	1-10 Minutes	International	5
302	01MAR94	LGA	WAS	No Delay	Domestic	-2
114	02MAR94	LGA	LAX	No Delay	Domestic	0
202	02MAR94	LGA	ORD	1-10 Minutes	Domestic	5
219	02MAR94	LGA	LON	11+ Minutes	International	18
622	02MAR94	LGA	FRA	No Delay	International	0
132	02MAR94	LGA	YYZ	1-10 Minutes	International	5
271	02MAR94	LGA	PAR	1-10 Minutes	International	4
302	02MAR94	LGA	WAS	No Delay	Domestic	0
114	03MAR94	LGA	LAX	No Delay	Domestic	-1
202	03MAR94	LGA	ORD	No Delay	Domestic	-1
219	03MAR94	LGA	LON	1-10 Minutes	International	4
622	03MAR94	LGA	FRA	No Delay	International	-2
132	03MAR94	LGA	YYZ	1-10 Minutes	International	6
271	03MAR94	LGA	PAR	1-10 Minutes	International	2
302	03MAR94	LGA	WAS	1-10 Minutes	Domestic	5
114	04MAR94	LGA	LAX	11+ Minutes	Domestic	15
202	04MAR94	LGA	ORD	No Delay	Domestic	-5
219	04MAR94	LGA	LON	1-10 Minutes	International	3
622	04MAR94	LGA	FRA	11+ Minutes	International	30
132	04MAR94	LGA	YYZ	No Delay	International	-5
271	04MAR94	LGA	PAR	1-10 Minutes	International	5
302	04MAR94	LGA	WAS	1-10 Minutes	Domestic	7
114	05MAR94	LGA	LAX	No Delay	Domestic	-2
202	05MAR94	LGA	ORD	1-10 Minutes	Domestic	2
219	05MAR94	LGA	LON	1-10 Minutes	International	3
622	05MAR94	LGA	FRA	No Delay	International	-6
132	05MAR94	LGA	YYZ	1-10 Minutes	International	3
271	05MAR94	LGA	PAR	1-10 Minutes	International	5
114	06MAR94	LGA	LAX	No Delay	Domestic	-1
202	06MAR94	LGA	ORD	No Delay	Domestic	-3
219	06MAR94	LGA	LON	11+ Minutes	International	27
132	06MAR94	LGA	YYZ	1-10 Minutes	International	7
302	06MAR94	LGA	WAS	1-10 Minutes	Domestic	1
114	07MAR94	LGA	LAX	No Delay	Domestic	-1
202	07MAR94	LGA	ORD	No Delay	Domestic	-2
219	07MAR94	LGA	LON	11+ Minutes	International	15
622	07MAR94	LGA	FRA	11+ Minutes	International	21
132	07MAR94	LGA	YYZ	No Delay	International	-2
271	07MAR94	LGA	PAR	1-10 Minutes	International	4
302	07MAR94	LGA	WAS	No Delay	Domestic	0

;

PROCLIB.EMP95

```
data proclib.emp95;
  input #1 idnum $4. @6 name $15.
        #2 address $42.
        #3 salary 6.;
  datalines;
```

```

2388 James Schmidt
100 Apt. C Blount St. SW Raleigh NC 27693
92100
2457 Fred Williams
99 West Lane Garner NC 27509
33190
2776 Robert Jones
12988 Wellington Farms Ave. Cary NC 27512
29025
8699 Jerry Capalleti
222 West L St. Oxford NC 27587
39985
2100 Lanny Engles
293 Manning Pl. Raleigh NC 27606
30998
9857 Kathy Krupski
1000 Taft Ave. Morrisville NC 27508
38756
0987 Dolly Lunford
2344 Persimmons Branch Apex NC 27505
44010
3286 Hoa Nguyen
2818 Long St. Cary NC 27513
87734
6579 Bryan Samosky
3887 Charles Ave. Garner NC 27508
50234
3888 Kim Siu
5662 Magnolia Blvd Southeast Cary NC 27513
77558
;

```

PROCLIB.EMP96

```

data proclib.emp96;
    input #1 idnum $4. @6 name $15.
           #2 address $42.
           #3 salary 6.;
    datalines;
2388 James Schmidt
100 Apt. C Blount St. SW Raleigh NC 27693
92100
2457 Fred Williams
99 West Lane Garner NC 27509
33190
2776 Robert Jones
12988 Wellington Farms Ave. Cary NC 27511
29025
8699 Jerry Capalleti
222 West L St. Oxford NC 27587
39985

```

```

3278 Mary Cravens
211 N. Cypress St. Cary NC 27512
35362
2100 Lanny Engles
293 Manning Pl. Raleigh NC 27606
30998
9857 Kathy Krupski
100 Taft Ave. Morrisville NC 27508
40456
0987 Dolly Lunford
2344 Persimmons Branch Trail Apex NC 27505
45110
3286 Hoa Nguyen
2818 Long St. Cary NC 27513
89834
6579 Bryan Samosky
3887 Charles Ave. Garner NC 27508
50234
3888 Kim Siu
5662 Magnolia Blvd Southwest Cary NC 27513
79958
6544 Roger Monday
3004 Crepe Myrtle Court Raleigh NC 27604
47007
;

```

PROCLIB.INTERNAT

```

data proclib.internat;
    input flight $3.  +5 date date7. +2 dest $3. +8 boarded;
    informat date date7.;
    format date date7.;
    datalines;
219      01MAR94  LON      198
622      01MAR94  FRA      207
132      01MAR94  YYZ      115
271      01MAR94  PAR      138
219      02MAR94  LON      147
622      02MAR94  FRA      176
132      02MAR94  YYZ      106
271      02MAR94  PAR      172
219      03MAR94  LON      197
622      03MAR94  FRA      180
132      03MAR94  YYZ       75
271      03MAR94  PAR      147
219      04MAR94  LON      232
622      04MAR94  FRA      137
132      04MAR94  YYZ      117
271      04MAR94  PAR      146
219      05MAR94  LON      160
622      05MAR94  FRA      185
132      05MAR94  YYZ      157

```

```

271    05MAR94  PAR      177
219    06MAR94  LON      163
132    06MAR94  YYZ      150
219    07MAR94  LON      241
622    07MAR94  FRA      210
132    07MAR94  YYZ      164
271    07MAR94  PAR      155
;

```

PROCLIB.LAKES

```

data proclib.lakes;
    input region $ 1-2 lake $ 5-13 pol_a1 pol_a2 pol_b1-pol_b4;
    datalines;
NE Carr      0.24    0.99    0.95    0.36    0.44    0.67
NE Duraleigh 0.34    0.01    0.48    0.58    0.12    0.56
NE Charlie   0.40    0.48    0.29    0.56    0.52    0.95
NE Farmer    0.60    0.65    0.25    0.20    0.30    0.64
NW Canyon    0.63    0.44    0.20    0.98    0.19    0.01
NW Morris    0.85    0.95    0.80    0.67    0.32    0.81
NW Golf      0.69    0.37    0.08    0.72    0.71    0.32
NW Falls     0.01    0.02    0.59    0.58    0.67    0.02
SE Pleasant  0.16    0.96    0.71    0.35    0.35    0.48
SE Juliette  0.82    0.35    0.09    0.03    0.59    0.90
SE Massey    1.01    0.77    0.45    0.32    0.55    0.66
SE Delta     0.84    1.05    0.90    0.09    0.64    0.03
SW Alumni    0.45    0.32    0.45    0.44    0.55    0.12
SW New Dam   0.80    0.70    0.31    0.98    1.00    0.22
SW Border    0.51    0.04    0.55    0.35    0.45    0.78
SW Red       0.22    0.09    0.02    0.10    0.32    0.01
;

```

PROCLIB.MARCH

```

data proclib.march;
    input flight $3. +5 date date7. +3 depart time5. +2 orig $3.
    +3 dest $3. +7 miles +6 boarded +6 capacity;
    format date date7. depart time5.;
    informat date date7. depart time5.;
    datalines;
114    01MAR94    7:10 LGA  LAX      2475      172      210
202    01MAR94   10:43 LGA  ORD       740      151      210
219    01MAR94    9:31 LGA  LON     3442      198      250
622    01MAR94   12:19 LGA  FRA     3857      207      250
132    01MAR94   15:35 LGA  YYZ       366      115      178
271    01MAR94   13:17 LGA  PAR     3635      138      250
302    01MAR94   20:22 LGA  WAS       229      105      180
114    02MAR94    7:10 LGA  LAX      2475      119      210
202    02MAR94   10:43 LGA  ORD       740      120      210
219    02MAR94    9:31 LGA  LON     3442      147      250

```


622	02MAR94	12:19	LGA	FRA	3857	176	250
132	02MAR94	15:35	LGA	YYZ	366	106	178
302	02MAR94	20:22	LGA	WAS	229	78	180
271	02MAR94	13:17	LGA	PAR	3635	104	250
114	03MAR94	7:10	LGA	LAX	2475	197	210
202	03MAR94	10:43	LGA	ORD	740	118	210
219	03MAR94	9:31	LGA	LON	3442	197	250
622	03MAR94	12:19	LGA	FRA	3857	180	250
132	03MAR94	15:35	LGA	YYZ	366	75	178
271	03MAR94	13:17	LGA	PAR	3635	147	250
302	03MAR94	20:22	LGA	WAS	229	123	180
114	04MAR94	7:10	LGA	LAX	2475	178	210
202	04MAR94	10:43	LGA	ORD	740	148	210
219	04MAR94	9:31	LGA	LON	3442	232	250
622	04MAR94	12:19	LGA	FRA	3857	137	250
132	04MAR94	15:35	LGA	YYZ	366	117	178
271	04MAR94	13:17	LGA	PAR	3635	146	250
302	04MAR94	20:22	LGA	WAS	229	115	180
114	05MAR94	7:10	LGA	LAX	2475	117	210
202	05MAR94	10:43	LGA	ORD	740	104	210
219	05MAR94	9:31	LGA	LON	3442	160	250
622	05MAR94	12:19	LGA	FRA	3857	185	250
132	05MAR94	15:35	LGA	YYZ	366	157	178
271	05MAR94	13:17	LGA	PAR	3635	177	250
114	06MAR94	7:10	LGA	LAX	2475	128	210
202	06MAR94	10:43	LGA	ORD	740	115	210
219	06MAR94	9:31	LGA	LON	3442	163	250
132	06MAR94	15:35	LGA	YYZ	366	150	178
302	06MAR94	20:22	LGA	WAS	229	66	180
114	07MAR94	7:10	LGA	LAX	2475	160	210
202	07MAR94	10:43	LGA	ORD	740	175	210
219	07MAR94	9:31	LGA	LON	3442	241	250
622	07MAR94	12:19	LGA	FRA	3857	210	250
132	07MAR94	15:35	LGA	YYZ	366	164	178
271	07MAR94	13:17	LGA	PAR	3635	155	250
302	07MAR94	20:22	LGA	WAS	229	135	180

;

PROCLIB.PAYLIST2

```
proc sql;
  create table proclib.paylist2
    (IdNum char(4),
      Gender char(1),
      Jobcode char(3),
      Salary num,
      Birth num informat=date7.
        format=date7.,
      Hired num informat=date7.
        format=date7.);

insert into proclib.paylist2
```

```

values('1919','M','TA2',34376,'12SEP66'd,'04JUN87'd)
values('1653','F','ME2',31896,'15OCT64'd,'09AUG92'd)
values('1350','F','FA3',36886,'31AUG55'd,'29JUL91'd)
values('1401','M','TA3',38822,'13DEC55'd,'17NOV93'd)
values('1499','M','ME1',23025,'26APR74'd,'07JUN92'd);

```

```

title 'PROCLIB.PAYLIST2 Table';
select * from proclib.paylist2;

```

PROCLIB.PAYROLL

This data set (table) is updated in Example 3 on page 1211 and its updated data is used in subsequent examples.

```

data proclib.payroll;
  input IdNumber $4. +3 Gender $1. +4 Jobcode $3. +9 Salary 5.
        +2 Birth date7. +2 Hired date7.;
  informat birth date7. hired date7.;
  format birth date7. hired date7.;
  datalines;
1919  M   TA2      34376 12SEP60 04JUN87
1653  F   ME2      35108 15OCT64 09AUG90
1400  M   ME1      29769 05NOV67 16OCT90
1350  F   FA3      32886 31AUG65 29JUL90
1401  M   TA3      38822 13DEC50 17NOV85
1499  M   ME3      43025 26APR54 07JUN80
1101  M   SCP      18723 06JUN62 01OCT90
1333  M   PT2      88606 30MAR61 10FEB81
1402  M   TA2      32615 17JAN63 02DEC90
1479  F   TA3      38785 22DEC68 05OCT89
1403  M   ME1      28072 28JAN69 21DEC91
1739  M   PT1      66517 25DEC64 27JAN91
1658  M   SCP      17943 08APR67 29FEB92
1428  F   PT1      68767 04APR60 16NOV91
1782  M   ME2      35345 04DEC70 22FEB92
1244  M   ME2      36925 31AUG63 17JAN88
1383  M   BCK      25823 25JAN68 20OCT92
1574  M   FA2      28572 27APR60 20DEC92
1789  M   SCP      18326 25JAN57 11APR78
1404  M   PT2      91376 24FEB53 01JAN80
1437  F   FA3      33104 20SEP60 31AUG84
1639  F   TA3      40260 26JUN57 28JAN84
1269  M   NA1      41690 03MAY72 28NOV92
1065  M   ME2      35090 26JAN44 07JAN87
1876  M   TA3      39675 20MAY58 27APR85
1037  F   TA1      28558 10APR64 13SEP92
1129  F   ME2      34929 08DEC61 17AUG91
1988  M   FA3      32217 30NOV59 18SEP84
1405  M   SCP      18056 05MAR66 26JAN92
1430  F   TA2      32925 28FEB62 27APR87
1983  F   FA3      33419 28FEB62 27APR87
1134  F   TA2      33462 05MAR69 21DEC88
1118  M   PT3      111379 16JAN44 18DEC80

```

1438	F	TA3	39223	15MAR65	18NOV87
1125	F	FA2	28888	08NOV68	11DEC87
1475	F	FA2	27787	15DEC61	13JUL90
1117	M	TA3	39771	05JUN63	13AUG92
1935	F	NA2	51081	28MAR54	16OCT81
1124	F	FA1	23177	10JUL58	01OCT90
1422	F	FA1	22454	04JUN64	06APR91
1616	F	TA2	34137	01MAR70	04JUN93
1406	M	ME2	35185	08MAR61	17FEB87
1120	M	ME1	28619	11SEP72	07OCT93
1094	M	FA1	22268	02APR70	17APR91
1389	M	BCK	25028	15JUL59	18AUG90
1905	M	PT1	65111	16APR72	29MAY92
1407	M	PT1	68096	23MAR69	18MAR90
1114	F	TA2	32928	18SEP69	27JUN87
1410	M	PT2	84685	03MAY67	07NOV86
1439	F	PT1	70736	06MAR64	10SEP90
1409	M	ME3	41551	19APR50	22OCT81
1408	M	TA2	34138	29MAR60	14OCT87
1121	M	ME1	29112	26SEP71	07DEC91
1991	F	TA1	27645	07MAY72	12DEC92
1102	M	TA2	34542	01OCT59	15APR91
1356	M	ME2	36869	26SEP57	22FEB83
1545	M	PT1	66130	12AUG59	29MAY90
1292	F	ME2	36691	28OCT64	02JUL89
1440	F	ME2	35757	27SEP62	09APR91
1368	M	FA2	27808	11JUN61	03NOV84
1369	M	TA2	33705	28DEC61	13MAR87
1411	M	FA2	27265	27MAY61	01DEC89
1113	F	FA1	22367	15JAN68	17OCT91
1704	M	BCK	25465	30AUG66	28JUN87
1900	M	ME2	35105	25MAY62	27OCT87
1126	F	TA3	40899	28MAY63	21NOV80
1677	M	BCK	26007	05NOV63	27MAR89
1441	F	FA2	27158	19NOV69	23MAR91
1421	M	TA2	33155	08JAN59	28FEB90
1119	M	TA1	26924	20JUN62	06SEP88
1834	M	BCK	26896	08FEB72	02JUL92
1777	M	PT3	109630	23SEP51	21JUN81
1663	M	BCK	26452	11JAN67	11AUG91
1106	M	PT2	89632	06NOV57	16AUG84
1103	F	FA1	23738	16FEB68	23JUL92
1477	M	FA2	28566	21MAR64	07MAR88
1476	F	TA2	34803	30MAY66	17MAR87
1379	M	ME3	42264	08AUG61	10JUN84
1104	M	SCP	17946	25APR63	10JUN91
1009	M	TA1	28880	02MAR59	26MAR92
1412	M	ME1	27799	18JUN56	05DEC91
1115	F	FA3	32699	22AUG60	29FEB80
1128	F	TA2	32777	23MAY65	20OCT90
1442	F	PT2	84536	05SEP66	12APR88
1417	M	NA2	52270	27JUN64	07MAR89
1478	M	PT2	84203	09AUG59	24OCT90
1673	M	BCK	25477	27FEB70	15JUL91

1839	F	NA1	43433	29NOV70	03JUL93
1347	M	TA3	40079	21SEP67	06SEP84
1423	F	ME2	35773	14MAY68	19AUG90
1200	F	ME1	27816	10JAN71	14AUG92
1970	F	FA1	22615	25SEP64	12MAR91
1521	M	ME3	41526	12APR63	13JUL88
1354	F	SCP	18335	29MAY71	16JUN92
1424	F	FA2	28978	04AUG69	11DEC89
1132	F	FA1	22413	30MAY72	22OCT93
1845	M	BCK	25996	20NOV59	22MAR80
1556	M	PT1	71349	22JUN64	11DEC91
1413	M	FA2	27435	16SEP65	02JAN90
1123	F	TA1	28407	31OCT72	05DEC92
1907	M	TA2	33329	15NOV60	06JUL87
1436	F	TA2	34475	11JUN64	12MAR87
1385	M	ME3	43900	16JAN62	01APR86
1432	F	ME2	35327	03NOV61	10FEB85
1111	M	NA1	40586	14JUL73	31OCT92
1116	F	FA1	22862	28SEP69	21MAR91
1352	M	NA2	53798	02DEC60	16OCT86
1555	F	FA2	27499	16MAR68	04JUL92
1038	F	TA1	26533	09NOV69	23NOV91
1420	M	ME3	43071	19FEB65	22JUL87
1561	M	TA2	34514	30NOV63	07OCT87
1434	F	FA2	28622	11JUL62	28OCT90
1414	M	FA1	23644	24MAR72	12APR92
1112	M	TA1	26905	29NOV64	07DEC92
1390	M	FA2	27761	19FEB65	23JUN91
1332	M	NA1	42178	17SEP70	04JUN91
1890	M	PT2	91908	20JUL51	25NOV79
1429	F	TA1	27939	28FEB60	07AUG92
1107	M	PT2	89977	09JUN54	10FEB79
1908	F	TA2	32995	10DEC69	23APR90
1830	F	PT2	84471	27MAY57	29JAN83
1882	M	ME3	41538	10JUL57	21NOV78
1050	M	ME2	35167	14JUL63	24AUG86
1425	F	FA1	23979	28DEC71	28FEB93
1928	M	PT2	89858	16SEP54	13JUL90
1480	F	TA3	39583	03SEP57	25MAR81
1100	M	BCK	25004	01DEC60	07MAY88
1995	F	ME1	28810	24AUG73	19SEP93
1135	F	FA2	27321	20SEP60	31MAR90
1415	M	FA2	28278	09MAR58	12FEB88
1076	M	PT1	66558	14OCT55	03OCT91
1426	F	TA2	32991	05DEC66	25JUN90
1564	F	SCP	18833	12APR62	01JUL92
1221	F	FA2	27896	22SEP67	04OCT91
1133	M	TA1	27701	13JUL66	12FEB92
1435	F	TA3	38808	12MAY59	08FEB80
1418	M	ME1	28005	29MAR57	06JAN92
1017	M	TA3	40858	28DEC57	16OCT81
1443	F	NA1	42274	17NOV68	29AUG91
1131	F	TA2	32575	26DEC71	19APR91
1427	F	TA2	34046	31OCT70	30JAN90

```

1036  F   TA3           39392  19MAY65  23OCT84
1130  F   FA1           23916  16MAY71  05JUN92
1127  F   TA2           33011  09NOV64  07DEC86
1433  F   FA3           32982  08JUL66  17JAN87
1431  F   FA3           33230  09JUN64  05APR88
1122  F   FA2           27956  01MAY63  27NOV88
1105  M   ME2           34805  01MAR62  13AUG90
;

```

PROCLIB.PAYROLL2

```

data proclib.payroll2;
    input idnum $4. +3 gender $1. +4 jobcode $3. +9 salary 5.
           +2 birth date7. +2 hired date7.;
    informat birth date7. hired date7.;
    format birth date7. hired date7.;
    datalines;
1639  F   TA3           42260  26JUN57  28JAN84
1065  M   ME3           38090  26JAN44  07JAN87
1561  M   TA3           36514  30NOV63  07OCT87
1221  F   FA3           29896  22SEP67  04OCT91
1447  F   FA1           22123  07AUG72  29OCT92
1998  M   SCP           23100  10SEP70  02NOV92
1036  F   TA3           42465  19MAY65  23OCT84
1106  M   PT3           94039  06NOV57  16AUG84
1129  F   ME3           36758  08DEC61  17AUG91
1350  F   FA3           36098  31AUG65  29JUL90
1369  M   TA3           36598  28DEC61  13MAR87
1076  M   PT1           69742  14OCT55  03OCT91
;

```

PROCLIB.SCHEDULE

```

data proclib.schedule;
    input flight $3. +5 date date7. +2 dest $3. +3 idnum $4.;
    format date date7.;
    informat date date7.;
    datalines;
132    01MAR94  YYZ    1739
132    01MAR94  YYZ    1478
132    01MAR94  YYZ    1130
132    01MAR94  YYZ    1390
132    01MAR94  YYZ    1983
132    01MAR94  YYZ    1111
219    01MAR94  LON    1407
219    01MAR94  LON    1777
219    01MAR94  LON    1103
219    01MAR94  LON    1125
219    01MAR94  LON    1350
219    01MAR94  LON    1332

```

271	01MAR94	PAR	1439
271	01MAR94	PAR	1442
271	01MAR94	PAR	1132
271	01MAR94	PAR	1411
271	01MAR94	PAR	1988
271	01MAR94	PAR	1443
622	01MAR94	FRA	1545
622	01MAR94	FRA	1890
622	01MAR94	FRA	1116
622	01MAR94	FRA	1221
622	01MAR94	FRA	1433
622	01MAR94	FRA	1352
132	02MAR94	YYZ	1556
132	02MAR94	YYZ	1478
132	02MAR94	YYZ	1113
132	02MAR94	YYZ	1411
132	02MAR94	YYZ	1574
132	02MAR94	YYZ	1111
219	02MAR94	LON	1407
219	02MAR94	LON	1118
219	02MAR94	LON	1132
219	02MAR94	LON	1135
219	02MAR94	LON	1441
219	02MAR94	LON	1332
271	02MAR94	PAR	1739
271	02MAR94	PAR	1442
271	02MAR94	PAR	1103
271	02MAR94	PAR	1413
271	02MAR94	PAR	1115
271	02MAR94	PAR	1443
622	02MAR94	FRA	1439
622	02MAR94	FRA	1890
622	02MAR94	FRA	1124
622	02MAR94	FRA	1368
622	02MAR94	FRA	1477
622	02MAR94	FRA	1352
132	03MAR94	YYZ	1739
132	03MAR94	YYZ	1928
132	03MAR94	YYZ	1425
132	03MAR94	YYZ	1135
132	03MAR94	YYZ	1437
132	03MAR94	YYZ	1111
219	03MAR94	LON	1428
219	03MAR94	LON	1442
219	03MAR94	LON	1130
219	03MAR94	LON	1411
219	03MAR94	LON	1115
219	03MAR94	LON	1332
271	03MAR94	PAR	1905
271	03MAR94	PAR	1118
271	03MAR94	PAR	1970
271	03MAR94	PAR	1125
271	03MAR94	PAR	1983
271	03MAR94	PAR	1443

622	03MAR94	FRA	1545
622	03MAR94	FRA	1830
622	03MAR94	FRA	1414
622	03MAR94	FRA	1368
622	03MAR94	FRA	1431
622	03MAR94	FRA	1352
132	04MAR94	YYZ	1428
132	04MAR94	YYZ	1118
132	04MAR94	YYZ	1103
132	04MAR94	YYZ	1390
132	04MAR94	YYZ	1350
132	04MAR94	YYZ	1111
219	04MAR94	LON	1739
219	04MAR94	LON	1478
219	04MAR94	LON	1130
219	04MAR94	LON	1125
219	04MAR94	LON	1983
219	04MAR94	LON	1332
271	04MAR94	PAR	1407
271	04MAR94	PAR	1410
271	04MAR94	PAR	1094
271	04MAR94	PAR	1411
271	04MAR94	PAR	1115
271	04MAR94	PAR	1443
622	04MAR94	FRA	1545
622	04MAR94	FRA	1890
622	04MAR94	FRA	1116
622	04MAR94	FRA	1221
622	04MAR94	FRA	1433
622	04MAR94	FRA	1352
132	05MAR94	YYZ	1556
132	05MAR94	YYZ	1890
132	05MAR94	YYZ	1113
132	05MAR94	YYZ	1475
132	05MAR94	YYZ	1431
132	05MAR94	YYZ	1111
219	05MAR94	LON	1428
219	05MAR94	LON	1442
219	05MAR94	LON	1422
219	05MAR94	LON	1413
219	05MAR94	LON	1574
219	05MAR94	LON	1332
271	05MAR94	PAR	1739
271	05MAR94	PAR	1928
271	05MAR94	PAR	1103
271	05MAR94	PAR	1477
271	05MAR94	PAR	1433
271	05MAR94	PAR	1443
622	05MAR94	FRA	1545
622	05MAR94	FRA	1830
622	05MAR94	FRA	1970
622	05MAR94	FRA	1441
622	05MAR94	FRA	1350
622	05MAR94	FRA	1352

```

132      06MAR94  YYZ   1333
132      06MAR94  YYZ   1890
132      06MAR94  YYZ   1414
132      06MAR94  YYZ   1475
132      06MAR94  YYZ   1437
132      06MAR94  YYZ   1111
219      06MAR94  LON   1106
219      06MAR94  LON   1118
219      06MAR94  LON   1425
219      06MAR94  LON   1434
219      06MAR94  LON   1555
219      06MAR94  LON   1332
132      07MAR94  YYZ   1407
132      07MAR94  YYZ   1118
132      07MAR94  YYZ   1094
132      07MAR94  YYZ   1555
132      07MAR94  YYZ   1350
132      07MAR94  YYZ   1111
219      07MAR94  LON   1905
219      07MAR94  LON   1478
219      07MAR94  LON   1124
219      07MAR94  LON   1434
219      07MAR94  LON   1983
219      07MAR94  LON   1332
271      07MAR94  PAR   1410
271      07MAR94  PAR   1777
271      07MAR94  PAR   1103
271      07MAR94  PAR   1574
271      07MAR94  PAR   1115
271      07MAR94  PAR   1443
622      07MAR94  FRA   1107
622      07MAR94  FRA   1890
622      07MAR94  FRA   1425
622      07MAR94  FRA   1475
622      07MAR94  FRA   1433
622      07MAR94  FRA   1352
;

```

PROCLIB.STAFF

```

data proclib.staff;
    input idnum $4. +3 lname $15. +2 fname $15. +2 city $15. +2
        state $2. +5 hphone $12.;
    datalines;
1919  ADAMS          GERALD          STAMFORD      CT      203/781-1255
1653  ALIBRANDI     MARIA          BRIDGEPORT   CT      203/675-7715
1400  ALHERTANI     ABDULLAH      NEW YORK     NY      212/586-0808
1350  ALVAREZ       MERCEDES      NEW YORK     NY      718/383-1549
1401  ALVAREZ       CARLOS        PATERSON     NJ      201/732-8787
1499  BAREFOOT      JOSEPH        PRINCETON    NJ      201/812-5665
1101  BAUCOM        WALTER        NEW YORK     NY      212/586-8060
1333  BANADYGA      JUSTIN        STAMFORD     CT      203/781-1777

```


1402	BLALOCK	RALPH	NEW YORK	NY	718/384-2849
1479	BALLETTI	MARIE	NEW YORK	NY	718/384-8816
1403	BOWDEN	EARL	BRIDGEPORT	CT	203/675-3434
1739	BRANCACCIO	JOSEPH	NEW YORK	NY	212/587-1247
1658	BREUHAUS	JEREMY	NEW YORK	NY	212/587-3622
1428	BRADY	CHRISTINE	STAMFORD	CT	203/781-1212
1782	BREWCZAK	JAKOB	STAMFORD	CT	203/781-0019
1244	BUCCI	ANTHONY	NEW YORK	NY	718/383-3334
1383	BURNETTE	THOMAS	NEW YORK	NY	718/384-3569
1574	CAHILL	MARSHALL	NEW YORK	NY	718/383-2338
1789	CARAWAY	DAVIS	NEW YORK	NY	212/587-9000
1404	COHEN	LEE	NEW YORK	NY	718/384-2946
1437	CARTER	DOROTHY	BRIDGEPORT	CT	203/675-4117
1639	CARTER-COHEN	KAREN	STAMFORD	CT	203/781-8839
1269	CASTON	FRANKLIN	STAMFORD	CT	203/781-3335
1065	COPAS	FREDERICO	NEW YORK	NY	718/384-5618
1876	CHIN	JACK	NEW YORK	NY	212/588-5634
1037	CHOW	JANE	STAMFORD	CT	203/781-8868
1129	COUNIHAN	BRENDA	NEW YORK	NY	718/383-2313
1988	COOPER	ANTHONY	NEW YORK	NY	212/587-1228
1405	DACKO	JASON	PATERSON	NJ	201/732-2323
1430	DABROWSKI	SANDRA	BRIDGEPORT	CT	203/675-1647
1983	DEAN	SHARON	NEW YORK	NY	718/384-1647
1134	DELGADO	MARIA	STAMFORD	CT	203/781-1528
1118	DENNIS	ROGER	NEW YORK	NY	718/383-1122
1438	DABBOUSSI	KAMILLA	STAMFORD	CT	203/781-2229
1125	DUNLAP	DONNA	NEW YORK	NY	718/383-2094
1475	ELGES	MARGARETE	NEW YORK	NY	718/383-2828
1117	EDGERTON	JOSHUA	NEW YORK	NY	212/588-1239
1935	FERNANDEZ	KATRINA	BRIDGEPORT	CT	203/675-2962
1124	FIELDS	DIANA	WHITE PLAINS	NY	914/455-2998
1422	FUJIHARA	KYOKO	PRINCETON	NJ	201/812-0902
1616	FUENTAS	CARLA	NEW YORK	NY	718/384-3329
1406	FOSTER	GERALD	BRIDGEPORT	CT	203/675-6363
1120	GARCIA	JACK	NEW YORK	NY	718/384-4930
1094	GOMEZ	ALAN	BRIDGEPORT	CT	203/675-7181
1389	GOLDSTEIN	LEVI	NEW YORK	NY	718/384-9326
1905	GRAHAM	ALVIN	NEW YORK	NY	212/586-8815
1407	GREGORSKI	DANIEL	MT. VERNON	NY	914/468-1616
1114	GREENWALD	JANICE	NEW YORK	NY	212/588-1092
1410	HARRIS	CHARLES	STAMFORD	CT	203/781-0937
1439	HASENHAUER	CHRISTINA	BRIDGEPORT	CT	203/675-4987
1409	HAVELKA	RAYMOND	STAMFORD	CT	203/781-9697
1408	HENDERSON	WILLIAM	PRINCETON	NJ	201/812-4789
1121	HERNANDEZ	ROBERTO	NEW YORK	NY	718/384-3313
1991	HOWARD	GRETCHEN	BRIDGEPORT	CT	203/675-0007
1102	HERMANN	JOACHIM	WHITE PLAINS	NY	914/455-0976
1356	HOWARD	MICHAEL	NEW YORK	NY	212/586-8411
1545	HERRERO	CLYDE	STAMFORD	CT	203/781-1119
1292	HUNTER	HELEN	BRIDGEPORT	CT	203/675-4830
1440	JACKSON	LAURA	STAMFORD	CT	203/781-0088
1368	JEPSEN	RONALD	STAMFORD	CT	203/781-8413
1369	JONSON	ANTHONY	NEW YORK	NY	212/587-5385
1411	JOHNSEN	JACK	PATERSON	NJ	201/732-3678

1113	JOHNSON	LESLIE	NEW YORK	NY	718/383-3003
1704	JONES	NATHAN	NEW YORK	NY	718/384-0049
1900	KING	WILLIAM	NEW YORK	NY	718/383-3698
1126	KIMANI	ANNE	NEW YORK	NY	212/586-1229
1677	KRAMER	JACKSON	BRIDGEPORT	CT	203/675-7432
1441	LAWRENCE	KATHY	PRINCETON	NJ	201/812-3337
1421	LEE	RUSSELL	MT. VERNON	NY	914/468-9143
1119	LI	JEFF	NEW YORK	NY	212/586-2344
1834	LEBLANC	RUSSELL	NEW YORK	NY	718/384-0040
1777	LUFKIN	ROY	NEW YORK	NY	718/383-4413
1663	MARKS	JOHN	NEW YORK	NY	212/587-7742
1106	MARSHBURN	JASPER	STAMFORD	CT	203/781-1457
1103	MCDANIEL	RONDA	NEW YORK	NY	212/586-0013
1477	MEYERS	PRESTON	BRIDGEPORT	CT	203/675-8125
1476	MONROE	JOYCE	STAMFORD	CT	203/781-2837
1379	MORGAN	ALFRED	STAMFORD	CT	203/781-2216
1104	MORGAN	CHRISTOPHER	NEW YORK	NY	718/383-9740
1009	MORGAN	GEORGE	NEW YORK	NY	212/586-7753
1412	MURPHEY	JOHN	PRINCETON	NJ	201/812-4414
1115	MURPHY	ALICE	NEW YORK	NY	718/384-1982
1128	NELSON	FELICIA	BRIDGEPORT	CT	203/675-1166
1442	NEWKIRK	SANDRA	PRINCETON	NJ	201/812-3331
1417	NEWKIRK	WILLIAM	PATERSON	NJ	201/732-6611
1478	NEWTON	JAMES	NEW YORK	NY	212/587-5549
1673	NICHOLLS	HENRY	STAMFORD	CT	203/781-7770
1839	NORRIS	DIANE	NEW YORK	NY	718/384-1767
1347	O'NEAL	BRYAN	NEW YORK	NY	718/384-0230
1423	OSWALD	LESLIE	MT. VERNON	NY	914/468-9171
1200	OVERMAN	MICHELLE	STAMFORD	CT	203/781-1835
1970	PARKER	ANNE	NEW YORK	NY	718/383-3895
1521	PARKER	JAY	NEW YORK	NY	212/587-7603
1354	PARKER	MARY	WHITE PLAINS	NY	914/455-2337
1424	PATTERSON	RENEE	NEW YORK	NY	212/587-8991
1132	PEARCE	CAROL	NEW YORK	NY	718/384-1986
1845	PEARSON	JAMES	NEW YORK	NY	718/384-2311
1556	PENNINGTON	MICHAEL	NEW YORK	NY	718/383-5681
1413	PETERS	RANDALL	PRINCETON	NJ	201/812-2478
1123	PETERSON	SUZANNE	NEW YORK	NY	718/383-0077
1907	PHELPS	WILLIAM	STAMFORD	CT	203/781-1118
1436	PORTER	SUSAN	NEW YORK	NY	718/383-5777
1385	RAYNOR	MILTON	BRIDGEPORT	CT	203/675-2846
1432	REED	MARILYN	MT. VERNON	NY	914/468-5454
1111	RHODES	JEREMY	PRINCETON	NJ	201/812-1837
1116	RICHARDS	CASEY	NEW YORK	NY	212/587-1224
1352	RIVERS	SIMON	NEW YORK	NY	718/383-3345
1555	RODRIGUEZ	JULIA	BRIDGEPORT	CT	203/675-2401
1038	RODRIGUEZ	MARIA	BRIDGEPORT	CT	203/675-2048
1420	ROUSE	JEREMY	PATERSON	NJ	201/732-9834
1561	SANDERS	RAYMOND	NEW YORK	NY	212/588-6615
1434	SANDERSON	EDITH	STAMFORD	CT	203/781-1333
1414	SANDERSON	NATHAN	BRIDGEPORT	CT	203/675-1715
1112	SANYERS	RANDY	NEW YORK	NY	718/384-4895
1390	SMART	JONATHAN	NEW YORK	NY	718/383-1141
1332	STEPHENSON	ADAM	BRIDGEPORT	CT	203/675-1497

1890	STEPHENSON	ROBERT	NEW YORK	NY	718/384-9874
1429	THOMPSON	ALICE	STAMFORD	CT	203/781-3857
1107	THOMPSON	WAYNE	NEW YORK	NY	718/384-3785
1908	TRENTON	MELISSA	NEW YORK	NY	212/586-6262
1830	TRIPP	KATHY	BRIDGEPORT	CT	203/675-2479
1882	TUCKER	ALAN	NEW YORK	NY	718/384-0216
1050	TUTTLE	THOMAS	WHITE PLAINS	NY	914/455-2119
1425	UNDERWOOD	JENNY	STAMFORD	CT	203/781-0978
1928	UPCHURCH	LARRY	WHITE PLAINS	NY	914/455-5009
1480	UPDIKE	THERESA	NEW YORK	NY	212/587-8729
1100	VANDEUSEN	RICHARD	NEW YORK	NY	212/586-2531
1995	VARNER	ELIZABETH	NEW YORK	NY	718/384-7113
1135	VEGA	ANNA	NEW YORK	NY	718/384-5913
1415	VEGA	FRANKLIN	NEW YORK	NY	718/384-2823
1076	VENTER	RANDALL	NEW YORK	NY	718/383-2321
1426	VICK	THERESA	PRINCETON	NJ	201/812-2424
1564	WALTERS	ANNE	NEW YORK	NY	212/587-3257
1221	WALTERS	DIANE	NEW YORK	NY	718/384-1918
1133	WANG	CHIN	NEW YORK	NY	212/587-1956
1435	WARD	ELAINE	NEW YORK	NY	718/383-4987
1418	WATSON	BERNARD	NEW YORK	NY	718/383-1298
1017	WELCH	DARIUS	NEW YORK	NY	212/586-5535
1443	WELLS	AGNES	STAMFORD	CT	203/781-5546
1131	WELLS	NADINE	NEW YORK	NY	718/383-1045
1427	WHALEY	CAROLYN	MT. VERNON	NY	914/468-4528
1036	WONG	LESLIE	NEW YORK	NY	212/587-2570
1130	WOOD	DEBORAH	NEW YORK	NY	212/587-0013
1127	WOOD	SANDRA	NEW YORK	NY	212/587-2881
1433	YANCEY	ROBIN	PRINCETON	NJ	201/812-1874
1431	YOUNG	DEBORAH	STAMFORD	CT	203/781-2987
1122	YOUNG	JOANN	NEW YORK	NY	718/384-2021
1105	YOUNG	LAWRENCE	NEW YORK	NY	718/384-0008

;

PROCLIB.SUPERV

```
data proclib.superv;
    input supid $4. +8 state $2. +5 jobcat $2.;
    label supid='Supervisor Id' jobcat='Job Category';
    datalines;
1677      CT      BC
1834      NY      BC
1431      CT      FA
1433      NJ      FA
1983      NY      FA
1385      CT      ME
1420      NJ      ME
1882      NY      ME
1935      CT      NA
1417      NJ      NA
1352      NY      NA
1106      CT      PT
```

```

1442      NJ      PT
1118      NY      PT
1405      NJ      SC
1564      NY      SC
1639      CT      TA
1401      NJ      TA
1126      NY      TA
;

```

RADIO

This DATA step uses an INFILE statement to read data that is stored in an external file.

```

data radio;
    infile 'input-file' missover;
    input /(time1-time7) ($1. +1);
    listener=_n_;
run;

```

Here is the data that is stored in the external file:

```

967 32 f 5 3 5
7 5 5 5 7 0 0 0 8 7 0 0 8 0
781 30 f 2 3 5
5 0 0 0 5 0 0 0 4 7 5 0 0 0
859 39 f 1 0 5
1 0 0 0 1 0 0 0 0 0 0 0 0 0
859 40 f 6 1 5
7 5 0 5 7 0 0 0 0 0 0 5 0 0
467 37 m 2 3 1
1 5 5 5 5 4 4 8 8 0 0 0 0 0
220 35 f 3 1 7
7 0 0 0 7 0 0 0 7 0 0 0 0 0
833 42 m 2 2 4
7 0 0 0 7 5 4 7 4 0 1 4 4 0
967 39 f .5 1 7
7 0 0 0 7 7 0 0 0 0 0 0 8 0
677 28 m .5 .5 7
7 0 0 0 0 0 0 0 0 0 0 0 0 0
833 28 f 3 4 1
1 0 0 0 0 1 1 1 1 0 0 0 1 1
677 24 f 3 1 2
2 0 0 0 0 0 0 2 0 8 8 0 0 0
688 32 m 5 2 4
5 5 0 4 8 0 0 5 0 8 0 0 0 0
542 38 f 6 8 5
5 0 0 5 5 5 0 5 5 5 5 5 0
677 27 m 6 1 1
1 1 0 4 4 0 0 1 4 0 0 0 0 0
779 37 f 2.5 4 7
7 0 0 0 7 7 0 7 7 4 4 7 8 0
362 31 f 1 2 2
8 0 0 0 8 0 0 0 0 0 8 8 0 0

```

```

859 29 m 10 3 4
4 4 0 2 2 0 0 4 0 0 0 4 4 0
467 24 m 5 8 1
7 1 1 1 7 1 1 0 1 7 1 1 1 1
851 34 m 1 2 8
0 0 0 0 8 0 0 0 4 0 0 0 8 0
859 23 f 1 1 8
8 0 0 0 8 0 0 0 0 0 0 0 8
781 34 f 9 3 1
2 1 0 1 4 4 4 0 1 1 1 1 4 4
851 40 f 2 4 5
5 0 0 0 5 0 0 5 0 0 5 5 0 0
783 34 m 3 2 4
7 0 0 0 7 4 4 0 0 4 4 0 0 0
848 29 f 4 1.5 7
7 4 4 1 7 0 0 0 7 0 0 7 0 0
851 28 f 1 2 2
2 0 2 0 2 0 0 0 0 2 2 2 0 0
856 42 f 1.5 1 2
2 0 0 0 0 0 0 2 0 0 0 0 0 0
859 29 m .5 .5 5
5 0 0 0 1 0 0 0 0 0 8 8 5 0
833 29 m 1 3 2
2 0 0 0 2 2 0 0 4 2 0 2 0 0
859 23 f 10 3 1
1 5 0 8 8 1 4 0 1 1 1 1 1 4
781 37 f .5 2 7
7 0 0 0 1 0 0 0 1 7 0 1 0 0
833 31 f 5 4 1
1 0 0 0 1 0 0 0 4 0 4 0 0 0
942 23 f 4 2 1
1 0 0 0 1 0 1 0 1 1 0 0 0 0
848 33 f 5 4 1
1 1 0 1 1 0 0 0 1 1 1 0 0 0
222 33 f 2 0 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0
851 45 f .5 1 8
8 0 0 0 8 0 0 0 0 0 8 0 0 0
848 27 f 2 4 1
1 0 0 0 1 1 0 0 4 1 1 1 1 1
781 38 m 2 2 1
5 0 0 0 1 0 0 0 0 0 1 1 0 0
222 27 f 3 1 2
2 0 2 0 2 2 0 0 2 0 0 0 0 0
467 34 f 2 2 1
1 0 0 0 0 1 0 1 0 0 0 0 1 0
833 27 f 8 8 1
7 0 1 0 7 4 0 0 1 1 1 4 1 0
677 49 f 1.5 0 8
8 0 8 0 8 0 0 0 0 0 0 0 0 0
849 43 m 1 4 1
1 0 0 0 4 0 0 0 4 0 1 0 0 0
467 28 m 2 1 7
7 0 0 0 7 0 0 7 0 0 1 0 0 0

```

732 29 f 1 0 2
 2 0 0 0 2 0 0 0 0 0 0 0 0 0
 851 31 m 2 2 2
 2 5 0 6 0 0 8 0 2 2 8 2 0 0
 779 42 f 8 2 2
 7 2 0 2 7 0 0 0 0 0 0 0 2 0
 493 40 m 1 3 3
 3 0 0 0 5 3 0 5 5 0 0 0 1 1
 859 30 m 1 0 7
 7 0 0 0 7 0 0 0 0 0 0 0 0 0
 833 36 m 4 2 5
 7 5 0 5 0 5 0 0 7 0 0 0 5 0
 467 30 f 1 4 1
 0 0 0 0 1 0 6 0 0 1 1 1 0 6
 859 32 f 3 5 2
 2 2 2 2 2 2 6 6 2 2 2 2 2 6
 851 43 f 8 1 5
 7 5 5 5 0 0 0 4 0 0 0 0 0 0
 848 29 f 3 5 1
 7 0 0 0 7 1 0 0 1 1 1 1 1 0
 833 25 f 2 4 5
 7 0 0 0 5 7 0 0 7 5 0 0 5 0
 783 33 f 8 3 8
 8 0 8 0 7 0 0 0 8 0 5 4 0 5
 222 26 f 10 2 1
 1 1 0 1 1 0 0 0 3 1 1 0 0 0
 222 23 f 3 2 2
 2 2 2 2 7 0 0 2 2 0 0 0 0 0
 859 50 f 1 5 4
 7 0 0 0 7 0 0 5 4 4 4 7 0 0
 833 26 f 3 2 1
 1 0 0 1 1 0 0 5 5 0 1 0 0 0
 467 29 m 7 2 1
 1 1 1 1 1 0 0 1 1 1 0 0 0 0
 859 35 m .5 2 2
 7 0 0 0 2 0 0 7 5 0 0 4 0 0
 833 33 f 3 3 6
 7 0 0 0 6 8 0 8 0 0 0 8 6 0
 221 36 f .5 1 5
 0 7 0 0 0 7 0 0 7 0 0 7 7 0
 220 32 f 2 4 5
 5 0 5 0 5 5 5 0 5 5 5 5 5 5
 684 19 f 2 4 2
 0 2 0 2 0 0 0 0 0 2 2 0 0 0
 493 55 f 1 0 5
 5 0 0 5 0 0 0 0 7 0 0 0 0 0
 221 27 m 1 1 7
 7 0 0 0 0 0 0 0 5 0 0 0 5 0
 684 19 f 0 .5 1
 7 0 0 0 0 1 1 0 0 0 0 0 1 1
 493 38 f .5 .5 5
 0 8 0 0 5 0 0 0 5 0 0 0 0 0
 221 26 f .5 2 1
 0 1 0 0 0 1 0 0 5 5 5 1 0 0

```

684 18 m 1 .5 1
0 2 0 0 0 0 1 0 0 0 0 1 1 0
684 19 m 1 1 1
0 0 0 1 1 0 0 0 0 0 1 0 0 0
221 29 m .5 .5 5
0 0 0 0 0 5 5 0 0 0 0 0 5 5
683 18 f 2 4 8
0 0 0 0 8 0 0 0 8 8 8 0 0 0
966 23 f 1 2 1
1 5 5 5 1 0 0 0 0 1 0 0 1 0
493 25 f 3 5 7
7 0 0 0 7 2 0 0 7 0 2 7 7 0
683 18 f .5 .5 2
1 0 0 0 0 0 5 0 0 1 0 0 0 1
382 21 f 3 1 8
0 8 0 0 5 8 8 0 0 8 8 0 0 0
683 18 f 4 6 2
2 0 0 0 2 2 2 0 2 0 2 2 2 0
684 19 m .5 2 1
0 0 0 0 1 1 0 0 0 1 1 1 1 5
684 19 m 1.5 3.5 2
2 0 0 0 2 0 0 0 0 0 2 5 0 0
221 23 f 1 5 1
7 5 1 5 1 3 1 7 5 1 5 1 3 1
684 18 f 2 3 1
2 0 0 1 1 1 1 7 2 0 1 1 1 1
683 19 f 3 5 2
2 0 0 2 0 6 1 0 1 1 2 2 6 1
683 19 f 3 5 1
2 0 0 2 0 6 1 0 1 1 2 0 2 1
221 35 m 3 5 5
7 5 0 1 7 0 0 5 5 5 0 0 0 0
221 43 f 1 4 5
1 0 0 0 5 0 0 5 5 0 0 0 0 0
493 32 f 2 1 6
0 0 0 6 0 0 0 0 0 0 0 0 4 0
221 24 f 4 5 2
2 0 5 0 0 2 4 4 4 5 0 0 2 2
684 19 f 2 3 2
0 5 5 2 5 0 1 0 5 5 2 2 2 2
221 19 f 3 3 8
0 1 1 8 8 8 4 0 5 4 1 8 8 4
221 29 m 1 1 5
5 5 5 5 5 5 5 5 5 5 5 5 5
221 21 m 1 1 1
1 0 0 0 0 0 5 1 0 0 0 0 0 5
683 20 f 1 2 2
0 0 0 0 2 0 0 0 2 0 0 0 0 0
493 54 f 1 1 5
7 0 0 5 0 0 0 0 0 0 5 0 0 0
493 45 m 4 6 5
7 0 0 0 7 5 0 0 5 5 5 5 5 5
850 44 m 2.5 1.5 7
7 0 7 0 4 7 5 0 5 4 3 0 0 4

```

220 33 m 5 3 5
 1 5 0 5 1 0 0 0 0 0 0 0 5 5
 684 20 f 1.5 3 1
 1 0 0 0 1 0 1 0 1 0 0 1 1 0
 966 63 m 3 5 3
 5 4 7 5 4 5 0 5 0 0 5 5 4 0
 683 21 f 4 6 1
 0 1 0 1 1 1 1 0 1 1 1 1 1 1
 493 23 f 5 2 5
 7 5 0 4 0 0 0 0 1 1 1 1 1 0
 493 32 f 8 8 5
 7 5 0 0 7 0 5 5 5 0 0 7 5 5
 942 33 f 7 2 5
 0 5 5 4 7 0 0 0 0 0 0 7 8 0
 493 34 f .5 1 5
 5 0 0 0 5 0 0 0 0 0 6 0 0 0
 382 40 f 2 2 5
 5 0 0 0 5 0 0 5 0 0 5 0 0 0
 362 27 f 0 3 8
 0 0 0 0 0 0 0 0 0 0 0 0 8 0
 542 36 f 3 3 7
 7 0 0 0 7 1 0 0 0 7 1 1 1 0 0
 966 39 f 3 6 5
 7 0 0 0 7 5 0 0 7 0 5 0 5 0
 849 32 m 1 .5 7
 7 0 0 0 5 0 0 0 7 4 4 5 7 0
 677 52 f 3 2 3
 7 0 0 0 0 7 0 0 0 7 0 0 3 0
 222 25 m 2 4 1
 1 0 0 0 1 0 0 0 1 0 1 0 0 0
 732 42 f 3 2 7
 7 0 0 0 1 7 5 5 7 0 0 3 4 0
 467 26 f 4 4 1
 7 0 1 0 7 1 0 0 7 7 4 7 0 0
 467 38 m 2.5 0 1
 1 0 0 0 1 0 0 0 0 0 0 0 0 0
 382 37 f 1.5 .5 7
 7 0 0 0 7 0 0 0 3 0 0 0 3 0
 856 45 f 3 3 7
 7 0 0 0 7 5 0 0 7 7 4 0 0 0
 677 33 m 3 2 7
 7 0 0 4 7 0 0 0 7 0 0 0 0 0
 490 27 f .5 1 2
 2 0 0 0 2 0 0 0 2 0 2 0 0 0
 362 27 f 1.5 2 2
 2 0 0 0 1 0 4 0 1 0 0 0 4 4
 783 25 f 2 1 1
 1 0 0 0 1 7 0 0 0 0 1 1 1 0
 546 30 f 8 3 1
 1 1 1 1 1 0 0 1 0 5 5 0 0 0
 677 30 f 2 0 1
 1 0 0 0 0 1 0 0 0 0 0 0 0 1
 221 35 f 2 2 1
 1 0 0 0 1 0 1 0 1 1 1 0 0 0


```

966 32 f 6 1 7
7 1 1 1 7 4 0 1 7 1 8 8 4 0
222 28 f 1 5 4
7 0 0 0 4 0 0 4 4 4 4 0 0 0
467 29 f 5 3 4
4 5 5 5 1 4 4 5 1 1 1 1 4 4
467 32 m 3 4 1
1 0 1 0 4 0 0 0 4 0 0 0 1 0
966 30 m 1.5 1 7
7 0 0 0 7 5 0 7 0 0 0 0 5 0
967 38 m 14 4 7
7 7 7 7 7 0 4 8 0 0 0 0 4 0
490 28 m 8 1 1
7 1 1 1 1 0 0 7 0 0 8 0 0 0
833 30 f .5 1 6
6 0 0 0 6 0 0 0 0 6 0 0 6 0
851 40 m 1 0 7
7 5 5 5 7 0 0 0 0 0 0 0 0 0
859 27 f 2 5 2
6 0 0 0 2 0 0 0 0 0 0 2 2 2
851 22 f 3 5 2
7 0 2 0 2 2 0 0 2 0 8 0 2 0
967 38 f 1 1.5 7
7 0 0 0 7 5 0 7 4 0 0 7 5 0
856 34 f 1.5 1 1
0 1 0 0 0 1 0 0 4 0 0 0 0 0
222 33 m .1 .1 7
7 0 0 0 7 0 0 0 0 0 7 0 0 0
856 22 m .50 .25 1
0 1 0 0 1 0 0 0 0 0 0 0 0 0
677 30 f 2 2 4
1 0 4 0 4 0 0 0 4 0 0 0 0 0
859 25 m 2 3 7
0 0 0 0 0 7 0 0 7 0 2 0 0 1
833 35 m 2 6 7
7 0 0 0 7 1 1 0 4 7 4 7 1 1
677 35 m 10 4 1
1 1 1 1 1 8 6 8 1 0 0 8 8 8
848 29 f 5 3 8
8 0 0 0 8 8 0 0 0 8 8 8 0 0
688 26 m 3 1 1
1 1 7 1 1 7 0 0 0 8 8 0 0 0
490 41 m 2 2 5
5 0 0 0 0 0 5 5 0 0 0 0 0 5
493 35 m 4 4 7
7 5 0 5 7 0 0 7 7 7 7 0 0 0
677 27 m 15 11 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
848 27 f 3 5 1
1 1 0 0 1 1 0 0 1 1 1 1 0 0
362 30 f 1 0 1
1 0 0 0 7 5 0 0 0 0 0 0 0 0
783 29 f 1 1 4
4 0 0 0 4 0 0 0 4 0 0 0 4 0

```

467 39 f .5 2 4
 7 0 4 0 4 4 0 0 4 4 4 4 4 4
 677 27 m 2 2 7
 7 0 0 0 7 0 0 7 7 0 0 7 0 0
 221 23 f 2.5 1 1
 1 0 0 0 1 0 0 0 0 0 0 0 0 0
 677 29 f 1 1 7
 0 0 0 0 7 0 0 0 7 0 0 0 0 0
 783 32 m 1 2 5
 4 5 5 5 4 2 0 0 0 0 3 2 2 0
 833 25 f 1 0 1
 1 1 0 0 0 0 0 0 0 0 0 0 0 0
 859 24 f 7 3 7
 1 0 0 0 1 0 0 0 0 1 0 0 1 0
 677 29 m 2 2 8
 0 8 8 0 8 0 0 0 8 8 8 0 0 0
 688 31 m 8 2 5
 7 5 5 5 5 7 0 0 7 7 0 0 0 0
 856 31 m 9 4 1
 1 1 1 1 1 0 0 0 0 0 0 0 1 0
 856 44 f 1 0 6
 6 0 0 0 6 0 0 0 0 0 0 0 0 0
 677 37 f 3 3 1
 0 0 1 0 0 0 0 0 4 4 0 0 0 0
 859 27 m 2 .5 2
 2 2 2 2 2 2 2 2 0 0 0 0 0 2
 781 30 f 10 4 2
 2 0 0 0 2 0 2 0 0 0 0 0 0 2
 362 27 m 12 4 3
 3 1 1 1 1 3 3 3 0 0 0 0 3 0
 362 33 f 2 4 1
 1 0 0 0 7 0 0 7 1 1 1 1 1 0
 222 26 f 8 1 1
 1 1 1 1 0 0 0 1 0 0 0 0 0 0
 779 37 f 6 3 1
 1 1 1 1 1 0 0 1 1 0 0 0 1 0
 467 32 f 1 1 2
 2 0 0 0 0 0 0 0 2 0 0 2 0 0
 859 23 m 1 1 1
 1 0 0 0 1 1 0 1 0 0 0 0 1 1
 781 33 f 1 .5 6
 6 0 0 0 6 0 0 0 0 0 0 0 0 0
 779 28 m 5 2 1
 1 1 1 1 1 0 0 0 0 7 7 1 1 0
 677 28 m 3 1 5
 7 5 5 5 5 6 0 0 6 6 6 6 6 0
 677 25 f 9 2 5
 1 5 5 5 5 1 1 0 1 1 1 1 1 1
 848 30 f 6 2 8
 8 0 0 0 2 7 0 0 0 0 2 0 2 0
 546 36 f 4 6 4
 7 0 0 0 4 4 0 5 5 5 5 2 4 4
 222 30 f 2 3 2
 2 2 0 0 2 0 0 0 2 0 2 2 0 0

```

383 32 m 4 1 2
2 0 0 0 2 0 0 2 0 0 0 0 0 0
851 43 f 8 1 6
4 6 0 6 4 0 0 0 0 0 0 0 0 0
222 27 f 1 3 1
1 1 0 1 1 1 0 0 1 0 0 0 4 0
833 22 f 1.5 2 1
1 0 0 0 1 1 0 0 1 1 1 0 0 0
467 29 f 2 1 8
8 0 8 0 8 0 0 0 0 0 8 0 0 0
856 28 f 2 3 1
1 0 0 0 1 0 0 0 1 0 0 1 0 0
580 31 f 2.5 2.5 6
6 6 6 6 6 6 6 6 1 1 1 1 6 6
688 39 f 8 8 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3
677 37 f 1.5 .5 1
6 1 1 1 6 6 0 0 1 1 6 6 6 0
859 38 m 3 6 3
7 0 0 0 7 3 0 0 3 0 3 0 0 0
677 25 f 7 1 1
0 1 1 1 2 0 0 0 1 2 1 1 1 0
848 36 f 7 1 1
0 1 0 1 1 0 0 0 0 0 0 1 1 0
781 31 f 2 4 1
1 0 0 0 1 1 0 1 1 1 1 1 0 0
781 40 f 2 2 8
8 0 0 8 8 0 0 0 0 0 8 8 0 0
677 25 f 3 5 1
1 6 1 6 6 3 0 0 2 2 1 1 1 1
779 33 f 3 2 1
1 0 1 0 0 0 1 0 1 0 0 0 1 0
677 25 m 7 1.5 1
1 1 0 1 1 0 0 0 0 0 1 0 0 0
362 35 f .5 0 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0
677 41 f 6 2 7
7 7 0 7 7 0 0 0 0 0 8 0 0 0
677 24 m 5 1 5
1 5 0 5 0 0 0 0 1 0 0 0 0 0
833 29 f .5 0 6
6 0 0 0 6 0 0 0 0 0 0 0 0 0
362 30 f 1 1 1
1 0 0 0 1 0 0 0 1 0 0 0 0 0
850 26 f 6 12 6
6 0 0 0 2 2 2 6 6 6 0 0 6 6
467 25 f 2 3 1
1 0 0 6 1 1 0 0 0 0 1 1 1 1
967 29 f 1 2 7
7 0 0 0 7 0 0 7 7 0 0 0 0 0
833 31 f 1 1 7
7 0 7 0 7 3 0 0 3 3 0 0 0 0
859 40 f 7 1 5
1 5 0 5 5 1 0 0 1 0 0 0 0 0

```

848 31 m 1 2 1
 1 0 0 0 1 1 0 0 4 4 1 4 0 0
 222 32 f 2 3 3
 3 0 0 0 0 7 0 0 3 0 8 0 0 0
 783 33 f 2 0 4
 7 0 0 0 7 0 0 0 4 0 4 0 0 0
 856 28 f 8 4 2
 0 2 0 2 2 0 0 0 2 0 2 0 4 0
 781 30 f 3 5 1
 1 1 1 1 1 1 0 0 1 1 1 1 1 0
 850 25 f 6 3 1
 7 5 0 5 7 1 0 0 7 0 1 0 1 0
 580 33 f 2.5 4 2
 2 0 0 0 2 0 0 0 0 0 8 8 0 0
 677 38 f 3 3 1
 1 0 0 0 1 0 1 1 1 0 1 0 0 4
 677 26 f 2 2 1
 1 0 1 0 1 0 0 0 1 1 1 0 0 0
 467 52 f 3 2 2
 2 6 6 6 6 2 0 0 2 2 2 2 0 0
 542 31 f 1 3 1
 1 0 1 0 1 0 0 0 1 1 1 1 1 0
 859 50 f 9 3 6
 6 6 6 6 6 6 6 6 6 3 3 3 6 6
 779 26 f 1 2 1
 7 0 1 0 1 1 4 1 4 1 1 1 4 4
 779 36 m 1.5 2 4
 1 4 0 4 4 0 0 4 4 4 4 0 0 0
 222 31 f 0 3 7
 1 0 0 0 7 0 0 0 0 0 0 0 0 0
 362 27 f 1 1 1
 1 0 1 0 1 4 0 4 4 1 0 4 4 0
 967 32 f 3 2 7
 7 0 0 0 7 0 0 0 1 0 0 1 0 0
 362 29 f 10 2 2
 2 2 2 2 2 2 2 2 2 2 2 7 0 0
 677 27 f 3 4 1
 0 5 1 1 0 5 0 0 0 1 1 1 0 0
 546 32 m 5 .5 8
 8 0 0 0 8 0 0 0 8 0 0 0 0 0
 688 38 m 2 3 2
 2 0 0 0 2 0 0 0 2 0 0 0 1 0
 362 28 f 1 1 1
 1 0 0 0 1 1 0 4 0 0 0 0 4 0
 851 32 f .5 2 4
 5 0 0 0 4 0 0 0 0 0 0 0 2 0
 967 43 f 2 2 1
 1 0 0 0 1 0 0 1 7 0 0 0 1 0
 467 44 f 10 4 6
 7 6 0 6 6 0 6 0 0 0 0 0 0 6
 467 23 f 5 3 1
 0 2 1 2 1 0 0 0 1 1 1 1 1 1
 783 30 f 1 .5 1
 1 0 0 0 1 0 0 0 0 0 0 7 0 0

```

677 29 f 3 1 2
2 2 2 2 2 0 0 0 0 0 0 0 0 0
859 26 f 9.5 1.5 2
2 2 2 2 2 0 0 2 2 0 0 0 0 0
222 28 f 3 0 2
2 0 0 0 2 0 0 0 0 0 2 0 0 0
966 37 m 2 1 1
7 1 1 1 7 0 0 0 7 0 0 0 0 0
859 31 f 10 10 1
0 1 1 1 1 0 0 0 1 1 0 0 1 0
781 27 f 2 1 2
2 0 0 0 1 0 0 0 4 0 0 0 0 0
677 31 f .5 .5 6
7 0 0 0 0 0 0 0 6 0 0 0 0 0
848 28 f 5 1 2
2 2 0 2 0 0 0 0 2 0 0 0 0 0
781 24 f 3 3 6
1 6 6 6 1 6 0 0 0 0 1 0 1 1
856 27 f 1.5 1 6
2 6 6 6 2 5 0 2 0 0 5 2 0 0
382 30 m 1 2 7
7 0 0 0 7 0 4 7 0 0 0 7 4 4
848 25 f 9 3 1
7 1 1 5 1 0 0 0 1 1 1 1 1 0
382 30 m 1 2 4
7 0 0 0 7 0 4 7 0 0 0 7 4 4
688 40 m 2 3 1
1 0 0 0 1 3 1 0 5 0 4 4 7 1
856 40 f .5 5 5
3 0 0 0 3 0 0 0 0 0 5 5 0 0
966 25 f 2 .5 2
1 0 0 0 2 6 0 0 4 0 0 0 0 0
859 30 f 2 4 2
2 0 0 0 0 2 0 0 0 0 2 0 0 0
849 29 m 10 1 5
7 5 5 5 7 5 5 0 0 0 0 0 7 0
781 28 m 1.5 3 4
1 0 0 0 1 4 4 0 4 4 1 1 4 0
467 35 f 4 2 6
7 6 7 6 6 7 6 7 7 7 7 7 6
222 32 f 10 5 1
1 1 0 1 1 0 0 1 1 1 0 0 1 0
677 32 f 1 0 1
1 0 1 0 0 0 0 0 0 0 0 0 0 0
222 54 f 21 4 3
5 0 0 0 7 0 0 7 0 0 0 0 0 0
677 30 m 4 6 1
7 0 0 0 0 1 1 1 7 1 1 0 8 1
683 29 f 1 2 8
8 0 0 0 8 0 0 0 0 8 8 0 0 0
467 38 m 3 5 1
1 0 0 0 1 0 0 1 1 0 0 0 0 0
781 29 f 2 3 8
8 0 0 0 8 8 0 0 8 8 0 8 8 0

```

781 30 f 1 0 5
 5 0 0 0 0 5 0 0 0 0 0 0 0 0
 783 40 f 1.5 3 1
 1 0 0 0 1 4 0 0 1 1 1 0 0 0
 851 30 f 1 1 6
 6 0 0 0 6 0 0 0 6 0 0 6 0 0
 851 40 f 1 1 5
 5 0 0 0 5 0 0 0 0 1 0 0 0 0
 779 40 f 1 0 2
 2 0 0 0 2 0 0 0 0 0 0 0 0 0
 467 37 f 4 8 1
 1 0 0 0 1 0 3 0 3 1 1 1 0 0
 859 37 f 4 3 3
 0 3 7 0 0 7 0 0 0 7 8 3 7 0
 781 26 f 4 1 2
 2 2 0 2 1 0 0 0 2 0 0 0 0 0
 859 23 f 8 3 3
 3 2 0 2 3 0 0 0 1 0 0 3 0 0
 967 31 f .5 0 1
 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 851 38 m 4 2 5
 7 5 0 5 4 0 4 7 7 0 4 0 8 0
 467 30 m 2 1 2
 2 2 0 2 0 0 0 0 2 0 2 0 0 0
 848 33 f 2 2 7
 7 0 0 0 0 7 0 7 7 0 0 0 7 0
 688 35 f 5 8 3
 2 2 2 2 2 0 0 3 3 3 3 3 0 0
 467 27 f 2 3 1
 1 0 1 0 0 1 0 0 1 1 1 0 0 0
 783 42 f 3 1 1
 1 0 0 0 1 0 0 0 1 0 1 1 0 0
 687 40 m 1.5 2 1
 7 0 0 0 1 1 0 0 1 0 7 0 1 0
 779 30 f 4 8 7
 7 0 0 0 7 0 6 7 4 2 2 0 0 6
 222 34 f 9 0 8
 8 2 0 2 8 0 0 0 0 0 0 0 0 0
 467 28 m 3 1 2
 2 0 0 0 2 2 0 0 0 2 2 0 0 0
 222 28 f 8 4 2
 1 2 1 2 2 0 0 1 2 2 0 0 2 0
 542 35 m 2 3 2
 6 0 7 0 7 0 7 0 0 0 2 2 0 0
 677 31 m 12 4 3
 7 3 0 3 3 4 0 0 4 4 4 0 0 0
 783 45 f 1.5 2 6
 6 0 0 0 6 0 0 6 6 0 0 0 0 0
 942 34 f 1 .5 4
 4 0 0 0 1 0 0 0 0 0 2 0 0 0
 222 30 f 8 4 1
 1 1 1 1 1 0 0 0 1 1 0 0 0 0
 967 38 f 1.5 2 7
 7 0 0 0 7 0 0 7 1 1 1 1 0 0

```

783 37 f 2 1 1
6 6 1 1 6 6 0 0 6 1 1 1 6 0
467 31 f 1.5 2 2
2 0 7 0 7 0 0 7 7 0 0 0 7 0
859 48 f 3 0 7
7 0 0 0 0 0 0 0 0 7 0 0 0 0
490 35 f 1 1 7
7 0 0 0 7 0 0 0 0 0 0 0 8 0
222 27 f 3 2 3
8 0 0 0 3 8 0 3 3 0 0 0 0 0
382 36 m 3 2 4
7 0 5 4 7 4 4 0 7 7 4 7 0 4
859 37 f 1 1 2
7 0 0 0 0 2 0 2 2 0 0 0 0 2
856 29 f 3 1 1
1 0 0 0 1 1 1 1 0 0 1 1 0 1
542 32 m 3 3 7
7 0 0 0 0 7 7 7 0 0 0 0 7 7
783 31 m 1 1 1
1 0 0 0 1 0 0 0 1 1 1 0 0 0
833 35 m 1 1 1
5 4 1 5 1 0 0 1 1 0 0 0 0 0
782 38 m 30 8 5
7 5 5 5 5 0 0 4 4 4 4 4 0 0
222 33 m 3 3 1
1 1 1 1 1 1 1 1 4 1 1 1 1 1
467 24 f 2 4 1
0 0 1 0 1 0 0 0 1 1 1 0 0 0
467 34 f 1 1 1
1 0 0 0 1 0 0 1 1 0 0 0 0 0
781 53 f 2 1 5
5 0 0 0 5 5 0 0 0 0 5 5 5 0
222 30 m 2 5 3
6 3 3 3 6 0 0 0 3 3 3 3 0 0
688 26 f 2 2 1
1 0 0 0 1 0 0 0 1 0 1 1 0 0
222 29 m 8 5 1
1 6 0 6 1 0 0 1 1 1 1 0 0 0
783 33 m 1 2 7
7 0 0 0 7 0 0 0 7 0 0 0 7 0
781 39 m 1.5 2.5 2
2 0 2 0 2 0 0 0 2 2 2 0 0 0
850 22 f 2 1 1
1 0 0 0 1 1 1 0 5 0 0 1 0 0
493 36 f 1 0 5
0 0 0 0 7 0 0 0 0 0 0 0 0 0
967 46 f 2 4 7
7 5 0 5 7 0 0 0 4 7 4 0 0 0
856 41 m 2 2 4
7 4 0 0 7 4 0 4 0 0 0 7 0 0
546 25 m 5 5 8
8 8 0 0 0 0 0 0 0 0 0 0 0 0
222 27 f 4 4 3
2 2 2 3 7 7 0 2 2 2 3 3 3 0

```

```

688 23 m 9 3 3
3 3 3 3 3 7 0 0 3 0 0 0 0 0
849 26 m .5 .5 8
8 0 0 0 8 0 0 0 0 8 0 0 0 0
783 29 f 3 3 1
1 0 0 0 4 0 0 4 1 0 1 0 0 0
856 34 f 1.5 2 1
7 0 0 0 7 0 0 7 4 0 0 7 0 0
966 33 m 3 5 4
7 0 0 0 7 4 5 0 7 0 0 7 4 4
493 34 f 2 5 1
1 0 0 0 1 0 0 0 7 0 1 1 8 0
467 29 m 2 4 2
2 0 0 0 2 0 0 2 2 2 2 2 2 2
677 28 f 1 4 1
1 1 1 1 1 0 0 0 1 0 1 0 0 0
781 27 m 2 2 1
1 0 1 0 4 2 4 0 2 2 1 0 1 4
467 24 m 4 4 1
7 1 0 1 1 1 0 7 1 0 0 0 0 0
859 26 m 5 5 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1
848 27 m 7 2 5
7 5 0 5 4 5 0 0 0 7 4 4 0 4
677 25 f 1 2 8
8 0 0 0 0 5 0 0 8 0 0 0 2 0
222 26 f 3.5 0 2
2 0 0 0 2 0 0 0 0 0 0 0 0 0
833 32 m 1 2 1
1 0 0 0 1 0 0 0 5 0 1 0 0 0
781 28 m 2 .5 7
7 0 0 0 7 0 0 0 4 0 0 0 0 0
783 28 f 1 1 1
1 0 0 0 1 0 0 0 0 0 1 1 0 0
222 28 f 5 5 2
2 6 6 2 2 0 0 0 2 2 0 0 2 2
851 33 m 4 5 3
1 0 0 0 7 3 0 3 3 3 3 3 7 5
859 39 m 2 1 1
1 0 0 0 1 0 0 0 0 0 0 1 0 0
848 45 m 2 2 7
7 0 0 0 7 0 0 0 7 0 0 0 0 0
467 37 m 2 2 7
7 0 0 0 0 7 0 0 0 7 0 0 7 0
859 32 m .25 .25 1
1 0 0 0 0 0 0 0 1 0 0 0 0 0

```

STATEPOP

```

data statepop;
  input State $ CityPop_1990 CityPop_2000 NonCityPop_1990 NonCityPop_2000 Region @@;
  label citypop_1990= '1990 metropolitan pop in millions'

```

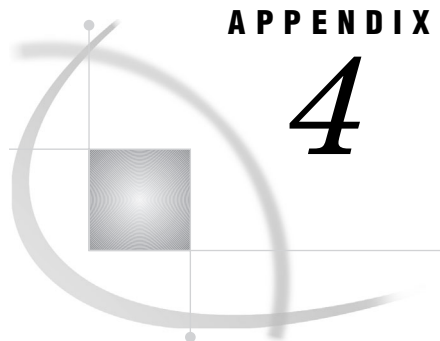


```

noncitypop_1990='1990 nonmetropolitan pop in millions'
citypop_2000=   '2000 metropolitan pop in millions'
noncitypop_2000='2000 nonmetropolitan pop in million'
region='Geographic region';

datalines;
ME   .443   .467   .785   .808   1   NH   .659   .740   .450   .496   1
VT   .152   .169   .411   .439   1   MA   5.788   6.088   .229   .261   1
RI   .938   .986   .065   .062   1   CT   3.148   3.257   .140   .149   1
NY  16.516  17.473  1.475  1.503  1   NJ   7.730   8.414   .A     .A     1
PA  10.084  10.392  1.799  1.890  1   DE   .553   .627   .113   .157   2
MD   4.438   4.911   .343   .385   2   DC   .607   .572   .A     .A     2
VA   4.775   5.528   1.414   1.550   2   WV   .748   .766   1.045   1.043   2
NC   4.380   5.437   2.253   2.612   2   SC   2.422   2.807   1.064   1.205   2
GA   4.351   5.667   2.127   2.520   2   FL  12.024  14.837   .915   1.145   2
KY   1.780   1.973   1.907   2.069   2   TN   3.311   3.862   1.567   1.827   2
AL   2.797   3.109   1.244   1.338   2   MS   .874   1.024   1.701   1.821   2
AR   1.109   1.321   1.242   1.352   2   LA   3.160   3.370   1.061   1.099   2
OK   1.870   2.098   1.276   1.352   2   TX  14.166  17.692   2.821   3.160   2
OH   8.826   9.214   2.021   2.139   3   IN   3.962   4.390   1.582   1.691   3
IL   9.574  10.542   1.857   1.878   3   MI   7.698   8.169   1.598   1.769   3
WI   3.331   3.640   1.561   1.723   3   MN   3.011   3.463   1.364   1.456   3
IA   1.200   1.326   1.577   1.600   3   MO   3.491   3.795   1.626   1.800   3
ND   .257   .284   .381   .358   3   SD   .221   .261   .475   .494   3
NE   .787   .900   .791   .811   3   KS   1.333   1.521   1.145   1.167   3
MT   .270   .306   .529   .597   4   ID   .362   .508   .645   .786   4
WY   .134   .148   .319   .346   4   CO   2.779   3.608   .515   .694   4
NM   .842   1.035   .673   .784   4   AZ   3.202   4.527   .463   .604   4
UT   1.341   1.708   .382   .525   4   NV   1.014   1.748   .188   .251   4
WA   4.036   4.899   .830   .995   4   OR   2.056   2.502   .787   .919   4
CA  28.797  32.750   .961   1.121   4   AK   .226   .260   .324   .367   4
HI   .836   .876   .272   .335   4
;
run;

```

APPENDIX

4

Recommended Reading

Recommended Reading 1673

Recommended Reading

For a complete list of SAS publications, refer to the current *SAS Publishing Catalog*. The catalog is produced twice a year. To order books or to receive a free copy of the catalog, write, call, or fax the Institute (or access the online version of the *SAS Publishing Catalog* via the World Wide Web).

When you order a title, we will provide you with the most current edition that is available.

SAS Institute
 Fulfillment Services Dept.
 SAS Campus Dr.
 Cary, NC 27513
 Telephone: 1-800-727-3228*
 Fax: 919-677-8166
 E-mail: sasbook@sas.com
 Web site: www.sas.com/pubs

* For other SAS Institute business, call 919-677-8000.

Customers outside the U.S. should contact their local SAS office.

The recommended reading list for this title includes

The Little SAS Book: A Primer, Second Edition

Output Delivery System: The Basics

PROC TABULATE by Example

SAS Guide to Report Writing: Examples

SAS Language Reference: Concepts

SAS Language Reference: Dictionary

SAS Output Delivery System User's Guide

SAS Programming by Example

SAS SQL Procedure User's Guide

Step-by-Step Programming with Base SAS Software

Index

A

- ACCELERATE= option
 - ITEM statement (PMENU) 786
- ACROSS option
 - DEFINE statement (REPORT) 987
 - PROC FORMS statement 498
- across variables 946, 987
- activities data set 81, 102
- adjusted logit odds ratio 579
- adjusted odds ratio 577
- AFTER= option
 - PROC CPORT statement 310
- AGE statement
 - DATASETS procedure 334
- aging data sets 405
- aging files 334
- AGREE option
 - TABLES statement (FREQ) 531
- AIRCRAFT data set 1615
- ALIGN= option
 - PROC FORMS statement 498
- ALL class variable 1324
- ALL keyword 1178
- ALL option
 - PROC UNIVARIATE statement 1445
 - TABLES statement (FREQ) 531
- ALLOBS option
 - PROC COMPARE statement 215
- ALLSTATS option
 - PROC COMPARE statement 215
- ALLVARS option
 - PROC COMPARE statement 215
- ALPHA option
 - PROC CORR statement 269
 - EXACT statement (FREQ) 523
 - HISTOGRAM statement (UNIVARIATE) 1458
 - PROBPLOT statement (UNIVARIATE) 1487
 - PROC MEANS statement 654
 - PROC TABULATE statement 1268
 - PROC UNIVARIATE statement 1445
 - QQPLOT statement (UNIVARIATE) 1500
 - TABLES statement (FREQ) 531
- ALTER= option
 - AGE statement (DATASETS) 334
 - CHANGE statement (DATASETS) 343
 - COPY statement (DATASETS) 347
 - DELETE statement (DATASETS) 353
 - EXCHANGE statement (DATASETS) 357
 - MODIFY statement (DATASETS) 367
 - PROC DATASETS statement 331
 - REPAIR statement (DATASETS) 371
 - SELECT statement (DATASETS) 374
- ALTER TABLE statement
 - SQL procedure 1125
- alternative hypotheses 1607
- ANALYSIS option
 - DEFINE statement (REPORT) 987
- analysis variables 946, 987
- SUMMARY procedure 1259
- weighting 1000
- weights for 59
- Anderson-Darling EDF test 1522
- Anderson-Darling statistic 1522
- ANNOKEY option
 - HISTOGRAM statement (UNIVARIATE) 1459
 - PROBPLOT statement (UNIVARIATE) 1488
 - QQPLOT statement (UNIVARIATE) 1500
- ANNOTATE= option
 - HISTOGRAM statement (UNIVARIATE) 1459
 - PROBPLOT statement (UNIVARIATE) 1488
 - PROC UNIVARIATE statement 1445
 - QQPLOT statement (UNIVARIATE) 1500
- ANOVA statistic 576
- ANSI Standard for SQL 1204
- APPEND 73
- APPEND procedure 73
 - overview 71
 - syntax 73
- APPEND statement
 - DATASETS procedure 335
- appending data
 - APPEND statement (DATASETS) 337
 - SET statement 337
- appending data sets 335
 - APPEND procedure 340
 - APPEND statement (DATASETS) 340
 - compressed data sets 338
 - data sets with different variables 339
 - generation groups for 339
 - indexed data sets 338
 - integrity constraints and 339
 - password-protected data sets 337
 - restricting observations 337
 - system failures 340
 - variables with different attributes 339
- appending observations 71
- APPENDVER= option
 - APPEND statement (DATASETS) 336
- arc-sine distribution 1531
- arithmetic mean 1581, 1587
- arithmetic operators 1205
- ASCENDING option
 - CHART procedure 178
 - CLASS statement (MEANS) 662
 - CLASS statement (TABULATE) 1276
- ASCII collating sequence 1094, 1101
- ASCII option
 - PROC SORT statement 1094
- ASIS option
 - PROC CPORT statement 310
- asterisk (*) notation 1143
- asymptotic tests 539, 551
- ATTR= option
 - TEXT statement (PMENU) 792
- ATTRIB statement 53
- audit files 342
- AUDIT statement
 - DATASETS procedure 341
- AUTOLABEL option
 - OUTPUT statement (MEANS) 671
- AUTONAME option
 - OUTPUT statement (MEANS) 671
- AXIS= option
 - CHART procedure 178
 - PLOT statement (TIMEPLOT) 1372

B

- bar charts 166
 - horizontal 175, 192, 1448
 - percentage charts 186
 - side-by-side 190
 - vertical 177, 187
- BARWIDTH= option
 - HISTOGRAM statement (UNIVARIATE) 1459
- BASE= argument
 - APPEND statement (DATASETS) 335
- base data set 210
- BASE= option
 - PROC COMPARE statement 216
- BASETYPE= option
 - PROC DBCSTAB statement 408
- batch mode
 - creating printer definitions 893

printing from 957
 BATCH option, PROC DISPLAY statement 414
 BDT option
 TABLES statement (FREQ) 531
 BEST= option
 PROC CORR statement 269
 beta distribution 1530, 1536
 BETA option
 HISTOGRAM statement (UNIVARI-
 ATE) 1459
 PROBPLOT statement (UNIVARIATE) 1488
 QQPLOT statement (UNIVARIATE) 1500
 HISTOGRAM statement (UNIVARI-
 ATE) 1459
 PROBPLOT statement (UNIVARIATE) 1488
 QQPLOT statement (UNIVARIATE) 1501
 BETWEEN condition 1155
 BETWEEN= option
 PROC FORMS statement 498
 BINOMIAL option
 TABLES statement (FREQ) 532
 binomial proportions 560, 599
 BINOMIALC option
 TABLES statement (FREQ) 532
 block charts 167, 173
 for BY groups 194
 BLOCK statement
 CHART procedure 173
 body file 44
 BOTH option
 CLEAR statement (TRANTAB) 1415
 LIST statement (TRANTAB) 1416
 SAVE statement (TRANTAB) 1418
 Bowker's test of symmetry 569
 BOX option
 PLOT statement (PLOT) 735
 PROC REPORT statement 962
 TABLE statement (TABULATE) 1283
 box plots 1448, 1513
 side-by-side 1448, 1514
 BREAK automatic variable 953
 break lines 952
 BREAK automatic variable 953
 creating 952
 order of 953, 977, 999
 BREAK statement
 REPORT procedure 974
 BREAK window, REPORT procedure 1001
 breaks 952
 Breslow-Day test 580
 BRIEFSSUMMARY option
 PROC COMPARE statement 216
 browsing external files 627
 BTRIM function (SQL) 1155
 BTYPE= option
 PROC DBCSTAB statement 408
 BY-group processing 19, 55
 formats and 30
 BY groups
 transposing 1400
 BY lines
 suppressing 19
 BY statement 54
 CALENDAR procedure 87
 CHART procedure 174
 COMPARE procedure 220
 CORR procedure 273

example 55
 FORMS procedure 501
 FREQ procedure 521
 MEANS procedure 660, 664, 687
 PLOT procedure 732
 PRINT procedure 829, 832
 procedures supporting 55
 RANK procedure 914
 REPORT procedure 978
 SORT procedure 1100
 STANDARD procedure 1248
 TABULATE procedure 1275
 TIMEPLOT procedure 1369
 TRANSPOSE procedure 1391
 UNIVARIATE procedure 1451, 1484
 BY variable names
 inserting into titles 22

C

C= option
 HISTOGRAM statement (UNIVARI-
 ATE) 1459
 PROBPLOT statement (UNIVARIATE) 1488
 QQPLOT statement (UNIVARIATE) 1501
 CACHED_UPDATES= option
 PROC SQL statement 1120
 CAL variable 101
 calculated columns 1156
 CALCULATED component 1156
 CALEDATA= option
 PROC CALENDAR statement 81
 CALENDAR 80
 calendar data set 81, 100, 101, 104
 CALENDAR procedure 80
 activities data set 102
 activity lines 108
 automating scheduling tasks 129
 BY statement 87
 calendar, defined 99
 calendar data set 104
 calendar types 97
 CALID statement 88, 101
 concepts 97
 customizing appearance 108
 date versus datetime values 104
 default calendars 98
 default workshifts 105, 106
 definitions 99
 DUR statement 89
 duration 89
 examples 108
 FIN statement 90
 holiday duration 91
 holidays data set 103
 HOLIDUR statement 90
 HOLIFIN statement 91
 HOLISTART statement 92
 HOLIVAR statement 92
 input data sets 102
 MEAN statement 93
 missing values 106
 multiple calendars 87, 100
 OUTDUR statement 94
 OUTFIN statement 94
 output, effects of size 108

output, quantity of 107
 OUTSTART statement 95
 overview 74
 PROC CALENDAR statement 81
 project management tasks 77
 results 107
 schedule calendars 75, 76, 97
 START statement 95
 SUM statement 96
 summary calendars 78, 98, 103
 syntax 80
 task tables 80, 81
 VAR statement 96
 workdays data set 105, 106
 calendar reports 99
 CALID statement
 CALENDAR procedure 88, 101
 CALL DEFINE statement
 REPORT procedure 979
 CAPS option, PROC FSLIST statement 629
 Cartesian product 1166, 1167
 case-control studies 564, 1238
 adjusted logit 579
 Mantel-Haenszel adjusted 578
 CASE expression 1157
 case-record data 541
 CATALOG 144
 CATALOG= argument
 PROC CATALOG statement 145
 PROC DISPLAY statement 414
 CATALOG= option
 CONTENTS statement (CATALOG) 147
 PROC DBCSTAB statement 408
 PROC PMENU statement 781
 CATALOG procedure 144
 CHANGE statement 146
 concatenating catalogs 157
 concepts 154
 CONTENTS statement 147
 COPY statement 148
 DELETE statement 150
 ending a step 154
 entry type specification 155
 ENTRYTYPE= options 155, 156
 error handling 154
 examples 158
 EXCHANGE statement 150
 EXCLUDE statement 151
 interactive processing 154
 MODIFY statement 152
 overview 143
 PROC CATALOG statement 145
 RUN groups 154
 SAVE statement 153
 SELECT statement 153
 syntax 144
 task tables 144, 145, 148
 catalogs
 concatenating 157
 copying entries 148, 153, 158
 deleting entries 146, 150, 153, 158
 displaying entry contents 162
 entry descriptions 152, 162
 excluding entries for copying 151
 exporting 317
 exporting entries 318, 321
 format catalogs 467

- listing contents of 147
- managing entries 143
- moving entries 158
- PMENU entries 781, 788, 794
- renaming entries 146, 162
- repairing 371
- routing log or output to entries 886
- storing informats/formats 467
- switching names of two entries 150
- writing to transport files 307
- categories 1264
- CAXIS= option
 - HISTOGRAM statement (UNIVARI-ATE) 1460
 - PROBPLOT statement (UNIVARIATE) 1489
 - QQPLOT statement (UNIVARIATE) 1501
- CBARLINE= option
 - HISTOGRAM statement (UNIVARI-ATE) 1460
- CC option
 - FSLIST command 631
 - PROC FORMS statement 498
 - PROC FSLIST statement 629
- cell count data 541
- CELLCHI2 option
 - TABLES statement (FREQ) 532
- CENSUS data set 1616
- CENTER option
 - DEFINE statement (REPORT) 987
 - PROC REPORT statement 962
- centiles
 - indexed variables and 362
- CENTILES option
 - CONTENTS statement (DATASETS) 344
- CFILL= option
 - HISTOGRAM statement (UNIVARI-ATE) 1460
 - INSET statement (UNIVARIATE) 1477
- CFILLH= option
 - INSET statement (UNIVARIATE) 1477
- CFRAME= option
 - HISTOGRAM statement (UNIVARI-ATE) 1460
 - INSET statement (UNIVARIATE) 1477
 - PROBPLOT statement (UNIVARIATE) 1489
 - QQPLOT statement (UNIVARIATE) 1501
- CFRAMESIDE= option
 - HISTOGRAM statement (UNIVARI-ATE) 1460
 - PROBPLOT statement (UNIVARIATE) 1489
 - QQPLOT statement (UNIVARIATE) 1501
- CFRAME TOP= option
 - HISTOGRAM statement (UNIVARI-ATE) 1461
 - PROBPLOT statement (UNIVARIATE) 1489
 - QQPLOT statement (UNIVARIATE) 1501
- CFREQ option
 - CHART procedure 179
- CGRID= option
 - HISTOGRAM statement (UNIVARI-ATE) 1461
 - PROBPLOT statement (UNIVARIATE) 1489
 - QQPLOT statement (UNIVARIATE) 1501
- CHANGE statement
 - CATALOG procedure 146
 - DATASETS procedure 342
- character data
 - converting to numeric values 480
- character sets, and translation tables 1410
- character strings
 - converting to lowercase 1176
 - converting to uppercase 1197
 - returning a substring 1189
 - trimming 1155
 - writing ranges for 490
- character values
 - formats for 476
- character variables
 - sorting orders for 1101
- CHARITY data set 1617
- CHART 171
- CHART procedure 171
 - bar charts 166, 190
 - block charts 167, 173, 194
 - BLOCK statement 173
 - BY statement 174
 - concepts 183
 - customizing charts 177
 - examples 184
 - formatting characters 171
 - frequency counts 184
 - HBAR statement 175
 - horizontal bar charts 175, 192
 - missing values 181, 183
 - options 178
 - overview 165
 - percentage bar charts 186
 - pie charts 168, 175
 - PIE statement 175
 - PROC CHART statement 171
 - results 183
 - star charts 169, 176
 - STAR statement 176
 - syntax 171
 - task table 177
 - variable characteristics 183
 - VBAR statement 177
 - vertical bar charts 177, 187
- charts 165
 - bar charts 166, 186, 190
 - block charts 167, 173, 194
 - customizing 177
 - horizontal bar charts 175, 192, 1448
 - pie charts 168, 175
 - star charts 169, 176
 - vertical bar charts 177, 187
- CHARTYPE option
 - PROC MEANS statement 654
- CHEADER= option
 - INSET statement (UNIVARIATE) 1477
- check boxes 782
 - active versus inactive 782
 - color of 782
 - definition of 784
- CHECKBOX statement
 - PMENU procedure 782
- chi-square tests 546
 - continuity-adjusted 548
 - for one-way tables 546, 596
 - for two-way tables 547
 - likelihood-ratio 547
 - Mantel-Haenszel 548
 - output data set for 605
- CHISQ option
 - TABLES statement (FREQ) 532
- CHREF= option
 - HISTOGRAM statement (UNIVARI-ATE) 1461
 - PROBPLOT statement (UNIVARIATE) 1489
 - QQPLOT statement (UNIVARIATE) 1501
- CIBASIC option
 - PROC UNIVARIATE statement 1445
- CIMPORT procedure 200
 - Data Control Blocks 205
 - examples 205
 - EXCLUDE statement 203
 - excluding files or entries 203
 - overview 199
 - PROC CIMPORT statement 200
 - results 205
 - SELECT statement 204
 - specifying entries or files to import 204
 - syntax 200
 - task table 200
 - translation tables with 1411
- CIPCTLDF option
 - PROC UNIVARIATE statement 1445
- CIPCTLNORMAL option
 - PROC UNIVARIATE statement 1446
- CIQUANTDF option
 - PROC UNIVARIATE statement 1446
- CIQUANTNORMAL option
 - PROC UNIVARIATE statement 1446
- CL option
 - TABLES statement (FREQ) 533
- CLASS statement
 - MEANS procedure 661
 - TABULATE procedure 1276
 - TIMEPLOT procedure 1370
 - UNIVARIATE procedure 1452
- class variables 675
 - BY statement (MEANS) with 687
 - CLASSDATA= option (MEANS) with 689
 - combinations of 1312
 - computing descriptive statistics with 685
 - formatting 1292
 - identifying 1276
 - level value headings 1280
 - missing 1302, 1303, 1304
 - missing values 704, 1279
 - multilabel value formats with 693
 - ordering 676
 - preloaded formats with 696, 1315
 - TIMEPLOT procedure 1370
- CLASSDATA= option
 - PROC MEANS statement 654, 689
 - PROC TABULATE statement 1268
- classifying data 27
- CLASSLEV statement
 - TABULATE procedure 1280
- CLEAR statement, TRANTAB procedure 1415
- CLEAR SASUSER option
 - PROC REGISTRY statement 926
- CLM keyword 1585
- CLONE option
 - COPY statement (DATASETS) 347
- CMH option
 - TABLES statement (FREQ) 533
- CMH1 option
 - TABLES statement (FREQ) 533

- CMH2 option
 - TABLES statement (FREQ) 533
- CNTLIN= option
 - PROC FORMAT statement 444
- CNTLOUT= option
 - PROC FORMAT statement 444, 446
- COALESCE function (SQL) 1158
- Cochran-Armitage test for trend 566, 611
- Cochran-Mantel-Haenszel statistics 574, 609
- Cochran's Q test 574
 - testing marginal homogeneity 618
- coefficient of variation 1580, 1593
- cohort studies 565, 579
- COLOR= option
 - BREAK statement (REPORT) 974
 - CHECKBOX statement (PMENU) 782
 - DEFINE statement (REPORT) 988
 - HISTOGRAM statement (UNIVARIATE) 1461
 - PROBPLOT statement (UNIVARIATE) 1489
 - QQPLOT statement (UNIVARIATE) 1502
 - RBREAK statement (REPORT) 997
 - RBUTTON statement (PMENU) 790
 - TEXT statement (PMENU) 793
- column aliases 1143
- column attributes 1160
 - REPORT procedure 979
- column-definition component 1159
- column dimension 1265
- column-header option
 - DEFINE statement (REPORT) 988
- column headings
 - customizing 1322
 - customizing text in 840
 - layout of 836
 - orientation of 822
 - variable labels as 822
- column-modifier component 1160
- column modifiers 1205
- column-name component 1161
- COLUMN statement
 - REPORT procedure 981
- column width
 - in page layout 836
- columns 1115
 - See also* variables
 - aliases 1143
 - attributes 1160
 - calculated 1156
 - combinations of values 1236
 - indexes on 1127, 1129, 1141
 - inserting values 1140
 - modifiers 1205
 - returning values 1158
 - selecting 1142, 1161
 - storing values of 1144
 - updating values 1153
- COLWIDTH= option
 - PROC REPORT statement 962
- COMMAND option
 - PROC REPORT statement 963
- COMMIT statement (SQL) 1206
- COMPARE 213
- COMPARE= option
 - PROC COMPARE statement 216
- COMPARE procedure 213
 - BY processing 221
- BY statement 220
 - comparing selected variables 224
 - comparing unsorted data 222
 - comparing variables in same data set 224
 - comparing variables of different names 224
 - comparison by position of observations 224
 - concepts 224
 - duplicate ID values 222
 - equality criterion 226
 - examples 239
 - ID statement 222
 - ID variables 222, 225
 - listing variables for matching 222
 - log and 228
 - macro return codes 228
 - output 230
 - output data set 236
 - output statistics data set 237
 - overview 209
 - PROC COMPARE statement 214
 - restricting the comparison 223
 - results 228
 - syntax 213
 - task tables 213, 214
 - VAR statement 223
 - variable formats 228
 - WITH statement 224
- COMPAREREG1 option
 - PROC REGISTRY statement 926
- COMPAREREG2 option
 - PROC REGISTRY statement 927
- COMPARETO= option
 - PROC REGISTRY statement 927
- comparison data set 210
- compass point position 1479
- COMPLETECOLS option
 - PROC REPORT statement 963
- COMPLETEROWS option
 - PROC REPORT statement 963
- COMPLETETYPES option
 - PROC MEANS statement 655
- composite indexes 1129
- compound names 1025
- COMPRESS option
 - PROC FREQ statement 519
- compressed data sets
 - appending 338
- compute blocks 949
 - compound names in 1025
- COMPUTE statement (REPORT) 983
 - contents of 950
 - processing 952
 - purpose of 950
 - referencing report items in 951
- COMPUTE statement
 - REPORT procedure 983
- COMPUTE window, REPORT procedure 1004
- COMPUTED option
 - DEFINE statement (REPORT) 989
- COMPUTED VAR window, REPORT procedure 1004
- computed variables 947, 989
 - storing 1072
- concatenating catalogs 157
- concatenating data sets 403
- CONDENSE option
 - TABLE statement (TABULATE) 1283
- confidence limits 522, 551
 - for parameters of normal distribution 1517
 - for quantiles 1528
 - for the mean 650, 699, 1268
 - MEANS procedure 679
- confidence limits, for the mean
 - keywords and formulas 1585
 - one-sided, above the mean 1586
 - one-sided, below the mean 1585
 - two-sided 1585
- CONNECT statement
 - SQL procedure 1128
- CONNECTION TO component 1162
- CONSTRAINT= option
 - COPY statement (DATASETS) 349
 - PROC CPORT statement 310
- CONTAINS condition 1163
- CONTENT= option
 - PROC PRINT statement 821
- CONTENTS 258
- CONTENTS= option
 - PROC REPORT statement 963
 - PROC TABULATE statement 1268
 - TABLE statement (TABULATE) 1284
 - TABLES statement (FREQ) 533
- CONTENTS procedure 258
 - overview 257
 - PROC CONTENTS statement 258
 - syntax 258
 - task table 258
 - versus CONTENTS statement (DATASETS) 347
- CONTENTS statement
 - CATALOG procedure 147
 - DATASETS procedure 344
- contingency coefficient 550
- contingency tables 515, 601, 1347
- continuation message 1265
- continuity-adjusted chi-square test 548
- CONTOUR= option, PLOT statement (PLOT) 735
- contour plots 735, 759
- CONVERGE= option
 - TABLES statement (FREQ) 534
- conversion tables
 - creating 409
 - for double-byte character sets 407
 - in Japanese 410
- converting files 307
- converting SAS files 199
- COPIES= option
 - PROC FORMS statement 499
- COPY 260
- COPY procedure 260
 - concepts 260
 - overview 259
 - syntax 260
 - transporting data sets between hosts 260
 - versus COPY statement (DATASETS) 352
- COPY statement
 - CATALOG procedure 148
 - DATASETS procedure 347
 - TRANSPOSE procedure 1393
- copying data libraries
 - entire data library 350
- copying datasets
 - with long variable names 351

copying files 259
 COPY statement (DATASETS) versus COPY procedure 352
 excluding files 358
 member type specification 350
 password-protected files 351
 selecting files 350, 374
 CORR 267
 CORR procedure 267
 BY statement 273
 computer resources 277
 concepts 276
 correlation coefficients, interpreting 276
 examples 291
 FREQ statement 274
 missing values 287
 ODS table names 287
 output 289
 output data sets 290
 overview 263
 PARTIAL statement 274
 PROC CORR statement 268
 results 287
 statistical computations 279
 syntax 267
 task tables 267, 268
 VAR statement 275
 WEIGHT statement 275
 WITH statement 276
 corrected sum of squares 1580
 corrected sums of squares and crossproducts 269
 correlated subqueries 1187
 correlation coefficients 263
 interpreting 276
 limited combinations of 276
 partial correlations 302
 printing, for each variable 269, 271
 rectangular 295
 suppressing probabilities 270
 correlation statistic 576
 CORRESPONDING keyword 1177
 COUNT(*) function 1191
 COV option
 PROC CORR statement 269
 covariances 269, 272
 CPERCENT option
 CHART procedure 179
 CPM procedure 77, 129
 CPORT 308
 CPORT procedure 308
 concepts 316
 Data Control Blocks 317
 examples 317
 EXCLUDE statement 313
 excluding files or entries 313
 including files or entries 314
 overview 307
 password-protected data sets 316
 PROC CPORT statement 308
 results 317
 SELECT statement 314
 syntax 308
 task table 308
 translation tables 315
 translation tables with 1411
 TRANTAB statement 315

CPROP= option
 HISTOGRAM statement (UNIVARIATE) 1461
 Cramer-von Mises EDF test 1522
 Cramer-von Mises statistic 1523
 Cramer's V 550
 CREATE INDEX statement
 SQL procedure 1129
 CREATE TABLE statement
 SQL procedure 1130
 CREATE VIEW statement
 SQL procedure 1134
 CRITERION= option
 PROC COMPARE statement 216
 Cronbach's coefficient alpha 284
 calculating and printing 269
 example 299
 for estimating reliability 264
 for multiple-item mixed-rating scale 299
 cross joins 1170
 crosstabulation tables 515, 528, 1347
 CSHADOW= option
 INSET statement (UNIVARIATE) 1477
 CSS keyword 1580
 CSSCP option
 PROC CORR statement 269
 CTEXT= option
 HISTOGRAM statement (UNIVARIATE) 1461
 INSET statement (UNIVARIATE) 1477
 PROBPLOT statement (UNIVARIATE) 1489
 QQPLOT statement (UNIVARIATE) 1502
 CTEXTSIDE= option
 HISTOGRAM statement (UNIVARIATE) 1461
 CTEXTTOP= option
 HISTOGRAM statement (UNIVARIATE) 1461
 CUMCOL option
 TABLES statement (FREQ) 534
 cumulative distribution function 1587
 CUSTOMER_RESPONSE data set 1619
 customizing output 48
 excluding output objects 48
 for output objects 50
 SAS output 48
 selecting output objects 48
 style definitions 47
 CV keyword 1580
 CVREF= option
 HISTOGRAM statement (UNIVARIATE) 1462
 PROBPLOT statement (UNIVARIATE) 1490
 QQPLOT statement (UNIVARIATE) 1502

D

DANISH option
 PROC SORT statement 1095
 DATA= argument
 PROC EXPORT statement 428
 DATA COLUMNS window, REPORT procedure 1005
 data component 39
 Data Control Blocks (DCBs) 205, 317
 data entry programs 811

data libraries
 copying entire library 350
 copying files 347
 deleting files 353, 373
 exchanging file names 357
 listing files in 326
 printing directories of 257, 344
 processing all data sets 30
 renaming files 342
 USER data library 17
 writing to transport files 307
 DATA option
 INSET statement (UNIVARIATE) 1477
 APPEND statement (DATASETS) 336
 CONTENTS statement (DATASETS) 345
 PROC CALENDAR statement 81
 PROC CHART statement 171
 PROC COMPARE statement 216
 PROC CORR statement 270
 PROC DBCSTAB statement 408
 PROC FORMS statement 499
 PROC FREQ statement 519
 PROC MEANS statement 655
 PROC OPTLOAD statement 722
 PROC OPTSAVE statement 724
 PROC PLOT statement 729
 PROC PRINT statement 821
 PROC PRTDEF statement 894
 PROC RANK statement 912
 PROC REPORT statement 963
 PROC SORT statement 1096
 PROC STANDARD statement 1246
 PROC TABULATE statement 1268
 PROC TIMEPLOT statement 1368
 PROC TRANSPOSE statement 1390
 PROC UNIVARIATE statement 1446
 DATA SELECTION window, REPORT procedure 1005
 data set options 17
 SQL procedure with 1197
 data sets
 aging 405
 appending 335
 appending observations to 71
 attribute listings 326
 base data set 210
 comparing 209
 comparing variables in 243, 246
 comparison data set 210
 compressed, appending to 338
 concatenating 403
 content descriptions 344
 contents of 257
 creating formats from 482
 creating informats/formats 442
 data component versus 39
 data set options 17
 describing 400
 exporting 319
 exporting data 427
 for examples 1615
 importing data 633
 indexed, appending to 338
 indexes 363
 input data sets 19
 integrity constraints 326
 loading system options from 721

- managing with DATASETS procedure 326
- modifying 398
- modifying attributes 326
- modifying variables 326
- output data sets 44
- password-protected, appending 337
- password-protected, transporting 316
- permanent 16
- printing all data sets in a library 875
- printing observations 817
- processing all data sets in a library 30
- producing with OUTPUT destination 40
- renaming variables 371
- repairing 371
- saving system option settings in 723
- standardizing variables 1243
- storing informat/format descriptions 442
- temporary 16
- transporting between hosts 260, 352
- writing printer attributes to 907
- writing to transport files 307
- DATA step variables 1024, 1030
- DATA step views 1115
- DATA step window applications, menus for 807
- data summaries 1191, 1324
- data summarization tools 650, 1257, 1436
- DATABASE= statement
 - EXPORT procedure 433
 - IMPORT procedure 640
- DATAFILE= argument
 - PROC IMPORT statement 634
- DATAROW= statement
 - IMPORT procedure 638
- DATASETS 330
- DATASETS procedure 330
 - AGE statement 334
 - APPEND statement 335
 - AUDIT statement 341
 - CHANGE statement 342
 - concepts 375
 - CONTENTS statement 344
 - COPY statement 347
 - DELETE statement 353
 - directory listings 381
 - ending 377
 - error handling 376
 - examples 392
 - EXCHANGE statement 357
 - EXCLUDE statement 358
 - execution of 375
 - FORMAT statement 358
 - IC CREATE statement 359
 - IC DELETE statement 361
 - IC REACTIVATE statement 362
 - INDEX CENTILES statement 362
 - INDEX CREATE statement 363
 - INDEX DELETE statement 364
 - INFORMAT statement 365
 - LABEL statement 366
 - member types 379
 - MODIFY statement 366
 - ODS and 382
 - output data sets 386
 - overview 326
 - passwords 377
 - PROC DATASETS statement 330, 378
 - procedure output 382
 - referencing generation group version 380
 - RENAME statement 371
 - REPAIR statement 371
 - restricting member types 378
 - results 381
 - RUN-group processing 375
 - RUN groups, forcing processing 377
 - SAVE statement 373
 - SELECT statement 374
 - syntax 330
 - task tables 330, 344, 366
- DATATYPE= option
 - PICTURE statement (FORMAT) 451
- date formats 478
- DATECOPY option
 - PROC CPORT statement 310
 - PROC SORT statement 1096
- COPY statement (DATASETS) 349
- DATETIME option
 - PROC CALENDAR statement 82
- DAYLENGTH= option
 - PROC CALENDAR statement 82
- DBCSLANG= option
 - PROC DBCSTAB statement 408
- DBCSTAB 407
- DBCSTAB procedure 407
 - conversion tables, creating 409
 - conversion tables, Japanese 410
 - examples 409
 - overview 407
 - PROC DBCSTAB statement 407
 - syntax 407
 - when to use 408
- DBMS connections
 - ending 1138
 - establishing 1128
 - LIBNAME statement for 1198
 - sending DBMS statements to 1139
 - SQL Procedure Pass-Through Facility for 1198
 - storing in views 1135
- DBMS= option
 - PROC EXPORT statement 429
 - PROC IMPORT statement 635
- DBMS queries 1162
- DBMS tables
 - exporting 432
 - importing 640
- DBPWD= statement
 - EXPORT procedure 433
 - IMPORT procedure 640
- DCBs (Data Control Blocks) 205, 317
- DDNAME= argument, PROC FSLIST statement 628
- DDNAME= option
 - PROC DATASETS statement 332
 - PROC FORMS statement 499
- DEBUGOFF option
 - PROC REGISTRY statement 927
- DEBUGON option
 - PROC REGISTRY statement 927
- DECSEP= option
 - PICTURE statement (FORMAT) 451
- DEF option
 - PROC UNIVARIATE statement 1448
- DEFAULT= option
 - FORMAT procedure 462
- INVALUE statement (FORMAT) 448
- PICTURE statement (FORMAT) 450
- RADIOBOX statement (PMENU) 789
- VALUE statement (FORMAT) 460
- DEFINE option
 - PROC OPTIONS statement 716
- DEFINE statement
 - REPORT procedure 986
- DEFINITION window, REPORT procedure 1006
- DELETE option
 - PROC PRDEF statement 894
- DELETE statement
 - CATALOG procedure 150
 - DATASETS procedure 353
 - SQL procedure 1136
- delimited files
 - exporting 432, 434
 - importing 638, 641
- DELIMITER= statement
 - EXPORT procedure 432
 - IMPORT procedure 638
- denominator definitions 1347
- density curves 1561
- density function 1587
- DESC option
 - PROC PMENU statement 781
 - PROC DBCSTAB statement 408
- DESCENDING option
 - BY statement 54
 - BY statement (CALENDAR) 87
 - BY statement (CHART) 174
 - BY statement (COMPARE) 221
 - BY statement (CORR) 273
 - BY statement (FORMS) 501
 - BY statement (FREQ) 521
 - BY statement (MEANS) 661
 - BY statement (PLOT) 732
 - BY statement (PRINT) 830
 - BY statement (RANK) 914
 - BY statement (REPORT) 978
 - BY statement (SORT) 1100
 - BY statement (STANDARD) 1248
 - BY statement (TABULATE) 1276
 - BY statement (TIMEPLOT) 1369
 - BY statement (TRANPOSE) 1391
 - BY statement (UNIVARIATE) 1451
- CHART procedure 179
- CLASS statement (MEANS) 662
- CLASS statement (TABULATE) 1276
- DEFINE statement (REPORT) 989
- ID statement (COMPARE) 222
- PROC RANK statement 912
- DESCENDTYPES option
 - PROC MEANS statement 655
- DESCRIBE statement
 - SQL procedure 1137
- DESCRIPTION= argument
 - MODIFY statement (CATALOG) 152
- DESCRIPTION= option
 - HISTOGRAM statement (UNIVARIATE) 1462
 - PROBPLOT statement (UNIVARIATE) 1490
 - QQPLOT statement (UNIVARIATE) 1502
- descriptive statistics 31, 650, 683, 1257
 - based on moments 650
 - computing with class variables 685

- keywords and formulas 1580
 - tabular format 1260
- destination-independent input 42
- detail reports 939
- detail rows 939
- DETAILS option
 - CONTENTS statement (DATASETS) 345
 - PROC DATASETS statement 331
- deviation from the mean 1593
- DEVIATION option
 - TABLES statement (FREQ) 534
- dialog boxes 783
 - check boxes in 782
 - collecting user input 798
 - color for 793
 - definition of 779
 - input fields 792
 - radio buttons in 790
 - searching multiple values 801
 - text for 792
- DIALOG statement
 - PMENU procedure 783
- DICTIONARY tables 1199
 - performance and 1201
 - reporting from 1218
 - retrieving information about 1200
 - uses for 1201
- difference
 - definition of 227
 - report of differences 239
- DIG3SEP= option
 - PICTURE statement (FORMAT) 451
- digit selectors 453
- dimension 1265
- dimension expressions
 - constructing 1286
 - definition of 1265
 - elements in 1287
 - operators in 1288
 - style elements in 1288
- directives 453
- DIRECTORY option
 - CONTENTS statement (DATASETS) 345
- DISCONNECT statement
 - SQL procedure 1138
- DISCRETE option
 - CHART procedure 179
- DISPLAY option
 - DEFINE statement (REPORT) 989
- DISPLAY PAGE window, REPORT procedure 1011
- DISPLAY procedure 413
 - example 414
 - overview 413
 - PROC DISPLAY statement 414
 - syntax 413
- display variables 945, 989
- distribution 1587
- distribution of numeric variables 1436
- DJIA data set 1621
- DMOPTLOAD command 721
- DMOPTSAVE command 723
- DOCUMENT destination 40, 43
- DOL option
 - BREAK statement (REPORT) 975
 - RBREAK statement (REPORT) 997

- double-byte character sets
 - conversion tables for 407
- DOUBLE option
 - PROC PRINT statement 821
 - PROC SQL statement 1121
- double overlining 975, 997
- double underlining 975, 997
- DOWN= option
 - PROC FORMS statement 499
- DQUOTE= option
 - PROC SQL statement 1121
- DROP statement
 - SQL procedure 1138
- DTC= option
 - MODIFY statement (DATASETS) 367
- DUL option
 - RBREAK statement (REPORT) 975, 997
- DUR statement
 - CALENDAR procedure 89

E

- EBCDIC collating sequence 1095, 1101
- EBCDIC option
 - PROC SORT statement 1095
- EDF goodness-of-fit tests 1521
 - probability values of 1523
- EDUCATION data set 1622
- EET= option
 - PROC CIMPORT statement 201
 - PROC CPORT statement 310
- efficiency issues
 - statistical procedures 7
- elementary statistics procedures 1577
- embedded LIBNAME statements 1135
- embedded SQL 1207
- EMPDATA data set 1623
- ENDCOMP statement
 - REPORT procedure 994
- ENDPOINTS= option
 - HISTOGRAM statement (UNIVARI-
ATE) 1462
- ENERGY data set 1625
- ENTRYTYPE= option
 - CHANGE statement (CATALOG) 147
 - COPY statement (CATALOG) 149
 - DELETE statement (CATALOG) 150
 - EXCHANGE statement (CATALOG) 151
 - EXCLUDE statement (CATALOG) 151
 - EXCLUDE statement (CIMPORT) 203
 - EXCLUDE statement (CPORT) 314
 - MODIFY statement (CATALOG) 152
 - PROC CATALOG statement 145
 - SAVE statement (CATALOG) 153
 - SELECT statement (CATALOG) 153
 - SELECT statement (CIMPORT) 204
 - SELECT statement (CPORT) 315
- equal kappa coefficients 573
- EQUALS option
 - PROC SORT statement 1096
- equijoins 1166
- error checking
 - formats and 30
- ERROR option
 - PROC COMPARE statement 216
- error processing
 - of BY-group specifications 24
- ERRORSTOP option
 - PROC SQL statement 1121
- estimates 1587
- ET= option
 - PROC CIMPORT statement 201
 - PROC CPORT statement 310
- ETYPE= option
 - EXCLUDE statement (CIMPORT) 203
 - EXCLUDE statement (CPORT) 314
 - SELECT statement (CIMPORT) 204
 - SELECT statement (CPORT) 315
- event logging 341
- EXACT option
 - TABLES statement (FREQ) 534
- EXACT statement
 - FREQ procedure 522
- exact statistics 581
- exact tests 522, 552
- examples
 - raw data for 1615
- EXCEPT operator 1181
- EXCHANGE statement
 - CATALOG procedure 150
 - DATASETS procedure 357
- EXCLNPWDTS option
 - PROC MEANS statement 655
 - PROC TABULATE statement 1268
- EXCLNPWGT option
 - PROC CORR statement 270
 - PROC REPORT statement 963
 - PROC STANDARD statement 1246
 - PROC UNIVARIATE statement 1446
- EXCLUDE statement
 - CATALOG procedure 151
 - CIMPORT procedure 203
 - CPORT procedure 313
 - DATASETS procedure 358
 - FORMAT procedure 446
 - PRTEXP procedure 906
- exclusion lists 48
- EXCLUSIVE option
 - CLASS statement (MEANS) 662
 - CLASS statement (TABULATE) 1277
 - DEFINE statement (REPORT) 989
 - PROC MEANS statement 655
 - PROC TABULATE statement 1268
- EXEC option
 - PROC SQL statement 1121
- EXECUTE statement
 - SQL procedure 1139
- EXISTS condition 1163
- EXPECTED option
 - TABLES statement (FREQ) 534
- expected value 1587
- EXPLODE procedure 420
 - ending PROC EXPLODE step 423
- examples 423
 - message lines 421
 - Null statement 423
 - options 421
 - overview 419
 - PARMCARDS statement 420
 - PARMCARDS4 statement 420
 - PROC EXPLODE statement 420
 - syntax 420

exploded output 419
 EXPLORE window, REPORT procedure 1012
 Explorer window
 displaying list of styles 47
 exponential distribution 1531, 1537
 EXPONENTIAL option
 HISTOGRAM statement (UNIVARIATE) 1463
 PROBPLOT statement (UNIVARIATE) 1490
 QQPLOT statement (UNIVARIATE) 1502
 EXPORT 428
 EXPORT= option
 PROC REGISTRY statement 927
 EXPORT procedure 428
 data source statements 432
 DBMS specifications 429
 DBMS table statements 432
 delimited files 432, 434
 examples 434
 Excel spreadsheets 431, 437
 Microsoft Access tables 433, 438
 overview 427
 PC files 432
 PROC EXPORT statement 428
 spreadsheets 432, 438
 syntax 428
 exporting
 printer definitions 894
 registry contents 927, 933
 exporting data 427
 DBMS tables 432
 delimited files 432, 434
 Excel spreadsheets 431, 437
 Microsoft Access tables 433, 438
 exporting files 307
 EXTENDSN= option, PROC CIMPORT statement 201
 Extensible Markup Language (XML) 38
 external data sources
 importing data 633
 writing to 427
 external files
 comparing registry with 934
 exporting delimited files 434
 importing delimited files 641
 routing output or log to 883
 external files, browsing 627
 extreme value distribution 1532
 extreme values 650, 706, 709, 1473,

F

features of ODS 42
 FEEDBACK option
 PROC SQL statement 1121
 FILE= option
 CONTENTS statement (CATALOG) 147
 PROC CIMPORT statement 202
 PROC CPORT statement 311
 PROC FORMS statement 499
 FILEREF= argument, PROC FSLIST statement 628
 files
 aging 334
 changing attributes of 366
 converting 307
 copying 259, 347, 358, 374
 DATASETS procedure for 326
 deleting 353, 373
 exchanging names 357
 exporting 307
 listing 326
 moving among environments 307
 passwords 326
 renaming 342
 renaming groups of 334
 repairing 371
 FILL option
 HISTOGRAM statement (UNIVARIATE) 1463
 PROC CALENDAR statement 82
 PICTURE statement (FORMAT) 451
 FIN statement
 CALENDAR procedure 90
 FINNISH option
 PROC SORT statement 1095
 FISHER option
 TABLES statement (FREQ) 534
 Fisher's exact test 548
 fitted continuous distributions 1530
 beta distribution 1530
 exponential distribution 1531
 gamma distribution 1532
 kernel density estimates 1534
 lognormal distribution 1533
 normal distribution 1533
 Weibull distribution 1534
 flip charts 419
 floating point exception (FPE) recovery 1274
 FLOW option
 DEFINE statement (REPORT) 989
 PROC SQL statement 1122
 FMTLEN option
 CONTENTS statement (DATASETS) 346
 FMTLIB option
 PROC FORMAT statement 445, 446, 459
 FONT= option
 HISTOGRAM statement (UNIVARIATE) 1463
 INSET statement (UNIVARIATE) 1478
 PROBPLOT statement (UNIVARIATE) 1490
 QQPLOT statement (UNIVARIATE) 1502
 FORCE option
 APPEND statement (DATASETS) 336
 COPY statement (DATASETS) 349
 PROC CATALOG statement 146
 PROC CIMPORT statement 202
 PROC DATASETS statement 332
 PROC DBCSTAB statement 408
 PROC SORT statement 1096
 FORCEHIST option
 HISTOGRAM statement (UNIVARIATE) 1463
 FOREIGN option
 PROC PRTDEF statement 894
 form units 495
 layout of 503
 single form unit per observation 505
 FORMAT 443
 format catalogs 467
 format-name formats 462
 FORMAT= option
 DEFINE statement (REPORT) 989

INSET statement (UNIVARIATE) 1478
 MEAN statement (CALENDAR) 93
 PROC TABULATE statement 1268
 SUM statement (CALENDAR) 96
 TABLES statement (FREQ) 534
 FORMAT procedure 443
 associating informats/formats with variables 465
 concepts 465
 examples 474
 EXCLUDE statement 446
 excluding entries from processing 446
 formats for character values 459
 informats for reading and converting raw data 447
 input control data set 471
 INVALU statement 447
 options for informats/formats 462
 output 472
 output control data set 468
 overview 442
 permanent informats/formats 467
 picture formats 450
 PICTURE statement 450
 PROC FORMAT statement 444
 ranges 464
 results 468
 SELECT statement 459
 selecting entries for processing 459
 storing informats/formats 467
 syntax 443
 task tables 443, 444, 447, 450,
 template for printing numbers 450
 temporary informats/formats 467
 VALUE statement 459
 values, specifying 464
 FORMAT statement 53
 DATASETS procedure 358
 formats
 See also picture formats
 associating with variables 465
 BY-group processing and 30
 comparing unformatted values 228
 creating 442
 creating from data sets 482
 creating groups with 1075
 DATASETS procedure and 358
 date formats 478
 definition of 442
 error checking and 30
 for character values 459, 476
 for columns 1160
 for values in tables 1293
 format-name formats 462
 formatted values 25
 missing 468
 multilabel 1320
 multilabel value formats 693
 permanent 467, 488
 picture-name formats 458
 preloaded 992, 1278, 1315
 preloaded, with class variables 696
 printing descriptions of 487
 retrieving permanent formats 488
 storing 467
 temporary 467
 temporary associations 28, 29

- user-defined 442
- FORMATS window, REPORT procedure 1013
- formatted values 25
 - classifying data 27
 - grouping data 27
 - printing 25
- formatting data
 - instructions in output objects 40
 - table definitions 39
- FORMCHAR option
 - PROC CHART statement 171
 - PROC FREQ statement 519
 - PROC PLOT statement 729
 - PROC REPORT statement 963
 - PROC TABULATE statement 1269
 - PROC CALENDAR statement 82
- forms
 - printing reports with 956
- FORMS 497
- FORMS procedure 497
 - BY statement 501
 - concepts 503
 - continuous mode 504
 - examples 505
 - form layout 503
 - FREQ statement 502
 - LINE statement 502
 - modes of operation 504
 - multiple copies of a label 510
 - overview 495
 - page mode 504
 - PROC FORMS statement 497
 - procedure output file 504
 - single form unit per observation 505
 - syntax 497
 - task tables 497, 502
 - two sets of mailing labels 508
- formulas, for statistics 1578
- FORTCC option
 - FSLIST command 631
 - PROC FSLIST statement 629
- FPE recovery 1274
- FRACTION option
 - PROC RANK statement 912
- FRAME applications
 - associating menus with 813
- FREQ 518
- FREQ option
 - CHART procedure 179
 - PROC UNIVARIATE statement 1446
 - CHART procedure 179
- FREQ procedure 43, 518
 - adjusted odds ratio 577
 - ANOVA statistic 576
 - asymptotic tests 551
 - binomial proportions 560, 599
 - Bowker's test of symmetry 569
 - Breslow-Day test 580
 - BY statement 521
 - chi-square tests 546, 596, 605
 - Cochran-Armitage test for trend 566, 611
 - Cochran-Mantel-Haenszel statistics 574, 609
 - Cochran's Q test 574, 618
 - computational algorithms 581
 - computational resources 543, 583
 - concepts 541
 - confidence limits 551

- contingency coefficient 550
- correlation statistic 576
- Cramer's V 550
- crosstabulation tables 528
- definitions 544
- equal kappa coefficients 573
- EXACT statement 522
- exact statistics 581
- exact tests 552
- examples 592
- Fisher's exact test 548
- formatting characters 519
- frequency counts, inputting 541
- frequency tables 528
- Friedman's Chi-square statistic 615
- gamma 552
- general association statistic 577
- grouping with formats 542
- Jonckheere-Terpstra test 567
- Kendall's tau-b 552
- lambda asymmetric 558
- lambda symmetric 559
- Mantel-Haenszel adjusted odds ratio 578
- Mantel-Haenszel chi-square test 548
- McNemar's test 569
- measures of agreement 569
- measures of association 550
- missing values 585
- Monte Carlo estimation 584
- negative weights 541
- notation 544
- odds ratio 564, 579
- ODS table names 586
- output 589
- output data sets 525, 590, 593, 605
- OUTPUT statement 525, 591
- overall kappa coefficient 573
- overview 515
- p-values 582
- Pearson correlation coefficient 555
- phi coefficient 549
- polychoric correlation 558
- PROC FREQ statement 519
- relative risk estimates 577
- relative risks 565, 579
- results 585
- risks and risk differences 562
- scores 545
- simple kappa coefficient 570
- Somers' D 554
- Spearman correlation statistics 556
- statistical computations 544
- Stuart's tau-c 553
- suppressing displayed output 590
- syntax 518
- TABLES statement 524, 525, 528, 590
- task tables 518, 519, 529
- TEST statement 539
- uncertainty coefficient asymmetric 559
- uncertainty coefficient symmetric 560
- WEIGHT statement 540
- weighted kappa coefficient 571
- zero weights 541
- FREQ statement 57
 - CORR procedure 274
 - example 57
 - FORMS procedure 502

- MEANS procedure 665
- procedures supporting 57
- REPORT procedure 994
- STANDARD procedure 1249
- TABULATE procedure 1280
- UNIVARIATE procedure 1455
- frequency counts 1347
 - CHART procedure 184
- frequency of observations 57
- frequency tables 515, 528
- Friedman's Chi-square statistic 615
- FROM clause, SQL procedure 1147
- FRONTREF option
 - HISTOGRAM statement (UNIVARIATE) 1463
- FSEDIT applications
 - menu bars for 796
- FSEDIT sessions
 - associating menu bar with 798, 805
- FSEDIT window
 - associating menu bar with 800
- FSLIST command 628, 630
- FSLIST procedure 627
 - FSLIST command 628, 630
 - overview 627
 - PROC FSLIST statement 628
 - syntax 627
- FULLSTATUS option
 - PROC REGISTRY statement 928
- functions
 - SQL procedure and 1182, 1206
- FUZZ= option
 - FORMAT procedure 462
 - INVALUE statement (FORMAT) 448
 - PICTURE statement (FORMAT) 450
 - PROC COMPARE statement 216
 - TABLE statement (TABULATE) 1284
 - VALUE statement (FORMAT) 460
- FW= option
 - PROC MEANS statement 655

G

- G100 option
 - CHART procedure 180
- gallery of samples 33
- gamma 552
- gamma distribution 1532, 1537
- GAMMA option
 - HISTOGRAM statement (UNIVARIATE) 1464
 - PROBPLOT statement (UNIVARIATE) 1490
 - QQPLOT statement (UNIVARIATE) 1502
- Gaussian distribution 1594
- general association statistic 577
- generation groups
 - appending with 339
 - copying 352
 - DATASETS procedure and 354, 370
- GENERATION option
 - PROC CPORT statement 311
- GENMAX= option
 - MODIFY statement (DATASETS) 367
- GENNUM= data set option 339
- GENNUM= option
 - AUDIT statement (DATASETS) 341

CHANGE statement (DATASETS) 343
 DELETE statement (DATASETS) 353
 MODIFY statement (DATASETS) 368
 PROC DATASETS statement 332
 REPAIR statement (DATASETS) 371
 GETDELETED= statement
 IMPORT procedure 639
 GETNAMES= statement
 IMPORT procedure 639
 Ghostview printer definition 900
 Gini's mean difference 1527
 global statements 18
 goodness-of-fit tests 1448, 1520
 Anderson-Darling statistic 1522
 Cramer-von Mises statistic 1523
 EDF 1521, 1523
 Kolmogorov D statistic 1522
 Shapiro-Wilk statistic 1521
 GOUT= option
 PROC UNIVARIATE statement 1447
 graphics 1436
 annotating 1445
 high-resolution 1436, 1514
 insets 1474, 1479, 1480
 probability plots 1485, 1514, 1515
 quantile-quantile plots 1497, 1514, 1515
 saving 1447
 GRAY option
 ITEM statement (PMENU) 786
 GRID option
 HISTOGRAM statement (UNIVARIATE) 1464
 PROBPLOT statement (UNIVARIATE) 1491
 QQPLOT statement (UNIVARIATE) 1503
 GROC data set 1626
 GROUP BY clause, SQL procedure 1149
 GROUP option
 DEFINE statement (REPORT) 990
 CHART procedure 179
 PROC OPTIONS statement 716
 group variables 946, 990
 grouping data 27
 GROUPINTERNAL option
 CLASS statement (MEANS) 662
 CLASS statement (TABULATE) 1277
 GROUPS= option
 PROC RANK statement 912
 GSPACE= option
 CHART procedure 180

H

HAVING clause, SQL procedure 1150
 HAXIS= option, PLOT statement (PLOT) 736
 HBAR statement
 CHART procedure 175
 HEADER= option
 INSET statement (UNIVARIATE) 1478
 PROC CALENDAR statement 84
 header pages 419
 HEADING= option
 PROC PRINT statement 822
 HEADLINE option
 PROC REPORT statement 965
 HEADSKIP option
 PROC REPORT statement 966

HEIGHT= option
 HISTOGRAM statement (UNIVARIATE) 1464
 INSET statement (UNIVARIATE) 1478
 PROBPLOT statement (UNIVARIATE) 1491
 QQPLOT statement (UNIVARIATE) 1503
 HELP= option
 ITEM statement (PMENU) 786
 PROC REPORT statement 966
 HEXPAND option, PLOT statement (PLOT) 738
 high-resolution graphics 1436, 1514
 See graphics
 HILOC option
 PLOT statement (TIMEPLOT) 1374
 HISTOGRAM statement
 UNIVARIATE procedure 1455
 histograms 1455, 1541
 extreme values and 1546
 intervals 1541
 quantiles 1541
 two-way comparative 1568
 HMINOR= option
 HISTOGRAM statement (UNIVARIATE) 1464
 PROBPLOT statement (UNIVARIATE) 1491
 QQPLOT statement (UNIVARIATE) 1503
 Hoeffding option
 PROC CORR statement 270
 Hoeffding's measure of dependence 264, 281
 calculating and printing 270
 example 292
 output data set with 271
 HOFFSET= option
 HISTOGRAM statement (UNIVARIATE) 1464
 HOLIDATA= option
 PROC CALENDAR statement 84
 holidays data set 84, 100, 101, 103
 HOLIDUR statement
 CALENDAR procedure 90
 HOLIFIN statement
 CALENDAR procedure 91
 HOLISTART statement
 CALENDAR procedure 92
 HOLIVAR statement
 CALENDAR procedure 92
 HOMELOANS data set 1627
 horizontal bar charts 175, 1448
 for subset of data 192
 horizontal separators 1328
 HOST option
 PROC OPTIONS statement 716
 host options listing 716
 host-specific procedures 1613
 HPERCENT= option, PROC PLOT statement 730
 HPOS= option, PLOT statement (PLOT) 738
 HREF= option
 HISTOGRAM statement (UNIVARIATE) 1464
 PROBPLOT statement (UNIVARIATE) 1491
 QQPLOT statement (UNIVARIATE) 1503
 HREF= option, PLOT statement (PLOT) 738
 HREFCHAR= option, PLOT statement (PLOT) 738

HREFLABELS= option
 HISTOGRAM statement (UNIVARIATE) 1464
 PROBPLOT statement (UNIVARIATE) 1491
 QQPLOT statement (UNIVARIATE) 1503
 HREFLABPOS= option
 HISTOGRAM statement (UNIVARIATE) 1465
 PROBPLOT statement (UNIVARIATE) 1491
 QQPLOT statement (UNIVARIATE) 1503
 HREVERSE option, PLOT statement (PLOT) 738
 HSCROLL= option, PROC FSLIST statement 630
 HSPACE= option, PLOT statement (PLOT) 738
 HTML destination 44
 HTML files 1078, 1357
 HTML output 35
 HTML reports 838, 856, 871
 Hypertext Markup Language (HTML) 35, 44
 hypotheses, testing 1607
 keywords and formulas 1585
 HZERO option, PLOT statement (PLOT) 738

I

IC CREATE statement
 DATASETS procedure 359
 IC DELETE statement
 DATASETS procedure 361
 IC REACTIVATE statement
 DATASETS procedure 362
 ID option
 DEFINE statement (REPORT) 990
 ITEM statement (PMENU) 786
 ID statement
 COMPARE procedure 222
 MEANS procedure 665
 PRINT procedure 830
 TIMEPLOT procedure 1370
 TRANSPose procedure 1393
 UNIVARIATE procedure 1473
 ID variables 990
 IDLABEL statement
 TRANSPose procedure 1394
 IDMIN option
 PROC MEANS statement 655
 IMPORT 634
 IMPORT= option
 PROC REGISTRY statement 928
 IMPORT procedure 634
 data source statements 638
 delimited files 638, 641
 examples 641
 Excel spreadsheets 645, 646
 input data sources 636
 Microsoft Access tables 641, 647
 overview 633
 PC files 638
 PROC IMPORT statement 634
 spreadsheets 638, 645
 syntax 634
 IMPORT statement
 DBMS table statements 640
 importing
 to registry 928, 932

- transport files 199
- importing data 633
 - delimited files 638, 641
 - Excel spreadsheets 645
 - Microsoft Access tables 641, 647
 - PC files 638
 - spreadsheets 638, 645
- IN condition 1164
- in-line views 1148, 1205
 - querying 1230
- IN= option
 - COPY statement (CATALOG) 149
 - COPY statement (DATASETS) 349
- INDENT= option
 - LINE statement (FORMS) 503
 - PROC FORMS statement 499
 - TABLE statement (TABULATE) 1284
- INDEX CENTILES statement
 - DATASETS procedure 362
- INDEX CREATE statement
 - DATASETS procedure 363
- INDEX DELETE statement
 - DATASETS procedure 364
- INDEX= option
 - COPY statement (DATASETS) 349
 - PROC CPORT statement 311
- indexes
 - appending indexed data sets 338
 - centiles and 362
 - composite indexes 1129
 - creating 363
 - deleting 364, 1138
 - managing 1129
 - on altered columns 1127
 - on columns 1129, 1141
 - simple indexes 1129
 - SQL procedure and 1129
 - UNIQUE keyword 1129
- INFILE= option
 - PROC CIMPORT statement 202
- INFONT= option
 - HISTOGRAM statement (UNIVARI-ATE) 1465
 - PROBPLOT statement (UNIVARIATE) 1491
 - QQPLOT statement (UNIVARIATE) 1503
- INFORMAT statement
 - DATASETS procedure 365
- informats
 - associating with variables 465
 - creating 442
 - DATASETS procedure and 365
 - definition of 442
 - for columns 1160
 - for converting raw data 480
 - for raw data values 447
 - missing 468
 - permanent 467
 - printing descriptions of 487
 - storing 467
 - temporary 467
 - user-defined 442
- INHEIGHT= option
 - HISTOGRAM statement (UNIVARI-ATE) 1465
 - PROBPLOT statement (UNIVARIATE) 1491
 - QQPLOT statement (UNIVARIATE) 1503

- INITIATE argument
 - AUDIT statement (DATASETS) 341
- inner joins 1167
- INOBS= option
 - PROC SQL statement 1122
- input data sets 19
- input fields 792
- input files
 - procedure output as 889
- INSERT statement
 - SQL procedure 1140
- INSET statement
 - UNIVARIATE procedure 1474
- insets 1474
 - positioning in margins 1480
 - positioning with compass point 1479
 - positioning with coordinates 1480
- integrity constraints 326
 - appending data sets and 339
 - copying data sets and 349
 - creating 359
 - deleting 361
 - names for 360
 - PROC SQL tables 1125, 1127, 1133, 11
 - reactivating 362
 - SORT procedure 1102
- interactive line mode
 - printing from 957
- interquartile range 1527, 1593
- INTERSECT operator 1181
- INTERTILE= option
 - HISTOGRAM statement (UNIVARI-ATE) 1465
 - PROBPLOT statement (UNIVARIATE) 1491
 - QQPLOT statement (UNIVARIATE) 1504
- INTERVAL= option
 - PROC CALENDAR statement 85
- INTO clause, SQL procedure 1144
- INTYPE= option
 - PROC CPORT statement 311
- INVALUE statement
 - FORMAT procedure 447
- INVERSE statement, TRANTAB procedure 1415
- IS condition 1164
- ITEM statement
 - PMENU procedure 785
- ITEMHELP= option
 - DEFINE statement (REPORT) 990

J

- Japanese conversion tables 410
- joined-table component 1165
- JOINREF option
 - PLOT statement (TIMEPLOT) 1374
- joins
 - cross joins 1170
 - definition of 1166
 - equijoins 1166
 - inner joins 1167
 - joining a table with itself 1167
 - joining more than two tables 1172
 - joining tables 1166
 - joining three tables 1227
 - joining two tables 1213

- natural joins 1171
- outer joins 1169, 1205, 1220
- reflexive joins 1167
- rows to be returned 1166
- subqueries compared with 1174
- table limit 1166
- types of 1166
- union joins 1171
- Jonckheere-Terpstra test 567
- JT option
 - TABLES statement (FREQ) 535
- JUST option
 - INVALUE statement (FORMAT) 448

K

- K= option
 - HISTOGRAM statement (UNIVARI-ATE) 1465
- kappa statistics
 - equal kappa coefficients 573
 - overall kappa coefficient 573
 - simple kappa coefficient 570
 - weighted kappa coefficient 571
- KEEPLN option
 - OUTPUT statement (MEANS) 671
- Kendall correlation statistics 271
 - Kendall's partial tau-b 264, 274
 - Kendall's tau-b 264, 270, 280, 552
- KENDALL option
 - PROC CORR statement 270
- kernel density estimates 1534
- KERNEL option
 - HISTOGRAM statement (UNIVARI-ATE) 1466
- key cell 1452
- KEY= option
 - PROC OPTLOAD statement 722
 - PROC OPTSAVE statement 724
- KEYLABEL statement
 - TABULATE procedure 1281
- KEYLEVEL= option
 - CLASS statement (UNIVARIATE) 1452
- KEYWORD statement
 - TABULATE procedure 1281
- keywords, for statistics 1578
- KILL option
 - PROC CATALOG statement 146
 - PROC DATASETS statement 332
- Kolmogorov D statistic 1522
- Kolmogorov-Smirnov EDF test 1522
- kurtosis 1594
- KURTOSIS keyword 1580

L

- L= option
 - HISTOGRAM statement (UNIVARI-ATE) 1466
 - PROBPLOT statement (UNIVARIATE) 1491
 - QQPLOT statement (UNIVARIATE) 1504
- LABEL option
 - PROC PRINT statement 822
 - MODIFY statement (DATASETS) 368

- PROC PRINTTO statement 881
 - PROC TRANSPOSE statement 1390
 - LABEL statement 53
 - DATASETS procedure 366
 - labels
 - for columns 1160
 - multiple copies of, within a set of labels 510
 - producing with FORMS procedure 495
 - lambda asymmetric 558
 - lambda symmetric 559
 - LASTNAME option
 - LINE statement (FORMS) 503
 - LCLM keyword 1585
 - LEFT option
 - DEFINE statement (REPORT) 990
 - LEGEND option
 - PROC CALENDAR statement 85
 - lengths
 - for columns 1161
 - LET option
 - PROC TRANSPOSE statement 1390
 - LEVELS option
 - OUTPUT statement (MEANS) 671
 - CHART procedure 180
 - LGRID= option
 - HISTOGRAM statement (UNIVARIATE) 1466
 - PROBPLOT statement (UNIVARIATE) 1492
 - QQPLOT statement (UNIVARIATE) 1504
 - LHREF= option
 - HISTOGRAM statement (UNIVARIATE) 1466
 - PROBPLOT statement (UNIVARIATE) 1492
 - QQPLOT statement (UNIVARIATE) 1504
 - LIBNAME statement
 - DBMS connections with 1198
 - embedding in views 1135
 - libraries
 - printing all data sets 875
 - LIBRARY= option
 - PROC DATASETS statement 332
 - PROC FORMAT statement 445
 - librefs
 - stored views and 1134
 - LIKE condition 1174
 - likelihood-ratio chi-square test 547
 - line-drawing characters 962
 - line printer plots 1512
 - box plots 1513, 1514
 - normal probability plots 1513
 - stem-and-leaf plots 1512
 - LINE statement
 - FORMS procedure 502
 - REPORT procedure 995
 - LINES= option
 - PROC FORMS statement 499
 - LIST option
 - PLOT statement (PLOT) 738
 - PROC PRTDEF statement 894
 - PROC REGISTRY statement 928
 - PROC REPORT statement 966
 - TABLES statement (FREQ) 535
 - LIST statement, TRANTAB procedure 1416
 - LISTALL option
 - PROC COMPARE statement 216
 - LISTBASE option
 - PROC COMPARE statement 217
 - LISTBASEOBS option
 - PROC COMPARE statement 217
 - LISTBASEVAR option
 - PROC COMPARE statement 217
 - LISTCOMP option
 - PROC COMPARE statement 217
 - LISTCOMPOBS option
 - PROC COMPARE statement 217
 - LISTCOMPVAR option
 - PROC COMPARE statement 217
 - LISTEQUALVAR option
 - PROC COMPARE statement 217
 - LISTHELP= option
 - PROC REGISTRY statement 928
 - LISTING destination 40, 43
 - open as default 46
 - Listing output 33
 - LISTOBS option
 - PROC COMPARE statement 217
 - LISTREG= option
 - PROC REGISTRY statement 928
 - LISTUSER option
 - PROC REGISTRY statement 929
 - LISTVAR option
 - PROC COMPARE statement 217
 - listwise deletion 287
 - LOAD REPORT window, REPORT procedure 1013
 - LOAD statement, TRANTAB procedure 1416
 - LOCALE option
 - PROC CALENDAR statement 85
 - LOCATION= option
 - PROC UNIVARIATE statement 1447
 - location parameters 1539
 - LOCCOUNT option
 - PROC UNIVARIATE statement 1447
 - log
 - default destinations 879
 - destinations for 879
 - displaying SQL definitions 1137
 - listing registry contents in 928
 - routing to catalog entries 886
 - routing to external files 883
 - routing to printer 883, 892
 - system options list in 713, 717
 - writing printer attributes to 907
 - writing registry contents to 928
 - LOG option
 - AUDIT statement (DATASETS) 341
 - PROC PRINTTO statement 881
 - logit odds ratio, adjusted 579
 - lognormal distribution 1533, 1537
 - LOGNORMAL option
 - HISTOGRAM statement (UNIVARIATE) 1466
 - PROBPLOT statement (UNIVARIATE) 1492
 - QQPLOT statement (UNIVARIATE) 1504
 - LONG option
 - PROC OPTIONS statement 716
 - LOOPS= option
 - PROC SQL statement 1122
 - LOWER function (SQL) 1176
 - LOWER= option
 - HISTOGRAM statement (UNIVARIATE) 1466
 - LPI= option
 - PROC CHART statement 172
 - LS= option
 - PROC REPORT statement 967
 - LVREF= option
 - HISTOGRAM statement (UNIVARIATE) 1467
 - PROBPLOT statement (UNIVARIATE) 1492
 - QQPLOT statement (UNIVARIATE) 1504
- ## M
- macro return codes 228
 - macro variables
 - set by SQL procedure 1202
 - MAD (median absolute deviation) 1527
 - mailing labels 495
 - examples 505
 - multiple copies of a label, within a set 510
 - printing two sets 508
 - Mantel-Haenszel adjusted odds ratio 578
 - Mantel-Haenszel chi-square test 548
 - marginal homogeneity 618
 - markers 971, 1272
 - MARKUP destination 40, 44
 - markup languages 40, 44
 - MATCH_11 data set 1641
 - matching observations 210
 - matching patterns 1174, 1238
 - matching variables 210
 - MAX keyword 1581
 - MAX= option
 - FORMAT procedure 463
 - INVALUE statement (FORMAT) 448
 - PICTURE statement (FORMAT) 450
 - VALUE statement (FORMAT) 460
 - MAXDEC= option
 - PROC MEANS statement 655
 - PROC TIMEPLOT statement 1368
 - maximum value 1581
 - MAXITER= option
 - TABLES statement (FREQ) 535
 - MAXLABELN= option
 - PROC FORMAT statement 445
 - MAXNBIN= option
 - HISTOGRAM statement (UNIVARIATE) 1467
 - MAXPRINT= option
 - PROC COMPARE statement 217
 - MAXSELEN= option
 - PROC FORMAT statement 445
 - MAXSIGMAS= option
 - HISTOGRAM statement (UNIVARIATE) 1467
 - MAXTIME= option
 - EXACT statement (FREQ) 523
 - MC option
 - EXACT statement (FREQ) 523
 - McNemar's test 569
 - mean 1587, 1588
 - MEAN keyword 1581
 - MEAN option
 - CHART procedure 180
 - PROC STANDARD statement 1246
 - MEAN statement
 - CALENDAR procedure 93
 - MEANS 652

- MEANS procedure 652
 - analysis variables 673
 - BY statement 660, 664, 687
 - CLASS statement 661
 - class variables 661, 672, 674, 675
 - column width for output 681
 - computational resources 677
 - computer resources 665
 - concepts 675
 - confidence limits 679
 - examples 683
 - FREQ statement 665
 - ID statement 665
 - missing values 664, 681, 704
 - N Obs statistic 681
 - output data set 665, 666, 682
 - OUTPUT statement 666
 - output statistics 700, 702, 704, 706,
 - overview 650
 - PROC MEANS statement 653
 - quantiles 680
 - results 681
 - statistic keywords 658, 666
 - statistical computations 678
 - Student's t test 680
 - syntax 652
 - task tables 652, 653
 - TYPES statement 672
 - VAR statement 673
 - WAYS statement 674
 - WEIGHT statement 675
 - weighting observations 675
 - MEANTYPE= option
 - PROC CALENDAR statement 86
 - measures of agreement 539, 569
 - zero rows or columns 574
 - measures of association 292, 539, 550
 - nonparametric 263
 - measures of location 1588
 - measures of shape 1593
 - measures of variability 1592
 - MEASURES option
 - TABLES statement (FREQ) 535
 - median 1588
 - median absolute deviation (MAD) 1527
 - MEDIAN keyword 1583
 - MEMOSIZE= statement
 - IMPORT procedure 639, 640
 - MEMTYPE= option
 - AGE statement (DATASETS) 334
 - CHANGE statement (DATASETS) 343
 - CONTENTS statement (DATASETS) 346
 - COPY statement (DATASETS) 349, 350
 - DELETE statement (DATASETS) 353
 - EXCHANGE statement (DATASETS) 357
 - EXCLUDE statement (CIMPORT) 203
 - EXCLUDE statement (CPORT) 314
 - EXCLUDE statement (DATASETS) 358
 - MODIFY statement (DATASETS) 368
 - PROC CIMPORT statement 202
 - PROC CPORT statement 312
 - PROC DATASETS statement 333
 - REPAIR statement (DATASETS) 372
 - SAVE statement (DATASETS) 373
 - SELECT statement (CIMPORT) 204
 - SELECT statement (CPORT) 315
 - SELECT statement (DATASETS) 374
 - menu bars
 - associating with FSEDIT sessions 798, 805
 - associating with FSEDIT window 800
 - definition of 779
 - for FSEDIT applications 796
 - items in 785
 - items in, defining 787
 - key sequences for 786
 - MENU statement
 - PMENU procedure 788
 - menus 779
 - activating 779
 - associating FRAME applications with 813
 - associating windows with 810
 - data entry programs and 811
 - for DATA step window applications 807
 - printing from 812
 - submenus 792
 - merging data 1192
 - message characters 453
 - MESSAGE= option
 - IC CREATE statement (DATASETS) 361
 - MESSAGES window, REPORT procedure 1014
 - METHOD= option
 - PROC COMPARE statement 217
 - Microsoft Access tables
 - exporting 431, 433, 438
 - importing 641, 647
 - Microsoft Excel
 - exporting spreadsheets 431, 437, 438
 - importing spreadsheets 637, 645, 646
 - Microsoft Word 45
 - MIDPERCENTS option
 - HISTOGRAM statement (UNIVARIATE) 1467
 - MIDPOINTS= option
 - CHART procedure 180
 - HISTOGRAM statement (UNIVARIATE) 1467
 - MIN keyword 1581
 - MIN= option
 - FORMAT procedure 463
 - INVALUE statement (FORMAT) 448
 - PICTURE statement (FORMAT) 450
 - VALUE statement (FORMAT) 460
 - minimum value 1581
 - missing formats 468
 - missing informats 468
 - MISSING option
 - CHART procedure 181
 - CLASS statement (MEANS) 662
 - CLASS statement (TABULATE) 1277
 - CLASS statement (UNIVARIATE) 1453
 - DEFINE statement (REPORT) 990
 - PROC CALENDAR statement 86
 - PROC MEANS statement 656
 - PROC PLOT statement 731
 - PROC REPORT statement 967
 - PROC TABULATE statement 1270
 - TABLES statement (FREQ) 535
 - missing values
 - NMISS keyword 1581
 - PLOT procedure 749, 767
 - MISSPRINT option
 - TABLES statement (FREQ) 535
 - MISSTEXT= option
 - TABLE statement (TABULATE) 1284
 - MLF option
 - CLASS statement (MEANS) 663
 - CLASS statement (TABULATE) 1277
 - MNEMONIC= option
 - ITEM statement (PMENU) 787
 - mode 1588
 - mode calculation 1530
 - MODE keyword 1581
 - MODES option
 - PROC UNIVARIATE statement 1447
 - MODIFY statement
 - CATALOG procedure 152
 - DATASETS procedure 366
 - Monte Carlo estimation 522, 584
 - MOVE option
 - COPY statement (CATALOG) 149
 - COPY statement (DATASETS) 350
 - MT= option
 - PROC CPORT statement 312
 - MTYPE= option
 - AGE statement (DATASETS) 334
 - CHANGE statement (DATASETS) 343
 - CONTENTS statement (DATASETS) 346
 - COPY statement (DATASETS) 349, 350
 - DELETE statement (DATASETS) 353
 - EXCLUDE statement (CIMPORT) 203
 - EXCLUDE statement (CPORT) 314
 - EXCLUDE statement (DATASETS) 358
 - MODIFY statement (DATASETS) 368
 - PROC DATASETS statement 333
 - REPAIR statement (DATASETS) 372
 - SAVE statement (DATASETS) 373
 - SELECT statement (CIMPORT) 204
 - SELECT statement (CPORT) 315
 - SELECT statement (DATASETS) 374
 - MU= option
 - HISTOGRAM statement (UNIVARIATE) 1468
 - PROBPLOT statement (UNIVARIATE) 1492
 - QQPLOT statement (UNIVARIATE) 1504
 - MU0 option
 - PROC UNIVARIATE statement 1447
 - multi-threaded sorting 1100
 - multilabel formats 1320
 - MULTILABEL option
 - PICTURE statement (FORMAT) 451
 - VALUE statement (FORMAT) 460
 - multilabel value formats 693
 - multiple-choice survey data 1338
 - multiple-response survey data 1333
 - MULTIPLIER= option
 - PICTURE statement (FORMAT) 452
- N**
- N keyword 1581
 - N Obs statistic 681
 - N option
 - PROC PRINT statement 822
 - EXACT statement (FREQ) 524
 - NADJ= option
 - PROBPLOT statement (UNIVARIATE) 1492
 - QQPLOT statement (UNIVARIATE) 1505
 - NAME= argument
 - PROC DBCSTAB statement 407
 - TRANSTAB statement (CPORT) 315

- NAME= option
 - HISTOGRAM statement (UNIVARI-ATE) 1468
 - PROBPLOT statement (UNIVARIATE) 1492
 - PROC TRANSPOSE statement 1390
 - QQPLOT statement (UNIVARIATE) 1505
- NAMED option
 - PROC REPORT statement 967
- NATIONAL option
 - PROC SORT statement 1095
- natural joins 1171
- NCOLS= option
 - HISTOGRAM statement (UNIVARI-ATE) 1468
 - PROBPLOT statement (UNIVARIATE) 1492
 - QQPLOT statement (UNIVARIATE) 1505
- NDOWN= option
 - PROC FORMS statement 500
- NEDIT option
 - PROC CIMPORT statement 202
 - PROC CPORT statement 312
- negative weights 541
- nested variables 1265
- NEW option
 - COPY statement (CATALOG) 149
 - PROC CIMPORT statement 202
 - PROC PRINTTO statement 882
 - APPEND statement (DATASETS) 336
- NEXTROBS= option
 - PROC UNIVARIATE statement 1447
- NEXTRVAL= option
 - PROC UNIVARIATE statement 1447
- NLS option
 - LOAD statement (TRANTAB) 1416
 - PROC TRANTAB statement 1414
- NMISS keyword 1581
- NOALIAS option
 - PROC REPORT statement 967
- NOBARS option
 - HISTOGRAM statement (UNIVARI-ATE) 1468
- NOBORDER option, PROC FSLIST state-ment 630
- NOBS keyword 1582
- NOBYLINE system option
 - BY statement (MEANS) with 661
 - BY statement (PRINT) with 830
- NOBPLOT option
 - PROC UNIVARIATE statement 1448
- NOCAPS option, PROC FSLIST statement 629
- NOCC option
 - FSLIST command 631
 - PROC FSLIST statement 629
- NOCENTER option
 - PROC REPORT statement 962
- NOCLONE option
 - COPY statement (DATASETS) 347
- NOCOL option
 - TABLES statement (FREQ) 535
- NOCOMPLETECOLS option
 - PROC REPORT statement 963
- NOCOMPLETEROWS option
 - PROC REPORT statement 963
- NOCOMPRESS option
 - PROC CPORT statement 312
- NOCONTINUED option
 - TABLE statement (TABULATE) 1284
- NOCORR option
 - PROC CORR statement 270
- NOCUM option
 - TABLES statement (FREQ) 535
- NODATE option
 - PROC COMPARE statement 218
- NODETAILS option
 - CONTENTS statement (DATASETS) 345
 - PROC DATASETS statement 331
- NODOUBLE option
 - PROC SQL statement 1121
- NODS option
 - CONTENTS statement (DATASETS) 346
- NODUPKEY option
 - PROC SORT statement 1097
- NODUPRECS option
 - PROC SORT statement 1097
- NOEDIT option
 - COPY statement (CATALOG) 149
 - PICTURE statement (FORMAT) 452
 - PROC CIMPORT statement 202
 - PROC CPORT statement 312
- NOEQUALS option
 - PROC SORT statement 1096
- NOERRORSTOP option
 - PROC SQL statement 1121
- NOEXEC option
 - PROC REPORT statement 967
 - PROC SQL statement 1121
- NOFEEDBACK option
 - PROC SQL statement 1121
- NOFLOW option
 - PROC SQL statement 1122
- NOFRAME option
 - HISTOGRAM statement (UNIVARI-ATE) 1468
 - INSET statement (UNIVARIATE) 1478
 - PROBPLOT statement (UNIVARIATE) 1493
 - QQPLOT statement (UNIVARIATE) 1505
- NOFREQ option
 - TABLES statement (FREQ) 536
- NOHEADER option
 - CHART procedure 181
 - PROC REPORT statement 967
- NOHLABEL option
 - HISTOGRAM statement (UNIVARI-ATE) 1468
 - PROBPLOT statement (UNIVARIATE) 1493
 - QQPLOT statement (UNIVARIATE) 1505
- NOHOST option
 - PROC OPTIONS statement 716
- NOINHERIT option
 - OUTPUT statement (MEANS) 671
- NOLEGEND option
 - CHART procedure 181
 - PROC PLOT statement 731
- NOLIST option
 - PROC DATASETS statement 333
- NOMISS option
 - INDEX CREATE statement (DATASETS) 364
 - PROC CORR statement 270
 - PROC PLOT statement 731
- NOMISSBASE option
 - PROC COMPARE statement 218
- NOMISSCOMP option
 - PROC COMPARE statement 218
- NOMISSING option
 - PROC COMPARE statement 218
- NONE option
 - RBUTTON statement (PMENU) 790
- noninteractive mode
 - printing from 957
- NONOBS option
 - PROC MEANS statement 656
- nonparametric measures of association 263
- NONUM option, PROC FSLIST statement 630
- NONUMBER option
 - PROC SQL statement 1123
- NOOBS option
 - PROC PRINT statement 823
- NOOVP option
 - FSLIST command 632
 - PROC FSLIST statement 630
- NOPERCENT option
 - TABLES statement (FREQ) 536
- NOPLOT option
 - HISTOGRAM statement (UNIVARI-ATE) 1468
- NOPRINT option
 - CONTENTS statement (DATASETS) 346
 - DEFINE statement (REPORT) 990
 - HISTOGRAM statement (UNIVARI-ATE) 1469
 - PROC COMPARE statement 218
 - PROC CORR statement 270
 - PROC FREQ statement 520
 - PROC MEANS statement 657
 - PROC SQL statement 1123
 - PROC SUMMARY statement 1258
 - PROC UNIVARIATE statement 1448
 - TABLES statement (FREQ) 536
- NOPROB option
 - PROC CORR statement 270
- NOPROMPT option
 - PROC SQL statement 1123
- NOREPLACE option
 - PROC FORMAT statement 445
- normal distribution 1533, 1538, 1587, 15
- NORMAL option
 - HISTOGRAM statement (UNIVARI-ATE) 1469
 - PROBPLOT statement (UNIVARIATE) 1493
 - PROC UNIVARIATE statement 1448
 - QQPLOT statement (UNIVARIATE) 1505
 - PROC RANK statement 912
- normal probability plots 1448, 1513
- reference lines on 1566
- NOROW option
 - TABLES statement (FREQ) 536
- NORWEGIAN option
 - PROC SORT statement 1095
- NOSEPS option
 - PROC TABULATE procedure 1271
- NOSIMPLE option
 - PROC CORR statement 270
- NOSORTMSG option
 - PROC SQL statement 1123
- NOSOURCE option
 - COPY statement (CATALOG) 149
- NOSRC option
 - PROC CIMPORT statement 202
 - PROC CPORT statement 312

- NOSTATS option
 - CHART procedure 181
 - NOSTIMER option
 - PROC SQL statement 1123
 - NO SUMMARY option
 - PROC COMPARE statement 218
 - NOSYMBOL option
 - CHART procedure 181
 - NOSYMNAM option
 - PLOT statement (TIMEPLOT) 1374
 - NOTE option
 - PROC COMPARE statement 218
 - NOTHEADS option
 - PROC MEANS statement 659
 - PROC SQL statement 1123
 - NOTRAP option
 - PROC MEANS statement 656
 - PROC TABULATE statement 1274
 - NOTSORTED option
 - BY statement 54
 - BY statement (CALENDAR) 87
 - BY statement (CHART) 174
 - BY statement (COMPARE) 221
 - BY statement (CORR) 273
 - BY statement (FORMS) 501
 - BY statement (FREQ) 521
 - BY statement (MEANS) 661
 - BY statement (PLOT) 732
 - BY statement (PRINT) 830
 - BY statement (RANK) 914
 - BY statement (REPORT) 978
 - BY statement (STANDARD) 1248
 - BY statement (TABULATE) 1276
 - BY statement (TIMEPLOT) 1369
 - BY statement (TRANSPPOSE) 1391
 - BY statement (UNIVARIATE) 1451
 - FORMAT procedure 463
 - ID statement (COMPARE) 222
 - INVALID statement (FORMAT) 448
 - PICTURE statement (FORMAT) 450
 - VALUE statement (FORMAT) 460
 - NOVALUES option
 - PROC COMPARE statement 218
 - NOVLABEL option
 - HISTOGRAM statement (UNIVARIATE) 1469
 - PROBPLOT statement (UNIVARIATE) 1493
 - QQPLOT statement (UNIVARIATE) 1505
 - NOVTICK option
 - HISTOGRAM statement (UNIVARIATE) 1469
 - PROBPLOT statement (UNIVARIATE) 1493
 - QQPLOT statement (UNIVARIATE) 1505
 - NOWARN option
 - PROC DATASETS statement 333
 - TABLES statement (FREQ) 536
 - NOWINDOWS option
 - PROC REPORT statement 973
 - NOZERO option
 - DEFINE statement (REPORT) 991
 - NOZEROS option
 - CHART procedure 181
 - NPLUS1 option
 - PROC RANK statement 913
 - NPP option
 - PLOT statement (TIMEPLOT) 1374
 - NROWS= option
 - HISTOGRAM statement (UNIVARIATE) 1469
 - PROBPLOT statement (UNIVARIATE) 1493
 - QQPLOT statement (UNIVARIATE) 1505
 - NSRC option
 - PROC CIMPORT statement 202
 - PROC CPORT statement 312
 - null hypothesis 1607
 - Null statement, EXPLODE procedure 423
 - NUM option, PROC FSLIST statement 630
 - NUMBER option
 - PROC SQL statement 1123
 - numeric values
 - converting raw character data to 480
 - numeric variables
 - distribution of 1436
 - ranks for 909
 - sorting orders for 1100
 - NWAY option
 - PROC MEANS statement 656
- ## O
- OBS= option
 - PROC PRINT statement 823
 - observations
 - statistics for groups of 7
 - observations 1115
 - See also* rows
 - appending to data sets 71
 - comparing 248, 251
 - frequency of 57
 - matching 210
 - printing 817
 - sorting 1091
 - total number of 1582
 - transposing variables into 1387
 - odds ratio 564
 - odds ratio, adjusted 577, 578
 - ODS destinations
 - definition of 40
 - descriptions of 42
 - destination-independent input 42
 - determining, for output objects 50
 - diagram of 40
 - opening and closing 46
 - SAS formatted destinations 43
 - system resources and 46
 - third-party formatted destinations 44
 - ODS output
 - definition of 40
 - figure of 40
 - ODS (Output Delivery System)
 - CORR procedure and 287
 - DATASETS procedure and 382
 - description 32
 - features 42
 - FREQ procedure and 586
 - how ODS works 40
 - printing reports 956
 - REPORT procedure and 1078, 1083
 - samples of formatted output 33
 - style elements for output 1078, 1083
 - table names 1541
 - TABULATE procedure and 1357
 - terminology 39
 - ODS processing 40
 - ODS statements
 - formatting options 45
 - OL option
 - BREAK statement (REPORT) 975
 - RBREAK statement (REPORT) 998
 - ON option
 - CHECKBOX statement (PMENU) 782
 - ONE option
 - CLEAR statement (TRANTAB) 1415
 - LIST statement (TRANTAB) 1416
 - SAVE statement (TRANTAB) 1418
 - one-tailed tests 1608
 - operands
 - values from 1182
 - operating environment-specific procedures 30, 1613
 - operators
 - arithmetic 1205
 - in dimension expressions 1288
 - order of evaluation 1183
 - set operators 1177, 1206
 - truncated string comparison operators 1184
 - values from 1182
 - OPT= option
 - TRANTAB statement (CPORT) 316
 - OPTION= option
 - PROC OPTIONS statement 717
 - OPTIONS 716
 - OPTIONS procedure 716
 - displaying options in groups 716
 - examples 717
 - format for display of settings 716
 - host options listing 716
 - log display 717
 - overview 713
 - portable options listing 716
 - PROC OPTIONS statement 716
 - results 717
 - setting of a single option 718
 - short descriptions of system options 717
 - short form of options listing 717
 - syntax 716
 - task table 716
 - OPTLOAD 721
 - OPTLOAD procedure 721
 - overview 721
 - PROC OPTLOAD statement 722
 - task table 722
 - OPTSAVE 723
 - OPTSAVE procedure 723
 - PROC OPTSAVE statement 724
 - task table 724
 - ORDER BY clause, SQL procedure 1151, 1205
 - ORDER option
 - DEFINE statement (REPORT) 991
 - CLASS statement (MEANS) 663
 - CLASS statement (TABULATE) 1277
 - CLASS statement (UNIVARIATE) 1453
 - DEFINE statement (REPORT) 991
 - PROC FREQ statement 520
 - PROC MEANS statement 656
 - PROC TABULATE statement 1271, 1308
 - order variables 945, 991
 - orthogonal expressions 1205

- OUT= argument
 - APPEND statement (DATASETS) 335
 - COPY statement (CATALOG) 148
 - COPY statement (DATASETS) 347
 - PROC IMPORT statement 635
 - OUT= option
 - CONTENTS statement (CATALOG) 147
 - CONTENTS statement (DATASETS) 346
 - OUTPUT statement (FREQ) 525
 - OUTPUT statement (MEANS) 666
 - OUTPUT statement (UNIVARIATE) 1482
 - PROC COMPARE statement 219, 236
 - PROC PRTEXP statement 906
 - PROC RANK statement 913
 - PROC REPORT statement 968
 - PROC SORT statement 1098
 - PROC STANDARD statement 1247
 - PROC TABULATE statement 1272
 - PROC TRANSPOSE statement 1391
 - TABLES statement (FREQ) 536
 - OUT2= option
 - CONTENTS statement (DATASETS) 346
 - OUTALL option
 - PROC COMPARE statement 219
 - OUTBASE option
 - PROC COMPARE statement 219
 - OUTCOMP option
 - PROC COMPARE statement 219
 - OUTCUM option
 - TABLES statement (FREQ) 536
 - OUTDIF option
 - PROC COMPARE statement 219
 - OUTDUR statement
 - CALENDAR procedure 94
 - outer joins 1169, 1205, 1220
 - OUTER UNION set operator 1178
 - OUTEXPECT option
 - TABLES statement (FREQ) 536
 - OUTFILE= argument
 - PROC EXPORT statement 429
 - OUTFIN statement
 - CALENDAR procedure 94
 - OUTH= option
 - PROC CORR statement 271
 - OUTHISTOGRAM= option
 - HISTOGRAM statement (UNIVARIATE) 1469, 1543
 - OUTK= option
 - PROC CORR statement 271
 - OUTLIB= option
 - PROC CPORT statement 312
 - OUTNOEQUAL option
 - PROC COMPARE statement 219
 - OUTOBS= option
 - PROC SQL statement 1123
 - OUTP= option
 - PROC CORR statement 271
 - OUTPCT option
 - TABLES statement (FREQ) 536
 - OUTPERCENT option
 - PROC COMPARE statement 219
 - output
 - components of 40
 - customizing 48
 - oversized text for printed output 419
 - output data sets 44
 - comparing observations 251
 - storing partial correlations in 302
 - summary statistics in 253
 - OUTPUT destination 40, 44
 - output objects
 - customizing output for 50
 - definition of 40
 - determining destinations for 50
 - excluding 48
 - selecting 48
 - OUTPUT= option
 - CALENDAR statement (CALENDAR) 88
 - OUTPUT statement
 - FREQ procedure 525, 591
 - MEANS procedure 666
 - UNIVARIATE procedure 1482, 1510
 - Output window
 - printing from 957
 - OUTREPT= option
 - PROC REPORT statement 968
 - OUTS= option
 - PROC CORR statement 271
 - OUTSTART statement
 - CALENDAR procedure 95
 - OUTSTATS= option
 - PROC COMPARE statement 220, 237
 - OUTTABLE= argument
 - PROC EXPORT statement 429
 - OUTTYPE= option
 - PROC CPORT statement 313
 - OUTWARD= option, PLOT statement (PLOT) 738
 - overall kappa coefficient 573
 - OVERLAY option
 - PLOT statement (PLOT) 739
 - PLOT statement (TIMEPLOT) 1374
 - overlining 975, 997, 998
 - OVP option
 - FSLIST command 632
 - PROC FSLIST statement 630
 - OVPCHAR= option
 - PLOT statement (TIMEPLOT) 1374
- ## P
- P keywords 1583
 - p-values 582, 1610
 - PACK option
 - LINE statement (FORMS) 503
 - page description languages 45
 - page dimension 1265, 1294
 - page dimension text 1265
 - page layout 834
 - customizing 869
 - with many variables 863
 - page numbering 883
 - PAGE option
 - BREAK statement (REPORT) 976
 - DEFINE statement (REPORT) 992
 - PROC FORMAT statement 445
 - PROC FREQ statement 520
 - RBREAK statement (REPORT) 998
 - PAGEBY statement
 - PRINT procedure 831
 - PAGESIZE= option
 - PROC FORMS statement 500
 - paired data 1518, 1552
 - pairwise deletion 287
 - PANELS= option
 - PROC REPORT statement 969, 1058
 - parameters 1587
 - PARMCARDS statement, EXPLODE procedure 420
 - PARMCARDS4 statement, EXPLODE procedure 420
 - partial correlations 282
 - example 302
 - PARTIAL statement
 - CORR procedure 274
 - partitioned data sets
 - multi-threaded sorting 1100
 - password-protected data sets
 - transporting 316
 - passwords 369
 - appending password-protected data sets 337
 - assigning 369
 - changing 370
 - copying password-protected files 351
 - removing 370
 - pattern matching 1174, 1238
 - PC files
 - exporting 432
 - importing 638
 - PCTLAXIS option
 - QQPLOT statement (UNIVARIATE) 1506
 - PCTLDEF= option
 - PROC MEANS statement 658
 - PROC TABULATE statement 1273
 - PROC UNIVARIATE statement 1448
 - PCTLMINOR option
 - PROBPLOT statement (UNIVARIATE) 1493
 - QQPLOT statement (UNIVARIATE) 1506
 - PCTLORDER= option
 - PROBPLOT statement (UNIVARIATE) 1493
 - PCTLSCALE option
 - QQPLOT statement (UNIVARIATE) 1506
 - PDF files 1078, 1357
 - PDF output 37
 - PDF reports 842
 - peakedness 1594
 - Pearson correlation statistics 263, 555
 - example 292
 - in output data set 271
 - Pearson partial correlation 264, 274
 - Pearson product-moment correlation 263, 271, 279, 292
 - Pearson weighted product-moment correlation 263, 275
 - suppressing 270
 - PEARSON option
 - PROC CORR statement 271
 - Pearson Type I or II distributions 1531
 - PENALTIES= option, PLOT statement (PLOT) 739
 - percent coefficient of variation 1580
 - percent difference
 - definition of 227
 - PERCENT option
 - CHART procedure 181
 - PROC RANK statement 913
 - percentage bar charts 186
 - percentages
 - denominator definitions and 1347
 - in reports 1064

- TABULATE procedure 1294, 1344, 1347
- percentiles 1588
 - calculating 1528
 - from quantile-quantile plots 1509
 - keywords and formulas 1583
 - probability plots and 1485
 - saving 1484, 1555
 - TABULATE output 1483
 - theoretical 1509
- PERCENTS= option
 - HISTOGRAM statement (UNIVARI-ATE) 1469
- permanent data sets 16
- permanent informats and formats 467
 - accessing 467
 - retrieving 488
- PFILL= option
 - HISTOGRAM statement (UNIVARI-ATE) 1470
- phi coefficient 549
- picture formats 450
 - creating 474
 - digit selectors 453
 - directives 453
 - filling 492
 - message characters 453
 - PRINT procedure and 824
 - steps for building 454
- picture-name formats 458
- PICTURE statement
 - FORMAT procedure 450
- pie charts 168, 175
- PIE statement
 - CHART procedure 175
- PLACEMENT= option, PLOT statement (PLOT) 739
- PLCORR option
 - TABLES statement (FREQ) 537
- PLOT procedure 729
 - BY statement 732
 - collision states 747
 - computational resources 748
 - concepts 744
 - contour plots 735, 759
 - examples 750
 - generating data with program statements 744
 - hidden label characters 748
 - hidden observations 749
 - labeling plot points with variable values 745
 - labeling points 727
 - missing values 749, 767
 - overlying plots 726, 748, 753
 - overview 726
 - penalties 746, 775
 - PLOT statement 733
 - pointer symbols 745
 - printed output 749
 - PROC PLOT statement 729
 - reference lines 748, 751
 - results 749
 - RUN groups 744
 - scale of axes 749
 - syntax 729
 - task tables 729, 733
 - variable combinations in plot requests 735
 - variable lists in plot requests 734
- PLOT statement
 - PLOT procedure 733
 - TIMEPLOT procedure 1371
- plots
 - box plots 1448, 1513, 1514
 - customizing axes 1378
 - horizontal bar charts 1448
 - line printer plots 1512
 - multiple observations, on one plot line 1384
 - normal probability plots 1448, 1513, 1566
 - page layout 1375
 - plotting a single variable 1376
 - probability plots 1485, 1514, 1515, 15
 - quantile-quantile plots 1497, 1514, 1536, 15
 - size of 1448
 - specifying 1371
 - stem-and-leaf 1448, 1512
 - superimposing 1382
- PLOTS option
 - PROC UNIVARIATE statement 1448
- PLOTSIZE= option
 - PROC UNIVARIATE statement 1448
- plotting symbols 1378, 1380
- plotting variables, over time intervals 1365
- PMENU 781
- PMENU catalog entries
 - naming 788
 - steps for building and using 794
 - storing 781
- PMENU command 779
- PMENU procedure
 - CHECKBOX statement 782
 - concepts 793
 - defining pull-down menus 788
 - DIALOG statement 783
 - ending 794
 - examples 796
 - execution of 793
 - ITEM statement 785
 - MENU statement 788
 - overview 779
 - PMENU catalog entries 794
 - PROC PMENU statement 781
 - RADIOBOX statement 789
 - RBUTTON statement 790
 - SELECTION statement 791
 - SEPARATOR statement 791
 - SUBMENU statement 792
 - syntax 781
 - task tables 781, 785
 - templates for 794
 - TEXT statement 792
- POINT option
 - EXACT statement (FREQ) 524
- polychoric correlation 558
- populations 1586
- Portable Document Format (PDF) 37
- PORTABLE option
 - PROC OPTIONS statement 716
- portable options listing 716
- POS= option
 - PLOT statement (TIMEPLOT) 1374
- POSITION= option
 - INSET statement (UNIVARIATE) 1478
- posters 419
- PostScript files 860
- PostScript output 35, 900
- power, of statistical tests 1609
- power-function distribution 1531
- PREFIX= option
 - PICTURE statement (FORMAT) 452
 - PROC TRANSPOSE statement 1391
- preloaded formats 992, 1278, 1315
 - class variables with 696
- PRELOADFMT option
 - CLASS statement (MEANS) 664
 - CLASS statement (TABULATE) 1278
 - DEFINE statement (REPORT) 992
- primary label 451
- PRINT 820
- PRINT option
 - PROC MEANS statement 657
 - PROC SQL statement 1123
 - PROC STANDARD statement 1247
 - PROC SUMMARY statement 1258
 - PROC PRINTTO statement 882
- PRINT procedure 43, 48, 820
 - BY statement 829, 832
 - column headings 836, 840
 - column width 836
 - examples 837
 - grouping observations 844
 - HTML reports 838, 856, 871
 - ID statement 830
 - listing report 837
 - order of variables 833
 - output 834
 - overview 817
 - page ejects 831
 - page layout 834, 863, 869
 - PAGEBY statement 831
 - PDF reports 842
 - PostScript files 860
 - printing all data sets in a library 875
 - PROC PRINT statement 820
 - results 834
 - RTF reports 846, 866
 - selecting variables 833, 837
 - style elements 825, 831, 832, 834
 - SUM statement 832
 - SUMBY statement 833
 - summing numeric values 832, 849, 853
 - sums, limiting number of 833, 858
 - syntax 820
 - task tables 820
 - VAR statement 833
 - XML files 851
- PRINTALL option
 - PROC COMPARE statement 220
- PRINTALLTYPES option
 - PROC MEANS statement 657
- printer attributes 905
 - writing to data sets 907
 - writing to log 907
- printer definitions 893, 906
 - available to all users 901
 - creating 906
 - deleting 894
 - exporting 894
 - for Ghostview printer 900
 - in SASHELP library 894
 - modifying 906
 - multiple 900
 - replicating 906

- PRINTER destination 40, 45
- Printer Family destination 40, 45
- printer forms 495
- printers
 - list of 894
 - routing log or output to 883, 892
- PRINTIDVARS option
 - PROC MEANS statement 657
- printing 817
 - all data sets in a library 875
 - data set contents 257
 - formatted values 25
 - from menus 812
 - grouping observations 844
 - informat/format descriptions 487
 - page ejects 831
 - page layout 834, 863, 869
 - reports 956
 - reports, from batch mode 957
 - reports, from interactive line mode 957
 - reports, from noninteractive mode 957
 - reports, from Output window 957
 - reports, from REPORT window 956
 - reports, with forms 956
 - reports, with ODS 956
 - reports, with PRINTTO procedure 957
 - selecting variables for 837
 - variable values 1370
- PRINTKWT option
 - TABLES statement (FREQ) 537
- PRINTMISS option
 - TABLE statement (TABULATE) 1284
- PRINTTO 880
- PRINTTO procedure 880
 - concepts 883
 - examples 883
 - overview 879
 - page numbering 883
 - printing reports 957
 - PROC PRINTTO statement 880
 - syntax 880
 - task table 880
- probability function 1587
- probability plots 1485, 1514
 - interpreting 1515
 - location and scale parameters 1539
 - shape parameters 1539
 - theoretical distributions for 1536
- probability values 286, 1610
- PROBPLOT statement
 - UNIVARIATE procedure 1485
- PROBT keyword 1585
- PROC CALENDAR statement 81
- PROC CATALOG 144
- PROC CATALOG statement 145
- PROC CHART 171
- PROC CHART statement 171
- PROC CIMPORT statement 200
- PROC COMPARE 213
- PROC COMPARE statement 214
- PROC CONTENTS statement 258
- PROC COPY 260
- PROC CORR 267
- PROC CORR statement 268
- PROC CPORT 308
- PROC CPORT statement 308
- PROC DATASETS 330
- PROC DATASETS statement 330, 378
- PROC DBCSTAB 407
- PROC DBCSTAB statement 407
- PROC DISPLAY statement 414
- PROC EXPLODE statement 420
- PROC EXPORT 428
- PROC EXPORT statement 428
- PROC FORMAT 443
- PROC FORMAT statement 444
- PROC FORMS 497
- PROC FORMS statement 497
- PROC FREQ 518
- PROC FREQ statement 519
- PROC FSLIST statement 628
- PROC IMPORT 634
- PROC IMPORT statement 634
- PROC MEANS 652
- PROC MEANS statement 653
- PROC OPTIONS 716
- PROC OPTIONS statement 716
- PROC OPTLOAD 721
- PROC OPTLOAD statement 722
- PROC OPTSAVE 723
- PROC OPTSAVE statement 724
- PROC PLOT statement 729
- PROC PMENU 781
- PROC PMENU statement 781
- PROC PRINT 820
- PROC PRINT statement 820
- PROC PRINTTO 880
- PROC PRINTTO statement 880
- PROC PRTDEF 893
- PROC PRTDEF statement 894
- PROC PRTEXP 905
- PROC PRTEXP statement 906
- PROC RANK 911
- PROC RANK statement 911
- PROC REGISTRY 926
- PROC REGISTRY statement 926
- PROC REPORT 959
- PROC REPORT statement 959
- PROC SORT 1093
- PROC SORT statement 1093
- PROC SQL 1117
- PROC SQL statement 1120
- PROC SQL tables 1115
 - aliases 1148, 1166
 - column attributes 1125, 1127
 - columns, adding 1125
 - columns, dropping 1125
 - columns, initial values of 1127
 - columns, renaming 1127
 - columns, selecting 1142
 - combining 1216
 - creating 1130, 1207
 - creating, from query expressions 1133
 - creating, from query results 1209
 - deleting 1138
 - inserting data 1207
 - inserting values 1141
 - integrity constraints 1125, 1127, 1133, 11
 - joining 1165, 1213, 1233
 - joining a table with itself 1165, 1167
 - joining more than two tables 1172
 - joining three tables 1227
 - recursive table references 1133
 - retrieving data from 1176
 - rows, adding 1140
 - rows, counting 1191
 - rows, deleting 1136
 - rows, deleting through views 1136
 - rows, ordering 1151
 - rows, selecting 1142
 - rows, without 1132
 - source tables 1147
 - table definitions 1137
 - table expressions 1177, 1196
 - updating 1211
 - updating column values 1153
 - updating through views 1153
 - without rows 1132
- PROC SQL views 1115
 - columns, selecting 1142
 - creating, from query expressions 1134
 - creating, from query results 1224
 - deleting 1138
 - embedding LIBNAME statements in 1135
 - librefs and stored views 1134
 - rows, adding 1140
 - rows, deleting 1136
 - rows, deleting through views 1136
 - rows, inserting through views 1141
 - rows, selecting 1142
 - sorting data retrieved by 1134
 - source views 1147
 - storing DBMS connection information 1135
 - updating 1135, 1203
 - updating column values 1153
 - updating tables through 1153
 - view definitions 1137, 1205
- PROC STANDARD 1245
- PROC STANDARD statement 1246
- PROC SUMMARY statement 1258
- PROC TABULATE 1266
- PROC TABULATE statement 1267
- PROC TIMEPLOT 1368
- PROC TIMEPLOT statement 1368
- PROC TRANSPOSE 1390
- PROC TRANSPOSE statement 1390
- PROC TRANTAB statement 1414
- PROC UNIVARIATE 1443
- PROC UNIVARIATE statement 1444
- procedure output
 - as input file 889
 - default destinations 879
 - destinations for 879
 - page numbering 883
 - routing to catalog entries 886
 - routing to external files 883
 - routing to printer 883, 892
- procedures
 - choosing 3
 - concepts 19
 - descriptions of 10
 - functional categories of 3
 - report-writing procedures 3, 4
 - statistical procedures 3, 6
 - utility procedures 4, 8
- processing diagram 40
- PROCLIB.DELAY data set 1642
- PROCLIB.EMP95 data set 1643
- PROCLIB.INTERNAT data set 1645
- PROCLIB.LAKES data set 1646
- PROCLIB.MARCH data set 1646

PROCLIB.PAYLIST2 data set 1647
 PROCLIB.PAYROLL data set 1648
 PROCLIB.PAYROLL2 data set 1651
 PROCLIB.SCHEDULE data set 1651
 PROCLIB.STAFF data set 1654
 PROCLIB.SUPERV data set 1657
 PROFILE= option
 PROC REPORT statement 969
 PROFILE window, REPORT procedure 1014
 PROMPT option
 PROC REPORT statement 970
 PROC SQL statement 1123
 PROMPTER window, REPORT procedure 1015
 PRTDEF 893
 PRTDEF procedure 893
 examples 899
 Ghostview printer definition 900
 input data set 895
 multiple printer definitions 900
 optional variables 897
 overview 893
 printer definition for all users 901
 PROC PRTDEF statement 894
 required variables 896
 task table 894
 valid variables 895
 PRTEXP 905
 PRTEXP procedure 905
 concepts 906
 examples 907
 EXCLUDE statement 906
 overview 905
 PROC PRTEXP statement 906
 SELECT statement 906
 syntax 905
 PS= option
 PROC FORMS statement 500
 PROC REPORT statement 970
 PSPACE= option
 PROC REPORT statement 970
 pull-down menus 779
 defining 788
 items in 785
 key sequences for 786
 separator lines for 791
 push buttons 785
 PUT statement
 compared with LINE statement (REPORT) 996
 PW= option
 MODIFY statement (DATASETS) 368
 PROC DATASETS statement 333
 PWD= statement
 EXPORT procedure 433
 IMPORT procedure 640

Q

Q keywords 1583
 Q-Q plots
 See quantile-quantile plots
 QMARKERS= option
 PROC MEANS statement 657
 PROC REPORT statement 971
 PROC TABULATE statement 1272

QMETHOD= option
 PROC MEANS statement 658
 PROC REPORT statement 971
 PROC TABULATE statement 1272
 QNTLDEF= option
 PROC MEANS statement 658
 PROC REPORT statement 971
 PROC TABULATE statement 1273
 QQPLOT statement
 UNIVARIATE procedure 1497
 QRANGE keyword 1583
 quantile-quantile plots 1497, 1514
 interpreting 1515
 location and scale parameters 1539
 percentiles of 1509
 reference lines 1566
 shape parameters 1539
 theoretical distributions for 1536
 quantiles 971, 1272, 1273
 confidence limits for 1528
 efficiency issues 7
 histograms and 1541
 MEANS procedure 650, 680
 TABULATE procedure 1448
 weighted 1529
 queries
 creating tables from results 1209
 creating views from results 1224
 DBMS queries 1162
 in-line view queries 1230
 query-expression component 1176
 query expressions 1177, 1185
 creating PROC SQL tables from 1133
 creating PROC SQL views from 1134
 validating syntax 1154
 QUIT statement 53, 58
 procedures supporting 58

R

radio boxes 784, 789
 radio buttons 790
 color of 790
 default 789
 definition of 784
 RADIO data set 1658
 RADIOBOX statement
 PMENU procedure 789
 range 1592
 RANGE keyword 1582
 RANGE= statement
 IMPORT procedure 639
 ranges
 for character strings 490
 FORMAT procedure and 464
 RANK 911
 RANK option
 PROC CORR statement 271
 RANK procedure 911
 BY statement 914
 computer resources 916
 concepts 916
 examples 917
 input variables 915
 missing values 916
 output data set 916
 overview 909
 PROC RANK statement 911
 RANKS statement 915
 RANKS statement with VAR statement 915
 results 916
 statistical applications 916
 syntax 911
 task tables 911
 VAR statement 915
 variables for rank values 915
 RANKADJ= option
 PROBPLOT statement (UNIVARIATE) 1493
 QQPLOT statement (UNIVARIATE) 1506
 ranks 909
 groups based on 920
 of multiple variables 917
 values within BY groups 918
 RANKS statement
 RANK procedure 915
 raw data 541
 for examples 1615
 informat for 447, 480
 RBREAK statement
 REPORT procedure 996
 RBUTTON statement
 PMENU procedure 790
 READ= option
 MODIFY statement (DATASETS) 368
 PROC DATASETS statement 333
 rectangular correlation statistics 295
 REF= option
 CHART procedure 181
 PLOT statement (TIMEPLOT) 1374
 REFCHAR= option
 PLOT statement (TIMEPLOT) 1375
 reference lines 1566
 reflexive joins 1167
 REFPOINT= option
 INSET statement (UNIVARIATE) 1478
 REFRESH option
 INDEX CENTILES statement (DATASETS) 363
 registry
 clearing SASUSER 926
 comparing file contents with 927, 934
 comparing registries 926, 927, 935
 debugging 927
 exporting contents of 927
 importing to 928, 932
 keys, subkeys, and values 928, 929
 listing 933
 listing contents in log 928
 loading system options from 721
 sample entries 931
 SASHELP specification 929
 saving system option settings in 723
 uppercasing key names 929
 writing contents to log 928
 writing SASHELP to log 928
 writing SASUSER to log 929
 REGISTRY 926
 registry files
 creating 930
 key names 930
 sample registry entries 931
 structure of 930
 values for keys 930

- REGISTRY procedure 926
 - creating registry files 930
 - examples 932
 - overview 925
 - PROC REGISTRY statement 926
 - syntax 926
 - task table 926
- relative risk estimates 577
- relative risks 565, 579
- reliability estimation 264
- RELRIK option
 - TABLES statement (FREQ) 537
- remerging data 1192
- Remote Library Services (RLS), with translation
 - tables 1412
- REMOVE option
 - LINE statement (FORMS) 503
- RENAME statement
 - DATASETS procedure 371
- REPAIR statement
 - DATASETS procedure 371
- REPLACE option
 - PROC EXPORT statement 431
 - PROC IMPORT statement 637
 - PROC PRTDEF statement 894
 - PROC STANDARD statement 1247
- REPLACE statement, TRANTAB procedure 1417
- REPORT 959
 - report definitions
 - specifying 971
 - storing and reusing 957, 1055
 - report items 986
- REPORT= option
 - PROC REPORT statement 971
- REPORT procedure 43, 48, 959
 - See also* REPORT procedure windows
 - break lines 952, 977, 999
 - BREAK statement 974
 - building reports 1024
 - BY statement 978
 - CALL DEFINE statement 979
 - column attributes 979
 - COLUMN statement 981
 - columns 981, 1049
 - compute blocks 949, 983
 - COMPUTE statement 983
 - concepts 944
 - customized summaries 995, 1060
 - DATA step variables 1024, 1030
 - default summaries 974, 996
 - DEFINE statement 986
 - ENDCOMP statement 994
 - ending program statements 994
 - examples 1037
 - formatting characters 963
 - FREQ statement 994
 - groups 1026, 1075
 - header arrangement 981
 - layout of reports 944
 - LINE statement 995
 - missing values 951, 990, 1067
 - output data set 1070
 - overview 939
 - panels 1058
 - percentage calculation 1064
 - printing reports 956
 - PROC REPORT statement 959
 - RBREAK statement 996
 - report definitions 957, 1055
 - report items 986
 - report types 939
 - report variables 1024
 - rows 1040, 1047
 - sample reports 939
 - selecting report variables 1038
 - statistics 949, 1043, 1053
 - style elements 953, 1078, 1083
 - summaries 1026
 - summary lines 1025
 - syntax 959
 - task tables 959, 974, 986, 996
 - WEIGHT statement 1000
 - weighting analysis variables 1000
 - windowing environment 939, 1001
- REPORT procedure windows 1001
 - BREAK 1001
 - COMPUTE 1004
 - COMPUTED VAR 1004
 - DATA COLUMNS 1005
 - DATA SELECTION 1005
 - DEFINITION 1006
 - DISPLAY PAGE 1011
 - EXPLORE 1012
 - FORMATS 1013
 - LOAD REPORT 1013
 - MESSAGES 1014
 - PROFILE 1014
 - PROMPTER 1015
 - REPORT 1016
 - ROPTIONS 1016
 - SAVE DATA SET 1021
 - SAVE DEFINITION 1021
 - SOURCE 1022
 - STATISTICS 1022
 - WHERE 1023
 - WHERE ALSO 1024
- report variables 1024
- REPORT window, REPORT procedure 1016
 - printing from 956
- report-writing procedures 3, 4
- reports 939
 - across variables 946, 987
 - analysis variables 946, 987, 1000
 - building 1024
 - code for 966
 - colors for 988, 997
 - column for each variable value 1049
 - computed variables 947, 989, 1072
 - detail reports 939
 - display variables 945, 989
 - from DICTONARY tables 1218
 - group variables 946, 990
 - help for 966
 - HTML 838, 856, 871
 - ID variables 990
 - layout of 944
 - multiple observations in one row 1047
 - on multiple-choice survey data 1338
 - order variables 945, 991
 - ordering rows in 1040
 - panels 1058
 - PDF 842
 - percentages in 1064
- PostScript 860
- printing 956
 - printing, from batch mode 957
 - printing, from interactive line mode 957
 - printing, from noninteractive mode 957
 - printing, from Output window 957
 - printing, from REPORT window 956
 - printing, with forms 956
 - printing, with ODS 956
 - printing, with PRINTTO procedure 957
 - printing observations 817
 - RTF 846, 866
 - samples of 939
 - selecting variables for 1038
 - sharing columns 948
 - statistics in 1043, 1053
 - stub-and-banner reports 1347
 - summary reports 939
 - suppressing 967
 - variables 945
 - variables, position and usage 947
 - XML 851
- RESET statement
 - SQL procedure 1141
- RESUME option
 - AUDIT statement (DATASETS) 342
- REVERSE option
 - PLOT statement (TIMEPLOT) 1375
 - PROC SORT statement 1098
- rich text format (RTF) 36, 45
- RIGHT option
 - DEFINE statement (REPORT) 992
- risk differences 562
- RISKDIFF option
 - TABLES statement (FREQ) 537
- RISKDIFFC option
 - TABLES statement (FREQ) 537
- risks 562
- RLS (Remote Library Services), with translation
 - tables 1412
- robust estimators 1525, 1549
 - robust measures of scale 1527
 - trimmed means 1526
 - Winsorized means 1525
- robust measures of scale 1527
- ROBUSTSCALE option
 - PROC UNIVARIATE statement 1448
- ROLLBACK statement (SQL) 1206
- ROPTIONS window, REPORT procedure 1016
- ROUND option
 - PICTURE statement (FORMAT) 453
 - PROC PRINT statement 823
 - PROC UNIVARIATE statement 1449
- rounding 1511
- row dimension 1265
- row headings
 - customizing 1322
 - eliminating 1326
 - indenting 1328
- row mean scores statistic 576
- ROW= option
 - TABLE statement (TABULATE) 1285
- rows 1115
 - See also* observations
 - adding to tables or views 1140
 - counting 1191
 - deleting 1136

- deleting through views 1136
- inserting through views 1141
- joins and 1166
- ordering 1151
- returned by subqueries 1163
- selecting 1155
- selecting from tables and views 1142
- ROWS= option
 - PROC PRINT statement 824
- RTF destination 40, 45
- RTF files 1078, 1357
- RTF output 36
- RTF reports 846, 866
- RTINCLUDE option
 - HISTOGRAM statement (UNIVARIATE) 1470
- RTSPACE= option
 - TABLE statement (TABULATE) 1285
- RUN-group processing 19

S

- S= option, PLOT statement (PLOT) 742
- samples 1587
- samples of formatted output 33
- sampling distribution 1597
- SAS/ACCESS views 1115
 - updating 1203
- SAS/AF applications, executing 413
- SAS data views 1115
 - DICTIONARY tables 1199
- SAS files
 - converting 199
- SAS formatted destinations 43
- SAS/GRAPH software, translation tables
 - in 1412
- SAS Registry
 - See REGISTRY procedure
- SASHELP views 1199
 - retrieving information about 1200
- SASUSER library
 - Ghostview printer definition in 900
- SAVAGE option
 - PROC RANK statement 913
- SAVE DATA SET window, REPORT procedure 1021
- SAVE DEFINITION window, REPORT procedure 1021
- SAVE statement
 - CATALOG procedure 153
 - DATASETS procedure 373
 - TRANTAB procedure 1418
- SCALE= option
 - HISTOGRAM statement (UNIVARIATE) 1470
 - PROBPLOT statement (UNIVARIATE) 1494
 - QQPLOT statement (UNIVARIATE) 1506
- scale parameters 1539
- schedule calendars 75, 76, 97
- scores 545
- SCORES= option
 - TABLES statement (FREQ) 537
- SCOROUT option
 - TABLES statement (FREQ) 538
- searching for patterns 1174, 1238
- SEED= option
 - EXACT statement (FREQ) 524
- SELECT clause, SQL procedure 1142
- SELECT statement
 - CATALOG procedure 153
 - CIMPORT procedure 204
 - CPORT procedure 314
 - DATASETS procedure 374
 - FORMAT procedure 459
 - PRTEXP procedure 906
 - SQL procedure 1142
- selection lists 48
- SELECTION statement
 - PMENU procedure 791
- SEPARATOR statement
 - PMENU procedure 791
- set membership 1164
- set operators 1177, 1206
- SET statement
 - appending data 337
- SETS= option
 - PROC FORMS statement 500
- SHAPE= option
 - HISTOGRAM statement (UNIVARIATE) 1470
 - PROBPLOT statement (UNIVARIATE) 1494
 - QQPLOT statement (UNIVARIATE) 1506
- shape parameters 1539
- Shapiro-Wilk statistic 1521
- Shapiro-Wilk test 1448
- SHEET= statement
 - EXPORT procedure 432
 - IMPORT procedure 640
- SHORT option
 - CONTENTS statement (DATASETS) 347
 - PROC OPTIONS statement 716
- SHOWALL option
 - PROC REPORT statement 971
- SIGMA= option
 - HISTOGRAM statement (UNIVARIATE) 1470
 - PROBPLOT statement (UNIVARIATE) 1494
 - QQPLOT statement (UNIVARIATE) 1506
- sign test 1518, 1519
 - paired data and 1552
- significance 1608
- simple indexes 1129
- simple kappa coefficient 570
- simple random sample 1587
- SINGULAR= option
 - PROC CORR statement 271
- singularity of variables 271
- skewness 1593
- SKEWNESS keyword 1582
- SKIP option
 - BREAK statement (REPORT) 976
 - RBREAK statement (REPORT) 998
 - PROC FORMS statement 500
- SLIST= option, PLOT statement (PLOT) 742
- SLOPE= option
 - PROBPLOT statement (UNIVARIATE) 1494
 - QQPLOT statement (UNIVARIATE) 1507
- Somers' D 554
- SORT 1093
- SORT procedure 1093
 - ASCII order 1101
 - BY statement 1100
- character variable sorting orders 1101
- collating-sequence options 1094
- concepts 1100
- EBCDIC order 1101
- examples 1103
- integrity constraints 1102
- maintaining relative order of observations 1107
- multi-threaded sorting 1100
- numeric variable sorting orders 1100
- output 1103
- output data set 1103
- overview 1091
- PROC SORT statement 1093
- results 1103
- retaining first observation of BY groups 1110
- sorting by multiple variable values 1103
- sorting in descending order 1105
- stored sort information 1102
- syntax 1093
- task tables 1093, 1103
- translation tables in 1411
- variables specification 1100
- SORTEDBY= option
 - MODIFY statement (DATASETS) 368
- sorting
 - by multiple variable values 1103
 - data retrieved by views 1134
 - in descending order 1105
 - multi-threaded 1100
 - stored sort information 1102
 - translation tables for 1429
- sorting observations 1091
- sorting orders
 - ASCII 1094, 1101
 - EBCDIC 1095, 1101
 - for character variables 1101
 - for numeric variables 1100
- SORTMSG option
 - PROC SQL statement 1123
- SORTSEQ= option
 - PROC SORT statement 1095
 - PROC SQL statement 1123
- SORTSIZE= option
 - PROC SORT statement 1098
- SOUNDS-LIKE operator 1231
- SOURCE window, REPORT procedure 1022
- SPACE= option
 - CHART procedure 181
- SPACING= option
 - DEFINE statement (REPORT) 992
 - PROC REPORT statement 972
- SPARSE option
 - TABLES statement (FREQ) 538
- Spearman correlation statistics 271, 272, 556
 - Spearman partial correlation 264, 274
 - Spearman rank-order correlation 263, 280, 292
- SPEARMAN option
 - PROC CORR statement 272
- SPLIT= option
 - PLOT statement (PLOT) 743
 - PROC PRINT statement 824
 - PROC REPORT statement 972
 - PROC TIMEPLOT statement 1368
- spread of values 1592

- spreadsheets
 - exporting 432
 - importing 638, 645
- SQL 1117
- SQL, embedded 1207
- SQL components 1154
 - BETWEEN condition 1155
 - BTRIM function 1155
 - CALCULATED 1156
 - CASE expression 1157
 - COALESCE function 1158
 - column-definition 1159
 - column-modifier 1160
 - column-name 1161
 - CONNECTION TO 1162
 - CONTAINS condition 1163
 - EXISTS condition 1163
 - IN condition 1164
 - IS condition 1164
 - joined-table 1165
 - LIKE condition 1174
 - LOWER function 1176
 - query-expression 1176
 - sql-expression 1182
 - SUBSTRING function 1189
 - summary-function 1190
 - table-expression 1196
 - UPPER function 1197
- sql-expression component 1182
- SQL procedure 1117
 - See also* PROC SQL tables
 - See also* PROC SQL views
 - See also* SQL components
 - ALTER TABLE statement 1125
 - ANSI Standard and 1204
 - coding conventions 1116
 - collating sequences 1205
 - column modifiers 1205
 - concepts 1197
 - CONNECT statement 1128
 - CREATE INDEX statement 1129
 - CREATE TABLE statement 1130
 - CREATE VIEW statement 1134
 - data set options with 1197
 - data types and dates 1159
 - DBMS connections 1128, 1138, 1198
 - DBMS statements 1139
 - DELETE statement 1136
 - DESCRIBE statement 1137
 - DICTIONARY tables 1199
 - DISCONNECT statement 1138
 - displaying list of styles 47
 - DROP statement 1138
 - examples 1207
 - EXECUTE statement 1139
 - FROM clause 1147
 - functions supported by 1182, 1206
 - GROUP BY clause 1149
 - HAVING clause 1150
 - indexes 1129, 1138
 - INSERT statement 1140
 - INTO clause 1144
 - macro variables set by 1202
 - missing values 1164, 1207, 1240
 - naming conventions 1206
 - ORDER BY clause 1151, 1205
 - orthogonal expressions 1205
 - overview 1115
 - PROC SQL statement 1120
 - reserved words 1204
 - RESET statement 1141
 - resetting options 1141
 - SELECT clause 1142
 - SELECT statement 1142
 - statistical functions 1206
 - subsetting grouped data 1150
 - subsetting output 1149
 - syntax 1117
 - task tables 1119, 1120
 - three-valued logic 1207
 - UPDATE statement 1153
 - user privileges 1207
 - VALIDATE statement 1154
 - WHERE clause 1149
- SQL Procedure Pass-Through Facility
 - DBMS connections 1198
 - return codes 1198
- SQLOBS macro variable 1202
- SQLOOPS macro variable 1202
- SQLRC macro variable 1202
- SQLXMSG macro variable 1202
- SQLXRC macro variable 1202
- SQUARE option
 - PROBPLOT statement (UNIVARIATE) 1495
 - QQPLOT statement (UNIVARIATE) 1507
- SSCP option
 - PROC CORR statement 272
- STANDARD 1245
- standard deviation 272, 1582, 1593
- standard error of the mean 1582, 1598
- STANDARD procedure 1245
 - BY statement 1248
 - examples 1251
 - FREQ statement 1249
 - frequency of observations 1249
 - missing values 1250
 - output data set 1250
 - overview 1243
 - PROC STANDARD statement 1246
 - results 1250
 - statistical computations 1250
 - syntax 1245
 - task tables 1245, 1246
 - VAR statement 1249
 - variables, order of 1249
 - variables, specifying 1249
 - WEIGHT statement 1249
 - weights for analysis variables 1249
- standardized variables 1243
- star charts 169, 176
- STAR statement
 - CHART procedure 176
- START statement
 - CALENDAR procedure 95
- STARTAT= option
 - PROC REGISTRY statement 929
- STATE= option
 - ITEM statement (PMENU) 787
- statements, with same function in multiple procedures 53
- STATEPOP data set 1670
- STATES option, PLOT statement (PLOT) 743
- statistic, defined 1587
- statistic option
 - DEFINE statement (REPORT) 992
- statistical analysis
 - transposing data for 1404
- statistical procedures 3, 6
 - efficiency issues 7
- statistical summaries 1190
- statistically significant 1608
- statistics 31, 1586
 - based on number of arguments 1191
 - computational requirements for 32
 - descriptive statistics 31, 1257
 - for groups of observations 7
 - formulas for 1578
 - in tabular format 1260
 - keywords for 1578
 - measures of location 1588
 - measures of shape 1593
 - measures of variability 1592
 - normal distribution 1594
 - percentiles 1588
 - populations 1586
 - REPORT procedure 949
 - samples 1587
 - sampling distribution 1597
 - summarization procedures 1586
 - TABULATE procedure 1291
 - testing hypotheses 1607
 - weights 1586
- statistics procedures 1577
- STATISTICS window, REPORT procedure 1022
- STATS option
 - PROC COMPARE statement 220
- STD keyword 1582
- STD= option
 - PROC STANDARD statement 1247
- STDDEV keyword 1582
- STDERR keyword 1582
- STDMEAN keyword 1582
- stem-and-leaf plots 1448, 1512
- STIMER option
 - PROC SQL statement 1123
- stratified tables 609
- string comparison operators
 - truncated 1184
- Structured Query Language (SQL)
 - See* SQL procedure
- Stuart's tau-c 553
- stub-and-banner reports 1347
- Student's t distribution 1609
- Student's t statistic 1585
 - two-tailed p-value 1585
- Student's t test 680, 1518
- STYLE= attribute
 - CALL DEFINE statement (REPORT) 981
- style attributes 46, 47
- style definitions 47
 - Base procedures with 48
 - list of 47
 - SAS-supplied 47
- style elements 44, 47
 - for ODS output 1078
 - in dimension expressions 1288
- PRINT procedure 825, 831, 832, 834
- REPORT procedure 953, 972, 1078, 1083
- TABULATE procedure 1273, 1281, 1298, 13

STYLE= option
 BREAK statement (REPORT) 976
 CLASS statement (TABULATE) 1279
 CLASSLEV statement (TABULATE) 1280
 COMPUTE statement (REPORT) 984
 DEFINE statement (REPORT) 993
 ID statement (PRINT) 831
 KEYWORD statement (TABULATE) 1281
 PROC PRINT statement 825
 PROC REPORT statement 972
 PROC TABULATE statement 1273
 RBREAK statement (REPORT) 998
 REPORT procedure 953
 SUM statement (PRINT) 832
 TABLE statement (TABULATE) 1285
 TABULATE procedure 1298
 VAR statement (PRINT) 834
 VAR statement (TABULATE) 1290
 SUBGROUP= option
 CHART procedure 181
 SUBMENU statement
 PMENU procedure 792
 submenus 792
 subqueries 1185
 compared with joins 1174
 correlated 1187
 efficiency and 1188
 returning rows 1163
 subsetting data 64, 1149, 1150
 SUBSTITUTE= option
 CHECKBOX statement (PMENU) 782
 RBUTTON statement (PMENU) 790
 SUBSTRING function (SQL) 1189
 subtables 1265
 SUM keyword 1583
 sum of squares, corrected 1580
 sum of squares, uncorrected 1583
 sum of the weights 1583
 SUM option
 CHART procedure 182
 SUM statement
 CALENDAR procedure 96
 PRINT procedure 832
 SUMBY statement
 PRINT procedure 833
 summarization procedures, data requirements 1586
 SUMMARIZE option
 BREAK statement (REPORT) 976
 RBREAK statement (REPORT) 998
 summarizing data 650, 1191
 summary calendars 78, 98, 103
 summary-function component 1190
 summary lines 939
 construction of 1025
 SUMMARY procedure 1258
 overview 1257
 PROC SUMMARY statement 1258
 syntax 1258
 VAR statement 1259
 summary reports 939
 summary statistics
 in output data sets 253
 insets of 1474
 sums of squares and crossproducts 272
 SUMSIZE= option
 PROC MEANS statement 659

SUMVAR= option
 CHART procedure 182
 SUMWGT keyword 1583
 superimposing plots 1382
 SUPPRESS option
 BREAK statement (REPORT) 977
 survey data
 multiple-choice 1338
 multiple-response 1333
 SUSPEND option
 AUDIT statement (DATASETS) 342
 SWAP statement, TRANTAB procedure 1419
 SWEDISH option
 PROC SORT statement 1095, 1096
 SYMBOL= option
 CHART procedure 182
 symbol variables 1370
 SYSINFO return code 228
 system options 17
 current settings list 713
 host options listing 716
 loading from registry or data sets 721
 log listing of 713
 portable options listing 716
 saving current settings 723
 system resources
 ODS destinations and 46

T

T keyword 1585
 t test 650
 Student's 1518
 Student's t test 680
 table aliases 1148, 1166
 TABLE= argument
 PROC DBCSTAB statement 407
 PROC IMPORT statement 635
 table attributes 46
 table definitions 39, 46, 1137
 for customizing output 50
 modifying 51
 table elements 46
 table-expression component 1196
 table expressions 1177
 TABLE= option
 SAVE statement (TRANTAB) 1418
 TABLE statement
 TABULATE procedure 1282
 tables
See also PROC SQL tables
 class variables combinations 1312
 crosstabulation 1347
 describing for printing 1282
 formatting values in 1293
 multipage 1330
 subtables 1265
 two-dimensional 1310
 TABLES statement, FREQ procedure 528
 options 529
 output data set 590
 requests 528
 with EXACT statement 524
 with OUTPUT statement 525
 without options 528
 tabular reports for statistics 1260
 TABULATE 1266
 TABULATE procedure 43, 48
 analysis variables 1289, 1291
 BY-group processing 1294
 BY statement 1275
 CLASS statement 1276
 class variables 1276
 class variables, formatting 1292, 1315
 class variables, level value headings 1280
 class variables, missing values 1279
 CLASSLEV statement 1280
 concepts 1291
 dimension expressions 1265, 1286
 examples 1310
 formatting characters 1269
 formatting values in tables 1293
 FREQ statement 1280
 frequency counts 1347
 headings 1305, 1307, 1308
 KEYLABEL statement 1281
 KEYWORD statement 1281
 missing values 1270, 1279, 1299
 ODS output 1357
 overview 1260
 page dimension 1265, 1294
 percentages 1294, 1344, 1347
 PROC TABULATE statement 1267
 results 1299
 statistics 1291
 style elements 1273, 1281, 1298, 13
 syntax 1266
 table descriptions 1282
 TABLE statement 1282
 task tables 1266, 1267, 1282
 terminology 1263
 VAR statement 1289
 WEIGHT statement 1291
 tagsets 44
 SAS-supplied and -supported 41
 TAGSORT option
 PROC SORT statement 1099
 TAPE option
 PROC CIMPORT statement 203
 PROC CPORT statement 313
 TEMPLATE procedure
 displaying list of styles 47
 temporary data sets 16
 TERMINATE option
 AUDIT statement (DATASETS) 342
 terminology 39
 TEST statement
 FREQ procedure 539
 TESTF= option
 TABLES statement (FREQ) 538
 TESTP= option
 TABLES statement (FREQ) 538
 tests for location 1518
 sign test 1519
 Student's t test 1518
 Wilcoxon signed rank test 1519
 text fields 784, 792
 TEXT statement
 PMENU procedure 792
 theoretical distributions 1536
 THETA= option
 HISTOGRAM statement (UNIVARIATE) 1470

PROBPLOT statement (UNIVARIATE) 1495
 QQPLOT statement (UNIVARIATE) 1507
 third-party formatted destinations 44
 formatting features 45
 HTML 44
 MARKUP 44
 PRINTER 45
 RTF 45
 threads
 multi-threaded sorting 1100
 THREADS option
 PROC MEANS statement 659
 PROC SQL statement 1123
 SORT procedure 1099
 three-parameter Weibull distribution 1538
 three-valued logic 1207
 THRESHOLD= option
 HISTOGRAM statement (UNIVARIATE) 1471
 PROBPLOT statement (UNIVARIATE) 1495
 QQPLOT statement (UNIVARIATE) 1508
 TIES= option
 PROC RANK statement 913
 tiles 1465
 TIMEPLOT 1368
 TIMEPLOT procedure 1368
 BY statement 1369
 CLASS statement 1370
 customizing axes 1378
 data considerations 1375
 examples 1376
 ID statement 1370
 missing values 1376
 multiple observations, on one plot line 1384
 output 1375
 overview 1365
 page layout 1375
 PLOT statement 1371
 plotting a single variable 1376
 plotting symbols 1378, 1380
 printing variable values 1370
 PROC TIMEPLOT statement 1368
 results 1375
 superimposing plots 1382
 symbol variables 1370
 syntax 1368
 task tables 1368, 1371
 titles
 BY-group information in 19, 20
 BY line in 23
 TOTPCT option
 TABLES statement (FREQ) 538
 trace records 50
 traditional SAS output 33
 TRANSLATE= option
 PROC CPORT statement 313
 translation tables 1409
 applying to transport files 320
 character sets and 1410
 creating 1420
 device-to-operating environment translation 1412
 editing 1423, 1425, 1431
 exchanging 1419
 hexadecimal representation of 1416
 in SAS/GRAPH software 1412
 in SORT procedure 1411

inverse tables 1415, 1427
 loading into memory for editing 1416
 modifying Institute-supplied tables 1411
 operating environment-to-device translation 1412
 outside TRANTAB procedure 1411
 positions 1409, 1410
 positions, setting to zero 1415
 purposes of 1410
 replacing characters in 1417
 saving 1418
 sorting data 1429
 storing 1410
 table one area 1412
 table two area 1412
 viewing 1419
 with CIMPORT procedure 1411
 with CPORT procedure 1411
 with Remote Library Sservices (RLS) 1412
 transport files 199
 applying translation tables to 320
 importing 199
 moving between hosts 260
 writing 307
 transporting
 data sets 260
 password-protected data sets 316
 TRANSPPOSE 1390
 TRANSPPOSE option
 PROC COMPARE statement 220
 TRANSPPOSE procedure 1390
 BY statement 1391
 COPY statement 1393
 copying variables without transposing 1393
 duplicate ID values 1393
 examples 1396
 formatted ID values 1393
 ID statement 1393
 IDLABEL statement 1394
 labeling transposed variables 1394
 listing variables to transpose 1395
 missing values 1394
 output data set 1395
 overview 1387
 PROC TRANSPPOSE statement 1390
 results 1395
 statistical analysis and 1404
 syntax 1390
 task table 1390
 transposing BY groups 1392, 1400
 VAR statement 1395
 variable names, from numeric values 1394
 transposed variables 1387
 attributes of 1396
 labeling 1394, 1399
 naming 1396, 1398, 1402
 TRANTAB procedure 1413
 character sets 1410
 CLEAR statement 1415
 concepts 1410
 examples 1419
 exchanging tables 1419
 hexadecimal tables 1416
 INVERSE statement 1415
 LIST statement 1416
 LOAD statement 1416
 loading tables into memory for editing 1416

modifying Institute-supplied translation tables 1411
 overview 1409
 positions, setting to zero 1415
 PROC TRANTAB statement 1414
 REPLACE statement 1417
 replacing characters in tables 1417
 SAVE statement 1418
 saving tables 1418
 storing translation tables 1410
 SWAP statement 1419
 syntax 1413
 table two, creating 1415
 task table 1413
 translation tables, outside TRANTAB procedure 1411
 TRANTAB statement
 CPORT procedure 315
 TRAP option
 PROC TABULATE procedure 1274
 TREND option
 TABLES statement (FREQ) 538
 trimmed means 1526
 TRIMMED= option
 PROC UNIVARIATE statement 1449
 truncated string comparison operators 1184
 TURNVLABELS option
 HISTOGRAM statement (UNIVARIATE) 1471
 two-dimensional tables 1310
 TWO option
 CLEAR statement (TRANTAB) 1415
 LIST statement (TRANTAB) 1416
 SAVE statement (TRANTAB) 1418
 two-parameter Weibull distribution 1538
 two-tailed tests 1608
 Type I error rate 1608
 Type II error rate 1609
 TYPE= option
 CHART procedure 182
 MODIFY statement (DATASETS) 369
 TRANTAB statement (CPORT) 316
 TYPES statement
 MEANS procedure 672

U

UCLM keyword 1586
 UID= statement
 EXPORT procedure 433
 IMPORT procedure 641
 UL option
 BREAK statement (REPORT) 977
 RBREAK statement (REPORT) 999
 uncertainty coefficient asymmetric 559
 uncertainty coefficient symmetric 560
 uncorrected sum of squares 1583
 underlining 975, 977, 997, 999
 UNDO_POLICY= option
 PROC SQL statement 1124
 UNIFORM option
 PROC PLOT statement 731
 PROC TIMEPLOT statement 1368
 UNINSTALL= option
 PROC REGISTRY statement 929
 union joins 1171

- UNION operator 1179
 - UNIQUE keyword 1129
 - UNIQUE option
 - CREATE INDEX statement (DATASETS) 364
 - UNIT= argument, PROC FSLIST statement 628
 - UNIT= option
 - PROC PRINTTO statement 882
 - univariate analysis
 - for multiple variables 1543
 - UNIVARIATE procedure 1443
 - analysis variables 1510, 1560
 - BY statement 1451, 1484
 - CLASS statement 1452
 - classification levels 1452
 - computer resources 1516
 - concepts 1511
 - confidence limits 1517
 - data distribution 1555
 - density curves 1561
 - examples 1543
 - extreme values 1473, 1546
 - fitted continuous distributions 1530
 - FREQ statement 1455
 - goodness-of-fit tests 1520
 - high-resolution graphics 1514
 - HISTOGRAM statement 1455
 - histograms 1541, 1546, 1568
 - ID statement 1473
 - INSET statement 1474
 - keywords 1474, 1482
 - line printer plots 1512
 - missing values 1453, 1540
 - mode calculation 1530
 - normal probability plots 1513, 1566
 - ODS table names 1541
 - OUTHISTOGRAM= data set 1543
 - output data set 1482, 1542, 1560
 - OUTPUT statement 1482, 1510
 - overview 1436
 - percentiles 1483, 1485, 1528, 15
 - probability plots 1485, 1514, 1515, 15
 - PROBPLOT statement 1485
 - PROC UNIVARIATE statement 1444
 - QQPLOT statement 1497
 - quantile-quantile plots 1497, 1514, 1515, 15
 - results 1540
 - robust estimators 1525, 1549
 - rounding 1511
 - sign test 1519, 1552
 - statistical computations 1517
 - summary statistics 1474
 - syntax 1443
 - task tables 1443, 1444, 1455, 14
 - tests for location 1518
 - VAR statement 1510
 - WEIGHT statement 1510
 - universe 1586
 - UPCASE option
 - INVALUE statement (FORMAT) 448
 - PROC REGISTRY statement 929
 - UPDATE statement
 - SQL procedure 1153
 - UPDATECENTILES= option
 - CREATE INDEX statement (DATASETS) 364
 - INDEX CENTILES statement (DATASETS) 363
 - UPPER function (SQL) 1197
 - UPPER= option
 - HISTOGRAM statement (UNIVARIATE) 1471
 - USER data library 17
 - USER literal 1183
 - USER_VAR option
 - AUDIT statement (DATASETS) 341
 - USESASHELP option
 - PROC PRTDEF statement 894
 - PROC REGISTRY statement 929
 - USESSASHELP option
 - PROC PRTEXP statement 906
 - USS keyword 1583
 - utility procedures 4, 8
- ## V
- V5FMT option
 - TABLES statement (FREQ) 539
 - VALIDATE statement
 - SQL procedure 1154
 - value formats
 - multilabel 693
 - VALUE option
 - PROC OPTIONS statement 717
 - value-range-sets 464
 - VALUE statement
 - FORMAT procedure 459
 - VAR keyword 1583
 - VAR statement
 - CALENDAR procedure 96
 - COMPARE procedure 223
 - CORR procedure 275
 - MEANS procedure 673
 - PRINT procedure 833
 - RANK procedure 915
 - STANDARD procedure 1249
 - SUMMARY procedure 1259
 - TABULATE procedure 1289
 - TRANSPOSE procedure 1395
 - UNIVARIATE procedure 1510
 - VARDEF= option
 - PROC CORR statement 272
 - PROC MEANS statement 659
 - PROC REPORT statement 972
 - PROC STANDARD statement 1247
 - PROC TABULATE statement 1274
 - PROC UNIVARIATE statement 1449
 - variability 1592
 - variable formats 228
 - variables 1115
 - See also* columns
 - associating informats/formats with 465
 - attributes of 366
 - class variables 1276
 - comparing 209, 243, 244, 246
 - formatted values 25
 - in reports 945
 - labels of 366
 - matching 210
 - modifying, within a data set 326
 - nested 1265
 - plotting over time intervals 1365
 - printing values of 1370
 - renaming 371
 - shortcuts for lists of variable names 24
 - standardizing 1243
 - symbol variables 1370
 - transposing into observations 1387
 - variance 1583, 1593
 - variances 272
 - VARNUM option
 - CONTENTS statement (DATASETS) 347
 - VAXIS= option
 - HISTOGRAM statement (UNIVARIATE) 1471
 - VAXIS= option, PLOT statement (PLOT) 743
 - VAXISLABEL= option
 - HISTOGRAM statement (UNIVARIATE) 1471
 - PROBPLOT statement (UNIVARIATE) 1495
 - QQPLOT statement (UNIVARIATE) 1508
 - VBAR statement
 - CHART procedure 177
 - VERIFY option
 - PROC DBCSTAB statement 408
 - vertical bar charts 177
 - subdividing bars 187
 - VEXPAND option, PLOT statement (PLOT) 743
 - view definitions 1137
 - views 1115
 - See also* PROC SQL views
 - DATA step views 1115
 - in-line 1148, 1205, 1230
 - SAS/ACCESS views 1115, 1203
 - SAS data views 1115, 1199
 - SASHELP views 1199, 1200
 - VMINOR= option
 - HISTOGRAM statement (UNIVARIATE) 1471
 - PROBPLOT statement (UNIVARIATE) 1495
 - QQPLOT statement (UNIVARIATE) 1508
 - VOFFSET= option
 - HISTOGRAM statement (UNIVARIATE) 1471
 - VPERCENT= option, PROC PLOT statement 731
 - VPOS= option, PLOT statement (PLOT) 743
 - VREF= option
 - HISTOGRAM statement (UNIVARIATE) 1471
 - PROBPLOT statement (UNIVARIATE) 1495
 - QQPLOT statement (UNIVARIATE) 1508
 - VREF= option, PLOT statement (PLOT) 743
 - VREFCHAR= option, PLOT statement (PLOT) 743
 - VREFLABELS= option
 - HISTOGRAM statement (UNIVARIATE) 1471
 - PROBPLOT statement (UNIVARIATE) 1495
 - QQPLOT statement (UNIVARIATE) 1508
 - VREFLABPOS= option
 - HISTOGRAM statement (UNIVARIATE) 1472
 - PROBPLOT statement (UNIVARIATE) 1495
 - QQPLOT statement (UNIVARIATE) 1508
 - VREVERSE option, PLOT statement (PLOT) 744

VSCALE= option
 HISTOGRAM statement (UNIVARI-
 ATE) 1472
 VSPACE= option, PLOT statement (PLOT) 744
 VTOH= option, PROC PLOT statement 731
 VZERO option, PLOT statement (PLOT) 744

W

W= option
 HISTOGRAM statement (UNIVARI-
 ATE) 1472
 PROBPLOT statement (UNIVARIATE) 1496
 QQPLOT statement (UNIVARIATE) 1508
 WARNING option
 PROC COMPARE statement 220
 WAXIS= option
 HISTOGRAM statement (UNIVARI-
 ATE) 1472
 PROBPLOT statement (UNIVARIATE) 1496
 QQPLOT statement (UNIVARIATE) 1508
 WAYS option
 OUTPUT statement (MEANS) 672
 WAYS statement
 MEANS procedure 674
 WBARLINE= option
 HISTOGRAM statement (UNIVARI-
 ATE) 1472
 WBUILD macro 806
 WEEKDAYS option
 PROC CALENDAR statement 86
 Weibull distribution 1534
 three-parameter 1538
 two-parameter 1538
 WEIBULL option
 HISTOGRAM statement (UNIVARI-
 ATE) 1472
 PROBPLOT statement (UNIVARIATE) 1496

 QQPLOT statement (UNIVARIATE) 1508
 WEIBULL2 option
 PROBPLOT statement (UNIVARIATE) 1496
 QQPLOT statement (UNIVARIATE) 1509
 WEIGHT= option
 DEFINE statement (REPORT) 993
 VAR statement (MEANS) 673
 VAR statement (TABULATE) 1290
 WEIGHT statement 53, 59
 calculating weighted statistics 60
 CORR procedure 275
 example 60
 FREQ procedure 540
 MEANS procedure 675
 procedures supporting 59
 REPORT procedure 1000
 STANDARD procedure 1249
 TABULATE procedure 1291
 UNIVARIATE procedure 1510
 weight values 963, 1268, 1446
 weighted kappa coefficient 571
 weighted quantiles 1529
 weighted statistics 60
 weights 1586
 weights for analysis variables 59
 WGDB= statement
 EXPORT procedure 433
 IMPORT procedure 641
 WGRID= option
 HISTOGRAM statement (UNIVARI-
 ATE) 1473
 WHERE ALSO window, REPORT proce-
 dure 1024
 WHERE clause, SQL procedure 1149
 WHERE statement 54, 64
 example 64
 procedures supporting 64
 WHERE window, REPORT procedure 1023

WIDTH= option
 CHART procedure 183
 DEFINE statement (REPORT) 993
 PROC FORMS statement 500
 PROC PRINT statement 828
 Wilcoxon signed rank test 1518, 1519
 windows
 associating with menus 810
 WINDOWS option
 PROC REPORT statement 973
 Winsorized means 1450, 1525
 WINSORIZED= option
 PROC UNIVARIATE statement 1450
 WITH statement
 COMPARE procedure 224
 CORR procedure 276
 WORKDATA= option
 PROC CALENDAR statement 86
 workdays data set 86, 105, 106
 WRAP option
 PROC REPORT statement 973
 WRITE= option
 MODIFY statement (DATASETS) 369

X

XML files 851
 XML output 38

Z

zero weights 541
 ZEROS option
 WEIGHT statement (FREQ) 541
 ZETA= option
 HISTOGRAM statement (UNIVARI-
 ATE) 1473
 PROBPLOT statement (UNIVARIATE) 1496
 QQPLOT statement (UNIVARIATE) 1509

Your Turn

If you have comments or suggestions about *Base SAS® 9 Procedures Guide*, please send them to us on a photocopy of this page, or send us electronic mail.

For comments about this book, please return the photocopy to

SAS Publishing
SAS Campus Drive
Cary, NC 27513
email: yourturn@sas.com

Send suggestions about the software, please return the photocopy to

SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
email: suggest@sas.com