统计软件和数学应用软件课程讲稿

原著：李东风
课件制作：李东风

2016 年 11 月 29 日

目录

前言

课程介绍—目的

- 掌握用统计软件处理实际统计问题的能力。
  - 学会 SAS 软件和 R 软件。
  - 基本数据整理能力。
  - 典型统计问题的计算及输出结果的分析。
  - 基本统计计算编程能力。
- 复习巩固基础统计课程知识。

SAS 系统介绍

- 国际上公认的权威统计软件系统；
- 统计专业求职的有力武器；
- 大型、复杂、专业、难学。
- 本课程采用深入浅出讲法。
- 功能包括：
  - 数据管理，包括高效、方便地访问大型数据库；
  - 统计分析；
  - 报表图形；
  - 信息系统开发。

S 语言介绍

- 统计计算编程的强大工具；
- 包括商业版本 S-Plus 和自由软件 R；
- 自由软件 R 在全世界统计界得到广泛支持，很多新的统计方法采用 S 编程并发表为 R 的软件包。
- 统计科研机构求职的有力武器。
- 功能：

- 强大的交互分析能力和图形功能；
- 容易学习的统计计算编程；
- 丰富的编程支持：向量、矩阵、对象等;
- 常用的统计分析方法及许多最新的实验性的统计方法。

第一章　SAS 初阶

1.1 初识 SAS

SAS 界面

- 启动：
  - 从桌面程序快捷方式图标；
  - 从开始菜单；
  - 从安装的文件夹中的 SAS.EXE 程序。
- SAS AWS 界面：
  - 窗口:程序 (Program Editor)、运行记录 (Log)、文本输出 (Output)；文件管理 (Explorer)；结果管理 (Results)。
  - 菜单、工具栏、状态栏。
- 从启动到退出称为一个 SAS **会话**。
- SAS 还可以有命令行交互、批运行、远程提交等运行方式。

SAS 快捷方式

- 不同的工作项目使用不同的子目录（文件夹）保存程序和数据。
- 设某个工作项目的目录为 "`C:\sub`", 在此子目录中建立一个 SAS 快捷方式。在资源管理器中用右键菜单 "属性" 打开属性对话框，把 "起始位置" 栏清空。
- 为了能够在中文版和英文版之间选择，对于中文版，把快捷方式属性的 "目标" 栏改为

```
"C:\Program Files\SAS\SAS 9.1\sas.exe"
 -config "C:\Program Files\SAS\SAS 9.1\nls\zh\sasv9.cfg"
```

(如果 SAS 安装位置和安装版本有差别请相应更改)

对于英文版，把快捷方式属性的 "目标" 栏改为

```
"C:\Program Files\SAS\SAS 9.1\sas.exe"
 -config "C:\Program Files\SAS\SAS 9.1\nls\en\sasv9.cfg"
```

简单运行样例

- 一个班学生的数学成绩（满分 100）和语文成绩（满分 120）数据，包括姓名和性别。
- 输入并计算平均分、显示数据、排序后显示。
- 从程序例子看 SAS 程序特点：
  - 以分号结尾；
  - 自由空白；
  - 大小写不区分；
  - 由“数据步”和“过程步”组成。
- 实际运行。Log 窗口的作用：显示运行情况和可能的错误。
- 程序有错时的表现。
- 不论 Output 有无显示都要看 Log 窗口。

1.2 SAS 基本概念

SAS 数据集

- 逻辑概念：行、列结构的表格，每行称为一个观测，每列称为一个变量。类似于矩阵，但可以整列都取字符型值，并且变量名也是数据集的一部分。
- 等同于数据库术语“表 (table)”。
- 物理存储：操作系统中的文件。包括描述部分（文件创建信息、变量名、变量的类型等属性）和数据部分。
- 名字（变量名、数据集名等）：大小写不区分，其它规定与一般的高级语言相同 (由字母、下划线、数字组成，第一个字符必须是字母或下划线，不允许使用汉字以及小数点、空格、减号等特殊字符)，不超过 32 个字符长。

SAS 逻辑库库

- SAS 逻辑库 (library，也称数据库) 与 SAS 数据集的关系，基本等同于操作系统中子目录与子目录中文件的关系。
- SAS 逻辑库有一个“库名 (libref)”，可以认做是操作系统中子目录的一个 SAS 中使用的别名。库名长度不超过 8 个字符。
- 建立 SAS 逻辑库，实际是指把一个操作系统子目录与一个库名联系起来。用 libname 语句或快捷图标中的“New library”图标。
- 如

```
libname samp 'c:\work\sas\sampled';
```

- 用仅有库名的 LIBNAME 语句取消一个库名：

```
libname samp;
```

- 为了查看逻辑库 SAMP 中的所有数据集的列表，可以运行如下的程序:

```
PROC DATASETS LIBRARY=samp
    MEMTYPE=DATA NOLIST;
  CONTENTS DATA=_all_ NODS;
RUN;QUIT;
```

- 三个预定义的库: WORK、SASUSER、SASHELP。
- 数据集的访问格式为“库名. 数据集名”格式，比如 SASHELP 库中的 CLASS 数据集写成 SASHELP.CLASS。
- WORK 库是“临时库”：其中的 SAS 文件在退出 SAS 系统时自动删除。
- 另外，WORK 库中的数据集可以用“一水平名”访问，即可以省略“WORK.”。
- SAS 资源管理器 (Explorer) 管理 SAS 文件。
- 除数据集外，SAS 数据库中还可以包含叫做“SAS **目录簿**”(SAS catalog) 的 SAS 文件，主要用来保存设置、程序代码、图形这样的非规整数据。
- 在 SAS 资源管理器中双击某一数据集进入 VIEWTABLE 数据管理界面。

1.3 SAS/INSIGHT

1.3.1 SAS/INSIGHT 简介

SAS/INSIGHT 简介

- 不需编程的图形操作模块。
- 功能：数据交互输入、数据探索、分布研究、相关分析、图形。
- 强大的**探索性数据分析**能力。
- 探索性数据分析的重要意义。

1.3.2 SAS/INSIGHT 的数据窗口

数据窗口

- 启动: Solutions—Analysis—Interactive Data Analysis.
- 打开 SASUSER.CLASS。
- 数据窗口组成：数据单元格、变量名、变量类型（区间型或名义型）、变量用途、变量个数；观测序号、观测个数、观测绘点符号；数据窗口菜单。
- 新建数据集。用“Define Variables”修改变量属性。从“Data Options”中选回车后的方向。
- 保存数据集。
- 编辑数据集。修改可以体现在分析中但需要保存数据集才能影响到保存在磁盘中的数据集。
- 选中方式：点击变量名、Shift 点击、Control 点击；点击观测号、Shift 点击、Control 点击；选中部分列后用 Control 点击选行，或选中部分行后用 Control 点击选列。
- 删除选定的行或列：主菜单中的“Edit—Delete”。

数据窗口菜单

- Find Next — 显示第一个被选定的观测在窗口第一行的位置。
- Move to First — 把选定的行或列移到最前。
- Move to Last — 把选定的行或列移到最后。
- Sort — 排序。按选定列排序（缺省升序）。没有选定列时出现对话框选择，可以选择降序、按输出格式结果排序等。
- New Observations — 用于快速添加若干个空数据行。
- New Variables — 用于快速添加若干个新变量。
- Define Variables — 设定变量的名字、标签、量测水平、缺省分析用途等。可以在生成新数据集时定义变量, 也可以对已有数据集的变量属性进行修改。
- Fill Values — 用于自动生成一个等差数列变量。
- Extract — 把选定的部分取出到另一个窗口。保存后得到数据子集的数据集。
- Data Options — 本数据窗口的一些设置。

计算新变量

- 主菜单的“Edit—Variables”可以计算新变量加入到数据窗口中。

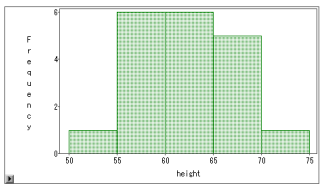
1.3.3 一维数据探索

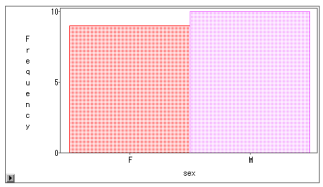
数据探索的图形方法

- 直方图 (Histogram)
- 盒形图 (Boxplot)
- 马赛克图 (Mosaic plot)
- 散点图 (Scatter plot)
- 曲线图 (Curves)
- 散点图矩阵 (Scatter plot matrix)
- 三维散点图 (Rotating plot)

直方图

- 直方图用以反映数据的分布情况，既可以针对连续型数据，也可以针对离散数据。





直方图制作

- 以 SASUSER.CLASS 数据集为例。
- 先选定变量 height 然后点击菜单 “Analyze—Histogram/Bar Charts(Y)”。
- 也可以不预先选定变量而在随后的对话框中选。
- 每个条形是一个组，两侧的值是区间端点，高度代表人数（观测个数，Frequency）。
- 点击条形选定这个组（在数据窗口和其它窗口同时选定）。作一个散点图以说明此点。
- 双击条形可以查看这部分观测。
- 取消选定：单击空白处。
- 改变图形大小、位置和角度：选定边框后拖动一个角或一个边。

设置菜单

- 设置菜单：图形中左下角的向右箭头或右键菜单。

- Ticks: 坐标轴刻度设定；
- Axes: 有没有坐标轴;
- Observations: 是画出所有观测还是只画选定的观测；
- Values: 标出纵坐标的具体值。

保存图形

- 选定图形后用菜单 “File—Save—Graphics”。可保存为:
  - BMP
  - GIF
  - PBM
  - PS
  - TIFF
- 扩展名自己提供。
- 保存在当前目录。

离散数据的直方图（条形图）

- 以 SEX 变量为例。
- 反映每个类的观测数。
- 单击一个条形可以快速选定一个类，包括在数据窗口和其它图形中, 如直方图和散点图中。
- 双击条形查看相应观测。

分组直方图

- 打开 SASUSER.GPA，不选中变量，用 “Analyze—Histogram/Bar Charts(Y)” 菜单弹出对话框，选 GPA 为 Y 变量，SEX 为分组变量。
- 得到男女生分别的直方图。
- 要达到同样效果，也可以：
  - 作 SEX 的条形图，选中女生的条形 (值为 F)。
  - 作 GPA 的条形图，然后在图形窗口菜单中去掉 “Observation” 的勾选。这样得到女生的 GPA 分数的直方图。
  - 类似得到男生的 GPA 分数的直方图。

盒形图 (Boxplot)

- 盒形图是另外一种表现连续型变量取值分布的图形，比直方图给出的信息更为精简。
- 直方图是分布密度的一种估计，可以直观看出分布密度形状，以及各不同取值区间的比例；盒形图直观地画出各重要分位点及边缘值，也能对分布大致形状有概要的了解。
- 盒形图较为简单，所以可以同时绘制多个盒形图以比较不同变量或不同组的分布。

盒形图的解释

- 演示：以 SASUSER.CLASS 中 height 和 SASUSER.GPA 中的 GPA 的盒形图为例。调用菜单 “Analyze—Box Plot/Mosaic Plot”。
- 纵轴代表身高的取值范围；
- 中间的粗线为身高分布的中位数的位置；概括分布位置信息；
- 盒子上边线是分布的四分之三分位数；
- 盒子下边线是分布的四分之一分位数；
- 盒子上下边线包含了分布的中间 50% 的观测，这个长度叫做 “四分位间距”(IQR)。可以反映数据分布分散程度。
- 从盒子边线向外画了两条线叫做触须线, 最长可以延伸到四分位间距的 1.5 倍, 但是如果已经到了数据的最小值或最大值处就不再延伸。
- 触须线外如果有单独的点则怀疑是分布的异常值（按照正态分布的经验）。

盒形图的解释 (续)

- 从盒形图可以看出数据的偏斜情况和重尾情况：
  - 上触须线长而下触须线短，分布右偏；
  - 下触须线长而上触须线短，分布左偏；
  - 任何触须线外有较多点则有重尾现象。
- 例如，GPA 分数有左偏和重尾问题。

- 用盒形图菜单中的 “Means” 选项可以在盒形图上加画一个菱形, 菱形的中间代表分布的平均值, 菱形端点到中间距离为两倍标准差。如果是变量服从正态分布, 菱形上下端点之间应该包含大约 95% 的观测。
- 单击或双击盒形图的一部分照样可以选定观测或查看观测。

盒形图的图形菜单

- Ticks: 控制坐标轴画法；
- Axes: 是否画坐标轴；
- Observations: 是画出所有观测还是只画选定的观测；
- Means: 在盒形图上加画一个菱形, 菱形的中间代表分布的平均值, 菱形端点到中间距离为两倍标准差。
- Serifs: 在触须线两端画小横线；
- Values: 标出各关键点的值；
- Reference Lines: 画坐标轴主要刻度的参考线。
- Marker Sizes: 散点的大小。

并排盒形图

- 盒形图较为简单，所以可以同时绘制多个盒形图以比较不同变量或不同组的分布。
- 以 SASUSER.GPA 数据集为例。
- 不选定任何变量调用 “Analyze—Box Plot/Mosaic Plot” 弹出对话框。
- 选定 “HSM、HSS、HSE” 三个变量为 Y 变量，可以画出这三门成绩的并排的盒形图。这样可以直观比较这三个变量的分布。
- 选 GPA 为 Y 变量，选 SEX 为 X 变量，可以并排画出女生和男生的 GPA 分数的盒形图。
- 即使有十几个组或十几个变量（如果可比）也能用这种方法粗略地概括其分布。

马赛克图

- 马赛克图 (Mosaic Plot) 描述离散变量分布。
- 用菜单 “Analyze—Box Plot/Mosaic Plot”。
- 如: SASUSER.GPA 中 SEX 的马赛克图。
- 如: SASUSER.CLASS 中 SEX 和 AGE 的交叉分组的马赛克图。选 SEX 为 Y 变量，AGE 为 X 变量。
- 单击某一组选定这一组并显示这一组的观测个数和占总观测数的百分比。这是选定交叉分组中的某一组的简便办法。

1.3.4 二维数据探索

二维数据探索

- 曲线图;
- 散点图;
- 散点图矩阵。

曲线图

- 以 SASUSER.AIR 为例。
- 曲线图描述一个或几个变量随另一个变量（如时间）变化而变化的情况。
- 用 “Analyze—Line Plot” 菜单绘制。
- 如: CO 对 DATETIME 的曲线图。
- 图形窗口菜单中的 “Observations” 这里用来选择曲线上是否显示中间实际存在观测的散点，不再用来选择是否使用所有观测绘图。
- 单击曲线上一个点选定此点。拖动出一个矩形可以选定此范围内的观测点。
- 双击曲线上一个点查看此点对应观测。
- 在数据窗口中把 HOUR 的用途改为 “Label”(标签)，重新绘制 CO 对 DATETIME 的曲线图，单击曲线上点时看到此观测点的时间。可以看出，CO 在中午最低，上下班高峰时最高。

- 以 CO、O3、SO2、NO、DUST、WIND 为 Y 变量，DATETIME 为 X 变量画曲线图。
- 不同颜色代表不同总坐标变量。
- 选定一个名字或点击一条曲线可以选中一条曲线加亮显示。
- 可以看出风速增大对污染有抑制作用。可以用散点图来验证。

散点图

- 散点图可以用来描述两个变量之间的相互关系。
- 先选中纵坐标变量再选中横坐标变量，然后用菜单 “Analyze—Scatter Plot” 作散点图；
- 或者不选中任何变量直接调用菜单 “Analyze—Scatter Plot” 然后在弹出的对话框中指定纵坐标变量和横坐标变量。
- 演示：SASUSER.AIR 中 CO 对 WIND 的散点图（先选中变量）；SASUSER.CLASS 中 WEIGHT 对 HEIGHT 的散点图（不先选变量）。
- CO 对 WIND 的图中看出有非线性的相关关系。
- WEIGHT 对 HEIGHT 的图中看出有线性相关关系。

对散点图的操作

- 点击一个点可以显示这个点的序号并在数据窗口中选中这个观测。
- 如果在作散点图的对话框中选了 NAME 为 Label 变量，则点击一个点可以显示该点对应的学生名字。
- 双击一个点可以查看对应观测。
- 可以用 Shift 或 Ctrl 点击选中更多的点。
- 在散点图中拖出一个矩形可以选中其中的点；拖动这个选择框可以改变选中的范围，这种技术称为 “刷亮 (Brushing)”。
- 双击选择框可以查看选中的观测。
- “投出” 选择框可以自动往复移动。
- 演示：在体重对身高散点图中刷亮并观察数据窗口和性别分布条形图中的变化。

散点图矩阵

- 散点图矩阵画出多个变量两两间的散点图以考察多变量关系。
- 只要选中多个变量后用菜单 “Analyze—Scatter Plot” 就可以作散点图矩阵。未选中变量时用菜单 “Analyze—Scatter Plot” 调出对话框然后选多个 Y 变量和多个 X 变量可以作出散点图矩阵。
- 用刷亮的方法可以观察多个变量的互相影响。

CLASS 数据集散点图矩阵分析

- 演示: SASUSER.CLASS 中体重、身高、年龄；
- 三者有正向的线性相关关系。
- 体重与身高关系较明显，年龄的影响稍弱。
- 年龄取值是离散的。
- 演示：刷亮观察。

GPA 数据集散点图矩阵分析

- 演示：SASUSER.GPA 中各分数。
- HSM、HSS、HSE 取值离散。
- 各分数基本呈正向的相关。
- SATM 和 SATV 的相关较明显。
- 演示：刷亮观察。

AIR 数据集散点图矩阵分析

- 演示: SASUSER.AIR 中各测量值。
- CO、NO、DUST 同方向且关系为线性相关，我们称这三者为主要污染；WIND 与这三个主要污染反方向变化且关系为非线性。
- O3 的变量与主要污染有关但是可以看出有一个外在因素的影响；风可以增加 O3 污染。
- SO2 也受外在因素的影响，风可以增加 SO2 污染。
- 演示：刷亮观察。

1.3.5 三维数据探索

旋转图 (三维散点图)

- SAS/INSIGHT 提供了绘制旋转图 (Rotating Plot) 的功能。旋转图实际是一个三维坐标的散点图，可以拖动旋转。
- 演示: SASUSER.CLUS 的旋转图。
- 可以拖动旋转或用图形窗口中的图标控制。
- 演示: SASUSER.CLASS 中身高、体重、年龄的旋转图。
- 演示: SASUSER.AIR 中 CO、NO、DUST 的旋转图。

曲面图

- SAS/INSIGHT 可以绘制曲面图。
- 设数据集中每一行有曲面上点的 x, y 坐标和相应的 z 坐标，数据集中的 x, y 坐标构成了一个网格。
- 以 SAMP.DNORM2 数据集为例。选 “分析—旋转图” 菜单，把变量 X, Y, Z 选入相应的变量框内。点击 “输出” 按钮，在弹出的对话框中勾选 “拟合曲面”。按 “确定” 后出现图形，适当旋转，在图形的局部菜单去掉 “观测” 的勾选，并选 “绘图模式” 为 “渐变颜色”。
- 如果需要加密网格，可以在初始的对话框中按 “方法” 按钮加大 “水平网格大小” 和 “垂直网格大小”。

1.3.6 图形的调整

图形的调整

- 可以调整坐标轴画法，点的大小、符号、颜色，线型、粗细、颜色，等等。
- 用 “Edit—Window—Tools” 调出图形工具窗口。
- 演示: 用 SASUSER.IRIS。
  - species: 三种类型 Virginica, Versicolor, Setosa。用 Bar Plot 显示。

  - 四个测量值 petalwid(花瓣宽), petallen(花瓣长), sepalwid(花萼宽), sepallen(花萼长)。作散点图。
  - 给不同种类使用不同颜色和不同符号。可以自动赋值或在条形图中选定每类后手工赋值。
- 演示: SASUSER.AIR 的曲线图中改变线型、颜色、粗细。

1.3.7 分布研究

分布研究

- 分布研究是探索性数据分析的重要方面。
- 关心：取值类型
  - 连续型
  - 名义型
  - 有序型
- 对于离散型，分布包括：
  - 取哪些值；
  - 各个值出现的次数和比例。
- 对于连续型，分布包括
  - 可取值的集合；
  - 位置信息，如均值、中位数、众数；
  - 分散程度信息，如标准差、四分位间距、极差、最小值和最大值；
  - 分布形状是否偏斜；
  - 分布是否重尾；
  - 更详细的分布密度形状。

SAS/INSIGHT 的一维分布研究功能

- 直方图、盒形图、密度估计图、分布函数估计图；
- 统计量表，包括各阶矩的估计，重要分位数的估计；
- 分布的拟合优度检验。
- 离散分布的条形图、马赛克图、累计分布表。

数字特征

- 演示: SASUSER.CLASS 中身高分布，用菜单 “Analyze — Distribution(Y)”。
- 详细解释各统计量和图形，定义和作用。

统计量定义

- N — 观测个数 n;
- Mean — 均值 $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$;
- Sum — 总和 $\sum_{i=1}^{n} y_i$;
- Std Dev — 标准差 $s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2}$;
- Variance — 方差 $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2$;
- Skewness — 偏度 $\frac{n}{(n-1)(n-2)}\sum\left(\frac{y_i-\bar{y}}{s}\right)^3$;
- Kurtosis — 峰度 $\frac{n(n+1)}{(n-1)(n-2)(n-3)}\frac{\sum(y_i-\bar{y})^4}{s^4} - \frac{3(n-1)^2}{(n-2)(n-3)}$;
- CV — 变异系数 $\frac{s}{\bar{y}} \cdot 100$;
- Std Mean — 均值的标准误差 $s/\sqrt{n}$。

偏度和峰度

- 理论偏度：
$$\text{偏度} = \frac{E(X-\mu)^3}{\sigma^3}$$
其中 μ 为均值，σ 为标准差。
- 偏度的正负代表分布偏斜的方向。
- 理论峰度:
$$\text{峰度} = \frac{E(X-\mu)^4}{\sigma^4} - 3$$
- 正态分布的峰度为 0。
- 峰度的数值代表重尾的情况。正的峰度表示重尾。
- 样本偏度和样本峰度有渐近的正态分布。

标准误差

- 标准误差是统计软件结果中重要的指标。
- 以正态分布 $X \sim \mathrm{N}(\mu, \sigma^2)$ 中 μ 的估计为例，$X_1, X_2, \ldots, X_n$ 是独立同分布样本。
- μ 的估计为 $\hat{\mu} = \frac{1}{n}\sum_i X_i$。
- $\hat{\mu}$ 也是一个随机变量，分布为 $\mathrm{N}(\mu, \sigma^2/n)$，称为抽样分布 (Sampling distribution)。
- 作为无偏估计量，$\hat{\mu}$ 的抽样分布越集中，说明估计越精确；抽样分布的标准差大小可以反映估计的精确程度。
- 抽样分布的标准差的估计叫做估计量的**“标准误差”**。
- 比如，估计量 $\hat{\mu}$ 的标准误差为 $\hat{\sigma}/\sqrt{n}$。
- 其它估计量也可以类似地估计标准误差以反映估计的精度。
- 估计量抽样分布经常是正态分布或渐近正态分布，这时估计量加减两倍标准误差可以作为参数的近似 95% 置信区间，估计量除以标准误差可以作为检验参数是否为零的 Z 统计量或 t 统计量。

分位数表

- Med: 中位数；
- Min, Max: 最小值和最大值；
- Q1, Q3: 四分之一和四分之三分位数；
- Range: 极差，为最大值减去最小值的差；
- Q3-Q1: 四分位间距，分布分散程度的指标；
- Mode: 众数。

添加统计表

- 分布研究窗口中 Tables、Graphs、Curves 菜单被激活。
- Tables 菜单可以选加一些统计表或统计结果。

- Basic Confidence Intervals—计算均值、标准差、方差的各种置信度的置信区间（假定正态分布）；
- Tests for Location—检验均值是否某一特定值；
- Frequency Counts—频数表，包括每一观测值的频数（出现次数）、百分比和累计频数。
- Robust Measures of Scale—变量分散程度的五种稳健估计，及稳健的标准差估计。
- Tests for Normality—正态性的检验；
- Trimmed/Winsorized Mean—截尾的或削平的均值，为稳健的均值估计。

QQ 图

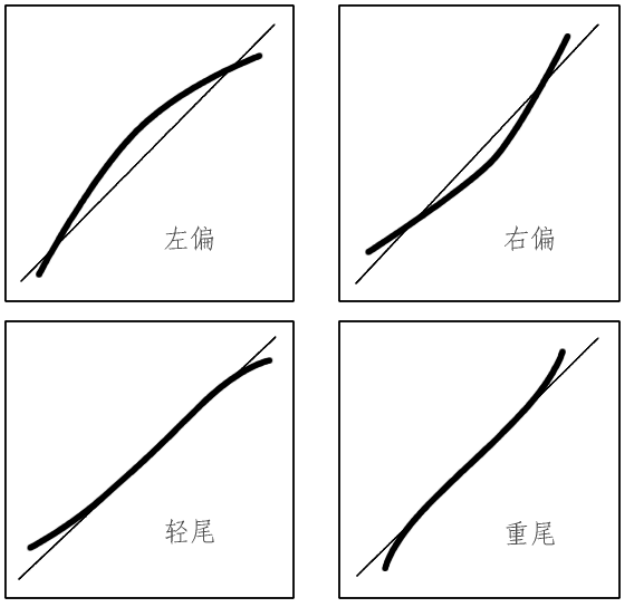
- 在 Graphs 菜单中选“QQ Plot”。常用的是关于正态分布的 QQ 图。
- 设有 n 个观测 y_1, y_2, ..., y_n, 并已从小到大排列, 则 y_i 是总体的 i/n 分位数的估计;
- 设 x_i 是标准正态分布的 i/n 分位数, 则在样本来自正态 $N(\mu, \sigma^2)$ 的情况下, 记 $\mathrm{N}(\mu, \sigma^2)$ 的分布函数为 $F(x)$， 有 $y_i \approx F^{-1}(\frac{i}{n})$;
- $F(y_i) = \Phi(\frac{y_i - \mu}{\sigma}) \approx \frac{i}{n}$;
- $\frac{y_i - \mu}{\sigma} \approx \Phi^{-1}(\frac{i}{n}) = x_i$;
- $y_i \approx \mu + \sigma x_i$。
- 用 $(x_i, y_i)(i = 1, 2, \ldots, n)$ 作为坐标画散点图应该近似呈现为截距 μ、斜率 σ 的一条直线。

QQ 图—连续型修正

- 上述的近似有一个小缺点: y_1 是观测到的最小值, 对应于 $1/n$ 分位数, y_n 是观测到的最大值, 却对应于 n / n = 100% 分位数, 对最小和最大值的处理不对称;
- 这相当于说总体分布不能超过 y_n, 是不合理的。
- 所以在实际画正态 QQ 图时, y_i 不是对应于标准正态的 i/n 分位数而是对应于其 $(i - 0.375)/(n + 0.25)$ 分位数, 这种做法叫做连续性修正。

- 这时, y_1 对应于 $\frac{0.625}{n+0.25}$ 分位数而 y_n 对应于 $1-\frac{0.625}{n+0.25}$ 分位数, 两边各留了 $\frac{0.625}{n+0.25}$。

QQ 图的典型特征



分布密度和分布函数估计

- 包括参数方法和非参数方法。
- 以 SASUSER.CLASS 中 HEIGHT 为例。
- 菜单 “Curves—Parametric Density” 可在直方图上添加正态分布密度估计。也可以用对数正态分布、指数分布或威布尔分布。
- 可以手动调节参数。
- 核估计图: “Curves—Kernel Density” 菜单。
- 调整平滑参数查看效果。
- 进行 SASUSER.GPA 中 GPA 的密度估计。
- 正态性检验：“Curves—Test for Distribution” 菜单。
- 经验分布函数: “Curves—Empirical CDF” 菜单。
- 参数方法分布函数估计：“Curves—Parametric CDF” 菜单。

第二章　SAS 语言与数据管理

2.1 SAS 语言构成

2.1.1 SAS 语句

介绍

- SAS 系统强大的数据管理能力、计算能力、分析能力依赖于作为其基础的 SAS 语言。
- SAS 语言是一个专用的数据管理与分析语言，它的数据管理功能类似于数据库语言 (如 FoxPro)，但又添加了一般高级程序设计语言的许多成分 (如分支、循环、数组)，以及专用于数据管理、统计计算的函数。
- SAS 系统的数据管理、报表、图形、统计分析等功能都可以用 SAS 语言程序来调用，只要指定要完成的任务就可以由 SAS 系统按照预先设计好的程序去执行，所以 SAS 语言和 FoxPro 等属于第四代语言。

本章内容

- SAS 语言的基本成分与规则;
- SAS 语言如何用来管理数据;
- SAS 语言作为一个统计计算语言的用法。

SAS 语句

- SAS 语言程序由**数据步**和**过程步**组成。
- 数据步用来生成数据集、计算、整理数据;
- 过程步用来对数据进行分析、报告。
- SAS 语言的基本单位是语句, 每个 SAS 语句一般由一个**关键字** (如 DATA, PROC, INPUT, CARDS, BY) 开头, 包含 SAS 名字、特殊字符、运算符等, 以分号结束。
- SAS 关键字是用于 SAS 语句开头的特殊单词, SAS 语句除了赋值、累加、注释、空语句以外都以关键字开头。

SAS 名字

- SAS **名字**在 SAS 程序中标识各种 SAS 成分, 如变量、数据集、数据库, 等等。SAS 名字由若干个字母、数字、下划线组成, 第一个字符必须是字母或下划线。SAS 关键字和 SAS 名字都不分大小写。变量名和数据集名不超过 32 个字符，数据库名 (libref) 不超过 8 个字符。

2.1.2 SAS 表达式

SAS 表达式

- SAS 数据步程序中的计算用**表达式**完成。
- 表达式把**常量**、**变量**、**函数调用**用**运算符**、括号连接起来得到一个计算结果。

SAS 常量

- SAS 常量主要有数值型、字符型两种, 并且还提供了用于表达日期、时间的数据类型。例如
  - 数值型：12, −7.5, 2.5E−10
  - 字符型：'Beijing', "Li Ming", " 李明"
- 数值型常数可以用整数、定点实数、科学计数法实数表示。
- 字符型常数为两边用单撇号或两边用双撇号包围的若干字符。当字符串内容有单撇号时，可以用双撇号界定，当字符串内容有双撇号时，可以用单撇号界定。另外，在字符串中，重复的单撇号可以作为一个单撇号内容，如'Tom" cat' 内容是“Tom's cat”。

SAS 日期和时间

- SAS 日期型常量如：'13JUL1998'd;
- 日期型常数是在表示日期的字符串后加一个字母 d(大小写均可), 中间没有空格。日期值保存为从 1960 年 1 月 1 日后经过的天数。
- 时间型常量如：'14:20:15.32't, '14:20't(没有秒);
- 时间型常数是在表示时间的字符串后加一个字母 t。保存为秒数。
- 日期时间型常量如: '13JUL1998:14:20:15.32'dt;

- 日期时间型常数在表示日期时间的字符串后加字母 dt。保存为从 1960 年 1 月 1 日 0 时后经过的秒数。

缺失值

- 数据处理时经常遇到缺失值，如：
  - 数据遗失；
  - 被访者拒绝回答；
  - 节假日无数据。
- SAS 程序中用一个单独的小数点来表示数值型缺失值常量。
- 对于字符型数据，用只有一个空格的字符串' ' 表示缺失值。
- 变量 X 是否数值型缺失值可以用 X=. 来判断，是否字符型缺失值可以用 X=' ' 来判断。
- 数值型缺失值在排序和比较时排在最小端。
- 数值型缺失值参与计算的结果也是缺失的。

SAS 变量

- SAS 变量的基本类型有两种：**数值型**和**字符型**。
- 日期、时间等变量存为数值型。
- **存储长度**：SAS 的数值型变量可以存储任意整数、定点实数、浮点实数, 一般不关心其区别。缺省用 8 个字符存储, 精度有 16 到 17 位有效数字。
- SAS 在读入字符型变量时缺省的长度是 8 个字符, 但是如果在 INPUT 语句中输入字符型变量时指定了长度则不受此限制，最多可存 32767 个字符。
- 直接用赋值语句定义的字符型变量的缺省长度是第一次赋值的字符串长度。
- 可以用 LENGTH 语句直接指定变量长度, LENGTH 语句一般应出现在变量定义之前, 格式为:

  LENGTH 字符型变量名 $ 长度;

SAS 算数运算符

- SAS 运算符包括算术、比较、逻辑等运算符。
- 算术运算符为 +, −, *, /, **, 运算优先级按通常的优先规则。其中两个星号表示乘方运算。

SAS 比较运算符

- 比较运算符用于比较值大小, 包括

=	^=	>	<	>=	<=	IN
EQ	NE	GT	LT	GE	LE	

- 比较运算符得到“真”或“假”的结果，真用 1 表示，假用 0 表示。
- 两个字符型的比较是把短的一个右边补空格到两个长度相同后比较。
- 在 =, >, <, >=, <= 后面加一个冒号变成如 =: 这样则可以只比较与短的一个等长的部分。
- 运算符 IN 是一个 SAS 特有的比较运算符, 用来检查某个变量的取值是否在一个给定列表中, 比如

  ```
  prov in (' 北京', ' 天津', ' 上海', ' 重庆')
  ```

  可以判断变量 prov 的取值是否为四个直辖市之一。
- 列表为圆括号界定的用逗号或空格分开的若干常量。
- 也可以使用 NOT IN 表示“不属于”。

SAS 逻辑运算符

- 逻辑运算符用来连接比较得到的结果以构成复杂的条件, 有三种逻辑运算符:

  & (AND)　| (OR)　^ (NOT)

- 例如

  (salary >= 1000) AND (salary < 2000)
  　表示工资收入在 1000 − 2000 之间 (不含 2000)

  (age <= 3) OR (sex = ’女’)
  　表示三岁以下 (含三岁) 的婴儿及妇女

  NOT ((salary >= 1000) AND (salary < 2000))
  　表示工资收入**不**在 1000 − 2000 之间

- 复杂的逻辑表达式最好用括号表示其运算优先级以免误记优先规则，这样也有利于阅读程序。

其它运算符

- || (两个连续的 | 号)：连接两个字符串；
- <>：用于取两个运算值中较大一个 (比如 3 <> 5 结果为 5)；
- ><：用于取两个运算值中较小一个 (比如 3 >< 5 结果为 3)。

2.1.3 SAS 程序规则

SAS 程序规则

- SAS 程序由语句构成。
- 每个语句以分号结尾 (最常见的 SAS 编程错误就是丢失分号)。
- 一个语句可以写到多行 (不需任何续行标志), 也可以在一行连续写几个语句。
- SAS 语言中只要允许用一个空格的地方就可以加入任意多个空白 (空格、制表符、回车)。
- 允许用空格的地方是名字周围、运算符周围。

SAS 程序规则示例

- 例: 程序

```
proc print
        data=c9501;
    by              avg;

run;
```

- 和程序

```
proc print data=c9501;by avg;run;
```

是等价的。

注释

- 在 SAS 程序中可以加入注释, 注释使用 C 语言语法, 用/* 和 */在两端界定注释, 这种注释可以出现在任何允许加入空格的位置, 可以占多行。
- 另一种注释是把以星号开头的行作为注释。
- 我们一般只把注释单独占一行或若干行, 不把注释与程序代码放在同一行。
- 注释的另一个作用是把某些代码暂时屏蔽使其不能运行。

数据步和过程步

- SAS 程序包括数据步和过程步两种结构, 每一个步是一段相对完整的可以单独运行的程序。
- 数据步用来生成、整理数据和自编程计算;
- 过程步调用 SAS 已编好的处理过程对数据进行处理。
- 自己用 SAS 编程序进行计算主要在数据步中进行。

SAS 数据步语句

- SAS 数据步以 DATA 语句开头, 以 RUN 语句结尾。
- DATA 步中可以使用 INPUT, CARDS, INFILE, SET, MERGE 等语句指定数据来源输入数据, 也可以用赋值、分支、循环等编程结构直接生成数据或对输入的数据进行修改。

2.1.4 全局语句

全局语句

- SAS 程序一般需要组成数据步或过程步来运行。
- 全局语句是不需要用在数据步和过程步内部的语句。
- 全局语句与一般语句不同:
  - 一般语句必须用在数据步或过程步内, 作为数据步或过程步的一部分;

  - 而全局语句则既可以用在数据步和过程步内, 又可以单独使用 (在数据步、过程步外部)。
  - 全局语句的作用有持续性, 全局语句的效果将持续到退出 SAS 系统或用另一个同样的全局语句来修改它。

OPTIONS 语句

- 全局语句 OPTIONS 语句可以规定系统运行的一些选择项。如

```
options formdlim='*' nonumber nodate
        linesize=64 pagesize=60;
```

- 各选项意义为:
  - FORMDLIM 选项指定两个输出之间的分隔线使用的字符。
  - NONUMBER 表示输出不显示页号 (改用 NUMBER 则规定显示页号)。
  - NODATE 表示不在每页显示运行日期和时间 (改用 DATE 则显示)。
  - LINESIZE = 64 规定输出每行最宽不超过 64 个字符 (这是允许的最小行宽)。指定较小的行宽可以使输出比较紧凑。
  - PAGESIZE=60 规定输出每页为 60 行, 不足时用空行补齐。
  - 用 NOCENTER 可以指定输出不需要居中对齐，缺省是 CENTER(居中)。

2.2 SAS 用作一般高级语言

SAS 用作一般高级语言

- SAS 是一种专用的数据处理、统计计算语言, 但是它也包含一般的高级语言编程能力并扩充了许多数学、统计等方面的函数。
- 我们先介绍 SAS 语言用来进行一般编程计算的功能。
- SAS 数据步的数据输入、整理功能很强, 希望进行复杂的数据管理的读者可以根据本节和下节的内容用 SAS 实现强大的数据管理功能。
- 希望实现自己的统计计算算法的读者可以用 SAS 数据步编程, 但是我们建议使用 S 来编算法程序, 因为用 S 编程更方便。
- SAS 语言的编程计算能力主要由 SAS 数据步提供 (另外 SAS 还提供了一个 SAS / IML 模块可以进行向量、矩阵运算, 读者有兴趣可以自己学习)。

SAS 数据步格式

- SAS 数据步以 DATA 语句开始，以 RUN 语句结尾。
- DATA 语句以关键字 DATA 开头, 后面给出一个数据集名, 这是本数据步要生成的数据集的名字, 例如:

```
data tmp1;
```

- 也可以省略数据集名, 这时 SAS 自动生成一个临时数据集名。
- 还可以使用特殊名字_NULL_, 表示本数据步不生成数据集。
- 所有例子都是在数据步中运行的。

2.2.1 赋值语句

赋值语句

- 在 SAS 中用赋值语句计算一个值并存放到变量中。格式为

  变量名 = 表达式;

- 例如：

```
avg = (math + chinese/120*100)/2;
isfem = (sex=' 女');
y=sin(x)**2;
newv = .;
```

- 注意需要在数据步中使用。
- 表达式中的数值型变量如果不存在则作为缺失值。

2.2.2 输出语句

输出语句

- SAS 数据步的输出一般是数据集, 用赋值语句计算的结果会自动写入数据集。
- SAS 也提供了一个 PUT 语句, 可以象其它语言程序的 PRINT, WRITE(*, *), printf 等语句一样立即显示输出结果。
- 主要用于调试目的。
- PUT 语句在关键字 PUT 后面列出要输出的各项, 每一项可以是变量名或字符串, 不能为数值常量或表达式, 各项之间用空格分开。
- PUT 语句的输出结果显示在 LOG 窗口。

输出语句—例子

- 程序

```
data;
  x=0.5;
  y=sin(x);
  put "sin(" x ")=" y;
run;
```

- 将在 Log 窗口显示结果

```
sin(0.5 )=0.4794255386
```

带变量名的输出

- 在 PUT 语句中使用“变量名＝”来指定输出项可以显示带有变量名的输出结果, 如：

```
data;
  x=0.5;
  y=sin(x);
  put x= y=;
run;
```

- 将显示结果

```
X=0.5 Y=0.4794255386
```

在指定列输出

- PUT 语句的输出项还可以指定具体列位置, 比如, 下面的 PUT 语句指定把 X 数值显示在第 10–20 列, 把 Y 数值显示在第 30–40 列, 并保留 6 位小数:

```
  put x 10-20 .6 y 30-40 .6;
```

- 在指定的列范围内, 数值型数据靠右对齐, 字符型数据靠左对齐。
- 要保留的小数位数写在一个小数点后面, 如果变量为整数值或者字符型则不必指定小数位数。

域宽. 精度格式

- PUT 语句还可以使用“域宽. 精度”方式指定输出的宽度和精度, 例如,

```
  put x 20.8 y 20.8;
```

- 使 X 占用第 1 — 20 列, 8 位小数, 右对齐；Y 占用第 21 — 40 列, 8 位小数, 右对齐。

- 其中 20.8 是多种 SAS 输出格式中最常见的一种, 用于输出数值型数据, 小数点前面为输出宽度, 小数点后面为输出精度 (小数位数)。

- 对于字符型变量, 要指定其输出宽度可以用“$ 宽度.”的格式, 如“$10.”指定字符型变量输出宽度为 10 位。

- 输出占不满指定宽度时, 数值型数据向右对齐, 字符型数据向左对齐。

- 字符型常量可以用于 PUT 语句中, 但不能规定占用的列位置或者宽度。

- 变量名后面还可以使用其它输出格式，如 COMMA20.8 等。详见第 3 节。

列指针和换行

- 用@ 和列号可以使 PUT 语句的输出位置直接跳到指定的列，如

```
put @1 name  @ 11 age @15 height;
```

- 在 PUT 语句中用'/' 可以使输出指针直接跳到下一行行首开始输出。如

```
put @1 name  @ 11 age / height weight;
```

- 如果希望 PUT 语句的输出不产生换行, 使下一个 PUT 的结果可以显示在同一行, 只要在 PUT 语句结尾处加一个@ 符, 如:

```
put  i  @;
```

输出改向

- PUT 语句的输出结果缺省情况下被送到运行记录窗口。
- 在 PUT 语句之前用 FILE 语句可以改变 PUT 语句的输出目的地。
- 如:

```
file print;
```

  可以把 PUT 语句的输出转向到输出窗口。

- 又如:

```
file 'tmp.out';
```

  把后续的 PUT 语句输出转向到当前工作目录下的文件“tmp.out”中, 生成输出文件 tmp.out。

- 注意当前工作目录在 SAS 状态栏的右方显示, 双击可以更改。
- 文件名也可以指定全路径。
- 在 FILE 语句末尾加上 DLM=',' 选项可以自动在输出的各项之间加上逗号分隔符。

2.2.3 分支结构

分支结构

- IF 条件 THEN 语句;
- 如

```
IF  x > 0  THEN  PUT  'X 为正数';
```

- 严格说来，SAS 中任何非 0 并且非缺失的数值型值都是真值，只有 0 和数值型缺失值是假值。

复合语句

- 使用 “DO;” 和 “END;” 包围起来的若干个语句可以看作是一个语句, 叫做**复合语句**。
- 例:

```
IF  x>0  THEN  DO;
    PUT  'X 为正数';
    x = 2*x;
    PUT  x=;
    END;
```

IF-ELSE 结构

- IF 条件 THEN 语句;
  ELSE 语句;
- 如

```
IF  x>=0  THEN  x=2*x;
ELSE  x = -x;
```

- 注意 IF-ELSE 结构不需要 END 或 ENDIF 结尾。

IF-ELSE IF-ELSE 结构

- IF 条件 1 THEN 语句 1;
  ELSE IF 条件 2 THEN 语句 2;
  ……
  ELSE 语句;
- 如

```
IF x < 0 THEN y=-1;
ELSE IF x=0 THEN y=0;
ELSE y=1;
```

SELECT 语句

- SELECT 语句也提供多重分支功能。有两种用法。

SELECT (选择表达式);
  WHEN(值列表) 语句;
  WHEN(值列表) 语句;
  ……
  OTHERWISE 语句;
END;

SELECT 例

```
SELECT(month);
  WHEN('Feb', 'Mar', 'Apr')  put  ' 春天';
  WHEN('May', 'Jun', 'Jul')  put  ' 夏天';
  OTHERWISE  put  ' 秋天或冬天';
END;
```

SELECT 第二种用法

- 类似于其它语言的 IF-ELSEIF-ELSE 结构。

SELECT;
  WHEN(条件) 语句;
  WHEN(条件) 语句;
  ……
  OTHERWISE 语句;
END;

SELECT 第二种用法: 例子

```
SELECT;
    WHEN(age<=12)  put  ' 少年';
    WHEN(age<35)  put  ' 青年';
```

```
    OTHERWISE  put  ' 中老年';
END;
```

注意每一个 WHEN 的条件都是在上面所有的 WHEN 的条件都不成立时才判断的。

2.2.4 循环结构

循环结构

- SAS 的循环结构包括
  - 计数循环

    DO 计数变量 = 起始值 TO 结束值 BY 步长;
    循环体语句……;
    END;

- – 当型循环

    DO WHILE (循环继续条件);
    循环体语句……;
    END;

  - 直到型循环

    DO UNTIL (循环退出条件);
    循环体语句……;
    END;

计数循环

```
data;
  DO  i = 1  TO  20  BY  2;
    j = i**3;
    put  i  3.  j  5.;
```

```
  END;
run;
```

可以输出一个 1, 3, 5, 7, ⋯, 19 的立方表。

计数循环 (逆序)

```
data;
  DO  i = 19  TO  1  BY  -2;
    j = i**3;
    put  i  3.  j  5.;
  END;
run;
```

可以输出一个 19, 17, ⋯, 1 的立方表。

列数循环

```
data;
  do mon='Jan', 'Feb', 'Mar';
     put mon $4. ' is spring.';
  end;
run;
```

可以对指定的常数值循环。

LEAVE 和 CONTINUE

- 在循环体中可以用 LEAVE 语句跳出循环, 相当于 C 语言的 break 语句。
- 在循环体内用 CONTINUE 语句可以立即结束本轮循环并转入下一轮循环的判断与执行。

当型循环

- 当型循环在条件成立时反复执行。
- 先判断条件，如果一开始条件就不满足则一次也不执行。

- 例: 判断 1333333 是不是素数:

```
data;
  x=1333333;
  i=3;
  DO  WHILE (mod(x,i) ^= 0);
    i=i+2;
  END;
  if i<x then put x ' 不是素数';
  else  put  x  ' 是素数';
run;
```

  其中 mod(x,i) 表示 x 除以 i 的余数。

- 注意，计数循环是当型的。

直到型循环

- 直到型循环一直重复直到循环退出条件满足为止。
- 至少执行一次。
- 例:

```
data;
  n=0;
  do until (n>=5);
     n+1;
     put n=;
  end;
run;
```

- `n+1` 这种写法叫做**累加**语句，等价于 `n=n+1`。

2.2.5 数组

数组

- SAS 可以把一组同为数值型或同为字符型的变量合在一起, 使用同一个名字称呼, 用下标来区分。
- 这与通常的程序设计语言中的数组不同, 通常的程序设计语言中数组元素没有对应的变量, 而 SAS 数组每个元素都有自己的变量。
- SAS 提供了“临时”数组，和其他语言中的数组相同，不对应数据集的变量。

数值型数组

ARRAY 数组名 (维数说明) 数组元素名列表 (初始值表);

- 例:

```
ARRAY  tests(3)  math chinese english  (0, 0, 0);
```

- 初始值表可简写为 (3*0) 这样的形式。
- 初始值表是可选的。如果某变量对应的数组元素有初始值，则此变量的值不会被数据步隐含循环自动赋缺失值（关于数据步隐含循环详见第 3 节）。
- 可以用带上下界的下标，如:

```
ARRAY  sales(95:97)  yr95-yr97 ;
```

- 可以使用星号作为数组维数说明，这时数组元素个数由数组元素列表确定。如：

```
ARRAY  tests(*)  math chinese english  (0, 0, 0);
```

对这样的数组可以用函数 DIM(数组名) 来获得其长度。

- ARRAY 语句中数组元素名表可省略。这时
- （1）数组元素自动对应以数组名为前缀的变量名。

- （2）如果当前没有这些变量，自动生成这些变量。
- 比如，数组说明为

```
ARRAY  sales(3);
```

- 如果当前数据集中有名为 sales1, sales2, sales3 的三个变量，则数组 sales 的三个元素与这三个变量对应；
- 如果当前数据集中没有这三个变量，以上数组声明自动生成名为 sales1, sales2, sales3 的三个变量并与之对应。

例

- 设数据中用 999 表示缺失，需要把每个数值型变量的 999 变成缺失值。

```
data one;
 input a $ b c d e f g;
 cards;
 999  23 17581 0.0023 126 85 06 13
 joe  54 34634 0.0018 165 30 15 12
 bill 36 70451 0.0020 134 62 09 14
 mary 999 52740 0.0017 148 59 999 16
 bob  47 999   0.0016 153 999 05 999
 jack 62 83598 0.0019 142 76 12 18
 ;
run;
```

```
data two;
  set one;
  array x(6)  b c d e f g;
  do i = 1 to 6;
    if x(i) = 999 then x(i) = .;
  end;
  drop i;
run;
```

```
proc print data = two;
run;
```

- array 语句中的变量列表可以用_NUMERIC_ 代表所有数值型变量，如：

```
data three;
  set one;
  array x(*) _numeric_;
  do i = 1 to dim(x);
    if x(i) = 999 then x(i) = .;
  end;
  drop i;
run;

proc print data = three;
run;
```

- 数组说明中 “*” 代表自动按变量个数计算数组长度，DIM(x) 求数组 x 的长度。

例

- 设数据中每行有 5 个性别，希望计算每行男性人数和女性人数，属于对列的循环处理。用数组和循环。

```
data counting;
  input gender1 - gender5;
  array gender(5) gender1-gender5;
  males = 0;  females=0;
  do i = 1 to 5;
    if gender(i)=1 then males = males + 1;
    else if gender(i)=2 then
           females = females + 1;
  end;
```

```
  drop i;
 cards;
 1 2 1 1 1
 2 1 2 2 1
 2 2 2 1 1
 ;
proc print;run;
```

-

例

- 设数据中每行有 4 个值，希望每行的 4 个值从小到大排列。

```
data one;
 input y1-y4;
 cards;
 15 36 27 4
 6 128 36 52
 14 29 54 43
 ;
run;
```

例

- 可以用 ordinal 函数：

```
data three;
 set one;
 array x(4);
   do i = 1 to 4;
     x(i) = ordinal(i, of y1-y4);
   end;
 drop i;
run;
```

```
proc print data = three;
run;
```

字符型数组

- 定义字符型数组时需要加一个 $ 符来说明数组元素类型为字符型, 并且要说明每一元素所能存储的字符串的最大长度。
- 说明格式如下:

  ARRAY 数组名 (维数说明) $ 元素长度说明 数组元素名列表 (初始值表);
- 其中 $ 说明不可省略。元素长度声明可以省略，这时默认长度为 8 字节。
- 例：

```
ARRAY  names(3)  $ 10  child  father  mother;
```

二维数组

- 二维数组在维数说明中指定用逗号分开的两个下标界。
- 如

```
array  table(2,2)  x11 x12 x21 x22;
```

- 注意：这样的数据集每行保存一个矩阵（二维数组）的所有元素，整个数据集保存了多个矩阵。
- 次序按行排列。

临时数组

- SAS 数组每一个元素对应数据集中的一个变量。
- 临时数组不对应数据集的变量。
- 定义格式是在数组说明中加一个 `_TEMPORARY_` 的关键字:

  ARRAY 数组名 (维数说明) _TEMPORARY_ (初始值表);

- 例:

```
ARRAY x(100) _TEMPORARY_ (100*0);
```

2.2.6 函数

函数

- 包括:
  - 数学函数;
  - 字符串函数;
  - 分布函数、分布密度函数、分位数函数、随机数函数;
  - 日期时间函数;
  - 数组函数;
  - 财政金融函数，等等。
- 调用:
  - 正常格式，如 `SUM(x1,x2,x3)`。
  - 变量名列表格式，如 `SUM(OF x1-x3)`, `MEAN(OF HSM HSS HSE)`。

数学函数

- ABS(x) 求 x 的绝对值。
- MAX(x1,x2,⋯,xn) 求所有自变量中的最大一个。
- MIN(x1,x2,⋯,xn) 求所有自变量中的最小一个。
- MOD(x,y) 求 x 除以 y 的余数。
- SQRT(x) 求 x 的平方根。

- LOG(x) 求 x 的自然对数。
- LOG10(x) 求 x 的常用对数。
- EXP(x) 指数函数 e^x。
- ROUND(x,eps) 求 x 按照 eps 指定的精度四舍五入后的结果, 比如 ROUND
  (5654.5654, 0.01) 结果为 5654.57, ROUND(5654.5654,10) 结果为 5650。
- CEIL(x) 求大于等于 x 的最小整数。当 x 为整数时就是 x 本身, 否则为 x 右边最近的整数。
- FLOOR(x) 求小于等于 x 的最大整数。当 x 为整数时就是 x 本身, 否则为 x 左边最近的整数。
- INT(x) 求 x 扔掉小数部分后的结果。
- FUZZ(x) 当 x 与其四舍五入整数值相差小于 1E−12 时取四舍五入。
- SIN(x), COS(x), TAN(x) 求 x 的正弦、余弦、正切函数。
- ARSIN(y) 计算函数 y=sin(x) 在 $x \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ 区间的反函数, y 取 [−1,1] 间值。
- ARCOS(y) 计算函数 y=cos(x) 在 $x \in [0, \pi]$ 的反函数, y 取 [−1,1] 间值。
- ATAN(y) 计算函数 y=tan(x) 在 $x \in (-\frac{\pi}{2}, \frac{\pi}{2})$ 的反函数, y 取 $(-\infty, \infty)$ 间值。
- SINH(x), COSH(x), TANH(x) 双曲正弦、余弦、正切。
- ERF(x) 误差函数 $\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$。
- GAMMA(x) 完全 Γ 函数 $\int_0^\infty t^{x-1} e^{-t} dt$。

数组函数

数组函数计算数组的维数、上下界, 有利于写出可移植的程序。数组函数包括:

- DIM(x) 求数组 x 第一维的元素的个数 (注意当下界为 1 时元素个数与上界相同, 否则元素个数不一定与上界相同)。

- DIMk(x) 求数组 x 第 k 维的元素的个数。如 DIM2(x) 计算二维数组 x 第二维长度。
- LBOUND(x) 求数组 x 第一维的下界。
- HBOUND(x) 求数组 x 第一维的上界。
- LBOUNDk(x) 求数组 x 第 k 维的下界。
- HBOUNDk(x) 求数组 x 第 k 维的上界。

字符串函数

- TRIM(s) 返回去掉字符串 s 的尾随空格的结果。主要用在连接两个字符串时去掉前一字符串的尾随空格，但是如果连接结果保存到变量中仍可能有尾随空格。
- 如：

```
data _null_;
  length s1 s2 $ 8;
  s1 = 'abcd';
  s2 = 'xyz';
  r1 = s1 || s2;
  r2 = trim(s1) || s2;
  put r1= r2=;
run;
```

- LEFT(s) 返回把 s 左对齐后的结果，空格个数不变。TRIM(LEFT(s)) 可以去掉 s 左右空格。
- RIGHT(s) 返回把 s 右对齐的结果，空格个数不变。
- COMPBL(s) 返回压缩 s 中的重复空白后的结果。
- COMPRESS(s) 返回删去 s 中空格后的结果。COMPRESS(s, < 要删去的字符串>, < 选项>) 返回删去 s 中的指定字符后的结果, 选项可以指定特定要删去的字符，如选项`'l'` 要求删去所有小写字母，选项`'s'` 要求删去所有空白，选项`'K'` 把第二个自变量的含义由要删去改为仅保留。

- TRANWRD(s,s1,s2) 从字符串 s 中把所有字符串 s1 替换成字符串 s2 后的结果。
- TRANSLATE(s, to, from) 把字符串 s 中出现的 from 中的字符替换成 to 中对应的字符。
- UPCASE(s) 把字符串 s 中所有小写字母转换为大写字母后的结果。
- LOWCASE(s) 把字符串 s 中所有大写字母转换为小写字母后的结果。在比较字符串时如果希望忽略大小写可以先把要比较的字符串都转换为小写或都转换为大写。
- RANK(s) 返回字符 s 的 ASCII 码值。BYTE(n) 返回第 n 个 ASCII 码值的对应字符。
- LENGTH(s) 返回字符串 s 的长度 (不包括尾随空格), 对空字符串返回 1。
- VLENGTH(v) 返回字符型变量 v 的存储长度。
- LENGTHC(s) 返回字符串 s 的长度 (包括尾随空格)。
- LENGTHN(s) 返回字符串 s 的长度 (不包括尾随空格)，对空字符串返回 0。
- 空字符串指只有若干个空格的字符串。
- SUBSTR(s,p,n) 从字符串 s 中的第 p 个字符开始抽取 n 个字符长的子串。
- SCAN(s, n, < 分隔符>) 把字符串 s 用“分隔符”分解为若干个子字符串，取其中第 n 个。SCAN 返回的结果为 200 个字符长，所以为了存储它返回的结果最好先用 LENGTH 语句说明变量长度, 或使用 TRIM 函数处理其返回结果。分隔符缺省为空白。
- SCAN 用法如：

```
data _null_;
  length name2 $ 10;
  name =scan('James, Bond', 2, ',');
  name2 =scan('James, Bond', 2, ',');
  s = '#' || name || '#';
  s2 = '#' || left(name2) || '#';
```

```
  put '#name#=' s;
  put '#name2#=' s2;
run;
```

- INDEX(s,s1) 查找 s1 在 s 中出现的位置。找不到时返回 0。
- INDEXC 函数查找字符集合的出现位置。
- INDEXW 函数查找用分隔符分隔后字串的出现位置。
- FIND 函数功能与 INDEX 类似，但可以指定不区分大小写、删除尾随空格后再查找、从某位置之后查找、反向查找。
- FINDC 函数功能与 INDEXC 类似，但象 FIND 函数那样增加了选项。
- COUNT 和 COUNTC 对某子字符串或字符集合的出现次数计数。
- 用 || 连接两个字符串。
- REPEAT(s,n) 字符表达式 s 重复 n 次。
- CAT(s1,s2,...) 连接字符串，不改变引导或尾随空格。
- CATS(s1,s2,...) 连接字符串，不含引导和尾随空格。
- CATT(s1,s2,...) 连接字符串，不含尾随空格。
- CATX(< 分隔符 >, s1,s2,...) 连接字符串，不含引导和尾随空格, 但插入分隔符。
- input(s, format.) 按照指定的格式 format. 把字符型值 s 转换为数值型。比如 `input('15.2', 8.)` 的结果为数值型的 15.2, `input('$123,456.78', comma16.)` 的结果为数值型的 123456.78;
- 字符型数据可以自动转换为数值型参加四则运算，但应尽量避免采用这种办法。
- input 在指定字符型输入格式后也可以用来读入字符型。
- put(x, format.) 按照指定的格式 format. 把数值型 x 转换为字符型值。比如 `put(15.2, 6.2)` 把数值 15.2 转换为字符串” 15.20”。
- 数值型可以自动转换为字符型参加字符运算，但应避免。

正则表达式函数

- 在数据步中，PRX 类的函数和 CALL 子程序实现了 PERL 语言中的正则表达式功能。
- 利用正则表达式，可以在字符串中搜索复杂的模式，取出满足模式的子串，并可以进行多处替换。
- 可以高效地对大量文本数据进行处理。
- 正则表达式的语法可以参看有关参考书籍。
- 正则表达式函数有 PRXMATCH, PRXCHANGE, PRXPARSE, PRXPAREN, PRXPOSN。
- 正则表达式 CALL 子程序有 CALL PRXSUBSTR, CALL PRXCHANGE, CALL PRXNEXT, CALL PRXPOSN, CALL PRXDEBUG, CALL PRXFREE 等。
- 参见 SAS 9.2 Language Reference Dictionary。

PRXMATCH 函数

- PRXMATCH(模式字符串, 源字符串) 返回在源字符串中模式的首次出现位置。
- 如

```
ii = prxmatch('at', 'a hat')
```

结果为 4。

PRXCHANGE 函数

- PRXCHANGE 函数的用法如

```
prxchange('s/pattern/replacement/', times, 'source string')
```

其中第一自变量开始的 s，中间的正斜杠是替换模式的规定，pattern 是待被替换的模式，replacement 是要替换成的内容，函数第二自变量表示替换多少次 (-1 表示不限制)，函数第三自变量是源字符串。

- 例如:

```
s = prxchange('s/at/onk/', 1, 'a hat');
```

结果为 a honk。

- 又如如

```
s = prxchange('s/(\w+), (\w+)/$2 $1/', -1, 'Jones, Fred');
```

结果为 Fred Jones。

- 这里，\w 表示任意的字母，加号表示前一字符重复一到多次，圆括号内的内容表示一个子表达式。
- 在替换内容中，用 $1 表示第一个子表达式，用 $2 表示第二个子表达式，等等。

日期和时间函数

- MDY(m,d,yr) 生成 yr 年 m 月 d 日的 SAS 日期值。
- HMS(h,m,s) 由小时 h、分钟 m、秒 s 生成 SAS 时间值。
- DHMS(d,h,m,s) 由 SAS 日期值 d、小时 h、分钟 m、秒 s 生成 SAS 日期时间值。
- YEAR(date)、MONTH(date)、DAY(date)、QTR(date) 由 SAS 日期值 date 得到年、月、日、季度。
- WEEKDAY(date) 由 SAS 日期值 date 得到星期几，1 表示星期日，2 表示星期一，等等。
- DATEPART(dt), TIMEPART(dt) 求 SAS 日期时间值 dt 的日期部分和时间部分。
- HOUR(time | dt)、MINUTE(time | dt)、SECOND(time | dt) 从一个时间值或日期时间值求小时、分、秒。
- TODAY() 返回运行时的日期。

- INTNX(interval,from,n) 计算日期 from 后面的第 n 个关于 interval 的分界点的 SAS 日期。所谓 “关于 interval 的分界点” 是指 interval 的第一天，其中 interval 可以取’YEAR’, ’QTR’, ’MONTH’, ’WEEK’, ’DAY’ 等，于是关于’MONTH’ 的分界点就是每月的 1 号。比如, INTNX(’MONTH’, ’16Dec1997’d, 3) 结果为 1998 年 3 月 1 日。

- INTCK(interval,from,to) 计算从日期 from(不含) 到日期 to(含) 中间经过的关于 interval 的分界点的个数, 其中 interval 取’MONTH’ 等。比如, INTCK(’YEAR’, ’31Dec1996’d, ’1Jan1998’d) 计算 1996 年 12 月 31 日到 1998 年 1 月 1 日经过的年分界点的个数, 因为中间有两个 1 月 1 日所以结果得 2, 尽管这两个日期之间实际只隔 1 年。

分布密度函数、分布函数

- 作为一个统计计算语言, SAS 提供了多种概率分布的密度、分布等函数。

- 分布密度、概率、累积分布函数等可以通过几种统一的格式调用, 格式为

  - 分布函数值 = CDF(’ 分布’, x <, 参数表>);
  - 密度值 = PDF(’ 分布’, x <, 参数表>);
  - 概率值 = PMF(’ 分布’, x <, 参数表>);
  - 对数密度值 = LOGPDF(’ 分布’, x <, 参数表>);
  - 对数概率值 = LOGPMF(’ 分布’, x <, 参数表>);

- 其中 PMF 可以用 PDF 代替，LOGPMF 可以用 LOGPDF 代替。

分布名称

BERNOULLI, BETA, BINOMIAL, CAUCHY, CHI-SQUARED, EXPONENTIAL, F, GAMMA, GEOMETRIC, HYPERGEOMETRIC, LAPLACE, LOGISTIC, LOGNORMAL, NEGBINOMIAL, NORMAL 或 GAUSSIAN, PARETO, POISSON, T, UNIFORM, WALD 或 IGAUSS, WEIBULL。

可以只写前四个字母。

单独提供的分布函数

- PROBNORM(x) 标准正态分布函数
- PROBT(x,df<,nc>) 自由度为 df 的 t 分布函数。可选参数 nc 为非中心参数。
- PROBCHI(x,df<,nc>) 自由度为 df 的卡方分布函数。可选参数 nc 为非中心参数。
- PROBF(x,ndf,ddf<,nc>) F(ndf,ddf) 分布的分布函数。可选参数 nc 为非中心参数。
- PROBBNML(p,n,m) 设随机变量 Y 服从二项分布 B(n,p), 此函数计算 Y ≤ m 的概率。
- POISSON(lambda,n) 均值为 lambda 的 Poisson 分布 Y≤n 的概率。
- PROBNEGB(p,n,m) 参数为 (n,p) 的负二项分布 Y≤m 的概率。
- PROBHYPR(M,K,n,x<,r>) 超几何分布的分布函数。设 M 个产品中有 K 个不合格品, 抽取 n 个样品, 其中不合格品数小于等于 x 的概率为此函数值。可选参数 r 是不匀率, 缺省为 1, r 代表抽到不合格品的概率是抽到合格品概率的多少倍。
- PROBBETA(x,a,b) 参数为 (a,b) 的 Beta 分布的分布函数。
- PROBGAM(x,a) 参数为 a 的 Gamma 分布的分布函数。
- PROBMC 计算多组均值的多重比较检验的概率值和临界值。
- PROBBNRM(x,y,r) 标准二元正态分布的分布函数, r 为相关系数。

分位数函数

对于连续型分布，分位数函数是分布函数的反函数。SAS 提供了六种，S 语言中实现的分位数函数很多。SAS 提供的分位数包括：

- PROBIT(p) 标准正态分布左侧 p 分位数。结果在 −5 到 5 之间。
- TINV(p, df <,nc>) 自由度为 df 的 t 分布的左侧 p 分位数。可选参数 nc 为非中心参数。
- CINV(p,df <,nc>) 自由度为 df 的卡方分布的左侧 p 分位数。可选参数 nc 为非中心参数。

- FINV(p,ndf,ddf <,nc>) F(ndf,ddf) 分布的左侧 p 分位数。可选参数 nc 为非中心参数。
- GAMINV(p,a) 参数为 a 的伽马分布的左侧 p 分位数。
- BETAINV(p,a,b) 参数为 (a,b) 的贝塔分布的左侧 p 分位数。

随机数函数

- 某分布 $F(x)$ 的伪随机数，就是在计算机上用某种算法产生一个数列 $x_1, x_2, \ldots, x_n$, 使得 $\{x_i, i = 1, \ldots, n\}$ 表现为 $F(x)$ 的独立同分布样本。用这样的数列可以模拟研究包含随机变量 $Y \sim F(x)$ 的模型。
- 计算机上产生的随机数序列需要一个开始点，称为" **种子**"。
- SAS 在数据步中提供了随机数函数用来产生某种分布的随机数序列。
- 在同一个数据步中对同一个随机数函数的多次调用将得到不同的结果;
- 同一种子出发得到的随机数序列是相同的。

SAS 中的随机数函数

- 均匀分布随机数：
  - UNIFORM(seed) 或 RANUNI(seed), seed 为小于 $2^{31} - 1$ 的任意非负整数。0 是一个特殊种子，取 0 作为种子实际是用每次都变化的一个值作为种子。
- 正态分布随机数：
  - NORMAL(seed) 或 RANNOR(seed), seed 为小于 $2^{31} - 1$ 的任意非负整数。0 是一个特殊种子，取 0 作为种子实际是用每次都变化的一个值作为种子。N(mu,sigma^2) 的随机数可用 `mu + sigma*RANNOR(seed)` 生成。
- 指数分布随机数: RANEXP(seed) 产生参数为 1 的指数分布的随机数。期望为 1/lambda 的指数分布可以用 RANEXP(seed)/lambda 得到。
- 伽马分布随机数: RANGAM(seed, alpha), alpha>0, 得到形状参数为 alpha 尺度参数为 1 的伽马分布。设 X=RANGAM(seed, alpha), 则 Y=beta*X 是形状参数为 alpha, 尺度参数为 beta 的 GAMMA 分布随机数。如果 alpha 是整数, 则 Y=2*X 是自由度为 2*alpha 的卡方分布随机数。

- 三角分布随机数：RANTRI(seed,h), $0 < h < 1$。此分布在 0 到 1 取值, 密度在 0 到 h 之间为 $2x/h$, 在 h 到 1 之间为 $2(1-x)/(1-h)$。
- 柯西分布随机数: RANCAU(seed) 产生位置参数为 0, 尺度参数为 1 的标准柯西分布随机数。Y=alpha+beta*RANCAU(seed) 为位置参数为 alpha, 尺度参数为 beta 的一般柯西分布随机数。
- 二项分布随机数: RANBIN(seed,n,p) 产生参数为 (n,p) 的二项分布随机数, seed 为任意数值。
- 泊松分布随机数: RANPOI(seed,lambda) 产生参数为 lambda>0 的泊松分布随机数, seed 为任意数值。
- 一般离散分布随机数: RANTBL(seed, p1, ⋯, pn) 生成取 1, 2, ⋯, n 的概率分别为 p1, ⋯, pn 的离散分布随机数。
- 可以使用 RAND(' 分布名', 参数) 来生成不同分布的随机数，这时使用 CALL STREAMINIT(种子) 来指定种子。

样本统计函数

- SAS 对数据集的基本统计是按照变量（列）进行的。
- **样本统计函数**则是对某一行中的若干个变量进行统计。
- 调用格式：
  - “函数名 (自变量 1, 自变量 2, ⋯, 自变量 n)”；
  - 或者“函数名 (OF 变量名列表)”。
- 计算时忽略缺失值。比如，四个数中有两个缺失，则 MEAN 函数只计算这两个数的平均值。

样本统计函数表

- MEAN 均值
- MAX 最大值; MIN 最小值; LARGEST(k, ...) 求从大到小的第 k 个; SMALLEST(k, ...) 求从小到大的第 k 个。均忽略缺失值。
- ORDINAL(k, ...) 求从小到大的第 k 个，缺失值排列在最前面。
- N 非缺失数据的个数

- NMISS 缺失数值的个数。
- SUM 求和
- VAR 方差
- STD 标准差
- STDERR 均值估计的标准误差, 用 STD / SQRT(N) 计算。
- CV 变异系数
- RANGE 极差
- CSS 离差平方和
- USS 平方和
- SKEWNESS 偏度
- KURTOSIS 峰度
- RMS 平方后平均再开根

2.2.7 SAS/IML 矩阵功能简介

SAS/IML 矩阵功能简介

- SAS/IML 用来进行向量、矩阵的编程计算，提供了一个基于矩阵的编程语言。
- 支持标量、行向量、列向量、矩阵及其运算，可以使用字符型数据。
- 可以交互运行或写出程序整体运行。
- 可以读写 SAS 数据集。
- 可以作图。

IML 运行

- 交互运行

```
proc iml;
  reset print;
```

- 退出用 quit 语句。
- RESET PRINT 表示后续赋值的结果自动显示到输出窗口。否则需要用 PRINT 语句显示变量值。

赋值

- 变量赋值：

```
sc = 15.25;
vh1 = {1 2};
vh2 = {11 22};
vh3=5:9;
vv1 = {3, 4, 5};
vv2 = {30, 40, 50};
mat1 = {1 2 3,
        1 1 1};
mat2 = { 1 2 -1,
         -1 0  1};
mat3 = {"Li"  "Ming",
       "Zhang"  "Chong"};
```

- 矩阵常量用大括号包围，行的各元素用空格分隔，行间用逗号分隔。
- 可以用 5:9 这样的格式表示等差数列。

对应元素四则运算和矩阵乘法

- $+, -, *$: 矩阵加法、减法、乘法。
- $\#, /$：矩阵对应元素相乘、相除。
- 标量可以和矩阵作加、减、乘、除运算。
- 例：

```
res1 = vh1 + vh2;
res2 = vv1 - vv2;
res3 = mat1 # mat2;
```

```
res4 = mat1 * vv1;
res5 = mat1 + sc;
```

矩阵子集运算

- 例

```
A = {11 12 13 14,
     21 22 23 24,
     31 32 33 34};
A1 = A[2,3];
A2 = A[,3];
A3 = A[2,];
```

- 用 A[2,3] 表示 A 的第 2 行第 3 列元素。
- 用 A[,3] 表示 A 的第 3 列对应的列向量。
- 用 A[2,] 表示 A 的第 2 行对应的行向量。

- 例

```
A4 = A[{1,3},{2,4}];
A5 = A[2:3,2:4];
sumr = A[,+];
sumc = A[+,];
```

- 用A[{1,3},{2,4}] 表示 A 的第 1,3 行和第 2,4 列交叉得到的子矩阵。
- 用 A[,+] 表示 A 的各行的和组成的列向量（每行把各个列下标求和掉）。
- 用 A[+,] 表示 A 的各列的和组成的行向量（每列把各个行下标求和掉）。

矩阵合并、转置、求逆

- x`表示 x 的转置，如。

```
mat4 = mat1`;
```

- INV(A) 表示矩阵 A 的逆矩阵。如

```
mat5 = mat1 * mat4;
mat6 = INV(mat5);
mat7 = mat5 * mat6;
```

- ||: 矩阵左右连接，如

```
comb1 = mat1 || mat2;
```

- //: 矩阵上下连接，如

```
comb1 = mat1 // mat2;
```

单位阵和常数值矩阵

- 例：

```
B1 = I(4);
B2 = J(3,1);
B3 = J(3,4);
B4 = J(3,4, 100);
```

- I(4) 产生 4 阶单位阵。
- J(3,1) 产生长度为 3 的元素恒为 1 的列向量。
- J(3,4) 产生 3×4 的元素恒为 1 的矩阵。
- J(3,4,100) 产生 3×4 的元素恒为 100 的矩阵。

对角阵

- 若 A 是矩阵（行数、列数均大于 1），则 `diag(A)` 取出 A 的对角元素组成一个对角阵，如

```
D1 = diag(A[1:3,1:3]);
```

- 若 d 为列向量或行向量，则 `diag(d)` 生成对角元素为 d 的对角阵。如

```
D2 = diag(vv1);
```

广义逆和线性方程组求解

- `ginv(A)` 求矩阵 A 的加号广义逆（Moor-Penrose 广义逆）。如

```
R1 = ginv(A);
```

- 用 `nrow(A)` 求矩阵 A 的行数。用 `ncol(A)` 求矩阵 A 的列数。
- 用 `solve(A,b)` 求方程组 $Ax = b$ 的解。如

```
AA = {1 2 3,
      1 3 2,
      2 1 3};
bb = {2,3,1};
xx = solve(AA, bb);
```

矩阵比较

- <, =, >, <=, >=: 元素两两间的比较。
- ALL(两两比较结果) 函数表示自变量的各元素均为真 (非零)；
- ANY(两两比较结果) 函数表示自变量的元素中至少一个为真。
- &、|、^: 两个逻辑型矩阵元素间的与、或、非。

读数据集

- 例：

```
USE  sasuser.class;
READ ALL;
ratio = weight/height;
```

- USE 语句打开数据集。
- READ ALL 把数据集的所有变量的所有观测读入 IML 中，每个变量作为一个列向量。
- ALL 表示所有观测。

读变量子集

- 可以只读入指定的变量，如

```
READ ALL VAR {height weight};
ratio = weight/height;
```

- 可以把所有数值型变量读入为一个矩阵，如

```
READ ALL INTO Mat;
ratio = Mat[,3]/Mat[,2];
```

INTO 后面给出读入的矩阵的名字。也可以把变量子集用 INTO 子句读入到矩阵中。

读入行子集

- 在 READ 语句中用 POINT 后的下标，下标集合或下标变量指定要读入的行，如

```
READ POINT {1 3} INTO Mat;
ind = 1:3;
READ POINT ind INTO Mat;
```

- 在 READ 语句中用 WHERE 子句给定读入的行子集的条件，如

```
READ ALL VAR {height weight}
  WHERE(sex='M');
```

显示矩阵的列名和行名

- 例

```
READ ALL INTO Mat;
cn = {'age' 'height' 'weight'};
READ ALL VAR {'name'} INTO name;
PRINT Mat[rowname=name colname=cn];
```

写入数据集

- 如

```
CREATE newd VAR{name ratio};
APPEND;
CLOSE newd;
```

- 用 CREATE 语句把 VAR 后指定的变量组转换为一个数据集，用 APPEND 语句把转换的结果写入，用 CLOSE 语句关闭新生成的数据集。

2.3 SAS 语言的数据管理功能

SAS 语言的数据管理功能

- SAS 语言是一种专用的数据管理、分析语言, 它提供了很强的数据操作能力。
  - 可以轻易地读入任意复杂格式的输入数据;
  - 可以对输入的数据进行计算、子集选择、更新、合并、拆分等操作;
  - 提供了用来访问其它数据库系统如 Oracle、Sybase 的接口;
  - 提供了访问各种微机用数据库文件如 Excel、FoxPro 的接口及向导;
  - 提供了一个 SQL 过程来实现数据库查询语言 SQL 的功能。
- 本节介绍数据管理方面的常用语句及这些语句的常用功能。

2.3.1 SAS 数据步的运行机制

数据步处理

- SAS 系统处理一个数据步经过两个阶段：
  - 编译阶段；
  - 执行阶段。
- 编译阶段检查程序中的错误并获取数据集中各变量的信息，包括变量名、类型、存储长度等。
- 执行阶段运行已编译的程序。
- 数据步中如果有 INPUT、SET 等语句，执行阶段有一个隐含的循环。

SAS 数据步的运行机制：例子

```
data a;
  put x= y= z=;
  input x y;
  z=x+y;
  put x= y= z=;
  datalines;
10 20
100 200
;
run;
```

```
X=. Y=. Z=.
X=10 Y=20 Z=30
X=. Y=. Z=.
X=100 Y=200 Z=300
X=. Y=. Z=.
```

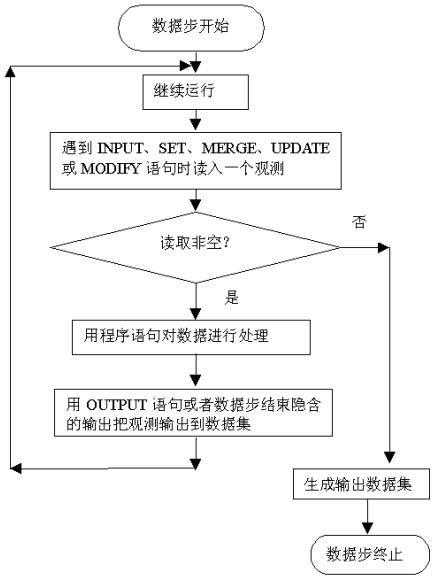
例子解释

- 在 INPUT 语句之前，变量缺失。
- INPUT 语句读入一行数据 10，20。
- 显示刚刚读入和赋值的变量值。
- 程序返回到数据步开头! 变量值置为缺失值。
- 再读入一行。
- 第三次读取时发现无数据，生成数据集，退出。
- 用 PRINT 过程查看生成的数据集。

数据步隐含循环

- SAS 数据步如果有数据输入, 比如用 INPUT、 SET、 MERGE、UPDATE、 MODIFY 等语句读入数据, 则数据步中隐含了一个循环。
- 数据步内的语句重复执行直到再无数据可以读取。
- 用 INPUT 语句从原始数据读取，每次隐含循环开始所有数据集变量均重置为缺失值，只有用 RETAIN 语句说明的、累加变量、临时数组元素和仅在数据步运行时有效的临时变量可以保留上一轮循环的值。
- 用 SET、MERGE、UPDATE、MODIFY 语句读取已有 SAS 数据集，也有隐含循环但是每轮循环开始时不把变量重置为缺失值。
- 特殊变量`_N_` 表示本轮循环的序号, 随隐含循环递增。
- 特殊变量`_ERROR_` 表示本轮循环的错误状态，一旦出错就保持为 1。

数据步流程图



用 PUT 语句调试数据步

- PUT 语句主要用来调试数据步程序。
- 为了显示当前的所有变量值，可以用

```
put _all_;
```

用 LIST 语句调试数据步

- LIST 语句可以显示 INPUT 语句正在读取的数据行, 如

```
data a;
  put x= y= z=;
  input x y;
  z=x+y;
  put _all_;
  list;
  cards;
10 20
100 200
```

```
;
run;
```

2.3.2 用 input 语句输入数据

用 input 语句输入数据—自由格式

用 input 语句输入数据—自由格式

- 数据步中把数据行写在 datalines 语句和一个只有一个顶头的分号的行之间，然后用 INPUT 语句读取。
- INPUT 语句引发数据步隐含循环，每次读取一个记录，在读取之前数据值置为缺失值。
- 最简单的 INPUT 语句使用**自由格式** (list input): 按顺序列出每个观测的各个变量名, 中间用空格分开。
- INPUT 语句中指定的变量如果是字符型的需要在变量名后面加一个 $ 符号。
- 数据行中一个观测的各变量只需要用一个或多个空格分开。

自由格式 INPUT 例子

```
data c9501;
  input name $ sex $ math chinese;
  datalines;
李明 男 92 98
张红艺 女 89 106
王思明 男 86 90
张聪 男 98 109
刘颖 女 80 110
;
run;
```

自由格式的特点

- 自由格式的优点:
  - 使用简单;
  - 输入数据时不必上下对齐;
  - 不需要知道每个变量的具体列数而只需知道它的次序。
- 使用自由格式的限制:
  - 数据每行为一个观测, 各数据值之间用一个或几个空格或制表符分隔。
  - 无论是字符型还是数值型缺失数据都必须用小数点表示。
  - 字符型数据长度不能超过 8 个字符 (但可以预先用 LENGTH 语句声明长度), 不允许完全是空白, 中间不允许有空白, 开头和结尾如果有空白将被忽略。
  - 在 INPUT 语句中必须列出观测中的每一项数据对应的变量名而不能省略中间的某一个。

用 input 语句输入数据—列格式

INPUT 语句的列格式

- 某个变量在数据行中是上下对齐的则可以对此变量指定**列格式**。
- 例:

```
data c9501;
  input name $ 1-10 sex $ 11-13
        math 14-16 chinese 17-20;
  datalines;
李明       男  92  98
张红艺     女  89 106
王思明     男  86  90
张聪       男  98 109
刘颍       女  80 110
;
run;
```

- 办法是在变量名 (及可能的 $ 符) 后面列出该变量在数据行中所占据的列起始位置 – 结束位置。
- 如果只有一个字符也可以只写开始位置。注意：一个汉字算两个字符。
- 列格式和自由格式可以混用。

列格式的特点

- 要求数据行各项上下对齐;
- 各项之间可以没有任何分隔, 连续写在一起;
- 字符型数据长度可以超过 8 个字符, 中间可以有空格, 头尾的空格仍将被忽略;
- 不论字符型变量还是数值型变量如果指定列位置都是空白则输入值为缺失值, 小数点仍表示数值型和字符型变量的缺失值;
- 可以只输入数据行中的某些项而忽略其它项，次序可以颠倒，可以重复读取某些已读取部分。

列格式输入例：紧凑格式

- 身份证号码的输入与解读。

```
data pids;
  input year 7-10  mon 11-12  day 13-14;
  birth = mdy(mon, day, year);
  format birth yymmdd10.;
  datalines;
110103197512092232
110101196902150059
;
run;
```

指定开始列读入

- 用 @ 列号 可以指定读入下一个变量的起始列号。
- 如

```
data;
  input @11 x y;
  datalines;
1 2  3   4 101 102
11 12      201 202
;
run;
proc print;run;
```

用 input 语句输入数据—有格式

有格式输入

- 日期在数据中格式多样，如“1998-10-9”, “19981009”, “9/10/98”。需要使用**有格式输入**。
- 格式说明跟在变量名 (及可能的 $ 符) 后面。
- 日期的输入格式有“YYMMDD10.”, “YYMMDD8.”等。
- 日期变量需要用 FORMAT 语句指定输出格式，一般为“YYMMDD10.”。
- FORMAT 语句用来指定变量值显示和打印的格式，格式都以句点截尾，如 10.2, YYMMDD10., $5. 等。

日期输入格式例子

```
data;
  input date yymmdd8. sales;
  format date yymmdd10.;
  datalines;
56-6-13    1100
```

```
67.12.15   1200
78 10 2    1300
891001     1400
19960101   1500
20020901   1600
;
run;
proc print;run;
```

日期输入格式例 2

```
data;
  input date yymmdd10. sales;
  format date yymmdd10.;
  datalines;
56-6-13    1100
67.12.15   1200
78 10 2    1300
891001     1400
19960101   1500
20020901   1600
1956-6-13  1100
1967.12.15 1200
1978 10 2  1300
19891001   1400
19960101   1500
20020901   1600
;
run;
proc print;run;
```

YEARCUTOFF 系统选项

- 输入两位数年份时，到底是 19xx 还是 20xx?
- 可以用 OPTIONS 语句规定 YEARCUTOFF 选项，如

```
options yearcutoff=1920;
```

- 此例认定所有两位数年份在 1920 到 2019 之间。
- 指定了 YEARCUTOFF，所有两位数年份当作 YEARCUTOFF 与 YEARCUTOFF+99 之间的年份。

INFORMAT 语句

- 在数据步中可以提前用 INFORMAT 语句说明一个变量的输入格式。
- 比如

```
data;
  informat date yymmdd10.;
  input sales date;
  format date yymmdd10.;
  datalines;
1100 56-6-13
1200 67.12.15
;
run;
proc print;run;
```

- 使用了 INFORMAT 的变量在 INPUT 语句中不需要再指定格式，字符型变量可以不再加 $ 符号。这样的输入是“修改的自由格式”，与下面使用了冒号修饰符的情况相同。

- 使用了 INFORMAT 的变量在 INPUT 语句中不需要再指定格式，字符型变量可以不再加 $ 符号。
- 这样的输入是“修改的自由格式”，与下面使用了冒号修饰符的情况相同。读入数据时可以自动跳过前导空格，并且遇到空格时停止读入。

冒号修饰符

- 如果日期变量输入值不在第一项, 并且与前一项用空白分隔，可以在日期变量名后面加一个冒号表示该变量从非空白值开始读取。
- 如

```
data;
  input sales date : yymmdd10. ;
  format date yymmdd10.;
  datalines;
1100         56-6-13
```

```
1200 67.12.15
;
run;
proc print;run;
```

修改的自由格式

- 在数据不按列对齐且需要使用输入格式进行转换时冒号修饰符可以解决不对齐的问题。
- 这叫做修改的自由格式。
- 预先使用 INFORAT 说明了输入格式的变量在 INPUT 语句中使用自由格式，这也是修改的自由格式，与加了冒号修饰符的效果相同。
- 修改的自由格式可以自动跳过前导空格, 读取到分隔符（缺省为空格）或行尾为止。

w.d 格式

- 在一个数值型变量后面可以用 w.d 或 w. 的输入格式，这里 w 表示输入宽度，d 表示输入的小数点位数，如果数据中已经有小数点则这里的 d 不起作用，如果数据中没有小数点则会把最后的 d 为作为小数部分。
- 如

```
data␣_null_;
␣␣input␣x␣␣6.2␣y;
␣␣put␣x=␣y=;
␣␣cards;
1.2␣␣␣␣10
123456␣20
3.4␣30
;
run;
```

- 结果第一行 x, y 均正确读入；第二行的 123456 被转换为 1345.56, y 正确读入；第三行程序出错，因为 y 的内容进入了 x 的 6 个字符宽度范围内。

带有 $ 符号和逗号的数字的有格式读取

- 用 DOLLARw.d 格式或 COMMAw.d 格式读取带有逗号、$ 符号以及括号的数据。括号内的数据表示负数。
- 用 percentw. 格式读百分数。
- 如

```
data;
  input num sales comma7. profit percent7.;
  datalines;
10 123,456 25.2%
15 245,889 9.8%
;
run;
proc print;run;
```

修改的自由格式与数值型输入格式

- 修改的自由格式与一个数值型输入格式配合时，不遵守数值型输入格式指定的宽度, 读到分隔符（缺省为空格）或行尾时结束。
- 如

```
data _null_;
  input x : 6.2 y;
  put x= y=;
  cards;
    123456.78 10
12345678 20
3.4 30
;
run;
```

- 结果三行的 x, y 都能正确读入，但是第二行的 12345678 被转换成了 123456.78。虽然第一行中 123456.78 带有前导空格而且宽度超过了 6, 第二行中 x 宽度超过 6，第三行中 x, y 两个数据的总宽度才达到 6，在冒号修饰符帮助下都正确读入了。

字符型变量有格式读入

- 其它类型的数据也可以用有格式读入方法进行转换读取。
- 用 $CHAR10. 的格式说明可以包括字符数据中间和前面的空格，以及单独的小数点。

```
data;
␣␣input␣a␣$␣1-5␣b␣$char5.␣c;
␣␣if␣b=''␣then␣put␣b=;
␣␣datalines;
abc␣␣␣␣␣3s␣10
abcd␣ab␣de20
abcde␣␣.␣␣␣␣30
;
run;
proc␣print;run;
```

- 字符型变量的另一个输入格式是 $w.，如 $10.。
- 用 $10. 的格式读入 10 个字符，包括前导空格和中间的空格，然后扔掉前导空格。
- 如果 10 个字符只有一个小数点，其它字符都是空格，则读入为缺失值，保存为 10 个空格。
- 在上例中如果变量 b 使用 $5. 读取，则第一行中的 3 个前导空格会被丢弃，第三行 b 变量的值会作为缺失值，也就是 5 个空格。

修改的自由格式与字符型输入格式

- 修改的自由格式如果与字符型输入格式配合使用，且该字符型变量为数据步中首次出现，则该字符型变量宽度由输入格式的宽度确定。读取该变量值时到分隔符（空格）或行尾时结束读取，但如果超过了格式规定的宽度就仅保存宽度规定的字符数。
- 如

```
data␣_null_;
␣␣input␣s␣:␣$6.␣t␣$;
␣␣put␣s=␣t=;
␣␣cards;
␣␣␣␣ab␣xy
abcdefgh␣zw
;
run;
```

- 结果第一行能正确读取 s 为 ab, t 为 xy, 第二行把 s 的值读入后仅保存了前 6 个字符内容为 abcdef, t 正确读取为 zw。

修改的自由格式与字符型输入格式 (续)

- 修改的自由格式如果与字符型输入格式配合使用，且该字符型变量不是首次出现，则该字符型变量的宽度已经确定。读取该变量值时到分隔符（空格）或行尾时结束读取，但如果超过了已经确定的宽度就仅保存宽度规定的字符数。
- 如

```
data␣_null_;
␣␣length␣s␣$␣3;
␣␣input␣s␣:␣$6.␣t␣$;
␣␣put␣s=␣t=;
␣␣cards;
␣␣␣␣ab␣xy
abcdefgh␣zw
;
run;
```

- 结果变量 s 的宽度确定为 3 个字符。第一行能正确读取 s 为 ab, t 为 xy, 第二行把 s 的值读入后仅保存了前 3 个字符内容为 abc, t 正确读取为 zw。

& 修饰符

- 在自由格式读取字符型数据时，不允许字符型数据中间有空格。
- 但是，如果字符型数据中间的空格都是单个空格，而字符型数据之后空出了至少两个空格，可以在该变量名后使用 & 修饰符读取这样的数据。
- 带有 & 修饰符和输入格式的变量输入也是修改的自由格式，除了需要用两个空格表示结束外，其它规定和冒号修饰符相同。
- 数据如

```
Li␣Ying␣␣abcd
Zhang␣Xiaoming␣␣␣␣efgh
␣␣␣Xu␣Jun␣␣limn
```

- 程序示例：

```
data;
␣␣length␣name␣$20;
␣␣input␣name␣&␣$␣code␣$char4.;
␣␣s␣=␣"#"␣||␣code␣||␣"#";
␣␣cards;
Li␣Ying␣␣abcd
Zhang␣Xiaoming␣␣␣␣efgh
␣␣␣Xu␣Jun␣␣limn
;
run;
proc␣print;run;
```

- 数据后的两个空格被作为分界使用。
- 数据之前的多余空格被忽略。
- 数据之后如果超过两个空格，多余空格会分配给后面的变量。比如上例中第二个观测的 code 为“ ef”。(可以对 code 使用冒号修饰符解决这个问题)

~ 修饰符

- 除了冒号、& 修饰符外，还有 ~ 修饰符, 通常与 INFILE 语句的 DSD 选项一起使用。
- 在 INPUT 语句中，~ 修饰符用于指定前面的变量为修改的自由格式读取，在与 DSD 选项配合时，变量值如果用撇号包围则读取时连撇号一同读取。
- 读取结束规则也改为遇到不被撇号包围的分隔符或遇到行尾时结束。
- 带有冒号修饰符、& 修饰符和 ~ 修饰符的变量输入使用“修改的自由格式”，可以跳过前导空格，并且在分隔符处才结束。如果读入的是字符型变量，读入内容超过变量宽度时超过部分被舍弃。
- 下例使用 DSD 选项指定了以逗号为分隔符，并用 ~ 修饰符读取以撇号包围的含有逗号的内容：

```
data scores;
  infile datalines dsd;
  input Name : $9. Score1-Score3 Team ~ $25. Div $;
  datalines;
Smith,12,22,46,"Green Hornets, Atlanta",AAA
Mitchel,23,19,25,"High Volts, Portland",AAA
Jones,09,17,54,"Vulcans, Las Vegas",AA
;
proc print data=scores noobs;
run;
```

- 读取 Team 的值时指定了 $25. 格式和 ~ 修饰符，这里所有 Team 内容长度都不超过 25 个字符，读取 Team 值时遇到双撇号外面的逗号分隔符时结束 Team 内容的读取。

多个变量共用输入格式

- 在 INPUT 语句中可以对多个变量可以指定相同的输入格式，只要把这些变量的列表用括号包围，并把输入格式用括号包围。
- 程序示例：

```
data name;
   input (s1-s3) ($1.);
   put s1-s3;
   datalines;
abc
;
run;
```

- 则变量 s1, s2, s3 分别读入为 a, b, c。

输入和输出格式补充

数值型输入格式补充

- 数值型输入格式缺省为 w.d，别名为 BESTw.d。如果数据中有小数点承认数据中的小数点 (这时 w.d 中 d 可省略)。如果数据中没有小数点会设置 d 位小数。
- 数据中有逗号、括号、$ 号时用 COMMAw.d 格式，别名为 DOLLARw.d。有括号的数被认作负数。
- PERCENTw. 读百分数。
- 例:

```
data _null_;
  x1 = input('123456.78', 16.); put x1=;
  x2 = input('123456.78', 16.2); put x2=;
  x3 = input('12345678', 16.2); put x3=;
  x4 = input('15.2E3', 16.); put x4=;
  x5 = input('$123,456.78', COMMA16.); put x5=;
  x6 = input('$123,456.78', COMMA16.4); put x6=;
```

```
  x7 = input('$(12345678)', COMMA16.2); put x7=;
  x8 = input('15.8%', PERCENT16.); put x8=;
run;
```

数值型输出格式补充

- 数值型输输出格式缺省为 BESTw.。在给定列宽内尽可能输出最大精度，可以用科学计数法。
- w.d 为指定宽度和小数位数的输出格式，省略 d 或 d 为零则输出整数。
- 用 COMMAw.d 格式输出用逗号分隔的结果。
- 用 DOLLARw.d 输出带有前导 $ 及用逗号分隔的结果。
- PERCENTw.d 输出百分数。负的百分数用括号包围。
- Zw.d 用零填充左边不满的位数。
- PVALUEw.d 为输出 p 值规定了特殊格式。

```
data _null_;
  s1 = '#' || put(123456.78, BEST16.) || '#';
  put s1=;
  s2 = '#' || put(123456.78, BEST4.) || '#';
  put s2=;
  s3 = '#' || put(123456.78, 16.4) || '#';
  put s3=;
  s4 = '#' || put(-123456.78, COMMA16.4) || '#';
  put s4=;
  s5 = '#' || put(-123456.78, DOLLAR16.4) || '#';
  put s5=;
  s6 = '#' || put(-1.782, PERCENT10.2) || '#';
  put s6=;
  s7 = 'x' || put(1, Z3.); put s7=;
  s8 = put(1.23E-6, PVALUE10.); put s8=;
run;
```

字符型输入格式补充

- 在 INPUT 语句中使用自由格式来读取字符型值不允许中间有空格。
- 缺省的字符型输入格式为 $w.。前导空格自动清除，如果数据只有一个小数点则读为空格 (小数点是缺失值的表示，字符型数据的缺失值为空格)。中间可以有空格。
- 用 $CHARw. 则允许读入前导、中置的空格和只有一个小数点字符值。

```
data a;
  input s $5.;
  ss = '#' || s || '#';
  datalines;
  cde
a cde
  .
run;
proc print;run;
data b;
  input s $CHAR5.;
  ss = '#' || s || '#';
  datalines;
  cde
a cde
  .
run;
proc print;run;
```

字符型输出格式补充

- 缺省为 $w. 格式，别名为 $CHARw.。不清除前导空格。如果需要去掉前导空格可用 LEFT() 函数。
- $QUOTEw. 输出双撇号包围的字符串。
- 如

```
data _null_;
  s1 = '  a b ';
  r1 = '#' || put(s1, $8.) || '#';
  r2 = '#' || put(left(s1), $8.) || '#';
  r3 = put(s1, $QUOTE8.);
  put '#' s1 $6. '#';
  put r1= r2= r3=;
run;
```

日期时间型输入格式补充

- 国内常用 YYMMDD10. 或 YYMMDD8.。
- 月日年格式用 MMDDYYw. 格式；日月年格式用 DDMMYYw. 格式。
- 像'23MAR1996' 这样的日期可以用 DATEw. 格式来读，如 DATE9.。
- 如'MAR1995' 这样的月份日期用 MONYYw. 格式读取。
- 如'1995Q1' 这样的季度日期用 YYQw. 格式读取。
- 如'15:28:25.32' 这样的时间用 TIMEw. 格式读，如 TIME12.。

日期时间型输出格式补充

- 国内常用 YYMMDD10. 格式输出日期。
- 月日年格式用 MMDDYYw. 格式；日月年格式用 DDMMYYw. 格式。
- DATEw. 格式可以把日期输出为像'23MAR1996' 这样的形式。
- TIMEw.d 格式可以把时间输出为如'15:28:25.32' 这样的格式。
- DATETIMEw.d 格式可以把日期时间输出为如'03MAY2011:15:33:25.21' 这样的格式。w.d 如 19., 21.2。
- HHMMw.d 格式把时间输出为如 12:35 这样的格式。

```
data _null_;
  d1 = input('12SEP1995', DATE9.);
  put d1 yymmdd10.;
  d2 = input('SEP1995', MONYY7.);
  put d2 yymmdd10.;
  t = input('15:28:25.22', TIME12.);
  put t time12.2;
  put t hhmm8.2;
  put t hhmm6.;
run;
```

input 语句高级技巧

input 语句中两个变量之间的衔接

- 一个观测中某个变量读完后，读取指针自动移到变量值之后。
- 例：

```
data;
␣␣input␣id␣$1-4␣name␣$char5.␣sex␣$;
␣␣s␣=␣"#"␣||␣name␣||␣"#";
␣␣cards;
1001KittyF
1002␣John␣M
1003␣␣Mary␣F
;
proc␣print;run;
```

- 对于无修饰符的列输入，读完后指针恰好在指定列之后的列。比如上例中 id 读取后 name 从第 5 列开始读。
- 对于无修饰符的有格式读入，开始读取的位置是前一个变量读完后的位置。比如上例中 Kitty 前没有空格也能正确读取。

- 无修饰符的有格式读入从开始位置恰好读取输入格式的宽度指定个数的字符，然后把读取指针移到其后。于是，Kitty 的性别从第 10 个字符即 F 处读取。
- 但 Marry 的姓名则读成了 `Mar`，从而性别也变成了 `y`。
- 对于无修饰符的自由格式，变量从上一个变量遗留的指针位置开始读取，会自动跳过任意前导空格，读到遇到一个空格位置，把读取指针设置在这个空格之后。
- 例如，前面例子中 name 改用自由格式读取：

```
data;
␣␣length␣id␣8␣name␣$5;
␣␣input␣id␣$1-4␣name␣$␣sex␣$1.;
␣␣s1␣=␣"#"␣||␣name␣||␣"#";
␣␣s2␣=␣"#"␣||␣sex␣||␣"#";
␣␣cards;
1001Kitty␣F
1002␣␣␣John␣M
1003␣Mary␣␣F
;
proc␣print;run;
```

- `Marry` 后的两个空格只用掉了一个，使得 sex 读入了空格。

列指针移动

- input 语句中两个变量衔接时，可以在第二个变量前面用 `+`n 移动到第二个变量的开始列。这是列指针相对移动。
- 也可以用`@`n 直接指定下一变量的开始列指针。

一行多个观测的情况

- 缺省情况下 INPUT 语句每次读新的一行。
- 在 INPUT 语句中用尾随的 `@@` 表示允许下一次隐含循环的 INPUT 运行时继续读取本次 INPUT 尚未读取的本行的数据。例如:

```
data;
  input x y @@;
  datalines;
1 1  2 4  3 9
4 16 5 25 6 36
run;
proc print;run;
```

同一隐含循环内多个 INPUT 语句的情形

- 如果在在 INPUT 语句中使用一个尾随的@，则表示允许本次隐含循环的其它 INPUT 语句继续读取本行剩余的数据。
- 例如:

```
data;
  input s $ @ ;
  if s^='yes' then delete;
  else input x y;
  datalines;
no 1 1
yes 2 4
yes 3 9
;
run;
proc print;run;
```

一个观测占多行的情况

- 一个观测可以占用多行，在 INPUT 语句中用#1, #2 等指定一个观测所属的行号。
- 如

```
data;
  input #1 x y #2 u w ;
  datalines;
1 1
2 4
3 9
4 16
;
run;
proc print;run;
```

- 在 input 语句中也可以用 “/” 命令指针跳到下一行开头。

读过行尾的问题

- INPUT 语句当从一行中不能把所有变量都读取到时，会自动进入下一行读取未完成的变量。
- 可以在 INPUT 语句前用

```
  infile datalines truncover;
```

  指示 INPUT 不自动进入下一行，如果仅仅是长度不足格式规定则接受读取到的部分，如果变量完全空缺则作为缺失值。如果输入的最后一行是空行，这个选项会造成一个全是缺失值的多余观测。

读过行尾的问题

- 把 TRUNCOVER 改为 MISSOVER，则不论是变量值空缺还是长度不足格式规定都作为缺失值。
- 把 TRUNCOVER 改成 STOPOVER，则发生这种错误时程序出错停止。
- 缺省的选择是 FLOWOVER，即试图从下一行读取未完成的变量。这种行为经常是不合适的。

2.3.3 变量属性

变量属性

- 变量的属性包括:
  - 字符型还是数值型。INPUT 语句读入字符型数据时要在变量名后面加 $ 符。
  - 变量标签 (LABEL)。可以给变量加一个长度不超过 256 个字符的标签 (可以用汉字，不超过 128 个汉字)，标签可以用在以后的报表中。
  - 变量存储长度 (LENGTH)。
  - 变量的输出格式 (FORMAT)。
  - 变量的输入格式 (INFORMAT)。
- 数据步中的 ATTRIB 语句可以指定变量的这些属性。
- 也可以使用单独的 LABEL、LENGTH、FORMAT、INFORMAT 语句来说明一种属性。

变量属性例子

- 属性为 “属性名 = 属性值” 这样的写法，可以为一个变量同时指定多个属性。
- 如

```
data sales;
  ATTRIB name LABEL=" 姓名" LENGTH=$10
         date LABEL=" 日期" FORMAT=yymmdd10.
              INFORMAT=mmddyy10.
         amount LABEL=" 金额" FORMAT=10.2;
  input name $ 1-10 date amount;
  datalines;
张鹏       10/15/1998 2000
李志明     1/3/99  1500
王敏       11/5/99 3000
;
run;
```

```
proc print noobs label;
run;
```

2.3.4 读入外部数据

读入文本格式的外部数据

- 少量数据可以在数据步程序中用 datalines 语句引入。
- 大量数据直接写在程序里不利于管理。
- 可以把数据放在文本格式的文件中，用 INFILE 语句指定从中读取。
- 例如:

```
data c9501;
  infile 'd:\users\sas\stud.txt';
  input name $ 1-10 sex $ math chinese;
run;
proc print;run;
```

- INFILE 语句要出现在 INPUT 语句前面。

INFILE 语句选项

- 为了读入用逗号分隔的文件，使用 INFILE 语句选项 DSD。DSD 把两个连续的分隔符中间看作缺失值，在两边的双撇号包围的字符串中的分隔符可以原样输入, 并默认逗号为分隔符。
- 读入 CSV 文件时如果需要使用输入格式转换，应该使用冒号修饰符指定修改的自由格式。
- 如果文件中第一行是表头，加 FIRSTOBS=2 选项。用 FIRSTOBS= 选项跳过开头的若干行。
- 用 OBS= 指定要读的最后一行。比如，有些输入文件的开头和末尾都有若干说明文字，只有中间的行是数据，就可以用 FIRSTOBS= 和 OBS= 配合指定要输入的行。在调试数据步程序时也可以指定如 OBS=10 来用前几行数据调试。

- 例：

```
data;
  infile 'class.csv' dsd firstobs=2;
  input name $ sex $ age height weight;
run;
proc print;run;
```

- 为了读入用制表符分隔的文件，在 INFILE 语句中加入 DELIMITER='09'x 选项, 或在 INFILE 语句中加上 EXPANDTABS 选项。如果用一个字符串作为分隔符，可以使用 DELSTR= 选项指定。

- 如 (在 SAS9.2 中)

```
data;
  infile 'tabd.txt' delimiter='09'x;
  input age height weight;
run;
proc print;run;
```

- 读入外部文本文件时缺省情况下每行宽度限制为 256 个字符，可以用 INFILE 语句的 LRECL= 指定读入宽度，加上 PAD 选项使得长度不足 LRECL 规定的行可以填充空白补足。

- INFILE 语句中用 DATALINES 关键字指定文件,可以使 INFILE 去读取 DATALINES 或 CARDS 语句后面的数据，这样可以利用 INFILE 的选项。

行尾的缺失值

- 如果用自由格式读入时尾部某些变量缺失而且没有用小数点表示，可以在 INFILE DATALINES 语句中加上 MISSOVER 选项表示不要到下一行去读取未找到的值，而是作为缺失值。如

```
data;
  infile datalines missover;
```

```
  input x y;
  datalines;
1
2 4
3 9
;
run;
proc print;run;
```

不同的缺失值

- 可以使用 MISING 语句指定一些字母表示不同类型的缺失值，这时可以在输入中用指定的字母 (不区分大小写) 表示缺失值。
- 如

```
data;
  missing x y z;
  input name $ age @@;
  if age=.X then
    put name 'have age of missing X type.';
  cards;
John 19  Mary x  Denny 21
Kitty y  Harte z
;
run;
proc print;run;
```

- 输入的年龄的 x, y, z 变成了特殊的缺失值。在程序中用.X 表示 X 型缺失值。

读入微机格式的外部数据

- “File—Import”：读入微机格式的数据库文件。包括:
  - 空格分隔的文件;

  - 逗号分隔的文件；
  - 制表符分隔的文件；
  - 用户自定义格式的文件；
  - dBase 格式的文件；
  - Excel 格式的文件；
  - Lotus 格式的文件；
  - 等等。

Import 例

- 以 class 数据集的 Excel 版本为例。此数据第一行为变量名，从第二行开始为数据行。
- 调用 File—Import;
- 选“Microsoft Excel 97 or 2000 Spreadsheets(*.xls)”格式;
- 选取文件“class.xls”。
- 选取保存转换结果的数据集位置和名字。
- 按 Finish 钮得到结果。
- 注意: Excel 文件格式复杂，不能完全等价于一个 SAS 数据集或一个数据库表。当有合并的行、列或独立的单元格时无法转换。Excel 的日期在转换时也比较容易出错。
- 可以在其它软件中把专用数据格式另存为 SAS 可以读取的格式。

转换用的 SAS 程序

- 可以调用 IMPORT 过程进行转换。
- 例：

```
PROC IMPORT OUT= WORK.class2
     DATAFILE= "E:\disk\course\statsoft\class.xls"
     DBMS=EXCEL REPLACE;
     SHEET="CLASS";
     GETNAMES=YES;
     MIXED=NO;
```

```
     SCANTEXT=YES;
     USEDATE=YES;
     SCANTIME=YES;
RUN;
```

- GETNAMES=YES: 以表的第一行为变量名;
- MIXED=YES|NO: 数值型与文本行混合列是否统一转换为字符型, 取 NO 则混合列中的数值型单元变成缺失值;
- SCANTEXT=YES|NO: 是否用所有观测确定字符型变量长度;
- USEDATE=YES|NO: 对日期时间型列，YES 按 DATE. 格式转换, NO 按 DATETIME. 格式转换;
- SCANTIME=YES: 对日期时间型列，如果仅有时间则转换为时间型。

与大型数据库的接口

- SAS / ACCESS 可以直接连接 Oracle、 Sybase、 SQL Server 等大型数据库。
- 有若干种办法连接到大型数据库服务器。
- 较简便的方法是用 LIBNAME 语句声明一个逻辑库 (Library)，这个逻辑库映射到大型数据库服务器中的一个数据库。外部库中的表直接可以看作 SAS 逻辑库中的表。
- 也可以对这些数据库中的表建立访问描述文件 (access descriptor)。
- 可以用 PROC SQL 的 SQL 直通功能建立到大型数据库的连接。
- 可以定义“**视图**”，视图可以当作数据集一样使用。

使用 LIBNAME 连接 Oracle 数据库的例子

- 例如, 在安装了 Oracle 数据库服务器客户端 SQL*Net 后，定义了数据库路径 hrdept_002, 库中有一个表 employees, 用户名 testuser 用密码 testpass 可以在运行 SAS 的计算机上通过客户端访问此库, 就可以用以下程序在 SAS 中直接访问这个 Oracle 库中的表:

```
libname mydblib oracle user=testuser
    password=testpass path=hrdept_002;
proc print data=mydblib.employees;
   where dept='CSR010';
run;
```

使用 LIBNAME 连接 ODBC 数据库的例子

- ODBC 是 MS Windows 操作系统中对数据库的一种通用接口。SAS 可以通过 ODBC 来访问数据库。
- 例如，我们在 IP 地址 192.168.0.101 上有一个 MySQL 数据库服务器，其中的库 sastest 我们有权限，用户名为 sasuser, 密码为 saspass。假设本机已经安装了 MySQL 的 ODBC 驱动。在 Windows 控制面板找到 ODBC 管理，然后添加一个命名为 sasmysql 的 MySQL 数据源，输入服务器 IP 地址, 端口为 3306，输入我们的用户名、密码、数据库名。在 Details 中选 Char Set 为 gbk。这样就生成了一个 ODBC 数据源。
- 在 SAS 中用 LIBNAME 来访问这个 ODBC 数据源:

```
libname mydb odbc datasrc=sasmysql;
```

- 这样，在 SAS 中访问逻辑库 mydb 实际上就是在访问 MySQL 数据库 sastest。

使用 LIBNAME 访问 Excel 文件的例子

- 从 SAS 9.1 开始 LIBNAME 也可以指定一个 Microsoft Excel 文件或 Microsoft Access 文件作为数据源。
- 一个 Excel 文件中可以有多个表 (sheets)，用 LIBNAME 指定了所有表所在的逻辑库。
- 语法如

  LIBNAME 库名 EXCEL 文件全路径字符串;

- 程序如

```
LIBNAME excl EXCEL 'class.xls';
proc datasets library=excl nolist;
  contents data=_all_ nods;
run;quit;
proc print data=excl.class;
run;
```

- 也可以用软件图形界面的 “New Library” 快捷图标对 Excel 文件定义逻辑库。

使用 LIBNAME 和 ODBC 访问 Excel 文件的例子

- 用 LIBNAME 可以访问在 ODBC 中定义为数据源 Excel 文件。
- 这种办法可以在 SAS9.1.3 中访问 Excel2007 的 xlxs 格式文件。
- 首先，在 Windows 控制面板中找到管理工具，在其中找到 “数据源 (ODBC)”，定义一个新数据源 xlsxSample，选择驱动程序时注意选择包括.xlsx 的 Excel 驱动程序，格式采用 Excel12.0，点击 “选择工作簿” 按钮选一个.xlsx 格式的 Excel 文件作为输入来源。
- 如下程序可以把这个 Excel 文件作为一个数据库访问，各个 sheet 是数据集：

```
LIBNAME myexcel ODBC datasrc=xlsxSample;
```

- 然后可以正常访问 Excel 文件中的数据表，如：

```
proc datasets library=myexcel nolist;
  contents data=_all_ nods;
run;quit;
proc print data=myexcel."c9501$"n;
run;
```

这里因为 sheet 命中有 “$” 符号所以用`myexcel."c9501$"n` 的格式。

- 使用完毕后用如下语句关闭 SAS 与 ODBC 的连接:

```
libname myexcel;
```

- 如果需要访问另外一个 Excel 文件，把要访问的文件复制到原来这个文件位置就可以了。

使用访问描述和视图的例子

- 例如, 在 Sybase 数据库服务器 DBIN 中有一个数据库 Finance, 其中有一个表 Sales, 用户名 guest 用密码 anyone 可以在运行 SAS 的计算机上访问此库, 就可以用以下程序在 SAS 中建立访问描述文件和视图文件:

```
PROC ACCESS DBMS=SYBASE;
    CREATE sasuser.sales.ACCESS;
    SERVER='DBIN';
    DATABASE='Finance';
    TABLE='Sales';
    USER='guest';
    PASSWORD='anyone';
    CREATE sasuser.salesall.VIEW;
    SELECT ALL;
RUN;
```

用 SQL 直连读取 ODBC 数据库

- 仍使用我们前面建立的 MySQL 数据库对应的 ODBC 数据源 sas-mysql。
- 可以用 PROC SQL 的 CONNECTION TO ODBC 来连接 ODBC 数据源。
- 程序如:

```
PROC SQL;
    CONNECT TO ODBC AS myos
      (DSN='sasmysql');
    CREATE VIEW sasuser.c9501b AS
        SELECT * FROM CONNECTION TO myos (
             SELECT name, math FROM c9501 );
    DISCONNECT FROM myos;
QUIT;
```

生成了视图。

- 非 ODBC 数据库也可以用这种办法连接。

2.3.5 数据集的复制与修改

SET 语句

- 数据步中用 SET 语句读入一个已有数据集，每次读入一行，直到读完(隐含循环)。
- 作用和 INPUT 语句相同，但读取数据的来源是已有数据集，而且读入前不把在 SET 语句中已读入的上一观测的变量值置为缺失，但是不是 SET 语句读入的变量仍会置为缺失。
- 对读入的行可以用赋值语句定义新变量，或修改原变量的值。
- 可以给定条件只保留满足条件的行。
- 可以只保留部分变量。

数据集信息

- 在图形界面交互运行时，用 SAS Explorer（SAS 资源管理器）可以查看数据库中有哪些数据集，用 Explorer 的“属性”右键菜单可以查看数据集的结构。
- 可以用 PROC CONTENTS 查看数据集结构，如

```
proc contents data=samp.c9501;
quit;
```

- 还可以用 PROC DATASETS 查看数据集结构，如

```
proc datasets library=sasuser nolist;
  contents data=c9501;
quit;
```

数据集复制和修改例子

- 复制：

```
data cls;
  set c9501;
run;
```

- 修改：

```
data c9501a;
  set c9501;
  if chinese>100 then chinese=100;
run;
```

- 注意 SET 语句有隐含循环。

取列子集

- 用 KEEP 语句指定要保留的变量；用 DROP 语句指定要丢弃的变量。
- 如

```
data c9501b;
  set c9501;
  keep name avg;
run;
```

- 或

```
drop sex math chinese;
```

取列子集：另一方法

- 用数据集的选项。如

```
data c9501b;
  set c9501(keep=name avg);
run;
```

这是输入选项。

- 或

```
data c9501b(keep=name avg);
  set c9501;
run;
```

这是输出选项。

数据集的横向拆分

- 用数据集的输出选项可以把数据集的不同变量拆分到不同数据集。
- 比如，希望把 C9501 的 name, sex 输出到数据集 A, name, math, chinese 输出的数据集 B，可用

```
data a(keep=name sex)
     b(keep=name math chinese);
  set c9501;
run;
```

- 在 DATA 语句中指定了两个输出数据集并对输出数据集指定了列子集。

数据集选项

- 在用 SET 语句引入数据集时可以给引入的数据集加选项，选项放在数据集名后的括号内:

  数据集名 (数据集选项)

- 选项包括:
  - KEEP=, 表示引入时只要指定的变量;
  - DROP=, 表示引入时舍弃指定的变量;
  - OBS=, 表示读取观测时读到指定的序号为止 (是序号而不是观测数);
  - FIRSTOBS=, 表示从指定序号的观测开始读取而跳过之前的观测不读。
- 输入数据集的选项只对原数据集中的变量有用。
- DATA 语句中的输出数据集也可以使用数据集选项。

数据集选项例

下面的程序生成了一个巨大的数据集 HUGE 然后用数据步从中复制了其前 100 行和前两个变量:

```
data huge;
  array x(10);
  do i=1 to 100000;
    do j=1 to 10;
      x(j) = normal(0);
    end;
    output;
  end;
  drop i j;
run;

data new;
  set huge(obs=100 keep=X1 X2);
run;
```

取行子集的子集 IF 语句

- IF 观测保留条件;
- 例:

```
data c9501c;
  set c9501;
  IF math>=90 and chinese>=100;
run;
```

- 注意子集 IF 和分支语句 IF 是不同的。子集 IF 本质上是对隐含循环的分支，条件成立时隐含循环继续并在数据步末尾处保存当前观测，条件不成立时忽略后面的语句直接进入下一轮隐含循环而不保存当前观测。
- 用 DELETE 语句可以删除当前观测。

用 WHERE 语句或 WHERE 选项取行子集

- 用 WHERE 语句指定保留观测的条件。如

```
data c9501c;
  set c9501;
  WHERE math>=90 and chinese>=100;
run;
```

- WHERE 语句只能使用读入数据集的原始值而不能使用修改后的值以及新定义的变量。
- WHERE 语句只能用于读入数据集，不能用在读取原始数据的情形。
- 可以在 SET 语句的数据集选项中使用 WHERE= 选项达到和 WHERE 语句类似功能。例:

```
data c9501c;
  set c9501(WHERE=
    (math>=90 and chinese>=100));
run;
```

WHERE 语句条件的增强功能

- 在 WHERE 语句中的条件除了已经讲过的比较运算和逻辑运算外，还支持其它一些增强的比较运算（仅用于 WHERE 语句和 WHERE 选项）。
- x BETWEEN a AND b 相当于 $x \in [a, b]$。x NOT BETWEEN a AND b 相当于 $x \notin [a, b]$。
- x IS NULL 或 x IS MISSING 相等于判断 x 为缺失值。判断 x 不是缺失值用 IS NOT NULL 或 IS NOT MISSING。
- 用 x CONTAINS ' 子字符串' 判断 x 中是否包含指定的子字符串，比较时区分大小写。
- 用 x LIKE ' 模式' 判断 x 是否与指定模式匹配。在模式中，用一个下划线代表一个任意字符，用一个百分号代表任意长度的子串。

2.3.6 用 set 和 output 语句拆分数据集

用 WHERE 输出选项纵向拆分数据集

- 用 SET 读入的观测行可以用输出选项拆分到不同数据集。
- 例如，把 C9501 的男生观测输出到 C9501M 数据集，女生输出到 C9501F 数据集，可以用

```
data c9501m(where=(sex=' 男'))
     c9501f(where=(sex=' 女'));
  set sasuser.c9501;
run;
proc print data=c9501m;run;
proc print data=c9501f;run;
```

用 set 和 output 语句拆分数据集

- 用 SET 读入的观测行可以按不同条件拆分到不同的数据集。
- OUTPUT 语句可以指定观测要写入的数据集。

拆分数据集例子

- 把 C9501 数据集拆分成男生、女生两个数据集:

```
data c9501m c9501f;
  set c9501;
  select(sex);
    when(' 男') output c9501m;
    when(' 女') output c9501f;
    otherwise put sex= ' 有错';
  end;
  drop sex;
run;
proc print data=c9501m;run;
proc print data=c9501f;run;
```

- 注意: DATA 语句中指定了两个数据集；OUTPUT 语句指定当前观测输出到哪一个数据集。

OUTPUT 的作用

- OUTPUT 语句还可以用来强行写入数据集，在用循环生成或处理数据时是必要的。
- 数据步中有了 OUTPUT 语句后数据步流程中不再有自动写入观测的操作, 而只能由 OUTPUT 语句指定输出。
- 不指定数据集名的 OUTPUT 语句输出到所有结果数据集。
- 例: 生成一个包含 1 到 10 及其平方的 10 个观测的数据集

```
data sq;
  do i=1 to 10;
    j=i*i;
    output;
  end;
run;
proc print;run;
```

- 如果删去上面的 OUTPUT 语句则结果数据集中只有 i=11, j=100 的一个观测。

2.3.7 SET 语句其他选项

POINT= 选项

- 为了读取数据集中指定的行，使用带有 POINT= 选项的 SET 语句，并配合使用 OUTPUT 语句和 STOP 语句。
- SET 语句中的 POINT= 选项指定要读取的行号。使用了 POINT 选项后就是文件直接访问而非通常的顺序访问，不能通过输入文件结尾来终止数据步，所以必须在数据步末尾用 STOP 语句使数据步终止。
- 必须显式使用 OUTPUT 语句输出读入的观测。
- 例：

```
data new;
  do i=1 to 19 by 2;
    set sasuser.class point=i;
    output;
  end;
  stop;
run;
proc print;run;
```

END= 选项

- 可以在 SET 语句中用 END= 指定一个临时变量名，此临时变量在读入的数据集的最后一行为 1，其它行为 0。
- INFILE 语句中也有 END= 选项，但是如果数据是内嵌在数据步中的情况就只能用 INFILE DATALINES LASTLINE= 转向行号的办法。
- 比如，为了用数据步计算某列 x 的平均值且只保留平均值，可用如

```
data one;
  input x @@; cards;
1 2 3 4 5
;
data two;
  set one end=lastline;
  avg + x;
  if lastline then do;
     avg = avg / _n_;
     output;
     end;
  keep avg;
run;
```

- 上例中，用了累加语句 (avg + x)。累加语句自动赋初值零且累加值不会在隐含循环中被置为缺失值。
- 用了特殊变量_n_, 表示循环的下标。
- 用了 OUTPUT 语句，只要有 OUTPUT 语句在数据步中则只有用 OUTPUT 指定的行才输出，其它行不再自动写入输出数据集。
- 对此例，如果我们希望把得到的平均值合并回原始数据集并对每行都有 AVG 的值，可用如

```
data three;
  set one;
  if _n_=1 then set two;
run;
```

- 这里，有两个 SET 语句。这时在读入 ONE 数据集第一行时又读入了 TWO 数据集的唯一一行，结果数据集的变量包括原来两个数据集中所有变量。
- SET 语句读入的变量不会在隐含循环开始时自动置为缺失值，只有 INPUT 语句这样做。

RETAIN 语句

- 数据步的隐含循环在每次循环开始时把所有或部分变量赋为缺失值，可以用 RETAIN 语句指定在隐含循环之间保持原来值的变量并赋初值，格式为

  RETAIN 变量名 初值

- 如

```
data;
  retain prod 1;
  input x @@;
  prod = prod * x;
  cards;
1 2 3 4 5
;
```

- 对于 SET 语句导致的隐含循环，为了给新定义的变量赋初值并保持上一步的值也应该使用 RETAIN 语句。
- RETAIN 也经常用于需要用到上一行的数据值的情况，如时间序列数据的处理。

2.3.8 数据集的纵向合并

数据集的纵向合并

- 几个结构相同的数据集可以上下地连接到一起。
- 例如: 三个班的学生的 3 个数据集连接。

class1		class2		class3	
ID	A	ID	A	ID	B
1	11	2	12	3	23
5	15	4	14	6	26

```
data classes;
  set class1 class2
      class3;
run;
```

合并的数据集变量为 ID, A, B，其中 ID 为 3,6 时 A 为缺失值，ID 为 1，5，2，4 时 B 为缺失值。

交错纵向合并

- 在使用 SET 语句进行纵向合并时，使用 BY 语句可以令合并结果按照 BY 语句指定的变量排序。
- 例如:

```
proc sort data=class1;
  by id;
proc sort data=class2;
  by id;
proc sort data=class3;
  by id;
data classes;
  set class1 class2 class3;
  by id;
run;
```

数据集选项 IN

- 数据集选项 IN 指定一个临时变量用来指示某观测是否来自该数据集。观测来自该数据集时 IN= 指定的变量取 1，否则取 0。
- 例如:

```
data new;
  set c9501m(in=male) c9501f(in=female);
  if male=1 then sex=' 男';
  if female=1 then sex=' 女';
run;
```

- 临时变量只在数据步运行期间起作用，不保存为数据集的一列。

结构不同的纵向合并

- 要纵向合并的数据集有同名变量但类型不同时，不能合并。可以用 PUT 函数或 INPUT 函数转换类型。
- 数据集中变量的次序可以变化，结果还是同名变量合并在一起。
- 不同名的变量都进入合并结果，但是只有存在此变量的数据集的对应观测中才有非缺失值。
- 如果要合并的数据集中同名变量的输出格式、标签等属性不同，则显式定义的属性优先采用，有多个显式定义的则取最先一个。
- 如果要合并的数据集中同名变量的长度不同，可以在纵向合并的数据步中用 LENGTH 语句显式说明合并后的变量长度。如果没有显式说明，则字符型变量长度取 SET 语句中第一个数据集中的变量长度, 这样有时会造成后续数据集中的变量值被截断。

用 PROC APPEND 纵向合并

- 两个结构完全相同的数据集可以用 PROC APPEND 纵向合并, 此过程直接把第二个数据集的内容合并到第一个数据集的尾部，这样比用数据步合并的效率要高，尤其是第一个数据集比较大的时候这样合并更为合理。
- 合并的结果存入第一个数据集中。
- 如

```
PROC APPEND BASE=class1 DATA=class2;
run;
```

  把 CLASS2 合并到了 CLASS1 的后面，结果数据集为 CLASS1。

2.3.9 数据集的横向合并

数据集的横向合并

- MERGE 语句用来把两个 (或多个) 包含了同样的一些观测的不同属性 (变量) 对数据集合并起来。

- 例如: 数据集 C9501U 包含学生的姓名、性别, 数据集 C9501V 包含学生的数学成绩, 数据集 C9501W 包含学生的语文成绩, 且各数据集的观测是按顺序一一对应的, 就可以用如下带有 MERGE 语句的数据步把它们左右横向合并到一个数据集 NEW:

```
data new;
  merge c9501u c9501v c9501w;
run;
```

- 合并时如果有同名变量，结果只能留下一个，变量值将取为第二个数据集中的变量值。
- 但是，一但数据集观测次序不能完全对应，这样合并就会错行。
- 按照学生姓名合并则不会错行。
- 用 BY 语句指定一个用来对齐对关键列，然后用 MERGE 语句合并。
- 要求合并对数据集预先按照关键列排好次序。
- 如果有其它同名变量，也是只能保留一个，变量值为第二个数据集中的值。

数据集的横向合并例子 (一对一)

```
data c9501x;
  set c9501;
  keep name sex;
data c9501y;
  set c9501;
  keep name math chinese;
run;

proc sort data=c9501x;
  by name;
proc sort data=c9501y;
  by name;
data new;
  merge c9501x c9501y;
  by name;
run;
proc print;run;
```

横向合并：不匹配观测

- 用 MERGE 和 BY 横向合并时，不匹配的观测也包含在结果中。
- 如数据

```
data d1;
  input id $ x;
  cards;
a 1
b 2
;
data d2;
  input id $ y;
  cards;
a 21
c 23
;
run;
```

- 合并为

```
proc sort data=d1;
  by id;
proc sort data=d2;
  by id;
data new;
  merge d1 d2;
  by id;
proc print;run;
```

- 结果包含 a,b,c 三行。
- 为了在结果中仅包含匹配行，可以使用数据集的 IN= 选项，如

```
data new;
  merge d1(IN=ina) d2(IN=inb);
```

```
  by id;
  IF ina and inb;
proc print;run;
```

- 类似地，可以要求仅包含出现于 D1 的行，如

```
data new;
  merge d1(IN=ina) d2;
  by id;
  IF ina;
proc print;run;
```

一对多的横向合并例子

- 数据：

d1		d2	
ID	X	ID	Y
a	11	a	21
b	12	a	22

- 合并：

```
data new;
  merge d1 d2;
  by id;
run;
proc print;run;
```

- 一对多合并时，d1 中的 a 的一行与 d2 中 a 的两行重复搭配两次。
- 如果有其它同名变量，只能留下一个，但是保留那些值变得更复杂。

多对多的横向合并例子

- 数据：

d3		d4	
ID	X	ID	Y
a	11	a	21
b	12	a	22
a	13	b	23
		a	24

- 合并：

```
proc sort data=d3; by id;
proc sort data=d4; by id;
data new;
  merge d3 d4;
  by id;
run;
proc print;run;
```

- d3 中 a 的两行和 d2 中 a 的三行合并，因为已经按 ID 排序, 会把所有的行依次搭配，缺少的行沿用上一行的值。
- 这与后面讲的 SQL 连接很不相同。

2.3.10 用 update 语句更新数据集

用 update 语句更新数据集

- 一个大数据集中如果有少量观测需要修改，打开大数据集然后直接编辑是不方便的。
- 另外，在正规的数据整理流程中，直接修改原始数据是不允许的；对原始数据的修改必须由可存档检查的程序来完成。
- 如果大数据集中有一个关键列，可以建立一个小的只包含要修改的观测的数据集，然后用数据步语句 UPDATE 来修改并把结果存入新的数据集。

- 格式:

DATA 新数据集名;
  UPDATE 原数据集 更新用数据集;
  BY 关键变量;
RUN;

- 小数据集中只包含需要修改的观测，不需要修改的变量可以排除在小数据集之外，不需要修改的值只要取缺失值。
- 大数据集的关键变量必须是能唯一区分观测的；而小数据集允许对每个要修改的观测分多个观测来输入。
- 如果希望小数据集中变量取缺失值时也把结果中变量置为缺失值，可以在 UPDATE 语句末尾加选项 UPDATEMODE=MISSINGCHECK。
- 小数据集中也可以包含全新的观测，更新后添加到结果数据集中。

UPDATE 例子

把数据集 C9501 中王思明的语文成绩实际改为 91 分, 张红艺性别应为男:

```
data upd;
  input name $ sex $ chinese;
  datalines;
张红艺 男 .
王思明 . 91
;
run;
```

UPDATE 例子 (续)

```
proc sort data=c9501;
  by name;
run;
proc sort data=upd;
  by name;
run;
```

```
data new;
  update c9501 upd;
  by name;
run;
proc print;run;
```

UPDATE 例子 (续 2)

需要计算新的平均分数。修改了数据集中原来根据某些列计算得到的列都需要重新计算。

```
data new;
  update c9501 upd(in=in_upd);
  if in_upd=1 then
     avg = math*0.5 + chinese/120*100*0.5;
  by name;
run;
proc print;run;
```

MODIFY 语句

- 数据步中 MODIFY 语句和 UPDATE 语句类似，但是 MODIFY 语句直接修改原数据集，而不是像 UPDATE 语句那样把修改的结果另存为新数据集。
- MODIFY 可以配合 OUTPUT 语句，在原数据集末尾添加观测；这时，需要修改的观测要用 REPLACE 语句命令进行替换。
- 为了保证数据安全，正式工作中应该尽可能用 UPDATE 取代 MODIFY。

2.4 SAS 宏介绍

SAS 宏介绍

- SAS 数据步是单独执行的, 不能定义成一个模块化的可以带输入参数的子程序。
- 这样, 我们在进行某些重复性工作时会感到不便。
- SAS 特别提供了一种宏语言功能, 部分地解决了这个问题。
- SAS 宏语言包括宏变量、宏函数和宏，本质是一种字符串替换，于数据步和过程步编译之前就执行。
- 使用 SAS 宏可以帮助我们：
  - 在 SAS 步之间传递数据；
  - 动态生成程序，比如重复执行某段代码；
  - 根据条件选择执行不同程序；
  - 写出灵活通用的程序。

2.4.1 实例

SAS 宏例子

10 个数据文件 df1.txt, df2.txt, ..., df10.txt 转换为数据集并连接:

```
%MACRO SALREAD;
%DO NP=1 %TO 10;
  %let ff="df&NP..txt";
  %let fd=df&NP;
  data &fd;
    infile &ff;
    input date yymmdd10. sales;
    persid=&NP;
  run;
%END;

%LET setstm=SET;
%DO NP=1 %TO 10;
  %LET setstm=&setstm df&NP;
```

```
%END;
%PUT &setstm;
data whole;
  &setstm;
run;
%MEND SALREAD;

%SALREAD;
```

2.4.2 宏变量

宏变量

- 宏变量保存一个字符串值，但可以把它的值替换到程序中。一个宏变量最多可以保存 64KB。宏变量名的命名规则与数据步变量名的命名规则相同。
- SAS 有一些预先定义的宏变量，叫做自动宏变量，比如 SYSDATE9, SYSTIME 等。
- 用 "& 宏变量名." 的格式可以在任何地方把宏变量的值替换到程序中。
- 在不混淆的时候可以省略以上格式中的小数点。
- 宏变量代换完全是一种字符串的替换，与 C 中的宏定义是同一类型。

使用系统自动宏变量的例子

- 例：

```
proc print data=samp.class noobs label;
    title 'Listing of the CLASS dataset';
    footnote1 "Created &systime. &sysday, &sysdate9.";
    footnote2 "on the &sysscp. System using Release &sysver.";
run;
```

- 使用了系统日期，时间，操作系统，SAS 版本等自动宏变量。
- 注意：双撇号界定的字符串内可以进行宏替换，但是单撇号界定的字符串内**不能**进行宏替换。

自定义宏变量

- 用 “%LET 宏变量名 = 值;” 的格式可以定义一个宏变量。
- 定义宏变量时，不需要用单撇号或双撇号包围文本，除非宏变量的内容本身需要这些标点；
- 数学表达式不被计算；
- 赋值的前导和尾随空格自动被删除；
- %LET 语句以分号结尾。
- 宏变量内容只要不超过系统规定的最大长度就可以，不像数据步字符型变量那样还要对每个变量规定长度。

自定义宏变量的例子

- 例

```
%let var1= SAS Macro;
%let var2 = "SAS Macro";
%let var3 = "SAS' Macro";
%let var4 = 3+4;
```

- 宏%PUT 语句把宏变量的值显示到日志窗口：

```
%PUT &var1. &var2. &var3.;
%PUT The value of var4 is &var4.;
```

- 宏%PUT 中非宏替换的文本原样显示，不需要用双撇号或单撇号界定。

宏替换与文本结合的例子

- 宏变量的内容在替换时可以直接与前后文本相连。如

```
%LET i=11;
%LET fname=file&i;
```

结果宏变量 fname 中保存了"file11", 两边的双撇号也是保存内容的一部分。

- 定义宏变量时可以引用已定义的宏变量值。
- 例如，已经定义了宏变量 fname, 在定义宏变量 ff 时就可以函数调用 fname 的值：

```
%LET ff="&fname..txt";
%PUT &ff.;
```

- 注意例中的连续两个点，第一个用于引用宏变量，第二个是文件名的一部分。
- 以 “& 宏变量名.” 格式引用宏变量值时如果不引起混淆可以省略其中的句点。

宏变量引用的多重扫描

- 有时需要用到编号的多个宏变量:

```
%LET sale95=beijing1995;
%LET sale96=beijing1996;
```

这里假设 beijing1995, beijing1996 是一些要处理的销售数据集。

- 引用宏变量时如果有两个连续 & 符号，会把两个 & 符号替换成一个 & 符号再进行宏替换，进行了两次代码扫描。
- 如

```
%LET k=95;
%PUT &&sale&k;
```

的 && 解析为一个 &, &k 解析为 95, 于是 “&&sale&k” 首先解析为 “&sale95”，然后再解析为 “beijing1995”。

- 这样的做法可以实现宏变量的数组功能。

- 如果用三个 & 符号：

```
%LET varpre=sale;
%LET k=95;
%PUT &&&varpre&k;
```

则第一遍扫描前两个 & 符替换为一个 & 符，第三个 & 符与 varpre 结合解析为 sale，&k 解析为 95，所以第一遍扫描后解析为“&sale95”，于是第二遍扫描将其替换为宏变量 sale95 的值“beijing1995”, 即所需的待处理数据集名。

- 这样实现了宏变量数组名和下标都是可变的。

宏%PUT 特殊变量列表

- 在宏%PUT 语句中，用_all_ 代表所有宏变量，如

```
%PUT _all_;
```

- 用_automatic_ 代表所有自动宏变量。
- 用_global_ 代表用户自定义的所有全局宏变量。
- 用_local_ 代表当前宏定义内的局部宏变量。
- 用_user_ 代表所有用户自定义变量。可以用于程序调试。
- 另外，如

```
%put ERROR: unexpected result.;
%put WARNING: maybe wrong.;
%put NOTE: some bad data?;
```

这样的用法可以在 LOG 窗口显示错误信息、警告信息或一般性备注，内容也自动以红色、绿色、蓝色显示。

删除宏变量

- 用%SYMDEL 宏语句删除一个或多个宏变量，如

```
%SYMDEL tmpvar1 var2;
```

- %SYMDEL 语句中列出要删除的宏变量名，不使用 & 格式。

用宏变量避免重复的例子

- 用宏变量可以写出适应性好的程序。
- 比如，为了对指定的数据集显示数据集结构和前 10 个观测，程序如

```
%LET dsn=samp.class;

title "Dataset &dsn";
proc contents data=&dsn.;
run;
proc print data=&dsn.(obs=10);
run;
title;
```

- 这样，不管程序中有多少个地方用到了数据集名，只要修改宏变量定义的一处就可以了。

自动宏变量—日期

- 自动宏变量是系统自动定义的宏变量，其中有些是用户不能修改的。
- 日期和时间：&sysdate 为 DATE7. 格式的当前日期。&sysdate9 为 date9. 格式的当前日期。&sysday 为星期几（单词格式）。如

```
%PUT &sysdate &sysdate9 &sysday;
```

自动宏变量—最新数据集

- &syslast 为最新生成的一个数据集, 库名和数据集名用句点连接。
- &sysdsn 保存最新生成的数据集的库名和数据集名，中间用空格分开。
- 如果本次 SAS 会话还没有生成数据集则值为_NULL_。

自动宏变量—错误状态

- 在宏中可能需要检查 SAS 步是否出错。宏变量 &syserr 保存一个 SAS 数据步或过程步结束后的错误状态代码，0 为正常。新的数据步或过程步重置错误状态。
- &syscc 为整个 SAS 会话的错误状态。

自动宏变量—运行环境

- &syssite 包含 SAS 软件安装地点代号。
- &sysscp 包含运行 SAS 的操作系统名称。
- &sysscpl 类似于 &sysscp 但更详细。
- &sysuserid 为登录用户名，缺省时为 default。
- 如

```
%PUT &syssite &sysscp &sysscpl &sysuserid;
```

- &sysmacroname 在运行一个宏时返回该宏的名字；否则为空值。

2.4.3 宏子程序

宏子程序

- SAS Macro 相当于子程序，但其中的子程序参数和宏变量都只有字符串替换的意义。
- 不带参数 (自变量) 的宏子程序定义格式如下

%MACRO 宏名;
…(定义宏的内容)
%MEND 宏名;

- 宏子程序定义中可以包括原样的文本、宏变量替换、宏程序语句、宏表达式和宏函数调用。
- 宏子程序在定义时并不运行，在调用时才运行。
- 调用为 “% 宏名” 的格式。

例：用宏子程序来替换程序片段

- 宏子程序可以简单到只包含原样的一段程序片段，如

```
%macro ds;
  c9501f c9501m
%mend ds;
data new2;
  set %ds;
run;
```

- 注意我们把%ds 类似于一个宏变量替换来使用。
- 调用结果是把宏子程序定义内的原样文本替换到调用所在位置。此处替换结果为

```
data new;
  set c9501f c9501m;
run;
```

- 替换的原样程序片段中可以含有分号。如

```
%macro dsc;
  c9501f c9501m;
%mend dsc;
data new3;
  set %dsc
run;
```

- 这里 set 语句就不需要再有分号，因为它引用的%dsc 的内容已经包含了分号。

例：宏子程序定义中包含宏变量替换

- 宏子程序定义中可以包含宏变量替换，如

```
%macro dsv;
  &d1 &d2
%mend dsv;
%let d1=c9501f;
%let d2=c9501m;
data new4;
  set %dsv;
run;
```

- 定义时并不要求其中用到的宏变量已经赋值，只要在调用时用到的宏变量已经赋值就可以了。

例：包含整段 SAS 程序的宏子程序

- 宏子程序定义中也可以包含一段 SAS 程序，如:

```
%MACRO dsinfo;
title "Dataset &dsn";
proc contents data=&dsn.;
run;
proc print data=&dsn.(obs=10);
run;
title;
%MEND dsinfo;
```

- 定义时其中的数据步、过程步、全局语句不运行。
- 这个例子的宏子程序定义中包含了一个 title 全局语句、两个过程步。

- 调用:

```
%LET dsn=samp.class;
%dsinfo
```

- 宏子程序调用本质上还是文本替换，而且替换出的内容只要构成一个全局语句、完整的数据步或完整的过程步就运行这些语句，运行完以后再继续宏子程序后续部分的替换。
- 调用宏时也不需要分号。
- 比如，此例执行宏子程序 DSINFO，即逐行语句进行替换，遇到 &dsn. 的地方就替换为 samp.class，得到四段可独立执行的 SAS 代码: 第一行 title 语句、第二、三行的 PROC CONTENTS 过程步、第四、五行的 PROC PRINT 过程步、第六行的 title 语句，每替换出其中一段后就执行一段，然后继续后面的替换。

带参数的宏

- 带参数的宏可以有更好的通用性。
- 定义格式为:

%MACRO 宏名 (参数表);
…(定义宏的内容)
%MEND 宏名;

- 调用格式为: "% 宏名 (参数值)"。类似于函数的调用方法。
- 调用时，其自变量是原样文本，不需要写成撇号界定的字符串。

带参数的宏：例子

- 例：

```
%MACRO dsinfov(dsn);
title "Dataset &dsn";
proc contents data=&dsn.;
run;
proc print data=&dsn.(obs=10);
```

```
run;
title;
%MEND dsinfov;

%dsinfov(samp.class)
```

- 注意定义中用到自变量 dsn 的地方要用宏变量替换格式 &dsn.。

- 调用结果相当于

```
title "Dataset samp.class";
proc contents data=samp.class;
run;
proc print data=samp.class(obs=10);
run;
title;
```

变量作用域

- 宏定义内部的参数和新定义的宏变量都是局部的。
- 在宏定义外部定义的宏变量是全局的。
- 用%GLOBAL 语句声明全局宏变量，此语句可以用在宏定义内，也可以用在宏定义外。
- 在宏定义内部可以用%LOCAL 语句显式地声明局部宏变量。这是比较严谨的编程习惯，可以避免错误地改变同名全局变量的值。

参数缺省值

- 定义带参数的宏时可以给参数缺省值。如

```
%MACRO dsinfod(dsn=&syslast, nobs=10);
title "Dataset &dsn";
proc contents data=&dsn.;
run;
```

```
proc print data=&dsn.(obs=&nobs.);
run;
title;
%MEND dsinfod;
```

- 调用时必须用“参数名 = 参数值”的格式调用。如

```
%dsinfod(dsn=samp.class, nobs=5)
```

- 缺省参数在调用时可以省略，如

```
data; x=1;y=2; run;
%dsinfod()
```

- 用“参数名 = 参数值”调用，次序可以改变，如

```
%dsinfod(nobs=5, dsn=samp.class)
```

- 在定义宏的时候参数可以一部分无缺省值，一部分有缺省值，但是无缺省值的参数必须都列在有缺省值的参数前面, 调用时，无缺省值的参数不能省略而且不能使用“参数名 = 参数值”的格式。

调试程序段

- 因为定义宏的时候内部的 SAS 程序并不运行，所以可以把一段 SAS 程序包围在宏定义中。即使程序中已经有/* 和 */界定的注释也可以。
- 这种方法也适用于一段用来调试的程序：把用来调试的程序包在宏定义中，后面紧跟着调用，调试成功后只要把调用部分删除或注释即可。
- 如

```
%MACRO debug1;
  proc contents data=&syslast;
  run;
```

```
  proc print data=&syslast(obs=5);
  run;
%MEND debug1;
%*debug1;
```

- 其中以%* 开始的语句叫做“宏注释”，是不执行的。
- 如果有多个程序段需要使用这种技术统一打开或关闭，可以使用如下技巧。
- 仍然把这些程序段分别包入宏定义中并适当命名；
- 定义一个开关宏变量如 DEBUG:
- 需要执行这些程序段时，定义 DEBUG 的值为空值（%LET 语句赋值时不写任何值）；
- 需要关闭这些程序段时，定义 DEBUG 的值为 *。
- 这样避免了使用宏%IF 语句，宏语句只能用在宏定义内部。
- 如

```
%LET DEBUG=*;
%MACRO debug1;
  .........
%MEND debug1;
&DEBUG %debug1;
......
%MACRO debug2;
  .........
%MEND debug2;
&DEBUG %debug2;
```

  这时 &DEBUG 替换后是星号，含有%debug1 或%debug2 的语句变成了注释语句。

- 如果第一行中把 DEBUG 定义为空值，含有%debug1 或%debug2 的语句会调用宏从而执行被包入宏定义内的程序段。

用宏组织一组分析

- 一组数据分析包括数据输入、整理和分析，通常由多个数据步、过程步组成。
- 比如，某次分析先生成数据集 NEW1、NEW2，然后合并数据集为 SASUSER.COMBINE，然后进行相关系数计算和方差分析。
- 可以把每个步写到一个宏中，步骤相同的宏可以合并为带有参数的宏。然后把这些宏的调用再写为一个宏。
- 在调试运行时，不需要运行的步只要注释掉即可。
- 如

```
...... (每个步的宏定义)
%MACRO MYANAL;
  %READD(NEW1, 15, 32) /* 生成 WORK.NEW1 数据 */
  %READD(NEW2, 20, 45) /* 生成 WORK.NEW2 数据集 */
  %*READD(NEW3, 51, 60) /* 生成 WORK.NEW3 数据集 */
  %COMBD(SASUSER.COMBINE) /* 合并所有 WORK.NEW? 数据集 */

  %CORREL  /* 进行相关系数计算 */
  %ANOVA   /* 进行方差分析 */
%MEND MYANAL;

%MYANAL
```

- 其中读入数据可以用统一的一个宏实现，调用时采用不同参数。第三个调用用宏注释取消了。
- 宏%COMBD 可以合并命名如 NEW1, NEW2, NEW3 的数据集。

2.4.4 流程控制结构

宏分支结构

- 宏分支结构格式为

  %IF 条件 %THEN 宏语句;

- 或

  %IF 条件 %THEN 宏语句;

  %ELSE 宏语句;

- 宏语句可以是复合的, 格式为：

  %DO;
  ...(多个语句)
  %END;

- 这些结构都只能用在宏定义内部。

- 最常见的条件是判断相等 (用等于号) 和不等于（用 NE)。条件只能使用宏变量值而**不能**使用数据步变量值。

- 宏语句的操作可以是宏变量赋值、调用其它宏、生成 SAS 代码片段，不能是运行数据步语句，因为宏的编译和执行都是在数据步编译执行之前就已经完成的。

用宏 IF 控制替换代码片段的例子

- 简单的替换如

```
data aa; input x y; cards;
1 2
11 12
;
run;
%MACRO myprog1(dsn=&syslast., neat=yes);
    proc print data=&dsn.(obs=10)
      %IF &neat=yes %THEN label noobs;
      %ELSE double;
    ;
    run;
%MEND myprog1;
options mprint;
%myprog1(dsn=aa, neat=yes)
%myprog1(dsn=aa, neat=no)
```

- 当以 neat=yes 调用时，proc print 得到了 label 和 noobs 选项；当以 neat=no 调用时，proc print 得到了 double 选项。
- 注意: %IF 语句中的宏变量要用 & 格式访问。
- 注意：与 neat 比较的 yes 不用撇号界定。
- 注意：此例中%IF 和%ELSE 宏语句都要以分号结尾，PROC PRINT 语句需要一个分号。用宏语句替换代码片段时一定要小心不要遗漏分号也不能多写分号。
- 对于整段程序，需要用宏复合语句包围，程序本身可以含有分号。见下例。

宏 IF 与复合语句

- 用 “%DO;” 和 “%END” 可以把若干 SAS 程序片段、语句或宏语句包含起来作为一个整体。
- 比如，上例中的%myprog1 可以改写为

```
%MACRO myprog1b(dsn=&syslast., neat=yes);
  %IF &neat=yes %THEN %DO;
    proc print data=&dsn.(obs=10) label noobs;
    %END;
  %ELSE %DO;
    proc print data=&dsn.(obs=10) double;
    %END;
    run;
%MEND myprog1b;
```

- 注意复合语句中的 SAS 程序带有分号。

用宏 IF 控制替换语句的例子

- 下面的宏中使用了宏 IF 判断是否处于调试阶段，如果在调试阶段则用 PROC PRINT 显示输入数据集的部分行：

```
%MACRO myprog2(dsn=&syslast., debug=);
  %IF &debug NE %THEN %DO;
    proc print data=&dsn.(obs=10);
    run;
  %END;
  proc means data=&dsn;
    var x y;
  run;
%MEND myprog2;
options mprint;
%myprog2(dsn=a, debug=true)
%myprog2(dsn=a)
```

- 用宏复合语句包含了一个 SAS 过程步作为条件成立时生成的 SAS 程序代码。

宏比较运算

- 宏 IF 中的条件通常是宏变量替换与原样文本的比较。
- 比较运算符和数据步所用的比较运算符相同:

```
=   ^=   >   <  >=  <=
EQ  NE  GT  LT  GE  LE  IN
```

- 注意：宏变量必须以 & 格式调用。
- 注意：与宏变量比较的文本不用撇号界定。
- 为了与空格比较，只要写成 “&debug NE ” 这样的条件即可。
- IN 运算符右面的列表是用空格分隔的。如

```
%IF &var IN aa bb cc %THEN %put &var;
```

- 宏变量的比较是区分大小写的，所以如果想不区分大小写可以用如

```
%IF %LOWCASE(&city)=beijing %THEN %PUT &city;
```

流程控制结构—宏计数 DO 循环

- 循环是我们需要用到宏对一个重要理由。为了重复地执行一些代码而不需复制代码很多遍，用宏循环就可以了。
- 宏计数循环只是把关键字 DO, TO, BY, END 都加上了% 前缀。
- 宏循环只能用在宏定义中。

宏 DO 循环例子

```
%MACRO SALREAD;
%DO NP=1 %TO 10;
  %let ff="df&NP..txt";
  %let fd=df&NP;
  data &fd;
    infile &ff;
    input date yymmdd10. sales;
    persid=&NP;
  run;
%END;

%LET setstm=SET;
%DO NP=1 %TO 10;
  %LET setstm=&setstm df&NP;
%END;
%PUT setstm;
data whole;
  &setstm;
run;

%MEND SALREAD;

%SALREAD;
```

宏子程序开发步骤

- 不使用任何宏程序机制写出程序模板；
- 修改程序模板，使用宏变量取代其中可变内容；
- 把程序修改为带有参数的宏子程序。
- 开发步骤的思路是先测试不包含宏程序的做法，然后仅用简单的宏替换，然后再加入较复杂的宏分支、宏循环。

宏当型和直到型循环

- 宏定义中可以使用当型和直到型循环。
- 当型循环如

```
%DO %WHILE (循环继续条件);
    原样文本或宏语句……;
%END;
```

- 当型循环先判断后执行。
- 直到型循环如

```
%DO %UNTIL (循环退出条件);
    原样文本或宏语句……;
%END;
```

- 直到型循环先执行后判断，至少执行一次。
- 注意：如果有计数变量，需要用%EVAL 函数计算增量。

2.4.5 宏程序调试

宏程序调试系统选项—MPRINT

- 有一些系统选项与宏程序调试有关。
- MPRINT 选项要求把宏程序运行生成的程序片段显示在 LOG 窗口中。如

```
options mprint;
data; x=11; y=12; run;
%MACRO debug1;
proc print data=&syslast;
run;
%MEND debug1;
%debug1
```

- 在日志窗口会额外显示调用宏%debug1 后替换出的 SAS 程序段，以 MPRINT(DEBUG1) 标识：

```
MPRINT(DEBUG1): proc print data=WORK.DATA2 ;
MPRINT(DEBUG1): run;
```

- 用 NOMPRINT 选项关闭此功能。

其它宏程序调试系统选项

- MLOGIC 选项跟踪并在 LOG 窗口显示宏分支、循环、调用的踪迹。用 NOMLOGIC 关闭。
- 打开 SYMBOLGEN 选项使得每次解析（替换）一个宏变量时都在 LOG 窗口显示解析情况。用 NOSYMBOLGEN 关闭。

2.4.6 宏程序库

宏程序库

- 以上的例子只能在一个 SAS 会话中定义并使用宏。
- 如何把编写好比较成熟的若干宏保存起来，调用时不需要每次运行其定义？
- 有两种办法。第一种办法是把这些宏保存在一个共同的子目录中，每个宏存入一个与宏名相同名称的扩展名为.SAS 的文件，这样的子目录称为一个宏库。比如，在"C:\saslib" 中保存了若干个宏 SAS 文件，用如下系统选项可以令其可以被找到：

```
options mautosource
    sasautos=("C:\saslib", sasautos);
```

- 其中 MAUTOSOURCE 选项打开宏库搜索功能，sasautos= 指定若干个搜索位置，列表中的 sasautos 是一个 SAS 自带宏库的位置。

编译存储宏

- 把宏放在共同子目录中组成宏库后，每次 SAS 会话第一次用到某个宏时还是需要编译。
- 另一个办法是把一组宏编译后保存在某个 SAS 数据库中命名为 SASMACR 的 SAS 目录簿 (catalog) 中，用系统选项 SASMSTORE 指出其所在的数据库，并用 MSTORED 系统选项打开此功能。
- 实际上，当前 SAS 会话中已编译的 SAS 宏就都临时保存在 WORK.SASMACR 目录簿中。
- 指定了编译存储位置后为了把某个宏存入此位置，需要在其%MACRO 语句最后加上“/ STORE”选项并运行。
- 如

```
options mstored sasmstore=samp;
%macro mysub(dsn,varl) / store;
........
%mend mysub;
```

将把宏 MYSUB 编译存入 SAS 目录簿 samp.sasmacr 中。

宏的搜索次序

- 调用一个宏时从那些位置找到这个宏？
- 首先查找的是 WORK.SASMACR 目录簿。这是当前 SAS 会话中编译过的宏的存储位置。

- 其次如果用 MSTORE 打开了编译存储宏功能并用 SASMSTORE= 选项指定了所在数据库，则从这个数据库的 SASMACR 目录簿中查找。
- 最后如果用 MAUTOSOURCE 打开了子目录存储宏的功能并用 SASAUTOS 指定了搜索路径，从这些搜索路径依次查找。
- 找到一个就不再继续查找。
- 如果是在子目录存储 (SASAUTOS) 中找到的宏，先编译并保存到 WORK.SASMACR 中再调用。这样，如果修改了子目录中的 SAS 宏需要重新人为运行其定义才能正确使用其新版本。

AUTOEXEC.SAS

- AUTOEXEC.SAS 是每次进入 SAS 时自动运行的一个 SAS 程序文件，可以在其中设定自己常用的设置，如宏库位置。
- 对于 MS Windows 版本的 SAS 系统，可以在自己的工作目录生成一个 SAS.EXE 的快捷方式，将其“起始位置”属性清空，然后把 AUTOEXEC.SAS 存放在此工作目录。
- AUTOEXEC.SAS 中内容如

```
libname samp '.\sampled';
options mautosource
    sasautos=(".\macros", sasautos);
```

其中定义了一个样例数据库，并设置了使用宏库及位置。

2.4.7 宏引文函数

宏引文函数

- SAS 宏的字符串替换很简单，但是也造成很多麻烦。
- 如何定义一个包含分号的宏变量？下面的例子肯定是错误的：

```
%LET printit=proc print;run;;
```

- 一个百分号到底应该解释为一个普通标点符号还是宏调用？
- 文本中的 OR 是某个州的缩写，还是逻辑计算的关键字？
- 需要用各种宏引文 (quoting) 函数来澄清。
- 最常用的是%STR, %NRSTR, %BQUOTE, %NRBQOTE, %SUPERQ。

宏引文函数分类

- 宏引文函数分别可以作用于宏的编译阶段或执行阶段。
- %STR, %NRSTR 作用于宏的编译阶段，%STR 保护自变量中的分号、撇号等特殊字符，%NRSTR 除此之外还保护% 和 & 不被解析。
- %BQUOTE, %NRBQUOTE 在把自变量中的替换执行完后，保护得到的替换结果不进行特殊字符的解释，其中%BQUOTE 不保护结果中的% 和 &，%NRBQUOTE 也保护结果中的% 和 & 不被解析。
- %SUPERQ 以宏变量名字本身为自变量（不需要引用），保护此宏变量的内容不进行特殊字符的解释，一般可以用%NRBQUOTE 取代。
- %UNQUOTE 可以把保护的文本开放进行解析。
- 这些函数中主要使用%BQUOTE 和%NRSTR 就可以应付大多数情况。

%STR 例子

- %STR 可以保护文本中的分号，不配对的撇号、括号、百分号等特殊字符，特殊字符需要用% 在前面标识出来：

```
%LET myvar=%STR(a%'); %PUT &myvar;
%LET myvar=%STR(b%"); %PUT &myvar;
%LET myvar=%STR(log%(12); %PUT &myvar;
%LET myvar=%STR(345%)); %PUT &myvar;

%LET myvar=%STR(90%%); %PUT &myvar;
%LET myvar=%STR(90%%%'); %PUT &myvar;

%LET printit=%STR(proc print; run;);
```

- %STR 中仍可以替换宏变量及宏，如

```
%LET var1=SAS Macro;
%LET var2=%STR(New &var1);
%PUT &var2;
```

%NRSTR 例子

- %NRSTR 保护文本中的特殊字符，并禁止宏变量替换和宏，如

```
%LET var1=SAS Macro;
%LET var2=%NRSTR(John%'s &var1);
%PUT &var2;
```

- 又如

```
%PUT This is the result of %NRSTR(%NRSTR);
```

%BQUOTE 例子

- %BQOTE 称为 “blind quoting”, 作用于宏的运行阶段，保护进行完宏变量和宏的替换之后的内容，而且不需要像%STR, %NRSTR 那样用百分号标识要保护的不配对撇号、括号等。如

```
%LET var1='abc';
%LET var2=%BQUOTE(%SUBSTR(&var1,1,3));
%PUT &var2;
```

- 此例中%SUBSTR 的结果返回三个字符'ab, 如果不加保护，对 var2 赋值时就有不配对的单撇号，导致程序出错，很难恢复。

- %BQUOTE 适合用来保护宏的未知的自变量值、用户输入、从数据集中获得的宏变量值。
- 比如，如果宏的自变量不用%BQUOTE 保护，在参与比较运算时就可以会产生歧义。设某个宏的自变量 dr 在运行时值为 x1-x3，宏中的某个比较为

```
%IF &dr NE %THEN ...;
```

运行时解析为

```
%IF x1-x3 NE %THEN ...;
```

自变量值中的减号就会被认为是减法运算而不是表示 x1 x2 x3 三个变量。程序应改为

```
%IF %BQUOTE(&dr) NE %THEN ...;
```

%NRBQUOTE 例子

- %NRBQOTE 是%BQUOTE 的非解析版本，它也作用于宏的运行阶段，保护进行完宏变量和宏的替换之后的内容，而且这些内容中及时有 & 和% 也不再进行替换。
- 如

```
data _null_;
  call symputx('var', 'B&G Coorporated');
run;
%LET com=%NRBQUOTE(&var);
%PUT &com;
```

数据步用 CALL SYMPUT 定义宏变量 var 时因为内容中有 &G 会在日志中警告没有宏变量 G，后面的%NRBQUOTE(&var) 会先解析其自变量 &var (这时并不保护 &)，然后把解析出的内容进行保护。

查看是否受到保护

- 宏引文函数保护过的内容一直保持被保护，但是用户不能直接显示出那些是被保护部分。
- 用

```
%PUT _user_;
```

在查看所有自定义宏变量时可以看到被保护部分。

解除保护

- 宏引文函数保护过的内容一直保持被保护，用%UNQUOTE 可以解除保护使其中的宏变量和宏调用可以再有效。
- 例如

```
%LET city=Beijing;
%LET oth=%nrstr(&city);
%LET unq=%unquote(&oth);
%put oth: &oth;
%put unq: &unq;
```

直接访问 oth 的内容时，它仍被保护，所以内容是 &city。用%unquote 解除保护后，unq 里保存的是 &city 不受保护从而解析为 Beijing 之后的结果。

宏引文函数小结

- 需要保护的文本内容包括：A 类（运算符）

```
; , + - * / > < = ^ | #
AND OR NOT EQ NE LE LT GE GT IN
```

- B 类：宏变量替换 & 或宏调用%。
- C 类：不匹配的单撇号、双撇号、括号。如果要在%STR 和%NRSTR 中保护这类字符需要用前导% 标记出来。

宏引文函数概要

函数	保护类型	作用时间
%STR	A	编译
%NRSTR	A, B	编译
%BQUOTE	A, C	运行
%NRBQUOTE	A,B,C	运行
%SUPERQ	A,B,C	运行

2.4.8 宏函数

宏函数

- 除宏引文函数以外，宏函数还可以：
- 处理宏变量中的文本，比如转换大小写、拆分数据项等；
- 对文本中的数学表达式进行计算；
- 执行 SAS 函数。
- 宏函数与数据步中的函数名字相似，作用也相似。但是注意宏函数仅针对原样文本和宏变量，不作用于数据步变量。

字符型宏函数

- %LENGTH 求长度。
- %UPCASE, %LOWCASE 转换大小写。
- %SUBSTR 求子串。%QSUBSTR 求子串并保护所得结果。
- %INDEX 查找子串首次出现位置。
- %SCAN 从保存了多项值的文本中拆分得到指定的一项。%QSCAN 作用类似但保护拆分结果。

%LENGTH 函数

- 宏变量存储内容只要不超过规定的最大长度就可以保存任意多个字符，但是在用%LET 定义是会被删去前导和尾随空格。用%LENGTH 求宏变量的长度。
- 如

```
%LET var1=Macro;
%LET VL=%LENGTH(&var1.);
%PUT &var1. 长度为 &vl.;
```

注意其中的%LENGTH 自变量用的是 var1 的宏引用格式。如果写成了%LENGTH(var1)，那么只能返回 var1 这 4 个字符的长度。

- 取空值的宏变量长度为 0，如

```
%LET var2=;
%LET VL2=%LENGTH(&var2.);
%PUT 宏变量 var2 长度为 &vl2.;
```

%UPCASE 和%LOWCASE 函数

- %UPCASE 和%LOWCASE 函数把自变量内容全部转换为大写或者小写。
- 在进行%IF 比较时如果想进行不区分大小写的比较，可以先用%UPCASE 把宏变量变成大写再与大写的文本比较，如

```
%IF %UPCASE(&city)=BEIJING %THEN ....;
```

%SUBSTR 函数

- %SUBSTR 取出第一自变量的子串，第二自变量指定开始位置，第三自变量为取出子串长度，省略第三自变量则取出剩余部分。
- 如

```
%LET ss=A brown fox;
%LET s1=%SUBSTR(&ss, 3, 5);
%PUT &s1;
```

%INDEX 函数

- %INDEX 函数从第一个自变量中查找第二个自变量的首次出现位置，找不到则返回 0。
- 如

```
%LET ss=A brown fox;
%LET ii=%INDEX(%BQUOTE(&ss), ow);
%PUT ow 在 &ss. 的第 &ii. 位置出现;
```

注意%INDEX 的第二个自变量 ow 没有用撇号界定。

%SCAN 函数

- %SCAN 函数从一个列表中取出其中某指定项。第一自变量是列表，第二自变量是序号，第三自变量省略时包括所有分隔符：

```
空格 . < ( + | & ~ $ * ) ; ^ - / , % > \
```

- 多个连续的分隔符看作一个。
- 例如

```
%LET var1=NO.1 student, excellent!;
%LET x1=%SCAN(%BQUOTE(&var1),1);
%LET x3=%SCAN(%BQUOTE(&var1),3);
%LET x4=%SCAN(%BQUOTE(&var1),4);
%LET xe=%SCAN(%BQUOTE(&var1),99);
%PUT &x1  &x3  &x4  &xe;
```

注意列表中有逗号，如果不保护起来就会与函数自变量之间分隔用的逗号相混淆。找不到的项返回空值。

- 如果要指定分隔符，可以用%STR 函数保护起来。
- 如

```
%LET var1=NO.1 student, excellent!;
%LET x = %SCAN(%BQUOTE(&var1), 2, %STR( ));
%PUT &var1. 中第二项为 &x.;
```

用了%STR 保护空格作为分隔符，用了%BQUOTE 保护自变量 &var1 的值（其中有逗号会干扰函数自变量的解读）。

用%QSCAN 函数遍历列表项

- %QSCAN 和%SCAN 类似，但保护其结果中的特殊字符如空格、逗号、不匹配的撇号、问号等。
- 下例演示了如何用循环遍历列表的每一项：

```
%MACRO exscan(source);
  %LET i=1;
  %LET x=%QSCAN(%BQUOTE(&source), &i, %STR( ));
  %DO %WHILE( &x ^= %STR());
    %PUT Item.&i  &x;
    %LET i=%EVAL(&i+1);
    %LET x = %QSCAN(%BQUOTE(&source), &i, %STR( ));
  %END;
%MEND exscan;
%LET var1=NO.1 student, excellent!;
%exscan(%BQUOTE(&var1))
```

宏文本中的计算

- SAS 宏主要依靠文本替换，但偶尔需要计算怎么办？
- 在宏 IF、宏循环的条件中，自动进行计算，如

```
%macro impcomp;
  %let x = 1+2;
  %IF &x=3 %THEN %put 自动计算表达式.;
```

```
  %ELSE %put 没有计算;
%mend impcomp;
%impcomp
```

其中定义宏变量 X 时并未计算加法，但在宏 IF 判断条件时则计算了加法结果。

- 但是，宏 IF、宏循环的条件中仅允许进行整数的运算，非整数的比较是按字符型比较处理的。如

```
%macro prb;
  %IF 10.0>9.0 %THEN %put 程序正确;
  %ELSE %put 不能进行非整数比较！;
%mend prb;
%prb
```

- 其它情况下，需要程序员自己用%EVAL 进行整数计算，用%SYSEVALF 进行实数计算和比较。
- 如：

```
%LET y1=2;
%LET y2 = 2+1;
%LET y3=%EVAL(&y2 + 1);
%LET y4=%SYSEVALF(4.8/2);
%PUT &y1 &y2 &y3 &y4;
```

- 上面宏 prb 的例子可以改为

```
%macro prbc;
  %IF %sysevalf(10.0>9.0) %THEN %put 程序正确;
  %ELSE %put 不能进行非整数比较！;
%mend prbc;
%prbc
```

用%SYSFUNC 调用数据步函数

- 用%SYSFUNC 可以调用许多数据步函数，这样很多只是为了调用某个函数进行计算的小数据步就可以省略。
- 格式为%SYSFUNC(数据步函数名 (参数表), < 输出格式 >)。数据步函数不允许嵌套调用。为保护其结果可以改用%QSYSFUNC。
- 如：

```
%LET x=3.14159/2;
%LET y=%SYSFUNC(sin(&x), 6.2);
%PUT &y;
```

- %SYSFUNC 不能使用数据步 PUT 函数，为了按指定格式转换数值可以用 PUTN 函数。PUTN 函数作用与 PUT 函数类似，但允许指定输出格式为数据步运行时才确定的一个字符型变量或字符型表达式的值。
- 如：

```
%LET s=%SYSFUNC(putn(1,Z3.));
%PUT &s;
```

- 宏函数的文本型自变量不需要用撇号界定，所以这里 PUTN 的自变量 Z3. 没有用撇号界定。
- 在%SYSFUNC 中用 PUTN 转换日期格式例：

```
%LET s=%SYSFUNC(putn("&sysdate9"d, yymmdd10.));
%PUT &sysdate9 &s;
footnote " 演示结果: "
  "%SYSFUNC(putn("&sysdate9"d, yymmdd10.))";
```

2.4.9 自定义宏函数

用宏变量传递结果

- SAS 宏相当于子程序，通常用来生成一段代码，有时我们需要返回一个简单结果。
- 一种办法是调用宏，在宏中定义全局变量并赋值，在后续程序中访问此宏变量值。这样比较繁琐，也容易发生同名全局变量冲突。
- 例如，为了查看某个数据集是否存在，用了如下定义全局变量的方法：

```
%macro dsexist(dsn);
  %global exist;
  %if &dsn ne %then %do;
    data _null_;
      stop;
      set &dsn;
    run;
    %end;
  %if &syserr=0 %then %let exist=1;
  %else %let exist=0;
%mend dsexist;
```

- 其中的数据步 STOP 表示数据步运行时停止，但 SET 语句在编译时发现要打开的数据集不存在会把自动宏变量 SYSERR 置为非零值。
- 使用方法如

```
%dsexist(sasuser.mydat)
data new;
  if &exist then dsname='sasuser.mydat';
run;
```

- 可以看出这样使用是很不方便的。

自定义宏函数

- 可以设法定义一个 SAS 宏，使其解析时仅解析成要返回的值。这样的 SAS 宏可以当作是自定义的宏函数。
- 为此，宏定义中只能有宏语句，不能有数据步和过程步，否则解析后的结果中也会有这些程序段。如果需要调用数据步中的功能可以尝试用%SYSFUNC 函数代替。
- 这样的宏应该只使用局部宏变量，而不应该给宏变量赋值。

%exist 宏函数

- 比如，下面的例子定义了一个一般性的判断数据集是否存在的宏函数%exist:

```
%macro exist(dsn);
  %sysfunc(exist(&dsn))
%mend exist;
```

- 使用如

```
%macro test(dsn);
  %if %exist(&dsn) %then
     %put &dsn. 数据集存在;
  %else %put &dsn. 数据集不存在;
%mend test;
%test(samp.class)
%test(samp.notexist)
```

%currdate 宏函数

- 下面的宏定义了一个以年月日形式返回当前日期的宏函数：

```
%macro currdate;
  %qsysfunc(date(), yymmdd10.)
%mend currdate;
```

- 使用如

```
title 'C9501 数据集列表';
footnote " 演示日期: %currdate";
proc print data=samp.c9501;run;
```

2.4.10 宏与数据步的信息交换

宏与数据步的信息交换

- SAS 的工作模式不容易利用前面的计算结果。
- SAS 的宏功能可以部分弥补 SAS 的这一缺陷;
- 我们可以用宏变量在不同的数据步、过程步之间传递信息。
- 在数据步中用 CALL 调用 SYMPUTX 子程序可以把数据步中当前变量值传递给一个宏变量，格式为:

  CALL SYMPUTX(字符型表达式, 表达式);

  其中的字符型表达式的值为要生成的宏变量名。两个自变量的前导空格和尾随空格自动被删除。
- 要从数据步中访问宏变量的值，可以直接使用宏替换，也可以把宏替换结果用于变量赋值、RETAIN 语句初值。
- 在数据步中用 SYMGET 函数可以根据数据步中当前变量值动态访问宏变量，格式为:

  SYMGET(宏变量名字符型表达式);

  其中的字符型表达式的值为要访问的宏变量名。

SYMPUTX 例子 I

```
%MACRO spe1;
data books;
  set samp.c9501bk end=lastobs;
  ta + amount;
  if lastobs then do;
    call symputx('nobs', _n_);
```

```
    call symputx('total', ta);
  end;
  drop ta;
run;
footnote "&nobs. 个人共花费 &total. 元";
proc print;run;
footnote;
%MEND spe1;

%spe1
```

- 上例中用数据步读入了 SAMP.C9501BK 数据集，并计算购书金额累计栏，到最后一个观测时（用 SET 语句的 END= 指定一个临时标志变量 LASTOBS）用 SYMPUTX 把观测个数和总金额存分别写入宏变量中，然后在后续的脚注中使用。

- 这里的两个 CALL SYMPUTX 不能用这样的%LET 代替：

```
.........
  if lastobs then do;
    %LET nobs = _n_;
    %LET total = ta;
  end;
.........
```

  这是因为宏语句的运行在数据步编译运行之前早已完成。这样定义的宏变量 NOBS 只能包含原样的_n_ 三个字符，而不是观测个数。

- 注意: SYMPUTX 是一个数据步子程序（用 CALL 调用）。其语法不使用宏语言的语法而是数据步的语法。

- 注意：在数据步中用 SYMPUTX 定义的宏变量不能用在本数据步中。因为宏变量替换发生在数据步编译运行之前。

- 为了使得存入宏变量中的值符合某种要求的格式，可以使用数据步的 PUT 函数，比如

```
call symputx('total', put(ta, 8.2));
```

- 已经运行完的数据步用 SYMPUTX 定义的宏变量可以在后面被替换使用，或在后续数据步中用 SYMGET 函数动态访问。

SYMPUTX 例子 II

```
data aa;
  input cid city $;
  cards;
110 北京
230 上海
;

%MACRO spe2;
data _null_;
  set aa;
  call symputx('city' || put(cid,best3.),
         city);
run;
%put &city110 &city230;
%MEND spe2;

%spe2
```

在数据步中静态访问宏变量值

- 在数据步中如果要访问某个已知变量名的宏变量，只要直接用宏替换的方式。
- 如

```
%LET nmax=5;
DATA aa;
  array x(&nmax);
```

```
  do i=1 to 20;
    do j=1 to &nmax;
      x(j) = normal(112233);
    end;
    output;
  end;
run;
```

- 可以把宏变量替换后赋值给数据集变量，如

```
%LET nobs=5;
DATA ab;
  n = &nobs;
  do i=1 to n;
    j = i*i; output;
  end;
run;
```

- 或把宏变量替换的值作为 RETAIN 语句初值，如

```
%LET nobs=5;
DATA ac;
  retain n &nobs;
  do i=1 to n;
    j = i*i; output;
  end;
run;
```

SYMGET 例子

```
data bb;
  input cid temp;
  cards;
110 20
```

```
230 30
;

%MACRO spe3;
data bc;
  set bb;
  city = symget('city' || put(cid, best3.));
run;
proc print;run;
%MEND spe3;

%spe3
```

- 上例说明了 SYMGET 函数的必要性：根据数据的变化而选取不同的宏变量的值。
- 如果直接用 & 格式访问宏变量，则宏变量名必须预先能够确定。
- SYMGET 在许多情形下有替代方法。很多情况下宏变量名是已知的，只要用宏替换；另外一些情况下可以用数据合并来获取信息，而不必要用宏变量在多个数据步之间传递信息。
- SYMGET 的自变量可以是：
  - 内容为宏变量名的字符型常量（写在单撇号或双撇号之间）；
  - 保存了宏变量名的字符型变量；
  - 结果为宏变量名的字符型表达式。
- SYMGET 的返回一个字符串，长度缺省为 200 个字符。如果需要数值可以用 INPUT 函数进行转换。

2.4.11 读取多个文件的例子

读取多个文件

- 实际中经常需要读取多个格式相同的文件，可以是文本型、Excel、CSV 等格式。
- 这些文件如果文件名是编号的，如 file1.txt, file2.txt 等，可以很容易地用宏循环生成文件名用 INFILE 等读取。

- 如果文件名是不规则的，可以先生成包含文件名列表的数据集，然后用宏循环从数据集中每次读取一行，用 CALL SYMPUTX 把当前文件名存放到宏变量中再用 INFILE 等读取。

读取多个 Excel 文件

- 假设需要读取多个 Excel 文件，这里给出一种用 ODBC 和 LIBNAME 读取的方法。
- 拿其中的一个作为模板，在 Windows 的控制面板的管理工具中找到数据源管理，添加一个数据源，如 excelsrc，数据源中对应的文件假设叫做 sample.xlsx。
- 假设文件名列表在数据集 INDEXF 中，变量名是 FILENAME。
- 用宏循环地读取每个文件名，在数据步中生成一个 DOS 命令并用 CALL SYMPUTX 存为宏变量。用 SAS 的 X 命令把当前文件复制为 sample.xlsx，用 LIBNAME 语句访问文件内容。访问完后用仅有逻辑库名的 LIBNAME 语句关闭连接。
- 程序如下：

```
OPTIONS XWAIT;
%MACRO multifile;
%DO i=1 %TO 10;
data _null_;
  set indexf(firstobs=&i obs=&i);
  cmd = 'copy ' || "'" filename "' "
    'sample.xlsx & exit';
  call symputx('cmd', cmd);
run;
x &cmd;
libname myexcel ODBC DATASRC=excelsrc;
data d&i;
  set myexcel.'$Sheet1'n;
run;
libname myexcel;
%MEND multifile;
```

2.4.12 动态程序

动态程序

- 好的 SAS 宏程序应该具有通用性：程序中应该不依赖于具体的数据集名、变量名、数据集观测个数、变量个数等信息。
- 称这样的程序为动态程序。有很多措施可以帮助我们写出动态程序：
  - 用控制数据集提供具体信息；
  - 用 SYMPUTX 或 PROC SQL SELECT INTO 的方法动态生成宏变量;
  - 使用诸如 &&vars&i 这样的宏变量数组，数组元素个数也保存为宏变量；
  - 用%DO 循环遍历宏变量数组。

例：用数据集控制程序运行

- 有时程序运行的控制参数保存在一个控制数据集中。这样的数据集可能是人为输入的，也可能是计算得到的。
- 例如

```
data cntr;
  length dsname $ 20;
  input dsname $ nobs;
  cards;
samp.class 5
;
run;
```

  生成了一个控制数据集。

- 使用控制数据集如

```
%MACRO controled;
  data _null_;
    set cntr;
    call symputx('dsn', dsname);
```

```
    call symputx('nobs', nobs);
  run;
  title "Dataset &dsn";
  proc contents data=&dsn.;
  run;
  proc print data=&dsn.(obs=&nobs.);
  run;
  title;
%MEND controled;
%controled
```

- 从控制数据集中把控制参数写入了宏变量，然后用宏变量值控制后续程序运行。

例：用多行数据集控制程序运行

- 控制参数可以有若干组，分别存放在数据集的不同行中。
- 如

```
data cntr;
  length dsname $ 20;
  input dsname $ nobs;
  cards;
samp.class 5
sashelp.air 8
;
run;
```

- 使用多组参数控制多组运行，生成多个宏变量如

```
%MACRO controled;
  data _null_;
    set cntr end=lastobs;
    call symputx('dsn' ||
      trim(left(put(_n_, best12.))), dsname);
```

```
    call symputx('nobs' ||
      trim(left(put(_n_, best12.))), nobs);
    if lastobs then
       call symputx('npars', _n_);
  run;
  .........
```

- 这里 symputx 调用中用数据步隐含循环计数变量_n_ 构造了一组宏变量名 dsn1, dsn2, nobs1, nobs2。在数据集最后一个观测时获得了行数存入宏变量 npars 中。

- 访问如 dsn1 这样的宏变量需要用宏循环和双 & 格式，如：

```
  %DO ipar=1 %TO &npars;
    title "Dataset &&dsn&ipar";
    proc contents data=&&dsn&ipar;
    run;
    proc print data=&&dsn&ipar
         (obs=&&nobs&ipar);
    run;
  %END;
  title;
%MEND controled;
options symbolgen mprint;
%controled
```

- 用了双 & 方法来访问一组 dsn1, dsn2 这样的宏变量, 相当于宏变量数组。
- 编译运行时，进行两遍扫描，两个 & 符先替换成一个 & 符，然后再进行宏变量替换。
- 比如，当宏变量 ipar 等于 2 时，&&dsn&ipar 先替换成 &dsn2, 然后替换成宏变量 dsn2 的值 sashelp.air。
- 用了宏循环处理数据集的多个观测，这样的程序适用于控制数据集有任意多个观测且不能预知的情况。

包含宏变量名和宏变量值的控制数据集

- 控制数据集中可以同时保存宏变量名和对应宏变量值，如

```
data cntr;
  length varname varval $ 40;
  input varname $ varval $;
  cards;
dsn   samp.class
;
run;
```

- 生成并访问相应的宏变量如

```
%macro controled;
  data _null_;
    set cntr;
    call symputx(varname, varval);
    call symputx('varname', varname);
  run;
  %PUT varname is &varname;
  %PUT &varname stores &&&varname;
%mend controled;
%controled
```

- 程序中第一个 call symputx 定义宏变量 dsn 的值为 samp.class，第二个 call symputx 定义宏变量 varname 为 dsn。
- %PUT 语句中的 &&&varname 首先替换为 &dsn, 然后再替换为宏变量 dsn 的值 samp.class。

包含多组宏变量名和宏变量值的控制数据集

- 控制数据集中也可以同时保存多个宏变量名和对应宏变量值，如

```
data cntr;
  length varname varval $ 40;
  input varname $ varval $;
  cards;
dsn   samp.class
nobs  5
;
run;
```

- 从控制数据集生成宏变量如

```
%macro controled;
  data _null_;
    set cntr end=lastobs;
    call symputx(varname, varval);
    call symputx('vars' ||
        trim(left(put(_n_, 8.))), varname);
    if lastobs then
        call symputx('nvars', _n_);
  run;
  .........
```

- 数据步隐含地循环 2 次。第一个 call symputx 分别定义宏变量 dsn 为 samp.class, 定义宏变量 nobs 为 5; 第二个 call symputx 分别定义宏变量 vars1 为 dsn, 宏变量 vars2 为 nobs。

- 访问宏变量如

```
  .........
  %DO i=1 %TO &nvars;
    %PUT vars&i is &&vars&i;
    %LET tmpname=&&vars&i;
    %PUT &&vars&i stores &&&tmpname;
  %END;
```

```
%mend controled;
%controled
```

- 这里宏循环执行 2 次。当宏变量 i=2 时，&&vars&i 解析为 &vars2 然后再解析为 nobs。
- 这时，&&&vars&i 不能访问宏变量 nobs 的值 5 (如果宏变量 vars 有值 xyz, &&&vars&i 先替换成 &xyz1 再试图解析宏变量 xyz1)。
- 用了临时宏变量 tmpname 来保存 nobs，然后用 &&&tmpname 先解析为 &nobs 再解析为 samp.class。

- 这个例子中，如果不需要按照控制数据集的要求生成宏变量（宏变量 dsn 定义为 samp.class, 宏彬和 nobs 定义为 5），就可以把数据集的两列分别定义为两个宏变量数组，其访问就可以不用三个 & 的格式。

2.4.13 用 PROC SQL 生成宏变量组

用 PROC SQL 生成宏变量

- 可以用 PROC SQL 的 SELECT INTO 方法生成宏变量, 用冒号引入宏变量名。
- 从一行的各列或统计结果生成宏变量，如

```
proc sql noprint;
  select count(*),
         sum(amount) format=best5.
    into :nstudent, :total
         from samp.c9501bk;
quit;
%put &nstudent. &total.;
```

- PROC SQL 的 NOPRINT 选项不自动显示查询结果。
- INTO 后面的宏变量名用前导冒号标识。

- 从多行的各列生成宏变量，如

```
proc sql noprint;
  select name, amount
    into :name1-:name9999,
         :amount1-:amount9999
    from samp.c9501bk;
  %let nstudents=&sqlobs;
quit;
%put &nstudents;
%put &name3 &amount3;
```

- INTO 后面代表同一列值的宏变量组用减号构造。虽然指定了变量名可以到 name9999，但是实际只生成有值的那些。
- 宏变量 sqlobs 在执行完一个 SQL SELECT 后保存查询结果行数。
- 从多行的各列生成宏变量，也可以把一个变量各行的值用空格分隔放入一个宏变量中，如

```
proc sql noprint;
  select name, amount
    INTO :names SEPARATED BY ' ',
         :amounts SEPARATED BY ' '
        FROM samp.c9501bk;
quit;
%put &names --- &amounts;
```

- 使用了 SEPERATED BY 格式。
- 也可以使用逗号等作分隔。

空格分隔列表利用

- 宏变量中用空格分隔的列表可以用%SCAN 函数逐个提取，直到提取到空值为止。
- 例：

```
%macro vl(vars);
  %do ivar=1 %to 100000;
    %let val=%scan(&vars., &ivar.);
    %if &val.= %then %goto endd;
    %put NO.&ivar.  &val;
  %end;
  %endd:
%mend;

%vl(&names)
```

2.5 用 proc sql 管理数据

2.5.1 简单查询

用 proc sql 管理数据

- SAS 系统除了统计分析之外还有很强对数据管理功能。
- SQL 语句是大型关系数据库管理专用语言，流行的大型数据库系统都支持 SQL 语言。
- SAS 的 SQL 过程实现了 SQL 语言。
- SAS 的 SQL 过程可以：
  - 从一个或多个表中查询信息；
  - 生成表；
  - 向表中插入行；
  - 更新表的内容；
  - 对表进行纵向合并、横向连接；
  - 直接连接外部数据库，等等。

SQL 查询

- 用 PROC SQL 作查询的最简单的用法如下:

```
PROC SQL;
    SELECT 第一项, 第二项, …, 第 n 项
        FROM 数据集
        WHERE 观测选择条件;
RUN;
```

例：全部内容

- 变量中用 * 表示全部变量，不加 WHERE 子句表示全部观测。
- 如

```
proc sql;
  select *
    from c9501;
quit;
```

- 如果数据集观测过多，可以在 PROC SQL 语句中加 OUTOBS= 选项指定输出的观测个数。

例：列子集

- SELECT 子句中指定若干变量，变量名之间以逗号分隔！
- 如

```
proc sql;
  select name, math
    from c9501;
quit;
```

例：行子集

- 用 WHERE 子句指定选择行子集的条件。
- 如

```
proc sql;
  select name, math
    from c9501
    WHERE sex=' 男';
quit;
```

例：DISTINCT

- 在变量名前面加上 DISTINCT 前缀则此变量的不同值只保留一次。
- 如

```
proc sql;
  select DISTINCT SEX
    from c9501;
quit;
```

- 如果 DISTINCT 后面有多个变量名则取所有不同组合。

例：计算新变量

- 在 SELECT 子句中用表达式，AS 关键字和新变量名定义新变量并显示。
- 如

```
proc sql;
  select name, math+chinese AS total
    from c9501;
quit;
```

- 如果 WHERE 子句中使用了新变量，需要用 CALCULATED 前缀修饰，如

```
proc sql;
  select name, math+chinese AS total
    from c9501
    where CALCULATED total>=200;
quit;
```

用 ORDER BY 子句排序

- 在 SELECT 语句中可以加入 ORDER BY 子句为查询结果排序。
- 例:

```
proc sql;
  select name, math
    from c9501
    where sex=' 男'
    ORDER BY math DESC;
quit;
```

- 对需要降序排列的变量，在变量名后加选项 DESC。

2.5.2 分组汇总

用 GROUP BY 子句分组汇总

- 在 SELECT 语句中可以加入 GROUP BY 子句分组，在组内用 SUM, MEAN, MAX, STD 等统计函数进行统计。
- 例:

```
proc sql;
  select sex, mean(math) as mm
    from c9501
    GROUP BY sex;
quit;
```

用 HAVING 子句按汇总结果选择

- 如果进行了分组，则只能使用 HAVING 子句选择子集。
- 例:

```
proc sql;
  select sex, mean(math) as mm
    from c9501
    GROUP BY sex
    HAVING mean(chinese)>=100;
quit;
```

- 结果为不同性别的平均数学成绩，但只显示平均语文成绩达到 100 分的组。

2.5.3 生成和删除数据集

生成数据集

- 用 CREATE TABLE 语句把查询结果存入数据集中。
- 例:

```
proc sql;
  CREATE TABLE subd AS
    select name, math
      from c9501
      where sex=' 男'
    order by math desc;
  select * from subd;
quit;
```

生成视图

- 用 CREATE VIEW 语句把查询结果存入数据集中。
- 例:

```
proc sql;
  CREATE VIEW totd AS
```

```
    select name,
        math+chinese AS total
      from c9501;
quit;
proc print data=totd;run;
```

- 视图的用途类似于数据集，但是不实际保存数据，而只保存数据的访问和生成方式。

删除表或视图

- 用 DROP TABLE 表名删除表（数据集）。
- 用 DROP VIEW 视图名删除视图。

2.5.4 连接

左右合并：一对一

- SQL 中 SELECT 的强大查询功能还表现在它可以从几个表联合查询。
- 例如，C9501X 数据集包含学生姓名、性别，C9501Y 数据集包含学生姓名、数学成绩、语文成绩，要显示男生的学生姓名、数学成绩:

```
proc sql;
  create table c9501x AS
    select name, sex from c9501;
  create table c9501y AS
    select name, math, chinese from c9501;
quit;
```

- 下面程序横向连接两个数据集进行查询:

```
proc sql;
  select a.name, math
    from c9501x AS a, c9501y AS b
```

```
    where a.name=a.name
          and sex=' 男'
    order by math desc;
quit;
```

- 在 FROM 子句中指定了两个数据集, 两个数据集之间用逗号分隔，在数据集名后面加 AS 关键字然后可以跟随一个临时别名；
- 在 WHERE 子句中要求按 name(姓名) 连接两个数据集，还可以加其它行子集条件。
- 共有的变量名应该用“别名. 变量名”的格式显式指出。

一对一合并时 SQL 与数据步的区别

- SQL 的一对一合并只包含匹配观测，不匹配观测被删除。
- 数据步中用 MERGE 和 BY 做一对一合并，缺省情况下既包含匹配观测，也包含所有不匹配的观测。
- SQL 中不需要输入数据集按关键列排序。
- 数据步中横向合并时需要输入数据集按关键列排序。
- SQL 中 SELECT 语句只显示合并的结果，并不生成数据集。需要生成数据集时可以把 SELECT 语句嵌入到 CREATE TABLE 语句中。
- 数据步中横向合并会生成输出数据集，但不显示。
- 为了在数据步中实现只有匹配观测的合并，可以用数据集选项 IN。如

```
proc sort data=c9501x; by name;
proc sort data=c9501y; by name;
data new;
  merge c9501x(IN=_a) c9501y(IN=_b);
  by name;
  IF _a AND _b;
run;
```

左右合并：一对多

- 数据例：

d1		d2	
ID	X	ID	Y
a	11	a	21
b	12	a	22

- 合并

```
proc sql;
  select d1.id, x, y
    from d1, d2
    where d1.id=d2.id;
quit;
```

- 结果有 (a, 11,21), (a, 11, 22) 两行。
- 不匹配行被丢弃。

一对多左右合并时 SQL 与数据步的区别

- 数据步中用 MERGE 和 BY 语句也能进行一对多的合并。
- 在 SQL 中，不匹配行被丢弃。
- 在数据步中，匹配和和不匹配行都保留。

左右合并：笛卡儿积

- 多个观测和多个观测的合并是笛卡儿积式的两两组合，如

```
data d1;
  input x @@; cards;
11 12
;
data d2;
  input y @@; cards;
21 22
```

```
;
proc sql;
  select x, y from d1, d2;
quit;
```

- 结果有 (11,21), (11,22), (12,21), (12,22) 四行。

左右合并：多对多

- 按关键变量左右合并时，数据集观测按关键变量分组，如果某组出现多对多合并，则左右两边按笛卡儿积方法（所有两两组合）合并。
- 数据例：

d3		d4	
ID	X	ID	Y
a	11	a	21
b	12	a	22
a	13	b	23
		a	24

- 合并

```
proc sql;
  select d3.id, x, y
    from d3, d4
    where d3.id=d4.id;
quit;
```

- 结果 a 有 6 行 (d3 的两行与 d4 的三行两两搭配)，b 有一行。

多对多左右合并时 SQL 和数据步的区别

- 在进行一对一和一对多合并时，用 SQL 合并与用数据步合并的结果差别不大。
- 在多对多合并时，两种方法结果截然不同。

- 数据步多对多合并的规则是：按关键变量值分组后，同组内左右数据集的观测按次序对应合并，但是如果一边的观测用完的时候沿用组内最后一个观测的值。

- 比如，用数据步对 d3 和 d4 横向合并，程序如：

```
proc sort data=d3; by id;
proc sort data=d4; by id;
data new;
  merge d3 d4;
  by id;
run;
proc print;run;
```

结果 a 组只有 3 个观测。

横向合并：不匹配观测

- SQL 连接会把不匹配的观测排除在外。

- 如数据

d5		d6	
ID	X	ID	Y
a	11	a	21
b	12	c	22

- 连接

```
proc sql;
  select d5.id, x, y
    from d5, d6
    where d5.id=d6.id;
quit;
```

- 结果只有 a 的行。

横向合并：左外连接

- 连接时为了包含左侧表的所有观测，使用 LEFT JOIN 来指定两个表，用 ON 指定连接条件。
- 如

```
proc sql;
  select d5.id, x, y
    from d5 LEFT JOIN d6
    ON d5.id=d6.id;
quit;
```

横向合并：右外连接

- 类似地, 用 RIGHT JOIN 来指定两个表则保持右侧表的所有观测。
- 如

```
proc sql;
  select d6.id, x, y
    from d5 RIGHT JOIN d6
    ON d5.id=d6.id;
quit;
```

横向合并：全外连接

- 用 FULL OUTER JOIN 来指定两个表则保持两个表的所有观测。
- 如

```
proc sql;
  select COALESCE(d5.id, d6.id) AS id, x, y
    from d5 FULL OUTER JOIN d6
    ON d5.id=d6.id;
quit;
```

- 用 COALESCE 函数合并关键列。

2.5.5 一些技巧

SQL 比较条件

- SQL 中可以使用数据步中的比较运算：

=	^=	>	<	>=	<=	IN
EQ	NE	GT	LT	GE	LE	

- 还有一些其它的比较运算。
- CONTAINS 表示字符串包含某子串。如

```
where title CONTAINS 'cat'
```

- 用 IS NULL 表示“是缺失值”，在 SQL 数据库的表中是“空值”。
- 用 BETWEEN ... AND ... 表示在某个范围内，包括两端的值。如

```
where salary between 7000 and 8000
```

- 可以用 LIKE 表示符合某种模式。在模式中，用“%”表示任意多个任意字符，如：

```
where LastName LIKE 'H%'
```

  可以匹配 Huang, Hong 等。

- 在模式中，用一个下划线代表一个任意字符，如：

```
where LastName LIKE 'S_n'
```

  可以匹配 Sun, Son, San 等。

- 用“=*”表示“发音相似”，仅针对英文，如

```
where LastName =* 'SMITH'
```

  除了 SMITH 本身以外还可以匹配 SMITT, SMYTHE, SMOTHE。

PROC SQL 选项

- 在要查询的数据表特别巨大时，往往要先小规模测试查询是否正确。
- 在 PROC SQL 语句中，用 INOBS=n 指定输入每个表仅允许输入 n 行，用 OUTOBS=n 指定输出结果时仅允许输出不超过 n 行，用 LOOPS=n 指定内层循环的最多循环次数。

2.5.6 高级技巧示例

子查询

- 可以把 SELECT 的结果作为一个数据集看待，用于其他查询的 WHERE 子句。
- 如

```
proc sql;
  select name, math
    from c9501 AS a
    where sex IN
      (SELECT sex
       from c9501 AS b
       group by sex
       having mean(chinese)>=100);
quit;
```

- 按男女生分组，只显示语文成绩组平均达到 100 分的学生。

例子: 找同生日

- 设有 C9501 数据集中 5 个人的生日数据如下：

```
title ' 找出生日相同的人';
data cb;
  input name $ 1-8 birth yymmdd10.;
  format birth yymmdd10.;
  label name=' 姓名'  birth=' 生日';
```

```
  cards;
李明       78-6-1
王思明     78-5-19
张聪       78-6-1
刘颖       78-10-18
张红艺     79-5-19
;
```

- 可以使用如下子查询方法：

```
proc sql;
  select name, birth
    from cb AS a
    where birth in (
      select birth
        from cb AS b
        where b.name ^= a.name)
    order by a.birth;
run;
```

- 这里，外层的 SELECT 对别名 A 的数据集 CB 的 5 行进行外层循环，内层的 SELECT 对别名 B 的数据集 CB 中名字与外层当前名字不同的 4 行进行内层循环，外层的 WHERE 子句的条件是要求外层循环中当前观测的生日输入内层循环中找出的其它人生日集合。

- 还可以使用如下横向合并方法:

```
proc sql;
  select a.name, a.birth
    from cb AS a, cb AS b
    where a.name ^= b.name
          and a.birth=b.birth
    order by a.birth;
quit;
```

- 这里，合并的两个数据集是相同的，因为合并是按照笛卡儿积方式（所有两两组合）进行的所以实现了任意两个人之间的比较，WHERE 子句要求同一个人自己不要比较，选择生日相同的人作为结果。

仅需月日相同时的处理

- 如果生日相同只需要生日的月和日相同，可以首先生成一个仅包含月、日信息的新变量，然后按此新变量查询。
- 如

```
proc sql;
  create view datamd as
    select name, birth,
      month(birth)*100+day(birth) AS md
      from cb;
  select name, birth
    from datamd AS a
    where md in (
      select md
        from datamd AS b
        where b.name ^= a.name)
    order by a.md;
  drop view datamd;
run;quit;
```

例: 不用 SQL 找同生日

- 找生日的问题也可以用其它方法完成。想法是，统计数据集中每个生日的出现次数，显示次数在 2 次以上的那些生日和对应的姓名。
- 首先用 PROC FREQ 统计生日的出现次数，保存到数据集 BFREQ 中：

```
title ' 找出生日相同的人: PROC FREQ';
proc freq data=cb noprint;
  tables birth / out=bfreq;
run;
```

- 这时，数据集 BFREQ 中仅有生日和重复次数，没有相应的姓名。需要把姓名信息与生日重复次数的信息横向合并，并只合并有重复生日的观测：

```
proc sort data=cb; by birth;
proc sort data=bfreq; by birth;
data bsame;
  merge cb bfreq;
  by birth;
  IF count>1;
run;
```

- 横向合并时用了子集 IF 语句来选择输出观测。注意这个选择不能改为对 BFREQ 的输入数据集选项，因为 MERGE BY 合并会把不匹配观测也包含进来。

- 然后显示合并后的内容：

```
proc print data=bsame label noobs;
  var name;
  by birth;
  id birth;
run;
```

- 最后，用 PROC DATESETS 的 DELETE 语句删除不再需要的数据集：

```
proc datasets library=work;
  delete bfreq bsame;
run;
```

第三章　SAS 功能基础

3.1 SAS 过程初步

SAS 过程初步

- SAS 过程步的一般用法和常用语句的含义;
- SAS 的简单报表、分析、绘图功能的使用;
- 介绍 SAS 的分析员 (Analyst) 模块。

3.1.1 SAS 过程用法

SAS 过程用法

- SAS 过程步的一般形式为：

```
PROC 过程名 DATA= 输入数据集 选项;
    过程语句 [/ 选项];
    过程语句 [/ 选项];
    ……
RUN;
```

- 其中 PROC 语句的“选项”是可选的, 用来规定过程运行的一些设置, 如果有多个选项用空格分开。
- 过程步一般以 RUN 语句结束，有些“交互式过程”以 QUIT 语句结束；其他数据步或过程步也可以结束上一个数据步或过程步。

过程语句

- 过程步可以包含过程语句。
- 过程语句与数据步中的语句不同, 数据步中的语句不能用在过程步中。
- 过程步语句一般以某一个关键字开头, 比如 VAR、BY、TABLES、WEIGHT 等, 语句中提供一些必要参数，如果有可选的选择项的话一般写在斜杠后。

过程输出

- 有些过程只进行变换，不生成显示结果；
- 多数过程步有结果显示在 OUTPUT 窗口；

- 绘图过程结果显示在 GRAPHICS 窗口；
- 对 OUTPUT 窗口的结果可以保存、复制、打印；
- 新版 SAS 支持 HTML 格式的输出。

3.1.2 SAS 过程步常用语句

VAR 语句

- VAR 语句在很多过程中用来指定分析变量。
- 在 VAR 关键字后面给出变量列表：

  VAR 变量名 1 变量名 2 …变量名 n;

- 变量名列表可以使用省略的形式, 如 X1−X3, MATH−−CHINESE 等。
- 例如：

```
var math chinese;
```

MODEL 语句

- MODEL 语句在一些统计建模过程中用来指定模型的形式。
- 一般形式为

  MODEL 因变量 = 自变量表 / 选项;

- 例如:

```
model math=chinese;
```

即用语文成绩预测数学成绩。

BY 语句

- BY 语句在过程中一般用来指定一个或几个分组变量, 根据这些分组变量值把观测分组, 然后对每一组观测分别进行本过程指定的分析。
- 在使用带有 BY 语句的过程步之前一般先用 SORT 过程对数据集排序。
- 比如, 下列 SORT 过程和 PRINT 过程可以把男、女生分别列出：

```
proc sort data=c9501;
  by sex;
run;

proc print data=c9501;
  by sex;
run;
```

CLASS 语句

- 在一些过程 (如 ANOVA) 中, 使用 CLASS 语句指定一个或几个分类变量。
- 在另一些过程 (如 MEANS) 中, CLASS 语句作用与 BY 语句类似, 可以指定分类变量, 把观测按分类变量分类后分别进行分析。
- 使用 CLASS 时不需要先按分类变量排序。

OUTPUT 语句

- 常用 OUTPUT 语句指定输出结果存放的数据集。
- 不同过程中把输出结果存入数据集的方法各有不同, OUTPUT 语句是用得最多的一种, 其一般格式为：

  OUTPUT OUT =输出数据集名
  关键字=变量名 关键字=变量名 ...;

  或

OUTPUT OUT =输出数据集名
关键字= 关键字= .../ AUTONAME;

- 用 OUT =给出了要生成的结果数据集的名字;
- 用 "关键字=变量名" 的方式指定了输出哪些结果并给结果变量命名, 或不指定结果变量名但用 AUTONAME 选项自动命名。

OUTPUT 语句例子

```
proc means data=sasuser.c9501;
  var math;
  output out=result n=n
         mean=meanmath var=varmath;
run;
proc print data=result; run;
```

FREQ 语句

- FREQ 语句指定一个重复数变量, 每个观测中此变量的值说明这个观测实际代表多少个完全相同的重复观测。FREQ 变量只取整数值。
- 如

```
freq numcell;
```

- WEIGHT 语句指定一个权重变量, 在某些允许加权的过程中代表权重, 其值与观测对应的方差的倒数成正比。

ID 语句

- 有些过程 (如 PRINT、UNIVARIATE) 需要输出观测的代号, 这一般使用观测的序号。
- 如果数据集中有一个变量可以用来区分观测 (如人名、省市名), 就可以用 ID 语句指定这个变量作为观测标识, 如:

```
id name;
```

WHERE 语句

- 用 WHERE 语句可以选择输入数据集的一个行子集来进行分析, 在 WHERE 关键字后指定挑选行的条件。
- 比如

```
where math>=60 and chinese>=60;
```

- 指定只分析数学、语文成绩都及格的学生。

LABEL 语句

- 过程步中的 LABEL 语句为变量指定一个临时标签, 很多过程可以使用这样的标签。
- LABEL 语句的格式为

  LABEL 变量名＝' 标签' 变量名＝' 标签' ...;

- 例如

```
proc print data=sasuser.c9501 label;
  id name;
  var math chinese;
  label name=' 姓名' math=' 数学成绩'
        chinese=' 语文成绩';
run;
```

FORMAT 语句

- 过程步中的 FORMAT 语句可以为变量输出规定一个输出格式。
- 比如

```
proc print data=sasuser.c9501;
  format math 5.1 chinese 5.1;
run;
```

- 使得列出的数学、语文成绩宽度占 5 位, 带一位小数。
- 在数据步中可以用 ATTRIB 定义永久性的输出格式和变量标签。
- 过程步中的输出格式和变量标签仅限于该过程步。

3.2 列表报告

3.2.1 PROC PRINT 基本使用

列表报告

- PRINT 过程用来生成列表报告。
- 本节讲:
  - PRINT 过程的使用;
  - 常用语句的使用示例;
  - SAS 输出管理。

PRINT 过程基本用法

- 显示刚生成的数据集:

```
proc print;run;
```

- 在 PROC PRINT 语句中用 DATA= 选项指定要列出的数据集:

```
proc print data=sasuser.gpa;
run;
```

- 用 VAR 语句指定要列出的变量及其次序:

```
proc print data=c9501;
  var name chinese sex;
run;
```

PRINT 过程中消除行号列

- 在 PROC PRINT 语句中用 NOOBS 选项抑制观测序号的显示:

```
proc print data=c9501 noobs;
run;
```

- 也可以用 ID 语句指定一个变量作为区分不同观测的输出列，输出时排在第一列，这样就不需要 NOOBS 选项也不显示观测序号，如：

```
proc print data=c9501;
  ID name;
run;
```

PRINT 过程中选择部分行进行显示

- 使用 WHERE 语句可以从输入数据集中选一个行子集:

```
proc print data=c9501;
  where name in (' 李明', ' 张聪');
run;
```

- 也可以使用输入数据集的 WHERE 选项：

```
proc print data=c9501(
  where=(name in (' 李明', ' 张聪')));
run;
```

用 BY 语句分组处理

- 在过程步中使用 BY 语句可以指定分类变量, 把观测分类处理。
- 在使用带有 BY 语句的过程之前先用 SORT 过程对数据集按照分类变量排序。
- 例如:

```
proc sort data=samp.c9501 out=c9501;
  by sex;
run;
proc print data=c9501;
  by sex;
run;
```

结果显示为分男女两组，分类单独显示在分隔线中。

- 可以有多个 BY 变量。
- 多个 BY 变量时可以用 PAGEBY 语句指定按哪个 BY 变量分页。

PRINT 过程中紧凑分组显示

- 如果数据集中某个变量有重复值，先把数据集按此变量分组（排序）后在 PRINT 过程中把此变量同时列入 BY 语句和 ID 语句，则此变量显示在结果的第一列而且重复值只显示一次，不再有单独的分组值的分隔线，如：

```
proc sort data=samp.c9501 out=c9501;
  by sex;
run;
proc print data=c9501;
  by sex;
  id sex;
run;
```

3.2.2 SAS 界面中的输出管理

SAS 中对输出结果的管理

- 新版本的 SAS 系统提供了一个结果管理窗口来管理输出，叫 Results 窗口。
- 如果没有可以从 “View - Results” 菜单打开。
- 过程的输出以树形目录方式显示，双击某一个结果项可以直接跳到该结果。
- 可以在这个窗口中删除、保存某些结果，选定某个结果用右键弹出菜单可以选择此窗口的管理功能。

HTML 输出

- SAS 除了一般文本输出外还提供了 HTML 格式 (即网页格式) 的输出。
- 设置方法: 调用 “Tools - Options - Preferences” 菜单，在弹出的对话框中找到 “Results” 页，选中其中的 “Create HTML”。
- 双击 Results 窗口中的 HTML 输出项目显示 HTML 输出窗口。
- Results 窗口中右键菜单的 “Edit Source” 可以把 HTML 结果的源代码打开到程序编辑窗口进行编辑和保存。

3.2.3 SAS ODS 介绍

SAS ODS 介绍

- Output 窗口的文本型输出是最传统的输出方式。优点是简单好用；缺点是难体现格式。
- 在 SAS 界面中通过选项可以每次生成 HTML 格式的输出，这样的输出也可以复制到 MS Word 中。
- 新版的 SAS 提供了 ODS(输出提交服务) 功能，优点为:
  - 支持的输出格式多，如 HTML、RTF、LaTeX、PDF、PostScript 等。
  - 支持把输出表格作为一个数据集保存。
  - 支持输出部分结果。
  - 支持新输出格式自定义。
  - SAS ODS 是较新功能，不同版本之间有所变化。

输出为 HTML

- 用 ODS HTML 语句指定一个 HTML 输出文件，用 ODS HTML CLOSE 关闭输出文件，用不指定输出文件的 ODS HTML 语句重新打开在窗口缺省显示的 HTML 结果。如

```
ods html body="testhtml.htm";
title ' 身高和体重数据';
proc print data=samp.class noobs;
```

```
run;
title ' 身高和体重基本统计';
proc means data=samp.class;
  var height weight;
  class sex;
run;
title;
ods html close;
ods html;
```

- 在 SAS 9.2 版本中以上程序会把不同的过程输出写到不同的 HTML 文件中。

输出为 CSV

- 用 ODS CSVALL 语句指定一个 CSV 输出文件，用 ODS CSVALL CLOSE 关闭输出文件，如

```
ods csvall body="testcsv.csv";
title ' 身高和体重数据';
proc print data=samp.class noobs;
run;
title ' 身高和体重基本统计';
proc means data=samp.class;
  var height weight;
  class sex;
run;
title;
ods csvall close;
```

- 在 SAS 9.2 版本中以上程序会把不同的过程输出写到不同的 CSV 文件中。
- 把 CSVALL 改成 MSOFFICE2K 或者 EXCELXP，可以输出到与 MS Excel 兼容的文件中。

输出为 RTF

- RTF 是与 MS Word 兼容的格式。例:

```
ods rtf file="testods.rtf" contents toc_data
  bodytitle startpage=no keepn;
ods rtf style=Minimal;
ods noproctitle;
title ' 身高和体重数据';
proc print data=samp.class noobs;
run;
title ' 身高和体重基本统计';
proc means data=samp.class;
  var height weight;
  class sex;
run;
ods rtf close;
```

- 其中的选项TOC_DATA 和 BODYTITLE 需要 SAS9.2 以上版本。
- 选项 CONTENTS 生成目录，TOC_DATA 在运行过程中收集目录条目，BODYTITLE 要求把大标题写在正文部分而非页眉部分，START-PAGE=NO 允许当若干个过程输出都较短时合并在一页中，KEEPN 要求尽可能不把表格拆分到两页。
- 为了显示 SAS 自动生成的目录，用 Microsoft Office 打开生成的 RTF 文件后，用 Ctrl+A 全选内容，然后按 F9 功能键可以更新目录。
- 用 ODS 的 STYLE= 选项指定一种样式，样式种类可以在 “Tools - Options - Preferences” 菜单打开的对话框中的找到 “Results” 页查看和试用。

输出为 LaTeX

- LaTeX 是另外一种常用的报告形式。下例把结果输出为 “sasout.tex”，但也输出了一个 SAS 的支持包 “sas.sty”, 用户可以在自己的 LaTeX 文件中插入 SAS 输出的 LaTeX 文件的内容，但在序言中需要引入 “sas” 包。

```
ods latex file='sasout.tex'
    stylesheet='sas.sty'(url='sas');
proc means data=sasuser.class;
  title 'Mean Height and Weight';
  var height weight;
run;
ods latex close;
```

输出为 PDF

- PDF 是一种可在不同操作系统、本地语言环境下打印、查看但基本上不能修改的格式。

```
ods pdf file='sasout.pdf';
title;
proc print data=sasuser.class noobs label;
run;
ods pdf close;
```

用 ODS 保存输出表为数据集

- 不同的 SAS 过程有不同输出格式和输出数据集选择，有时我们需要的信息只出现在显示结果中，SAS 过程没有提供保存的选项。
- 利用 ODS，可以把任何显示结果保存为数据集。
- 首先，我们要知道该过程输出的各部分结果的名称，办法如下例：

```
ods trace on;
proc univariate data=samp.class;
  var height;
run;
ods trace off;
```

- 在日志中将显示各输出部分的名称：Moments、BasicMeasures、TestsForLocation、Quantiles、ExtremeObs。

- 为了保存显示输出为数据集，只要使用 ODS OUTPUT 语句，其中用“输出名 = 数据集名” 的方法指定输出。如

```
ods output Moments=mom Quantiles=qu;
proc univariate data=samp.class;
  var height;
run;
ods output close;
```

- 上例保存了显示结果中的矩部分为数据集 MOM，分位数部分为 QU。
- 注意保存的数据集是显示格式的，不是最方便使用的。

3.2.4 标题和全局语句

使用中文列标题

- 为了对列标题使用中文, 可以在过程内用 LABEL 语句给变量指定标签, 同时在 PROC PRINT 语句中加 LABEL 选项。
- 如果已经在生成数据集的数据步中用 LABEL 语句或 ATTRIB 语句为变量指定了标签则不必重复定义标签。
- 如:

```
proc print data=c9501 noobs label;
  var name sex math chinese avg;
  label name=' 姓名' sex=' 性别' math=' 数学'
        chinese=' 语文' avg=' 平均分';
run;
```

- 在 PROC PRINT 语句中指定 SPLIT= 选项可以指定标签中的一个字符为换行标志，使得比较长的标签可以在指定的位置拆分为两行或多行。

使用输出格式

- 和 LABEL 语句类似，FORMAT 语句临时为变量指定输出格式。
- 如:

```
proc print data=c9501 noobs;
  var name sex math chinese avg;
  format math 7.1 chinese 7.1 avg 7.2;
run;
```

标题

- 可以指定自己的标题来取代 SAS 缺省的标题。
- 指定标题的 TITLE 语句的格式为： TITLE ’标题内容’;
- 例如:

```
title '95 级 1 班成绩表';
proc print data=c9501 noobs label;
  var name sex math chinese avg;
  label name=' 姓名' sex=' 性别' math=' 数学'
        chinese=' 语文' avg=' 平均分';
run;
```

- TITLE 语句也是一个全局语句。
- 全局语句的作用有持续性, 全局语句的效果将持续到退出 SAS 系统或用另一个同样的全局语句来修改它。
- 取消标题用没有内容的空的 TITLE 语句:

```
title;
```

- 用全局语句 FOOTNOTE 可以为输出加脚注:

```
footnote ' 第三章例子输出';
```

3.2.5 分组与总计

计算总计

- 在 PRINT 过程中可以用 SUM 语句计算某个变量的总计 (总和)。
- 例如, 9501 班的同学购买课外书所用的钱数计算总和:

```
data bkmoney;
  input name $  amount;
  cards;
李明  20
张红艺 15
王思明 10
张聪 20
刘颍 50
;
run;
proc print data=bkmoney noobs;
  sum amount;
run;
```

计算小计和总计

- 用 BY 语句与 SUM 语句就可以既计算总和也计算分组小计，观测分组显示，用带有分组变量值的分隔线分隔。
- 比如, 我们除了要计算学生购买课外书总支出外还想分男、女生计算总支出，首先我们需要把学生性别和课外书支出数据合并，如：

```
proc sort data=c9501;  by name;
proc sort data=bkmoney;  by name;
data c9501bk;
  merge c9501 bkmoney;
  by name;
run;
```

- 然后，用 BY 语句指定分组，与 SUM 语句配合就可以分组计算小计，最后计算总计，如

```
proc sort data=c9501bk;  by sex;
proc print data=c9501bk;
  by sex;
  sum amount;
run;
```

- 如果使用了多个 BY 变量，可以用 SUMBY 语句指定按哪个 BY 变量计算小计。

小计和总计的紧凑显示格式

- 在用 BY 语句与 SUM 语句配合计算小计和总计时，把 BY 变量列入 ID 语句可以使得分组变量列在输出表的第一行而不再有单独的组分隔线，且观测中分组变量的重复值仅显示一次。

- 如：

```
proc print data=c9501bk;
  by sex;
  id sex;
  var name amount;
  sum amount;
run;
```

显示观测个数

- 在 PROC PRINT 语句中中加入 N=' 总人数:' 选项可以在表最后显示' 总人数: xx'。如：

```
proc print data=c9501bk n=' 总人数：';
  sum amount;
run;
```

- 但是，当使用了 BY 进行分组显示时，观测个数在组内显示。如需要指定如

```
proc print data=c9501bk n=' 小组人数: ';
  by sex;
  id sex;
run;
```

- 如果使用了 BY 进行分组且用 SUM 语句计算小计和总计，则选项 N= 需要指定如 N=' 小组人数：' ' 总人数：'，这样在每个组显示' 小组人数: xx'，在表最后显示' 总人数: xx'。如：

```
proc print data=c9501bk
    n=' 小组人数: ' ' 总人数: ';
  by sex;
  id sex;
  sum amount;
run;
```

3.3 汇总表格

TABULATE 过程与汇总表格

- PRINT 过程列出观测。
- TABULATE 过程绘制概括统计量的表格。
- TABULATE 可以作出很复杂的表, 其一般格式为：

```
PROC TABULATE DATA= 数据集名;
    CLASS 分类变量;
    VAR 分析变量;
    TABLE 页维说明, 行维说明, 列维说明 / 选项;
RUN;
```

- CLASS 语句给出分类变量;
- VAR 语句给出区间变量。
- 一个 TABULATE 过程步中允许使用多个 TABLE 语句。

例: 男、女生的课外书支出总和

- 对 C9501BK 数据集,

```
proc tabulate data=c9501bk;
  class sex;
  var amount;
  table sex, amount;
run;
```

- 行维是 SEX，作为列维是 AMOUNT 的总和。

例：男女生人数统计

- TABLE 语句中只有一个变量 SEX，计算分类变量 SEX 每类的人数。

```
proc tabulate data=c9501bk;
  class sex;
  table sex;
run;
```

- 区间变量的缺省统计量是总和;
- 分类变量的缺省统计量是频数。
- 分类变量各个值的次序缺省为数据中出现的先后次序，但是可以用 PROC TABULATE 语句或 CLASS 语句中的 ORDER= 选项指定，可取值为 DATA(缺省)、FORMATTED(按格式化输出内容排序)、FREQ(按出现次数由大到小排序)、UNFORMATTED(按存储内容排序)。

计算统计量

- 要计算统计量, 可以用 “变量名 * 统计量名” 的形式。
- 统计量名包括 N, NMISS, MEAN, STD, MIN, MAX, RANGE, SUM, USS, CSS, STDERR, CV, T(检验均值为 0 的 t 统计量值), PRT(t 统计量的 p 值), VAR, SUMWGT(权数变量的和), PCTN(某类观测占总观测个数的百分比), PCTSUM(某类观测的总和占全部总和的百分比)。

计算统计量 (续)

- 例如，求男、女生的数学、语文成绩平均值及标准差：

```
proc tabulate data= c9501bk;
  class sex;
  var math chinese;
  table sex, (math chinese)*(mean std);
run;
```

- 变量名和统计量名的次序也可以颠倒过来，比如 `(mean std)*(math chinese)`，得到的表格中也将以统计量为大栏目而以变量为小栏目。

合并分类的统计

- 为了在分类计算同时还计算不分类的结果，只要在 TABLE 语句中分类变量后面加上 ALL 关键字。
- 如:

```
table sex all, (math chinese)*(mean std);
```

- 可以用星号连接分类变量和分类变量可以构成交叉分组；
- 可以用星号连接分类变量和区间变量可以分类计算区间变量的统计量；
- 用星号连接变量名与统计量名表示计算该变量的相应统计量。
- 例如: 计算性别频数 (N) 和百分比 (PCTN)

```
table (sex all)*(N PCTN);
```

- 上例没有行维, 只有列维。

分类变量的分类作用

- CLASS 语句指定的分类变量如果不是要对本身计算 N、PCTN 这样的统计量而是希望分类计算其他变量的统计量, 还需要指定需要统计的区间变量, 如:

```
table (sex all)*math*(mean std);
```

- 上例没有行维, 只有列维。

缺失值处理

- CLASS 语句指定的分类变量如果有缺失值，正常情况下缺失值会被忽略。
- 在 PROC TABULATE 语句中加入 MISSING 选项，则要求所有的分类变量如果有缺失值则缺失值作为一个单独的分类。

- 在 CLASS 语句中加入 MISSING 选项，则此 CLASS 语句中指定的分类变量的缺失值作为单独分类。如

```
CLASS a b / missing;
CLASS c;
```

- 则变量 A, B 包含缺失值作为单独分类，而 C 则舍弃缺失值。

标签

- 可以在 TABULATE 过程中使用 KEYLABEL 语句指定各统计量的标签。
- 其格式为

  KEYLABEL 关键字＝’ 标签’;

- 例如:

```
proc tabulate data=c9501bk;
  class sex;
  var math chinese;
  table (sex all),
        (math chinese)*(mean std);
  keylabel mean=' 平均值'
           std=' 标准差' all=' 总计';
  label sex=' 性别' math=' 数学'
        chinese=' 语文';
run;
```

标签 (续)

另一种指定统计量标签的办法是在 TABLE 语句中直接指定，例如:

```
proc tabulate data=c9501bk;
  class sex;
  var math chinese;
  table (sex=' 性别' 'all'=' 总计'),
```

```
        (math=' 数学'  chinese=' 语文') *
          ('mean'=' 平均值' 'std'=' 标准差');
run;
```

格式

- 在 PROC TABULATE 语句中用 FORMAT= 指定数值的显示格式，如 FORMAT=comma10.2。
- 也可以对某一统计量用“*F=“指定输出格式。例如:

```
proc tabulate data=c9501bk;
  class sex;
  var math chinese;
  table (sex=' 性别' 'all'=' 总计'),
        (math=' 数学'  chinese=' 语文') *
          ('mean'=' 平均值'*F=5.1
           'std'=' 标准差'*F=6.2);
run;
```

输出到数据集

- 在 PROC TABULATE 语句中用 OUT= 指定一个输出数据集。
- 一般生成汇总和分类汇总数据集应该用 PROC MEANS 比较方便，PROC TABULATE 也有一些独特的统计量如 PCTSUM 之类。
- 例如:

```
proc tabulate data=samp.c9501bk OUT=summd;
  class sex;
  var amount;
  table sex, amount*(N SUM PCTSUM);
run;
```

SAS 表格线乱码问题

- 有时用 TABULATE 过程作表时表格线是一些乱码, 这是 SAS 系统设置文件中关于 FORMCHAR 的设置的问题。
- 在 SAS 系统的安装目录下查找到一个名为 SASV8.CFG 的文件, 把其中的所有以 "−FORMCHAR" 开始的行都注释掉, 只留下如下的行不注释:

```
-FORMCHAR "|----|+|---+=|-/\<>*"
```

就可以解决这个问题。

3.4 数据排序

SORT 过程

- 在 SAS 过程中用 BY 语句可以把观测分类进行处理, 但在此之前需要先用 SORT 过程排序。
- SORT 过程可以把数据集按某一个或若干个变量的次序进行排序。
- 例如:

```
proc sort data=c9501;
  by sex;
run;
```

- 用 DATA= 指定的数据集既是输入数据集又是输出数据集。
- 用 OUT= 选项把结果存到新数据集。

获得变量取值集合

- 在 PROC SORT 语句中加 NODUPKEY 选项可以在按指定的 BY 变量排序时舍弃重复的 BY 变量值，只留下那些互不相同的 BY 变量值。
- 如

```
proc sort data=c9501
    out=c9501sex(keep=sex) nodupkey;
  by sex;
run;
```

- 一个变量名前面加上 DESCENDING 关键字表示此变量的排序是由大到小的。
- 在 INSIGHT 中也可以排序。

按几个变量排序

- 按几个变量排序, 比如, 要按男、女性别排序, 并在男生、女生内部按平均分由高到低排序, 可以：

```
proc sort data=c9501;
  by sex descending avg;
run;
```

- 一个变量名前面加上 DESCENDING 关键字表示此变量的排序是由大到小的。
- 在 INSIGHT 中也可以排序。

数据步中排序处理

- 在数据步中用 SET 语句读入的数据集如果按照某个变量排序，就可以加 BY 语句按此变量分组，可以使用临时变量 FIRST. 变量名判断是否分组的首个观测，LAST. 变量名判断是否分组的最后一个观测。
- 例如，为了只保留男生中最高和女生中最高，可用

```
proc sort data=sasuser.class out=cl2;
  by sex descending height;
run;
data new;
  set cl2;
  if first.sex;
  by sex;
run;
proc print;run;
```

3.5 数据集转置

数据集转置

- 一个 SAS 数据集（或其列子集）可以看成是某种矩阵，观测为矩阵的行，变量为列。
- 用 TRANSPOSE 过程转置：行变成列，列变成行。

简单的矩阵转置

```
data mat;
  input x1 x2 x3;
  cards;
1   2  3
4   5  6
7   8  9
10 11 12
;
run;

proc transpose data=mat out=matt;
  var x1 x2 x3;
run;
proc print;run;
```

合并观测例：数据

待合并观测的数据集 ONECOL

病历号 NUM	药物 TEST	药效 VAL
1	a	11
2	a	12
3	a	13
1	b	21
2	b	22
3	b	23

合并观测后的数据集 **TWOTEST**

病历号 NUM	来源变量 _NAME_	药物 A A	药物 B B
1	val	11	21
2	val	12	22
3	val	13	23

合并观测例: PROC TRANSPOSE

```
proc sort data=onecol;
  by num;
run;
proc transpose data=onecol out=twotest;
  var val;
  id test;
  by num;
run;
```

- 用 BY 语句对每个病人分别处理。
- 用 VAR 语句指定要转置的列子集。VAR 指定的变量每一行会变成结果的一列，但有 BY 分组时只对当前组的行转置。
- 用 ID 语句指定各行变成列时的变量名数据。
- 加输出数据集选项DROP=_NAME_ 可以去掉多余的_NAME_ 列。

合并观测: 使用数据步

- 把 TEST=”a” 的观测输出到数据集 DA, TEST=”a” 的观测输出到数据集 DB；
- 使用标准的横向合并方法按关键列 NUM 合并数据集 DA 和 DB。

```
data da;
  set onecol;  where test='a';
  a = val;  keep num a;
data db;
  set onecol;  where test='b';
  b = val;  keep num b;
proc sort data=da;  by num;
proc sort data=db;  by num;
```

```
data twocol;
  merge da db;  by num;
run;
```

- 其中数据子集的程序可以用数据集选项，如

```
data a(drop=test);
  set onecol(where=(test='a')
      rename=(val=a) );
data b(drop=test);
  set onecol(where=(test='b')
      rename=(val=b) );
```

合并观测: PROC SQL

把 ONECOL 看成两个并排的数据集进行合并:

```
title ' 合并行: PROC SQL';
proc sql;
  create table twoc as
    select a.num as num, a.val as a,
           b.val as b
      from samp.onecol a, samp.onecol b
      where a.num=b.num
            and a.test='a'
            and b.test='b';
quit;
```

拆分观测例: 数据

待拆分观测的数据集 TWOCOL

病历号	测量 1	测量 2
NUM	TEST1	TEST2
1	11	21
2	12	22
3	13	23

拆分观测后的数据集 ONETEST

病历号 NUM	来源变量 _NAME_	测量 COL1
1	test1	11
1	test2	21
2	test1	12
2	test2	22
3	test1	13
3	test2	23

拆分观测例: PROC TRANSPOSE

```
proc sort data=twocol;
  by num;
run;
proc transpose data=twocol out=onetest;
  var test1 test2;
  by num;
run;
```

- BY 语句要求对每个病人分别处理;
- VAR 语句指定要转置的两列, 每个病人的这两列在结果中被转置为同一列的两行。
- 新的列变量自动起名为 COL1；
- 结果中_NAME_ 表示从列变成的行原来是哪一列。
- 为了把自动命名的 COL1 改名，可以用数据集选项的 RENAME，如

```
proc sort data=twocol;
  by num;
run;
proc transpose data=twocol
     out=onetest(rename=(col1=val));
  var test1 test2;
  by num;
run;
```

- 为了把区分两次试验的 test1, test2 改为数值 1 和 2，可以使用数据步的 SUBSTR 和 INPUT 函数：

```
data new;
  set onetest(rename=(_name_=test));
  testid = input(substr(test, 5,1), 1.);
  drop test;
run;
proc print;run;
```

- 其中在输入数据集选项中把变量名_NAME_ 改为了 TEST。

拆分观测：数据步

在数据步中用多次 OUTPUT 可以把一个观测拆分为多个观测：

```
data new1;
  attrib num length=8
    test length=$1
        val length=8;
  set twocol;
  val=test1; test='a'; output;
  val=test2; test='b'; output;
  drop test1 test2;
run;
proc print;run;
```

拆分观测：PROC SQL

PROC SQL 中 UNION 进行上下合并：

```
title ' 拆分行: PROC SQL';
proc sql;
  select num, 'a' As test, test1 AS val
    from samp.twocol a
  union
  select num, 'b' As test, test2 AS val
    from samp.twocol;
quit;
```

3.6 描述统计

描述统计

- MEANS、UNIVARIATE 和 FREQ 这三个过程用来计算简单的描述统计量。
- MEANS 和 UNIVARIATE 过程对区间变量计算均值、标准差等数字特征;
- FREQ 过程计算取值频数分布。

MEANS 过程例子

- 例:

```
proc means data=c9501;
  var math chinese;
run;
```

- 可以在 PROC MEANS 语句中指定 SUM, CV 等统计量名，要求只输出这些统计量。如

```
proc means data=sasuser.class sum cv;
  var height;
run;
```

MEANS 过程: 置信区间

- 在 PROC MEANS 语句中用 CLM 选项要求计算均值的置信区间，用 ALPHA= 指定置信度为 $1-\alpha$:

```
proc means data=sasuser.class
    alpha=0.10 mean std clm;
  var height;
run;
```

MEANS 过程: OUTPUT 语句 (第一种)

- 可以使用 OUTPUT 语句把结果保存到数据集，如:

```
proc means data=sasuser.class;
  var height weight;
  output out=mc1 mean=mh mw;
run;
```

- 完全由用户指定变量名，结果数据集有一行。

MEANS 过程: OUTPUT 语句 (第二种)

- 使用 AUTONAME 要求对统计量自动命名，结果数据集为一行：

```
proc means data=samp.class;
  var height weight;
  output out=mc2 mean= std= / autoname;
run;
```

MEANS 过程: OUTPUT 语句 (第三种)

- 在 PROC MEANS 语句中指定要输出的统计量，OUTPUT 语句中不指定统计量和变量名，这时输出的数据集中每种统计量占一行，每个分析变量占一列，如

```
proc means data=samp.class
    NOPRINT MEAN MIN MAX;
  var height weight;
  output out=mc3;
run;
proc print;run;
```

MEANS 过程: 输出结果数据集的利用

- 为了从 CLASS 数据集的身高中减去全班平均身高，可用如下数据步：

```
proc means data=sasuser.class;
  var height;
  output out=mc1 mean=mh;
run;
data cc;
  set sasuser.class;
  if _n_=1 then set mc1(keep=mh);
  height2 = height - mh;
run;
proc print;run;
```

- 这里用了两个 SET 语句，使用了数据步临时变量_N_。

MEANS 过程: CLASS 语句

- 用 CLASS 语句指定分组变量进行分组汇总或交叉分组汇总，如

```
proc means data=sasuser.class;
  class sex;
  var height;
  output out=mcs mean=mh;
run;
proc print;run;
```

- 生成的结果数据集 MCS 中包含男女生各一行结果，还包含一行不分男女生的结果。用变量_TYPE_ 区分分组级别。

- 与使用 BY 语句的效果类似，但是

  (1) 使用 BY 语句需要预先对数据集排序；

  (2) 用 CLASS 语句分组汇总，生成的结果数据集包含不分组的结果行。

- 如果希望生成的数据集中不包含不分组的结果行，在 PROC MEANS 语句中使用 NWAY 选项。
- 在上例中得到分男女生的平均身高后，可以用如下程序对男女生身高分别减去本组的平均身高：

```
proc sort data=sasuser.class;
  by sex;
proc sort data=mcs(where=(_type_=1));
  by sex;
data new;
  merge sasuser.class mcs;
  by sex;
  height2 = height - mh;
run;
proc print;run;
```

MEANS 过程: 用 CLASS 语句交叉分组

- 数据集 GRADE 中包含了 10 个学生的姓名、性别、状态（status, 1 或 2）、年级 (year, 97 或 98)、分区 (section, A 或 B)、分数 (score)、总评分数 (FinalGrade)。
- 在 CLASS 语句中指定多个分类变量进行交叉分组的汇总：

```
proc means data=samp.grade maxdec=3;
   var Score;
   class Status Year;
   title ' 按学生状态和毕业年分类的分数统计';
run;
```

- 显示的结果有 4 类。

MEANS 过程: 用 CLASS 语句汇总数据

- CLASS 语句配合 OUTPUT 语句可以把分类汇总的统计量保存到数据集，如：

```
proc means data=samp.grade;
  var Score;
  class Status Year;
  output out=mgsy mean=ms std=ss;
run;
proc print;run;
```

- 结果中除了交叉分组得到的 4 行外，还包括只按 Status 分组的两行，只按 Year 分组的两行，和不分组的一行。用变量_TYPE_ 区分不同的分组级别。

SUMMARY 过程

- PROC SUMMARY 与 PROC MEANS 用法相同，但主要用来生成汇总数据集，缺省情况下不显示。

UNIVARIATE 过程例子

```
proc univariate data=sasuser.gpa;
  var gpa;
run;
```

UNIVARIATE 过程的结果

- 矩统计量；
- 基本的位置和分散程度统计量；
  - 均值、中位数、众数；
  - 标准差、方差、极差、四分位间距。
- 关于均值等于零的三种检验的结果:
  - t 检验;
  - 符号检验;
  - 符号秩检验。

- 各个重要的分位数估计;
- 五个最低值和五个最高值。

茎叶图

- 在 PROC UNIVARIATE 语句中加一个 PLOT 选项可作茎叶图、盒形图和正态 QQ 图。
- 茎叶图是一种特殊的直方图–每个叶子代表一个观测值。
- 例: GPA 的茎叶图的解释。

FREQ 过程

- FREQ 过程可以列出变量的取每个值的次数、比例、累计次数、累计比例。
- 在 TABLES 语句中指定要列表的变量。
- 例:

```
proc freq data=c9501;
  tables sex;
run;
```

- 可以把结果保存到一个输出数据集，如:

```
proc freq data=c9501;
  tables sex / out=sext;
run;
proc print;run;
```

- 输出数据集中如果需要累计频数和累计百分比则应使用 OUTCUM 选项。

FREQ 过程例: GPA

- 例:

```
proc freq data=sasuser.gpa;
  tables gpa;
run;
```

- 结果中变量值列和累计百分比列构成经验分布函数表格。
- 经验分布函数:

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I_{\{X_i \le x\}}.$$

列联表

- FREQ 过程可以列出两个变量所有不同组合的频数、百分比以及组内百分比。
- 在 TABLES 语句中，两个变量用星号连接。
- 例:

```
proc freq data=samp.class;
  tables age*sex;
run;
```

- 列联表的结果可以输出为每个组合一行的数据集，如

```
proc freq data=samp.class NOPRINT;
  tables sex*age / OUT=crosstab;
run;
proc print;run;
```

表格数据例

- 例: 出租车费面值与该面值张数。

```
data sasuser.taxif;
  input amount num @@;
  cards;
10  4    12  6    13  1    15  1
16  1    19  5    20  3    23  1
24  1    25  1    26  3    27  1
32  1    47  1    48  2    49  1
52  1    55  1    58  1    81  1
;
run;
```

- INPUT 语句结尾处的两个@ 表示允许在同一行读取不同观测。

FREQ 语句例

- 分析车费情况：用 FREQ 指定重复次数。

```
proc means data=sasuser.taxif
        mean std sum min max;
  var amount;
  freq num;
run;
```

3.7 相关系数计算

相关系数矩阵

- CORR 过程用来计算变量的相关系数。
- 相关系数可以反映变量两两之间的线性相关的强弱, 又称为 Pearson 积矩相关系数。
- 例: 计算 SASUSER. GPA 中的三个变量 HSM, HSS, HSE 两两之间的相关系数:

```
proc corr data=samp.gpa;
  var hsm hss hse;
run;
```

- 结果包括：变量基本统计量，两两的相关系数，相关系数等于零的检验的 p 值。
- INSIGHT 中也可以计算相关系数。

仅部分相关系数

- 用 VAR 语句和 WITH 语句指定两组变量，可以仅在这两组变量之间计算相关系数。
- 如

```
proc corr data=samp.gpa;
  var hsm hss hse;
  with satm satv;
run;
```

- 在 PROC CORR 语句中用 OUTP= 选项把 Pearson 相关系数结果输出到指定数据集。

秩相关系数

- Pearson 积矩相关系数主要适合于联合正态分布情况。
- 对于严重偏斜或重尾的分布，可以把变量 X 和 Y 的观测值分别计算秩（名次）统计量，然后计算秩之间的 Pearson 积矩相关系数，结果称为 Spearman 秩相关系数。
- 计算秩时相同秩取平均值。
- 使用 PROC CORR 的 SPEARMAN 选项计算 Spearman 秩相关系数。如

```
proc corr data=samp.gpa spearman;
  var satm satv;
  with gpa;
run;
```

- 在 PROC CORR 语句中用 OUTS= 选项把 Spearman 相关系数结果输出到指定数据集。

3.8 用 SAS/GRAPH 绘图

用 SAS/GRAPH 绘图

- SAS/GRAPH 是 SAS 的绘图模块，可以作:
  - 散点图、曲线图;
  - 直方图、扇面图;
  - 三维曲面图、等高线图;
  - 地图;
  - 等等

3.8.1 散点图和曲线图

散点图

- 用 GPLOT 过程绘制散点图和曲线图。
- 例如，绘制 SASUSER.GPA 中 SATV 对 SASM 的散点图:

```
proc gplot data=sasuser.gpa;
  symbol i=none v=star;
  plot satv*satm;
run;
```

- PLOT 语句指定纵坐标和横坐标变量，形式为 “y*x”;
- SYMBOL 语句指定散点或曲线的绘制方式。可以有多个 SYMBOL 语句，如 SYMBOL2, SYMBOL3 等。
- 结果在 GRAPHICS 窗口中。

曲线图

- 绘制连线只要在 SYMBOL 语句中指定 I=JOIN。
- 例如，对 SASUSER.AIR 数据集中以 DATETIME 为横轴、以 CO 为纵轴绘曲线图:

```
proc gplot data=sasuser.air;
  symbol i=join v=star;
  plot co*datetime;
run;
```

- SYMBOL 中 I= 指定连接散点的方式，V= 指定散点符号。
- V=NONE 表示取消散点。

图形选项

- SYMBOL 语句与 TITLE，OPTIONS 等语句一样，是全局语句，其作用会持续到后续的绘图过程。
- 为了重置所有的图形选项，用

```
goptions reset=global;
```

- GOPTIONS 语句是一个全局语句，用来指定共同的图形选项。

叠加的曲线

- 可以在一个坐标平面作多条曲线的图形。
- 只要在 PLOT 语句中指定多个因变量 (自变量一般应为同一个), 并使用 OVERLAY 选项, 如:

```
proc gplot data=sasuser.air;
  symbol1 color=black i=join v=none line=1;
  symbol2 color=blue i=join v=none line=2;
  plot co*datetime=1
       so2*datetime=2 / overlay;
run;
```

- 用了两个 SYMBOL 语句分别规定两条曲线的画法；
- 用“y*x=n”的形式选择对应的 SYMBOL 语句。

SYMBOL 语句的连线选项

- I=JOIN 直接连线；
- I=SPLINE 连接样条插值曲线；
- I=SM*nn* 连接样条平滑曲线；
- I=NEEDLE 绘制每个点到横轴的垂线;
- I=RL 绘制线性回归直线, I=RQ 为二次曲线, I=RC 为三次曲线;
- I=RLCLI95 在回归直线之外绘制预测值的 95%置信限曲线.

3.8.2 直方图和扇形图

直方图

- 用 GCHART 过程绘制直方图、扇面图 (饼图)、三维直方图等表示变量分布的图形。
- 例如, 要绘制 SASUSER.GPA 中 GPA 的分布直方图, 只要用：

```
proc gchart data=sasuser.gpa;
  vbar gpa;
run;
```

- 把 VBAR 改成 HBAR 则条形方向变为横向；
- 可以用 VBAR3D 或 HBAR3D 作出立体效果的直方图。
- 横轴标的是区间的中点值。
- VBAR 语句的 TYPE= 选项指定纵轴代表的值。横轴是分类变量或区间变量的分组区间，纵轴的 TYPE= 可以取 FREQ(该类取值次数)、PERCENT(百分比)、CFREQ(累计次数)、CPERCENT(累计百分比)。还可以加一个 SUMVAR= 选项，对每类中另一个变量表示总和或平均值，这时 TYPE= 取 SUM 或 MEAN。
- 例如，下面的例子绘制男女生两个条形，条形长度代表 SATM 分数的平均值：

```
proc gchart data=sasuser.gpa;
  vbar sex / type=mean sumvar=satm;
run;
```

直方图 (续)

- 还可以用 Univariate 过程作直方图，如:

```
proc univariate data=Sasuser.gpa noprint;
   var gpa;
   histogram;
run;
```

分组的直方图

- VBAR 语句的 GROUP= 选项按分类变量做并排直方图。
- 例如，按性别分组绘制 GPA 的两个直方图并排放置:

```
proc gchart data=sasuser.gpa;
  vbar gpa / group=sex;
run;
```

分段的直方图

- SUBGROUP= 选项做分段直方图。
- 例如，按性别分段绘制 GPA 的直方图:

```
proc gchart data=sasuser.gpa;
  vbar gpa / subgroup=sex;
run;
```

扇形图

- 在 GCHART 中用 PIE 语句绘制表示频数的扇形图, 例如：

```
proc gchart data=sasuser.gpa;
  pie sex;
run;
```

- 可以用 PIE3D 作出立体效果的扇形图。
- 如果想显示百分比值, 只要在 PIE 语句中加入 TYPE = PERCENT 选项, 如 “pie sex / type=percent;”。

盒形图

- 盒形图使用 BOXPLOT 过程，需要一个分组变量，如:

```
data new;
  set sasuser.gpa;
  g = 1;
proc boxplot data=new;
  plot gpa * g;
run;
proc sort data=new;
  by sex;
proc boxplot data=new;
  plot gpa * sex;
run;
```

三维曲面图

- 假设对一个二元函数 z=f(x,y), 我们有了 x 取等间隔值、y 取等间隔值时 z 的值, 这时我们可以用 G3D 过程绘制曲面图形, 用 GCONTOUR 绘制曲面的等高线图。

- 例如, 假设 (X, Y) 服从联合正态分布, 其均值都是 0, 方差分别为 1 和 a, 相关系数为 r。则 (X, Y) 的联合密度函数

$$f(x,y) = \frac{1}{2\pi\sqrt{a\left(1-r^2\right)}} \exp\left(-\frac{1}{2a\left(1-r^2\right)}\left(ax^2+y^2-2r\sqrt{a}xy\right)\right)$$

二维曲面例：数据

```
data dnorm2;
  a=2;
  a2=sqrt(a);
  r=0.5;
  det=a*(1-r*r);
  do x=-3 to 3 by 0.3;
    do y=-3*a2 to 3*a2 by 0.3*a2;
      z=1/(2*3.1415926*sqrt(det))
        *exp(-0.5/det*
        (a*x*x + y*y - 2*r*a2*x*y));
      output;
    end;
  end;
  keep x y z;
run;
```

二维曲面：作曲面图和等值线图

- 用 G3D 过程来绘制曲面图:

```
proc g3d data=dnorm2;
  plot x*y=z;
run;
```

- 用 GCONTOUR 过程可以绘制等值线图：

```
proc gcontour data=dnorm2;
  plot x*y=z / nolegend autolabel;
run;
```

- 在 INSIGHT 中画旋转图时在 Output 选项中选中 Fit surface 可以作曲面图；用 “Analyze - Contour Plot(Z Y X)” 菜单可以作等值线图。

旋转和倾斜

- 用 G3D 过程绘制曲面图时，在 PLOT 语句中用 ROTATE= 指定旋转度数，用 TILT= 指定倾斜度数。
- 如

```
data hat;
   do x=-5 to 5 by 0.25;
      do y=-5 to 5 by 0.25;
         z=sin(sqrt(x*x+y*y));
         output;
      end;
   end;
run;

proc g3d data=hat;
   plot y*x=z / rotate=45 tilt=15;
run;quit;
```

三维散点图

- 在 G3D 过程中用 SCATTER 语句绘制三维散点图。
- 如

```
proc g3d data=samp.iris;
   scatter petallen*petalwid=sepallen;
run;
```

- 可以在 SCATTER 语句中用 COLOR= 和 SHAPE= 选项指定区分不同绘点颜色、符号的变量。

3.8.3 图形的调整与输出

图形中的汉字

- 各绘图过程中都指定了丰富的选项来调整图形。
- 在图形中也可以用 TITLE 语句和 FOOTNOTE 语句给图形加标题和脚注。
- 在 GOPTIONS 语句中指定了汉字字体后可以在图形中使用汉字，如:

```
goptions ftext=" 宋体"
    htitle=2 cells htext=1 cells;
proc gplot data=sasuser.class;
  title " 试验 SAS 图形的汉字功能";
  symbol i=none v=square;
  plot weight * height;
  label weight = " 体重" height=" 身高";
run;
```

图形的保存和打印

- 在图形窗口，选 “File – Export” 菜单, 在出现的输入文件名的窗口选择适当的文件类型, 如 BMP、 WMF、 JPEG、 PS, 等等, 可以把图形保存为图形文件。
- 在图形窗口，选 “File – Print” 可以打印图形。

3.9 分析员模块介绍

分析员模块介绍

- Analyst 也是用图形界面调用 SAS 功能的一个模块；
- 它基于正常的 SAS 程序，界面自动生成要用的 SAS 程序，可以查看所用的 SAS 程序，作为学习参考；
- 可以管理数据、进行描述统计、作图、假设检验、回归分析、方差分析、多元分析。
- 从主菜单 “Solutions – Analysis – Analyst” 可以进入 Analyst 系统。
- 窗口分左右两部分，项目管理和数据窗口。结果显示在单独的窗口。

3.9.1 数据管理

数据管理—编辑

- 菜单 “File – Open by SAS Name” 打开数据集，以 SASUSER.CLASS 为例。
- 如需修改数据要选 “Edit – Mode – Edit” 菜单。
- 修改后需要另存才真的修改数据集。
- 行或列上有右键菜单提供编辑功能。
- 右键菜单功能包括:
  - Add 或 Insert —添加行或列；
  - Duplicate — 复制行或列；
  - Delete — 删除行或列；
  - Move — 移动列的排列次序；
  - Hide — 隐藏某列不显示；
  - Hold — 指定到本列为止的列在左右翻看时不动；
  - Sort — 按变量值排序；
  - Labels — 切换显示变量标签；
  - Properties — 查看变量属性。

数据筛选

- Analyst 的 Data 菜单提供了对数据的一些操作。
- “Data — Filter — Subset Data”: 用 Where 表达式选择满足条件的行子集；
- 需要另存才能把筛选结果保存下来。

数据变换

- Compute — 计算新变量；
- Rank — 计算秩统计量；
- Standardize — 标准化变量；
- Recode Values — 变量离散值重新编码；
- Recode Ranges — 连续性变量离散化；
- Change Type — 改变类型（数值型和字符型转换）；
- 可以从自动生成的程序中学习。

随机数生成

- “Data – Random Variates” 菜单可以生成一列随机数。

分类汇总

- “Summarize By Group” 数据汇总功能。
- 以 TOTALS 数据集为例。
- 计算按照分公司 (SITE) 和部门 (DEPT) 分类的总销售额，用 “Data – Summarize By Group”。
- 汇总结果可以保存为数据集。

横向合并

- “Data – Combine Tables – Merge By Columns” 相当于数据步 MERGE 和 BY 语句配合进行横向合并。
- 例：合并 C9501X 和 C9501Y。以 NAME 作为“MERGE BY”变量。
- 可以从“Combine table will keep”选择：
  - Matches only 指两个数据集都出现同一个名字才保留合并的行；
  - Matches to Table 1 指只要在第一个数据集中出现的名字就予以保留；
  - All rows 指两个数据集中出现的名字不管能否匹配都要保留 (这是数据步中 MERGE 语句的缺省操作)。
- 查看对应的程序。

纵向合并

- “Data – Combine Tables – Concatenate By Rows” 作纵向合并。
- 可以选择：
  - Append 是直接上下合并，这是 SET 的缺省操作；
  - Interleave(交错) 是合并时按照某个变量的值合并，该变量值相同的观测相邻放置，确保最后得到的数据集按此变量排序。

行列转置

- “Data – Stack Columns” 菜单堆把若干列合并到一列。比如把 3.5 节的 TWOCOL 中的 TEST1 和 TEST2 合并到一列 VAL。
- “Data – Split Columns” 把一列拆分为若干列，比如把 3.5 节的 ONECOL 中的 VAL 拆分为两列。
- “Data – Transpose” 对数据集进行转置，比如 3.5 节的 MAT 数据集。

随机抽样

- “Data – Random Sample” 菜单对数据集的观测进行随机抽样。
- 可以看成是从一个有限总体做无放回抽样。
- 可以指定要抽取的样本量 (Rows) 或百分比 (Ratio)。

3.9.2 报表

报表—列表

- 相当于 PRINT 过程。
- 选 “Reports – List Data” 菜单。
- 可设置 ID 变量；标题 (Titles 按钮)；选项 (Options 按钮)；总计 (Options 界面的 SUM 页)。

报表—汇总表

- 相当于 TABULATE 过程。可以制作五种常见汇总表，简单易用。
- 选 “Reports – Tables” 菜单。
- 如: GPA 数据集分男女计算 SATM 和 SATV 的平均值、标准差、中位数。

3.9.3 描述统计

描述统计

- “Statistics – Descriptive” 菜单可以计算描述统计量。
- “Summary Statistics” 相当于 MEANS 过程。
- 可指定分析变量 (Analysis)；
- 用 Statistics 的选项可以指定要计算的统计量 (包括所有的矩统计量)；
- 用 Plots 的选项可以要求绘制变量的盒形图和直方图；
- 用 Output 的选项可以指定数值输出格式和是否使用变量标签；
- 用 Variables 的选项可以指定分组变量 (By Group)、权重变量 (Weight) 和频数变量 (Frequency)，
- 用 Save Data 的选项把统计量写到一个 SAS 数据集。
- “Statistics – Descriptive – Distributions” 相当于 UNIVARIATE 过程，但是在 Analyst 中又通过编程提供了增强的功能。
- 先选定分析变量 (Analysis)；

- 用 Fit 选项可以要求对数据拟合正态、对数正态、指数、威布尔等四种分布。

- 用 Plots 可以要求绘制盒形图、直方图、概率图和 QQ 图等四种图形，如果已经要求了拟合分布则直方图中将叠加拟合的分布密度，概率图和 QQ 图也是相对于要拟合的分布来画；

- **概率图** (Probability Plot) 和 QQ 图基本相同，两者纵坐标都是分析变量由小到大排序后的数值 y_i，若 y_i 相当于 v_i 经验分位数 (可以是修正过的)，这时正态 QQ 图横坐标是标准正态的 v_i 分位数，横坐标的标度也是正态分位数；而正态概率图的横坐标仍用标准正态分位数但是标度却标分位数相应的概率值即 v_i 本身。

相关系数和频数分布

- “Statistics – Descriptive - Correlations” 菜单计算变量的相关系数，相当于 CORR 过程，这里增强的地方是可以画散点图并在散点图上画置信椭圆。

- “Statistics – Descriptive - Frequency Counts” 菜单进行变量分布频数统计，相当于 FREQ 过程，同时还可以画离散变量的条形图。

3.9.4 画图

画图

- 用 Analyst 的 Graphs 菜单可以做图。

- “Graphs – Bar Chart” 画条形图。

- 可以画通常的竖立的条形图，也可以画水平的条形图。

- 可以把条形画成三维形状。

- 区间变量的条形图与直方图没有本质区别，但是分类变量只能作条形图不能作直方图。

- “Graphs – Pie Chart” 画扇形图，可以画成三维的形状。

- “Graphs – Histogram” 画直方图。 Analyst 的这个菜单有增强的功能。通过 Fit 的选项可以要求拟合正态、对数正态、指数、威布尔四种分布并把拟合的分布密度曲线画在直方图上面。

画图 (续)

- “Graphs – Box Plot” 画盒形图。设定盒形图的对话框中有一个 Display 按钮可以选择具体画法，我们通常见的盒形图在这里相当于 Style 为 Schematic。
- “Graphs – Probability Plot” 画概率图，以数据集中的变量值为纵坐标，以拟合的理论分布的对应分位数为横坐标但横坐标轴标的是概率值。可以选正态分布、对数正态分布、指数分布、威布尔分布。
- “Graphs – Scatter Plot” 画二维散点图或三维散点图。
- “Graphs – Contour Plot” 关于 X, Y, Z 三个坐标画等值线图。
- “Graphs – Surface Plot” 关于 X, Y, Z 三个坐标画曲面图。

3.10 补充

3.10.1 程序纠错

SAS 编程纠错

- SAS 程序的各种错误会反映在日志 (LOG) 窗口。
- 程序错误类型有：
  - 语法错误，如关键字拼写错误，丢失分号，语句或选项错误。SAS 在编译阶段发现语法错误。
  - 运行错误，如错误的数学运算，使用了 BY 语句但输入数据集没有正确排序，文件名错误等。可以导致程序出错终止，也可能只在日志窗口显示错误或注 (NOTE) 然后继续运行。
  - 数据错误，为读取数据时不符合要求，比如应该读取数值的时候输入了字符型。
  - 句法错误，也是一种运行错误，为语句格式正确但使用错误，比如调用函数时自变量个数不匹配，该用字符型变量的地方使用了数值型，调用为定义的库名。

控制程序质量的自检表

- 为控制程序质量，可以进行如下检查：
- 检查语法，特别是：
  - 每个语句以分号结尾；
  - 字符串的两个撇号必须匹配；
  - 大多数 SAS 语句以关键字开始，不要拼写错；
  - DO 和 SELECT 语句要以 END 语句结尾。
- 检查程序语句的次序。比如，INFILE 语句必须在 INPUT 语句之前出现。每个数据步和过程步以 RUN 语句结尾。
- 检查 INPUT 语句和数据。尤其是数值型和字符型的分别不要错。选用 INPUT 语句的自由格式、列格式还是有格式需要与输入数据情况相符。

3.10.2 DATASETS 过程

DATASETS 过程

- DATASETS 过程管理数据库中的 SAS 文件。
- 可以列出数据库中的所有数据集；
- 删除、重命名、修复数据集；
- 查询和修改数据集以及数据集中的变量的属性；
- 在库之间复制；
- 在数据集末尾附加其他数据集内容；
- 创建和管理密码、索引 (indexes)、审核文件 (audit files)、一致性约束。
- DATASETS 过程的部分功能与 CONTENTS、APPEND、COPY、CATALOG 等过程的功能重叠，有时单独调用这些过程更易用。

列出一个逻辑库中的所有数据集

- 要列出 SAMP 库中所有数据集详细信息，程序如

```
PROC DATASETS LIBRARY=samp
    MEMTYPE=DATA NOLIST;
  CONTENTS DATA=_all_ DETAILS NODS;
RUN;QUIT;
```

- 结果显示在输出窗口。

- 程序中 PROC DATASETS 的 LIBRARY= 指定要列表的数据库; MEMTYPE=DATA 选项要求仅列出数据集；NOLIST 选项取消缺省的数据集列表（否则自动有数据集列表到日志窗口）。
- CONTENTS 语句要求列出指定的数据集，`DATA=_all_` 选项指定列出库中所有数据集。DETAILS 选项要求列出观测个数、变量个数等信息，NODS 选项说明不列出数据集中的变量信息，否则会列出数据集的变量信息。
- 还可以使用 PROC CONTENTS 列出逻辑库中所有数据集。

- 程序如

```
PROC CONTENTS DATA=samp._ALL_ NODS;
RUN;
```

其中 NODS 选项取消数据集中变量信息的显示。

把逻辑库的数据集列表存入数据集

- 为了把某逻辑库中的数据集列表保存到一个数据集中，可以使用 PROC SQL 程序查询隐含的逻辑库 DICTIONARY 中的 TABLES 表。
- 程序如

```
PROC SQL;
  CREATE TABLE sampdir AS
    SELECT *
      FROM DICTIONARY.TABLES
      WHERE LIBNAME='SAMP';
QUIT;
```

列出一个逻辑库中指定数据集的变量

- 要列出 SAMP 库中 CLASS 数据集的变量信息，程序如

```
PROC DATASETS LIBRARY=samp
    MEMTYPE=DATA NOLIST;
  CONTENTS DATA=class OUT=vlist;
RUN;QUIT;
```

- 结果显示在输出窗口，同时用 OUT= 选项存入了 VLIST 数据集中。
- 程序中 CONTENTS 语句用 DATA= 选项指定要列出哪个数据集的变量信息，OUT= 选项指定输出数据集。
- 显示变量名信息时缺省按变量名排序而不是按数据集中变量次序排序，可以使用 VARNUM 选项要求按数据集中变量次序排列。

用 CONTENTS 过程查看数据集结构

- 要列出 SAMP 库中 CLASS 数据集的变量信息,也可以用 CONTENTS 过程，程序如

```
PROC CONTENTS data=samp.class;
RUN;
```

- 程序中 PROC CONTENTS 语句用 DATA= 选项指定要列出哪个数据集的变量信息。
- 结果显示在输出窗口。
- 如果还要把变量列表保存为数据集，只要用 OUT= 选项，如

```
PROC CONTENTS data=samp.class OUT=vlist;
RUN;
```

- 变量列表存入了 VLIST 数据集中。
- 可以使用 VARNUM 选项要求显示各变量信息时按数据集中变量次序排列。

数据集改名

- 用 CHANGE 语句对 LIBRARY= 指定的逻辑库中的数据集改名，格式为 “旧名字 = 新名字”。
- 程序如

```
PROC DATASETS LIBRARY=samp
    MEMTYPE=DATA NOLIST;
  CHANGE c9501f=c9501nv c9501m=c9501nan;
RUN;QUIT;
```

删除数据集

- 用 DELETE 语句删除数据集或其它 SAS 文件。如
- 程序如

```
PROC DATASETS LIBRARY=work NOLIST;
  DELETE class gpa;
RUN;QUIT;
```

- 可以在斜杠后用 MEMTYPE= 指定要删除的类型，如

```
PROC DATASETS LIBRARY=work NOLIST;
  DELETE c9501v / MEMTYPE=VIEW;
RUN;QUIT;
```

- 可以删除多个数据集，数据集列表可以使用简写，如 “test1-test5” 表示编号的 5 个数据集，或 “t:” 表示所有名字以字母 t 开头的数据集。

复制逻辑库

- 为了把某个逻辑库全部内容复制到另一个逻辑库，使用 COPY 语句，其中用 OUT= 指定目的地，IN= 指定来源。如

```
PROC DATASETS NOLIST;
  COPY OUT=WORK IN=SAMP;
RUN;QUIT;
```

- 在 COPY 语句中可以用 MEMTYPE=DATA 要求仅复制数据集。

复制数据集

- 为了有选择地复制数据集，在 COPY 语句之后用 SELECT 语句选择要复制数据集，或用 EXCLUDE 语句指定不需复制的数据集。如

```
PROC DATASETS NOLIST;
  COPY OUT=WORK IN=SAMP;
  SELECT c9501 class gpa;
RUN;QUIT;
```

- 在 SELECT 语句中用数据集选项 MEMTYPE=DATA 选择数据集，用 MEMTYPE=VIEW 选择视图，如

```
  SELECT gpa(MEMTYPE=DATA);
```

在数据集末尾增添内容

- 用 APPEND 语句在一个主数据集末尾增加另一个数据集的内容。APPEND 语句中 BASE= 选项指定主数据集，DATA= 选项指定要添加进来的数据集。
- 如果 BASE= 或 DATA= 没有指定库名，则缺省为 PROC DATASETS 语句中 LIBRARY= 选项中指定的逻辑库。
- 如果 BASE= 指定的数据集不存在，则把用来添加的数据集内容复制生成此数据集。
- 当主数据集较大时，用 APPEND 语句添加比使用数据步的 SET 语句做纵向合并效率更高。
- 程序如：

```
PROC DATASETS LIBRARY=work
    MEMTYPE=DATA NOLIST;
  APPEND BASE=c9501fm DATA=samp.c9501m;
  APPEND BASE=c9501fm DATA=samp.c9501f;
RUN;QUIT;
```

  首先把 SAMP.C9501M 内容复制到 WORK.C9501FM 中，然后又把 SAMP.C9501F 内容添加到 WORK.C9501FM 尾部。

- DATA= 指定的添加数据集可以使用 WHERE= 数据集选项来选择观测。

修改数据集变量属性

- 用 MODIFY 语句指定要修改的数据集，然后用 ATTRIB 语句修改变量属性，包括变量标签 (LABEL)、变量输出格式 (FORMAT) 和变量输入格式 (INFORMAT)。
- 程序如

```
PROC DATASETS LIBRARY=WORK NOLIST;
  MODIFY class;
  ATTRIB name LABEL=' 姓名'
    weight FORMAT=8.2;
QUIT;
PROC PRINT DATA=class label;
RUN;
```

修改数据集变量名

- 用 MODIFY 语句指定要修改的数据集，然后用 RENAME 语句对变量改名，格式为 “旧名 = 新名”。
- 程序如

```
PROC DATASETS LIBRARY=WORK NOLIST;
  MODIFY class;
  RENAME height=h weight=w;
QUIT;
PROC PRINT DATA=class;run;
```

3.10.3 RANK 过程

RANK 过程

- RANK 过程计算数据集变量的秩（名次）。
- PROC RANK 语句中用 DATA= 指定输入数据集，用 OUT= 指定输出数据集。

- 用 VAR 语句指定用来排名次的变量名，用 RANKS 语句规定用来保存名次（秩）的变量名。
- 名次缺省为由低到高排列，在 PROC RANK 语句中加 DESCENDING 选项可以改为由高到低排列。
- 同名次时，缺省使用平均名次，比如，第 2，3 名变量值相同，则名次同取为 2.5。
- 在 PROC RANK 语句中用 TIES= 选项可以规定同名次时的处理方法。TIES=LOW 使用低名次（第 2，3 名变量值相同时，都算是第 2 名），TIES=HIGH 使用高名次，TIES=MEAN 使用平均名次（缺省）。

- 如：

```
PROC RANK DATA=samp.class OUT=rc;
  VAR age;
  RANKS agerank;
run;
```

- 下面的程序的年龄名次是由高到低排名，且同名次时取小名次值：

```
PROC RANK DATA=samp.class OUT=rc
    DESCENDING TIES=LOW;
  VAR age;
  RANKS agerank;
run;
```

- 用 BY 语句可以分组计算名次。
- RANK 过程可以计算多种秩统计量，比如名次除以 n 或 $n+1$ 后变成 $[0,1]$ 之间的分数名次，用标准正态分位数变换得到正态得分 (normal score)，等等。

3.10.4 STANDARD 过程

STANDARD 过程

- STANDARD 过程对数据集变量进行标准化, 使得数据集变量的样本均值变成 M, 标准差变成 S:

$$Y_i = M + S \cdot \frac{X_i - \bar{X}}{S_x}$$

  其中 $\bar{X}$ 和 S_X 是原始变量 X 的样本均值和样本标准差。

- 结果保存到 OUT= 指定的输出数据集，变换后的变量名使用原变量名。

- 用 MEAN= 和 STD= 指定目标 M 和 S。

- 如:

```
PROC STANDARD DATA=samp.class OUT=sc
    MEAN=0 STD=1;
  VAR age;
run;
```

3.10.5 FORMAT 过程

FORMAT 过程

- 用 PROC FORMAT 定义用户自己的输出格式和输入格式。
- VALUE 语句定义不同值、值范围的输出方式。
- PICTURE 语句定义数字的特殊输出格式。
- INVALUE 语句定义输入格式。
- 还可以把格式保存到数据集、从数据集生成输出格式。
- 自定义的格式和输入格式也分为数值型和字符型，调用时都要用句点结尾，字符型格式和输入格式名以 $ 符号开头。定义时不需要句点。
- 如果输出格式用于数值型变量，则输出格式是数值型的。
- 如果输出格式用于字符型变量，则输出格式是字符型的。

- 如果输入格式转换后的结果是数值型的，此输入格式称为数值型输入格式。
- 如果输入格式转换后的结果是字符型的，此输入格式称为字符型输入格式。

一一变换的输出格式

- 下例中定义了数值型型输出格式 sexotf., 把 1 显示成' 男', 把 2 显示成' 女':

```
proc format;
  VALUE sexotf
    1=' 男'
    2=' 女';
run;
data sexd;
  input sex;
  format sex sexotf.;
  cards;
1
2
;
run;
proc print;run;
```

- 下例中定义了字符型输出格式 “$sexf.”，把'F' 转换为' 女', 把'M' 转换为' 男'：

```
proc format;
  VALUE $sexf
    'F'=' 女'
    'M'=' 男';
run;
data sexd;
  input sex $;
  format sex $sexf.;
```

```
  cards;
F
M
;
run;
proc print;run;
```

——变换的输入格式

- 下例中定义了字符型输入格式 $sexotinf., 把 1 转换为' 男', 把 2 转换为' 女':

```
proc format;
  INVALUE $sexotinf
    1=' 男'
    2=' 女';
run;
data sexd;
  input sex :$sexotinf. @@;
  cards;
1 2
;
run;
proc print;run;
```

- 下例中定义了数值型输入格式 “sexinf.”，把' 男' 转换为 1, 把' 女' 转换为 2：

```
proc format;
  INVALUE sexinf
    ' 男'=1
    ' 女'=2;
run;
data sexd;
  input sex :sexinf. @@;
```

```
  cards;
男 女
;
run;
proc print;run;
```

输入范围的转换

- 输入格式可以把一个范围的输入转换为一个值。如

```
proc format;
  INVALUE trial
    'A'-'M'=1
    'N'-'Z'=2
    1-999,1001-1999,2001-2999=3
    9999=.
    other=_error_;
run;
```

- OTHER 表示所有其他输入。
- 等号右边_ERROR_ 表示该类输入作为错误数据。
- 等号右边的_SAME_ 表示该类输入保持原样不变。

值范围的写法

- 值范围写成 “开始值-结束值”。
- 可以用 LOW 表示最小值，用 HIGH 表示最大值。
- 用 “a< −b” 表示 a 到 b 范围但不含 a，用 “a− <b” 表示 a 到 b 范围但不含 b。
- VALUE 和 INVALUE 不需要穷举所有范围，未包含的范围也没有 OTHER 项时，自动使用原始值。

临时定义

- 以上 PROC FORMAT 定义的格式和输入格式保存在临时的 SAS catalog WORK.FORMATS 中，仅能用于当前的 SAS 会话中。
- 要使用这些格式和输入格式，只要在数据步中直接用在 FORMAT 语句、INFORMAT 语句和 INPUT 语句中，或过程步 FORMAT 语句中。

永久定义

- 在 PROC FORMAT 语句中用 LIBRARY=libref 指定一个逻辑库，可以把此 FORMAT 过程定义的格式和输入格式保存到此逻辑库的名为 FORMATS 的 SAS catalog 中。
- 为了调用这样保存的格式和输入格式，需要先用 OPTIONS 语句的 FMTSEARCH 选项指定要搜索的逻辑库，才能在数据步和过程步中用格式名直接使用这些格式和输入格式：

```
proc format LIBRARY=mylib;
 ....
run;
OPTIONS FMTSEARCH=(mylib);
```

- 另外，如果格式保存到了名为 LIBRARY 的逻辑库中的 FORMATS catalog 中，不需要 OPTIONS 语句的 FMTSEARCH 选项也可以直接访问 LIBRARY.FORMATS 中保存的格式。

列出已定义的格式

- 在 PROC FORMAT 语句中加上 FMTLIB 选项可以显示已定义的格式，如

```
proc format fmtlib;
run;
```

- 为了列出保存在永久逻辑库中的 SAS catalog 中保存的格式，在 PROC FORMAT 语句中用 LIBRARY= 指定逻辑库名，如果保存格式的 SAS catalog 没有命名为 FORMATS，可以在 LIBRARY= 后面指定 “库名.catalog 名”。

把定义的格式转换为数据集

- 在 PROC FORMATS 语句中用 CNTLOUT= 指定一个数据集，可以把已定义的格式保存到数据集中。如

```
proc format cntlout=tfm;
run;
proc print;run;
```

- 用 LIBRARY= 指定一个逻辑库可以把保存在该库中的 FORMATS catalog 中的格式转换到数据集中。
- 转换得到的数据集可以用在 PROC FORMATS 的 CNTLIN= 中作为输入，再转换成格式。

3.10.6 REPORT 过程

REPORT 过程

- REPORT 过程是比 PRINT 过程功能更强大的报表制作过程, 包含了 PRINT 过程，TABULATE 过程，MEANS 过程和数据步的部分功能。
- 可以列出所有观测；简洁格式分组显示所有观测；分组计算总和或其它统计量；按照分组变量值显示到不同列，等等。
- 用 COLUMN 语句列出要显示的变量，用 DEFINE 语句详细规定变量的作用以及如何显示。

显示所有观测

- 如下程序显示 C9501 数据集所有观测：

```
PROC REPORT DATA==samp.c9501 NOWINDOWS;
RUN;
```

- PROC REPORT 语句的 NOWINDOWS 选项要求程序非交互式运行，否则会出现表格设计窗口要求用户交互设计。

显示总和

- 如果数据集中仅有数值型变量，以上简单程序不显示每个观测而是显示数值型变量的总和。
- 如

```
PROC REPORT DATA==samp.c9501(
  KEEP=math chinese) NOWINDOWS;
RUN;
```

- 这是因为，在 REPORT 过程中，数值型变量的缺省作用是分析变量，所以在不存在起其它作用的变量时，只能对数值型变量显示缺省统计量—总和。

用 COLUMN 语句指定要显示的变量

- 用 COLUMN 语句指定要显示的变量和排列顺序，如：

```
PROC REPORT DATA=samp.c9501 NOWINDOWS;
  COLUMN name sex math chinese;
RUN;
```

排序显示

- 用 DEFINE 语句指定变量的作用和列标题等规定。
- 字符型变量缺省作用是 DISPLAY，每个观测显示一行，不排序。
- 变量可以指定为 ORDER 作用，这时数据集每个观测显示为一行，按此变量次序排列，此变量每个值仅显示一次，重复值显示成空白，不需要数据集预先已排序。
- 也可以指定多个 ORDER 作用的变量，这时排序按其在 COLUMN 语句的先后次序决定主要排序变量和次要排序变量。
- 如

```
PROC REPORT DATA=samp.class NOWINDOWS;
  COLUMN sex age name height weight;
  DEFINE sex / ORDER;
  DEFINE age / ORDER;
RUN;
```

- 在 DEFINE 语句中用 ORDER 和 DESCENDING 选项指定此变量为降序排列变量。
- 变量值的缺省次序为显示值（经过输出格式转换后的值）的次序。用 ORDER=DATA 规定次序为数据中出现的先后次序，ORDER=INTERNAL 规定为数据内部存储值的次序，ORDER=FREQ 为数据值频次由少到多的次序。
- 如

```
PROC REPORT DATA=samp.class NOWINDOWS;
  COLUMN sex age name height weight;
  DEFINE sex / ORDER ORDER=FREQ DESCENDING;
  DEFINE age / ORDER DESCENDING;
RUN;
```

分组汇总

- 数值型变量的缺省作用是分析变量，在 DEFINE 语句中用 ANALYSIS 选项规定变量作用为分析变量。用 SUM, MEAN 等关键字指定要做的汇总统计量。
- 统计量名称包括 N, NMISS, MEAN, STD, SUM, VAR, CV, PCTN, PCTSUM, MEDIAN, Q1, Q3, P1, P5, P10, P90, P95, P99 等。
- 在 DEFINE 语句中用 GROUP 选项规定变量作用为分组变量, 分析变量在每组内计算汇总统计量。

- 如

```
PROC REPORT DATA=samp.class NOWINDOWS;
  COLUMN sex height weight;
  DEFINE sex / GROUP;
  DEFINE height / ANALYSIS MEAN;
  DEFINE weight / ANALYSIS MEAN;
RUN;
```

分组变量值排序

- GROUP 选项同时要求结果显示按分组变量值排序。
- 为了使用不同次序显示，可以使用 DESCENDING 选项和 ORDER= 选项。
- 如

```
PROC REPORT DATA=samp.class NOWINDOWS;
  COLUMN sex age height weight;
  DEFINE sex / GROUP ORDER=FREQ DESCENDING;
  DEFINE age / GROUP;
  DEFINE height / ANALYSIS MEAN;
  DEFINE weight / ANALYSIS MEAN;
RUN;
```

计算并显示新变量

- REPORT 过程可以从输入数据计算新变量并显示为一列，新变量不必保存在数据集中。
- 为此，需要把新变量列入 COLUMN 语句中，并有一个 DEFINE 语句说明新变量作用为 COMPUTED，并使用一个计算代码块 (compute block) 定义新变量的计算方法。
- 计算代码块以 “COMPUTE 新变量” 语句开始，以 ENDCOMP 语句结尾，中间可以使用类似数据步的语句以及一些 REPORT 过程特有语句。

- 如下例计算了体重与身高比值作为新变量：

```
PROC REPORT DATA=samp.class NOWINDOWS;
  COLUMN name height weight ratio;
  DEFINE name / ORDER;
  DEFINE height / DISPLAY;
  DEFINE weight / DISPLAY;
  DEFINE ratio / COMPUTED FORMAT=6.2;
  COMPUTE ratio;
    ratio = weight / height;
  ENDCOMP;
RUN;
```

输出到数据集

- 在 PROC REPORT 语句中用 OUT= 指定输出数据集，输出数据集格式与显示内容类似。
- 如

```
PROC REPORT DATA=samp.class
    OUT=repd NOWINDOWS;
  COLUMN sex age height weight;
  DEFINE sex / GROUP;
  DEFINE age / GROUP;
  DEFINE height / ANALYSIS MEAN;
  DEFINE weight / ANALYSIS MEAN;
RUN;
PROC PRINT;RUN;
```

统计量数值格式

- 对 ANALYSIS 作用的变量，在 DEFINE 语句中用 FORMAT= 选项指定结果显示格式。
- 如

```
PROC REPORT DATA=samp.class NOWINDOWS;
  COLUMN sex age height weight;
  DEFINE sex / GROUP;
  DEFINE age / GROUP;
  DEFINE height / ANALYSIS MEAN FORMAT=8.2;
  DEFINE weight / ANALYSIS MEAN FORMAT=8.2;
RUN;
```

- 也可以使用一个 FORMAT 语句指定若干个变量显示格式。

空白和对齐方式

- PROC REPORT 语句的 SPACING= 选项指定列间的空格个数。DEFINE 语句的 SPACING= 选项指定该变量左侧与前一项分隔使用的空格数。
- PROC REPORT 语句的 COLWIDTH= 选项指定每一列的字符数, DEFINE 语句的 WIDTH= 指定该列占用的字符数。
- 以上选项仅对 LISTING 输出, 即文本型输出有效。
- DEFINE 语句的 CENTER、LEFT、RIGHT 选项指定该栏居中对齐、左对齐、右对齐。
- 如

```
PROC REPORT DATA=samp.class
    NOWINDOWS SPACING=5;
  COLUMN name sex age;
  DEFINE name / WIDTH=10 RIGHT;
  DEFINE sex / WIDTH=2;
  DEFINE age / WIDTH=2;
RUN;
```

自定义列标题

- 在 DEFINE 语句中用字符串作为选项可以自定义列标题。如

```
PROC REPORT DATA=samp.class NOWINDOWS ;
  COLUMN sex height;
  DEFINE sex / GROUP ' 性 别' WIDTH=2;
  DEFINE height / ' 身高' ANALYSIS
    MEAN FORMAT=8.2;
RUN;
```

多行列标题

- 列标题中用'/' 表示换行，或者把列标题用两个或多个字符串表示也可以在上下多行显示。如

```
PROC REPORT DATA=samp.class NOWINDOWS ;
  COLUMN sex height weight;
  DEFINE sex / GROUP ' 性 别' WIDTH=2;
  DEFINE height / ' 身高/平均值' ANALYSIS
    MEAN FORMAT=8.2;
  DEFINE weight / ' 体重' ' 平均值' ANALYSIS
    MEAN FORMAT=8.2;
RUN;
```

- 可以用 PROC REPORT 的 SPLIT= 选项指定在列标题字符串中表示换行的字符。

标题栏下划线和空行

- PROC REPORT 的 HEADLINE 选项在列标题行下划线，HEADSKIP 选项在列标题行下空行。

- 如

```
PROC REPORT DATA=samp.class
    NOWINDOWS HEADLINE HEADSKIP;
  COLUMN sex height;
  DEFINE sex / GROUP ' 性 别' WIDTH=2;
  DEFINE height / ' 身高' ANALYSIS
    MEAN FORMAT=8.2;
RUN;
```

合并栏目标题

- 在 COLUMN 语句中用圆括号把一个字符串和多个输出项组合在一起，可以使得多个输出项上方共享此字符串作为合并栏目标题。
- 如

```
PROC REPORT DATA=samp.class NOWINDOWS;
  COLUMN sex age (' 平均值' height weight);
  DEFINE sex / GROUP;
  DEFINE age / GROUP;
  DEFINE height / ANALYSIS MEAN;
  DEFINE weight / ANALYSIS MEAN;
RUN;
```

计算多个统计量

- 在 COLUMN 语句用 “数据集变量名 = 别名” 的格式可以为一个数据集变量指定多个别名，对原始变量名和每个别名可以使用单独的 DEFINE 语句，这样可以对一个变量计算多种统计量。
- 如

```
PROC REPORT DATA=samp.class NOWINDOWS;
  COLUMN sex height
    height=htmin height=htmax;
  DEFINE sex / ' 性别' GROUP WIDTH=4;
```

```
  DEFINE height / ' 平均身高' ANALYSIS
    MEAN WIDTH=10 FORMAT=6.2;
  DEFINE htmin / ' 最低' ANALYSIS MIN
    FORMAT=5.1;
  DEFINE htmax / ' 最高' ANALYSIS MAX
    FORMAT=5.1;
RUN;
```

分类变量横列

- 在 DEFINE 中用 ACROSS 选项使得分类变量横列显示，每个值占一列，统计该值的频数。
- 特殊变量 N 代表组内的非缺失观测个数（如果有分组变量）或观测非缺失观测总数。
- 如

```
PROC REPORT DATA=samp.class NOWINDOWS;
  COLUMN sex N;
  DEFINE sex / ACCROSS WIDTH=2;
  DEFINE N / ' 总计';
RUN;
```

- 这像是 PROC TABULATE 过程中把分类变量放在列维的效果。

交叉分类例子

- 如果同时包含分组变量 (GROUP 选项)，又有横列分类变量 (ACROSS 选项)，就形成交叉分类。如

```
PROC REPORT DATA=samp.class NOWINDOWS;
  COLUMN age sex N;
  DEFINE age / GROUP;
  DEFINE sex / ACCROSS WIDTH=2;
  DEFINE N / ' 总计';
RUN;
```

- 用 DEFINE 语句的 ACROSS 选项指定的横列分类变量，可以在 COLUMN 语句中该变量后面写逗号然后写一个或多个按其分类后概括统计的变量，多个时用圆括号包围。
- 如

```
PROC REPORT DATA=samp.class NOWINDOWS;
  COLUMN sex, (height weight);
  DEFINE sex / ACROSS WIDTH=2;
  DEFINE height / ' 平均身高'
    ANALYSIS MEAN FORMAT=8.2;
  DEFINE weight / ' 平均体重'
    ANALYSIS MEAN FORMAT=8.2;
RUN;
```

- 这类似于 PROC TABULATE 中列维的分类变量与分析变量用 “*” 连接的效果。
- 当然，也可以同时有 GROUP 变量形成交叉分类。

分组变量横列时的列名

- 用 ACROSS 选项指定横列分组变量后，输出的一列不再对应于原数据集中的一个变量。如果需要引用这样的列（比如在计算代码块中），可以用_C1_, _C2_ 这样的特殊变量名，其中的数字是列的序号。
- 如

```
PROC REPORT DATA=samp.class NOWINDOWS;
  COLUMN sex, (height weight) ratio;
  DEFINE sex / ACROSS WIDTH=2;
  DEFINE height / ' 平均身高'
    ANALYSIS MEAN FORMAT=8.2;
  DEFINE weight / ' 平均体重'
    ANALYSIS MEAN FORMAT=8.2;
  DEFINE ratio / ' 男女平均身高比'
```

```
    COMPUTED FORMAT=6.2;
  COMPUTE ratio;
    ratio = _C3_ / _C1_;
  ENDCOMP;
RUN;
```

每组额外显示

- 用 BREAK AFTER 语句指定某个分组变量，在此分组变量每个组末尾显示额外信息；用 BREAK BEFORE 在指定的分组变量每个组开头显示额外信息。
- RBREAK AFTER 规定如何对所有输出最后额外显示，RBREAK BEFORE 规定如何对所有输出开头额外显示。
- BREAK 的 SUMMARIZE 选项要求显示分析变量在本组的概括统计量。
- SKIP 选项使得额外输出后空行。
- OL 选项使得额外输出数据行上面画线，DOL 选项使得额外输出数据行上面画双线。
- SUPPRESS 选项可以使额外输出行的分组值不显示。

- 如

```
PROC REPORT DATA=samp.class NOWINDOWS;
  COLUMN sex age height weight;
  DEFINE sex / GROUP;
  DEFINE age /GROUP;
  DEFINE height / ANALYSIS MEAN FORMAT=8.2;
  DEFINE weight / ANALYSIS MEAN FORMAT=8.2;
  BREAK AFTER sex / SUMMARIZE SKIP OL;
  RBREAK AFTER / SUMMARIZE SKIP DOL;
RUN;
```

定制的每组额外显示

- BREAK 语句指定的额外显示是固定格式的。
- 用 “COMPUTE AFTER 分组变量” 或 “COMPUTE BEFORE 分组变量” 可以开始计算代码块并在其中规定如何在每组最后或开始输出自定义的显示内容，也可以做一些计算。
- 在计算代码块中用 LINE 语句和与数据步中 PUT 语句类似的语法自定义显示内容。变量的统计量值访问如 “height.mean”，但这时变量的 DEFINE 语句中必须指定相同的统计量。变量值必须跟随输出格式。
- 为了在整个表的末尾或开头增加定制输出，用不指定变量名的 COMPUTE AFTER 计算代码块或 COMPUTE BEFORE 计算代码块。
- 如

```
PROC REPORT DATA=samp.class NOWINDOWS;
  COLUMN sex name height weight;
  DEFINE sex / ORDER;
  DEFINE name / DISPLAY;
  DEFINE height / ANALYSIS MEAN FORMAT=8.2;
  DEFINE weight / ANALYSIS MEAN FORMAT=8.2;
  COMPUTE AFTER sex;
    LINE @11 ' 平均身高 (' sex $2. '): '
      @25 height.mean 6.2
      @41 ' 平均体重 (' sex $2. '): '
      @55 weight.mean 6.2;
  ENDCOMP;
  COMPUTE AFTER;
    LINE '';
    LINE @11 ' 总平均身高:   ' height.mean 6.2
      @41 ' 总平均体重:   ' weight.mean 6.2;
  ENDCOMP;
RUN;
```

NOPRINT 选项

- 计算代码块中访问的统计量或别名必须出现在 COLUMN 语句和 DEFINE 语句中。
- 有些统计量或别名仅需要出现在概括行中，不需要对具体数据行显示，可以在 DEFINE 语句中指定 NOPRINT 选项。
- 在计算代码块中访问别名时只需要使用别名本身，而不能用 “别名. 统计量名” 这样的格式。
- 例如，下例显示按性别分组的身高并在概括行中显示每组的最低和最高：

```
PROC REPORT DATA=samp.class NOWINDOWS;
  COLUMN sex height
    height=htmin height=htmax;
  DEFINE sex / ' 性别' ORDER WIDTH=4;
  DEFINE height / ' 身高' ANALYSIS SUM
    FORMAT=6.2;
  DEFINE htmin / ' 最低'
    ANALYSIS MIN NOPRINT;
  DEFINE htmax / ' 最高'
    ANALYSIS MAX NOPRINT;
  COMPUTE AFTER sex;
    LINE @21 ' 最低身高:' htmin 5.1
      +3 ' 最高身高:' htmax 5.1;
    LINE ' ';
  ENDCOMP;
RUN;
```

第四章　SAS 的基本统计分析功能

SAS 的基本统计分析功能

- 基本的统计检验;
- 线性回归;
- 方差分析;
- 拟合优度检验;
- 列联表检验，等等。
- 用程序和 INSIGHT 及 Analyst 解决。

4.1 几种假设检验

4.1.1 单总体 t 检验和 p 值

单总体 t 检验

- 设总体 $X \sim \mathrm{N}(\mu, \sigma^2)$, μ, σ^2 未知。
- 给定检验水平 α，对常数 μ_0 要检验

$$H_0 : \mu = \mu_0 \longleftrightarrow H_a : \mu \neq \mu_0,$$

- 在 H_0 成立时

$$t = \frac{\bar{X} - \mu_0}{S/\sqrt{n}} \sim t(n-1)$$

- 检验的否定域为 $W = \{|t| > \lambda\}$, 其中 λ 为 $t(n-1)$ 分布的双侧 α 分位数。

p 值

- 设 t 统计量值为 a，则 $|a| > \lambda$ 时否定 H_0，且 $|a|$ 越大说明零假设 H_0 越不可能成立。
- 令

$$p = \Pr(|t(n-1)| > |a|)$$

- 则 $|a| > \lambda$ 当且仅当 $p < \alpha$(图示);
- 且 p 越小，$|t|$ 越大，说明零假设 H_0 越不可能成立。
- 我们称这样的 p 为检验的 p **值**，当且仅当 p 值小于检验水平 α 时否定 H_0。

p 值 (续)

- 对于其它检验可以类似定义 p 值。
- 对单边检验

$$H_0 : \mu = \mu_0 \longleftrightarrow H_a : \mu > \mu_0,$$

则否定域为 $W' = \{t > \lambda'\}$, λ' 为 $t(n-1)$ 分布的上侧 α 分位数;

- 这时 p 值的定义为

$$p = \Pr(t(n-1) > a),$$

- 当且仅当 $p < \alpha$ 时 $t > \lambda'$。
- 注意: $a < 0$ 时 p 值至少为 0.5, 不能否定零假设。

SAS 的 t 检验

- 在 SAS 中用 TTEST 过程可以进行某个变量均值为 (μ_0) 的 t 检验, 称为单样本 t 检验。
- 例如，为了检验某类法律案件的审理是否平均经过 80 天，随机选取了 20 个这样的案件，然后用 proc ttest 执行单样本 t 检验。
- 程序如

```
title 'One-Sample t Test';
data casetime;
  input time @@;
  datalines;
43  90  84  87  116   95  86   99   93  92
121 71  66  98   79  102  60  112  105  98
;
run;
proc ttest data=casetime h0=80 alpha=0.1;
   var time;
run;
```

- 用 VAR 语句指定要检验的变量，用 PROC TTEST 的 H0= 选项指定 H_0 下的均值 μ_0，用 PROC TTEST 语句选项 ALPHA= 指定计算置信区间时所用的置信度 (1-alpha)。

- 结果包括基本统计量、均值和标准差的置信区间和 t 检验结果。
- 均值估计为 89.85，均值的置信度 90% 的置信区间为 $(82.447, 97.253)$。
- t 检验统计量值为 2.30, p 值为 0.0329, 在水平 $\alpha = 0.10$ 显著。应否认此类案件审理时间平均为 80 天的假设。
- 在 Analyst 中也可以进行单样本 t 检验: “统计—假设检验—均值的单样本 T 检验”。

4.1.2 正态性检验

正态性检验

- 使用 t 检验需要总体来自正态分布。
- 在 PROC UNIVARIATE 语句中加上 NORMAL 选项可以进行正态性检验。
- 零假设是总体分布为正态分布。
- 例如，检验上面的案件审理时间是否服从正态分布:

```
proc univariate data=casetime normal;
  var time;
run;
```

- 结果主要用 Shapiro-Wilk 的结果。p 值 0.6042 不显著，不拒绝总体来自正态分布的零假设，可以把总体作为正态分布处理。

用 Analyst 进行分布检验

- 在 Analyst 中选 “Statistics - Descriptive - Distributions” 调出分布研究对话框, 选了分析变量后按 Fit 钮可以要求进行分布拟合, 最后的结果中包含了分布拟合的三种检验 (为上面 Univariate 的四种检验的后三种)。
- 还可以要求画盒形图、直方图 (要求拟合时直方图上面会附加拟合的正态密度曲线)、概率图、QQ 图。
- 可以进行对数正态、指数和 Weibull 分布的拟合和检验。

4.1.3 两独立样本的均值检验

两独立样本的均值检验

- 假设两组样本分别来自两个独立总体, 需要检验两个总体的均值或中心位置是否一样。
- 如果两个总体都分别服从正态分布, 可以使用 TTEST 过程，这种检验叫做**两样本 t 检验**。
- 例如，SASUSER.GPA 数据集中男生和女生的 SATM 是否有显著差异的检验:

```
proc ttest data=sasuser.gpa;
  class sex;
  var satm;
run;
```

- 用 CLASS 语句指定分组变量, 用 VAR 语句指定要比较的变量。

两样本 t 检验的结果

- 两个样本各自的简单统计量；
- 两样本 t 检验结果，包括方差相等条件下的检验与方差不要求相等条件下的检验 (Satterthwaite)；
- 方差相等条件下，使用合并的方差估计 (Pooled variance)；方差不等条件下，构造类 F 统计量并用 F 分布近似。

检验统计量

- 方差相等条件下的检验统计量：

$$t = \frac{\bar{x} - \bar{y}}{S_{\text{pool}}\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \sim t(n_1 + n_2 - 2) \text{ (在 } H_0 \text{ 下)}.$$

$$S^2_{\text{pool}} = \frac{1}{n_1 + n_2 - 2}\left[\sum (x_i - \bar{x})^2 + \sum (y_i - \bar{y})^2\right]$$

- 方差不等条件下的检验统计量:

$$t = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{S_x^2}{n_1} + \frac{S_y^2}{n_2}}} \sim t(\mathrm{df}_{\mathrm{Satterthwaite}}) \text{ (在 } H_0 \text{ 下)}.$$

$$\mathrm{df}_{\mathrm{Satterthwaite}} = \frac{(n_1 - 1)(n_2 - 1)}{(n_1 - 1)(1 - c)^2 + (n_2 - 1)c^2}$$

$$c = \frac{S_x^2/n_1}{\frac{S_x^2}{n_1} + \frac{S_y^2}{n_2}}.$$

双边和单边检验

- 双边检验的 p 值小于 0.0001，所以男女生的 SATM 分数有显著差异。
- 男生平均分为 565.03, 女生平均分为 611.77，所以是女生成绩更好。
- 如果检验是单边的，则对立假设应为女生平均值较高，p 值是双边 p 值的一半。
- Analyst 中菜单 “Statistics – Hypothesis Tests – Two-Sample t-test for Means”。可以直接作单边检验。

检验方法的前提条件

- 两独立样本；
- 各自正态分布；
- 合并方差方法要求两总体方差相等，Satterthwaite 方法不要求。

Wilcoxon 秩和检验

- GPA 分数不服从正态分布，不能用 t 检验。
- NPAR1WAY 过程加 Wilcoxon 选项进行 Wilcoxon 秩和检验：只要求独立性，不要求分布，另外只关心数据大小次序而不关心数据值，可以用于有序型数据。
- 秩和检验法解释。
- 如: 男女生 GPA 分数比较

```
proc npar1way data=sasuser.gpa wilcoxon;
  class sex;
  var gpa;
run;
```

- 也使用 VAR 语句和 CLASS 语句。

Wilcoxon 秩和检验结果

- 两样本的基本统计量，包括秩和、平均秩和等。从秩和看可能男生好。
- 秩和检验结果，统计量用正态近似，检验 p 值为 Prob > | Z | = 0.5978, 没有显著差异。
- Kruskal-Wallis，不用。
- 如果考虑单侧检验，对立假设为男生分数高于女生分数，p 值为双侧 p 值的一半 One-Sided Pr > Z=0.2989，不显著。
- 在 Analyst 中用 “Statistics – ANOVA – Nonparametric One-Way ANOVA” 进行 Wilcoxon 秩和检验。

4.1.4 成对总体均值检验

成对总体均值检验

- 成对总体均值检验用于比较同一总体两个指标的比较。
- 如：为了研究某种刺激对血压的影响，随机抽取若干人后，在刺激前测量血压 SBPbefore, 施加刺激后再次测量血压 SBPafter, 这两次测量不是独立的，不能使用独立两样本 t 检验。
- 用 TTEST 过程的 PAIRED 语句检验。

成对均值检验例

- 试图比较施加刺激前后的血压有无显著差异。

```
title 'Paired Comparison';
data pressure;
  input SBPbefore SBPafter @@;
  datalines;
120 128   124 131   130 131   118 127
140 132   128 125   140 141   135 137
126 118   130 132   126 129   127 135
```

```
;
run;
proc ttest data=pressuer;
  paired SBPbefore*SBPafter;
run;
```

- 结果给出了前后血压差值的基本统计量、置信区间以及成对检验结果。
- 成对检验实际是关于两个变量差是否均值为零的检验。
- t 检验结果 p 值为 0.2992，没有显著差异。
- Analyst 中 “Statistics – Hypothesis Tests – Two-Sample Paired t-test for Means” 进行成对 t 检验。
- 再比如要比较 GPA 数据集中 SATM 分数和 SATV 分数有无显著差异。也是不独立的，可以使用成对 t 检验。

```
proc ttest data=sasuser.gpa;
  paired satm*satv;
run;
```

- SATM−SATV 均值为 90.737, 检验 p 值为 $< .0001$，在 0.05 水平下显著。
- 两科成绩在 0.05 水平下有显著差异。SATM 分数较高。

4.2 回归分析

回归分析

- 用 SAS/INSIGHT 进行曲线拟合;
- 用 SAS / INSIGHT 进行线性回归;
- 用 SAS/INSIGHT 作广义线性模型拟合;
- 用 PROC REG 作回归分析;
- 用 Analyst 作回归分析。

4.2.1 用 SAS/INSIGHT 进行曲线拟合

回归与曲线拟合

- 回归关系:

$$Y = f(X) + \varepsilon.$$

- 常见的 $f(x)$ 是 x 的线性函数。
- 更一般的回归关系:

$$Y|X \sim F(y, f(X)).$$

(广义 (线性) 回归).

- X 是一元自变量时，Y 对 X 的回归函数 $f(x)$ 构成一条曲线。
- 可以从数据中估计 $f(x)$。

INSIGHT 曲线拟合

- INSIGHT 的回归函数 $f(x)$ 可以用直线、多项式、样条函数、核估计和局部多项式来估计。
- 例：SASUSER.CLASS 中体重对身高的回归。
- 先看散点图。
- “Analyze – Fit (Y X)” 菜单拟合直线并打开回归窗口。
- 在回归窗口，选 “Curves – Polynomial” 进行多项式拟合。
- 注意：过度拟合 (over-fitting) 问题。统计建模要遵循模型尽可能简单的原则。过度拟合导致外推失效。

INSIGHT 曲线拟合 (续)

- "Curves – Spline" 进行样条平滑。可以自己选光滑系数。
- "Curves – Kernel" 作核回归:

$$\hat{f}(x) = \frac{\sum_{i=1}^{N} K\left(\frac{x-X_i}{\lambda}\right) Y_i}{\sum_{i=1}^{N} K\left(\frac{x-X_i}{\lambda}\right)}$$

  窗宽 λ 控制光滑程度。
- "Curves – Loess" 作局部多项式回归。
- "Curves – Local Polynomial, Fixed Bandwidth" 作固定带宽局部多项式回归。

4.2.2 用 SAS/INSIGHT 进行线性回归分析

线性回归分析

- 用菜单 "Analyze – Fit (Y X)" 就可以拟合一条回归直线。回归方程为

$$y = a + bx + \varepsilon$$

- 可以有多个自变量。
- 线性回归分析理论复习:

$$\mathbf{Y} = \mathbf{X}\beta + \varepsilon$$

- $\mathbf{Y}$ 为 $n \times 1$ 向量, $\mathbf{X}$ 为 $n \times p$ 矩阵, 一般第一列元素全是 1, 代表截距项。
- β 为 $p \times 1$ 未知参数向量;
- ε 为 $n \times 1$ 随机误差向量, ε 的元素独立且方差为相等的 σ^2(未知)。
- 系数的估计为

$$\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$$

  (假设系数矩阵满秩);
- 拟合值 (或称预报值) 为

$$\widehat{\mathbf{Y}} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y} = \mathbf{H}\mathbf{Y}$$

- 其中 $\mathbf{H}=\mathbf{X}\left(\mathbf{X}'\mathbf{X}\right)^{-1}\mathbf{X}'$ 是 R^n 空间的向量向 $\mathbf{X}$ 的列张成的线性空间 $\mu(\mathbf{X})$ 投影的投影算子矩阵, 叫做**帽子矩阵**。设

$$H=(h_{ij})_{n\times n}$$

- **拟合残差**为

$$\boldsymbol{e}=\mathbf{Y}-\widehat{\mathbf{Y}}=(\mathbf{I}-\mathbf{H})\mathbf{Y}$$

- **残差平方和**为

$$\mathrm{ESS}=\boldsymbol{e}'\boldsymbol{e}=\sum_{i=1}^{n}\left(Y_i-\hat{Y}_i\right)^2$$

- 误差项方差的估计 (要求设计阵 $\mathbf{X}$ 满秩) 为均方误差 (MSE)

$$s^2=\mathrm{MSE}=\frac{1}{n-p}\mathrm{ESS}$$

（其中 p 在有截距项时是自变量个数加 1）

- 在线性模型的假设下, 若设计阵 $\mathbf{X}$ 满秩, $\hat{\beta}$ 和 s^2 分别是 β 和 σ^2 的无偏估计。

- 系数估计的方差阵 $\mathrm{Var}(\hat{\beta})=\sigma^2\left(\mathbf{X}'\mathbf{X}\right)^{-1}$。

- 判断回归结果优劣的一个重要指标为**复相关系数平方** (决定系数)

$$R^2=1-\frac{\mathrm{ESS}}{\mathrm{TSS}}$$

(其中 $\mathrm{TSS}=\sum\limits_{i=1}^{n}\left(Y_i-\bar{Y}\right)^2$), R^2 代表在因变量的变差中用模型能够解释的部分的比例, 所以 R^2 越大说明模型越好。

- 但是，模型中每增加一个自变量，ESS 都会变小，R^2 都会变大，不能判断过度拟合。

- 修正的 R^2:

$$R^{*2}=\begin{cases}1-\frac{n-1}{n-p}(1-R^2) & \text{若模型含截距项}\\ 1-\frac{n}{n-p}(1-R^2) & \text{若模型不含截距项}\end{cases}$$

(p 在有截距项时为自变量个数加 1, 在无截距项时为自变量个数)

体重对身高的回归：结果说明

- 包括模型说明、模型方程、拟合概况、方差分析表、第三类检验、参数估计及检验、残差图。
- 方差分析表用来检验整个模型是否有意义，如果所有斜率项均为零则模型无意义。
- 第三类检验是分别对每个斜率项系数作的等于零的检验。
- 参数估计中给出了估计值、估计的标准误差、检验相应参数为零的统计量和 p 值。注意标准误差在统计输出中的重要作用。
- 用残差图进行回归诊断，考察模型对数据是否适合。可以增加残差的正态 QQ 图。
- INSIGHT 中为了保存窗口中的结果表格, 在进行分析之前选中菜单 “File – Save – Initial Tables”, 这样结果表格会复制一份到 Output 窗口。

回归诊断—三种残差

- **回归残差**

$$\begin{aligned} e_i =& y_i - \hat{y}_i, \quad i = 1, 2, \ldots, n \\ \mathrm{Var}(e_i) =& \sigma^2(1 - h_{ii}) \quad (h_{ii} \text{ 是 } H \text{ 的主对角线元素}) \end{aligned}$$

- 把 e_i 标准化：

$$r_i = \frac{e_i}{s\sqrt{1 - h_{ii}}}, \quad i = 1, 2, \ldots, n$$

  在 INSIGHT 中称为**标准化残差**，也可或**内部学生化残差**。渐近正态分布。在 INSIGHT 数据表中显示为 RS_+ 因变量名。

- 如果计算 y_i 的预测值时，删除第 i 个观测后建立回归模型得到 σ^2 的估计 $s^2_{(i)}$，则**外部学生化残差** (RStudent) 为

$$t_i = \frac{e_i}{s_{(i)}\sqrt{1 - h_{ii}}}$$

  近似服从 $t(n - p - 1)$ 分布（有截距项时 p 等于自变量个数加 1）。在 INSIGHT 称为**学生化残差**，数据表中显示为 RT_+ 因变量名。

- 其中 $s_{(i)}$ 有简单公式：

$$s^2_{(i)} = \frac{n - p - r_i^2}{n - p - 1} s^2$$

回归诊断—残差图

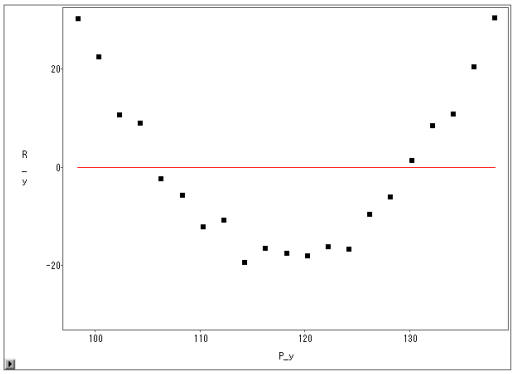
- 以残差 e_i、学生化残差（内 r_i 或外 t_i）为纵坐标画散点图可以识别模型的不足。
- (x_i, e_i): 因变量是否与模型自变量 X_i 有非线性关系？（可以增加 X_i 的高次项）是否明显丢失重要影响变量？
- (x_i, e_i): 误差 ε_i 方差是否随 X_i 而变化？
- (x_j, e_i): 模型外的变量 X_j 是否对模型有额外的贡献？
- $(\hat{y}_i, e_i)$: 误差方差大小是否与预测值大小有关？是否随机在纵坐标 0 的水平线上下变动？
- $(\hat{y}_i, r_i)$: 是否有明显拟合很差的观测？
- (i, e_i): 是否有序列相关？

残差图—需要非线性项

实际模型为

$$\begin{aligned} Y =& 100 + 2X + 0.5X^2 + \varepsilon \\ \varepsilon \sim& \mathrm{N}(0, 2^2) \end{aligned}$$

但只作了一元线性回归:

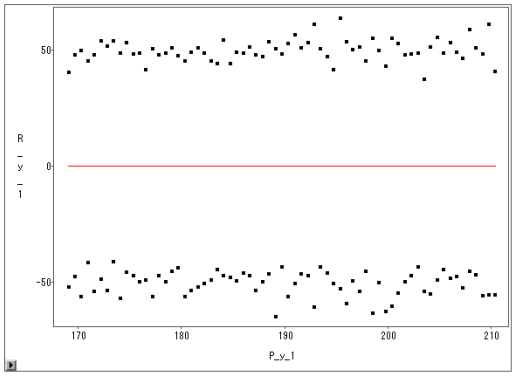


残差图—丢失重要影响因素

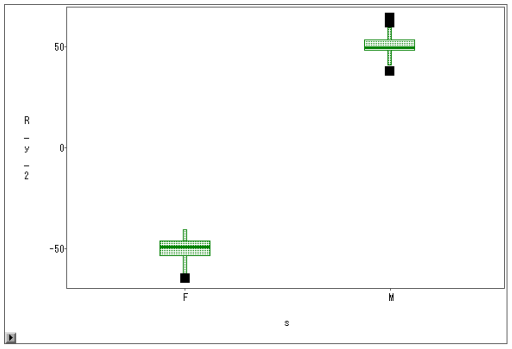
实际模型为

$$\begin{aligned} Y =& \begin{cases} 100 + 2X + \varepsilon, & s = \mathrm{F} \\ 200 + 2X + \varepsilon, & s = \mathrm{M} \end{cases} \\ \varepsilon \sim& \mathrm{N}(0, 5^2) \end{aligned}$$

但只作了 Y 对 X 的一元线性回归, 残差对自变量的散点图：

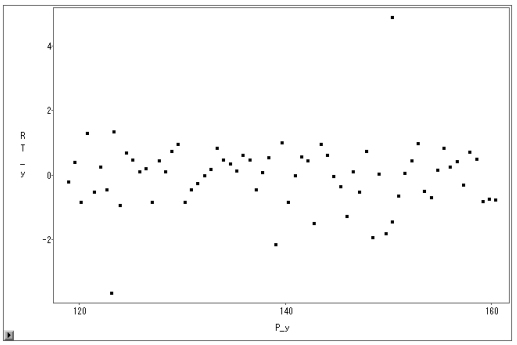


残差按外部分类变量 S 分组的并排盒形图：



残差图—离群值

$(\hat{y}_i, t_i)$ 的散点图：

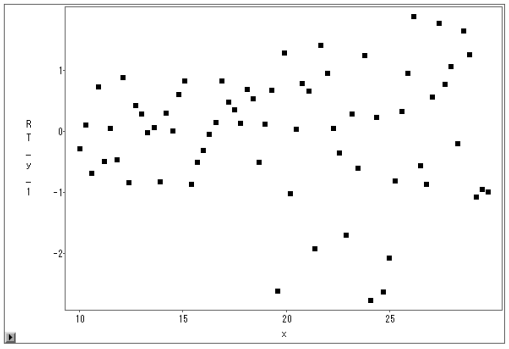


残差图—方差非齐性

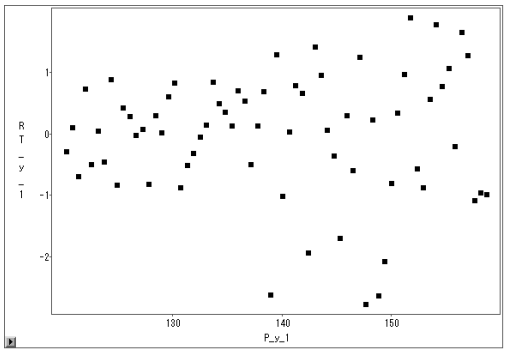
设真实模型为：

$$\begin{aligned} Y =& 100 + 2X + 0.25X \cdot \varepsilon \\ \varepsilon \sim& \mathrm{N}(0,1) \end{aligned}$$

(x_i, t_i) 的散点图：



$(\hat{y}_i, t_i)$ 的散点图：

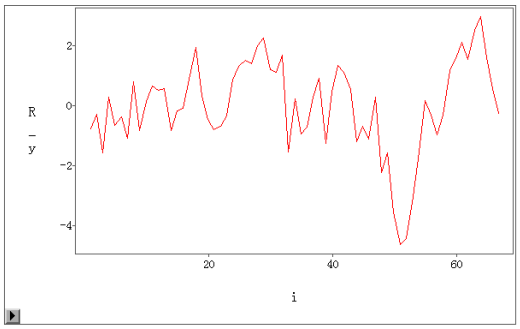


残差图—序列相关

设真实模型为：

$$\begin{aligned} Y_i =& 100 + 2X_i + \xi_i \\ \xi_i =& 0.8\xi_{i-1} + \varepsilon_i \\ \varepsilon_i \sim& \mathrm{N}(0,1) \end{aligned}$$

(i, e_i) 的散点图：



回归诊断—共线性

- 用 Tables 菜单可以加入共线 (Colinearity Diagnostics) 诊断统计量。
- 做共线诊断的条件数 (Conditional Index): 正数，衡量 $(X'X)^{-1}$ 的稳定性，定义为 $X'X$ 的最大特征值与最小特征值之比。
- 条件数在 0—100 之间时认为无共线性，在 100—1000 之间时认为自变量之间有中等或较强共线性，在 1000 以上认为自变量之间有强共线性。共线使得估计方差变大。
- 方差膨胀因子 (VIF) 衡量因共线问题引起的估计方差变大问题，VIF>10 可说明共线问题严重。
- 设模型中自变量集合为 $X_1, X_2, \ldots, X_p$，R_j^2 是自变量 X_j 对其它自变量回归得到的复相关系数平方，则

$$\mathrm{VIF}_j = \frac{1}{1 - R_j^2}, \quad j = 1, 2, \ldots, p$$

- 为了计算 VIF, 首先把矩阵 $X^T X$ 看成一个协方差阵，把它转换为相关系数阵设为 M，则 M^{-1} 的各主对角线元素就是各个 VIF。
- VIF 的倒数称为 tolerance。

回归诊断—影响分析

- 强影响点是删去以后严重改变参数估计值的观测。包括自变量取值离群和因变量拟合离群的点。
- 杠杆 (leverage) 指帽子矩阵的对角线元素 h_{ii},

$$\frac{1}{n} \le h_{ii} \le \frac{1}{d_i}$$

其中 d_i 是第 i 个观测的重复观测次数。在 tables 菜单中选择 Hat diag 输出观测的杠杆值到数据表。某观测杠杆值高说明该观测自变量有异常值。杠杆值大于 $2p/n$ 的观测需要仔细考察（有截距项时 p 等于自变量个数加 1）。

- Studentized residual 即外学生化残差 t_i，绝对值超过 2 的观测拟合误差大，在 y 方向离群，需要关注。
- Cook's D 统计量，Dffits，Covration 统计量考察删去第 i 个观测对整体估计的影响, Dfbetas 考察删去第 i 个观测对某个回归系数的影响。
- Cook's D 统计量：

$$D_i = \frac{1}{p} r_i^2 \frac{h_{ii}}{1 - h_{ii}}$$

包含了 y 方向的离群 r_i 和 x 方向的离群 h_{ii} 的信息。超过 $\frac{4}{n}$ 的值需要注意。

- 偏杠杆值衡量每个自变量 (包括截距项) 对杠杆的贡献。把第 j 个自变量关于其它自变量回归得到残差，第 i 个残差的平方占总残差平方和的比例为第 j 自变量在第 i 观测处的偏杠杆值。偏杠杆值影响自变量选择时对该变量的选择。

偏回归杠杆图

- 用 Graphs 菜单可以加入偏回归杠杆图 (Partial Leverage Plot), 也叫偏相关图。
- 把因变量 Y 与某自变量 X_j 都关于其它自变量作回归分别得到残差 $\tilde{Y}_i$ 与 $\tilde{X}_{ij}$（$i = 1, 2, \ldots, n$），作 $\{(\tilde{X}_{ij}, \tilde{Y}_i), i = 1, 2, \ldots, n\}$ 的散点图。
- 此散点图对应的一元线性回归的截距项恒为零，斜率等于完全模型中 X_j 对应的斜率 $\hat{\beta}_j$，残差等于完全模型的残差。
- 偏回归杠杆图呈现出的非随机模式说明 X_j 在模型中已经有其它自变量的条件下仍是对因变量解释有贡献的。

Vars 菜单增加的其它输出

- **Predicted** 为拟合值 (预报值)$\hat{y}_i$;
- **Linear Predictor** 为使用线性模型拟合的结果, 在线性回归时与 Predicted 相同;

- **Residual** 为残差 e_i;
- **Residual Normal Quantile** 是残差由小到大排序后对应的标准正态的分位数, 第 i 个残差的正态分位数用 $\Phi^{-1}\left(\frac{i-0.375}{n+0.25}\right)$ 计算, 其中 Φ 为标准正态分布函数, 参见 1.3.7 关于 QQ 图的解释。
- **Standardized Residual**(标准化残差) 即 r_i。

4.2.3 用 SAS/INSIGHT 拟合广义线性模型

广义线性模型

- 普通回归模型

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon, \varepsilon \sim \mathrm{N}(0, \sigma^2).$$

- 可改写成

$$\begin{cases} \mu = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p \\ Y|x_1, \ldots, x_p \sim \mathrm{N}(\mu, \sigma^2). \end{cases}$$

- 推广为

$$\begin{cases} g(\mu) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p \\ Y|x_1, \ldots, x_p \sim \mathrm{Dist}(\mu, \zeta) \\ E(Y|x_1, \ldots, x_p) = \mu \end{cases}$$

- g — 联系函数；Dist — 某分布；ζ 其它参数。

指数族分布

- 密度

$$f(y; \theta, \phi) = \exp\left(\frac{\theta y - b(\theta)}{a(\phi)} + c(y, \phi)\right)$$

- 包括正态、逆高斯、伽马、泊松、二项分布等分布。
- θ 为自然参数或称经典参数;
- ϕ 为分散度参数 (与尺度参数有关);
- a, b, c 为已知函数。

- 均值和方差:

$$\mathrm{E}(Y) = b'(\theta)$$
$$\mathrm{Var}(Y) = a(\phi)b''(\theta)$$

- **经典联系函数：**$b'(\theta)$ 的反函数 $g(\mu)$ 把 EY 变到经典参数 θ，叫做经典联系函数。

- 广义线性模型对 $g(\mu)$ 建立 $x_1, x_2, \ldots, x_p$ 的线性函数。$\mu = EY$。

常用联系函数

- 联系函数:
  - Identity: 恒等变换;
  - Log: 自然对数;
  - Logit: $g(\mu) = \log \frac{\mu}{1-\mu}, 0 < \mu < 1$;
  - Probit: $g(\mu) = \Phi^{-1}(\mu)$, 其中 Φ 为标准正态分布函数;
  - Comp. Log-Log: 重对数变换 $g(\mu) = \log(-\log(1-\mu)), 0 < \mu < 1$;
  - Power: $g(\mu) = \begin{cases} \mu^\lambda & \lambda \neq 0 \\ \log(\mu) & \lambda = 0 \end{cases}$, λ 在对话框的 Power 输入框指定。

- 经典 (canonical) 联系函数:
  - 正态分布：恒等变换；
  - 逆高斯分布：-2 次方变换；
  - 伽玛分布：-1 次方变换；
  - 泊松分布：对数变换；
  - 二项分布：逻辑变换 (Logit)。

用 INSIGHT 建立广义线性模型

- 在选 “Analyze - Fit (Y X)” 之后, 在出现的对话框中先选定因变量和自变量, 然后按 “Method” 按钮, 出现选择模型的对话框, 在这里可以选因变量的分布类型 (Response Dist.), 选联系函数, 选估计尺度参数的方法。

- 例：SASUSER.INGOTS 为 HEAT 加热时间和 SOAK 浸泡时间条件下，N 个铸件中不合要求的个数 R 的数据。二项分布。
- 取 R 为因变量，HEAT, SOAK 为自变量，因变量分布为二项分布。使用经典联系函数 (Logit)。
- 模型：

$$R \sim B(N, \mu)$$
$$\log \frac{\mu}{1-\mu} = \beta_0 + \beta_1 \cdot \text{HEAT} + \beta_2 \cdot \text{SOAK}$$

用 INSIGHT 建立广义线性模型 (续)

- 结果显著。
- 变量 SOAK 没有显著影响。
- 去掉 SOAK 后重新拟合。
- HEAT 的系数 0.0807 为正数，说明加热时间越长不合要求概率越大。
- 考察残差图，可去掉三个异常点。

4.2.4 用 REG 过程进行回归分析

SAS 中的回归分析过程

- REG 进行线性回归分析并进行回归诊断。
- ORTHREG 用 Gentleman-Givens 计算方法进行回归计算，适用于严重共线情形。
- PLS 进行偏最小二乘回归、主成分回归、削减秩回归。目的是同时解释因变量变差和自变量的变差。
- RSREG 建立二次多项式回归模型并可进行因变量最优化。
- NLIN 进行非线性回归建模。
- TRANSREG 各种复杂的回归模型，并可执行样条等非线性变换。
- GLM 一般的线性模型，包括多项式回归、方差分析和协方差分析。
- GENMOD 广义线性模型。

- LOGISTIC 对二值或多值因变量进行逻辑斯谛回归和回归诊断。
- PROBIT 进行 probit 回归、logistic 回归和有序 logistic 回归。适用于因变量是二分类或多分类而自变量是连续型的情况。
- CATMOD 对表达为列联表数据的情形建立线性回归模型、logistic 回归模型。
- LIFEREG 生存分析模型，数据可右删失。
- AUTOREG 建立误差项服从 AR 模型的回归模型。
- PDLREG 进行多项分布滞后回归分析。
- SYSLIN 计量经济模型中线性联立方程组建模。
- MODEL 计量经济学中进行非线性联立方程组建模。

用 REG 过程进行回归分析

- REG 过程的基本用法为：

```
PROC REG DATA= 输入数据集 选项;
        VAR 可参与建模的变量列表;
        MODEL 因变量＝自变量表 / 选项;
        PLOT 诊断图形;
RUN;
```

REG 例子

- 对 SASUSER.CLASS 中的 WEIGHT 用 HEIGHT 和 AGE 建模:

```
proc reg data=sasuser.class;
  var weight height age;
  model weight=height age;
run;
```

- 结果看:
  - Analysis of Variance 的结果决定模型是否有意义；
  - Parameter Estimates 看系数估计及检验结果。
- AGE 作用不显著可删去再建模。

回归自变量选择

- 自变量选择：在 MODEL 语句中加上 “SELECTION = 选择方法” 的选项。
- 方法包括：NONE(全用, 这是缺省), FORWARD (逐步引入法), BACKWARD(逐步剔除法), STEPWISE (逐步筛选法), MAXR(最大 R^2 增量法), MINR(最小 R^2 增量法), RSQUARE(R^2 选择法), ADJRSQ(修正 R^2 选择法), CP(Mallows 的 C_p 统计量法)。
- 如

```
model weight=height age
   / selection=stepwise;
run;
```

  作逐步回归，得到只有身高为自变量的模型。

- MODEL 语句选项 INCLUDE=n 可以指定前 n 个自变量保持在模型中不用进行选择。

回归诊断图形

- 新版本 SAS 的 REG 过程提供了自动绘制回归诊断图形的功能。
- 在 REG 过程之前使用 ODS GRAPHICS ON 语句，在之后使用 ODS GRAPHICS OFF 语句，可以自动绘制回归诊断图。如

```
ods graphics on;
proc reg data=samp.class;
   model weight=height;
quit;
ods graphics on;
```

回归诊断图形解释

- 残差 (e_i) 对预测值 ($\hat{y}_i$);
- 外学生化残差 (t_i) 对预测值 ($\hat{y}_i$);
- 外学生化残差 (t_i) 对杠杆 (h_{ii});
- 残差 QQ 图;
- 因变量 (y_i) 对预测值 ($\hat{y}_i$);
- Cook 的 D 统计量 (D_i) 对 i;
- 残差的直方图;
- 中心化的拟合值的正态概率图与残差的正态概率图的对比，如果残差的纵向散布比中心化拟合值的纵向散布宽提示模型拟合不良；
- 一元回归时残差对自变量图；
- 一元回归时因变量对自变量图，其中叠加了回归直线、均值置信界限、预测界限。对于多项式回归，可以在 PROC REG 语句加入选项 PLOTS=PREDICTIONX(X= 自变量名) 来获得此图。

预测和残差

- MODEL 语句选项 P 显示预测值和残差。
- MODEL 语句选项 R 显示预测值的标准误差、残差、残差的标准误差、标准化残差、Cook's D 统计量。
- MODEL 语句选项 CLM 显示均值置信区间，CLI 显示预测区间，ALPHA=α 指定置信区间的置信度为 $1-\alpha$。

残差图

- (x_i, e_i) 散点图：

```
plot r. * height;
```

- $(\hat{y}_i, e_i)$ 散点图：

```
plot r. * p.;
```

- $(\hat{y}_i, t_i)$ 散点图：

```
plot rstudent. * p.;
```

- (i, t_i) 散点图：

```
plot rstudent. * obs.;
```

- r_i 的正态 QQ 图：

```
plot student. * nqq.;
```

影响分析和共线诊断

- MODEL 语句选项 INFLUENCE 与选项 P 配合，可以显示每个观测的杠杆 h_{ii}，（外部）学生化残差 t_i、Cov Ratio、DFFITS、DFBETAS。
- PROC REG 语句选项 COLLIN、COLLINOINT、TOL 计算共线诊断统计量，MODEL 语句选项 VIF 显示每个变量的方差膨胀因子。

偏回归

- MODEL 语句选项 PARTIAL 作偏回归杠杆图。
- 偏回归杠杆图呈现出的非随机模式说明 X_j 在模型中已经有其它自变量的条件下仍是对因变量解释有贡献的。

系数的联合检验

- 用 TEST 语句检验关于系数的线性组合的假设。
- 例如，下例在 FITNESS 数据集中对 oxygen 关于 age, weight, runtime, rstpulse, runpulse, maxpulse 进行回归。

```
proc reg data=sasuser.fitness;
  model oxygen = age weight runtime
      rstpulse runpulse maxpulse;
  test1: test rstpulse=runpulse,
              rstpulse=maxpulse;
  test2: test weight=0, rstpulse=0;
quit;
```

- 为了检验 rstpulse, runpulse, maxpulse 的斜率项是否相同，用第一个 test 语句。
- 为了检验 weight 和 rstpulse 的斜率项是否都为零，用第二个 test 语句。

多项式回归

- 考虑美国人口随年份的变化。数据为美国从 1790 到 2000 每十年的人口数。
- 直接线性拟合发现残差呈现非线性。
- 增加平方项后，残差较正常。
- 数据:

```
data USPopulation;
  input Population @@;
  retain Year 1780;
  Year=Year+10;
  YearSq=Year*Year;
  Population=Population/1000;
  datalines;
3929 5308 7239 9638 12866 17069
23191 31443 39818 50155
62947 75994 91972 105710 122775
131669 151325 179323 203211
226542 248710 281422
```

```
;
run;
```

- 程序:

```
symbol1 c=blue;
proc reg data=USPopulation;
  var YearSq;
  model Population=Year / r cli clm;
  plot r.*p. / cframe=ligr;
run;
  add YearSq;
  print;
  plot / cframe=ligr;
run;
symbol1 v=dot     c=yellow h=.3;
  symbol2 v=square  c=red;
  symbol3 f=simplex c=blue  h=2 v='-';
  symbol4 f=simplex c=blue  h=2 v='-';
  plot (Population predicted. u95. l95.)*Year
       / overlay cframe=ligr;
run;
quit;
```

多项式回归与共线性

- 加入高次项或交叉项容易引起共线性，如上例中 YearSq 的 VIF 达到了 4490。
- 把自变量中心化可以减轻共线性问题：

```
data new;
  set USPopulation;
  Year = Year - 1880;
  YearSq = Year*Year;
```

```
run;
proc reg data=new;
  model Population = Year YearSq /
     r clm cli vif;
quit;
```

4.2.5 用 Analyst 进行回归分析

用 Analyst 进行回归分析

- Analyst 的 “Statistics - Regression” 菜单提供了三种回归：
  - 一元回归: 一元的线性或多项式回归；
  - 线性回归: 多个自变量的线性回归；
  - Logistic 回归。
- 以 SASUSER.CLASS 数据集为例。选 “Statistics - Regression - Linear”, 先指定因变量和自变量；
- Model 按钮可以指定模型选择方法及具体选择方法的细节；
- Statisics 按钮可以要求输出与模型拟合优度和模型诊断有关的统计量；
- Predictions 可以要求计算对数据集中各观测的预测值、残差值、预测界限；也可以指定一个包含模型自变量的数据集要求对其进行预测；
- Plots 可以要求画各种回归诊断图形，如残差图、杠杆图等；
- Save Data 可以把指定的结果保存到数据集中。

4.3 方差分析入门

方差分析

- 方差分析研究多组比较的问题。
- 最简单的情况是一种分组方式，比较各组之间指标值的差异，是两样本 t 检验的推广。
- 可以有多种分组方式。
- 要研究的指标作为因变量。
- 分组方式由分类变量表示，一个分组用的变量叫做一个**因素**，因素的不同取值（分组的代号）叫做因素的**水平**。
- 方差分析的问题:
  - 因素对指标有无显著影响；
  - 如果因素有显著影响，最好的水平是什么。

4.3.1 用 ANOVA 过程进行单因素方差分析

用 ANOVA 过程进行单因素方差分析

- 单因素方差分析是独立两样本比较问题的一个自然延续。
- 要比较多个组的均值是否有显著差异。
- 如果各组之间有显著差异, 说明这个因素 (分类变量) 对指标是有显著影响的。
- 方差分析把指标的方差分解为:
  - 由因素的不同取值能够解释的部分;
  - 剩余的不能解释的部分;

  然后比较两部分, 当能用因素解释的部分明显大于剩余的部分时认为因素是显著的 (因素对指标有显著影响)。

单因素方差分析假定

- 观测是**彼此独立**的;
- 观测为**正态分布**的样本;
- 由因素各水平分成的各组的**方差相等**。

单因素方差分析的程序

PROC ANOVA DATA =数据集;
CLASS 因素;
MODEL 指标=因素;
RUN;

ANOVA 例—VENEER 数据

- 例: VENEER 数据。
  - 变量 WEAR：指标；
  - 变量 BRAND: 因素，有 5 个水平 (牌子)，每个水平试验 4 次。
- 考察不同牌子的磨损率是否有差别，或：因素“牌子”对指标“磨损量”有无显著影响。
- 方差分析不考虑两两之间的差别而只考虑各组是否完全没有显著差异。
- 程序:

```
proc anova data=sasuser.veneer;
  class brand;
  model wear=brand;
run;
```

方差分析的结果

- 因素水平信息；
- 总方差分析表；
- 模型诊断信息；
- 各效应的检验。

4.3.2 用 NPAR1WAY 进行非参数单因素方差分析

用 NPAR1WAY 进行非参数单因素方差分析

- 当方差分析的正态分布假定或方差相等假定不能满足时, 对单因素问题, 可以使用称为 Kruskal-Wallis 检验的非参数方差分析方法。
- 不要求观测来自正态分布总体, 不要求各组的方差相等, 甚至指标可以是**有序变量**。
- 程序：

```
PROC NPAR1WAY DATA= 数据集  WILCOXON;
    CLASS 因素;
    VAR 指标;
RUN;
```

NPAR1WAY 例

- 对 VENEER 数据：

```
proc npar1way data=sasuser.veneer wilcoxon;
  class brand;
  var wear;
run;
```

- 结果:
  - 各组秩和；
  - Kruskal-Wallis 检验的结果。
- 非参数检验一般比参数检验的功效低。

4.3.3 多重比较

多重比较

- 方差分析只检验各组是否没有任何两两之间的差异, 但不检验到底是哪两组之间有显著差异。
- 在三个或多个组之间进行两个或多个比较的检验叫做多重比较。

- 没有公认的解决方法。
- 两种检验水平：
  - 总错误率 (experimentwise error rate, 或 Family-wise Error Rate, FWER);
  - 单次比较错误率 (comparisonwise error rate, CER)。
- 总错误率比单次比较错误率高。
- 在 ANOVA 过程中使用 MEANS 语句可以进行多重比较。格式如下：

  MEANS 因素 / 选项;

用重复 t 检验控制单次比较错误率

- 只控制单次比较错误率而不控制总错误率。
- 在 MEANS 语句中用 T 选项。可以用 ALPHA= 指定单次比较检验水平。
- 如:

```
means brand / t;
```

- 结果中标有同一字母的水平之间没有显著差异。
- 重复 t 检验的结果把五种牌子分成了 A、B、C 三个组, TUFFY 单独是一组, 它的磨损量最大；XTRA、CHAMP、ACME 是一组, 这三种两两之间没有显著差异；AJAX 单独是一组, 其磨损量最小。

用 Bonferroni t 检验控制总错误率

- 如果控制总错误率在 α，需要进行 m 次两两比较，则控制每一次两两比较的水平为 α/m。
- 比较保守。
- 程序例子:

```
means brand / bon;
```

- 最后只发现了 TUFFY 和 AJAX 之间有显著差异。

用 REGWQ 检验控制总错误率

- REGWQ 方法可以控制总错误率并且一般比 Bonferroni t 检验功效高。
- 在 MEANS 语句中用 REGWQ 选项可以进行 REGWQ 检验。
- 如:

```
means brand/ regwq;
```

- 结果 TUFFY、XTRA、CHAMP、ACME 作为一组，AJAX 作为一组。

控制错误发现率 (FDR)

- 前述方法控制单次比较的第一类错误概率或者控制所有比较至少有一次错误的概率。
- FDR 方法控制所有比较中显著结果的假显著比例。
- 设对同一组数据一共进行了 m 个假设检验，其中 ξ 个得到显著结果，但是这 ξ 个显著结果中，有 η 个实际是 H_0 成立的，另外 $\xi - \eta$ 个才是 H_0 不成立的，定义 $\frac{\eta}{\xi}$ 的期望值为 FDR(False discovery rate)。
- 控制 FDR，比控制单次比较错误率要严得多（能得到的显著结果更少），比控制总错误率则要松一些（能得到的显著结果多一些）。
- SAS 的 PROC MULTTEST 可以执行多重比较，其中提供了用 FDR 方法计算调整 p 值的功能。

4.3.4 多因素方差分析

多因素方差分析

- SAS 提供了若干个方差分析过程, 可以考虑多个因素、有交互作用、有嵌套等情况的方差分析。用 GLM 过程还可以用一般线性模型来处理方差分析问题。
- 在这里我们只介绍如何用 ANOVA 过程进行均衡设计的多因素方差分析。
- 例：橡胶定强试验。
  - 指标：定强。

  - 因素：促进剂 (因素 A, 三个水平); 氧化锌量 (因素 B, 四个水平)。
  - 完全试验，每种组合试验两次。

橡胶定强例子的数据输入

- 可以直接输入 24 行数据。
- 按表格输入更便于查对：

```
data rubber;
  do  a=1 to 3;
    do  b=1 to 4;
      do  r=1 to 2;
        input stren @@;
        output;
      end;
    end;
  end;
  cards;
31 33  34 36  35 36  39 38
33 34  36 37  37 39  38 41
35 37  37 38  39 40  42 44
;
run;
```

橡胶定强例子的方差分析

- 为了研究两个因素的主效应和交互作用, 使用如下 ANOVA 过程：

```
proc anova data=rubber;
  class a b;
  model  stren = a b a*b;
run;
```

- 结果:
  - 因素情况；

– 总方差分析表；

– 模型诊断信息；

– 各效应的检验。

橡胶定强例子的方差分析 (续)

- 交互作用效应不显著，可以拟合只有主效应的模型:

```
model stren = a b;
```

- A 和 B 的 F 统计量都变大了。

- 用 MEANS 找到不同水平下的平均指标值:

```
means a b / regwq;
```

正交设计的方差分析例

- 正交设计不是完全设计，但可以用 ANOVA 分析。

- 例子有 5 个因素，每个因素两个水平，用 $L_8(2^7)$ 表安排试验。

- 数据：

```
data  exp;
  input temp time conc manu mix  prod;
  cards;
1 1 1 1 1  65
1 1 1 2 2  74
1 2 2 1 2  71
1 2 2 2 1  73
2 1 2 1 2  70
2 1 2 2 1  73
2 2 1 1 1  62
2 2 1 2 2  69
;
run;
```

正交设计的方差分析例 (续)

- 正交设计的数据进行方差分析：

```
proc anova data=exp;
  class temp time conc manu mix;
  model prod = temp--mix;
  means temp--mix / t;
run;
```

- 0.05 水平下模型显著。
- 硫酸浓度、产地、操作方式是显著的, 必须采用它们的最好水平；
- 温度、时间不显著, 但试验次数很少所以不一定真的没有影响，仍应采取最好水平。

4.3.5 用 Analyst 作方差分析

用 Analyst 作方差分析

- Analyst 的 “Statistics – ANOVA” 菜单提供了七种方差分析方法。
- 例：VENEER 数据，用 “Statistics – ANOVA – One-Way ANOVA”(单因素方差分析)。
- 用 “Statistics – ANOVA – Nonparametric One-Way ANOVA” 可以作 Kruskal-Wallis 检验。
- 用 “Statistics – ANOVA – Factorial ANOVA” 可以进行多元方差分析。它调用 PROC GLM。

4.4 属性数据分析

属性数据分析

- 统计模型:
  - 回归分析：因变量为区间变量，自变量为区间变量；
  - 广义线性模型：因变量为区间变量或离散变量，自变量为区间变量；
  - 方差分析：因变量为区间变量，自变量为名义变量；
  - 属性数据分析：因变量和自变量都为名义变量；
- 在回归分析和广义线性模型中，可以通过哑变量编码把名义型自变量变成数值型。
- 本节介绍：
  - 离散变量分布的拟合优度检验；
  - 两个离散变量独立性的列联表检验；
  - 有序变量的关联性量度。

4.4.1 拟合优度的卡方检验

单个比例的检验

- 关于单样本的比例的假设检验和置信区间，可以用 PROC FREQ。比如，检验 SASUSER.CLASS 数据集中男生比例是否等于 0.5, 可用程序

```
proc freq data=sasuser.class;
  tables sex / binomial(p=0.5 level=2);
  exact binomial;
run;
```

- 在 TABLES 语句中用 BINOMIAL 选项指定比例的假设检验和置信区间，用 ALPHA= 选项指定置信度。用 EXACT 语句加 BINOMIAL 要求计算精确检验和精确置信区间。

例子

- 实际中还需要检验离散变量的取值规律 (分布) 是否某种预想的情况。
- 例如：某工厂发生事故在星期几是否概率都是 1/6 的检验。
星期	一	二	三	四	五	六
次数	9	10	11	8	13	12
- X — 事故发生的星期号码 (1–6)。
- 检验:
$$H_0 : \Pr(X = i) = \frac{1}{6} \quad (i = 1, 2, \ldots, 6)$$
- 使用拟合优度的卡方检验：
$$\chi^2 = \sum_{i=1}^{6} \frac{(n_i - np_i)^2}{np_i}$$
其中 $p_i = 1/6$ 是零假设下的概率。在 H_0 下 χ^2 近似 $\chi^2(5)$ 分布。当 χ^2 统计量超过临界值时否定 H_0。

例子 (续)

- 在 SAS 中用 PROC FREQ 做卡方检验:

```
data accident;
  input day times @@;
  cards;
1 9  2 10  3 11  4 8  5 13  6 12
;
run;
proc freq data=accident;
  tables day / chisq
    testp=(0.1666667 0.1666667 0.1666667
           0.1666667 0.1666667 0.1666667);
  weight times;
run;
```

例子 (续 2)

- PROC FREQ 中用 WEIGHT 语句给定表示重复数的变量, 作用和其它过程中的 FREQ 语句作用相同。
- 用 TABLES 语句的 CHISQ 和 TESTP 选项进行单变量关于已知分布的拟合优度检验。
- 程序的结果 p 值为 0.8931，不显著，不能认为事故发生与星期几有关。

4.4.2 列联表的输入与制表

列联表

- 列联表：按两个或多个分类变量把观测分类得到每个交叉分类的频数的表格。
- 例如: 学生性别、来源分布表

	男生	女生
本地	4	6
外地	14	7

- 输入方法: 可以输入原始数据，每个观测代表一个人；也可以输入表格数据，表的每个格子作为一个观测，观测中包含分类变量的值和频数。

列联表输入和制表例

- 学生性别和来源的列联表输入和制表程序:

```
data classt;
  input from $ sex $ numcell;
  label sex=' 性别' from=' 来源';
  cards;
本地 男 4
本地 女 6
外地 男 14
外地 女 7
;
run;
proc freq data=classt;
  tables from * sex;
  weight numcell;
run;
```

- 用 WEIGHT 语句指定代表观测频数的变量。

列联表结果

- TABLES 语句中用星号分开行变量和列变量。
- 每个交叉分类形成的格子中有四个数:
  1. Frequency—频数；
  2. Percent—百分比；
  3. Row Pct—行百分比；
  4. Col Pct—列百分比。
- 每行的末尾有该行的总数和百分比；每列的末尾有该列的总数和百分比。
- 在 TABLES 语句中指定 NOFREQ, NOPERCENT, NOROW, NOCOL 等选项可以抑制相应的统计量的输出。

4.4.3 列联表独立性检验

列联表独立性检验

- 列联表独立性检验可以检验两个分类变量是否独立。
- H_0 : 变量 X 与 Y 独立。
- 检验的方法是计算一个近似 χ^2 分布的统计量，基于列联表中观测频数与零假设下期望频数之差, 当 χ^2 统计量很大时否定零假设。
- 例如，吸烟与慢性支气管炎的调查数据:

	患慢性支气管炎	未患慢性支气管炎
吸烟	43	162
不吸烟	13	121

- X: 是否吸烟；Y：是否患有慢性支气管炎。
- H_0: X 与 Y 独立。

独立性检验的程序

- 吸烟与慢性支气管炎独立性检验和数据输入的程序如下:

```
data bron;
  input smoke $ bron $ numcell;
  label smoke=' 吸烟'  bron=' 慢性支气管炎';
  cards;
吸烟  患病  43
吸烟  未患  162
不吸烟  患病  13
不吸烟  未患  121
;
proc freq data=bron;
  tables smoke*bron / nopercent norow
         nocol  chisq expected;
weight numcell;
run;
```

独立性检验的程序 (续)

- TABLES 语句中的 CHISQ 选项要求进行列联表独立性 χ^2 检验，expected 选项要求显示每个单元格在 H_0 下的期望频数。
- 输出结果中的 “Chi-Square” 一行为检验的 p 值所在。
- p 值为 0.0063，在 0.05 水平下否认零假设，作出结论: 吸烟与患慢性支气管炎是不独立的。从期望值中也可以看出偏差。
- 使用 χ^2 检验要求每个单元格期望频数不少于 5。在条件不满足的时候还可以使用 Fisher 精确检验。
- 卡方检验也可以看成是对两个比例相等的双侧假设检验。Fisher 精确检验还给出了单侧结果，但 Fisher 精确检验要求假定所有边缘频数（列联表行和与列和）都固定。

4.4.4 属性变量关联度计算

属性变量关联度计算

- 两个有序变量我们可以计算类似于相关系数的关联性量度。
- 其中一种关联性量度叫做 Kendal Tau-b(τ_b) 统计量, 取值在-1 到 1 之间, 值接近于 1 表示正关联, 接近于-1 表示负关联, 接近于 0 表示没有相关关系。
- 例: 奶牛种群大小 (1,2,3) 和患病情况 (0,1,2) 的关联度。

例子—数据

```
data cows;
  input herdsize  disease  numcell;
  label herdsize=' 牛群大小'
        disease=' 患病程度';
  cards;
1 0 9
1 1 5
1 2 9
2 0 18
2 1 4
2 2 19
3 0 11
3 1 88
3 2 136
;
run;
```

例子

```
proc freq data=cows;
  tables herdsize*disease / measures
      expected nopercent norow nocol;
  weight numcell;
```

```
  title ' 奶牛疾病数据分析';
run;
```

- Kendall Tau-b 统计量值为 0.2173;
- 渐近标准误差 (ASE) 为 0.0606;
- 用统计量值加减两倍标准误差作为 Kendall Tau-b 的 95% 置信区间, 可算得 (0.0961, 0.3385) 在零点右边, 所以可认为奶牛患病程度与种群大小有正的关联。

第五章　SAS 多元统计分析

SAS 多元分析

- SAS/STAT 中提供了多元分析的程序，常用的有:
  - 主分量分析 (proc princomp)、因子分析 (proc factor)；
  - 判别分析 (proc discrim);
  - 聚类分析 (proc cluster)。

5.1 多变量分析

5.1.1 主分量分析

主分量分析

- 一个对象有太多的属性时，不易从这些属性看出对象之间的差异。
- 主分量分析、因子分析可以从多个属性压缩维数得到较少的能反映个体差异的分量。
- 设 $\mathbf{X} = (X_1, X_2, \ldots, X_p)^T$ 是 p 维随机向量，试图用 $\mathbf{X}$ 的线性组合保存不同对象的 $\mathbf{X}$ 的差异信息。
- 令 $Y_1 = \mathbf{l}_1^T\mathbf{X}$ 为 $\mathbf{X}$ 的一个线性组合，$\mathbf{l}_1^T\mathbf{l}_1 = 1$, 求 $\mathbf{l}_1$ 使 $\mathrm{Var}(Y_1)$ 最大。Y_1 称为 $\mathbf{X}$ 的第一主分量 (principle component)。
- 取 $Y_2 = \mathbf{l}_2^T\mathbf{X}$, 其中 $\mathbf{l}_2^T\mathbf{l}_2 = 1$, 且 $\mathrm{Cov}(Y_1, Y_2) = 0$，取 $\mathbf{l}_2$ 使 $\mathrm{Var}(Y_2)$ 最大。Y_2 称为 $\mathbf{X}$ 的第二主分量。
- $\cdots\cdots$.
- 设 $\mu = \mathrm{E}(\mathbf{X})$, $\Sigma = \mathrm{Var}(\mathbf{X})$。
- $\mathrm{Var}(Y_1) = \mathbf{l}_1^T\Sigma\mathbf{l}_1$;
- $\mathrm{Var}(Y_2) = \mathbf{l}_2^T\Sigma\mathbf{l}_2$;
- $\mathrm{Cov}(Y_1, Y2) = \mathbf{l}_1^T\Sigma\mathbf{l}_2$;
- $\cdots\cdots$.
- 设半正定矩阵 Σ 的特征值为 $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$, 对应的单位特征向量为 $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_p$。
- 则 $Y_i = \mathbf{a}_i^T\mathbf{X}$。

- 记正交矩阵 $A = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_p)$, 记 $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_p)$, $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_p)^T = A^T\mathbf{X}$。
- $\mathrm{Var}(\mathbf{Y}) = \Lambda$, $\mathbf{Y}$ 的各分量为 $\mathbf{X}$ 的各主分量。
- 设 $\Sigma = (\sigma_{ij})_{p\times p}$。
- $\sum_1^p \lambda_i = \sum_1^p \sigma_{ii}$。
- **因子负荷量** (factor loading): $\rho(Y_k, X_i)$。
- 主分量 Y_k 的**贡献率**: $\lambda_k / \sum_1^p \lambda_i$。
- 前 m 个主分量的**累计贡献率**: $\sum_1^m \lambda_i / \sum_1^p \lambda_i$。
- 一般取 m 使累计贡献率达到 $70\% \sim 80\%$ 以上。
- 用 $\mathbf{X}$ 的协方差阵 Σ 的特征值分解计算主分量，使得方差大的分量起的作用大；不同的量纲会对主分量选取起作用。
- 用相关系数阵 R 的特征值分解来计算主分量可以避免这个问题。

- 设有 $\mathbf{X}$ 的 n 组样本 $x_{(t)} = (x_{t1}, x_{t2}, \ldots, x_{tp})$, $t = 1, 2, \ldots, n$。
- 如果要从协方差阵计算主分量分解，把 $\mathbf{X}$ 各列中心化 (减去该列的样本平均值) 得到 $\mathbf{X}$，用 $\hat{\Sigma} \triangleq \mathbf{X}'\mathbf{X}$ 来估计协方差阵 Σ。
- 如果要从相关系数阵计算主分量分解，把 $\mathbf{X}$ 各列标准化 (减去该列的样本平均值并除以该列样本标准差) 得到 $\mathbf{X}$，用 $\hat{R} \triangleq \mathbf{X}'\mathbf{X}$ 来估计相关系数阵 R。
- 从 $\hat{\Sigma}$ 或 $\hat{R}$ 得到主分量分解。
- 设特征值为 $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$, 对应特征向量组成正交阵 $A = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_p)$。
- $Y_{(t)} = \tilde{x}_{(t)}A = \left(\tilde{x}_{(t)}\mathbf{a}_1, \tilde{x}_{(t)}\mathbf{a}_2, \ldots, \tilde{x}_{(t)}\mathbf{a}_p\right)$ 称为第 t 个观测的主分量得分，它包含 p 个主分量在第 t 个观测上的值。
- 可以只保留前 m 个主分量。

PROC PRINCOMP 的功能

PROC PRINCOMP 可以:

- 完成主分量分析。
- 主分量的个数、名字和得分是否标准化可以由用户确定。
- 可以选择是从协方差阵计算还是从相关阵计算。
- 计算结果包括：简单统计量；相关阵或协方差阵；从大到小排列的特征值和相应特征向量；贡献率、累计贡献率；包含原始数据和得分的数据集。
- 如果特征值很接近零则说明原来 **X** 的各分量之间接近线性相关。

PROC PRINCOMP 语法

用 PROC PRINCOMP 语句和 VAR 语句。VAR 语句指定对那些分量计算主分量分析。

PROC PRINCOMP 语句的选项:

- data= 输入数据集。
- out= 包含原始数据和得分的输出数据集。
- outstat= 统计量输出数据集。
- cov 要求从协方差阵出发计算主分量。缺省为从相关阵出发计算。
- n= 要计算的主分量个数。缺省时全算。
- noint 要求在模型中不使用截距项。
- std 要求在 OUT= 的数据集中把主分量得分标准化为单位方差。不规定时方差为相应特征值。

PROC PRINCOMP 例子—平均气温

- 例 1. 一月和七月平均气温。
- 使用协方差阵，因为温度是可比的。
- 一个主分量解释了 94% 的方差。
- 绘制主分量散点图。主分量是原始变量的正交旋转。
- 第一主分量可以解释为加权平均气温；第二主分量可以解释为冬季与夏季的差别。

PROC PRINCOMP 例子—各种犯罪

- 例 2. 七种犯罪率指标，希望得到概括的指标。
- 使用相关阵。
- 第一主分量贡献率 59%; 前二主分量累计贡献率 76%; 前三主分量累计贡献率 97%。
- 第一主分量是加权平均。
- 第二主分量代表谋财犯罪与暴力犯罪的对比。

用 SAS/INSIGHT 和 Analyst 作主分量分析

- 在 SAS/INSIGHT 中选 “Annalyze – Multivariate(Y's)”。
- 在 Analyst 中选 “Statistics – Multivariate – Principle Components”。

5.1.2 因子分析

因子分析

- 因子分析的目的与主分量分析类似，但不要求最后降维得到的“因子”是原始分量的线性组合。
- 模型为
$$\boldsymbol{X} = \boldsymbol{\mu} + \Lambda \boldsymbol{F} + \boldsymbol{U}$$
- 其中 $\Lambda_{p\times k}$ 是未知常数阵，称为因子载荷矩阵；$F_{k\times 1}$ 是未知的随机向量，称为公共因子；$U_{p\times 1}$ 也是未知的随机向量，称为特殊因子。
- 我们关心的是公共因子。
- 分解可以通过 $\mathrm{Var}(\mathbf{X}) = \Sigma$ 的分解式得到:
$$\Sigma = \Lambda\Lambda^T + \Psi$$
- 分解不唯一: 对任正交阵 Γ,
$$\boldsymbol{X} = \boldsymbol{\mu} + (\Lambda\Gamma)(\Gamma^T\boldsymbol{F}) + \boldsymbol{U}$$
- 可以用正交阵 Γ 对因子进行旋转，使因子载荷阵元素尽可能都接近 1 或 0。

PROC FACTOR

基本的模式为:

```
PROC FACTOR DATA= 数据集 选项;
    VAR 原始变量;
RUN;
```

输出结果包括:

- 特征值情况;
- 因子载荷;
- 公因子解释比例;
- 在 proc factor 语句加入 score 选项和 outstat= 选项可以输出计算公因子得分所需的统计量，然后用如下程序计算公因子得分:

```
PROC SCORE DATA= 原始数据集
            SCORE=FACTOR 过程的输出数据集
            OUT= 得分输出数据集;
      VAR 用来计算得分的原始变量集合;
RUN;
```

例子—警察评价指标因子分析

- 103 位警察，14 个评价指标。希望找到评价指标的内在因素。
- 不指定 corr 选项，使用协方差阵。
- 没有方法选项时做主分量分析，但输出结果按因子分析输出。
- 加 priors=smc 指定估计先验 community 估计的方法用复相关系数平方。未指定方法，做主因子分析，计算 $\Sigma - \Psi$ 的估计的特征值。
- rotate=varimax 是做正交的旋转。
- 结果选定 3 个因子。未旋转前，因子载荷 (Factor patterns) 阵代表用因子估计各标准化原始变量的回归系数，这时也等于原始变量和公因子的相关系数。
- 用 proc factor 的 score 和 outstat= 选项与 proc score 配合得到因子得分输出数据。

- 因子 1 的值为相近的正数，说明第一个因子是能力综合（平均）。
- 因子 2 的值反映了执行任务能力和人际关系能力的对比。
- 因子 3 难以解释。
- 经过 varimax 正交旋转后，以因子载荷阵超过 0.5 为限，因子 1 代表人际关系能力，因子 2 代表体能和工作热情，因子 3 代表判断力。

例子-五个社会经济指标的因子分析

- 洛杉矶 12 个地区统计的五个社会经济指标。
- 用 corr 选项指定用相关阵。
- 没有方法选项时做主分量分析，但输出结果按因子分析输出。
- 加 priors=smc 做主因子分析，计算 $\Sigma - \Psi$ 的估计的特征值。
- 加上 rotate=promax 和 rotate 可以进行旋转以得到容易解释的因子。
- 用 proc factor 的 score 和 outstat= 选项与 proc score 配合得到因子得分输出数据。

5.2 判别分析

5.2.1 统计背景

判别分析理论

- **判别分析**的目的是对已知分类的数据建立由数值指标构成的分类规则, 然后把这样的规则应用到未知分类的样本去分类。
- 另外的名称叫做**有监督学习**。
- 例如, 我们有了患胃炎的病人和健康人的一些化验指标, 就可以从这些化验指标发现两类人的区别, 把这种区别表示为一个判别公式, 然后对怀疑患胃炎的人就可以根据其化验指标用判别公式诊断。
- **参数方法**: 假定每个类的观测来自 (多元) 正态分布总体, 各类的分布的均值 (中心) 可以不同。
- **非参数方法**不要求知道各类所来自总体的分布, 它对每一类使用非参数方法估计该类的分布密度, 然后据此建立判别规则。
- 记 $\boldsymbol{X}$ 为用来建立判别规则的 p 维随机变量, S 为合并协方差阵估计, $t = 1, \ldots, G$ 为组的下标, 共有 G 个组。
- 记 n_t 为第 t 组中训练样本的个数, $\boldsymbol{m}_t$ 为第 t 组的自变量均值向量, S_t 为第 t 组的协方差阵, $|S_t|$ 为 S_t 的行列式, q_t 为第 t 组出现的先验概率;
- $p(t|\boldsymbol{x})$ 为自变量等于 $\boldsymbol{x}$ 的观测属于第 t 组的后验概率, $f_t(\boldsymbol{x})$ 为第 t 组的分布密度在 $\boldsymbol{X} = \boldsymbol{x}$ 处的值, $f(\boldsymbol{x})$ 为非条件密度 $\sum_{i=1}^{G} q_t f_t(\boldsymbol{x})$。
- 按照 Bayes 理论, 自变量为 $\boldsymbol{x}$ 的观测属于第 t 组的后验概率 $p(t|\boldsymbol{x}) = q_t f_t(\boldsymbol{x})/f(\boldsymbol{x})$。
- 于是, 可以把自变量 $\boldsymbol{X}$ 的取值空间 R^p 划分为 G 个区域 $R_t, t = 1, \ldots, G$, 使得当 $\boldsymbol{X}$ 的取值 $\boldsymbol{x}$ 属于 R_t 时后验概率在第 t 组最大, 即

$$p(t|\boldsymbol{x}) = \max_{s=1,\ldots,G} p(s|\boldsymbol{x}), \quad \forall \boldsymbol{x} \in R_t$$

- 建立的判别规则为：计算自变量 $\boldsymbol{x}$ 到每一个组中心的广义平方距离, 并把 $\boldsymbol{x}$ 判入最近的类。

- 广义平方距离的计算可能使用合并的协方差阵估计或者单独的协方差阵估计, 并与先验概率有关, 定义为

$$D_t^2(\boldsymbol{x}) = d_t^2(\boldsymbol{x}) + g_1(t) + g_2(t)$$

- 其中

$$\begin{aligned} d_t^2(\boldsymbol{x}) =& (\boldsymbol{x} - \boldsymbol{m}_t)^T V_t^{-1} (\boldsymbol{x} - \boldsymbol{m}_t) \\ g_1(t) =& \begin{cases} \ln |S_t| & \text{采用单类的协方差阵估计} \\ 0 & \text{采用合并协方差阵估计} \end{cases} \\ g_2(t) =& \begin{cases} \ln q_t & \text{各类先验概率不等} \\ 0 & \text{各类先验概率相等} \end{cases} \end{aligned}$$

- $V_t = S_t$(使用单个类的协方差阵估计) 或 $V_t = S$(使用合并的协方差阵估计)。

- 在使用合并协方差阵时:

$$\begin{aligned} D_t^2(\boldsymbol{x}) =& (\boldsymbol{x} - \bar{\boldsymbol{X}}_t)^T S^{-1} (\boldsymbol{x} - \bar{\boldsymbol{X}}_t) - 2 \ln q_t \\ =& \boldsymbol{x}^T S^{-1} \boldsymbol{x} + (\bar{\boldsymbol{X}}_t^T S^{-1} \bar{\boldsymbol{X}}_t - 2 \ln q_t) - 2 \boldsymbol{x}^T S^{-1} \bar{\boldsymbol{X}}_t \end{aligned}$$

- 其中 $\boldsymbol{x}^T S^{-1} \boldsymbol{x}$ 是共同的可以不考虑, 于是在比较 $\boldsymbol{x}$ 到各组中心的广义平方距离时, 只要计算线性判别函数

- $\tilde{D}_t^2(\boldsymbol{x}) = \left(-\frac{1}{2} \bar{\boldsymbol{X}}_t^T S^{-1} \bar{\boldsymbol{X}}_t + \ln q_t \right) + \boldsymbol{x}^T S^{-1} \bar{\boldsymbol{X}}_t$;

- 当 $\boldsymbol{x}$ 到第 t 组的线性判别函数最大时把 $\boldsymbol{x}$ 对应观测判入第 t 组。

- 如果使用单个类的协方差阵估计 $V_t = S_t$ 则距离函数是 x 的二次函数, 称为二次判别函数。

参数方法的判别规则为:

- 先决定是使用合并协方差阵还是单个类的协方差阵;

- 计算 $\boldsymbol{x}$ 到各组的广义距离;

- 把 $\boldsymbol{x}$ 判入最近的组。

5.2.2 PROC DISCRIM 用法

PROC DISCRIM 用法

SAS/STAT 的 DISCRIM 过程可以进行参数判别分析和非参数判别分析, 其一般格式如下：

```
PROC DISCRIM DATA= 输入数据集 选项;
    CLASS 分类变量;
    VAR 判别用自变量集合;
RUN;
```

PROC DISCRIM 选项和语句

- PROC DISCRIM 的 method=normal 选项指定参数方法，pool=yes 指定用合并方差阵，pool=no 指定用单独的方差阵。
- outstat= 指定输出判别规则的数据集。
- out= 指定存放训练样本、后验概率、交叉核实分类结果的数据集。
- testdata= 指定一个检验数据集。
- testclass 语句指定检验数据集中真实分类变量。
- testout= 指定检验数据集分类结果的数据集。

5.2.3 因子分析例子

因子分析例子—卫星遥感

- 用卫星遥感分辨作物的种类。四种遥感指标变量。

5.3 聚类分析

聚类分析

- 假设有一批样本, 不知道它们的分类, 甚至连分成几类也不知道, 希望用某种方法把观测进行合理的分类，使得同一类的观测比较接近, 不同类的观测相差较多, 就是**聚类分析**问题。
- 或称为**无监督学习**。
- 聚类分析依赖于对观测间的接近程度 (距离) 或相似程度的理解, 定义不同的距离量度和相似性量度就可以产生不同的聚类结果。

5.3.1 谱系聚类方法介绍

谱系聚类方法介绍

- 谱系聚类是一种逐次合并类的方法, 最后得到一个聚类的二叉树聚类图。
- 其想法是, 对于 n 个观测, 先计算其两两的距离得到一个距离矩阵;
- 然后把离得最近的两个观测合并为一类, 于是我们现在只剩了 $n-1$ 个类 (每个单独的未合并的观测作为一个类)。
- 计算这 $n-1$ 个类两两之间的距离, 找到离得最近的两个类将其合并, 就只剩下了 $n-2$ 个类;
- ……;
- 直到剩下两个类, 把它们合并为一个类为止。
- 聚类过程应该在某个类水平数 (即未合并的类数) 停下来。
- 决定聚类个数是一个比较困难的问题。
- 设观测个数为 n, 变量个数为 ν, G 为在某一聚类水平上的类的个数, x_i 为第 i 个观测, C_K 是当前 (水平 G) 的第 K 类, N_K 为 C_K 中的观测个数, $\bar{X}$ 为均值向量, $\bar{X}_K$ 为类 C_K 中的均值向量 (中心), $\|x\|$ 为欧氏长度;
- $T=\sum_{i=1}^{n}\|x_i-\bar{X}\|^2$ 为总离差平方和, $W_K=\sum_{i\in C_K}\|x_i-\bar{X}_K\|^2$ 为类 C_K 的类内离差平方和, $P_G=\sum W_J$ 为聚类水平 G 对应的各类的类内离差平方和的总和。

- 假设某一步聚类把类 C_K 和类 C_L 合并为下一水平的类 C_M, 则定义 $B_{KL} = W_M - W_K - W_L$ 为合并导致的类内离差平方和的增量。
- 用 $d(x, y)$ 代表两个观测之间的距离或非相似性测度, D_{KL} 为第 G 水平的类 C_K 和类 C_L 之间的距离或非相似性测度。
- 进行谱系聚类时, 类间距离可以直接计算, 也可以从上一聚类水平的距离递推得到。
- 观测间的距离可以用欧氏距离或欧氏距离的平方, 如果用其它距离或非相似性测度得到了一个观测间的距离矩阵也可以作为谱系聚类方法的输入。

距离定义

不同的类间距离定义对应不同的聚类方法。

- 类平均法 (METHOD=AVERAGE): 测量两类每对观测间的平均距离。
- 重心法 (METHOD=CENTROID): 重心法测量两个类的重心 (均值) 之间的 (平方) 欧氏距离。
- 最长距离法 (METHOD=COMPLETE): 计算两类观测间最远一对的距离。
- 最短距离法 (METHOD=SINGLE)：计算两类观测间最近一对的距离。
- 密度估计法 (METHOD=DENSITY)：按非参数密度来定义两点间的距离 d^*。如果两个点 x_i 和 x_j 是近邻 (两点距离小于某指定常数或 x_j 在距离 x_i 最近的若干点内) 则距离是两点密度估计的倒数的平均, 否则距离为正无穷。密度估计有最近邻估计 (K=)、均匀核估计 (R=) 和 Wong 混合法 (HYBRID)。
- Ward 最小方差法 (METHOD=WARD)：也称 Ward 离差平方和法。组间距离为

$$D_{KL} = B_{KL} = \|\bar{X}_K - \bar{X}_L\|^2/(1/N_K + 1/N_L)$$

5.3.2 谱系聚类类数的确定

谱系聚类类数的确定—R^2 统计量

- 决定类数的一些方法来自统计的方差分析的思想。

- R^2 统计量:
$$R^2 = 1 - \frac{P_G}{T}$$
- 其中 P_G 为分类数为 G 个类时的总类内离差平方和, T 为所有变量的总离差平方和。
- R^2 越大, 说明分为 G 个类时每个类内的离差平方和都比较小, 也就是分为 G 个类是合适的。
- 但是, 显然分类越多, 每个类越小, R^2 越大;
- 所以我们只能取 G 使得 R^2 足够大, 但 G 本身比较小, 而且 R^2 不再大幅度增加。

谱系聚类类数的确定—半偏相关

- 在把类 C_K 和类 C_L 合并为下一水平的类 C_M 时, 定义半偏相关
$$\text{半偏}R^2 = \frac{B_{KL}}{T}$$
- 其中 B_{KL} 为合并类引起的类内离差平方和的增量, 半偏相关越大, 说明这两个类越不应该合并;
- 所以如果由 G 类合并为 $G-1$ 类时如果半偏相关很大就应该取 G 类。

谱系聚类类数的确定—伪 F 统计量

- $$F = \frac{(T - P_G)/(G-1)}{P_G/(n-G)}$$
- 伪 F 统计量评价分为 G 个类的效果。
- 如果分为 G 个类合理, 则类内离差平方和 (分母) 应该较小, 类间平方和 (分子) 相对较大。
- 所以应该取伪 F 统计量较大而类数较小的聚类水平。

谱系聚类类数的确定—伪 t^2 统计量

- $$t^2 = B_{KL} / \left((W_K + W_L)/(N_K + N_L - 2)\right)$$
- 用此统计量评价合并类 C_K 和类 C_L 的效果;
- 该值大说明不应合并这两个类, 所以应该取合并前的水平。

5.3.3 谱系聚类类数的确定

PROC CLUSTER 用法

```
PROC CLUSTER DATA= 输入数据集
                 METHOD= 聚类方法 选项;
    VAR 聚类用变量;
    COPY 复制变量;
RUN;
```

其中的 VAR 语句指定用来聚类的变量。COPY 语句把指定的变量复制到 OUTTREE =的数据集中。

PROC CLUSTER 语句选项

- METHOD= 选项: 选择类间距离。包括 VERAGE, CENTROID, COMPLETE, SINGLE, DENSITY, WARD, EML, FLEXIBLE, MCQUITTY, MEDIAN, TWOSTAGE 等, 其中 DENSITY, TWOSTAGE 等方法还要额外指定密度估计方法 (K=, R= 或 HYBRID)。
- OUTTREE= 输出谱系聚类树数据集, 把谱系聚类树输出到一个数据集, 可以用 TREE 过程绘图并实际分类。
- STANDARD 选项, 把变量标准化为均值 0, 标准差 1。
- PSEUDO 选项和 CCC 选项。PSEUDO 选项要求计算伪 F 和伪 t^2 统计量, CCC 选项要求计算 R^2、半偏 R^2 和 CCC 统计量。其中 CCC 统计量也是一种考察聚类效果的统计量, CCC 较大的聚类水平是较好的。

PROC TREE 用法

TREE 过程可以把 CLUSTER 过程产生的 OUTTREE =数据集作为输入, 画出谱系聚类的树图, 并按照用户指定的聚类水平 (类数) 产生分类结果数据集。一般格式如下：

```
PROC TREE DATA =输入聚类结果数据集
          OUT= 输出数据集 GRAPHICS
          NCLUSTER= 类数 选项;
    COPY 复制变量;
RUN;
```

其中 COPY 语句把输入数据集中的变量复制到输出数据集 (实际上这些变量也必须在 CLUSTER 过程中用 COPY 语句复制到 OUTTREE =数据集)。

NCLUSTERS 选项指定保留多少个类。OUT= 选项指定保存分类结果的数据集。

5.3.4 聚类分析例子

对鸢尾花数据聚类

- 三种不同鸢尾花 (Setosa、Versicolor、Virginica), 每种 50 个观测，每个观测包括花瓣长、宽，花萼长、宽。
- 已知分类，正好用来验证聚类分析结果是否与实际相符。
- 程序:

```
proc cluster data=sasuser.iris method=ward
             outtree=otree pseudo ccc;
  var petallen petalwid sepallen sepalwid;
  copy species;
run;
```

例子结果

- 从 Cluster History 可以看到从 150 个类逐步聚类最后只剩一个类的过程。
- OB*xxx* 表示哪一个原始观测, 而 CL*xxx* 表示在哪一个聚类水平上产生的类。
- 为了找到合适的类数，注意到输出数据集 otree 中包含了各聚类水平上的 CCC、伪 F、伪 t^2 和半偏 R^2 统计量值，在 INSIGHT 中打开 otree 数据集并对这些统计量绘图。
- CCC 统计量建议取 5 类或 3 类 (局部最大值)；
- 伪 F 建议 3 类 (局部最大值)；

- 伪 t^2 建议 2 或 3 类 (局部最大值处是不应合并的, 即局部最大值处的类数加 1);
- 半偏 R^2 建议 2 或 3 类。
- 由这些指标看比较一致的是 3 类, 其次是 2 类和 5 类。
- 作出 150 个聚类水平的聚类树图会使图形过于杂乱，我们只绘制最后 30 个聚类水平:

```
proc tree data=otree graphics horizontal;
  where _ncl_ <= 30;
run;
```

- 为了得到分类结果，用 nclusters=3 指定类数，用 out=oclust 指定输出分类结果的数据集:

```
proc tree data=otree
     nclusters=3 out=oclust;
  copy species;
run;
```

- 为了对比聚类结果与实际分类，用 FREQ 过程作列联表:

```
proc freq data=oclust;
  tables species*cluster /
      nopct norow nocol;
run;
```

- 结果中 Virginica 被分错较多。

第六章　S 语言介绍

6.1 S 快速入门

S 语言历史

- S 语言: Rick Becker, John Chambers 等人在贝尔实验室开发 [1] ，著名的 C 语言、Unix 系统也是贝尔实验室开发的。
- 商业版本为 S-PLUS, 1988 年发布，现在为 Tibco Software 拥有。
- R 是一个自由软件，GPL 授权，最初由新西兰 Auckland 大学的 Ross Ihaka 和 Robert Gentleman 于 1997 年发布，实现了与 S 语言基本相同的功能和统计功能。现在由 R 核心团队开发，但全世界的用户都可以贡献软件包。见 R 的网站: `http://www.r-project.org/`。

R 的特点

- 自由软件，免费；
- 完整的程序设计语言，基于函数和对象，可以自定义函数，调入 C、C++、Fortran 编译的代码；
- 具有完善的数据类型，如向量、矩阵、因子、数据集、一般对象等，代码像伪代码一样简洁、可读。
- 强调交互式数据分析，支持复杂算法描述，图形功能强。

[1] 第一个版本开发于 1976-1980，基于 Fortran；于 1980 年移植到 Unix, 并对外发布源代码。1984 年出版的“棕皮书”

> Becker, Richard A. and John M. Chambers (1984), “S: An Interactive Environment for Data Analysis and Graphics”, Wadsworth Advanced Books Program, Belmont CA

总结了 1984 年为止的版本, 并开始发布授权的源代码。这个版本叫做旧 S。与我们现在用的 S 语言有较大差别。

1989–1988 对 S 进行了较大更新，变成了我们现在使用的 S 语言，称为第二版。1988 年出版的“蓝皮书”做了总结：

> Becker, Richard A., John M. Chambers and Allan R. Wilks (1988), “The New S Language”, Chapman and Hall, New York.

1992 年出版的“白皮书”描述了在 S 语言中实现的统计建模功能，增强了面向对象的特性。软件称为第三版，这是我们现在用的多数版本。

> Chambers, John M. and Trevor Hastie, eds. (1992), “Statistical Models in S”, Chapman and Hall, New York.

1998 年出版的“绿皮书”描述了第四版 S 语言，主要是编程功能的深层次改进。现行的 S 系统并没有都采用第四版，S-PLUS 的第 5 版才采用了 S 语言第四版。

> John M. Chambers (1998), “Programming with Data,” New York: Springer

- 实现了经典的、现代的统计方法，如参数和非参数假设检验、线性回归、广义线性回归、非线性回归、可加模型、树回归、混合模型、方差分析、判别、聚类、时间序列分析等。
- 统计科研工作者广泛使用 R 进行计算和发表算法。R 有数以千计的软件包。

R 安装、启动、退出

- 从 R 的网站 `http://www.r-project.org/`获得软件并安装。
- 在 MS Windows 系统中，把 R 的程序快捷方式复制到工作目录如 `C:\work` 中，从右键菜单选择“属性”，把“起始位置”栏清空。
- 双击 R 图标可以启动 R，为命令行界面：输入一行命令，就在后面显示计算结果。
- 可以用向上和向下箭头访问历史命令；可以从上面的命令中用 Ctrl+C 复制或 Ctrl+X 复制，Ctrl+X 会自动粘贴到当前行。
- 关闭窗口退出，退出时询问是否保存工作空间，如果保存，下次进入时可以访问前面各次已定义的变量。

R 向量例子

- R 语言以向量为最小单位。用 `<-`赋值。

```
x1 <- 0:100
```

- 可以对向量进行通常的数乘，或与一个常数相加、相减等:

```
x2 <- x1 * 2 * pi / 100
```

- 一元函数可以对向量的每个元素取值:

```
y1 <- sin(x2)
```

方便的绘图功能

- 曲线图：

```
plot(x2, y1, type="l")
```

- 添加参考线:

```
abline(h=0, lwd=2)
abline(v=(0:4)/2*pi, lty=3, col="gray")
```

- 添加一条余弦曲线:

```
y2 <- cos(x2)
lines(x2, y2, lty=2, col="green")
```

- 其它图形、图像演示:

```
demo("graphics")
demo("image")
```

R 基本统计功能演示

- 读入 CSV 格式的数据集:

```
cl <- read.csv("class.csv", header=TRUE)
```

- 概括统计:

```
summary(cl)
```

- 统计函数:

```
mean(cl$height)
var(cl$height)
```

包括 sum, mean, var, sd, min, max, range 等。

回归分析

- 回归结果返回为一个“**对象**”，可以进一步利用回归结果。

```
lm1 <- lm(weight ~ height + age + sex,
          data=cl)
print(summary(lm1))
```

- 逐步回归：

```
lm2 <- step(lm1,
  weight ~ height + age + sex, data=cl)
```

矩阵计算

- 定义矩阵:

```
A <- matrix(c(1,2, 7,3),
            ncol=2, byrow=TRUE)
A
```

  或

```
A <- rbind(c(1, 2),
           c(7, 3))
```

- 求逆:

```
Ai <- solve(A)
```

- 矩阵乘法:

```
A %*% Ai
```

- 求特征值和特征向量:

```
eigen(A)
```

- 返回结果是一个“**列表** (list)”，在一个对象中包含了多个成员，每个成员都可以是标量、向量、矩阵或列表。这种数据类型有助于从函数中返回多个值。

记录输出结果

- 例:

```
sink("myres.txt", split=TRUE)
print(A)
print(Ai)
sink()
```

- `sink` 函数指定一个输出记录文件，结果在屏幕显示的同时输出到指定文件中。
- 用空 `sink()` 关闭输出记录。

在线帮助

- 用帮助菜单中“html 帮助”查看帮助文档。“Search engine and keywords”项下面有分类的帮助。
- 在命令行，用问号后面跟随函数名查询某函数的帮助。

6.2 S 向量

S 数据类型

- 数据类型有向量、矩阵、列表、对象等。
- 常量和变量。
- 变量名规则。句点可以作为变量名的一部分，但有部分隐含作用。

6.2.1 常量

常量

- 数值型：包括整型、单精度、双精度等，一般不需要区分。
- 字符型。
- 逻辑性：TRUE 和 FALSE。
- 缺失值: 用 NA 表示。
- 复数：如 `2.2 + 3.5i`, `1i` 等。

6.2.2 向量与赋值

向量与赋值

- 用 `<-`赋值。
- 用 `c()` 函数组合向量元素。

```
marks <- c(10, 6, 4, 7, 8)
x <- c(1:3, 10:13)
x1 <- c(1, 2)
x2 <- c(3, 4)
x <- c(x1, x2)
x
```

- 显示向量时在左边方括号中提示显示行的第一个元素的下标:

```
> 1234501:1234520
 [1] 1234501 1234502 1234503 1234504 1234505 1234506
 [7] 1234507 1234508 1234509 1234510 1234511 1234512
[13] 1234513 1234514 1234515 1234516 1234517 1234518
[19] 1234519 1234520
```

比如，第 13 号元素 1234513，前面有一个“[13]”。

- `length(x)` 可以求 x 的长度。

6.2.3 向量运算

向量与标量运算

- 向量与标量的运算为每个元素与标量的运算。
- 四则运算: `+ - * / ^`。
- 如:

```
> x <- c(1, 10)
> x + 2
[1]  3 12
> x - 2
[1] -1  8
> x * 2
[1]  2 20
> x / 2
[1] 0.5 5.0
> x ^ 2
[1]   1 100
> 2 / x
[1] 2.0 0.2
> 2 ^x
[1]    2 1024
```

等长向量运算

- 为对应元素两两运算。
- 如:

```
> x1 <- c(1, 10)
> x2 <- c(4, 2)
> x1 + x2
[1]  5 12
> x1 - x2
[1] -3  8
> x1 * x2
[1]  4 20
> x1 / x2
[1] 0.25 5.00
```

不等长向量的运算

- 如果长度为倍数关系，每次从头重复利用短的一个。如

```
> x1 <- c(1, 10)
> x2 <- c(1, 3, 5, 7)
> x1 + x2
[1]  2 13  6 17
> x1 * x2
[1]  1 30  5 70
```

- 如果长度不是倍数关系，会给出警告信息。如

```
> c(1,2) + c(1,2,3)
[1] 2 4 4
警告信息:
In c(1, 2) + c(1, 2, 3) :
  长的对象长度不是短的对象长度的整倍数
```

向量函数

- 一元函数以向量为自变量，对每个元素计算。如 sqrt, log, exp, sin, cos, tan 等。
- 统计函数: sum, mean, var, sd, min, max, range 等。
- cumsum 和 cumprod 计算累加和累乘积。
- `sort(x)` 返回排序结果；`order(x)` 返回排序用的下标。如

```
> sort(c(3, 5, 1))
[1] 1 3 5
> order(c(3, 5, 1))
[1] 3 1 2
```

- 有缺失值的运算: 缺失元素参加的运算相应结果元素仍缺失。

6.2.4 产生有规律的数列

seq 函数

- seq 函数是冒号运算符的推广。
- 如: `seq(5)` 等同于 `1:5`。
- `seq(2,5)` 等同于 `2:5`。
- `seq(11, 15, by=2)` 产生 11,13,15。
- `seq(0, 2*pi, length=100)` 产生从 0 到 2π 的等间隔序列，序列长度指定为 100。
- S 函数可以带自变量名调用。
- `seq(to=5, from=2)` 等同于 2:5。

rep 函数

- 产生一个初值为零的长度为 n 的向量: `x <- rep(0, n)`。
- `rep(c(1,3), 2)` 相当于 `c(1,3,1,3)`。
- `rep(c(1,3), c(2,4))` 相当于 `c(1,1,3,3,3,3)`。
- `rep(c(1,3), each=2)` 相当于 `c(1,1,3,3)`。

6.2.5 逻辑向量

逻辑向量

- 逻辑值: TRUE, FALSE, 缺失时为 NA。
- 向量比较结果为逻辑型向量。
- 比较运算符为 `<`, `<=`, `>`, `>=`, `==`, `!=`。
- 逻辑运算符为`&`, `|`, `!`, 另外`&&`, `||` 是短路的标量逻辑与和逻辑或。
- `all(cond)` 测试 cond 的所有元素为真；`any(cond)` 测试 cond 至少一个元素为真。

6.2.6 字符型向量

字符型向量

- 字符型向量是元素为字符串的向量。
- `paste` 函数：连接两个字符型向量，元素一一对应连接。缺省用空格连接。如 `paste(c("ab", "cd"), c("ef", "gh"))` 相当于 `c("ab ef", "cd gh")`。
- 可以一对多连接,可以在连接时把数值型转换为字符型,如`paste("x", 1:3)` 相当于 `c("x 1", "x 2", "x 3")`。
- 用 `sep=` 指定分隔符，如 `paste("x", 1:3, sep="")` 相当于 `c("x1", "x2", "x3")`。
- 使用 `collapse=` 参数连接字符型向量的各个元素为一个字符串。如 `paste(c("a", "b"), collapse="")` 相等于 `"ab"`。

字符型数据获取

- 利用字符串处理的方法可以从网页文件或不规则的 Excel 文件读取数据。
- `grep`，`grepl` 函数从字符串中查询某个模式，模式用 perl 格式的**正则表达式** (regular expression) 定义。
- `sub`，`gsub` 替换某模式。
- 正则表达式功能强大但也不容易掌握。
- `substr` 和 `substring` 函数抽取子字符串，也可以修改子字符串。

6.2.7 复数向量

复数向量

- 复数常数表示如 `3.5+2.4i`, `1i`。
- 用函数 complex 生成复数向量,指定实部和虚部。如 `complex(c(1,0,-1,0), c(0,1,0,-1))` 相当于 `c(1+0i, 1i, -1+0i, -1i)`。
- 可以用 mod 和 arg 指定模和辐角,如 `complex(mod=1, arg=(0:3)/2*pi)` 结果同上。
- 用 `Re(z)` 求 z 的实部，用 `Im(z)` 求 z 的虚部，用 `Mod(z)` 或 `abs(z)` 求 z 的模，用 `Arg(z)` 求 z 的辐角，用 `Conj(z)` 求 z 的共轭。
- `log`, `exp`, `sin` 等函数对复数也有定义。

6.2.8 向量下标

向量下标

- 设 `x <- c(1, 4, 6.25)`。
- `x[2]` 取出第二个元素；`x[2] <- 99` 修改第二个元素。
- `x[c(1,3)]` 取出第 1、3 号元素；`x[c(1,3)] <- c(11, 13)` 修改第 1、3 号元素。
- 下标可重复，如 `x[c(1,3,1)]` 为 1, 6.25, 1。
- 负下标表示“扣除”，如 `x[-c(1,3)]` 为第二个元素 4。

逻辑下标

- 下标可以是一个条件，如 `x[x>3]` 取出 4，6.25。
- 例：示性函数。设输入向量 x, 求每个元素的示性函数值 (元素非负时取 1，否则取 0)。

```
y <- numeric(length(x))
y[x >= 0] <- 1
y[x < 0] <- 0 # 此语句多余
```

元素名

- 向量可以为每个元素命名。如

```
ages <- c(" 李明"=30, " 张聪"=25, " 刘颖"=28)
```

  或

```
ages <- c(30, 25, 28)
names(ages) <- c(" 李明", " 张聪", " 刘颖")
```

- 这时可以用 `ages[" 张聪"]`, `ages[c(" 李明", " 刘颖")]` 这样的方法访问。
- 这样建立了字符串到数值的映射表。
- 在矩阵和数据框中尤其有用。

整个数组

- 用 `x[] <- 0` 把 x 的所有元素赋 0，但长度不变。
- 这与 `x <- 0` 不同，后者把 x 变成标量 0。