R 语言教程

李东风

2023-04-26

目录

前言 **21**

I 介绍 **25**

1 R 语言介绍 **27**

1.1 R 的历史和特点 27

1.2 参考资料 28

1.3 R 的下载与安装 30

1.4 R 扩展软件包的安装与管理 30

1.5 基本 R 软件的用法 34

1.6 RStudio 软件 35

1.7 Qmd 文件 39

1.8 练习 40

2 R 语言入门运行样例 **41**

2.1 命令行界面 41

2.2 四则运算 41

2.3 数学函数 42

2.4 函数调用和自定义函数 44

2.5 输出 45

2.6 向量计算与变量赋值 46

2.7 工作空间介绍 47

2.8 绘图示例 48

2.9 汇总统计示例 52
2.10 运行源程序文件 55
2.11 附录：数据 56

II 数据类型与相应的运算 57

3 常量与变量 59

3.1 常量 59
3.2 变量 59
3.3 R 数据类型 60

4 数值型向量及其运算 61

4.1 数值型向量 61
4.2 向量运算 62
4.3 向量函数 64
4.4 复数向量 66
4.5 练习 66

5 逻辑型向量及其运算 69

5.1 逻辑型向量与比较运算 69
5.2 逻辑运算 70
5.3 逻辑运算函数 71

6 字符型数据及其处理 73

6.1 字符型向量 73
6.2 转义字符和原始字符串 {p-t-char-raw} 73
6.3 paste 函数 74
6.4 转换大小写 74
6.5 字符串长度 74
6.6 取子串 74
6.7 类型转换 75
6.8 字符串替换功能 76

6.9 正则表达式 76

7 R 向量下标和子集 79

7.1 正整数下标 79

7.2 负整数下标 79

7.3 空下标与零下标 80

7.4 下标超界 80

7.5 逻辑下标 81

7.6 `which()`、`which.min()`、`which.max()` 函数 82

7.7 元素名 82

7.8 用 R 向量下标作映射 83

7.9 集合运算 84

7.10 练习 85

8 R 数据类型的性质 87

8.1 存储模式与基本类型 87

8.2 类型转换与类型升档 89

8.3 属性 89

8.4 类属 91

8.5 `str()` 函数 92

9 R 日期时间 93

9.1 R 日期和日期时间类型 93

9.2 从字符串生成日期数据 93

9.3 时区和时区转换 94

9.4 从数值生成日期数据 95

9.5 日期和日期时间之间的转换 96

9.6 日期显示格式 96

9.7 访问日期时间的组成值 97

9.8 日期舍入计算 98

9.9 日期计算 99

9.10 基本 R 软件的日期功能 103
9.11 练习 106

10 R 因子类型 **109**

10.1 因子 109
10.2 `cut()` 函数 110
10.3 `table()` 函数 111
10.4 `tapply()` 函数 112
10.5 forcats 包的因子函数 112
10.6 练习 116

11 列表类型 **119**

11.1 列表 119
11.2 列表元素访问 120
11.3 列表类型转换 122
11.4 连接列表 123
11.5 返回列表的函数示例–`strsplit()` 124

12 R 矩阵和数组 **127**

12.1 R 矩阵 127
12.2 矩阵子集 128
12.3 `cbind()` 和 `rbind()` 函数 131
12.4 矩阵运算 132
12.5 逆矩阵与线性方程组求解 135
12.6 `apply()` 函数 136
12.7 多维数组 137
12.8 列表类型的数组 139

13 数据框 **141**

13.1 数据框 141
13.2 `tibble` 类型 146
13.3 练习 151

14 工作空间和变量赋值 **153**

14.1 工作空间 153

14.2 非法变量名 154

14.3 变量赋值与绑定 154

14.4 环境 158

III 编程 **159**

15 R 输入输出 **161**

15.1 输入输出的简单方法 161

15.2 读取 CSV 文件 163

15.3 保存与恢复 R 变量 173

15.4 Excel 表访问 173

15.5 文件访问 175

15.6 中文编码问题 180

15.7 目录和文件管理 184

16 程序控制结构 **187**

16.1 表达式 187

16.2 分支结构 187

16.3 循环结构 191

16.4 R 中判断条件 194

16.5 管道控制 194

17 函数 **197**

17.1 函数基础 197

17.2 变量作用域 207

18 R 程序效率 **213**

18.1 R 的运行效率 213

18.2 性能比较和向量化编程示例 214

18.3 减少显式循环 218

18.4 避免制作副本 219
18.5 用性能分析工具找到程序效率的瓶颈 220
18.6 R 的计算函数 223
18.7 并行计算 229

19 函数进阶 237

19.1 函数调用的各种形式 237
19.2 嵌套定义与句法作用域 (lexical scoping) 239
19.3 辅助嵌套函数 241
19.4 懒惰求值 241
19.5 程序调试 242
19.6 单元测试 249
19.7 环境 250

IV 研究项目管理和研究报告 259

20 研究项目管理 261

20.1 可重复式科学研究 261
20.2 备份 261
20.3 源代码管理 262
20.4 单元测试 262
20.5 阶段任务管理 262
20.6 文学式编程 268

21 Markdown 格式 271

21.1 介绍 271
21.2 Markdown 格式文件的应用 271
21.3 markdown 格式说明 272
21.4 附录：pandoc 软件介绍 285

22 Quarto 格式文件 **287**

22.1 介绍 287

22.2 安装 287

22.3 简单样例文件 288

22.4 RStudio 的可视化编辑 288

22.5 生成输出文件 289

22.6 插入 R 程序代码 291

22.7 代码段选项 292

22.8 插图 292

22.9 插入表格 293

22.10数学公式 294

22.11设置 297

22.12定理类 302

22.13引用 303

22.14代码段缓存 304

22.15制作演示文稿 305

22.16运行 Julia 程序 307

22.17用作科学计算笔记本 307

22.18问题 308

22.19附录：TinyTex 的安装使用 308

V 数据整理与汇总 **311**

23 数据整理 **313**

23.1 tidyverse 系统 313

23.2 查看数据框一般信息 315

23.3 用 `filter()` 选择行子集 316

23.4 按行序号选择行子集 316

23.5 用 `sample_n()` 对观测随机抽样 317

23.6 用 `distinct()` 去除重复行 318

23.7 用 drop_na() 去除指定的变量有缺失值的行 . . . 319
23.8 用 select() 选择列子集 . . . 319
23.9 取出单个变量为向量 . . . 321
23.10用 arrange() 排序 . . . 322
23.11用 rename() 修改变量名 . . . 324
23.12用 relocate() 调整变量次序 . . . 325
23.13用 mutate() 计算新变量 . . . 325
23.14用 tranmute() 生成新变量的数据框 . . . 327
23.15缺失值填补 . . . 327
23.16用管道连接多次操作 . . . 329
23.17expand_grid() 函数 . . . 330
23.18宽表转换为长表 . . . 331
23.19长表转换为宽表 . . . 337
23.20拆分数据列 . . . 342
23.21合并数据列 . . . 346
23.22数据框纵向合并 . . . 346
23.23横向合并 . . . 348
23.24数据集的集合操作 . . . 351
23.25标准化 . . . 352
23.26读入多个文件 . . . 354
23.27数据库访问与 SQL . . . 357

24 数据汇总 **359**

24.1 用 dplyr 作数据汇总 . . . 359
24.2 多个变量的汇总 . . . 362
24.3 用 dplyr 作数据分组汇总 . . . 368
24.4 交叉分类的汇总 . . . 371
24.5 在自定义函数中使用 dplyr 功能 . . . 374
24.6 数据分析流程控制 . . . 377
24.7 基本 R 的汇总功能 . . . 377

24.8 用基本 R 作分类汇总 385
24.9 用 plyr 包进行分类概括 387
24.10练习 389

25 函数式编程和数据框列表列 391

25.1 函数式编程介绍 391
25.2 泛函 393
25.3 函数工厂 418
25.4 tibble 中的列表列 422

VI 绘图 431

26 基本 R 绘图 433

26.1 常用高级图形 433
26.2 低级图形函数 463
26.3 图形参数 473
26.4 图形输出 479
26.5 包含多种中文字体的图形 479
26.6 其它图形 481

27 ggplot 作图入门 485

27.1 介绍 485
27.2 作图的一般原则 488
27.3 散点图：ggplot 入门 489
27.4 折线图、分组、切片 505
27.5 数据变换与条形图 519
27.6 直方图与密度估计 538
27.7 绘图语法介绍 545
27.8 更多图形种类 547
27.9 刻度 (scale) 576
27.10如何使用颜色 583

27.11标题、标注、指南 590
27.12拼接图形 596
27.13主题 600
27.14图形定制调整 600
27.15在自定义函数中调用图形函数 604
27.16交互作图、动态图形 605
27.17参考文献 605

28 ggplot 的各种图形 607

28.1 介绍 607
28.2 表现数量 607
28.3 表现分布 628
28.4 表现比例 678
28.5 表现多个变量间的关系 691
28.6 时间序列图 713
28.7 拟合曲线图 732
28.8 表现不确定性 743

VII 统计模型 757

29 探索性数据分析 759

29.1 单个分类变量分布 762
29.2 单个区间型变量分布 765
29.3 用 skimr 进行简单概括 776
29.4 两个分类变量之间的关系 777
29.5 一个分类变量与一个区间型变量之间的关系 784
29.6 两个区间型变量之间的关系 788
29.7 一个分类变量对两个区间型变量的影响 792
29.8 时间序列数据 793

30 R 初等统计分析 **795**

30.1 概率分布 795

30.2 最大似然估计 796

30.3 单样本均值的检验和置信区间 801

30.4 均值比较 815

30.5 单个比例的假设检验和置信区间 825

30.6 两个比例的比较 830

30.7 方差的假设检验和置信区间 835

30.8 拟合优度检验 838

30.9 检验分布类型 845

30.10列联表独立性卡方检验 846

30.11非参数检验 851

30.12使用 infer 包计算假设检验和置信区间 852

31 R 相关与回归 **877**

31.1 相关分析 877

31.2 一元回归分析 895

32 R 多元回归 **911**

32.1 模型 911

32.2 参数估计 912

32.3 R 的多元回归程序 912

32.4 模型的检验 916

32.5 回归自变量筛选 919

32.6 过度拟合示例 925

32.7 残差诊断 934

32.8 多重共线性 942

32.9 强影响点分析 943

32.10协方差分析 944

32.11嵌套模型的比较 953

32.12非嵌套模型的比较 954

32.13参数线性组合的检验 955
32.14拟合与预测 962
32.15利用线性回归模型做曲线拟合 965
32.16分组建立多个模型 977
32.17R 语言公式界面 981
32.18R 的设计阵 982
32.19R 对照矩阵 989
32.20R 信息提取函数 993

33 R 非参数回归 995

33.1 模型 995
33.2 样条平滑 996
33.3 局部多项式曲线平滑 998
33.4 样条函数回归 1000
33.5 线性可加模型 1002

34 R 方差分析 1007

34.1 单因素方差分析 1007
34.2 协方差分析 1017
34.3 两因素方差分析 1018

35 线性混合模型 1021

35.1 介绍 1021
35.2 加权线性回归 1023
35.3 线性混合模型 1026
35.4 非线性混合模型介绍 1046

36 广义线性模型 1047

36.1 模型 1047
36.2 泊松回归 1049
36.3 逻辑斯谛回归 1067
36.4 多水平有序因变量的逻辑斯谛回归 1088

37 整洁建模 **1093**

37.1 建模一般步骤 1093

38 统计学习介绍 **1095**

38.1 统计学习的基本概念和一般步骤 1096

38.2 Hitters 数据分析 1098

38.3 Heart 数据分析 1131

38.4 汽车销量数据分析 1143

38.5 波士顿郊区房价数据 1153

38.6 支持向量机方法 1164

38.7 附录 1174

39 R 时间序列分析 **1183**

39.1 基本概念 1183

39.2 时间序列数据类型 1184

39.3 基本分析与作图 1191

39.4 ARIMA 建模和模拟 1194

39.5 谱密度估计 1228

39.6 GARCH 类模型 1231

39.7 多元时间序列基础统计 1231

39.8 VAR 模型 1231

39.9 协整分析 1231

39.10因果性检验 1232

40 随机模拟 **1233**

40.1 随机数 1233

40.2 `sample()` 函数 1234

40.3 随机模拟示例 1234

40.4 bootstrap 1238

40.5 附录：完全试验方案设计函数 1244

VIII 特殊应用 1247

41 R 语言的文本处理 1249

41.1 介绍 1249

41.2 字符型常量与字符型向量 1249

41.3 字符串连接、重复 1250

41.4 格式化输出 1252

41.5 字符串长度 1254

41.6 取子串 1255

41.7 字符串变换 1257

41.8 简单匹配与查找 1259

41.9 字符串替换 1260

41.10字符串拆分 1261

41.11文本文件读写 1263

42 正则表达式 1267

42.1 正则表达式详解 1267

42.2 stringr 包的正则表达式函数 1287

42.3 附录：利用基本 R 函数进行正则表达式处理 1295

43 文本应用 1309

43.1 正则表达式应用例子 1309

43.2 网站数据获取 1324

43.3 中文分词与词云 1326

44 数据库访问 1329

44.1 介绍 1329

44.2 SQLite 数据库访问 1329

44.3 duckdb 数据库访问 1333

44.4 用 dbplyr 包访问数据库 1333

44.5 SQL 命令简介 1335

44.6 用 dm 包管理表 1343

44.7 访问 Oracle 数据库 1343
44.8 MySQL 数据库访问 1344
44.9 利用 RODBC 访问 Access 数据库 1346
44.10arrow 包 1346
44.11层次数据 1348

IX 用 Rcpp 访问 C++ 代码 1359

45 Rcpp 介绍 1361
45.1 Rcpp 的用途 1362
45.2 Rcpp 入门样例 1362

46 R 与 C++ 的类型转换 1373
46.1 用 `wrap()` 把 C++ 变量返回到 R 中 1373
46.2 用 `as()` 函数把 R 变量转换为 C++ 类型 1374
46.3 `as()` 和 `wrap()` 的隐含调用 1374

47 Rcpp 属性 1375
47.1 Rcpp 属性介绍 1375
47.2 在 C++ 源程序中指定要导出的 C++ 函数 1376
47.3 在 R 中编译链接 C++ 代码 1377
47.4 Rcpp 属性的其它功能 1378

48 Rcpp 提供的 C++ 数据类型 1383
48.1 RObject 类 1383
48.2 IntegerVector 类 1384
48.3 NumericVector 类 1387
48.4 NumericMatrix 类 1391
48.5 Rcpp 的其它向量类 1394
48.6 Rcpp 提供的其它数据类型 1396

49 Rcpp 糖 **1403**

49.1 简单示例 1403

49.2 向量化的运算符 1404

49.3 用 Rcpp 访问数学函数 1405

49.4 用 Rcpp 访问统计分布类函数 1405

49.5 在 Rcpp 中产生随机数 1407

49.6 返回单一逻辑值的函数 1407

49.7 返回糖表达式的函数 1408

49.8 R 与 Rcpp 不同语法示例 1411

49.9 用 RcppArmadillo 执行矩阵运算 1412

50 用 Rcpp 帮助制作 R 扩展包 **1417**

50.1 不用扩展包共享 C++ 代码的方法 1417

50.2 生成扩展包 1418

50.3 重新编译 1420

50.4 建立 C++ 用的接口界面 1420

X 其它 **1421**

51 R 编程例子 **1423**

51.1 R 语言 1423

51.2 概率 1427

51.3 科学计算 1428

51.4 统计计算 1429

51.5 数据处理 1436

51.6 文本处理 1440

52 使用经验 **1441**

52.1 文件管理 1441

52.2 程序格式 1441

A R Markdown 文件格式 1443

A.1 R Markdown 文件 1443

A.2 R Markdown 文件的编译 1444

A.3 在 R Markdown 文件中插入 R 代码 1446

A.4 输出表格 1447

A.5 利用 R 程序插图 1448

A.6 冗余输出控制 1449

A.7 代码段选项 1449

A.8 章节目录链接问题 1458

A.9 其它编程语言引擎 1459

A.10 交互内容 1459

A.11 属性设置 1459

A.12 LaTeX 和 PDF 输出 1464

A.13 生成期刊文章 1468

A.14 附录：经验与问题 1469

B 用 bookdown 制作图书 1471

B.1 介绍 1471

B.2 一本书的设置 1472

B.3 章节结构 1474

B.4 书的编译 1475

B.5 交叉引用 1476

B.6 定理类编号 1478

B.7 文献引用 1479

B.8 插图 1481

B.9 表格 1482

B.10 数学公式的设置 1483

B.11 使用经验 1484

B.12 bookdown 的一些使用问题 1487

C 用 R Markdown 制作简易网站 1489

C.1 介绍 1489

C.2 简易网站制作 1489

C.3 用 blogdown 制作网站 1492

D 制作幻灯片 1499

D.1 介绍 1499

D.2 Slidy 幻灯片 1499

D.3 MS PowerPoint 幻灯片 1503

D.4 Bearmer 幻灯片格式 1504

D.5 R Presentation 格式 1504

前言

这是李东风开设《统计软件》等课程的讲义，也可以用作《数据科学》的入门教材。本书使用了其它教材的例子和讲法，仅供学生内部使用，不是公开出版图书。鉴于本人水平有限，错漏之处难免，欢迎指出错误或提出改进意见。

尽管本书中包含假设检验、回归分析等统计方法内容，但是侧重点是这些方法的使用，不能当作统计学的入门教材使用。

相关下载：

- `Rbook-data.zip`：一些配套数据的打包文件
- `bookdown-template-v0-6.zip`：R Markdown 和 bookdown 的模板

书中的数学公式使用 MathJax 库显示，下面是数学公式测试。如果数学公式显示的中文不正常，在浏览器中用鼠标右键单击中文公式，在弹出的菜单中选择 `Math Settings--Math Renderer` 选 HTML-CSS 或 SVG 即可。

公式测试：

$$f(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

中文公式测试：

$$\text{相对误差} = \frac{a - A}{A}$$

使用本教程必须安装的软件包：

- tidyverse
- bookdown
- xtable
- bench

本教程中用到的软件包列表（更新 R 软件后可以在一个基本 R 中运行如下命令）：

```
pkgs <- c(
  "assertthat",
  "backports", "base64enc", "BH", "bindr", "bindrcpp", "bookdown", "broom",
  "callr", "car", "cellranger", "cli", "clipr", "clorspace",
```

```
  "cowplot", "crayon", "curl",
  "DBI", "dbplyr", "dichromat", "digest", "dplyr",
  "evaluate",
  "forcats",
  "ggplot2", "ggrepel", "glue", "gtable",
  "haven", "highr", "hms", "htmltools", "httr",
  "jsonlite",
  "knitr",
  "labeling", "lazyeval", "lubridate",
  "magrittr", "markdown", "microbenchmark", "mime", "mnormt", "modelr", "munsell",
  "NHANES",
  "openssl",
  "pillar", "pkgconfig", "plogr", "plyr", "psych", "purr",
  "R6", "RColorBrewer", "Rcpp", "readr", "readxl",
  "rematch", "reprex", "reshape2", "rlang", "rmarkdown",
  "rprojroot", "rstudioapi", "rvest",
  "scales", "selectr", "socviz", "stringi", "stringr",
  "tibble", "tidyr", "tidyselect", "tidyverse",
  "utf8",
  "viridisLite",
  "whisker",
  "xml2", "xtable",
  "yaml"
)
install.packages(unique(pkgs))
```

编译本教程所用的 R 软件环境：

```
devtools::session_info()
```

```
## - Session info ---------------------------------------------------------------
##  setting  value
##  version  R version 4.2.1 (2022-06-23 ucrt)
##  os       Windows 10 x64 (build 19044)
##  system   x86_64, mingw32
##  ui       RTerm
##  language (EN)
##  collate  Chinese (Simplified)_China.utf8
##  ctype    Chinese (Simplified)_China.utf8
##  tz       Asia/Taipei
```

```
## date     2023-04-26
## pandoc   2.19.2 @ D:/RStudio/resources/app/bin/quarto/bin/tools/ (via rmarkdown)
##
## - Packages ---------------------------------------------------------------
## package     * version    date (UTC) lib source
## bookdown      0.29       2022-09-12 [1] CRAN (R 4.2.1)
## cachem        1.0.6      2021-08-19 [1] CRAN (R 4.2.1)
## callr         3.7.2      2022-08-22 [1] CRAN (R 4.2.1)
## cli           3.4.1      2022-09-23 [1] CRAN (R 4.2.2)
## crayon        1.5.1      2022-03-26 [1] CRAN (R 4.2.1)
## devtools      2.4.4      2022-07-20 [1] CRAN (R 4.2.1)
## digest        0.6.29     2021-12-01 [1] CRAN (R 4.2.1)
## ellipsis      0.3.2      2021-04-29 [1] CRAN (R 4.2.1)
## evaluate      0.16       2022-08-09 [1] CRAN (R 4.2.1)
## fastmap       1.1.0      2021-01-25 [1] CRAN (R 4.2.1)
## fs            1.5.2      2021-12-08 [1] CRAN (R 4.2.1)
## glue          1.6.2      2022-02-24 [1] CRAN (R 4.2.1)
## htmltools     0.5.3      2022-07-18 [1] CRAN (R 4.2.1)
## htmlwidgets   1.5.4      2021-09-08 [1] CRAN (R 4.2.1)
## httpuv        1.6.5      2022-01-05 [1] CRAN (R 4.2.1)
## knitr         1.42       2023-01-25 [1] CRAN (R 4.2.2)
## later         1.3.0      2021-08-18 [1] CRAN (R 4.2.1)
## lifecycle     1.0.3      2022-10-07 [1] CRAN (R 4.2.2)
## magrittr      2.0.3      2022-03-30 [1] CRAN (R 4.2.1)
## memoise       2.0.1      2021-11-26 [1] CRAN (R 4.2.1)
## mime          0.12       2021-09-28 [1] CRAN (R 4.2.0)
## miniUI        0.1.1.1    2018-05-18 [1] CRAN (R 4.2.1)
## pkgbuild      1.3.1      2021-12-20 [1] CRAN (R 4.2.1)
## pkgload       1.3.0      2022-06-27 [1] CRAN (R 4.2.1)
## prettyunits   1.1.1      2020-01-24 [1] CRAN (R 4.2.1)
## processx      3.7.0      2022-07-07 [1] CRAN (R 4.2.1)
## profvis       0.3.7      2020-11-02 [1] CRAN (R 4.2.1)
## promises      1.2.0.1    2021-02-11 [1] CRAN (R 4.2.1)
## ps            1.7.1      2022-06-18 [1] CRAN (R 4.2.1)
## purrr         1.0.0      2022-12-20 [1] CRAN (R 4.2.2)
## R6            2.5.1      2021-08-19 [1] CRAN (R 4.2.1)
## Rcpp          1.0.9      2022-07-08 [1] CRAN (R 4.2.1)
## remotes       2.4.2      2021-11-30 [1] CRAN (R 4.2.1)
## rlang         1.0.6      2022-09-24 [1] CRAN (R 4.2.1)
## rmarkdown     2.20       2023-01-19 [1] CRAN (R 4.2.2)
```

```
## rstudioapi    0.14       2022-08-22 [1] CRAN (R 4.2.1)
## sessioninfo   1.2.2      2021-12-06 [1] CRAN (R 4.2.1)
## shiny         1.7.2      2022-07-19 [1] CRAN (R 4.2.1)
## stringi       1.7.8      2022-07-11 [1] CRAN (R 4.2.1)
## stringr       1.5.0.9000 2022-12-04 [1] Github (tidyverse/stringr@2d2f81b)
## urlchecker    1.0.1      2021-11-30 [1] CRAN (R 4.2.1)
## usethis       2.1.6      2022-05-25 [1] CRAN (R 4.2.1)
## vctrs         0.5.1      2022-11-16 [1] CRAN (R 4.2.2)
## xfun          0.37       2023-01-31 [1] CRAN (R 4.2.2)
## xtable        1.8-4      2019-04-21 [1] CRAN (R 4.2.1)
## yaml          2.3.5      2022-02-21 [1] CRAN (R 4.2.0)
##
##  [1] D:/R/R-4.2.1/library
##
## ------------------------------------------------------------------------------
```

Part I

介绍

Chapter 1

R 语言介绍

1.1 R 的历史和特点

1.1.1 R 的历史

R 语言来自 S 语言，是 S 语言的一个变种。S 语言由 Rick Becker, John Chambers 等人在贝尔实验室开发，著名的 C 语言、Unix 系统也是贝尔实验室开发的。

S 语言第一个版本开发于 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 & Chambers, 1984) 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” (Becker et al., 1988) 做了总结。

1992 年出版的 “白皮书” (J. M. Chambers & Hastie, 1992) 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。

1998 年出版的 “绿皮书” (J. M. Chambers, 2008) 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。

S 语言商业版本为 S-PLUS, 1988 年发布，现在为 Tibco Software 拥有。命运多舛，多次易主。现在已经基本被 R 语言取代。

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，R 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。R 的网站: http://www.r-project.org/

1.1.2 R 的特点

1.1.2.1 R 语言一般特点

- 自由软件，免费、开放源代码，支持各个主要计算机系统；
- 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，支持缺失值，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;
- 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 有上万扩展包 (截止 2021 年 7 月在 R 扩展包主要分发网站 CRAN 上有一万七千多个)。

1.1.2.2 R 语言和 R 软件的技术特点

S 语言作者，R 语言专家 J. M. Chambers((J. Chambers, 2016)) 指出 R 的本质特征：

- R 中所有的存在都是对象；
- R 中发生的动作都是函数调用。

详细地说 R 有如下技术特点：

- 函数编程（functional programming）。R 语言虽然不是严格的 functional programming 语言，但可以遵照其原则编程，得到可验证的可靠程序。
- 支持对象类和类方法。基于对象的程序设计。
- 是动态类型语言，解释执行，运行速度较慢。
- 数据框是基本的观测数据类型，类似于数据库的表。
- 开源软件（Open source software）。可深入探查，开发者和用户交互。
- 可以用作 C 和 C++、FORTRAN 语言编写的算法库的接口。
- 主要数值算法采用已广泛测试和采纳的算法实现，如排序、随机数生成、线性代数（LAPACK 软件包）。

1.2 参考资料

1.2.1 推荐参考书

- Hadley Wickham and Garrett Grolemund(2022). R for Data Science，https://r4ds.hadley.nz/, 2nd ed. 讲基本的数据整理、汇总。
- Hadley Wickham(2019). Advanced R, 2nd ed., https://adv-r.hadley.nz/, Chapman & Hall/CRC The R Series. 高级 R 编程，属于对 R 高级编程技术的讲解。

- Hadley Wickham(2016). ggplot2 Elegant Graphics for Data Analysis, 2nd ed., https://ggplot2-book.org/, Springer. 优雅易用的 R 作图功能。
- Susan Holmes, Wolfgang Huber(2020). Modern Statistics for Modern Biology, https://www.huber.embl.de/msmb/index.html.
  R 的统计功能在生物学中的应用。

1.2.2 其它参考书

- R 网站上的初学者手册 “An Introduction to R” 和其它技术手册。
- John M. Chambers(2008), Software for Data Analysis-Programming with R, Springer.
- Venables, W. N. & Ripley, B. D.(2002). Modern Applied Statistics with S (MASS), Springer.
- Max Kuhn and Julia Silge(2022). Tidy Modeling with R. https://www.tmwr.org/
- Chester Ismay and Albert Y. Kim(2019). Statistical Inference via Data Science: A moderndive into R and the tidyverse. CRC Press. https://moderndive.com/index.html 免费的入门统计学教材，用 R 和 tidyverse。
- David Diez, Mine Cetinkaya-Rundel, and Christopher Barr(2019). OpenIntro Statistics, 4th Ed. https://www.openintro.org/index.php 免费统计学教材。
- Kieran Healy. Data Visualization: A practical introduction. https://socviz.co/index.html 数据可视化。
- Claus O. Wilke. Fundamentals of Data Visualization. https://serialmentor.com/dataviz/ 数据可视化。
- 赵鹏，谢益辉，黄湘云 (2021). 现代统计图形. 人民邮电出版社。讲 R 的各种各样的图形。
- Winston Chang(2019). R Graphics Cookbook, 2nd edition. https://r-graphics.org/ 各种 R 图形的代码集锦。
- Jenny Bryan. Happy GIT with R. https://happygitwithr.com/ GIT 是软件版本管理和协作工具软件。
- Jenny Bryan. Stat 545. https://stat545.com/ 讲 R, Github, 数据管理，作图, make.
- Trevor Hastie, Jerome Friedman, Robert Tibshirani(2009). The Elements of Statistical Learning. 2nd Ed. Springer. 统计学习引论第二版。
- Paul Roback and Julie Legler(2021). Beyond Multiple Linear Regression: Applied Generalized Linear Models and Multilevel Models in R. https://bookdown.org/roback/bookdown-BeyondMLR/ 讲广义线性模型、相关误差、多层次模型。
- R.L. Kabacoff(2012)《R 语言实战》，人民邮电出版社。
- 薛毅、陈立萍（2007）《统计建模与 R 软件》，清华大学出版社。
- 汤银才（2008），《R 语言与统计分析》，高等教育出版社。
- 李东风（2006）《统计软件教程》，人民邮电出版社。

1.3 R 的下载与安装

1.3.1 R 的下载

以 MS Windows 操作系统为例。R 的主网站在https://www.r-project.org/。从 CRAN 的镜像网站下载软件，其中一个镜像如http://mirror.bjtu.edu.cn/cran/。选 “Download R for Windows-base-Download R 4.2.2 for windows” (4.2.2 是版本号，应下载网站上给出的最新版本）链接进行下载。在 “Download R for Windows” 链接的页面，除了 base 为 R 的安装程序，还有 contrib 为 R 附加的扩展软件包下载链接（一般不需要从这里下载），以及 Rtools 链接，是在 R 中调用 C、C++ 和 Fortran 程序代码时需要用的编译工具。

RStudio（https://www.rstudio.com/）是功能更强的一个 R 图形界面，在安装好 R 的官方版本后安装 RStudio 可以更方便地使用 R。

1.3.2 R 软件安装

下载官方的 R 软件后按提示安装。安装后获得一个桌面快捷方式，如 “R x64 4.2.1”。

安装官方的 R 软件后，可以安装 RStudio。平时使用可以使用 RStudio，其界面更方便，对 Qmd 文件支持更好。

如果使用 RStudio，每个分析项目需要单独建立一个 “项目”（project），每个项目也有一个工作文件夹。

1.3.3 辅助软件

R 可以把一段程序写在一个以.r 或.R 为扩展名的文本文件中，如 “date.r”, 称为一个 _ 源程序 _ 文件，然后在 R 命令行用

```
source("date.r")
```

运行源程序。这样的文件可以用记事本生成和编辑。

如果不使用 RStudio，在 MS Windows 操作系统中建议使用 notepad++ 软件，这是 MS Windows 下记事本程序的增强型软件。安装后，在 MS Windows 资源管理器中右键弹出菜单会有 “edit with notepadpp” 选项。notepad++ 可以方便地在不同的中文编码之间转换。

RStudio 则是一个集成环境，可以在 RStudio 内进行源程序文件编辑和运行。

1.4 R 扩展软件包的安装与管理

1.4.1 扩展包使用

R 有一万多个扩展软件包，提供了各种各样的功能。在安装基本 R 软件时，已经伴随安装了一些必要的扩展包，如 base, stats, graphics 等，这些包在启动 R 时会默认载入，不需要用户干预。

其它扩展包在安装好以后如果需要调用，一般需要先用 `library()` 函数载入运行。例如，载入 `readr` 扩展包，并使用其中的 `read_csv()` 函数读入一个 CSV 文件：

```
library(readr)
d <- read_csv("class.csv")
```

有些扩展包允许不载入而直接用 “扩展包名:: 函数名 ()” 的格式调用其中的功能，如：

```
d <- readr::read_csv("class.csv")
```

因为不同的扩展包可能会使用相同的函数名称，所以在两个扩展包中的函数名字有冲突时，也应该使用 “扩展包名:: 函数名 ()” 这种语法。例如，stats 包和 dplyr 包都提供了名为 `filter()` 的函数，在调用此函数时，就应该指定为 `stats::filter()` 和 `dplyr::filter()`。

在使用 `library()` 函数载入扩展包时，会显示一些诊断信息、提示信息、警告信息。在交互运行时这些信息是必要的，但是在使用 Quarto 或者 R Markdown 格式制作论文、技术报告、图书时，这些信息则会分散注意力，显得不专业。在 R Markdown 文件中，可以设置代码段选项 `results="hide"`, `warning=FALSE`, `message=FALSE`。如：

```
```{r, results="hide", warning=FALSE, message=FALSE}
library(tidyverse)
```
```

在 Quarto 文件中，只要设置代码段选项 `output: false`。如：

```
```{r}
#| output: false
library(tidyverse)
```
```

还有一种办法是用如下的比较复杂的调用方式：

```
library(tidyverse,
  warn.conflicts=FALSE,
  quietly=TRUE,
  verbose=FALSE) |>
  suppressMessages() |>
  suppressWarnings()
```

1.4.2 安装

以安装 sos 包为例。sos 包用来搜索某些函数的帮助文档。在 RStudio 中调用用 “Tools” 菜单的 “Install Packages”，输入或者在列表中选中 sos 就可以安装该扩展包。

如果不用 RStudio，在 R 图形界面选菜单 “程序包-安装程序包”，在弹出的 “CRAN mirror” 选择窗口中选择一个中国的镜像如 “China (Beijing 2)”，然后在弹出的 “Packages” 选择窗口中选择要安装的扩展软件包名称，即可完成下载和安装。

还可以用如下程序指定镜像网站 (例子中是位于清华大学的镜像网站) 并安装指定的扩展包：

```
options(repos=c(CRAN="https://mirror.tuna.tsinghua.edu.cn/CRAN/"))
install.packages("sos")
```

还可以选择扩展包的安装路径，如果权限允许，可以选择安装在 R 软件的主目录内或者用户自己的私有目录位置。由于用户的对子目录的读写权限问题，有时不允许一般用户安装扩展包到 R 的主目录中。用 `.libPaths()` 查看允许的扩展包安装位置，在 `install.packages()` 中用 `lib=` 指定安装位置：

```
print(.libPaths())
## [1] "D:/R/R-4.1.0/library"
install.packages("sos", lib=.libPaths()[1])
```

1.4.3 Github 和 BioConductor 的扩展包

还有一些扩展包没有在 CRAN 系统提供，而是放在了 Github 网站。对于这样的包，安装方法举例如下：

```
if(!require(devtools)) install.packages('devtools')
devtools::install_github("kjhealy/socviz")
```

其中 `kjhealy` 是 Github 网站的某个作者的名称，`socviz` 是该作者名下的一个 R 扩展包。

还有一些包需要从 Bioconductor 网站安装，可利用镜像网站。利用清华大学镜像的示例如下：

```
options(repos=c(CRAN="https://mirror.tuna.tsinghua.edu.cn/CRAN/"))
if (!requireNamespace("BiocManager", quietly = TRUE)){
  install.packages("BiocManager")
  BiocManager::install()
}
options(BioC_mirror="https://mirrors.tuna.tsinghua.edu.cn/bioconductor")
BiocManager::install(c("Biostrings"))
```

随后安装 Bioconductor 的其它包，也应该先设置 `repos` 和 `Bioc_mirror` 选项。

1.4.4 更新扩展包

在 RStudio 中用 “Tools–Check for Package Updates” 菜单，可以显示有新版本的扩展包，并选择进行更新。

或者在命令行用如下命令更新本地安装的所有有新版本的 CRAN 扩展包：

```
options(repos=c(CRAN="http://mirror.tuna.tsinghua.edu.cn/CRAN/"))
update.packages(checkBuilt=TRUE, ask=FALSE)
```

RStudio 在运行时会载入某些包，如 rlang，这使得 RStudio 无法更新这些包，需要在 R 的命令行程序（就是基本 R 软件）中更新。

1.4.5 迁移扩展包

在每一次 R 软件更新后，需要重新安装原来的软件包，这个过程很麻烦。如果仅仅是小的版本更新，比如从 3.5.1 变成 3.5.2，或者从 3.4.2 变成 3.5.0，可以在安装新版本后，临时将新版本的 library 子目录更名为 library0，将老版本的 library 子目录剪切为新版本的 library 子目录，然后将 library0 中所有内容复制并覆盖进入 library 子目录，删除 library0 即可。然后在基本 R 中（不要用 RStudio）运行如下命令以更新有新版本的包：

```
options(repos=c(CRAN="http://mirror.tuna.tsinghua.edu.cn/CRAN/"))
update.packages(checkBuilt=TRUE, ask=FALSE)
```

如果版本改变比较大，可以用如下方法批量地重新安装原有的软件包。首先，在更新 R 软件前，在原来的 R 中运行：

```
packages <- .packages(TRUE)
dump("packages", file="packages-20180704.R")
```

这样可以获得要安装的软件包的列表。在更新 R 软件后，运行如下程序：

```
options(repos=c(CRAN="http://mirror.tuna.tsinghua.edu.cn/CRAN/"))
source("packages-20180704.R")
install.packages(packages)
```

安装时如果提问是否安装需要编译的源代码包，最好选择否，因为安装源代码包速度很慢还有可能失败。

1.4.6 项目私有扩展包目录

在使用了 R 一段较长时间以后，会安装了许多扩展包，这些扩展包在某个时期是有用的，但是一旦某个任务完成了就不再有用。但是，用户自己无法判断哪些包已经不需要。

R 的 renv 扩展包支持每个项目保存私有的扩展包目录，这样，不同的项目使用不同的扩展包集合，不至于引发版本冲突，也不必总是为公用的 R 扩展包目录增加许多仅是短暂使用的扩展包。用 renv 管理项目私有扩展包目录，也有利于将项目迁移到其它电脑中，并且保证每次使用的扩展包版本号不变，避免多个项目共用的扩展包升级造成兼容性问题。

那些不需要安装许多扩展包的项目仍可以不启用 renv，使用公用的 R 扩展包目录。

在生成新的 RStudio 项目时，可以点击选中 “Use renv with this project” 复选框；对已有的 RStdio 项目，如果要启用 renv，可以选菜单 “Tools – Project Options – Environment”，选中 “Use renv with this project” 复选框。

启用了 renv 的项目，在安装新的扩展包时，将安装在项目目录中，而不再修改 R 的公用的扩展包目录。这也有助于将项目迁移到其它计算机上。

1.5 基本 R 软件的用法

1.5.1 基本运行

在 MS Windows 操作系统中的 R 软件有一个 R GUI 软件，即图形窗口模式的 R 软件，如图1.1。

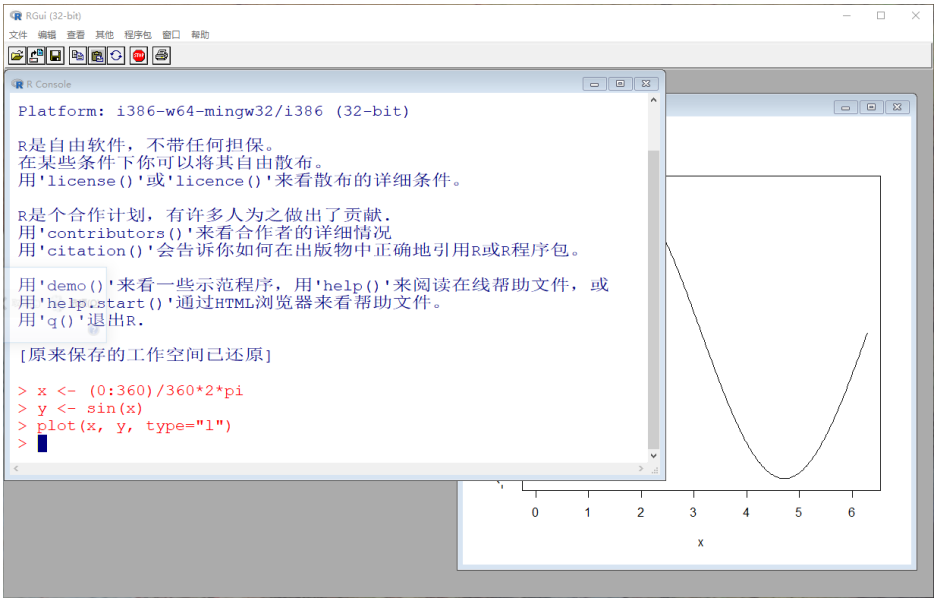


图 1.1: RGUI 截图

R GUI 中有一个命令行窗口 (R Console)，以大于号为提示符，在提示符后面键入命令，命令的文字型结果马上显示在命令下方，命令的图形结果单独显示在一个图形窗口中。

在命令行可以通过左右光标键移动光标到适当位置进行修改。可以用上下光标在已经运行过的历史命令中切换，调回已经运行过的命令，修改后重新执行。

如果某个文件如 `myprog.R` 在当前工作目录中，保存的都是 R 程序，称这样的文件为**源程序**文件。可以在命令行用如下命令运行其中的程序：

```
source("myprog.R")
```

但是，在 MS Windows 操作系统中，默认的中文编码是 GB18030 编码。R 源程序文件的中文编码可能是 GB18030 也可能是 UTF-8。UTF-8 是在世界范围更通用的编码。如果发现用上述命令运行时出现中文乱码，可能是因为源程序用了 UTF-8 编码，这时 `source()` 命令要加上编码选项如下：

```
source("myprog.R", encoding="UTF-8")
```

1.5.2 项目目录

用 R 进行数据分析，不同的分析问题需要放在不同的文件夹中。以 MS Windows 操作系统为例，设某个分析问题的数据与程序放在了 `c:\work` 文件夹中。把 R 的快捷方式从桌面复制入此文件夹，在 Windows 资源管理器中，右键单击此快捷方式，在弹出菜单中选“属性”，把“快捷方式”页面的“起始位置”的内容清除为空白，把“目标”的内容中的`--cd-to-userdocs` 删去，点击确定按钮。启动在 work 文件夹中的 R 快捷方式，出现命令行界面。这时，`C:\work` 称为**当前工作目录**。

在命令行运行如下命令可以显示当前工作目录位置：

```
getwd()
## "C:/work"
```

显示结果中的目录、子目录、文件之间的分隔符用了`/`符号，在 Windows 操作系统中一般应该使用`\\`符号，但是，在 R 的字符串中一个`\`需要写成两个，所以等价的写法是`"C:\\work"`。

不同的分析项目需要存放在不同的文件夹中，每个文件夹都放置一个“起始位置”为空的 R 快捷方式图标，分析哪一个项目，就从对应的快捷图标启动，而不是从桌面上的 R 图标启动。这样做的好处是，用到源文件和数据文件时，只要文件在该项目的文件夹中，就不需要写完全路径而只需要用文件名即可。

1.6 RStudio 软件

1.6.1 介绍

RStudio 软件是 R 软件的应用界面与增强系统，可以在其中编辑、运行 R 的程序文件，可以跟踪运行，还可以构造文字、R 结果图表融合在一起的研究报告、论文、图书、网站等。一个运行中的 RStudio 界面见图1.2。

界面一般分为四个窗格，其中编辑窗口与控制台（Console）是最重要的两个窗格。编辑窗格用来查看和编辑程序、文本型的数据文件、程序与文字融合在一起的 Rmd 文件等。控制台与基本 R 软件的命令行窗口基本相同，功能有所增强。

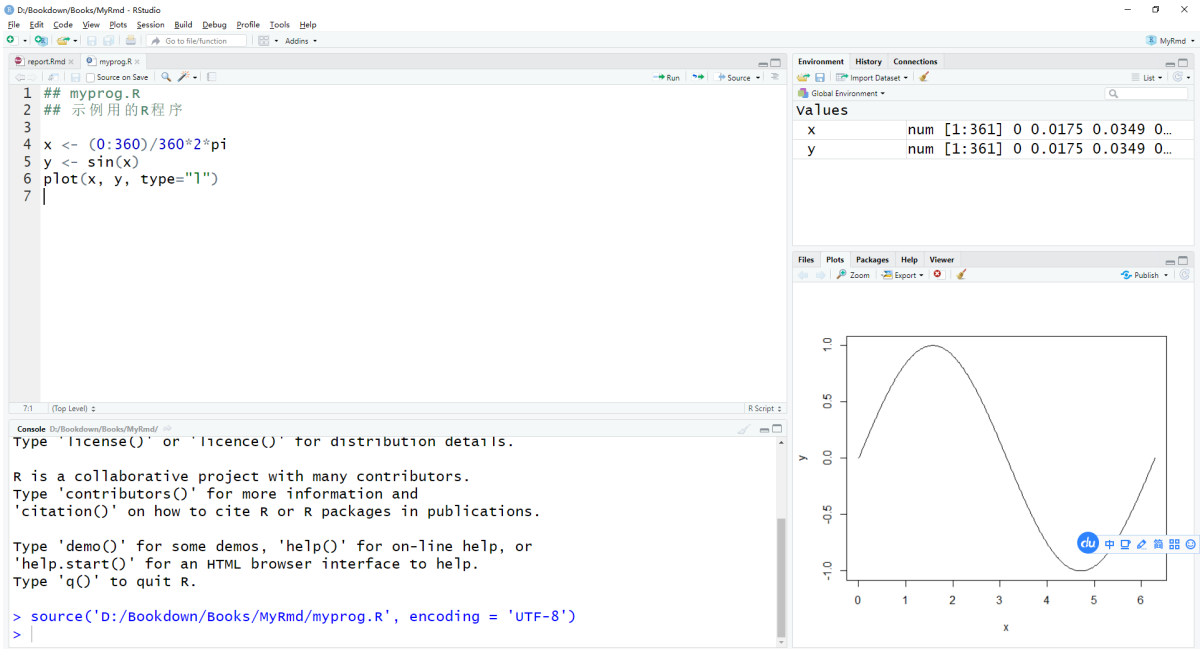


图 1.2: RStudio 截图

在编辑窗口中可以用操作系统中常用的编辑方法对源文件进行编辑，如复制、粘贴、查找、替换，还支持基于正则表达式的查找替换（关于正则表达式见41）。

其它的一些重要窗格包括：

- Files: 列出当前项目的目录（文件夹）内容。其中以`.R`或者`.r`为扩展名的是 R 源程序文件，单击某一源程序文件就可以在编辑窗格中打开该文件。
- Plots: 如果程序中有绘图结果，将会显示在这个窗格。因为绘图需要足够的空间，所以当屏幕分辨率过低或者 Plots 窗格太小的时候，可以点击“Zoom”图标将图形显示在一个单独的窗口中，或者将图形窗口作为唯一窗格显示。如何放大窗格见下面的使用技巧。
- Help: R 软件的文档与 RStudio 的文档都在这里。
- Environment: 已经有定义的变量、函数都显示在这里。
- History: 以前运行过的命令都显示在这里。不限于本次 RStdudio 运行期间，也包括以前使用 RStudio 时运行过的命令。
- Packages: 显示已安装的 R 扩展包及其文档。
- Viewer, Connection, Build, Git 等窗格。

1.6.2 项目

用 R 和 RStudio 进行研究和数据分析，每个研究问题应该单独建立一个文件夹（目录）。该问题的所有数据、程序都放在对应的文件夹中。在 RStudio 中，用“File – New Project – Existing Directory”选中该问题的目录，建立一个新的“项目”（project）。

再次进入 RStudio 后，用菜单 “File – Recent Projects” 找到已有的项目打开，然后就可以针对该项目进行分析了。这样分项目进行研究的好处是，不同项目的可以使用同名的文件而不会有冲突，程序中用到某个文件时，只需要写文件名而不需要写文件所在的目录。

一个项目还可以有项目本身的一些特殊设置，用 “Tools – Project Options” 菜单打开设置。

1.6.3 帮助

在 RStudio 中有一个单独的 Help 窗格，如果需要，可以用菜单 “View–Panes–Zoom help” 将其放大到占据整个窗口空间。但是，这一功能目前不支持放大显示字体的功能，不如在浏览器中方便。

RStudio 的帮助窗格中包含 R 软件的官方文档，以及 RStudio 软件的的文档。“Search engine and keywords” 项下面有分类的帮助。有软件包列表。

在基本 R 软件而不是 RStudio 的命令行中运行命令 `help.start()` 或者用 RGUI 的帮助菜单中 “html 帮助” 可以打开系统默认的互联网浏览器，在其中查看帮助文档。

在命令行，用问号后面跟随函数名查询某函数的帮助。用 `example(函数名)` 的格式可以运行此函数的样例，如:

```
?mean
example(mean)
```

有时仅知道一些方法的名字而不知道具体的扩展包和函数名称，可以安装 sos 扩展包（package），用 `findFn("函数名")` 查询某个函数，结果显示在互联网浏览器软件中。

1.6.4 使用技巧

RStudio 使用方法概要 PDF 下载：

rstudio-ide.pdf。

1.6.4.1 使用历史

在控制台（命令行窗格）中，除了可以用左右光标键移动光标位置，用上下光标键调回以前运行过的命令，还有一个重要的增强（以 MS Windows 操作系统为例）：键入要运行的命令的前几个字母，如 `book`，按 “Ctrl+ 向上光标键”，就可以显示历史命令中以 `book` 开头的所有命令，单击哪一个，哪一个就自动复制到命令行。这一技巧十分重要，我们需要反复运行同一命令或类似命令时，这一方法让我们很容易从许多命令历史中找到所需的命令。

1.6.4.2 放大显示某一窗格

当屏幕分辨率较低时，将整个 RStudio 界面分为四个窗格会使得每个窗格都没有足够的显示精度。为此，可以将某个窗格放大到整个窗口区域，需要使用其它窗格时再恢复到四个窗格的状态或者直接放大其它窗格到整个窗口区域。

使用菜单 “View – Panes – Zoom Source” 可以将编辑窗格放到最大，在 MS Windows 下也可以使用快捷键 “Ctrl+Alt+1”。其它操作系统也有类似的快捷键可用。使用菜单 “View – Panes – Show All Panes” 可以显示所有四个窗格。

放大其它窗格也可以用 “Ctrl + Alt + 数字”，数字与窗格的对应关系为：

- 1: 编辑窗格；
- 2: 控制台（Console）；
- 3: 帮助；
- 4: 历史；
- 5: 文件；
- 6: 图形；
- 7: 扩展包；
- 8: 已定义变量和函数；
- 9: 研究报告或网站结果显示。

1.6.4.3 运行程序

可以在命令行直接输入命令运行，文字结果会显示在命令行窗口，图形结果显示在 “Plots” 窗格中。在命令行窗口（Console）中可以用左右光标键移动光标，用上下光标键查找历史命令，输入命令的前几个字母后用 “Ctrl+ 向上光标键” 可以匹配地查找历史命令。

一般情况下，还是应该将 R 源程序保存在一个源程序文件中运行。RStudio 中 “File – New File – R Script” 可以打开一个新的无名的 R 源程序文件窗口供输入 R 源程序用。输入一些程序后，保存文件，然后点击 “Source” 快捷图标就可以运行整个文件中的所有源程序，并会自动加上关于编码的选项。

编写 R 程序的正常做法是一边写一边试验运行，运行一般不是整体的运行而是写完一部分就运行一部分，运行没有错误才继续编写下一部分。在 R 源程序窗口中，当插入光标在某行程序上的时候，点击窗口的 “Run” 快捷图标或者用快捷键 “Ctrl+Enter 键” 可以运行该行；选中若干程序行后，点击窗口的 “Run” 快捷图标或者用快捷键 “Ctrl+Enter 键” 可以运行这些行。

1.6.4.4 注释和源文件框架

R 源文件中用井号（`#`）开头的行，以及行内从井号开始的部分为注释。

对于较长的源文件，应适当分节，在 RStudio 软件中，可以用特殊的注释进行分节，并可点击源程序编辑窗格右上角的 “Outline” 快捷图标显示源程序分节结构。

这种特殊的注释格式如：

```
# 1. 辅助函数 ####
## 1.1 函数 A ####
## 1.2 函数 B ####
```

```
### 1.2.1 说明
# 2. 输入数据
# 3. 建模
```

特殊注释的末尾有 4 个以上的井号，开头用 1 到多个井号表示框架分级，开头井号越少，级别越高。

目前这样的特殊注释对中文的识别能力不好，所以需要用编号数字帮助正确识别结构层级。

1.6.4.5 中文编码问题

对于中文内容的 R 源程序、R Markdown 源文件（.Rmd 文件）、文本型数据文件 (.txt，.csv)，其中的中文内容可能有不同的编码选择，在中国除港澳台以外的地区主要使用 GB18030(基本兼容于 GB, GBK) 和 UTF-8，UTF-8 是国际上更普遍使用的统一文字编码，涉及到计算机编程时应尽可能使用此编码系统。

在 RStudio 中新生成的 R 源程序、Rmd 源文件一般自动用 UTF-8 编码。点击 RStudio 的文件窗格中显示的源文件，可以打开该源文件，但是因为已有源文件的编码不一定与 RStudio 的默认编码一致，可以会显示成乱码。为此，RStdio 提供了 “File – Reopen with Encoding” 命令，我们主要试验其中 GB18030 和 UTF-8 两种选择一般就可以解决问题。如果选择 GB18030 显示就没有乱码了，最好再用菜单 “File – Save with Encoding” 并选择 UTF-8 将其保存为 UTF-8 编码。

其它的文本格式的文件也可以类似地处理，后面将会陆续提及。

1.7 Qmd 文件

在科学研究中，R 软件可以用来分析数据，生成数据分析报表和图形。Quarto(简称 Qmd) 是一种特殊的文件格式，在这种文件中，既有 R 程序，又有说明文字，通过 R 和 RStudio 软件，可以运行其中的程序，并将说明文字、程序、程序的文字结果、图形结果统一地转换为一个研究报告，支持 Word、PDF、网页、网站、幻灯片等许多种输出格式。在打开的 Qmd 源文件中，也可以选择其中的某一段 R 程序单独运行。所以，Qmd 文件也可以作为一种特殊的 R 源程序文件。

用 RStudio 的 “File – New File – Quarto Document” 菜单就可以生成一个新的 Qmd 文件并显示在编辑窗格中，其中已经有了一些样例内容，可以修改这些样例内容为自己的文字和程序。

Qmd 文件中用`` ```{r} ``开头，用`` ``` ``结尾的段落是 R 程序段，在显示的程序段的右侧有一个向右箭头形状的小图标（类似于媒体播放图标），点击该图标就可以运行该程序段。

打开 Qmd 文件后，用编辑窗口的 Render 命令可以选择将文件整个地转换为 HTML(网页) 或者 MS Word 格式，如果操作系统中安装有 LaTeX 软件，还可以以 LaTeX 为中间格式转换为 PDF 文件。

为了将网页转换为 PDF 文件，建议使用 Chrome 浏览器打开 HTML 文件，然后选择菜单 “打印”，选打印机为 “另存为 PDF”，然后选 “更多设置”，将其中的 “缩放” 改为自定义，比例改为 “90%”。

详见20、21、22。

1.8 练习

1. 下载 R 安装程序，安装 R，建立 work 文件夹并在其中建立 R 的快捷方式。Windows 用户还需要下载 RTools 软件并安装。
2. 下载 RStudio 软件并安装。
3. 在 work 文件夹中建立一个新的项目。
4. 下载安装 notepad++ 软件 (仅 MS Windows 用户)。
5. 在 RStudio 中下载安装 sos 扩展软件包。

Chapter 2

R 语言入门运行样例

2.1 命令行界面

启动 R 软件后进入命令行界面，每输入一行命令，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；可以从已经运行过的命令中用鼠标拖选加亮后，用 Ctrl+C 复制后用 Ctrl+V 粘贴（这是 MS Windows 下的快捷键组合），粘贴的目标是当前命令行。

如果使用 RStudio 软件，有一个 “Console 窗格” 相当于命令行界面。在 RStudio 中，可以用 New File–Script file 功能建立一个源程序文件（脚本文件），在脚本文件中写程序，然后用 Run 图标或者 Ctrl+Enter 键运行当前行或者选定的部分。

2.2 四则运算

四则运算如:

```
5 + (2.3 - 1.125)*3.2/1.1 + 1.23E3
## [1] 1238.418
```

结果为 1238.418, 前面显示的结果在行首加了井号，这在 R 语言中表示注释。本教程的输出前面一般都加了井号以区分于程序语句。输出前面的方括号和序号 1 是在输出有多个值时提供的提示性序号，只有单个值时为了统一起见也显示出来了。这里 `1.23E3` 是科学记数法，表示 1.23×10^3。用星号 `*` 表示乘法，用正斜杠/表示除法。

用 `^` 表示乘方运算，如

```
2^10
## [1] 1024
```

重要提示：关闭中文输入法，否则输入一些中文标点将导致程序错误。

例：从 52 张扑克牌中任取 3 张，有多少种不同的组合可能？

解答：有

$$C_{52}^{3}=\frac{52!}{3!(52-3)!}=\frac{52\times 51\times 50}{3\times 2\times 1}$$

种，在 R 中计算如:

```
52*51*50/(3*2)
## [1] 22100
```

2.2.1 练习

1. 某人存入 10000 元 1 年期定期存款，年利率 3%, 约定到期自动转存（包括利息）。问：

   (1) 10 年后本息共多少元？

   (2) 需要存多少年这 10000 元才能增值到 20000 元？

2. 成语说："智者千虑，必有一失；愚者千虑，必有一得"。设智者作判断的准确率为 $p_1=0.99$, 愚者作判断的准确率为 $p_2=0.01$，计算智者做 1000 次独立的判断至少犯一次错误的概率，与愚者做 1000 次独立判断至少对一次的概率。

2.3 数学函数

2.3.1 数学函数——平方根、指数、对数

例:

```
sqrt(6.25)
## [1] 2.5
exp(1)
## [1] 2.718282
log10(10000)
## [1] 4
```

`sqrt(6.25)` 表示 $\sqrt{6.25}$，结果为 2.5。`exp(1)` 表示 e^1，结果为 $e=2.718282$。`log10(10000)` 表示 $\lg 10000$，结果为 4。`log` 为自然对数。

2.3.2 数学函数——取整

例:

```
round(1.1234, 2)
## [1] 1.12
round(-1.9876, 2)
## [1] -1.99
floor(1.1234)
## [1] 1
floor(-1.1234)
## [1] -2
ceiling(1.1234)
## [1] 2
ceiling(-1.1234)
## [1] -1
```

`round(1.1234, 2)` 表示把 1.1234 四舍五入到两位小数。`floor(1.1234)` 表示把 1.1234 向下取整，结果为 1。`ceiling(1.1234)` 表示把 1.1234 向上取整，结果为 2。

2.3.3 数学函数——三角函数

例:

```
pi
## [1] 3.141593
sin(pi/6)
## [1] 0.5
cos(pi/6)
## [1] 0.8660254
sqrt(3)/2
## [1] 0.8660254
tan(pi/6)
## [1] 0.5773503
```

`pi` 表示圆周率 π。`sin` 正弦, `cos` 余弦, `tan` 正切, 自变量以弧度为单位。`pi/6` 是 30°。

2.3.4 数学函数——反三角函数

例:

```
pi/6
## [1] 0.5235988
asin(0.5)
```

```
## [1] 0.5235988
acos(sqrt(3)/2)
## [1] 0.5235988
atan(sqrt(3)/3)
## [1] 0.5235988
```

asin 反正弦, acos 反余弦, atan 反正切，结果以弧度为单位。

2.3.5 分布函数和分位数函数

例:

```
dnorm(1.98)
## [1] 0.05618314
pnorm(1.98)
## [1] 0.9761482
qnorm(0.975)
## [1] 1.959964
```

dnorm(x) 表示标准正态分布密度 $\phi(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^2}$. pnorm(x) 表示标准正态分布函数 $\Phi(x) = \int_{-\infty}^{x}\phi(t)\,dt$。qnorm(y) 表示标准正态分布分位数函数 $\Phi^{-1}(x)$。还有其它许多分布的密度函数、分布函数和分位数函数。例如，

```
qt(1 - 0.05/2, 10)
## [1] 2.228139
```

求自由度为 10 的 t 检验的双侧临界值。其中 qt(y,df) 表示自由度为 df 的 t 分布的分位数函数。

为了获得更详细的与分布有关的函数的帮助信息，可以在命令行中用?Distributions 命令打开相应的帮助。

2.4 函数调用和自定义函数

R 中调用函数，与调用数学函数的规则相近，如 sqrt(1.44)。有些函数允许有可选参数，比如，round(1.2345) 结果是四舍五入取整，加选项 round(1.2345, digits=2) 结果则变成四舍五入到两位小数精度。

复合函数的调用也与数学中用法类似，如 log(exp(1.0)) 表示 $\ln e^{1.0}$：

```
log(exp(1.0))
```

```
## [1] 1
```

另外，R 还提供了另外一种复合函数的调用格式：

```
exp(1.0) |> log()
```

```
## [1] 1
```

其中的运算符 `|>` 称为管道运算符，这种语法在复合函数有多层调用时更简明易读，将每次调用函数看成对自变量的加工，加工完以后通过管道传送给下一道工序加工，如 `f3(f2(f1(x)))` 写成 `x |> f1() |> f2() |> f3()` 就更容易看清楚数据的流向。

R 支持自定义函数，比如，将一个复杂的计算表达式封装在函数定义中：

```
frat <- function(x){
  (1 + x + 2*x^2)/(1 + 3*x + 2*x^2 + x^3)
}
```

定义中最后一个语句的结果作为函数输出，也可以用 `return(y)` 输出变量 `y` 的值。

自定义函数的好处是一次定义可以反复使用，而且函数内用的变量不会与其它变量造成混杂干扰。

2.5 输出

2.5.1 简单输出

命令行的计算结果直接显示在命令的后面。在用 `source()` 运行程序文件时，需要用 `print()` 函数显示一个表达式的结果，如：

```
print(sin(pi/2))
## [1] 1
```

用 `cat()` 函数显示多项内容，包括数值和文本，文本包在两个单撇号或两个双撇号中，如:

```
cat("sin(pi/2)=", sin(pi/2), "\n")
## sin(pi/2)= 1
```

`cat()` 函数最后一项一般是`"\n"`, 表示换行。忽略此项将不换行。

再次提示：要避免打开中文输入法导致误使用中文标点。

2.5.2 用 sink() 函数作运行记录

R 使用经常是在命令行逐行输入命令（程序），结果紧接着显示在命令后面。如何保存这些命令和显示结果？在 R 命令行中运行过的命令会被保存在运行的工作文件夹中的一个名为.Rhistory 的文件中。用 sink() 函数打开一个文本文件开始记录文本型输出结果。结束记录时用空的 sink() 即可关闭文件不再记录。如

```
sink("tmpres01.txt", split=TRUE)
print(sin(pi/6))
print(cos(pi/6))
cat("t(10) 的双侧 0.05 分位数（临界值）=", qt(1 - 0.05/2, 10), "\n")
sink()
```

sink() 用作输出记录主要是在测试运行中使用，正常的输出应该使用 cat() 函数、write.table()、write.csv() 等函数。

2.5.3 练习

1. 用 cat() 函数显示

   ```
   log10(2)=*** log10(5)=***
   ```

   其中 *** 应该代以实际函数值。

2. 用 sink() 函数开始把运行过程记录到文件 “log001.txt” 中，在命令行试验几个命令，然后关闭运行记录，查看生成的 “log001.txt” 的内容。

2.6 向量计算与变量赋值

R 语言以向量为最小单位。用 <-赋值。如

```
x1 <- 1:10
x1
##  [1]  1  2  3  4  5  6  7  8  9 10
```

一般的向量可以用 c() 生成，如

```
marks <- c(3, 5, 10, 5, 6)
```

在程序语言中，变量用来保存输入的值或计算的结果。变量可以存放各种不同类型的值，如单个数值、多个数值（称为向量）、单个字符串、多个字符串（称为字符型向量），等等。单个数值称为**标量**。

技术细节：用程序设计语言的术语描述，R 语言是动态类型的，其变量的类型不需要预先声明，运行过程中允许变量类型改变，实际上变量赋值是一种“绑定”（binding），将一个变量的名称（变量名）与实际的一个存储位置联系在一起。在命令行定义的变量称为**全局变量**。

用 `print()` 函数显示向量或在命令行中显示向量时，每行显示的行首会有方括号和数字序号，代表该行显示的第一个向量元素的下标。如

```
12345678901:12345678920
##  [1] 12345678901 12345678902 12345678903 12345678904 12345678905
##  [6] 12345678906 12345678907 12345678908 12345678909 12345678910
## [11] 12345678911 12345678912 12345678913 12345678914 12345678915
## [16] 12345678916 12345678917 12345678918 12345678919 12345678920
```

向量可以和一个标量作四则运算，结果是每个元素都和这个标量作四则运算，如：

```
x1 + 200
## [1] 201 202 203 204 205 206 207 208 209 210
2*x1
## [1]  2  4  6  8 10 12 14 16 18 20
2520/x1
## [1] 2520 1260  840  630  504  420  360  315  280  252
```

两个等长的向量可以进行四则运算，相当于对应元素进行四则运算，如

```
x2 <- x1 * 3
x2
## [1]  3  6  9 12 15 18 21 24 27 30
x2 - x1
## [1]  2  4  6  8 10 12 14 16 18 20
```

R 的许多函数都可以用向量作为自变量，结果是自变量的每个元素各自的函数值。如

```
sqrt(x1)
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751
## [8] 2.828427 3.000000 3.162278
```

结果是 1 到 10 的整数各自的平方根。

2.7 工作空间介绍

在命令行中定义的变量，在退出 R 时，会提问是否保存工作空间，初学时可选择保存，真正用 R 进行数据分析时往往不保存工作空间。如果退出时保存了工作空间，再次启动 R 后，能够使用以前定义的各个变量以及自定义函数。

在使用 R 的官方版本时，如果在 MS Windows 中使用，一般把不同的数据分析项目放在不同的文件夹中。将 R 的程序快捷图标复制到每一个项目的文件夹中，并用右键菜单讲快捷图标的 “属性” 中 “起始位置” 改为空白。要分析哪一个项目的数据，就从那个项目文件夹中的 R 快捷图标启动，这样可以保证不同的项目有不同的工作空间。

如果使用 RStudio 软件，也需要把不同项目放在不同文件夹，并且每个项目在 RStudio 中单独建立一个 “项目”（project）。要分析那个项目的数据，就打开那个项目。不同项目使用不同的工作空间。

RStudio 中的 “Environment” 窗格会显示当前已定义的 R 变量与函数。

2.7.1 练习

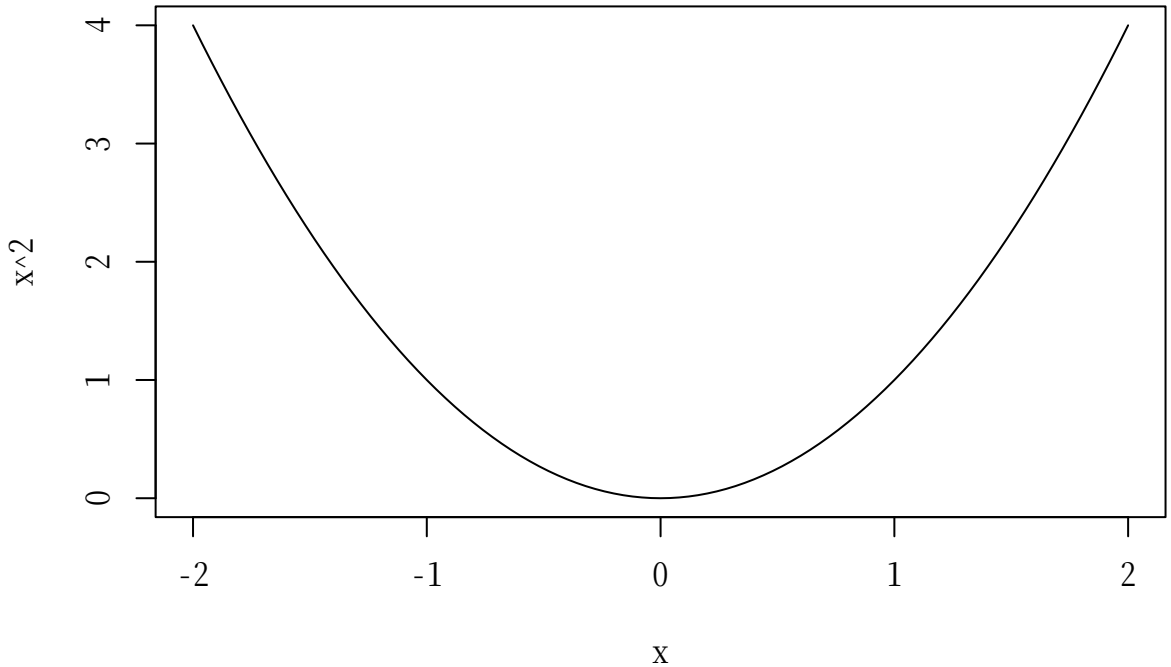
1. 某人存入 10000 元 1 年期定期存款，年利率 3%, 约定到期自动转存（包括利息）。列出 1、2、……、10 年后的本息金额。
2. 显示 2 的 $1, 2, \ldots, 20$ 次方。
3. 定义 x1 为 1 到 10 的向量，定义 x2 为 x1 的 3 倍，然后退出 R，再次启动 R，查看 x1 和 x2 的值。

2.8 绘图示例

2.8.1 函数曲线示例

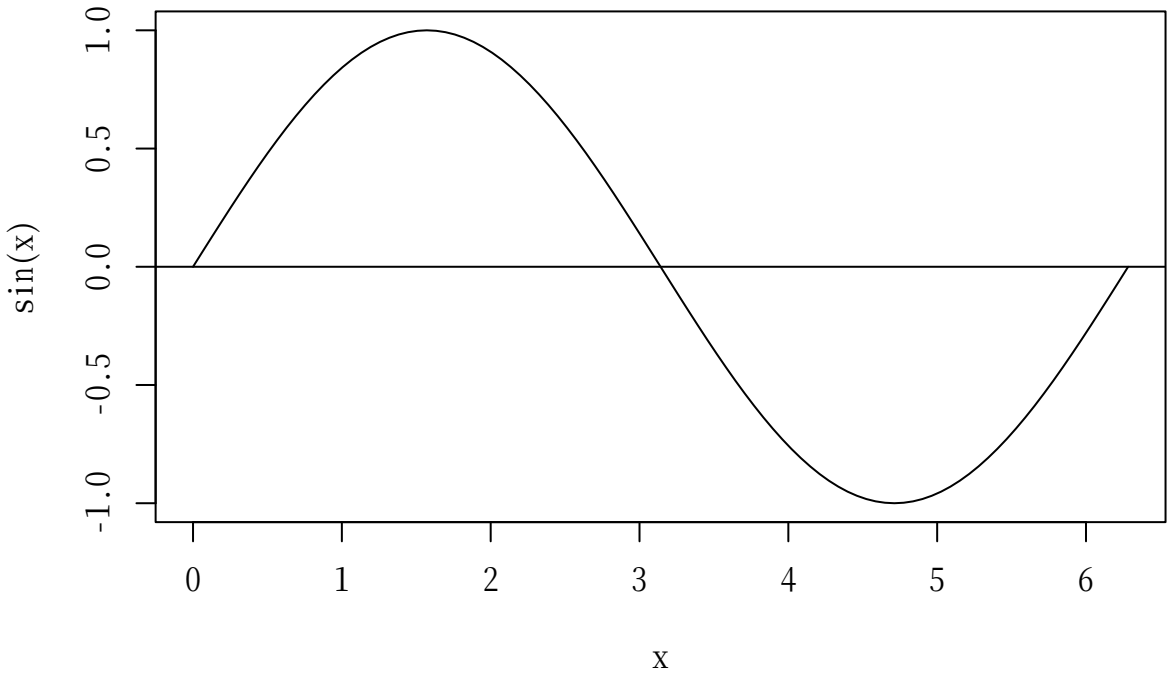
如下程序用 `curve()` 函数制作 $y = x^2$ 函数的曲线图，`curve()` 函数第二、第三自变量是绘图区间：

```
curve(x^2, -2, 2)
```



类似地，$\sin(x)$ 函数曲线图用如下程序可制作, 用 `abline()` 函数添加参考线:

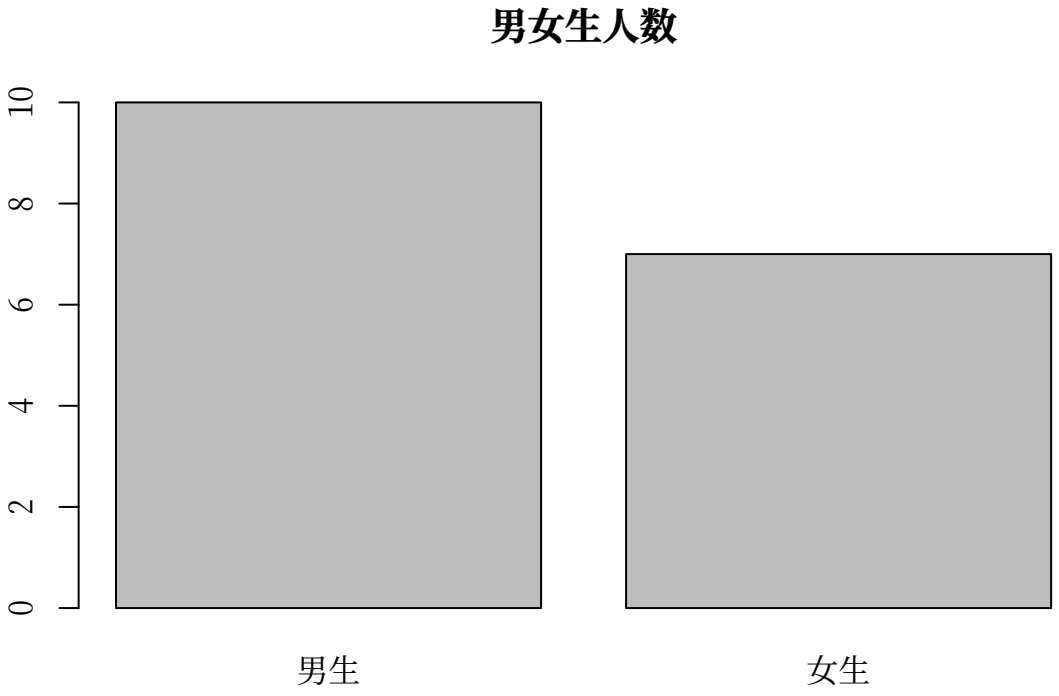
```
curve(sin(x), 0, 2*pi)
abline(h=0)
```



2.8.2 条形图示例

假设有 10 个男生，7 个女生，如下程序绘制男生、女生人数的条形图：

```
barplot(c(" 男生"=10, " 女生"=7),
    main=" 男女生人数")
```

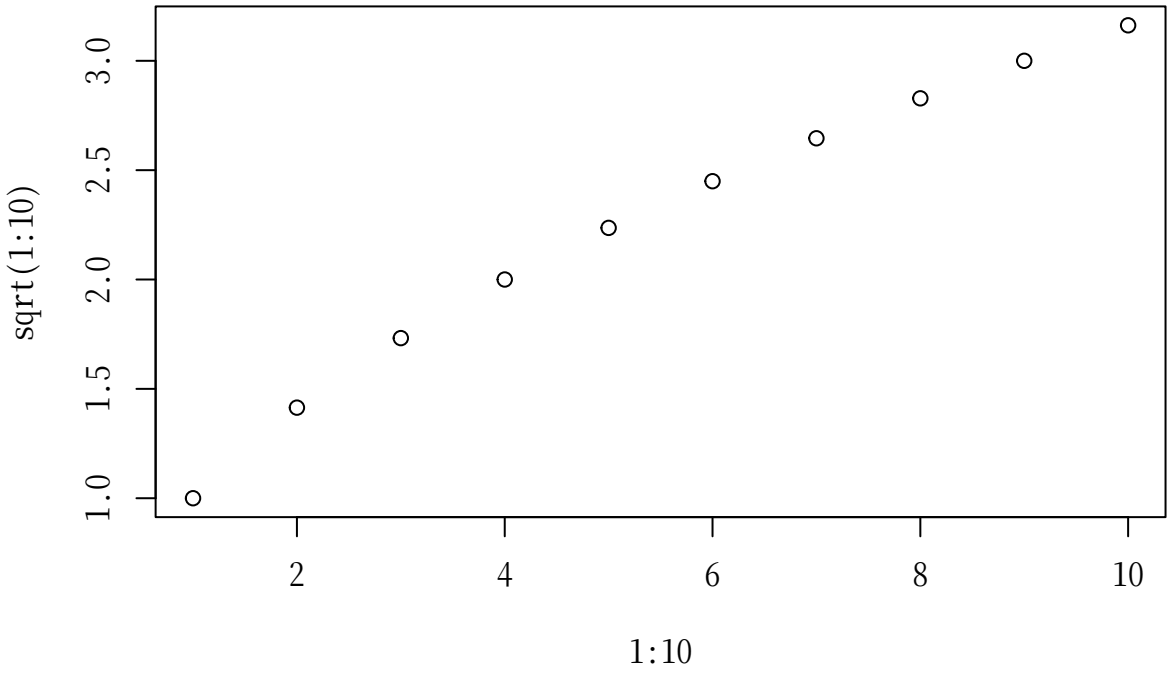


利用适当选项可以人为定制颜色、控制条形宽窄。实际问题中，个数（频数）一般是从数据中统计得到的。

2.8.3 散点图示例

下面的例子用 `plot()` 函数做了散点图, `plot()` 函数第一个自变量是各个点的横坐标值，第二个自变量是对应的纵坐标值。

```
plot(1:10, sqrt(1:10))
```



2.8.4 R 软件自带的图形示例

R 软件中自带了一些演示图形。通过如下程序可以调用：

```
demo("graphics")
demo("image")
```

2.8.5 练习

1. 画 $\exp(x)$ 在 $(-2, 2)$ 区间的函数图形。
2. 画 $\ln(x)$ 在 $(0.01, 10)$ 区间的函数图形。

2.9 汇总统计示例

2.9.1 表格数据

统计用的输入数据典型样式是 Excel 表那样的表格数据。表格数据特点：每一列应该是相同的类型（或者都是数值，或者都是文字，或者都是日期），每一列应该有一个名字。

这样的表格数据，一般可以保存为.csv 格式：数据项之间用逗号分开，文件本身是文本型的，可以用普通记事本程序查看和编辑。Excel 表可以用 “另存为” 命令保存为.csv 格式。常用的数据库管理系统一般也可以把表保存为.csv 格式。

2.9.2 读入表格数据

例如，`taxsamp.csv`是一个 csv 格式表格数据文件，可以用 Excel 打开，也可以用记事本程序或 notepad++ 打开。内容见本页面后面的附录。

用如下程序可以把.csv 文件读入到 R 中，并显示开头几行的内容：

```
library(readr)
tax.tab <- read_csv("data/taxsamp.csv",
  locale=locale(encoding="GB18030"))
print(head(tax.tab))
```

这里用了选项来说明输入文件的中文编码用了 GB18030 编码，而不是默认的 UTF-8 编码。

2.9.3 练习

1. 用 Excel 软件查看 “taxsamp.csv” 的内容 (双击即可)。
2. 用记事本程序或 notepad++ 软件查看 “taxsamp.csv” 的内容。
3. 读入 “taxsamp.csv” 到 R 数据框 tax.tab 中，查看 tax.tab 内容。

2.9.4 分类变量频数统计

在 tax.tab 中，“征收方式” 是一个分类变量。用 `table()` 函数计算每个不同值的个数，称为**频数** (frequency):

```
table(tax.tab[[" 征收方式"]])
```

```
##
##     查帐征收 定期定额征收 定期定率征收
##           31           16            2
```

类似地可以统计 “申报渠道” 的取值频数:

```
table(tax.tab[[" 申报渠道"]])
```

```
##
## 大厅申报 网上申报
##       18       31
```

也可以用 table() 函数统计“征收方式”和“申报渠道”交叉分类频数，如：

```
table(tax.tab[[" 征收方式"]], tax.tab[[" 申报渠道"]])
```

```
##
##                 大厅申报 网上申报
##   查帐征收            9       22
##   定期定额征收        9        7
##   定期定率征收        0        2
```

上述结果制表如下:

```
knitr::kable(table(tax.tab[[" 征收方式"]], tax.tab[[" 申报渠道"]]) )
```

	大厅申报	网上申报
查帐征收	9	22
定期定额征收	9	7
定期定率征收	0	2

2.9.5 数值型变量的统计

数值型变量可以计算各种不同的统计量, 如平均值、标准差和各个分位数。summary() 可以给出最小值、最大值、中位数、四分之一分位数、四分之三分位数和平均值。如

```
summary(tax.tab[[" 营业额"]])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       0     650    2130  247327    9421 6048000
```

中位数是从小到大排序后排在中间的值。四分之一和四分之三分位数类似。

统计函数以一个数值型向量为自变量，包括 sum(求和), mean(平均值), var(样本方差), sd(样本标准差), min(最小值), max(最大值), range(最小值和最大值) 等。如

```
mean(tax.tab[[" 营业额"]])
```

```
## [1] 247327.4
```

```
sd(tax.tab[[" 营业额"]])
```

```
## [1] 1036453
```

如果数据中有缺失值，可以删去缺失值后计算统计量，这时在 mean, sd 等函数中加入 na.rm=TRUE 选项。

2.9.6 练习

用如下程序定义一个变量 x, 然后求 x 的平均值和最小值、最大值。

```
x <- c(3, 5, 10, 5, 6)
```

2.10 运行源程序文件

用 `source()` 函数可以运行保存在一个文本文件中的源程序。比如，如下内容保存在文件 `ssq.r` 中：

```
sum.of.squares <- function(x){
  sum(x^2)
}
```

用如下 `source()` 命令运行：

```
source("ssq.r")
```

运行后就可以调用自定义函数 `sum.of.squares()` 了。

2.10.1 源文件编码

源程序文件存在编码问题。对于源程序编码与系统默认编码不同的情况，在 `source()` 函数中可以添加 `encoding=` 选项。例如，保存为 UTF-8 编码的源程序在简体中文 MS Windows 系统的 R 中运行，可以在 `source()` 函数中可以添加 `encoding="UTF-8"` 选项。保存为 GBK 编码的源程序文件在 MAC 系统的 R 中运行，可以在 `source()` 函数中可以添加 `encoding="GBK"` 选项。

在 RStudio 中，可以打开一个源程序文件查看与编辑。用快捷键 “Ctrl+Enter” 或快捷图标 “Run” 可以运行当前行或者加亮选中行，快捷图标 “Source” 可以运行整个文件。如果发现中文乱码，可以用菜单 “Reopen with encoding” 选择合适的编码打开，用菜单 “Save with encoding” 选择需要的编码保存。

2.10.2 当前工作目录

在用 `source()` 调用源程序文件或者用 `read.csv()` 读入数据文件时，如果不写文件名的全路径，就认为文件位置是在所谓 “当前工作目录”。用 `getwd()` 函数可以查询当前工作目录，用 `setwd()` 函数可以设置当前工作目录。在 RStudio 中用菜单 “Session–Set working directory” 设置当前工作目录。

在 MS Windows 操作系统中使用 R 软件时，一种好的做法是把某个研究项目所有数据和程序放在某个文件夹如 `c:\work` 中，把 R 的程序快捷图标复制到该目录中，在资源管理器中对该图标调用鼠标右键菜单 “属性”，从弹出对话框中，把 “起始位置” 一栏清除。这样，每次从这个快捷图标启动 R，就可以自动以所在子目录为当前工作目录，工作空间和命令历史记录也默认存放在这里。

在 MS Windows 操作系统的 R 中使用文件路径时，要用正斜杠作为连接符，使用反斜杠则需要成对使用，如 `setwd("d:/work")` 或 `setwd("d:\\work")`。

如果使用 RStudio 软件，将某个研究项目所有数据和程序放在某个文件夹中，然后建立一个新项目（project）指向该文件夹。

2.10.3 练习

编辑生成 `ssq.r` 源程序文件并用 `source()` 函数运行，然后计算：

```
sum.of.squares(1:5)
```

2.11 附录：数据

2.11.1 公司纳税数据样例

本数据是某地区 2013 年 12 月所属税款申报信息的一个子集，仅含 20 个公司的数据。

数据文件显示

Part II

数据类型与相应的运算

Chapter 3

常量与变量

3.1 常量

常量是指直接写在程序中的值，包括数值、字符串等。

数值型常量包括整型、单精度、双精度等，一般不需要区分。写法如 `123`, `123.45`, `-123.45`, `-0.012`, `1.23E2`, `-1.2E-2` 等。为了表示 `123` 是整型，可以写成 `123L`。

字符型常量用两个双撇号或两个单撇号包围，如`"Li Ming"` 或`'Li Ming'`。字符型支持中文，如`" 李明"` 或`'李明'`。国内的中文编码主要有 GBK 编码和 UTF-8 编码，有时会遇到编码错误造成乱码的问题，RStudio 软件默认采用 UTF-8 编码。在 R 程序运行时字符串一般用 UTF-8 编码保存。R 语言没有“字符”类型，只有字符串类型。

逻辑型常量只有 TRUE 和 FALSE，表示真值和假值。

缺失值用 NA 表示。统计计算中经常会遇到缺失值，表示记录丢失、因为错误而不能用、节假日没有数据等。除了数值型，逻辑型和字符型也可以有缺失值，而且字符型的空白值不会自动辨识为缺失值，需要自己规定。

R 支持复数类型，复数常量写法如 `2.2 + 3.5i`, `1i` 等。

3.2 变量

程序语言中的变量用来保存输入的值或者计算得到的值。在 R 中，变量可以保存所有的数据类型，比如标量、向量、矩阵、数据框、函数等。

变量都有变量名，R 变量名必须以字母、数字、下划线和句点组成，变量名的第一个字符不能取为数字。在中文环境下，汉字也可以作为变量名的合法字符使用。变量名是区分大小写的，`y` 和 `Y` 是两个不同的变量名。

变量名举例: `x`, `x1`, `X`, `cancer.tab`, `clean_data`, `diseaseData`。

用 `<-`赋值的方法定义变量。`<-`也可以写成 `=`，但是 `<-`更直观。如

```
x5 <- 6.25
x6 = sqrt(x5)
```

R 的变量没有固定的类型，给已有变量赋值为新的类型，该变量就变成新的类型，但一般应避免这样的行为。R 是“动态类型”语言，赋值实际上是“绑定”（binding），即将一个变量名与一个存储地址联系在一起，同一个存储地址可以有多个变量名与其联系。

3.3 R 数据类型

R 语言基本的数据类型有数值，逻辑型（TRUE, FALSE），文本（字符串）。支持缺失值，有专门的复数类型。

R 语言数据结构包括向量，矩阵和数据框，多维数组，列表，对象等。数据中元素、行、列还可以用名字访问。最基本的是向量类型。向量类型数据的访问方式也是其他数据类型访问方式的基础。

Chapter 4

数值型向量及其运算

4.1 数值型向量

向量是将若干个基础类型相同的值存储在一起，各个元素可以按序号访问。如果将若干个数值存储在一起可以用序号访问，就叫做一个数值型向量。

用 `c()` 函数把多个元素或向量组合成一个向量。如

```
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
```

```
## [1] 1 2 3 4
```

10:13 这样的写法表示从 10 到 13 的整数组成的向量。

用 `print()` 函数显示向量或在命令行中显示向量时，每行显示的行首会有方括号和数字序号，代表该行显示的第一个向量元素的下标。如

```
12345678901:12345678920
```

```
##  [1] 12345678901 12345678902 12345678903 12345678904 12345678905 12345678906
##  [7] 12345678907 12345678908 12345678909 12345678910 12345678911 12345678912
## [13] 12345678913 12345678914 12345678915 12345678916 12345678917 12345678918
## [19] 12345678919 12345678920
```

`length(x)` 可以求 x 的长度。长度为零的向量表示为 `numeric(0)`。`numeric()` 函数可以用来初始化一个指定元素个数而元素都等于零的数值型向量，如 `numeric(10)` 会生成元素为 10 个零的向量。

4.2 向量运算

4.2.1 标量和标量运算

单个数值称为**标量**，R 没有单独的标量类型，标量实际是长度为 1 的向量。

R 中四则运算用 `+ - * / ^` 表示 (加、减、乘、除、乘方)，如

```
1.5 + 2.3 - 0.6 + 2.1*1.2 - 1.5/0.5 + 2^3
```

```
## [1] 10.72
```

R 中四则运算仍遵从通常的优先级规则，可以用圆括号 `()` 改变运算的先后次序。如

```
1.5 + 2.3 - (0.6 + 2.1)*1.2 - 1.5/0.5 + 2^3
```

```
## [1] 5.56
```

除了加、减、乘、除、乘方，R 还支持整除运算和求余运算。用`%/%` 表示整除，用`%%` 表示求余。如

```
5 %/% 3
## [1] 1
5 %% 3
## [1] 2
5.1 %/% 2.5
## [1] 2
5.1 %% 2.5
## [1] 0.1
```

4.2.2 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, 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
```

一个向量乘以一个标量，就是线性代数中的数乘运算。

四则运算时如果有缺失值，缺失元素参加的运算相应结果元素仍缺失。如

```
c(1, NA, 3) + 10
## [1] 11 NA 13
```

4.2.3 等长向量运算

等长向量的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
```

两个等长向量的加、减运算就是线性代数中两个向量的加、减运算。

4.2.4 不等长向量的运算

两个不等长向量的四则运算，如果其长度为倍数关系，规则是每次从头重复利用短的一个。如

```
x1 <- c(10, 20)
x2 <- c(1, 3, 5, 7)
x1 + x2
## [1] 11 23 15 27
```

```
x1 * x2
## [1]  10  60  50 140
```

不仅是四则运算，R 中有两个或多个向量按照元素一一对应参与某种运算或函数调用时，如果向量长度不同，一般都采用这样的规则。

如果两个向量的长度不是倍数关系，会给出警告信息。如

```
c(1,2) + c(1,2,3)
## Warning in c(1, 2) + c(1, 2, 3): longer object length is not a multiple of
## shorter object length
## [1] 2 4 4
```

4.3 向量函数

4.3.1 向量化的函数

R 中的函数一般都是向量化的: 在 R 中，如果普通的一元函数以向量为自变量，一般会对每个元素计算。这样的函数包括 sqrt, log10, log, exp, sin, cos, tan 等许多。如

```
sqrt(c(1, 4, 6.25))
## [1] 1.0 2.0 2.5
```

为了查看这些基础的数学函数的列表，运行命令 `help.start()`，点击链接 “Search Engine and Keywords”，找到 “Mathematics” 栏目，浏览其中的 “arith” 和 “math” 链接中的说明。常用的数学函数有：

- 舍入：`ceiling`, `floor`, `round`, `signif`, `trunc`, `zapsmall`
- 符号函数 `sign`
- 绝对值 `abs`
- 平方根 `sqrt`
- 对数与指数函数 `log`, `exp`, `log10`, `log2`
- 三角函数 `sin`, `cos`, `tan`
- 反三角函数 `asin`, `acos`, `atan`, `atan2`
- 双曲函数 `sinh`, `cosh`, `tanh`
- 反双曲函数 `asinh`, `acosh`, `atanh`

有一些不太常用的数学函数：

- 贝塔函数 `beta`, `lbeta`
- 伽玛函数 `gamma`, `lgamma`, `digamma`, `trigamma`, `tetragamma`, `pentagamma`

- 组合数 choose, lchoose
- 富利叶变换和卷积 fft, mvfft, convolve
- 正交多项式 poly
- 求根 polyroot, uniroot
- 最优化 optimize, optim
- Bessel 函数 besselI, besselK, besselJ, besselY
- 样条插值 spline, splinefun
- 简单的微分 deriv

如果自己编写的函数没有考虑向量化问题，可以用 Vectorize() 函数将其转换成向量化版本。

4.3.2 排序函数

sort(x) 返回排序结果。rev(x) 返回把各元素排列次序反转后的结果。order(x) 返回排序用的下标。如

```
x <- c(33, 55, 11)
sort(x)
## [1] 11 33 55
rev(sort(x))
## [1] 55 33 11
order(x)
## [1] 3 1 2
x[order(x)]
## [1] 11 33 55
```

例子中，order(x) 结果中 3 是 x 的最小元素 11 所在的位置下标，1 是 x 的第二小元素 33 所在的位置下标，2 是 x 的最大元素 55 所在的位置下标。

4.3.3 统计函数

sum(求和), mean(求平均值), var(求样本方差), sd(求样本标准差), min(求最小值), max(求最大值), range(求最小值和最大值) 等函数称为统计函数，把输入向量看作样本，计算样本统计量。prod 求所有元素的乘积。

cumsum 和 cumprod 计算累加和累乘积。如

```
cumsum(1:5)
## [1]  1  3  6 10 15
cumprod(1:5)
## [1]   1   2   6  24 120
```

其它一些类似函数有 pmax, pmin, cummax, cummin 等。

4.3.4 生成规则序列的函数

seq 函数是冒号运算符的推广。比如，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.out=100) 产生从 0 到 2π 的等间隔序列，序列长度指定为 100。

从这些例子可以看出，S 函数可以带自变量名调用。每个函数的变量名和用法可以查询其帮助信息，在命令行界面用 "? 函数名" 的方法查询。在使用变量名时次序可以颠倒，比如 seq(to=5, from=2)} 仍等同于 2:5。

rep() 函数用来产生重复数值。为了产生一个初值为零的长度为 n 的向量，用 x <- rep(0, n)。rep(c(1,3), 2) 把第一个自变量重复两次，结果相当于 c(1,3,1,3)。

rep(c(1,3), c(2,4)) 则需要利用 R 的一般向量化规则，把第一自变量的第一个元素 1 按照第二自变量中第一个元素 2 的次数重复，把第一自变量中第二个元素 3 按照第二自变量中第二个元素 4 的次数重复，结果相当于 c(1,1,3,3,3,3)。

如果希望重复完一个元素后再重复另一元素，用 each= 选项，比如 rep(c(1,3), each=2) 结果相当于 c(1,1,3,3)。

有一点技术性的小问题：1:5 和 seq(5) 的结果是整型（integer）的，c(1,3,5) 和 seq(1, 5, by=2) 的结果是浮点型（double）的。

4.4 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(real = c(1,0,-1,0), imaginary = c(0,1,0,-1)) 相当于 c(1+0i, 1i, -1+0i, -1i)。

在 complex() 中可以用 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。

sqrt, log, exp, sin 等函数对复数也有定义，但是函数定义域在自变量为实数时可能有限制而复数无限制，这时需要区分自变量类型。如

```
sqrt(-1)
## [1] NaN
## Warning message:
## In sqrt(-1) : NaNs produced
sqrt(-1 + 0i)
## [1] 0+1i
```

4.5 练习

1. 显示 1 到 100 的整数的平方根和立方根（提示：立方根就是三分之一次方）。

2. 设有 10 个人的小测验成绩为:

$$77\ 60\ 91\ 73\ 85\ 82\ 35\ 100\ 66\ 75$$

   (1) 把这 10 个成绩存入变量 x;

   (2) 从小到大排序；

   (3) 计算 `order(x)`，解释 `order(x)` 结果中第 3 项代表的意义。

   (4) 计算这些成绩的平均值、标准差、最小值、最大值、中位数。

3. 生成 $[0,1]$ 区间上等间隔的 100 个格子点存入变量 x 中。

Chapter 5

逻辑型向量及其运算

5.1 逻辑型向量与比较运算

逻辑型是 R 的基本数据类型之一，只有两个值 TRUE 和 FALSE, 缺失时为 NA。逻辑值一般产生自比较，如

```
sele <- (log10(15) < 2); print(sele)
## [1] TRUE
```

向量比较结果为逻辑型向量。如

```
c(1, 3, 5) > 2
## [1] FALSE  TRUE  TRUE
(1:4) >= (4:1)
## [1] FALSE FALSE  TRUE  TRUE
```

从例子可以看出，向量比较也遵从 R 的向量间运算的一般规则：向量与标量的运算是向量每个元素与标量都分别运算一次，等长向量的运算时对应元素的运算，不等长但长度为倍数关系的向量运算是把短的从头重复利用。

与 NA 比较产生 NA，如

```
c(1, NA, 3) > 2
## [1] FALSE    NA  TRUE
NA == NA
## [1] NA
```

为了判断向量每个元素是否 NA，用 `is.na()` 函数，如

```
is.na(c(1, NA, 3) > 2)
## [1] FALSE  TRUE FALSE
```

用 `is.finite()` 判断向量每个元素是否 Inf 值。

比较运算符包括

```
<   <=  >  >=  ==  !=  %in%
```

分别表示小于、小于等于、大于、大于等于、等于、不等于、属于。要注意等于比较用了两个等号。

`%in%` 是比较特殊的比较，`x %in% y` 的运算把向量 y 看成集合，运算结果是一个逻辑型向量，第 i 个元素的值为 x 的第 i 元素是否属于 y 的逻辑型值。如

```
c(1,3) %in% c(2,3,4)
## [1] FALSE  TRUE
c(NA,3) %in% c(2,3,4)
## [1] FALSE  TRUE
c(1,3) %in% c(NA, 3, 4)
## [1] FALSE  TRUE
c(NA,3) %in% c(NA, 3, 4)
## [1] TRUE TRUE
```

函数 `match(x, y)` 起到和 `x %in% y` 运算类似的作用，但是其返回结果不是找到与否，而是对 `x` 的每个元素，找到其在 `y` 中首次出现的下标，找不到时取缺失值，如

```
match(c(1, 3), c(2,3,4,3))
## [1] NA  2
```

5.2 逻辑运算

为了表达如 "$x > 0$ 而且 $x < 1$", "$x \leq 0$ 或者 $x \geq 1$" 之类的复合比较，需要使用逻辑运算把两个比较连接起来。逻辑运算符为 `&`, `|` 和`!`, 分别表示 "同时成立"、"两者至少其一成立"、"条件的反面"。比如，设 `age<=3` 表示婴儿，`sex=='女'`表示女性，则 `age<=3 & sex=='女'`表示女婴, `age<=3 | sex=='女'`表示婴儿或妇女, `!(age<=3 | sex=='女')` 表示既非婴儿也非妇女。为了确定运算的先后次序可以用圆括号 `()` 指定。

用 `xor(x, y)` 表示 `x` 与 `y` 的异或运算，即值不相等时为真值，相等时为假值，有缺失值参加运算时为缺失值。

逻辑向量与逻辑标量之间的逻辑运算，两个逻辑向量之间的逻辑运算规则遵从一般 R 向量间运算规则。

在右运算符是缺失值时，如果左运算符能够确定结果真假，可以得到非缺失的结果。例如，`TRUE | NA` 为 `TRUE`, `FALSE & NA` 为 `FALSE`。不能确定结果时返回 `NA`，比如，`TRUE & NA` 为 `NA`, `FALSE | NA` 为 `NA`。

&& 和 || 分别为短路的标量逻辑与和短路的标量逻辑或，仅对两个标量进行运算，如果有向量也仅使用第一个元素。一般用在 if 语句、while 语句中，且只要第一个比较已经决定最终结果就不计算第二个比较。例如

```
if(TRUE || sqrt(-1)>0) next
```

其中的 sqrt(-1) 部分不会执行。其中条件的结果为 TRUE, 第二部分没有参加计算，否则第二部分的计算会发生函数自变量范围错误。

5.3 逻辑运算函数

因为 R 中比较与逻辑运算都支持向量之间、向量与标量之间的运算，所以在需要一个标量结果时要特别注意，后面讲到的 if 结构、while 结构都需要逻辑标量而且不能是缺失值。这时，应该对缺失值结果单独考虑。

若 cond 是逻辑向量，用 all(cond) 测试 cond 的所有元素为真；用 any(cond) 测试 cond 至少一个元素为真。cond 中允许有缺失值，结果可能为缺失值。如

```
c(1, NA, 3) > 2
## [1] FALSE    NA  TRUE
all(c(1, NA, 3) > 2)
## [1] FALSE
any(c(1, NA, 3) > 2)
## [1] TRUE
all(NA)
## [1] NA
any(NA)
## [1] NA
```

函数 which() 返回真值对应的所有下标，如

```
which(c(FALSE, TRUE, TRUE, FALSE, NA))
## [1] 2 3
which((11:15) > 12)
## [1] 3 4 5
```

函数 identical(x,y) 比较两个 R 对象 x 与 y 的内容是否完全相同，结果只会取标量 TRUE 与 FALSE 两种。如

```
identical(c(1,2,3), c(1,2,NA))
## [1] FALSE
identical(c(1L,2L,3L), c(1,2,3))
## [1] FALSE
```

其中第二个结果假值是因为前一向量是整数型，后一向量是实数型。

函数 `all.equal()` 与 `identical()` 类似，但是在比较数值型时不区分整数型与实数型，而且相同时返回标量 TRUE，但是不同时会返回一个说明有何不同的字符串。如

```
all.equal(c(1,2,3), c(1,2,NA))
## [1] "'is.NA' value mismatch: 1 in current 0 in target"
all.equal(c(1L,2L,3L), c(1,2,3))
## [1] TRUE
```

函数 `duplicated()` 返回每个元素是否为重复值的结果，如：

```
duplicated(c(1,2,1,3,NA,4,NA))
## [1] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE
```

用函数 `unique()` 可以返回去掉重复值的结果。

Chapter 6

字符型数据及其处理

6.1 字符型向量

字符型向量是元素为字符串的向量。字符串在程序中写成用两个双撇号包围或者用两个单撇号包围的内容。如

```
s1 <- c('abc', '', 'a cat', NA, '李明')
```

注意空字符串并不能自动认为是缺失值，字符型的缺失值仍用 `NA` 表示。

字符串内容一般从文件、网络、数据库获得，在程序中直接用双撇号或者单撇号写出只是输入字符串的办法之一。

6.2 转义字符和原始字符串 {p-t-char-raw}

为了在字符串中表示一个双撇号，可以用反斜杠在前面标明，称为 “转义”，如：

```
cat("\"\n")
```

```
## "
```

其中`"\n"` 也是转义的字符，表示换行。

当需要转义的内容较多时，可以使用原始字符串 (raw string)，方法是用 `r"(...)"` 的格式，其中`...` 是实际内容。如：

```
cat(r"(C:\disk\course\math\nFinished!\n)")
```

```
## C:\disk\course\math\nFinished!\n
```

注意其中\n 也被当作普通字符解释了，不再当作换行符。

原始字符串如果内容中包含了圆括号，可以将边界的圆括号改为方括号 `[]` 或者大括号 `{}`。如果这样也不能避免歧义，可以在开始和结尾加上相同个数的减号，格式为 `r"--(...)--"`，其中`...` 为实际内容，减号个数可以根据需要增加。

6.3 paste 函数

针对字符型数据最常用的 R 函数是 `paste()` 函数。`paste()` 用来连接两个字符型向量，元素一一对应连接，默认用空格连接。如 `paste(c("ab", "cd"), c("ef", "gh"))` 结果相当于 `c("ab ef", "cd gh")`。

`paste()` 在连接两个字符型向量时采用 R 的一般向量间运算规则，而且可以自动把数值型向量转换为字符型向量。可以作一对多连接，如 `paste("x", 1:3)` 结果相当于 `c("x 1", "x 2", "x 3")`。

用 `sep=` 指定分隔符，如 `paste("x", 1:3, sep="")` 结果相当于 `c("x1", "x2", "x3")`。

使用 `collapse=` 参数可以把字符型向量的各个元素连接成一个单一的字符串，如 `paste(c("a", "b", "c"), collapse="")` 结果相当于`"abc"`。

6.4 转换大小写

`toupper()` 函数把字符型向量内容转为大写，`tolower()` 函数转为小写。比如，`toupper('aB cd')` 结果为`"AB CD"`，`tolower(c('aB', 'cd'))` 结果相当于 `c("ab" "cd")`。这两个函数可以用于不区分大小写的比较，比如，不论 x 的值是`'JAN'`, `'Jan'`还是`'jan'`，`toupper(x)=='JAN'`的结果都为 TRUE。

6.5 字符串长度

用 `nchar(x, type='bytes')` 计算字符型向量 `x` 中每个字符串的以字节为单位的长度，这一点对中英文是有差别的，中文通常一个汉字占两个字节，英文字母、数字、标点占一个字节。用 `nchar(x, type='chars')` 计算字符型向量 `x` 中每个字符串的以字符个数为单位的长度，这时一个汉字算一个单位。

在画图时可以用 `strwidth()` 函数计算某个字符串或表达式占用的空间大小。

6.6 取子串

`substr(x, start, stop)` 从字符串 x 中取出从第 start 个到第 stop 个的子串，如

```
substr('JAN07', 1, 3)
## [1] "JAN"
```

如果 x 是一个字符型向量，`substr` 将对每个元素取子串。如

```
substr(c('JAN07', 'MAR66'), 1, 3)
## [1] "JAN" "MAR"
```

用 substring(x, start) 可以从字符串 x 中取出从第 start 个到末尾的子串。如

```
substring(c('JAN07', 'MAR66'), 4)
## [1] "07" "66"
```

6.7 类型转换

用 as.numeric() 把内容是数字的字符型值转换为数值，如

```
substr('JAN07', 4, 5)
## [1] "07"
substr('JAN07', 4, 5) + 2000
## Error in substr("JAN07", 4, 5) + 2000 :
##   non-numeric argument to binary operator
as.numeric(substr('JAN07', 4, 5)) + 2000
## [1] 2007
as.numeric(substr(c('JAN07', 'MAR66'), 4, 5))
## [1]  7 66
```

as.numeric() 是向量化的，可以转换一个向量的每个元素为数值型。

用 as.character() 函数把数值型转换为字符型，如

```
as.character((1:5)*5)
## [1] "5"  "10" "15" "20" "25"
```

如果自变量本来已经是字符型则结果不变。

为了用指定的格式数值型转换成字符型，可以使用 sprintf() 函数，其用法与 C 语言的 sprintf() 函数相似，只不过是向量化的。例如

```
sprintf('file%03d.txt', c(1, 99, 100))
```

```
## [1] "file001.txt" "file099.txt" "file100.txt"
```

readr 包的 parse_number() 输入一个字符串向量，对每个字符串，找到第一个能识别为数值的内容，舍弃其它内容，返回转换浮点型结果。没有数值时返回缺失值，并增加一个表格用来记录所有的不成功转换。如：

```
readr::parse_number(c(
  "123", "output-123.txt", "a123.456bc04",
  "30.2%", "abc"
))
```

```
## Warning: 1 parsing failure.
## row col expected actual
##   5  -- a number    abc

## [1]  123.000 -123.000  123.456   30.200       NA
## attr(,"problems")
## # A tibble: 1 x 4
##     row   col expected actual
##   <int> <int> <chr>    <chr>
## 1     5    NA a number abc
```

readr 中还有 parse_integer, parse_double, parse_logical, parse_character 等函数，这些函数不允许要读取内容以外的内容存在，比如 readr::parse_number("text-123") 能正确读取-123，而 readr::parse_integer("text-123") 则会返回缺失值，并带有一个说明缺失情况的表格作为属性。

6.8 字符串替换功能

用 gsub() 可以替换字符串中的子串，这样的功能经常用在数据清理中。比如，把数据中的中文标点改为英文标点，去掉空格，等等。如

```
x <- '1, 3; 5'
gsub(';', ',', x, fixed=TRUE)
## [1] "1, 3, 5"
```

字符串 x 中分隔符既有逗号又有分号，上面的程序用 gsub() 把分号都换成逗号。

更多的文本数据（字符型数据）功能参见41。

6.9 正则表达式

正则表达式 (regular expression) 是一种匹配某种字符串模式的方法。用这样的方法，可以从字符串中查找某种模式的出现位置，替换某种模式，等等。这样的技术可以用于文本数据的预处理，比如用网络爬虫下载的大量网页文本数据。R 中支持 perl 语言格式的正则表达式，grep() 和 grepl() 函数从字符串中查询某个模式，sub() 和 gsub() 替换某模式。比如，下面的程序把多于一个空格替换成一个空格

```
gsub('[[:space:]]+', ' ',
    'a   cat  in a box', perl=TRUE)
```

```
## [1] "a cat in a box"
```

正则表达式功能强大但也不容易掌握。详见41。

Chapter 7

R 向量下标和子集

在 R 中下标与子集是极为强大的功能，需要一些练习才能熟练掌握，许多其它语言中需要多个语句才能完成的工作在 R 中都可以简单地通过下标和子集来完成。

7.1 正整数下标

对向量 x, 在后面加方括号和下标可以访问向量的元素和子集。

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。例如

```
x <- c(1, 4, 6.25)
x[2]
## [1] 4
x[2] <- 99; x
## [1]  1.00 99.00  6.25
x[c(1,3)]
## [1] 1.00 6.25
x[c(1,3)] <- c(11, 13); x
## [1] 11 99 13
x[c(1,3,1)]
## [1] 11 13 11
```

7.2 负整数下标

负下标表示扣除相应的元素后的子集，如

```
x <- c(1,4,6.25)
x[-2]
## [1] 1.00 6.25
x[-c(1,3)]
## [1] 4
```

负整数下标不能与正整数下标同时用来从某一向量中取子集，比如，`x[c(1,-2)]` 没有意义。

7.3 空下标与零下标

`x[]` 表示取 `x` 的全部元素作为子集。这与 `x` 本身不同，比如

```
x <- c(1,4,6.25)
x[] <- 999
x
## [1] 999 999 999
x <- c(1,4,6.25)
x <- 999
x
## [1] 999
```

`x[0]` 是一种少见的做法，结果返回类型相同、长度为零的向量，如 `numeric(0)`。相当于空集。

当 0 与正整数下标一起使用时会被忽略。当 0 与负整数下标一起使用时也会被忽略。

7.4 下标超界

设向量 `x` 长度为 n, 则使用正整数下标时下标应在 $\{1, 2, \ldots, n\}$ 中取值。如果使用大于 n 的下标，读取时返回缺失值，并不出错。给超出 n 的下标元素赋值，则向量自动变长，中间没有赋值的元素为缺失值。例如

```
x <- c(1,4,6.25)
x[5]
## [1] NA
x
## [1] 1.00 4.00 6.25
x[5] <- 9
x
## [1] 1.00 4.00 6.25   NA 9.00
```

虽然 R 的语法对下标超界不视作错误，但是这样的做法往往来自不良的程序思路，而且对程序效率有影响，所以实际编程中应避免下标超界。

7.5 逻辑下标

下标可以是与向量等长的逻辑表达式，一般是关于本向量或者与本向量等长的其它向量的比较结果，如

```
x <- c(1,4,6.25)
x[x > 3]
## [1] 4.00 6.25
```

取出 x 的大于 3 的元素组成的子集。

逻辑下标除了用来对向量取子集，还经常用来对数据框取取子集，也用在向量化的运算中。例如，对如下示性函数

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

输入向量 x，结果 y 需要也是一个向量，程序可以写成

```
f <- function(x){
  y <- numeric(length(x))
  y[x >= 0] <- 1
  y[x < 0] <- 0 # 此语句多余

  y
}
```

事实上，向量化的逻辑选择有一个 `ifelse()` 函数，比如，对上面的示性函数，如果 x 是一个向量，输出 y 向量可以写成 `y <- ifelse(x>=0, 1, 0)`。

要注意的是，如果逻辑下标中有缺失值，对应结果也是缺失值。所以，在用逻辑下标作子集选择时，一定要考虑到缺失值问题。正确的做法是加上`!is.na` 前提，如

```
x <- c(1, 4, 6.25, NA)
x[x > 2]
## [1] 4.00 6.25   NA
x[!is.na(x) & x > 2]
## [1] 4.00 6.25
```

7.6 which()、which.min()、which.max() 函数

函数 which() 可以用来找到满足条件的下标，如

```
x <- c(3, 4, 3, 5, 7, 5, 9)
which(x > 5)
## [1] 5 7
seq(along=x)[x > 5]
## [1] 5 7
```

这里 seq(along=x) 会生成由 x 的下标组成的向量。用 which.min()、which.max 求最小值的下标和最大值的下标，不唯一时只取第一个。如

```
which.min(x)
## [1] 1
which.max(x)
## [1] 7
```

7.7 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")
```

或

```
ages <- setNames(c(30, 25, 28), c(" 李明", " 张聪", " 刘颖"))
```

这时可以用元素名或元素名向量作为向量的下标，如

```
ages[" 张聪"]
## 张聪
##   25
ages[c(" 李明", " 刘颖")]
## 李明 刘颖
```

```
##   30   28
ages[" 张聪"] <- 26
```

这实际上建立了字符串到数值的映射表。

用字符串作为下标时，如果该字符串不在向量的元素名中，读取时返回缺失值结果，赋值时该向量会增加一个元素并以该字符串为元素名。

带有元素名的向量也可以是字符型或其它基本类型，如

```
sex <- c(" 李明"=" 男", " 张聪"=" 男", " 刘颖"=" 女")
```

除了给向量元素命名外，在矩阵和数据框中还可以给行、列命名，这会使得程序的扩展更为容易和安全。

R 允许仅给部分元素命名，这时其它元素名字为空字符串。不同元素的元素名一般应该是不同的，否则在使用元素作为下标时会发生误读，但是 R 语法允许存在重名。

用 `unname(x)` 返回去掉了元素名的 `x` 的副本，用 `names(x) <- NULL` 可以去掉 `x` 的元素名。

7.8 用 R 向量下标作映射

R 在使用整数作为向量下标时，允许使用重复下标，这样可以把数组 `x` 看成一个 $1:n$ 的整数到 `x[1]`, `x[2]`, ..., `x[n]` 的一个映射表, 其中 n 是 `x` 的长度。比如，某商店有三种礼品，编号为 1,2,3，价格分别为 68, 88 和 168。令

```
price_map <- c(68, 88, 168)
```

设某个收银员在一天内分别售出礼品编号为 3,2,1,1,2,2,3，可以用如下的映射方式获得售出的这些礼品对应的价格：

```
items <- c(3,2,1,1,2,2,3)
y <- price_map[items]; print(y)
## [1] 168  88  68  68  88  88 168
```

R 向量可以用字符型向量作下标，字符型下标也允许重复，所以可以把带有元素名的 R 向量看成是元素名到元素值的映射表。比如，设 sex 为 10 个学生的性别（男、女）

```
sex <- c(" 男", " 男", " 女", " 女", " 男", " 女", " 女", " 女", " 女", " 男")
```

希望把每个学生按照性别分别对应到蓝色和红色。首先建立一个 R 向量当作映射

```
sex_color <- c(" 男"="blue", " 女"="red")
```

用 R 向量 sex.color 当作映射，可以获得每个学生对应的颜色

```
cols <- sex_color[sex]; print(cols)
##     男     男     女     女     男     女     女     女     女     男
## "blue" "blue"  "red"  "red" "blue"  "red"  "red"  "red"  "red" "blue"
```

这样的映射结果中带有不必要的元素名，用 unname() 函数可以去掉元素名，如

```
unname(cols)
##  [1] "blue" "blue" "red"  "red"  "blue" "red"  "red"  "red"  "red"  "blue"
```

7.9 集合运算

可以把向量 x 看成一个集合，但是其中的元素允许有重复。用 unique(x) 可以获得 x 的所有不同值。如

```
unique(c(1, 5, 2, 5))
## [1] 1 5 2
```

用 a %in% x 判断 a 的每个元素是否属于向量 x，如

```
5 %in% c(1,5,2)
## [1] TRUE
c(5,6) %in% c(1,5,2)
## [1]  TRUE FALSE
```

与%in 运算符类似，函数 match(x, table) 对向量 x 的每个元素，从向量 table 中查找其首次出现位置并返回这些位置。没有匹配到的元素位置返回 NA_integer_(整数型缺失值)。如

```
match(5, c(1,5,2))
## [1] 2
match(5, c(1,5,2,5))
## [1] 2
match(c(2,5), c(1,5,2,5))
## [1] 3 2
match(c(2,5,0), c(1,5,2,5))
## [1]  3  2 NA
```

用 intersect(x,y) 求交集，结果中不含重复元素，如

```
intersect(c(5, 7), c(1, 5, 2, 5))
## [1] 5
```

用 union(x,y) 求并集，结果中不含重复元素，如

```
union(c(5, 7), c(1, 5, 2, 5))
## [1] 5 7 1 2
```

用 setdiff(x,y) 求差集，即 x 的元素中不属于 y 的元素组成的集合，结果中不含重复元素，如

```
setdiff(c(5, 7), c(1, 5, 2, 5))
## [1] 7
```

用 setequal(x,y) 判断两个集合是否相等，不受次序与重复元素的影响，如

```
setequal(c(1,5,2), c(2,5,1))
## [1] TRUE
setequal(c(1,5,2), c(2,5,1,5))
## [1] TRUE
```

7.10 练习

设文件class.csv内容如下:

```
name,sex,age,height,weight
Alice,F,13,56.5,84
Becka,F,13,65.3,98
Gail,F,14,64.3,90
Karen,F,12,56.3,77
Kathy,F,12,59.8,84.5
Mary,F,15,66.5,112
Sandy,F,11,51.3,50.5
Sharon,F,15,62.5,112.5
Tammy,F,14,62.8,102.5
Alfred,M,14,69,112.5
Duke,M,14,63.5,102.5
Guido,M,15,67,133
James,M,12,57.3,83
Jeffrey,M,13,62.5,84
John,M,12,59,99.5
Philip,M,16,72,150
Robert,M,12,64.8,128
Thomas,M,11,57.5,85
William,M,15,66.5,112
```

用如下程序可以把上述文件读入为 R 数据框 d.class, 并取出其中的 name 和 age 列到变量 name 和 age 中：

```
d.class <- read.csv("class.csv", header=TRUE, stringsAsFactors=FALSE)
name <- d.class[,"name"]
age <- d.class[,"age"]
```

(1) 求出 age 中第 3, 5, 7 号的值；
(2) 用变量 age, 求出达到 15 岁及以上的那些值；
(3) 用变量 name 和 age, 求出 Mary 与 James 的年龄。
(4) 求 age 中除 Mary 与 James 这两人之外的那些人的年龄值，保存到变量 age1 中。
(5) 假设向量 `x` 长度为 `n`, 其元素是 $\{1, 2, \ldots, n\}$ 的一个重排。可以把 x 看成一个 i 到 x[i] 的映射 (i 在 $\{1, 2, \ldots, n\}$ 中取值)。求向量 y, 保存了上述映射的逆映射，即：如果 x[i]=j, 则 y[j]=i。

Chapter 8

R 数据类型的性质

8.1 存储模式与基本类型

R 的变量可以存储多种不同的数据类型，可以用 `typeof()` 函数来返回一个变量或表达式的类型。比如

```
typeof(1:3)
## [1] "integer"
typeof(c(1,2,3))
## [1] "double"
typeof(c(1, 2.1, 3))
## [1] "double"
typeof(c(TRUE, NA, FALSE))
## [1] "logical"
typeof('Abc')
## [1] "character"
typeof(factor(c('F', 'M', 'M', 'F')))
## [1] "integer"
```

注意因子的结果是 `integer` 而不是因子。

R 还有两个函数 `mode()` 和 `storage.mode()` 起到与 `typeof()` 类似的作用，这是为了提供与 S 语言兼容所遗留的，应停止使用。

R 中数据的最基本的类型包括 logical, integer, double, character, complex, raw, 其它数据类型都是由基本类型组合或转变得到的。character 类型就是字符串类型，raw 类型是直接使用其二进制内容的类型。为了判断某个向量 `x` 保存的基本类型，可以用 `is.xxx()` 类函数，如 `is.integer(x)`, `is.double(x)`, `is.numeric(x)`, `is.logical(x)`, `is.character(x)`, `is.complex(x)`, `is.raw(x)`。其中 `is.numeric(x)` 对 integer 和 double 内容都返回真值。

在 R 语言中数值一般看作 double, 如果需要明确表明某些数值是整数，可以在数值后面附加字母 L，如

```
is.integer(c(1, -3))
## [1] FALSE
is.integer(c(1L, -3L))
## [1] TRUE
```

整数型的缺失值是 NA，而 double 型的特殊值除了 NA 外，还包括 Inf, -Inf 和 NaN，其中 NaN 也算是缺失值，Inf 和-Inf 不算是缺失值。如:

```
c(-1, 0, 1)/0
## [1] -Inf  NaN  Inf
is.na(c(-1, 0, 1)/0)
## [1] FALSE  TRUE FALSE
```

对 double 类型，可以用 is.finite() 判断是否有限值，NA、Inf, -Inf 和 NaN 都不是有限值；用 is.infinite() 判断是否 Inf 或-Inf；is.na() 判断是否 NA 或 NaN；is.nan() 判断是否 NaN。

严格说来，NA 表示逻辑型缺失值，但是当作其它类型缺失值时一般能自动识别。NA_integer_ 是整数型缺失值，NA_real_ 是 double 型缺失值，NA_character_ 是字符型缺失值。

在 R 的向量类型中，integer 类型、double 类型、logical 类型、character 类型、还有 complex 类型和 raw 类型称为原子类型 (atomic types)，原子类型的向量中元素都是同一基本类型的。比如，double 型向量的元素都是 double 或者缺失值。

除了原子类型的向量，在 R 语言的定义中，向量还包括后面要讲到的列表（list），列表的元素不需要属于相同的基本类型，而且列表的元素可以不是单一基本类型元素。用 typeof() 函数可以返回向量的类型，列表返回结果为"list":

```
typeof(list("a", 1L, 1.5))
## [1] "list"
```

原子类型的各个元素除了基本类型相同，还不包含任何嵌套结构，如：

```
c(1, c(2,3, c(4,5)))
## [1] 1 2 3 4 5
```

R 有一个特殊的 NULL 类型，这个类型只有唯一的一个 NULL 值，表示不存在。NULL 长度为 0，不能有任何属性值。用 is.null() 函数判断某个变量是否取 NULL。

NULL 值可以用来表示类型未知的零长度向量，如 c() 没有自变量时返回值就是 NULL；也经常用作函数缺省值，在函数内用 is.null() 判断其缺省后再用一定的计算逻辑得到真正的缺省情况下的数值。

要把 NULL 与 NA 区分开来，NA 是有类型的（integer、double、logical、character 等), NA 表示存在但是未知。数据库管理系统中的 NULL 值相当于 R 中的 NA 值。

8.2 类型转换与类型升档

可以用 `as.xxx()` 类的函数在不同类型之间进行强制转换。如

```
as.numeric(c(FALSE, TRUE))
## [1] 0 1
as.character(sqrt(1:4))
## [1] "1"                "1.4142135623731"  "1.73205080756888" "2"
```

类型转换也可能是隐含的，比如，四则运算中数值会被统一转换为 double 类型，逻辑运算中运算元素会被统一转换为 logical 类型。逻辑值转换成数值时，`TRUE` 转换成 1，`FALSE` 转换成 0。

在用 `c()` 函数合并若干元素时，如果元素基本类型不同，将统一转换成最复杂的一个，复杂程度从简单到复杂依次为：`logical<integer<double<character`。这种做法称为类型升档，如

```
c(FALSE, 1L, 2.5, "3.6")
## [1] "FALSE" "1"     "2.5"   "3.6"
```

不同类型参与要求类型相同的运算时，也会统一转换为最复杂的类型，也称为类型升档，如：

```
TRUE + 10
## [1] 11
paste("abc", 1)
## [1] "abc 1"
```

8.3 属性

除了 `NULL` 以外，R 的变量都可以看成是对象，都可以有属性。在 R 语言中，属性是把变量看成对象后，除了其存储内容（如元素）之外的其它附加信息，如维数、类属等。R 对象一般都有 `length` 和 `mode` 两个属性。

常用属性有 `names`, `dim`，`class` 等。

8.3.1 attributes 函数

对象 `x` 的所有属性可以用 `attributes()` 读取，如

```
x <- table(c(1,2,1,3,2,1)); print(x)
##
## 1 2 3
## 3 2 1
attributes(x)
```

```
## $dim
## [1] 3
##
## $dimnames
## $dimnames[[1]]
## [1] "1" "2" "3"
##
##
## $class
## [1] "table"
```

table() 函数用了输出其自变量中每个不同值的出现次数，称为频数。从上例可以看出，table() 函数的结果有三个属性，前两个是 dim 和 dimnames, 这是数组 (array) 具有的属性；另一个是 class 属性，值为"table"。因为 x 是数组，可以访问如

```
x[1]
## 1
## 3
x["3"]
## 3
## 1
```

也可以用 attributes() 函数修改属性，如

```
attributes(x) <- NULL
x
## [1] 3 2 1
```

如上修改后 x 不再是数组，也不是 table。

8.3.2 attr 函数

可以用 attr(x, " 属性名") 的格式读取或定义 x 的属性。如：

```
x <- c(1,3,5)
attr(x, "theta") <- c(0, 1)
print(x)
## [1] 1 3 5
## attr(,"theta")
## [1] 0 1
```

可以让向量 x 额外地保存一个 theta 属性，这样的属性常常成为 “元数据”(meta data)，比如，用来保存数据的说明、模拟数据的真实模型参数，等等。

8.3.3 names 属性

有元素名的向量、列表、数据框等都有 names 属性，许多 R 函数的输出本质上也是列表，所以也有 names 属性。用 names(x) 的格式读取或设定。如：

```
x <- 1:5
y <- x^2
lmr <- lm(y ~ x)
print(names(lmr))
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

对于没有元素名的向量 x，names(x) 的返回值是 NULL。

8.3.4 dim 属性

dim 属性的存在表明对象是矩阵或一维、多维数组。如：

```
x <- matrix(1:12, nrow=3, ncol=4)
attr(x, "dim") # 等同于 dim(x)
## [1] 3 4
```

修改 dim 属性就将向量转换成矩阵（数组），或修改了矩阵的性质，元素按列次序重排填入新的矩阵。如：

```
x <- 1:4
dim(x) <- c(2,2)
x
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

R 允许 dim 仅有一个元素，这对应于一维向量，与普通的没有 dim 属性的向量有区别。另外要注意，取矩阵子集时如果结果仅有一列或一行，除非用了 drop=FALSE 选项，结果不再有 dim 属性，退化成了普通向量。

8.4 类属

R 具有一定的面向对象语言特征，其数据类型有一个 class 属性，函数 class() 可以返回变量类型的类属，比如

```
typeof(factor(c('F', 'M', 'M', 'F')))
## [1] "integer"
mode(factor(c('F', 'M', 'M', 'F')))
## [1] "numeric"
storage.mode(factor(c('F', 'M', 'M', 'F')))
## [1] "integer"
class(factor(c('F', 'M', 'M', 'F')))
## [1] "factor"
class(as.numeric(factor(c('F', 'M', 'M', 'F'))))
## [1] "numeric"
```

class 属性是特殊的。如果一个对象具有 class 属性，某些所谓 “通用函数 (generic functions)” 会针对这样的对象进行专门的操作，比如，`print()` 函数在显示向量和回归结果时采用完全不同的格式。这在其它程序设计语言中称为 “重载”(overloading)。

class 属性用来支持 R 的 S3 风格的类，常用的有 factor, 日期，日期时间，数据框，tibble。R 还有 S4、R6 等风格的类。

8.5 str() 函数

用 `print()` 函数可以显示对象内容。如果内容很多，显示行数可能也很多。用 `str()` 函数可以显示对象的类型和主要结构及典型内容。例如

```
s <- 101:200
attr(s,'author') <- '李小明'
attr(s,'date') <- '2016-09-12'
str(s)
##  int [1:100] 101 102 103 104 105 106 107 108 109 110 ...
##  - attr(*, "author")= chr " 李小明"
##  - attr(*, "date")= chr "2016-09-12"
```

Chapter 9

R 日期时间

9.1 R 日期和日期时间类型

R 日期可以保存为 Date 类型，一般用整数保存，数值为从 1970-1-1 经过的天数。

R 中用一种叫做 POSIXct 和 POSIXlt 的特殊数据类型保存日期和时间，可以仅包含日期部分，也可以同时有日期和时间。技术上，POSIXct 把日期时间保存为从 1970 年 1 月 1 日零时到该日期时间的时间间隔秒数，所以数据框中需要保存日期时用 POSIXct 比较合适，需要显示时再转换成字符串形式；POSIXlt 把日期时间保存为一个包含年、月、日、星期、时、分、秒等成分的列表，所以求这些成分可以从 POSIXlt 格式日期的列表变量中获得。日期时间会涉及到所在时区、夏时制等问题，比较复杂。

为了获得专用的时间类型，可以使用 hms 扩展包。

基础的 R 用 `as.Date()`、`as.POSIXct()` 等函数生成日期型和日期时间型，R 扩展包 lubridate 提供了多个方便函数，可以更容易地生成、转换、管理日期型和日期时间型数据。

```
library(lubridate)
```

9.2 从字符串生成日期数据

函数 `lubridate::today()` 返回当前日期：

```
today()
## [1] "2023-04-26"
```

函数 `lubridate::now()` 返回当前日期时间：

```
now()
## [1] "2023-04-26 09:16:25 CST"
```

结果显示中出现的 CST 是时区，这里使用了操作系统提供的当前时区。CST 不是一个含义清晰的时区，在不同国家对应不同的时区，在中国代表中国标准时间（北京时间）。

用 lubridate::ymd(), lubridate::mdy(), lubridate::dmy() 将字符型向量转换为日期型向量，如：

```
ymd(c("1998-3-10", "2018-01-17", "18-1-17"))
## [1] "1998-03-10" "2018-01-17" "2018-01-17"
mdy(c("3-10-1998", "01-17-2018"))
## [1] "1998-03-10" "2018-01-17"
dmy(c("10-3-1998", "17-01-2018"))
## [1] "1998-03-10" "2018-01-17"
```

在年号只有两位数字时，默认对应到 1969-2068 范围。

lubridate 包的 ymd、mdy、dmy 等函数添加 hms、hm、h 等后缀，可以用于将字符串转换成日期时间。如

```
ymd_hms("1998-03-16 13:15:45")
## [1] "1998-03-16 13:15:45 UTC"
```

9.3 时区和时区转换

上面例子结果显示中 UTC 是时区，UTC 是协调世界时 (Universal Time Coordinated) 英文缩写，是由国际无线电咨询委员会规定和推荐, 并由国际时间局 (BIH) 负责保持的以秒为基础的时间标度。UTC 相当于本初子午线 (即经度 0 度) 上的平均太阳时，过去曾用格林威治平均时 (GMT) 来表示。北京时间比 UTC 时间早 8 小时，以 1999 年 1 月 1 日 0000UTC 为例，UTC 时间是零点，北京时间为 1999 年 1 月 1 日早上 8 点整。日期时间字符串中日期部分和时间部分以空格分隔，也可以用字母 T 分隔。

在 Date()、as.DateTime()、ymd() 等函数中，可以用 tz= 指定时区，比如北京时间可指定为 tz="Etc/GMT-8" 或 tz="Asia/Shanghai"。

为了将某个时间转换到指定的时区，而不改变真正的时间，用 with_tz() 函数，如：

```
with_tz(ymd_hms(c("1998-03-16 13:15:45",
  "2023-03-14 10:11:12"),
  tz="Asia/Shanghai"), tzone="UTC")
```

```
## [1] "1998-03-16 05:15:45 UTC" "2023-03-14 02:11:12 UTC"
```

为了保持表面的时间（时钟显示的日期时间）不变，但将真正的时间修改到另外的时区，用 force_tz() 或 force_tzs()，其中 force_tzs() 可以将每个时间单独应用不同的时区。如：

```
force_tz(ymd_hms(c("1998-03-16 13:15:45",
  "2023-03-14 10:11:12"),
  tz="Asia/Shanghai"), tzone="UTC")
```

```
## [1] "1998-03-16 13:15:45 UTC" "2023-03-14 10:11:12 UTC"
```

上例的结果显示中，仅时区改变了，显示的日期、时间都没有变，但这样就改变了真实的时间。输入的日期中的时区"Asia/Shanghai" 不起作用，仅该时区的钟面时间起作用。

又如：

```
force_tzs(ymd_hms(c("1998-03-16 13:15:45",
  "2023-03-14 10:11:12"),
  tz="Asia/Shanghai"),
  tzones=c("Etc/GMT-6", "Etc/GMT-10"))
```

```
## [1] "1998-03-16 07:15:45 UTC" "2023-03-14 00:11:12 UTC"
```

这将北京时间"1998-03-16 13:15:45" 改成了东 6 区时间，对应到 UTC 就是钟面时间减 6，所以 13 点变成 7 点，将北京时间"2023-03-14 10:11:12" 改成了东 10 区时间，对应到 UTC 就是钟面时间减 10，所以 10 点变成 0 点。输入的日期中的时区"Asia/Shanghai" 不起作用，仅该时区的钟面时间起作用。

9.4 从数值生成日期数据

lubridate::make_date(year, month, day) 可以从三个数值构成日期向量。如

```
make_date(1998, 3, 10)
## [1] "1998-03-10"
```

lubridate::make_datetime(year, month, day, hour, min, sec) 可以从最多六个数值组成日期时间，其中时分秒缺省值都是 0。如

```
make_datetime(1998, 3, 16, 13, 15, 45.2)
## [1] "1998-03-16 13:15:45 UTC"
```

在 make_date() 和 make_datetime() 都可以用 tz 选项指定时区，但仅支持单个时区。如果多个时间的时区也不同，应该使用 force_tzs() 函数。

9.5 日期和日期时间之间的转换

用 lubridate::as_date() 可以将日期时间型转换为日期型，如

```
as_date(as.POSIXct("1998-03-16 13:15:45"))
## [1] "1998-03-16"
```

用 lubridate::as_datetime() 可以将日期型数据转换为日期时间型，如

```
as.Date("1998-03-16") |>
  as_datetime() |>
  class()
## [1] "POSIXct" "POSIXt"
```

9.6 日期显示格式

用 as.character() 函数把日期型数据转换为字符型, 如

```
x <- as.POSIXct(c('1998-03-16', '2015-11-22'))
as.character(x)
## [1] "1998-03-16" "2015-11-22"
```

在 as.character() 中可以用 format 选项指定显示格式，如

```
as.character(x, format='%m/%d/%Y')
## [1] "03/16/1998" "11/22/2015"
```

格式中 “%Y” 代表四位的公元年号，“%m” 代表两位的月份数字，“%d” 代表两位的月内日期号。

这些格式缩写也被用在从字符串转换在日期时，比如 readr::read_csv 在指定某一列为 col_date() 时，可以指定格式如 format="%m/%d/%Y"。又如：

```
as.Date(c("12/6/2022", "1/1/2023"), format="%m/%d/%Y")
## [1] "2022-12-06" "2023-01-01"
```

"15Mar98" 这样的日期在英文环境中比较常见，但是在 R 中的处理比较复杂。在下面的例子中，R 日期被转换成了类似"Mar98" 这样的格式，在 format 选项中用了 “%b” 代表三英文字母月份缩写，但是因为月份缩写依赖于操作系统默认语言环境，需要用 Sys.setlocale() 函数设置语言环境为"C"。示例程序如下

```
x <- as.POSIXct(c('1998-03-16', '2015-11-22'))
old.lctime <- Sys.getlocale('LC_TIME')
Sys.setlocale('LC_TIME', 'C')
## [1] "C"
as.character(x, format='%b%y')
## [1] "Mar98" "Nov15"
Sys.setlocale('LC_TIME', old.lctime)
## [1] "Chinese (Simplified)_China.utf8"
```

format 选项中的 "%y" 表示两位数的年份，应尽量避免使用两位数年份以避免混淆。

包含时间的转换如

```
x <- as.POSIXct('1998-03-16 13:15:45')
as.character(x)
## [1] "1998-03-16 13:15:45"
as.character(x, format='%H:%M:%S')
## [1] "13:15:45"
```

这里 "%H" 代表小时（按 24 小时制），"%M" 代表两位的分钟数字，"%S" 代表两位的秒数。

9.7 访问日期时间的组成值

lubridate 包的如下函数可以取出日期型或日期时间型数据中的组成部分：

- year() 取出年
- month() 取出月份数值
- mday() 取出日数值
- yday() 取出日期在一年中的序号，元旦为 1
- wday() 取出日期在一个星期内的序号，但是一个星期从星期天开始，星期天为 1, 星期一为 2，星期六为 7。
- hour() 取出小时
- minute() 取出分钟
- second() 取出秒

比如, 2018-1-17 是星期三，则

```
month(as.POSIXct("2018-1-17 13:15:40"))
## [1] 1
mday(as.POSIXct("2018-1-17 13:15:40"))
## [1] 17
```

```
wday(as.POSIXct("2018-1-17 13:15:40"))
## [1] 4
```

lubridate 的这些成分函数还允许被赋值，结果就修改了相应元素的值，如

```
x <- as.POSIXct("2018-1-17 13:15:40")
year(x) <- 2000
month(x) <- 1
mday(x) <- 1
x
## [1] "2000-01-01 13:15:40 CST"
```

update() 可以对一个日期或一个日期型向量统一修改其组成部分的值，如

```
x <- as.POSIXct("2018-1-17 13:15:40")
y <- update(x, year=2000)
y
## [1] "2000-01-17 13:15:40 CST"
```

update() 函数中可以用 year, month, mday, hour, minute, second 等参数修改日期的组成部分。

用 lubridate 包的功能计算周岁如下：

```
age.int <- function(birth, now){
  age <- year(now) - year(birth)
  sele <- (month(now) * 100 + mday(now)
             < month(birth) * 100 + mday(birth))
  ## sele 是那些没有到生日的人
  age[sele] <- age[sele] - 1

  age
}
```

9.8 日期舍入计算

lubridate 包提供了 floor_date(), round_date(), ceiling_date() 等函数，对日期可以用 unit= 指定一个时间单位进行舍入。时间单位为字符串，如 seconds, 5 seconds, minutes, 2 minutes, hours, days, weeks, months, years 等。

比如，以 10 minutes 为单位，floor_date() 将时间向前归一化到 10 分钟的整数倍，ceiling_date() 将时间向后归一化到 10 分钟的整数倍，round_date() 将时间归一化到最近的 10 分钟的整数倍，时间恰好是 5 分钟倍数时按照类似四舍五入的原则向上取整。例如

```
x <- ymd_hms("2018-01-11 08:32:44")
floor_date(x, unit="10 minutes")
## [1] "2018-01-11 08:30:00 UTC"
ceiling_date(x, unit="10 minutes")
## [1] "2018-01-11 08:40:00 UTC"
round_date(x, unit="10 minutes")
## [1] "2018-01-11 08:30:00 UTC"
```

如果单位是星期，会涉及到一个星期周期的开始是星期日还是星期一的问题。用参数 `week_start=7` 指定开始是星期日，`week_start=1` 指定开始是星期一。

9.9 日期计算

在 lubridate 的支持下日期可以相减，可以进行加法、除法。lubridate 包提供了如下的三种与时间长短有关的数据类型：

- 时间长度 (duration)，按整秒计算;
- 时间周期 (period)，如日、周;
- 时间区间 (interval)，包括一个开始时间和一个结束时间。

9.9.1 时间长度

R 的 POSIXct 日期时间之间可以相减，如

```
d1 <- ymd_hms("2000-01-01 0:0:0")
d2 <- ymd_hms("2000-01-02 12:0:5")
di <- d2 - d1; di
## Time difference of 1.500058 days
```

结果显示与日期之间差别大小有关系，结果是类型是 difftime。为了转换成数值，比如秒数，可以用 `as.double()` 和 `units="secs"`，如：

```
as.double(di, units="days")
## [1] 1.500058
as.double(di, units="hours")
## [1] 36.00139
as.double(di, units="mins")
## [1] 2160.083
as.double(di, units="secs")
## [1] 129605
```

lubridate 包提供了 duration 类型，固定以秒作为基本单位，所以处理更方便：

```
as.duration(di)
## [1] "129605s (~1.5 days)"
```

lubridate 的 `dseconds()`, `dminutes()`, `dhours()`, `ddays()`, `dweeks()`, `dyears()` 函数可以直接生成时间长度类型的数据，如

```
dhours(1)
```

```
## [1] "3600s (~1 hours)"
```

lubridate 的时间长度类型总是以秒作为单位，可以在时间长度之间相加，也可以对时间长度乘以无量纲数，如

```
dhours(1) + dseconds(5)
## [1] "3605s (~1 hours)"
dhours(1)*10
## [1] "36000s (~10 hours)"
```

可以给一个日期加或者减去一个时间长度，结果严格按推移的秒数计算，如

```
d2 <- ymd_hms("2000-01-02 12:0:5")
d2 - dhours(5)
## [1] "2000-01-02 07:00:05 UTC"
d2 + ddays(10)
## [1] "2000-01-12 12:00:05 UTC"
```

时间的前后推移在涉及到夏时制时有可能出现难以预料到的情况。

9.9.2 时间周期

时间长度的固定单位是秒，但是像月、年这样的单位，因为可能有不同的天数，所以日历中的时间单位往往没有固定的时长。

lubridate 包的 `seconds()`, `minutes()`, `hours()`, `days()`，`weeks()`, `years()` 函数可以生成以日历中正常的周期为单位的时间长度，不需要与秒数相联系，可以用于时间的前后推移。这些时间周期的结果可以相加、乘以无量纲整数：

```
years(2) + 10*days(1)
## [1] "2y 0m 10d 0H 0M 0S"
```

lubridate 的月度周期因为与已有函数名冲突,所以没有提供,需要使用 `lubridate::period(num, units="month")` 的格式，其中 `num` 是几个月的数值。

为了按照日历进行日期的前后平移，而不是按照秒数进行日期的前后平移，应该使用这些时间周期。

例如，因为 2016 年是闰年，按秒数给 2016-01-01 加一年，得到的并不是 2017-01-01：

```
ymd("2016-01-01") + dyears(1)
## [1] "2016-12-31 06:00:00 UTC"
```

使用时间周期函数则得到预期结果：

```
ymd("2016-01-01") + years(1)
## [1] "2017-01-01"
```

9.9.3 时间区间

lubridate 提供了`%--%` 运算符构造一个时间期间 (time interval)。时间区间可以求交集、并集等。

构造如：

```
d1 <- ymd_hms("2000-01-01 0:0:0")
d2 <- ymd_hms("2000-01-02 12:0:5")
din <- (d1 %--% d2); din
## [1] 2000-01-01 UTC--2000-01-02 12:00:05 UTC
```

对一个时间区间可以用除法计算其时间长度，如

```
din / ddays(1)
## [1] 1.500058
din / dseconds(1)
## [1] 129605
```

生成时间区间，也可以用 `lubridate::interval(start, end)` 函数，如

```
interval(ymd_hms("2000-01-01 0:0:0"),
  ymd_hms("2000-01-02 12:0:5"))
## [1] 2000-01-01 UTC--2000-01-02 12:00:05 UTC
```

可以指定时间长度和开始日期生成时间区间，如

```
d1 <- ymd("2018-01-15")
din <- as.interval(dweeks(1), start=d1); din
## [1] 2018-01-15 UTC--2018-01-22 UTC
```

注意这个时间区间表面上涉及到 8 个日期，但是实际长度还是只有 7 天，因为每一天的具体时间都是按零时计算，所以区间末尾的那一天实际不含在内。

用 lubridate::int_start() 和 lubridate::int_end() 函数访问时间区间的端点，如：

```
int_start(din)
## [1] "2018-01-15 UTC"
int_end(din)
## [1] "2018-01-22 UTC"
```

可以用 as.duration() 将一个时间区间转换成时间长度，用 as.period() 将一个时间区间转换为可变时长的时间周期个数。

用 lubridate::int_shift() 平移一个时间区间，如

```
din2 <- int_shift(din, by=ddays(3)); din2
## [1] 2018-01-18 UTC--2018-01-25 UTC
```

用 lubridate::int_overlaps() 判断两个时间区间是否有共同部分，如

```
int_overlaps(din, din2)
## [1] TRUE
```

时间区间允许开始时间比结束时间晚，用 lubridate::int_standardize() 可以将时间区间标准化成开始时间小于等于结束时间。lubridate() 现在没有提供求交集的功能，一个自定义求交集的函数如下：

```
int_intersect <- function(int1, int2){
  n <- length(int1)
  int1 <- lubridate::int_standardize(int1)
  int2 <- lubridate::int_standardize(int2)
  sele <- lubridate::int_overlaps(int1, int2)
  inter <- rep(lubridate::interval(NA, NA), n)
  if(any(sele)){
    inter[sele] <-
      lubridate::interval(pmax(lubridate::int_start(int1[sele]),
                               lubridate::int_start(int2[sele])),
                          pmin(lubridate::int_end(int1[sele]),
                               lubridate::int_end(int2[sele])))
```

```
  }
  inter
}
```

测试如：

```
d1 <- ymd(c("2018-01-15", "2018-01-18", "2018-01-25"))
d2 <- ymd(c("2018-01-21", "2018-01-23", "2018-01-30"))
din <- interval(d1, d2); din
## [1] 2018-01-15 UTC--2018-01-21 UTC 2018-01-18 UTC--2018-01-23 UTC
## [3] 2018-01-25 UTC--2018-01-30 UTC
int_intersect(rep(din[1], 2), din[2:3])
## [1] 2018-01-18 UTC--2018-01-21 UTC NA--NA
```

此自定义函数还可以进一步改成允许两个自变量长度不等的情形。

9.10 基本 R 软件的日期功能

9.10.1 生成日期和日期时间型数据

`Sys.date()` 返回 Date 类型的当前日期。`Sys.time()` 返回 POSIXct 类型的当前日期时间。

对 `yyyy-mm-dd` 或 `yyyy/mm/dd` 格式的数据，可以直接用 `as.Date()` 转换为 Date 类型，如：

```
x <- as.Date("1970-1-5"); x
## [1] "1970-01-05"
as.numeric(x)
## [1] 4
```

`as.Date()` 可以将多个日期字符串转换成 Date 类型，如

```
as.Date(c("1970-1-5", "2017-9-12"))
## [1] "1970-01-05" "2017-09-12"
```

对于非标准的格式，在 `as.Date()` 中可以增加一个 `format` 选项，其中用`%Y` 表示四位数字的年，`%m` 表示月份数字，`%d` 表示日数字。如

```
as.Date("1/5/1970", format="%m/%d/%Y")
## [1] "1970-01-05"
```

用 `as.POSIXct()` 函数把年月日格式的日期转换为 R 的标准日期，没有时间部分就认为时间在午夜。如

```
as.POSIXct(c('1998-03-16'))
## [1] "1998-03-16 CST"
as.POSIXct(c('1998/03/16'))
## [1] "1998-03-16 CST"
```

年月日中间的分隔符可以用减号也可以用正斜杠，但不能同时有减号又有斜杠。

待转换的日期时间字符串，可以是年月日之后隔一个空格以 “时: 分: 秒” 格式带有时间。如

```
as.POSIXct('1998-03-16 13:15:45')
## [1] "1998-03-16 13:15:45 CST"
```

用 `as.POSIXct()` 可以同时转换多项日期时间，如

```
as.POSIXct(c('1998-03-16 13:15:45', '2015-11-22 9:45:3'))
```

```
## [1] "1998-03-16 13:15:45 CST" "2015-11-22 09:45:03 CST"
```

转换后的日期变量有 class 属性，取值为 POSIXct 与 POSIXt, 并带有一个 tzone（时区）属性。

```
x <- as.POSIXct(c('1998-03-16 13:15:45', '2015-11-22 9:45:3'))
attributes(x)
## $class
## [1] "POSIXct" "POSIXt"
##
## $tzone
## [1] ""
```

在 `as.POSIXct()` 函数中用 `format` 参数指定一个日期格式。如

```
as.POSIXct('3/13/15', format='%m/%d/%y')
## [1] "2015-03-13 CST"
```

如果日期仅有年和月，必须添加日（添加 01 为日即可）才能读入。比如用’1991-12’ 表示 1991 年 12 月，则如下程序将其读入为’1991-12-01’:

```
as.POSIXct(paste('1991-12', '-01', sep=''), format='%Y-%m-%d')
## [1] "1991-12-01 CST"
```

又如

```
old.lctime <- Sys.getlocale('LC_TIME')
Sys.setlocale('LC_TIME', 'C')
## [1] "C"
as.POSIXct(paste('01', 'DEC91', sep=''), format='%d%b%y')
## [1] "1991-12-01 CST"
Sys.setlocale('LC_TIME', old.lctime)
## [1] "Chinese (Simplified)_China.utf8"
```

把'DEC91'转换成了'1991-12-01'。

如果明确地知道时区，在 as.POSIXct() 和 as.POSIXlt() 中可以加选项 tz= 字符串。选项 tz 的缺省值为空字符串，这一般对应于当前操作系统的默认时区。但是，有些操作系统和 R 版本不能使用默认值，这时可以为 tz 指定时区，比如北京时间可指定为 tz='Etc/GMT+8'。如

```
as.POSIXct('1949-10-01', tz='Etc/GMT+8')
## [1] "1949-10-01 -08"
```

9.10.2 取出日期时间的组成值

把一个 R 日期时间值用 as.POSIXlt() 转换为 POSIXlt 类型，就可以用列表元素方法取出其组成的年、月、日、时、分、秒等数值。如

```
x <- as.POSIXct('1998-03-16 13:15:45')
y <- as.POSIXlt(x)
cat(1900+y$year, y$mon+1, y$mday, y$hour, y$min, y$sec, '\n')
## 1998 3 16 13 15 45
```

注意 year 要加 1900，mon 要加 1。另外，列表元素 wday 取值 1-6 时表示星期一到星期六，取值 0 时表示星期天。

对多个日期，as.POSIXlt() 会把它们转换成一个列表（列表类型稍后讲述），这时可以用列表元素 year, mon, mday 等取出日期成分。如

```
x <- as.POSIXct(c('1998-03-16', '2015-11-22'))
as.POSIXlt(x)$year + 1900
## [1] 1998 2015
```

9.10.3 日期计算

因为 Date 类型是用数值保存的，所以可以给日期加减一个整数，如：

```
x <- as.Date("1970-1-5")
x1 <- x + 10; x1
## [1] "1970-01-15"
x2 <- x - 5; x2
## [1] "1969-12-31"
```

所有的比较运算都适用于日期类型。

可以给一个日期加减一定的秒数，如

```
as.POSIXct(c('1998-03-16 13:15:45')) - 30
## [1] "1998-03-16 13:15:15 CST"
as.POSIXct(c('1998-03-16 13:15:45')) + 10
## [1] "1998-03-16 13:15:55 CST"
```

但是两个日期不能相加。

给一个日期加减一定天数，可以通过加减秒数实现，如

```
as.POSIXct(c('1998-03-16 13:15:45')) + 3600*24*2
## [1] "1998-03-18 13:15:45 CST"
```

这个例子把日期推后了两天。

用 `difftime(time1, time2, units='days')` 计算 time1 减去 time2 的天数，如

```
x <- as.POSIXct(c('1998-03-16', '2015-11-22'))
c(difftime(x[2], x[1], units='days'))
## Time difference of 6460 days
```

函数结果用 `c()` 包裹以转换为数值, 否则会带有单位。

调用 `difftime()` 时如果前两个自变量中含有时间部分，则间隔天数也会带有小数部分。如

```
x <- as.POSIXct(c('1998-03-16 13:15:45', '2015-11-22 9:45:3'))
c(difftime(x[2], x[1], units='days'))
## Time difference of 6459.854 days
```

`difftime()` 中 `units` 选项还可以取为 `'secs'`, `'mins'`, `'hours'`等。

9.11 练习

设文件dates.csv中包含如下内容：

```
"出生日期","发病日期"
"1941/3/8","2007/1/1"
"1972/1/24","2007/1/1"
"1932/6/1","2007/1/1"
"1947/5/17","2007/1/1"
"1943/3/10","2007/1/1"
"1940/1/8","2007/1/1"
"1947/8/5","2007/1/1"
"2005/4/14","2007/1/1"
"1961/6/23","2007/1/2"
"1949/1/10","2007/1/2"
```

把这个文件读入为 R 数据框 dates.tab，运行如下程序定义 `date1` 和 `date2` 变量:

```
date1 <- dates.tab[,'出生日期']
date2 <- dates.tab[,'发病日期']
```

(1) 把 date1、date2 转换为 R 的 POSIXct 日期型。

(2) 求 date1 中的各个出生年。

(3) 计算发病时的年龄，以周岁论（过生日才算）。

(4) 把 date2 中发病年月转换为'monyy' 格式，这里 mon 是如 FEB 这样英文三字母缩写，yy 是两数字的年份。

(5) 对诸如'FEB91', 'OCT15' 这样的年月数据，假设 00—20 表示 21 世纪年份，21—99 表示 20 实际年份。编写 R 函数，输入这样的字符型向量，返回相应的 POSIXct 格式日期，具体日期都取为相应月份的 1 号。这个习题和后两个习题可以预习函数部分来做。

(6) 对 R 的 POSIXct 日期，写函数转换成'FEB91', 'OCT15' 这样的年月表示，假设 00—20 表示 21 世纪年份，21—99 表示 20 实际年份。

(7) 给定两个 POSIXct 日期向量 birth 和 work，birth 为生日，work 是入职日期，编写 R 函数，返回相应的入职周岁整数值（不到生日时周岁值要减一）。

Chapter 10

R 因子类型

10.1 因子

R 中用因子代表数据中分类变量, 如性别、省份、职业。有序因子代表有序量度，如打分结果，疾病严重程度等。

用 `factor()` 函数把字符型向量转换成因子，如

```
x <- c(" 男", " 女", " 男", " 男",  " 女")
sex <- factor(x)
sex
## [1] 男 女 男 男 女
## Levels: 男 女
attributes(sex)
## $levels
## [1] " 男" " 女"
##
## $class
## [1] "factor"
```

因子有 class 属性，取值为`"factor"`，还有一个 `levels`(水平值) 属性，此属性可以用 `levels()` 函数访问，如

```
levels(sex)
## [1] " 男" " 女"
```

因子的 `levels` 属性可以看成是一个映射，把整数值 1,2,... 映射成这些水平值，因子在保存时会保存成整数值 1,2,... 等与水平值对应的编号。这样可以节省存储空间，在建模计算的程序中也比较有利于进行数学运算。

事实上，`read.csv()` 函数的默认操作会把输入文件的字符型列自动转换成因子，这对于性别、职业、地名这样的列是合适的，但是对于姓名、日期、详细地址这样的列则不合适。所以，在 `read.csv()` 调用中经常加选项

stringsAsFactors=FALSE 选项禁止这样的自动转换，还可以用 colClasses 选项逐个指定每列的类型。建议改用 readr 包的 read_csv() 函数，这个函数读入 CSV 时不自动转换因子，生成 data.frame 的替代类型 tibble。

用 as.numeric() 可以把因子转换为纯粹的整数值，如

```
as.numeric(sex)
## [1] 1 2 1 1 2
```

因为因子实际保存为整数值，所以对因子进行一些字符型操作可能导致错误。用 as.character() 可以把因子转换成原来的字符型，如

```
as.character(sex)
## [1] "男" "女" "男" "男" "女"
```

为了对因子执行字符型操作（如取子串），保险的做法是先用 as.character() 函数强制转换为字符型。

factor() 函数的一般形式为

```
factor(x, levels = sort(unique(x), na.last = TRUE),
       labels, exclude = NA, ordered = FALSE)
```

可以用选项 levels 自行指定各水平值, 不指定时由 x 的不同值来求得。可以用选项 labels 指定各水平的标签, 不指定时用各水平值的对应字符串。可以用 exclude 选项指定要转换为缺失值 (NA) 的元素值集合。如果指定了 levels, 则当自变量 x 的某个元素等于第 j 个水平值时输出的因子对应元素值取整数 j, 如果该元素值没有出现在 levels 中则输出的因子对应元素值取 NA。ordered 取真值时表示因子水平是有次序的 (按编码次序)。

在使用 factor() 函数定义因子时，如果知道自变量元素的所有可能取值，应尽可能使用 levels= 参数指定这些不同可能取值，这样，即使某个取值没有出现，此变量代表的含义和频数信息也是完整的。自己指定 levels= 的另一好处是可以按正确的次序显示因子的分类统计值。

因为一个因子的 levels 属性是该因子独有的，所以用 c() 或者 rbind() 合并两个因子有可能造成错误，保险的做法是在定义两个因子时用 levels 参数指定相同的水平值集合。较新版本的 R 软件已经解决了这个问题。

10.2 cut() 函数

连续取值的变量，可以用 cut() 函数将其分段，转换成因子。使用 breaks() 参数指定分点，最小分点要小于数据的最小值，最大分点要大于等于数据的最大值，默认使用左开右闭区间分组，如：

```
cut(1:10, breaks=c(0, 5, 10))
```

```
##  [1] (0,5]  (0,5]  (0,5]  (0,5]  (0,5]  (5,10] (5,10] (5,10] (5,10] (5,10]
## Levels: (0,5] (5,10]
```

可以指定 breaks 为一个正整数，表示将数据范围略扩大后进行等间距分组，如：

```
set.seed(1)
x <- sort(round(rnorm(20), 2))
f <- cut(x, breaks=4); f
```

```
##  [1] (-2.21,-1.26]  (-1.26,-0.305] (-1.26,-0.305] (-1.26,-0.305] (-1.26,-0.305]
##  [6] (-1.26,-0.305] (-0.305,0.647] (-0.305,0.647] (-0.305,0.647] (-0.305,0.647]
## [11] (-0.305,0.647] (-0.305,0.647] (-0.305,0.647] (-0.305,0.647] (0.647,1.6]
## [16] (0.647,1.6]    (0.647,1.6]    (0.647,1.6]    (0.647,1.6]    (0.647,1.6]
## Levels: (-2.21,-1.26] (-1.26,-0.305] (-0.305,0.647] (0.647,1.6]
```

可以修改水平标记，如：

```
levels(f) <- c("a", "b", "c", "d"); f
```

```
##  [1] a b b b b b c c c c c c c c d d d d d d
## Levels: a b c d
```

为了实现各组个数比较平均的分组，可以利用 quantile() 函数计算分位数作为分组，如：

```
cu <- quantile(x, c(0, 1/4, 1/2, 3/4, 1))
cu[1] <- cu[1] - 0.01*(cu[5] - cu[1])
f2 <- cut(x, breaks=cu, oredered_result=TRUE)
levels(f2) <- c("a", "b", "c", "d"); f2
```

```
##  [1] a a a a a b b b b b c c c c c d d d d d
## Levels: a b c d
```

参数 oredered_result=TRUE 表示生成有序因子。

10.3 table() 函数

用 table() 函数统计因子各水平的出现次数（称为频数或频率）。也可以对一般的向量统计每个不同元素的出现次数。如

```
table(sex)
## sex
## 男 女
##  3  2
```

对一个变量用 table 函数计数的结果是一个特殊的有元素名的向量，元素名是自变量的不同取值，结果的元素值是对应的频数。单个因子或单个向量的频数结果可以用向量的下标访问方法取出单个频数或若干个频数的子集。

10.4 tapply() 函数

可以按照因子分组然后每组计算另一变量的概括统计。如

```
h <- c(165, 170, 168, 172, 159)
tapply(h, sex, mean)
##       男       女
## 168.3333 164.5000
```

这里第一自变量 h 与与第二自变量 sex 是等长的，对应元素分别为同一人的身高和性别，tapply() 函数分男女两组计算了身高平均值。

10.5 forcats 包的因子函数

```
library(forcats)
```

在分类变量类数较多时，往往需要对因子水平另外排序、合并等，forcats 包提供了一些针对因子的方便函数。

10.5.1 fct_relevel

有时在因子水平数较多时仅想将特定的一个或几个水平次序放到因子水平最前面,可以用 forcats::fct_relevel() 函数，如：

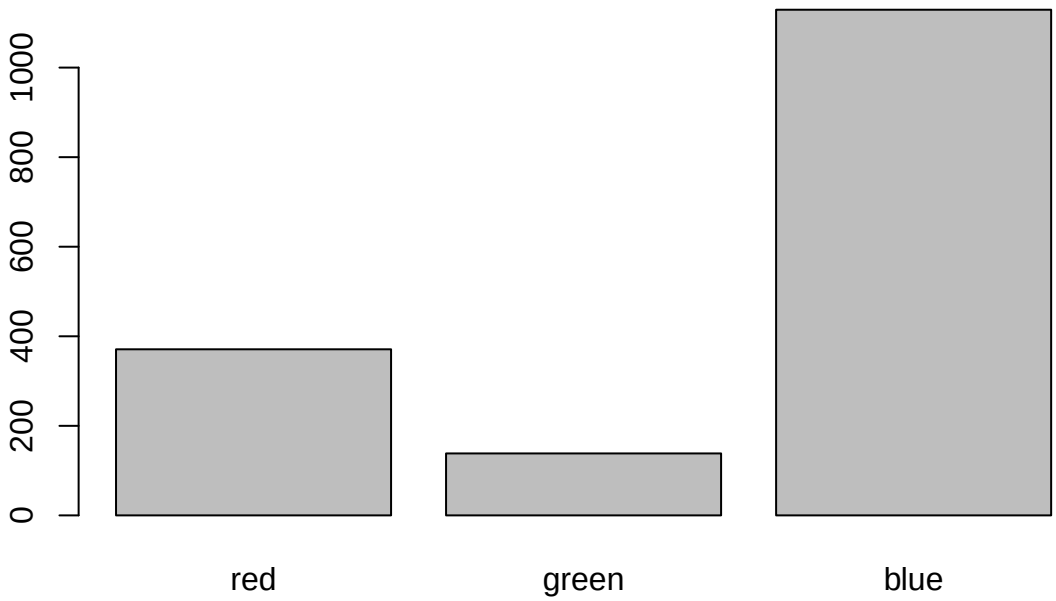
```
set.seed(1)
fac <- sample(c("red", "green", "blue"), 30, replace=TRUE) |>
  factor(levels=c("red", "green", "blue"))
levels(fac)
## [1] "red"   "green" "blue"
fac3 <- fct_relevel(fac, "blue"); levels(fac3)
## [1] "blue"  "red"   "green"
```

fct_relevel() 第一个参数是要修改次序的因子，后续可以有多个字符型参数表示要提前的水平。当然，也可以提供所有因子水平进行次序重排。

10.5.2 fct_reorder

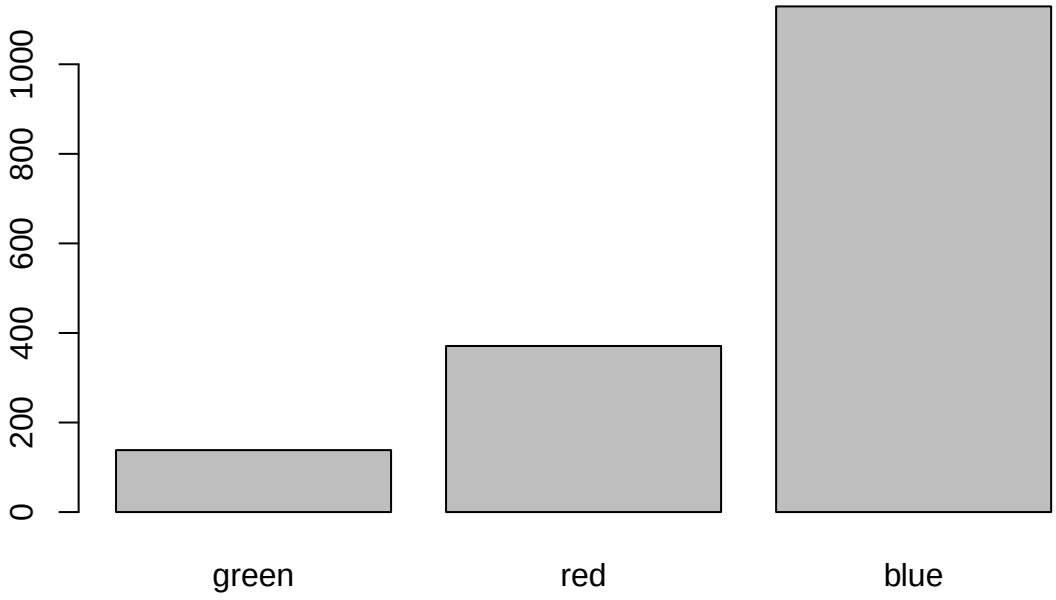
forcats::fct_reorder() 可以根据不同因子水平分成的组中另一数值型变量的统计量值排序。如：

```
set.seed(1)
fac <- sample(c("red", "green", "blue"), 30, replace=TRUE)
fac <- factor(fac, levels=c("red", "green", "blue"))
x <- round(100*(10+rt(30,2)))
res1 <- tapply(x, fac, sd); res1
##       red     green      blue
##  370.9222  138.3185 1129.2587
barplot(res1)
```



如果希望按照统计量次序对因子排序，可以用 `forcats::fct_reorder()` 函数。下面的程序将因子按变量 `x` 的标准差排序：

```
fac2 <- fct_reorder(fac, x, sd)
res2 <- tapply(x, fac2, sd)
barplot(res2)
```



`fct_reorder` 如果不指定按什么统计量排序，就默认使用中位数。

`forcats::fct_reorder2(f, x, y)` 也调整因子 `f` 的水平的次序，但是根据与每组中最大的 x 值相对应的 y 值大小调整次序，这样在作多个因子水平对应的多条曲线图时，因子次序按照每条曲线最右端的的纵坐标排列，而曲线颜色的图例也放在图的右侧，可以比较容易地将图例颜色与曲线颜色对应起来。

比如，下面的作图用了 ggplot2 包（见第27章）：

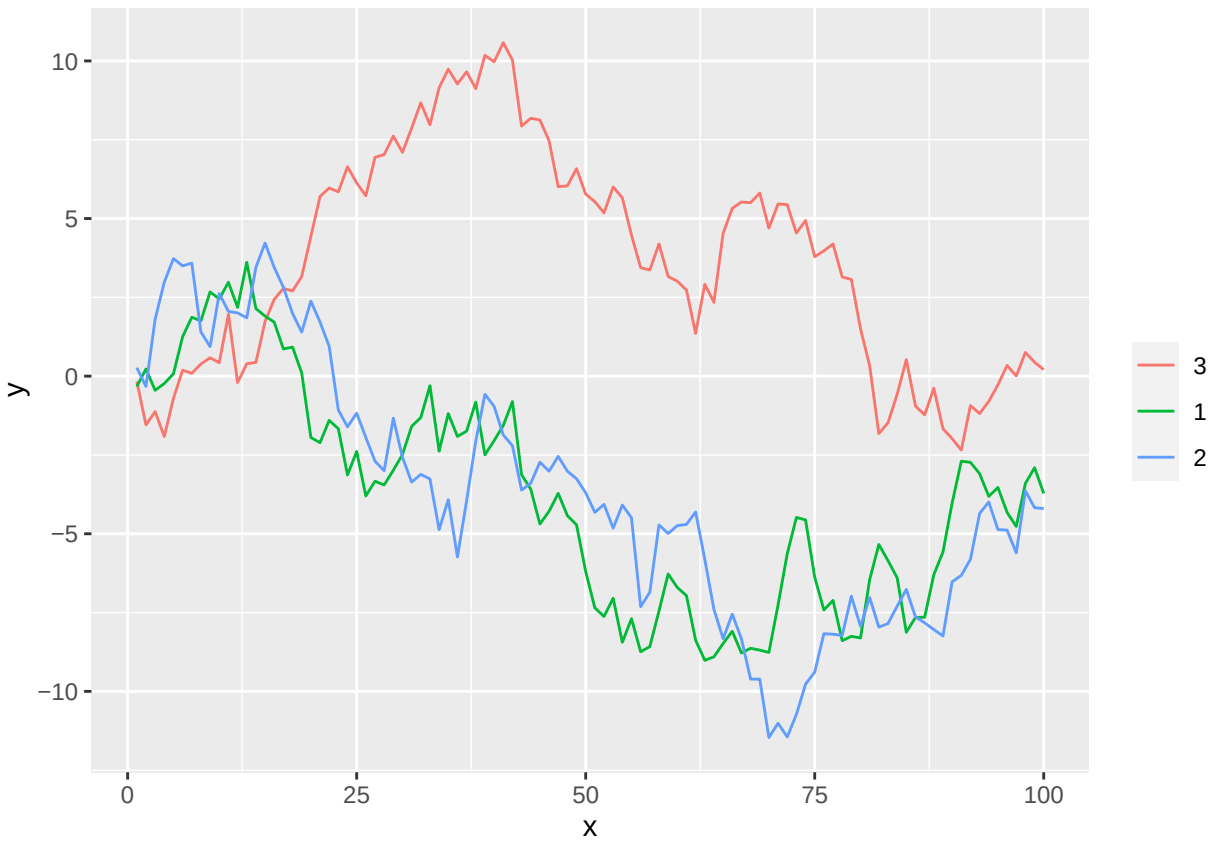
```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6           v purrr   1.0.0
## v tibble  3.1.8           v dplyr   1.0.99.9000
## v tidyr   1.2.1.9001      v stringr 1.5.0.9000
## v readr   2.1.2

## Warning: package 'purrr' was built under R version 4.2.2

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
set.seed(101)
d.rd <- tibble(
  x = rep(1:100, 3),
  f = factor(rep(1:3, each=100)),
  y = c(cumsum(rnorm(100)), cumsum(rnorm(100)), cumsum(rnorm(100)))
)
ggplot(d.rd, aes(
  x = x,
  y = y,
  color = forcats::fct_reorder2(f, x, y))) +
  geom_line() +
  scale_color_discrete(name="")
```



可以看出三条曲线右端的次序与图例的次序一致。

10.5.3 fct_infreq

为了将因子水平按照频数从高到低重新排序，可以用 `fct_infreq(f)`。如果需要按频数从低到高排序，可以用 `fct_rev(fct_infreq(f))`。

10.5.4 fct_recode

forcats::fct_recode() 可以修改每个水平的名称，如：

```
fac4 <- fct_recode(
  fac,
  " 红"="red",
  " 绿"="green",
  " 蓝"="blue")
table(fac4)
## fac4
## 红 绿 蓝
## 13 10  7
```

其中的格式是新水平名 = 老水平名。

fct_recode() 在修改水平名时允许多个旧水平对应到一个新水平，从而合并原来的水平。如果合并很多，可以用 fct_collapse() 函数，如

```
compf <- fct_collapse(
  comp,
  " 其它"=c("", " 无名", " 无应答"),
  " 联想"=c(" 联想", " 联想集团"),
  " 百度"=c(" 百度", " 百度集团")
  )
```

如果某个因子有很多个类别的观测频数很少,在统计时有过多水平不易展示主要的类别,可以用 forcats::fct_lump_n(f, n) 合并，用参数 n 指定要保留多少个类。类似函数有 forcats::fct_lump_min(f, min)，合并频数小于指定界限的类；forcats::fct_lump_prop(f, prop)，合并小于指定比例的类。

函数 forcats::fct_lump_lowfreq(f) 可以从最少的类合并一直到 “其它” 类超过未被合并的最小类之前。

10.6 练习

设文件class.csv中包含如下内容:

```
name,sex,age,height,weight
Alice,F,13,56.5,84
Becka,F,13,65.3,98
Gail,F,14,64.3,90
Karen,F,12,56.3,77
```

```
Kathy,F,12,59.8,84.5
Mary,F,15,66.5,112
Sandy,F,11,51.3,50.5
Sharon,F,15,62.5,112.5
Tammy,F,14,62.8,102.5
Alfred,M,14,69,112.5
Duke,M,14,63.5,102.5
Guido,M,15,67,133
James,M,12,57.3,83
Jeffrey,M,13,62.5,84
John,M,12,59,99.5
Philip,M,16,72,150
Robert,M,12,64.8,128
Thomas,M,11,57.5,85
William,M,15,66.5,112
```

用如下程序把该文件读入为 R 数据框 d.class, 其中的 sex 列已经自动转换为因子。取出其中的 sex 和 age 列到变量 sex 和 age 中

```r
d.class <- read.csv('class.csv', header=TRUE)
sex <- d.class[,'sex']
age <- d.class[,'age']
```

(1) 统计并显示列出 sex 的不同值频数；

(2) 分男女两组分别求年龄最大值；

(3) 把 sex 变量转换为一个新的因子，F 显示成 “Female”，M 显示成 “Male”。

Chapter 11

列表类型

11.1 列表

R 中列表 (list) 类型来保存不同类型的数据。一个主要目的是提供 R 分析结果输出包装：输出一个变量，这个变量包括回归系数、预测值、残差、检验结果等等一系列不能放到规则形状数据结构中的内容。实际上，数据框也是列表的一种，但是数据框要求各列等长，而列表不要求。

列表可以有多个元素，但是与向量不同的是，列表的不同元素的类型可以不同，比如，一个元素是数值型向量，一个元素是字符串，一个元素是标量，一个元素是另一个列表。

定义列表用函数 `list()`, 如

```
rec <- list(name=" 李明", age=30,
  scores=c(85, 76, 90))
rec
## $name
## [1] " 李明"
##
## $age
## [1] 30
##
## $scores
## [1] 85 76 90
```

用 `typeof()` 函数判断一个列表，返回结果为 `list`。可以用 `is.list()` 函数判断某个对象是否列表类型。

为了生成元素为空的长度为 `n` 的列表，程序如：

```
vector("list", n)
```

11.2 列表元素访问

列表的一个元素也可以称为列表的一个“变量”，单个列表元素必须用两重方括号格式访问，如

```
rec[[3]]
## [1] 85 76 90
rec[[3]][2]
## [1] 76
rec[["age"]]
## [1] 30
```

列表的单个元素也可以用 $ 格式访问，如

```
rec$age
## [1] 30
```

如果使用单重方括号对列表取子集，结果还是列表而不是列表元素，如

```
rec[3]
## $scores
## [1] 85 76 90
is.list(rec[3])
## [1] TRUE
```

列表一般都应该有元素名，元素名可以看成是变量名，列表中的每个元素看成一个变量。用 names() 函数查看和修改元素名。如

```
names(rec)
## [1] "name"   "age"    "scores"
names(rec)[names(rec)=="scores"] <- " 三科分数"
names(rec)
## [1] "name"     "age"      " 三科分数"
rec[[" 三科分数"]]
## [1] 85 76 90
```

可以修改列表元素内容。如

```
rec[[" 三科分数"]][2] <- 0
print(rec)
## $name
## [1] " 李明"
##
## $age
## [1] 30
##
## $ 三科分数
## [1] 85  0 90
```

直接给列表不存在的元素名定义元素值就添加了新元素，而且不同于使用向量，对于列表而言这是很正常的做法，比如

```
rec[[" 身高"]] <- 178
print(rec)
```

```
## $name
## [1] "李明"
##
## $age
## [1] 30
##
## $三科分数
## [1] 85  0 90
##
## $身高
## [1] 178
```

把某个列表元素赋值为 NULL 就删掉这个元素。如

```
rec[["age"]] <- NULL
print(rec)
## $name
## [1] " 李明"
##
## $ 三科分数
## [1] 85  0 90
##
```

```
## $ 身高
## [1] 178
```

在 list() 函数中允许定义元素为 NULL，这样的元素是存在的，如：

```
li <- list(a=120, b="F", c=NULL); li
## $a
## [1] 120
##
## $b
## [1] "F"
##
## $c
## NULL
```

但是，要把已经存在的元素修改为 NULL 值而不是删除此元素，或者给列表增加一个取值为 NULL 的元素，这时需要用单重的方括号取子集，这样的子集会保持其列表类型，给这样的子列表赋值为 list(NULL)，如：

```
li["b"] <- list(NULL)
li["d"] <- list(NULL)
li
## $a
## [1] 120
##
## $b
## NULL
##
## $c
## NULL
##
## $d
## NULL
```

11.3 列表类型转换

用 as.list() 把一个其它类型的对象转换成列表；用 unlist() 函数把列表转换成基本向量。如

```
li1 <- as.list(1:3)
li1
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
li2 <- list(x=1, y=c(2,3))
unlist(li2)
##  x y1 y2
##  1  2  3
```

11.4 连接列表

两个列表可以用 c 连接成一个列表，如：

```
li1 <- list(1:2)
li2 <- list(11:12)
li <- c(li1, li2)
li
## [[1]]
## [1] 1 2
##
## [[2]]
## [1] 11 12
```

如下的做法没有达到目的：

```
li <- list()
li <- c(li, 1:3)
li <- c(li, 11:13)
li
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
```

```
## [[3]]
## [1] 3
##
## [[4]]
## [1] 11
##
## [[5]]
## [1] 12
##
## [[6]]
## [1] 13
```

需要改为：

```
li <- list()
li <- c(li, list(1:3))
li <- c(li, list(11:13))
li
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] 11 12 13
```

11.5 返回列表的函数示例–strsplit()

strsplit() 输入一个字符型向量并指定一个分隔符，返回一个项数与字符型向量元素个数相同的列表，列表每项对应于字符型向量中一个元素的拆分结果。如

```
x <- c("10, 8, 7", "5, 2, 2", "3, 7, 8", "8, 8, 9")
res <- strsplit(x, ","); res
## [[1]]
## [1] "10" " 8" " 7"
##
## [[2]]
## [1] "5"  " 2" " 2"
##
## [[3]]
## [1] "3"  " 7" " 8"
```

```
##
## [[4]]
## [1] "8"  " 8" " 9"
```

为了把拆分结果进一步转换成一个数值型矩阵，可以使用 `sapply()` 函数如下：

```
t(sapply(res, as.numeric))
##      [,1] [,2] [,3]
## [1,]   10    8    7
## [2,]    5    2    2
## [3,]    3    7    8
## [4,]    8    8    9
```

`sapply()` 函数是 apply 类函数之一，稍后再详细进行讲解。

Chapter 12

R 矩阵和数组

12.1 R 矩阵

矩阵用 matrix 函数定义，实际存储成一个向量，根据保存的行数和列数对应到矩阵的元素，存储次序为按列存储。定义如

```
A <- matrix(11:16, nrow=3, ncol=2); print(A)
```

```
##      [,1] [,2]
## [1,]   11   14
## [2,]   12   15
## [3,]   13   16
```

```
B <- matrix(c(1,-1, 1,1), nrow=2, ncol=2, byrow=TRUE); print(B)
```

```
##      [,1] [,2]
## [1,]    1   -1
## [2,]    1    1
```

`matrix()` 函数把矩阵元素以一个向量的形式输入，用 `nrow` 和 `ncol` 规定行数和列数，向量元素填入矩阵的缺省次序是按列填入，用 `byrow=TRUE` 选项可以转换成按行填入。

用 `nrow()` 和 `ncol()` 函数可以访问矩阵的行数和列数，如

```
nrow(A)
```

```
## [1] 3
```

```
ncol(A)
```

```
## [1] 2
```

矩阵有一个 dim 属性，内容是两个元素的向量，两个元素分别为矩阵的行数和列数。dim 属性可以用 `dim()` 函数访问。如

```
attributes(A)
```

```
## $dim
## [1] 3 2
```

```
dim(A)
```

```
## [1] 3 2
```

函数 `t(A)` 返回 `A` 的转置。

12.2 矩阵子集

用 `A[1,]` 取出 A 的第一行，变成一个普通向量。用 `A[,1]` 取出 A 的第一列，变成一个普通向量。用 `A[c(1,3),1:2]` 取出指定行、列对应的子矩阵。如

```
A
```

```
##      [,1] [,2]
## [1,]   11   14
## [2,]   12   15
## [3,]   13   16
```

```
A[1,]
```

```
## [1] 11 14
```

```
A[,1]
```

```
## [1] 11 12 13
```

```
A[c(1,3), 1:2]
```

```
##      [,1] [,2]
## [1,]   11   14
## [2,]   13   16
```

用 colnames() 函数可以给矩阵每列命名，也可以访问矩阵列名，用 rownames() 函数可以给矩阵每行命名，也可以访问矩阵行名。如

```
colnames(A) <- c('X', 'Y')
rownames(A) <- c('a', 'b', 'c')
A
```

```
##    X  Y
## a 11 14
## b 12 15
## c 13 16
```

矩阵可以有一个 dimnames 属性，此属性是两个元素的列表（列表见稍后部分的介绍），两个元素分别为矩阵的行名字符型向量与列名字符型向量。如果仅有其中之一，缺失的一个取为 NULL。

有了列名、行名后，矩阵下标可以用字符型向量，如

```
A[,'Y']
```

```
##  a  b  c
## 14 15 16
```

```
A['b',]
```

```
##  X  Y
## 12 15
```

```
A[c('a', 'c'), 'Y']
```

```
##  a  c
## 14 16
```

注意在对矩阵取子集时，如果取出的子集仅有一行或仅有一列，结果就不再是矩阵而是变成了 R 向量，R 向量既不是行向量也不是列向量。如果想避免这样的规则起作用，需要在方括号下标中加选项 drop=FALSE，如

```
A[,1,drop=FALSE]
```

```
##    X
## a 11
## b 12
## c 13
```

取出了 A 的第一列，作为列向量取出，所谓列向量实际是列数等于 1 的矩阵。如果用常量作为下标，其结果维数是确定的，不会出问题；如果用表达式作为下标，则表达式选出零个、一个、多个下标，结果维数会有不同，加 drop=FALSE 则是安全的做法。

矩阵也可以用逻辑下标取子集，比如

```
A
```

```
##    X  Y
## a 11 14
## b 12 15
## c 13 16
```

```
A[A[,1]>=12,'Y']
```

```
##  b  c
## 15 16
```

矩阵本质上是一个向量添加了 dim 属性，实际保存还是保存成一个向量，其中元素的保存次序是按列填入，所以，也可以向对一个向量取子集那样，仅用一个正整数向量的矩阵取子集。如

```
A
```

```
##    X  Y
## a 11 14
## b 12 15
## c 13 16
```

```
A[c(1,3,5)]
```

```
## [1] 11 13 15
```

为了挑选矩阵的任意元素组成的子集而不是子矩阵，可以用一个两列的矩阵作为下标，矩阵的每行的两个元素分别指定一个元素的行号和列号。如

```
ind <- matrix(c(1,1, 2,2, 3,2), ncol=2, byrow=TRUE)
A
```

```
##    X  Y
## a 11 14
## b 12 15
## c 13 16
```

```
ind
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    2
## [3,]    3    2
```

```
A[ind]
```

```
## [1] 11 15 16
```

用 c(A) 或 A[] 返回矩阵 A 的所有元素。如果要修改矩阵 A 的所有元素，可以对 A[] 赋值。

对矩阵 A，diag(A) 访问 A 的主对角线元素组成的向量。另外，若 x 为正整数值标量，diag(x) 返回 x 阶单位阵；若 x 为长度大于 1 的向量，diag(x) 返回以 x 的元素为主对角线元素的对角矩阵。

12.3 cbind() 和 rbind() 函数

若 x 是向量，cbind(x) 把 x 变成列向量，即列数为 1 的矩阵，rbind(x) 把 x 变成行向量。

若 x1, x2, x3 是等长的向量，cbind(x1, x2, x3) 把它们看成列向量并在一起组成一个矩阵。cbind() 的自变量可以同时包含向量与矩阵，向量的长度必须与矩阵行数相等。如

```
cbind(c(1,2), c(3,4), c(5,6))
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
cbind(A, c(1,-1,10))
```

```
##    X  Y
## a 11 14  1
## b 12 15 -1
## c 13 16 10
```

cbind() 的自变量中也允许有标量，这时此标量被重复使用。如

```
cbind(1, c(1,-1,10))
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    1   -1
## [3,]    1   10
```

rbind() 用法类似，可以等长的向量看成行向量上下摞在一起，可以是矩阵与长度等于矩阵列数的向量上下摞在一起，向量长度为 1 也可以。

12.4 矩阵运算

12.4.1 四则运算

矩阵可以与标量作四则运算，结果为每个元素进行相应运算，如

```
A
```

```
##    X  Y
## a 11 14
## b 12 15
## c 13 16
```

```
C1 <- A + 2; C1
```

```
##    X  Y
## a 13 16
## b 14 17
## c 15 18
```

```
C2 <- A / 2; C2
```

```
##     X   Y
## a 5.5 7.0
## b 6.0 7.5
## c 6.5 8.0
```

当运算为矩阵乘以一个标量时，就是线性代数中的矩阵的数乘运算。

两个同形状的矩阵进行加、减运算，即对应元素相加、相减，用 `A + B`，`A - B` 表示，如

```
C1 + C2
```

```
##      X    Y
## a 18.5 23.0
## b 20.0 24.5
## c 21.5 26.0
```

```
C1 - C2
```

```
##     X    Y
## a 7.5  9.0
## b 8.0  9.5
## c 8.5 10.0
```

这就是线性代数中矩阵的加、减运算。

对两个同形状的矩阵，用 `*` 表示两个矩阵对应元素相乘 (注意这不是线性代数中的矩阵乘法)，用/表示两个矩阵对应元素相除。如

```
C1 * C2
```

```
##      X     Y
## a 71.5 112.0
## b 84.0 127.5
## c 97.5 144.0
```

```
C1 / C2
```

```
##          X        Y
## a 2.363636 2.285714
## b 2.333333 2.266667
## c 2.307692 2.250000
```

12.4.2 矩阵乘法

用%*% 表示矩阵乘法而不是用 * 表示，注意矩阵乘法要求左边的矩阵的列数等于右边的矩阵的行数。如

```
A
```

```
##    X  Y
## a 11 14
## b 12 15
## c 13 16
```

```
B
```

```
##      [,1] [,2]
## [1,]    1   -1
## [2,]    1    1
```

```
C3 <- A %*% B; C3
```

```
##   [,1] [,2]
## a   25    3
## b   27    3
## c   29    3
```

12.4.3 向量与矩阵相乘

矩阵与向量进行乘法运算时，向量按需要解释成列向量或行向量。当向量左乘矩阵时，看成行向量；当向量右乘矩阵时，看成列向量。如

```
B
```

```
##      [,1] [,2]
## [1,]    1   -1
## [2,]    1    1
```

```
c(1,1) %*% B
```

```
##      [,1] [,2]
## [1,]    2    0
```

```
B %*% c(1,1)
```

```
##      [,1]
## [1,]    0
## [2,]    2
```

```
c(1,1) %*% B %*% c(1,1)
```

```
##      [,1]
## [1,]    2
```

注意矩阵乘法总是给出矩阵结果，即使此矩阵已经退化为行向量、列向量甚至于退化为标量也是一样。如果需要，可以用 `c()` 函数把一个矩阵转换成按列拉直的向量。

12.4.4 内积

设 `x`, `y` 是两个向量，计算向量内积，可以用 `sum(x*y)` 表示。

设 A, B 是两个矩阵，A^TB 是广义的内积，也称为叉积 (crossprod)，结果是一个矩阵，元素为 A 的每列与 B 的每列计算内积的结果。A^TB 在 R 中可以表示为 `crossprod(A, B)`, A^TA 可以表示为 `crossprod(A)`。要注意的是，`crossprod()` 的结果总是矩阵，所以计算两个向量的内积用 `sum(x,y)` 而不用 `crossprod(x,y)`。

12.4.5 外积

R 向量支持外积运算，记为`%o%`, 结果为矩阵。`x %o% y` 的第 i 行第 j 列元素等于 x[i] 乘以 y[j]。如

```
c(1,2,3) %o% c(1, -1)
```

```
##      [,1] [,2]
## [1,]    1   -1
## [2,]    2   -2
## [3,]    3   -3
```

这种运算还可以推广到 `x` 的每一元素与 `y` 的每一元素进行其它的某种运算，而不限于乘积运算，可以用 `outer(x,y,f)` 完成，其中 `f` 是某种运算，或者接受两个自变量的函数。

12.5 逆矩阵与线性方程组求解

用 `solve(A)` 求 `A` 的逆矩阵，如

```
solve(B)
```

```
##      [,1] [,2]
## [1,]  0.5  0.5
## [2,] -0.5  0.5
```

用 `solve(A,b)` 求解线性方程组 $Ax = b$ 中的 x, 如

```
solve(B, c(1,2))
```

```
## [1] 1.5 0.5
```

求解了线性方程组

$$\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} x = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

12.6 apply() 函数

`apply(A, 2, FUN)` 把矩阵 `A` 的每一列分别输入到函数 FUN 中，得到对应于每一列的结果，如

```
D <- matrix(c(6,2,3,5,4,1), nrow=3, ncol=2); D
```

```
##      [,1] [,2]
## [1,]    6    5
## [2,]    2    4
## [3,]    3    1
```

```
apply(D, 2, sum)
```

```
## [1] 11 10
```

`apply(A, 1, FUN)` 把矩阵 `A` 的每一行分别输入到函数 FUN 中，得到与每一行对应的结果，如

```
apply(D, 1, mean)
```

```
## [1] 5.5 3.0 2.0
```

如果函数 FUN 返回多个结果，则 `apply(A, 2, FUN)` 结果为矩阵，矩阵的每一列是输入矩阵相应列输入到 FUN 的结果，结果列数等于 `A` 的列数。如

```
apply(D, 2, range)
```

```
##      [,1] [,2]
## [1,]    2    1
## [2,]    6    5
```

如果函数 FUN 返回多个结果，为了对每行计算 FUN 的结果，结果存入一个与输入的矩阵行数相同的矩阵，应该用 `t(apply(A, 1, FUN))` 的形式，如

```
t(apply(D, 1, range))
```

```
##      [,1] [,2]
## [1,]    5    6
## [2,]    2    4
## [3,]    1    3
```

12.7 多维数组

矩阵是多维数组 (array) 的特例。矩阵是 $x_{ij}, i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, m$ 这样的两下标数据的存贮格式，三维数组是 $x_{ijk}, i = 1, 2, \ldots, n,\ j = 1, 2, \ldots, m,\ k = 1, 2, \ldots, p$ 这样的三下标数据的存贮格式，s 维数组则是有 s 个下标的数据的存贮格式。实际上，给一个向量添加一个 `dim` 属性就可以把它变成多维数组。

多维数组的一般定义语法为

```
数组名 <- array(数组元素,
  dim=c(第一下标个数, 第二下标个数, ..., 第s下标个数))
```

其中数组元素的填入次序是第一下标变化最快，第二下标次之，最后一个下标是变化最慢的。这种次序称为 FORTRAN 次序。

下面是一个三维数组定义例子。

```
ara <- array(1:24, dim=c(2,3,4)); ara
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
```

```
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,]   13   15   17
## [2,]   14   16   18
##
## , , 4
##
##      [,1] [,2] [,3]
## [1,]   19   21   23
## [2,]   20   22   24
```

这样的数组保存了 $x_{ijk}, i=1,2,\ j=1,2,3,\ k=1,2,3,4$。三维数组 ara 可以看成是 4 个 2×3 矩阵。取出其中一个如 `ara[,,2]`(取出第二个矩阵)

```
ara[,,2]
```

```
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
```

多维数组可以利用下标进行一般的子集操作，比如 `ara[,2, 2:3]` 是 $x_{ijk}, i=1,2,\ j=2,\ k=2,3$ 的值，结果是一个 2×2 矩阵:

```
ara[,2,2:3]
```

```
##      [,1] [,2]
## [1,]    9   15
## [2,]   10   16
```

多维数组在取子集时如果某一维下标是标量，则结果维数会减少，可以在方括号内用 `drop=FALSE` 选项避免这样的规则发生作用。

类似于矩阵，多维数组可以用一个矩阵作为下标，如果是三维数组，矩阵就需要有 3 列，四维数组需要用 4 列矩阵。下标矩阵的每行对应于一个数组元素。

12.8 列表类型的数组

矩阵一般用于数值计算，所以其元素一般是数值。但是，在更复杂的编程场景中，可能需要在矩阵（或数组）元素中保存复杂结构，比如，每个元素是一个长度可变的向量。这时，可以将矩阵元素设为列表类型，但访问其元素需要使用部分列表访问语法格式。

例如，生成未初始化的列表类型的矩阵：

```
L <- matrix(list(), nrow=2, ncol=2); L
```

```
##      [,1] [,2]
## [1,] NULL NULL
## [2,] NULL NULL
```

给每个矩阵元素赋值为仅有一个元素的列表：

```
L[1,1] <- list(1:2)
L[1,2] <- list(3:5)
L[2,1] <- list(6:10)
L[2,2] <- list(11:15)
L
```

```
##      [,1]      [,2]
## [1,] integer,2 integer,3
## [2,] integer,5 integer,5
```

元素为仅一个元素的列表：

```
L[2,2]
```

```
## [[1]]
## [1] 11 12 13 14 15
```

为了访问元素的实际内容，需要继续用 `[[]]` 格式：

```
L[2,2][[1]]
```

```
## [1] 11 12 13 14 15
```

但是，这样每个矩阵元素必须为长度 1 的列表。下面的程序会出错

```
L[1,1] <- list(101:102, 103:104)
## Error in L[1, 1] <- list(101:102, 103:104) :
##   number of items to replace is not
##   a multiple of replacement length
```

需要写成：

```
L[1,1][[1]] <- list(101:102, 103:104)
L[1,1]
```

```
## [[1]]
## [[1]][[1]]
## [1] 101 102
##
## [[1]][[2]]
## [1] 103 104
```

这样，$(1,1)$ 元素的实际内容仍是一个列表：

```
L[1,1][[1]]
```

```
## [[1]]
## [1] 101 102
##
## [[2]]
## [1] 103 104
```

访问其中的内容：

```
L[1,1][[1]][[1]]
```

```
## [1] 101 102
```

所以，如果矩阵元素是长度为 1 的列表，则只要给 `L[1,1]` 赋值为一个长度为 1 的列表，访问元素内容时用 `L[1,1][[1]]` 的语法。如果需要在矩阵元素中保存长度大于 1 的列表，就需要将元素的实际内容保存成一个长度为大于 1 的列表，如上例，这时 `L[1,1][[1]]` 是长度大于 1 的列表。

Chapter 13

数据框

13.1 数据框

13.1.1 数据框定义

统计分析中最常见的原始数据形式是类似于数据库表或 Excel 数据表的形式。这样形式的数据在 R 中叫做数据框 (data.frame)。数据框类似于一个矩阵，有 n 个横行、p 个纵列，但各列允许有不同类型：数值型向量、因子、字符型向量、日期时间向量。同一列的数据类型相同。在 R 中数据框是一个特殊的列表，其每个列表元素都是一个长度相同的向量。事实上，数据框还允许一个元素是一个矩阵，但这样会使得某些读入数据框的函数发生错误。

函数 `data.frame()` 可以生成数据框，如

```
d <- data.frame(
    name=c(" 李明", " 张聪", " 王建"),
    age=c(30, 35, 28),
    height=c(180, 162, 175),
    stringsAsFactors=FALSE)
print(d)
```

```
##   name age height
## 1 李明  30    180
## 2 张聪  35    162
## 3 王建  28    175
```

`data.frame()` 函数会将字符型列转换成因子，加选项 `stringsAsFactors=FALSE` 可以避免这样的转换。

如果数据框的某一列为常数，可以在 `data.frame()` 调用中仅给该列赋一个值，生成的结果会自动重复这个值使得该列与其他列等长。

nrow(d) 求 d 的行数，ncol(d) 或 length(d) 求 d 的列数。数据框每列叫做一个变量，每列都有名字，称为列名或变量名，可以用 names() 函数和 colnames() 函数访问。如

```
names(d)
## [1] "name"   "age"    "height"
colnames(d)
## [1] "name"   "age"    "height"
```

给 names(d) 或 colnames(d) 赋值可以修改列名。

用 as.data.frame(x) 可以把 x 转换成数据框。如果 x 是一个向量，转换结果是以 x 为唯一一列的数据框。如果 x 是一个列表并且列表元素都是长度相同的向量，转换结果中每个列表变成数据框的一列。如果 x 是一个矩阵，转换结果把矩阵的每列变成数据框的一列。

数据框是一个随着 R 语言前身 S 语言继承下来的概念，现在已经有一些不足之处，tibble 包提供了 tibble 类，这是数据框的一个改进版本。见13.2。

13.1.2 数据框显示

在 R markdown 文件中，可以将数据框保存的表格显示为富文本格式，方法是使用 knitr::kable() 函数。详见A.4。

程序示例如：

```
knitr::kable(d)
```

name	age	height
李明	30	180
张聪	35	162
王建	28	175

13.1.3 数据框内容访问

数据框可以用矩阵格式访问，如

```
d[2,3]
## [1] 162
```

访问单个元素。

```
d[[2]]
## [1] 30 35 28
```

访问第二列，结果为向量。

```
d[,2]
## [1] 30 35 28
```

也访问第二列，但是这种作法与 tibble 不兼容，所以应避免使用。

按列名访问列可用如

```
d[["age"]]
## [1] 30 35 28
d[,"age"]
## [1] 30 35 28
d$age
## [1] 30 35 28
```

其中 `d[,"age"]` 的用法与 tibble 不兼容，应避免使用。

因为数据框的一行不一定是相同数据类型，所以数据框的一行作为子集，结果还是数据框，而不是向量。如

```
x <- d[2,]; x
##   name age height
## 2 张聪  35    162
is.data.frame(x)
## [1] TRUE
```

可以同时取行子集和列子集，如

```
d[1:2, "age"]
## [1] 30 35
d[1:2, c("age", "height")]
##   age height
## 1  30    180
## 2  35    162
d[d[,"age"]>=30,]
##   name age height
## 1 李明  30    180
## 2 张聪  35    162
```

与矩阵类似地是，用如 `d[,"age"]`, `d[,2]` 这样的方法取出的数据框的单个列是向量而不再是数据框。但是，如果取出两列或者两列以上，结果则是数据框。如果取列子集时不能预先知道取出的列个数，则子集结果有可能是向量也有可能是数据框，容易造成后续程序错误。对一般的数据框，可以在取子集的方括号内加上 `drop=FALSE` 选项，

确保取列子集的结果总是数据框。数据框的改进类型 tibble 在用 `d[,ind]` 这种语法取出列子集时保持为 tibble 格式，为了取出 tibble 中的一列作为向量，应该使用 `d[[ind]]` 这样的语法。

对数据框变量名按照字符串与集合进行操作可以实现复杂的列子集筛选，但是建议使用23中所述的借助于 tidyverse 库的做法。

13.1.4 数据框的行名

数据框每一行可以有行名，这在原始的 S 语言和传统的 R 语言中是重要的技术，但是在改进类型 tibble 中则取消了行名，需要用行名实现的功能一般改用 `left_join()` 函数实现。建议新的 R 程序不要再利用数据框的行名功能。

比如，每一行定义行名为身份证号，则可以唯一识别各行。下面的例子以姓名作为行名:

```
rownames(d) <- d$name
d$name <- NULL
d
##      age height
## 李明  30    180
## 张聪  35    162
## 王建  28    175
```

用数据框的行名可以建立一个值到多个值的对应表。比如，有如下的数据框：

```
dm <- data.frame(
  " 年级"=1:6,
  " 出游"=c(0, 2, 2, 2, 2, 1),
  " 疫苗"=c(TRUE, FALSE, FALSE, FALSE, TRUE, FALSE)
)
```

其中“出游”是每个年级安排的出游次数，“疫苗”是该年级有全体无计划免疫注射。把年级变成行名，可以建立年级到出游次数与疫苗注射的对应表：

```
rownames(dm) <- dm[[" 年级"]]
dm[[" 年级"]] <- NULL
```

这样，假设某个社区的小学中抽取的 4 个班的年级为 `c(2,1,1,3)`，其对应的出游和疫苗注射信息可查询如下：

```
ind <- c(2,1,1,3)
dm[as.character(ind),]
##     出游  疫苗
## 2      2 FALSE
```

```
## 1       0  TRUE
## 1.1     0  TRUE
## 3       2 FALSE
```

结果中包含了不必要也不太合适的行名，可以去掉，以上程序改成：

```
ind <- c(2,1,1,3)
xx <- dm[as.character(ind),]
rownames(xx) <- NULL
xx
```

```
##   出游  疫苗
## 1    2 FALSE
## 2    0  TRUE
## 3    0  TRUE
## 4    2 FALSE
```

如果要从多个值建立映射，比如，从省名与县名映射到经度、纬度，可以预先用 `paste()` 函数把省名与县名合并在一起，中间以适当字符（如 '-"）分隔，以这样的合并字符串为行名。

实际上，这个例子可以不用行名而是用 `match()` 函数实现。`match(x, table)` 对 `x` 的每个元素返回其在 `table` 中出现的位置序号。找不到的元素返回 `NA`。如：

```
match(c(12, 15), 11:14)
## [1]  2 NA
```

对于上面的学校年级信息查询的例子，可以首先查找每个班对应的年级在数据框中的行序号，然后再返回这些行组成的数据框：

```
dm <- data.frame(
  " 年级"=1:6,
  " 出游"=c(0, 2, 2, 2, 2, 1),
  " 疫苗"=c(T, F, F, F, T, F)
)
ind <- match(c(2,1,1,3), dm[[" 年级"]]); ind
## [1] 2 1 1 3
dm[ind,]
##     年级 出游  疫苗
## 2      2    2 FALSE
## 1      1    0  TRUE
## 1.1    1    0  TRUE
## 3      3    2 FALSE
```

对于代替数据框的 tibble 类型，如果要实现行名的功能，可以将行名作为单独的一列，然后用 dplyr 包的 `inner_join()`、`left_join()`、`full_join()` 等函数横向合并数据集。参见23.23。

13.1.5 数据框与矩阵的区别

数据框不能作为矩阵参加矩阵运算。需要时，可以用 `as.matrix()` 函数转换数据框或数据框的子集为矩阵。如

```
d2 <- as.matrix(d[,c("age", "height")])
d3 <- crossprod(d2); d3
##          age height
## age     2909  15970
## height 15970  89269
```

这里 `crossprod(A)` 表示 A^TA。

13.2 tibble 类型

13.2.1 生成方法

tibble 类型是一种改进的数据框。readr 包的 `read_csv()` 函数是 `read.csv()` 函数的一个改进版本，它将 CSV 文件读入为 tibble 类型，如文件`class.csv`的读入:

```
library(tibble)
library(readr)
d.class <- read_csv("data/class.csv")
```

```
## Rows: 19 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): name, sex
## dbl (3): age, height, weight
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
d.class
```

```
## # A tibble: 19 x 5
##    name    sex     age height weight
```

```
##    <chr>   <chr> <dbl>  <dbl>  <dbl>
##  1 Alice   F        13   56.5   84
##  2 Becka   F        13   65.3   98
##  3 Gail    F        14   64.3   90
##  4 Karen   F        12   56.3   77
##  5 Kathy   F        12   59.8   84.5
##  6 Mary    F        15   66.5  112
##  7 Sandy   F        11   51.3   50.5
##  8 Sharon  F        15   62.5  112.
##  9 Tammy   F        14   62.8  102.
## 10 Alfred  M        14   69    112.
## 11 Duke    M        14   63.5  102.
## 12 Guido   M        15   67    133
## 13 James   M        12   57.3   83
## 14 Jeffrey M        13   62.5   84
## 15 John    M        12   59     99.5
## 16 Philip  M        16   72    150
## 17 Robert  M        12   64.8  128
## 18 Thomas  M        11   57.5   85
## 19 William M        15   66.5  112
```

tibble 类型的类属依次为 `spec_tbl_df`, `tbl_df`, `tbl`, `data.frame`：

```
class(d.class)
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

用 `as_tibble()` 可以将一个数据框转换为 tibble, dplyr 包提供了 `filter()`、`select()`、`arrange()`、`mutate()` 等函数用来对 tibble 选取行子集、列子集，排序、修改或定义新变量，等等。见23。

可以用 `tibble()` 函数生成小的 tibble，如

```
d.bp <- tibble(
  `序号`=c(1,5,6,9,10,15),
  `收缩压`=c(145, 110, " 未测", 150, " 拒绝", 115))
knitr::kable(d.bp)
```

序号	收缩压
1	145
5	110
6	未测
9	150
10	拒绝
15	115

在调用 `tibble()` 函数时，定义在后面的列可以调用前面的列的值。

用 `tribble` 可以按类似于 CSV 格式输入一个 tibble, 如

```
tribble(
~`序号`,~`收缩压`,
1,145,
5,110,
6,NA,
9,150,
10,NA,
15,115
) |> knitr::kable()
```

序号	收缩压
1	145
5	110
6	NA
9	150
10	NA
15	115

注意 `tribble()` 中数据每行末尾也需要有逗号，最后一行末尾没有逗号。这比较适用于在程序中输入小的数据集。一列中混杂数值型和字符型会出错。

实际上，`readr::read_csv()` 也支持从一个多行字符串直接读入数据，如：

```
readr::read_csv(" 序号, 收缩压
1,145
5,110
6,NA
9,150
10,NA
15,115
") |> knitr::kable()
```

```
## Rows: 6 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (2): 序号, 收缩压
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

序号	收缩压
1	145
5	110
6	NA
9	150
10	NA
15	115

tibble 与数据框的一大区别是在显示时不自动显示所有内容，这样可以避免显示很大的数据框将命令行的所有显示都充满。没有显示的列的列名会罗列在显示下方，显示时每列的类型同时显示，也会显示 tibble 的行列数。可以在 `print()` 用 `n=` 和 `width=` 选项指定要显示的行数和列数。如果指定 `print(df, n=Inf, width=Inf)`，则等同于对数据框的默认显示方式，能够显示整个 tibble，但列数过多时会分页显示。

也可以用 `view(df)` 在一个单独的窗口显示整个 tibble。

tibble 在生成或输入时不自动将字符型列转换为因子。

13.2.2 列子集问题

另外，用 `d[,ind]` 这样的单重的方括号取列子集时，即使仅取一列，从 tibble 取出的一列结果仍是 tibble 而不是向量，为了提取一列为向量应使用双方括号格式或 `$` 格式。因为这个原因有些原来的程序输入 tibble 会出错，这时可以用 `as.data.frame()` 转换成数据框。如：

```
d.bp[," 收缩压"]
```

```
## # A tibble: 6 x 1
##   收缩压
##   <chr>
## 1 145
## 2 110
## 3 未测
## 4 150
## 5 拒绝
## 6 115
```

如下的语法取出一列向量：

```
d.bp[[" 收缩压"]]
## [1] "145"  "110"  " 未测" "150"  " 拒绝" "115"
```

tibble 在定义时不需要列名为合法变量名，但是这样的变量名在作为变量名使用时需要用反单撇号包裹。

13.2.3 行名问题

tibble 默认不使用行名 (rownames)。有行名的数据框用 `as_tibble()` 转换为 tibble 时，可以用 `rownames="变量名"` 选项将行名转换成 tibble 的一列，该列的变量名由选项值确定。

原来用行名完成的功能，可以改用 dplyr 包的 `left_join()` 等函数，这些函数进行数据框的横向连接。详见23。

实际上，旧式数据框支持行名，有如下的缺点：

- 行名本身往往也是有效的数据，如身份证号，将有效数据以数据框中的列和行名两种不同形式保存，增加了复杂度；
- 为了使用某些变量辨识不同的行（观测），行名也具有局限性：行名必须是相互不同的，必须是字符型，而用来区分各个观测的变量有可能有多个，也可能不是字符型。
- 行名要求互不相同是有局限性的，如果用来辨识各行的变量有重复值，就可以构成对各行的一种自然的分组。

13.2.4 bibble 和数据框的转换

因为有一些老的 R 函数仅支持 data.frame，所以有时需要将 tibble 转换为数据框。设 `dt` 为 tibble, 如果不需要添加行名，则 `as.data.frame(dt)` 就可以转换为 data.frame。

如果需要添加行名，设 `dt` 中第一列要转换为行名，转化的模板为

```
da <- as.data.frame(dt[,-1])
rownames(da) <- dt[[1]]
```

如果 `dt` 中要转化为行名的列为 `id` 列，转化的模板为

```
da <- as.data.frame(dt[,-match("id", names(dt))]),
rownames(da) <- dt[["id"]]
```

反过来，如果 da 是一个数据框，要转化为 tibble，如果不需要保留行名，则 `as_tibble(da)` 就可以。

如果需要将行名转换为结果的一列，模板为

```
dt <- as_tibble(da, rownames="id")
```

则原来的行名变成了 dt 的 id 列。必要的话，tibble 也支持行名，这时，只要用

```
dt <- as_tibble(da, rownames=NA)
```

就表示转换成 tibble 后仍保留原来的行名。

13.2.5 列表类型的列

tibble 类型允许其中的列是列表类型，这样，该列的每个元素就可以是复杂类型，比如建模结果（列表），元素之间可以保存不等长的值。如：

```
tibble(x = 1:3,
  y = list(1, 1:2, 1:3))
```

```
## # A tibble: 3 x 2
##       x y
##   <int> <list>
## 1     1 <dbl [1]>
## 2     2 <int [2]>
## 3     3 <int [3]>
```

13.3 练习

假设class.csv已经读入为 R 数据框 d.class, 其中的 sex 列已经自动转换为因子。

(1) 显示 d.class 中年龄至少为 15 的行子集；

(2) 显示女生且年龄至少为 15 的学生姓名和年龄；

(3) 取出数据框中的 age 变量赋给变量 x。

Chapter 14

工作空间和变量赋值

14.1 工作空间

R 把在命令行定义的变量都保存到工作空间中，在退出 R 时可以选择是否保存工作空间。这也是 R 与其他如 C、Java 这样的语言的区别之一。

用 `ls()` 命令可以查看工作空间中的内容。

随着多次在命令行使用 R，工作空间的变量越来越多，使得重名的可能性越来越大，而且工作空间中变量太多也让我们不容易查看其内容。在命令行定义的变量称为“全局变量”，在编程实践中，全局变量是需要慎用的。

可以用 `rm()` 函数删除工作空间中的变量，格式如

```
rm(d, h, name, rec, sex, x)
```

要避免工作空间杂乱，最好的办法还是所有的运算都写到自定义函数中。自定义函数中定义的变量都是临时的，不会保存到工作空间中。这样，仅需要时才把变量值在命令行定义，这样的变量一般是读入的数据或自定义的函数（自定义函数也保存在工作空间中）。

可以定义如下的 `sandbox()` 函数：

```
sandbox <- function(){
  cat('沙盘：接连的空行回车可以退出。\n')
  browser()
}
```

运行 `sandbox()` 函数，将出现如下的 browser 命令行：

```
沙盘：接连的空行回车可以退出。
Called from: sandbox()
Browse[1]>
```

提示符变成了 “Browser[n]”，其中 n 代表层次序号。在这样的 browser 命令行中随意定义变量，定义的变量不会保存到工作空间中。用 “Q” 命令可以退出这个沙盘环境，接连回车也可以退出。

14.2 非法变量名

R 的变量名要求由字母、数字、下划线、小数点组成，开头不能是数字、下划线、小数点，中间不能使用空格、减号、井号等特殊符号，变量名不能与 if、NA 等保留字相同。

有时为了与其它软件系统兼容，需要使用不符合规则的变量名，这只要将变量名两边用反向单撇号 “`” 保护，如：

```
`max score` <- 100
99 + `max score`
```

```
## [1] 199
```

如果变量名（元素名、列名等）是以字符串形式使用，就不需要用 “`” 保护。如：

```
x <- c("score a"=85, "score b"=66)
x
```

```
## score a score b
##      85      66
```

14.3 变量赋值与绑定

本小节内容技术上比较复杂，初学者可以略过。

在 R 中赋值本质上是把一个存储的对象与一个变量名 “绑定”(bind) 在一起，比如：

```
x <- c(1,2,3)
```

并不是像 C++、JAVA 等语言那样，x 代表某个存储位置，“x <- c(1,2,3)” 代表将 1 到 3 这些值存储到 x 所指向的存储位置。实际上，<-右边的 c(1,2,3) 是一个表达式，其结果为一个 R 对象 (object)，而 x 只是一个变量名，并没有固定的类型、固定的存储位置，赋值的结果是将 x 绑定到值为 (1,2,3) 的 R 对象上。R 对象有值，但不必有对应的变量名；变量名必须经过绑定才有对应的值和存储位置。

这样，同一个 R 对象也可以被两个或多个变量名绑定。对于基本的数据类型如数值型向量，两个指向相同对象的变量当一个变量被修改时自动制作副本。tracemem(x) 可以显示变量名 x 绑定的地址并在其被制作副本时显示地址变化。如：

```
x <- c(1,2,3)
cat(tracemem(x), "\n")
```

```
## <000001C5906FF1C8>
```

```
y <- x    # 这时 y 和 x 绑定到同一 R 对象
cat(tracemem(y), "\n")
```

```
## <000001C5906FF1C8>
```

```
y[3] <- 0 # 这时 y 制作了副本
```

```
## tracemem[0x000001c5906ff1c8 -> 0x000001c590813018]: eval eval eval_with_user_handlers withVisible
```

```
x
```

```
## [1] 1 2 3
```

```
y
```

```
## [1] 1 2 0
```

```
untracemem(x); untracemem(y)
rm(x, y)
```

可见 `y <- x` 并没有制作副本，但是修改 `y[3]` 值时就对 `y` 制作了副本。

修改某个变量名所指向的对象可能会制作副本，如：

```
x <- c(1,2,3)
cat(tracemem(x), "\n")
```

```
## <000001C590E5B648>
```

```
x[3] <- 0
```

```
## tracemem[0x000001c590e5b648 -> 0x000001c590ef1b48]: eval eval eval_with_user_handlers withVisible
```

```
x
```

```
## [1] 1 2 0
```

```
untracemem(x)
```

在调用函数时，如果函数内部不修改自变量的元素值，输入的自变量并不制作副本，而是直接被函数使用实参绑定的对象。如：

```
x <- c(1,2,3)
cat(tracemem(x), "\n")
```

```
## <000001C591441C08>
```

```
f <- function(v){ return(v) }
z <- f(x)
cat(tracemem(z), "\n")
```

```
## <000001C591441C08>
```

```
untracemem(x); untracemem(z)
```

从上面的例子可以看出，函数 f 以 x 为实参，但不修改 x 的元素，不会生成 x 的副本，返回的值是 x 指向的对象本身，再次赋值给 z，也不制作副本，z 和 x 绑定到同一对象。

如果函数内部修改自变量的元素值，则输入的自变量也会制作副本。如：

```
x <- c(1,2,3)
cat(tracemem(x), "\n")
```

```
## <000001C591A29D68>
```

```
f2 <- function(v){ v[1] <- -999; return(v) }
z <- f2(x)
```

```
## tracemem[0x000001c591a29d68 -> 0x000001c591ab8f78]: f2 eval eval eval_with_user_handlers withVisib
```

```
cat(tracemem(z), "\n")
```

```
## <000001C591AB8F78>
```

```
untracemem(x); untracemem(z)
```

从程序输出看，函数 f2() 以 x 为实参，并修改 x 的内部元素，就制作了 x 的副本，返回的结果赋给变量 z，绑定的是修改后的副本。

如果在函数中对自变量重新赋值，这实际是重新绑定，也不会制作输入的实参的副本。

如果修改 y 的元素值时还修改了其存储类型，比如整型改为浮点型，则会先制作 y 的副本，然后制作类型改变后的副本，然后再修改其中的元素值。

在当前的 R 语言中，一个对象的引用（如绑定的变量名）个数，只区分 0 个、1 个或多个这三种情况。在没有引用时，R 的垃圾收集器会定期自动清除这些对象。rm(x) 只是删除绑定，并不会马上清除 x 绑定的对象。如果已经有多个引用，即使是只有 2 个，减少一个引用也还是“多个”状态，不会变成 1 个。

垃圾收集器是在 R 程序要求分配新的对象空间时自动运行的，R 函数 gc() 可以要求马上运行垃圾收集器，并返回当前程序用到的存储量；lobstr 包的 mem_used() 函数则报告当前会话内存字节数。

在上面的示例中，用了基本类型的向量讲解是否制作副本。考虑其它类型的复制。

如果 x 是一个有 5 个元素的列表，则 y <- x 使得 y 和 x 指向同一个列表对象。但是，列表对象的每个元素实际上也相当于一个绑定，每个元素指向一个元素值对象。所以如果修改 y：y[[3]] <- 0，这时列表 y 首先被制作了副本，但是每个元素指向的元素值对象不变，仍与 x 的各个元素指向的对象相同；然后，y[[3]] 指向的元素值进行了重新绑定，不再指向 x[[3]]，而是指向新的保存了值 0 的对象，但 y 的其它元素指向的对象仍与 x 公用。列表的这种复制方法称为浅拷贝，列表对象及各个元素绑定被复制，但各个元素指向（保存）的对象不变。这种做法节省空间也节省运行时间。在 R 的 3.1.0 之前则用的深拷贝方法，即复制列表时连各个元素保存的值也制作副本。

如果 x 是一个数据框，这类似于一个列表，每个变量相当于一个列表元素，数据框的每一列实际绑定到一个对象上。如果 y <- x，则修改 y 的某一列会对 y 进行浅拷贝，然后仅该列被制作了副本并被修改，其它未修改的列仍与 x 共用值对象。

但是如果修改数据框 y 的一行，因为这涉及到所有列，所以整个数据框的所有列都会制作副本。

对于字符型向量，实际上 R 程序的所有字符型常量都会建立一个全局字符串池，这样有许多重复值时可以节省空间。

用 lobstr 包的 obj_size() 函数可以求变量的存储大小，如 obj_size(x)，也可以求若干个变量的总大小，如 obj_size(x,y)。因为各种绑定到同一对象的可能性，所以变量的存储大小可能会比想象要少，比如，共用若干列的两个数据框，字符型向量，等等。基本 R 软件的 object.size() 则不去检查是否有共享对象，所以对列表等变量的存储大小估计可能会偏高。

从 R 3.5.0 版开始，1:n 这种对象仅保存其开始值和结束值。

在自定义函数时，自变量通常是按引用使用的，函数内部仅使用自变量的值而不对其进行修改时不会制作副本，但是如果函数内部修改了自变量的值，就会制作副本，所以如果自变量的存储量很大而且被修改后返回，调用这个函数时就会造成运行速度缓慢。在函数内应慎重修改自变量的值。

在循环中修改数据框的列，也会造成反复的复制。

14.4 环境

前面讲的工作空间就是一个环境，称为“全局环境”。环境可以认为是包含了变量绑定和函数定义的一个对象，对环境的修改都不会制作副本。一般我们只需要用到全局环境，在自定义函数的运行过程中会有一个局部的运行环境，但是在函数内定义的嵌套函数可以带有其定义处的环境，这样可以实现有历史记忆的函数，类似于其它面向对象程序设计语言如 C++ 等，对象可以同时有状态（变量成员）和适用操作（方法成员）。

环境的应用将在“函数进阶”部分讲到内嵌函数、函数工厂时详细讲解。

Part III

编程

Chapter 15

R 输入输出

15.1 输入输出的简单方法

15.1.1 简单的输出

用 print() 函数显示某个变量或表达式的值，如

```
x <- 1.234
print(x)
## [1] 1.234
y <- c(1,3,5)
print(y[2:3])
## [1] 3 5
```

在命令行使用 R 时，直接以变量名或表达式作为命令可以起到用 print() 函数显示的相同效果。

用 cat() 函数把字符串、变量、表达式连接起来显示，其中变量和表达式的类型一般是标量或向量，不能是矩阵、列表等复杂数据。如

```
cat("x =", x, "\n")
## x = 1.234
cat("y =", y, "\n")
## y = 1 3 5
```

注意 cat() 显示中需要换行需要在自变量中包含字符串"\n"，即换行符。

cat() 默认显示在命令行窗口，为了写入指定文件中，在 cat() 调用中用 file= 选项，这时如果已有文件会把原有内容覆盖，为了在已有文件时不覆盖原有内容而是在末尾添加，在 cat() 中使用 append=TRUE 选项。如:

```
cat("=== 结果文件 ===\n", file="res.txt")
cat("x =", x, "\n", file="res.txt", append=TRUE)
```

函数 sink() 可以用来把命令行窗口显示的运行结果转向保存到指定的文本文件中，如果希望保存到文件的同时也在命令行窗口显示，使用 split=TRUE 选项。如

```
sink("allres.txt", split=TRUE)
```

为了取消这样的输出文件记录，使用不带自变量的 sink() 调用，如

```
sink()
```

在 R 命令行环境中定义的变量、函数会保存在工作空间中，并在退出 R 会话时可以保存到硬盘文件中。用 save() 命令要求把指定的若干个变量（直接用名字，不需要表示成字符串）保存到用 file= 指定的文件中，随后可以用 load() 命令恢复到工作空间中。虽然允许保存多个变量到同一文件中，但尽可能仅保存一个变量，而且使用变量名作为文件名。用 save() 保存的 R 特殊格式的文件是通用的，不依赖于硬件和操作系统。如

```
save(scores, file="scores.RData")
load("scores.RData")
```

保存多个变量，如 x, zeta，命令如：

```
save(x, zeta, file="myvars20200315.RData")
```

或

```
save(list = c("x", "zeta"), file="myvars20200315.RData")
```

对于一个数据框，可以用 write.csv() 或 readr::write_csv() 将其保存为逗号分隔的文本文件，这样的文件可以很容易地被其它软件识别访问，如 Microsoft Excel 软件可以很容易地把这样的文件读成电子表格。用如

```
da <- tibble("name"=c(" 李明", " 刘颖", " 张浩"),
    "age"=c(15, 17, 16))
write_csv(da, path="mydata.csv")
```

结果生成的 mydata.csv 文件内容如下：

```
name,age
李明,15
刘颖,17
张浩,16
```

但是，在 Microsoft 的中文版 Windows 操作系统中，默认编码是 GB 编码，用 `write_csv()` 生成的 CSV 文件总是使用 UTF-8 编码，系统中的 MS Office 软件不能自动识别这样编码的 CSV 文件，可以改用 `write_excel_csv()` 函数。

15.1.2 简单的输入

用 `scan()` 函数可以输入文本文件中的数值向量，文件名用 `file=` 选项给出。文件中数值之间以空格和换行分开。如

```
cat(1:12, "\n", file="d:/work/x.txt")
x <- scan("d:/work/x.txt")
```

程序中用全路径给出了输入文件位置，注意路径中用了正斜杠/作为分隔符，如果在 MS Windows 环境下使用\作为分隔符，在 R 的字符串常量中\必须写成\\。

如果 `scan()` 中忽略输入文件参数，此函数将从命令行读入数据。可以在一行用空格分开多个数值，可以用多行输入直到空行结束输入。

这样的方法也可以用来读入矩阵。设文件 `mat.txt` 包含如下矩阵内容:

```
3  4  2
5 12 10
7  8  6
1  9 11
```

可以先把文件内容读入到一个 R 向量中，再利用 `matrix()` 函数转换成矩阵，注意要使用 `byrow=TRUE` 选项，而且只要指定 `ncol` 选项，可以忽略 `nrow` 选项。如

```
M <- matrix(scan("mat.txt", quiet=TRUE), ncol=3, byrow=TRUE)
M
```

`scan()` 中的 `quite=TRUE` 选项使得读入时不自动显示读入的数值项数。

上面读入数值矩阵的方法在数据量较大的情形也可以使用，与之不同的是，`read.table()` 或 `readr::read_table()` 函数也可以读入这样的数据，但是会保存成数据框而不是矩阵，而且 `read.table()` 函数在读入大规模的矩阵时效率很低。

15.2 读取 CSV 文件

15.2.1 CSV 格式

对于保存在文本文件中的电子表格数据，R 可以用 `read.csv()`, `read.table()`, `read.delim()`, `read.fwf()` 等函数读入, 但是建议在 readr 包的支持下用 `read_csv()`, `read_table2()`, `read_delim()`, `read_fwf()` 等函数

读入，这些将读入的数据框保存为 tibble 类型，tibble 是数据框的一个变种，改善了数据框的一些不适当的设计。readr 的读入速度比基本 R 软件的 `read.csv()` 等函数的速度快得多，速度可以相差十倍，也不自动将字符型列转换成因子，不自动修改变量名为合法变量名，不设置行名。

对于中小规模的数据，CSV 格式作为文件交换格式比较合适，兼容性强，各种数据管理软件与统计软件都可以很容易地读入和生成这样格式的文件，但是特别大型的数据读入效率很低。

CSV 格式的文件用逗号分隔开同一行的数据项，一般第一行是各列的列名（变量名）。对于数值型数据，只要表示成数值常量形式即可。对于字符型数据，可以用双撇号包围起来，也可以不用撇号包围。但是，如果数据项本身包含逗号，就需要用双撇号包围。例如，下面是一个名为 `testcsv.csv` 的文件内容，其中演示了内容中有逗号、有双撇号的情况。

```
id,words
1,"PhD"
2,Master's degree
3,"Bond,James"
4,"A ""special"" gift"
```

为读入上面的内容，只要用如下程序:

```
d <- read_csv("testcsv.csv")
```

读入的数据框显示如下:

```
# A tibble: 4 × 2
     id            words
  <int>            <chr>
1     1              PhD
2     2  Master's degree
3     3       Bond,James
4     4 A "special" gift
```

15.2.2 从字符串读入

`read_csv()` 还可以从字符串读入一个数据框，如

```
d.small <- read_csv("name,x,y
John, 33, 95
Kim, 21, 64
Sandy, 49, 100
")
```

```
## Rows: 3 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): name
## dbl (2): x, y
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
d.small
```

```
## # A tibble: 3 x 3
##   name      x     y
##   <chr> <dbl> <dbl>
## 1 John     33    95
## 2 Kim      21    64
## 3 Sandy    49   100
```

15.2.3 read_csv 选项

read_csv() 的 skip= 选项跳过开头的若干行。当数据不包含列名时，只要指定 col_names=FALSE，变量将自动命名为 X1, X2, ...，也可以用 col_names= 指定各列的名字，如

```
d.small <- read_csv("John, 33, 95
Kim, 21, 64
Sandy, 49, 100
", col_names=c("name", "x", "y") )
```

```
## Rows: 3 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): name
## dbl (2): x, y
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
d.small
```

```
## # A tibble: 3 x 3
```

```
##   name      x     y
##   <chr> <dbl> <dbl>
## 1 John     33    95
## 2 Kim      21    64
## 3 Sandy    49   100
```

15.2.4 编码设置

CSV 文件是文本文件，是有编码问题的，尤其是中文内容的文件。readr 包的默认编码是 UTF-8 编码。例如，文件`data/bp.csv`以 GBK 编码（有时称为 GB18030 编码，这是中文 Windows 所用的中文编码）保存了如下内容：

```
序号,收缩压
1,145
5,110
6, 未测
9,150
10, 拒绝
15,115
```

如果直接用 `read_csv()`：

```
d <- read_csv("data/bp.csv")
```

可能在读入时出错,或者访问时出错。为了读入用 GBK 编码的中文 CSV 文件,需要利用 `locale` 参数和 `locale()` 函数：

```
d <- read_csv("data/bp.csv",
  locale=locale(encoding="GBK"))
```

```
## Rows: 6 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): 收缩压
## dbl (1): 序号
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
d
```

```
## # A tibble: 6 x 2
##    序号 收缩压
##   <dbl> <chr>
## 1     1 145
## 2     5 110
## 3     6 未测
## 4     9 150
## 5    10 拒绝
## 6    15 115
```

15.2.5 缺失值设置

read_csv() 将空缺的值读入为缺失值，将"NA" 也读入为缺失值。可以用 na= 选项改变这样的设置。也可以将带有缺失值的列先按字符型原样读入，然后再进行转换。

比如，上面的 bp.csv 文件中，先将血压列按字符型读入，再增加一列转换为数值型的列，非数值转换为 NA:

```
d <- read_csv("data/bp.csv",
  locale=locale(encoding="GBK"))
```

```
## Rows: 6 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): 收缩压
## dbl (1): 序号
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
d[[" 收缩压数值"]] <- as.numeric(d[[" 收缩压"]])
```

```
## Warning: NAs introduced by coercion
```

```
d
```

```
## # A tibble: 6 x 3
##    序号 收缩压 收缩压数值
##   <dbl> <chr>       <dbl>
```

```
## 1      1 145            145
## 2      5 110            110
## 3      6 未测            NA
## 4      9 150            150
## 5     10 拒绝            NA
## 6     15 115            115
```

15.2.6 各列类型设置

对每列的类型，readr 用前 1000 行猜测合理的类型，并在读取后显示猜测的每列类型。

但是有可能类型改变发生在 1000 行之后。`col_types` 选项可以指定每一列的类型，如`"col_double()"`, `"col_integer()"`, `"col_character()"`, `"col_factor()"`, `"col_date()"`, `"col_datetime"` 等。`cols()` 函数可以用来规定各列类型，并且有一个`.default` 参数指定缺省类型。对因子，需要在 `col_factor()` 中用 `lelvels=` 指定因子水平。

可以复制 readr 猜测的类型作为 `col_types` 的输入，这样当数据变化时不会因为偶尔猜测错误而使得程序出错。如

```
d <- read_csv("data/bp.csv", locale=locale(encoding="GBK"),
    col_types=cols(
      `序号` = col_integer(),
      `收缩压` = col_character()
      ))
d
```

```
## # A tibble: 6 x 2
##    序号 收缩压
##   <int> <chr>
## 1      1 145
## 2      5 110
## 3      6 未测
## 4      9 150
## 5     10 拒绝
## 6     15 115
```

当猜测的文件类型有问题的时候，可以先将所有列都读成字符型，然后用 `type_convert()` 函数转换，如：

```
d <- read_csv("filename.csv",
              col_types=cols(.default = col_character()))
d <- type_convert(d)
```

读入有错时，对特大文件可以先少读入一些行，用 `nmax=` 可以指定最多读入多少行。调试成功后再读入整个文件。

15.2.7 因子类型设置

设文件data/class.csv内容如下：

```
name,sex,age,height,weight
Alice,F,13,56.5,84
Becka,F,13,65.3,98
Gail,F,14,64.3,90
Karen,F,12,56.3,77
Kathy,F,12,59.8,84.5
Mary,F,15,66.5,112
Sandy,F,11,51.3,50.5
Sharon,F,15,62.5,112.5
Tammy,F,14,62.8,102.5
Alfred,M,14,69,112.5
Duke,M,14,63.5,102.5
Guido,M,15,67,133
James,M,12,57.3,83
Jeffrey,M,13,62.5,84
John,M,12,59,99.5
Philip,M,16,72,150
Robert,M,12,64.8,128
Thomas,M,11,57.5,85
William,M,15,66.5,112
```

最简单地用 read_csv() 读入上述 CSV 文件，程序如:

```r
d.class <- read_csv("data/class.csv")
```

```
## Rows: 19 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): name, sex
## dbl (3): age, height, weight
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

从显示看出，读入后显示了每列的类型。对性别变量，没有自动转换成因子，而是保存为字符型。为了按自己的要求转换各列类型，用了 read_csv() 的 coltypes= 选项和 cols() 函数如下：

```
d.class <- read_csv(
  "data/class.csv",
  col_types=cols(
  .default = col_double(),
  name=col_character(),
  sex=col_factor(levels=c("M", "F")) ))
str(d.class)
```

```
## spec_tbl_df [19 x 5] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ name  : chr [1:19] "Alice" "Becka" "Gail" "Karen" ...
##  $ sex   : Factor w/ 2 levels "M","F": 2 2 2 2 2 2 2 2 2 1 ...
##  $ age   : num [1:19] 13 13 14 12 12 15 11 15 14 14 ...
##  $ height: num [1:19] 56.5 65.3 64.3 56.3 59.8 66.5 51.3 62.5 62.8 69 ...
##  $ weight: num [1:19] 84 98 90 77 84.5 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   .default = col_double(),
##   ..   name = col_character(),
##   ..   sex = col_factor(levels = c("M", "F"), ordered = FALSE, include_na = FALSE),
##   ..   age = col_double(),
##   ..   height = col_double(),
##   ..   weight = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

其中 `str()` 函数可以显示数据框的行数 (obs.) 和变量数 (variables)，以及每个变量（列）的类属等信息。

15.2.8 读入日期

设文件`data/dates.csv`中包含如下内容，并设其文件编码为 GBK:

```
序号,出生日期,发病日期
1,1941/3/8,2007/1/1
2,1972/1/24,2007/1/1
3,1932/6/1,2007/1/1
4,1947/5/17,2007/1/1
5,1943/3/10,2007/1/1
6,1940/1/8,2007/1/1
7,1947/8/5,2007/1/1
```

```
8,2005/4/14,2007/1/1
9,1961/6/23,2007/1/2
10,1949/1/10,2007/1/2
```

可以先把日期当作字符串读入:

```
d.dates <- read_csv('data/dates.csv',
  locale=locale(encoding="GBK"))
```

```
## Rows: 10 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): 出生日期, 发病日期
## dbl (1): 序号
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

然后用 lubridate::ymd() 函数转换为 R 日期类型:

```
d.dates[[" 出生日期 ct"]] <- lubridate::ymd(
  d.dates[[" 出生日期"]], tz='Etc/GMT-8')
d.dates[[" 发病日期 ct"]] <- lubridate::ymd(
  d.dates[[" 发病日期"]], tz='Etc/GMT-8')
```

也可以用 R 本身的 as.POSIXct 函数转换:

```
d.dates[[" 出生日期 ct"]] <- as.POSIXct(
  d.dates[[" 出生日期"]], format='%Y/%m/%d', tz='Etc/GMT-8')
d.dates[[" 发病日期 ct"]] <- as.POSIXct(
  d.dates[[" 发病日期"]], format='%Y/%m/%d', tz='Etc/GMT-8')
```

这时保存的是 POSIXct 类型。经过转换后的数据为:

```
knitr::kable(d.dates)
```

序号	出生日期	发病日期	出生日期 ct	发病日期 ct
1	1941/3/8	2007/1/1	1941-03-08	2007-01-01
2	1972/1/24	2007/1/1	1972-01-24	2007-01-01
3	1932/6/1	2007/1/1	1932-06-01	2007-01-01
4	1947/5/17	2007/1/1	1947-05-17	2007-01-01
5	1943/3/10	2007/1/1	1943-03-10	2007-01-01
6	1940/1/8	2007/1/1	1940-01-08	2007-01-01
7	1947/8/5	2007/1/1	1947-08-05	2007-01-01
8	2005/4/14	2007/1/1	2005-04-14	2007-01-01
9	1961/6/23	2007/1/2	1961-06-23	2007-01-02
10	1949/1/10	2007/1/2	1949-01-10	2007-01-02

也可以用 R 本身的 `as.Date` 函数转换：

```
d.dates[[" 出生日期 ct"]] <- as.Date(
  d.dates[[" 出生日期"]], format='%Y/%m/%d')
d.dates[[" 发病日期 ct"]] <- as.Date(
  d.dates[[" 发病日期"]], format='%Y/%m/%d')
```

这时保存的是 Date 类型。

上面将日期先读入为字符型再转换是比较保险的做法。还可以直接在 `read_csv()` 函数中指定某列为 `col_date()`：

```
d.dates <- read_csv(
  'data/dates.csv',
  locale=locale(encoding="GBK"),
  col_types=cols(
    `序号`=col_integer(),
    `出生日期`=col_date(format="%Y/%m/%d"),
    `发病日期`=col_date(format="%Y/%m/%d")
  ))
print(d.dates)
```

```
## # A tibble: 10 x 3
##      序号 出生日期   发病日期
##     <int> <date>     <date>
##  1      1 1941-03-08 2007-01-01
##  2      2 1972-01-24 2007-01-01
##  3      3 1932-06-01 2007-01-01
##  4      4 1947-05-17 2007-01-01
```

```
## 5      5 1943-03-10 2007-01-01
## 6      6 1940-01-08 2007-01-01
## 7      7 1947-08-05 2007-01-01
## 8      8 2005-04-14 2007-01-01
## 9      9 1961-06-23 2007-01-02
## 10    10 1949-01-10 2007-01-02
```

15.2.9 其它函数

除了 `read_csv()` 函数以外，R 扩展包 readr 还提供了其它的从文本数据读入数据框的函数，如 `read_table()`, `read_tsv()`, `read_fwf()` 等。这些函数读入的结果保存为 tibble。`read_table()` 读入用空格作为间隔的文本文件，同一行的两个数据项之间可以用一个或多个空格分隔，不需要空格个数相同，也不需要上下对齐。`read_tsv()` 读入用制表符分隔的文件。`read_fwf()` 读入上下对齐的文本文件。

另外，`read_lines()` 函数将文本文件各行读入为一个字符型向量。`read_file()` 将文件内容读入成一整个字符串，`read_file_raw()` 可以不管文件编码将文件读入为一个二进制字符串。

对特别大的文本格式数据，data.table 扩展包的 `fread()` 读入速度更快。

readr 包的 `write_excel_csv()` 函数将 tibble 保存为 csv 文件，总是使用 UTF-8 编码，结果可以被 MS Office 读取。

文本格式的文件都不适用于大型数据的读取与保存。大型数据可以通过数据库接口访问，可以用 R 的 `save()` 和 `load()` 函数按照 R 的格式访问，还有一些特殊的针对大数据集的 R 扩展包。

15.3 保存与恢复 R 变量

基本 R 的 `save()` 命令可以将一个或者多个当前的 R 变量保存到文件中，保存结果是经过压缩的，在不同的 R 运行环境中兼容，但是有可能依赖于 R 的软件版本。如 `save(x, y, file="mydata.RData")`。随后可以用 `load()` 命令恢复这些变量到当前环境中，如 `load("mydata.RData")` 可以恢复保存的变量 `x, y`。

推荐使用 `saveRDS()`, 它仅保存单个变量，如 `save(x, file="x.RData")`。用 `loadRDS()` 载入并返回变量，如 `xold <- loadRDS("x.RData")`。但是，`saveRDS()` 保存的文件不适合用于不同计算机之间的数据交换。

R 的 arrow 扩展包提供了对 parquet 数据文件格式的支持，这种文件适用于在不同计算机、不同编程语言之间传递变量值，包括数据框类型，对于大规模数据的读写访问效率也比 RDS 格式文件高。

15.4 Excel 表访问

15.4.1 借助于文本格式

为了把 Microsoft Excel 格式的数据读入到 R 中，最容易的办法是在 Excel 软件中把数据表转存为 CSV 格式，然后用 `read.csv()` 读取。

为了把 R 的数据框保存为 Excel 格式，只要用 `write.csv()` 或 `readr::write_excel_csv()` 把数据框保存成 CSV 格式，然后在 Excel 中打开即可。例如，下面的程序演示了 `write.csv()` 的使用:

```
d1 <- tibble(" 学号"=c("101", "103", "104"),
             " 数学"=c(85, 60, 73),
             " 语文"=c(90, 78, 80))
write.csv(d1, file="tmp1.csv", row.names=FALSE)
```

保存在文件中的结果显示如下：

```
学号,数学,语文
101,85,90
103,60,78
104,73,80
```

15.4.2 使用剪贴板

为了把 Excel 软件中数据表的选中区域读入到 R 中，可以借助于剪贴板。在 Excel 中复制选中的区域，然后在 R 中用如

```
myDF <- read.delim("clipboard")
```

就可以把选中部分转换成一个 R 的数据框。如果复制的区域不含列名，应加上 `header=FALSE` 选项。

这种方法也可以从 R 中复制数据到在 Excel 中打开的电子表格中，例如

```
write.table(iris, file="clipboard", sep = "\t", col.names = NA)
```

首先把指定的数据框（这里是 iris）写入到了剪贴板，然后在用 Excel 软件打开的工作簿中只要粘贴就可以。上述程序中 `write.table()` 函数把指定的数据框写入到指定的文件中, 其中的 `col.names=NA` 选项是一个特殊的约定，这时保存的文件中第一行是列名，如果有行名的话，行名所在的列对应的列名是空白的（但是存在此项）。

如果从 R 中复制数据框到打开的 Excel 文件中时不带行名，但是带有列名，可以写这样一个通用函数

```
write.clipboard <- function(df){
  write.table(df, file="clipboard", sep="\t",
              row.names=FALSE)
}
```

15.4.3 利用 readxl 扩展包

readxl 扩展包的 `readxl()` 函数利用独立的 C 和 C++ 库函数读入.xls 和.xlsx 格式的 Excel 文件。一般格式为

```
read_excel(path, sheet = 1, col_names = TRUE,
    col_types = NULL, na = "",  skip = 0)
```

结果返回读入的表格为一个数据框。各个自变量为：

- `path`: 要读入的 Excel 文件名，可以是全路径，路径格式要符合所用操作系统要求。
- `sheet`: 要读入哪一个工作簿 (sheet)，可以是整数序号，也可以是工作簿名称的字符串。缺省为第一个工作簿。
- `col_names`: 是否用第一行内容作为列名，缺省为是。
- `col_types`: 可以在读入时人为指定各列的数据类型，缺省时从各列内容自动判断，有可能会不够准确。人为指定时，指定一个对应于各列的字符型向量，元素可取值为:
  - `blank`: 自动判断该列；
  - `numeric`: 数值型；
  - `date`: 日期；
  - `text`: 字符型。
- `na`: 指定作为缺失值的编码，默认完全空的单元格表示缺失值，如果还想包括 `NA`, `N/A`，可表示为 ‘na = c(“”,”NA”, “N/A”, “na”, “n/a”)。

当 Excel 文件中包含多个工作簿时，可以用 `excel_sheets()` 函数返回所有的工作簿名，随后可以在 `read_excel()` 中用 `sheet=` 指定要读入的工作簿。多个工作簿如果属于同一大数据集的不同观测，可以用 `bind_rows()` 函数作上下合并。

writexl 扩展包可以用来将数据框保存为 Excel 格式，如 `writexl::write_xlsx(df, path)`。

除了 readxl 和 writexl 扩展包，XLConnect, xlsx, tidyxl 也可以进行与 Excel 文件或者 Excel 软件的交互。为了输出多个工作簿以及进行一些不同格式设置，可以使用 openxlsx 扩展包。

15.5 文件访问

15.5.1 连接

输入输出可以针对命令行，针对文件，R 支持扩展的文件类型，称为 “连接 (connection)”。

函数 `file()` 生成到一个普通文件的连接，函数 `url()` 生成一个到指定的 URL 的连接，函数 `gzfile`, `bzfile`, `xzfile`, `unz` 支持对压缩过的文件的访问（不是压缩包，只对一个文件压缩）。这些函数大概的用法如下：

```
file("path", open="", blocking=T,
     encoding = getOption("encoding"),
     raw = FALSE)

url(description, open = "", blocking = TRUE,
    encoding = getOption("encoding"))

textConnection(description, open="r",
    local = FALSE,
    encoding = c("", "bytes", "UTF-8"))

gzfile(description, open = "",
       encoding = getOption("encoding"),
       compression = 6)

bzfile(description, open = "",
       encoding = getOption("encoding"),
       compression = 9)

xzfile(description, open = "",
       encoding = getOption("encoding"),
       compression = 6)

unz(description, filename, open = "",
    encoding = getOption("encoding"))
```

生成连接的函数不自动打开连接。给定一个未打开的连接，读取函数从中读取时会自动打开连接，函数结束时自动关闭连接。用 `open()` 函数打开连接，返回一个句柄；生成连接时可以用 `open` 参数要求打开连接。要多次从一个连接读取时就应该先打开连接，读取完毕用 `close` 函数关闭。

函数 `textConnection()` 打开一个字符串用于读写。

在生成连接与打开连接的函数中用 `open` 参数指定打开方式，取值为：

- `r`—文本型只读;
- `w`—文本型只写;
- `a`—文本型末尾添加;
- `rb`—二进制只读;
- `wb`—二进制只写;
- `ab`—二进制末尾添加;
- `r+` 或 `r+b`—允许读和写;

- w+ 或 w+b—允许读和写，但刚打开时清空文件;
- a+ 或 a+b—末尾添加并允许读。

15.5.2 文本文件访问

函数 readLines()、readr::read_lines()、scan() 可以从一个文本型连接读取。

给定一个打开的连接或文件名，用 readLines 函数可以把文件各行读入为字符型向量的各个元素，不包含文件中用来分开各行的换行标志。可以指定要读的行数。如

```
ll <- readLines("data/class.csv")
print(head(ll, 3))
```

```
## [1] "name,sex,age,height,weight" "Alice,F,13,56.5,84"
## [3] "Becka,F,13,65.3,98"
```

用 readr 包的 read_lines() 如:

```
ll <- readr::read_lines("data/class.csv")
print(head(ll, 3))
```

```
## [1] "name,sex,age,height,weight" "Alice,F,13,56.5,84"
## [3] "Becka,F,13,65.3,98"
```

用 writeLines 函数可以把一个字符型向量各元素作为不同行写入一个文本型连接。如

```
vnames <- strsplit(ll, ",")[[1]]
writeLines(vnames, "class-names.txt")
```

其中的第二参数应该是一个打开的文本型写入连接，但是可以直接给出一个要写入的文件名。用 readr 包的 write_lines():

```
readr::write_lines(vnames, "class-names.txt")
```

用 scan 函数读入用空格和空行分隔的字符串向量:

```
vnames <- scan(
  "class-names.txt", what=character(),
  quiet=TRUE)
vnames
## [1] "name"   "sex"    "age"    "height" "weight"
```

15.5.3 文本文件分批读写

readLines()、readr::read_lines()、writeLines()、readr::write_lines() 支持分批读写。这需要预先打开要读取和写入的文件，所有内容都处理一遍以后关闭读取和写入的文件。

使用 file() 函数打开文件用于读写，使用 close() 函数关闭打开的文件。打开文件时可以用 encoding= 指定编码。

下面的程序每次将 GB18030 编码的文件 cancer.csv 读入至多 10 行，转为 UTF-8 写入 tmp.csv 中，使用 readLines() 和 writeLines():

```
fin <- file("data/cancer.csv", "rt", encoding="GBK")
fout <- file("tmp.csv", "wt", encoding="UTF-8")
repeat{
  lines <- readLines(fin, n=10)
  cat("Read", length(lines), "lines.", "\n")
  if(length(lines)==0) break
  writeLines(lines, fout)
}
close(fout)
close(fin)
```

下面是使用 readr::read_lines() 的版本，需要打开的文件为二进制格式，并且需要在打开文件时和读取时都说明编码为 GBK：

```
fin <- file("data/cancer.csv", "rb", encoding="GBK")
fout <- file("tmp.csv", "wt", encoding="UTF-8")
repeat{
  lines <- read_lines(fin, n_max=10, locale=locale(encoding="GBK"))
  cat("Read", length(lines), "lines.", "\n")
  if(length(lines)==0) break
  writeLines(lines, fout)
}
close(fout)
close(fin)
```

15.5.4 二进制文件访问

函数 save 用来保存 R 变量到文件，函数 load 用来从文件中读取保存的 R 变量。

```
save(x, y, file="saved20210811.RData")
```

或

```
save(list=c("x", "y"), file="saved20210811.RData")
```

读入如

```
load("saved20210811.RData")
```

对较大的数据，常常每个保存文件仅保存一个变量，并以变量名为文件名，如：

```
save(data01, file="data01.RData")
```

函数 readBin 和 writeBin 对 R 变量进行二进制文件存取。

如果要访问其它软件系统的二进制文件，请参考 R 手册中的 “R Data Import/Export Manual”。

15.5.5 字符型连接

函数 textConnection 打开一个字符串用于读取或写入，是很好用的一个 R 功能。可以把一个小文件存放在一个长字符串中，然后用 textConnection 读取，如

```
fstr <-
"name,score
王芳,78
孙莉,85
张聪,80
"
d <- read.csv(textConnection(fstr), header=TRUE)
print(d)
##   name score
## 1 王芳    78
## 2 孙莉    85
## 3 张聪    80
```

读取用的 textConnection 的参数是一个字符型变量。readr 包的函数不支持从 textConnection 读取。

在整理输出结果时，经常可以向一个字符型变量连接写入，最后再输出整个字符串值。例如：

```
tc <- textConnection("sres", open="w")
cat("Trial of text connection.\n", file=tc)
cat(1:10, "\n", file=tc, append=TRUE)
close(tc)
print(sres)
```

注意写入用的 `textConnection` 的第一个参数是保存了将要写入的字符型变量名的字符串，而不是变量名本身，第二个参数表明是写入操作，使用完毕需要用 `close` 关闭。

15.6 中文编码问题

读写文本格式的数据，或者用 `readLines()`、`readr::read_lines()` 读写文本文件，可能会遇到中文编码不匹配的问题。这里总结一些常用解决方法，所用的操作系统为中文 Windows10, 在 RStudio 中运行，R 版本为 4.2。常见的中文编码有 GBK(或 GB18030, GB)，UTF-8，UTF-8 有 BOM 标志等。

可以用 `iconvlist()` 查看 R 支持的编码名称。

假设有如下的含有中文的文件：

```
序号,收缩压
1,145
5,110
6, 未测
9,150
10, 拒绝
15,115
```

这个文件是在中文版 MS Office 的 Excel 软件中输入后，用 Office 的 “文件——另存为——.csv 格式” 生成的，结果的编码是 GBK 编码，或 GB18030 编码。文件下载：data/bp.csv

我们用工具软件将其转换成 UTF-8 无 BOM 格式，下载链接：data/bp-utf8nobom.csv

转为 UTF-8 有 BOM 格式，下载链接：data/bp-utf8bom.csv

15.6.1 用基本 R 的读取函数读取

对于 GBK 编码的文本文件，较早期的 R 的基本函数 `read.csv()`、`read.table()`、`readLines()` 函数都可以正常读取，但是，现在的版本就需要使用 `file` 函数打开并使用 `encoding="GBK"` 说明才能正常读取。比如，GBK 编码的 `bp.csv` 的读取：

```
read.csv(file("data/bp.csv", encoding="GBK"))
```

```
##   序号 收缩压
## 1    1    145
## 2    5    110
## 3    6   未测
## 4    9    150
## 5   10   拒绝
## 6   15    115
```

```
readLines(file("data/bp.csv", encoding="GBK"))
```

```
## [1] "序号,收缩压" "1,145"       "5,110"       "6, 未测"     "9,150"
## [6] "10, 拒绝"    "15,115"
```

以 UTF-8 编码的文件，不论有没有 BOM 开头，在当前的 R 版本中都可以正确读入而不需说明编码：

```
read.csv("data/bp-utf8nobom.csv")
```

```
##   序号 收缩压
## 1    1    145
## 2    5    110
## 3    6   未测
## 4    9    150
## 5   10   拒绝
## 6   15    115
```

```
readLines("data/bp-utf8bom.csv")
```

```
## [1] "序号,收缩压" "1,145"       "5,110"       "6, 未测"     "9,150"
## [6] "10, 拒绝"    "15,115"
```

15.6.2 用 readr 包读取

readr 包的 `read_csv()`、`read_table2()`、`read_lines()` 函数默认从 UTF-8 编码的文件中读取，无 BOM 或者有 BOM 都可以。如：

```r
read_csv("data/bp-utf8nobom.csv")
```

```
## Rows: 6 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): 收缩压
## dbl (1): 序号
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 6 x 2
##    序号 收缩压
##   <dbl> <chr>
## 1     1 145
## 2     5 110
## 3     6 未测
## 4     9 150
## 5    10 拒绝
## 6    15 115
```

```r
read_csv("data/bp-utf8bom.csv")
```

```
## Rows: 6 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): 收缩压
## dbl (1): 序号
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 6 x 2
##    序号 收缩压
##   <dbl> <chr>
## 1     1 145
## 2     5 110
## 3     6 未测
## 4     9 150
```

```
## 5    10 拒绝
## 6    15 115
```

```
read_lines("data/bp-utf8nobom.csv")
```

```
## [1] "序号,收缩压" "1,145"       "5,110"       "6, 未测"     "9,150"
## [6] "10, 拒绝"    "15,115"
```

```
read_lines("data/bp-utf8bom.csv")
```

```
## [1] "序号,收缩压" "1,145"       "5,110"       "6, 未测"     "9,150"
## [6] "10, 拒绝"    "15,115"
```

但是，对 GBK 编码的文件，不能直接读取：

```
read_csv("data/bp.csv")
## Error in nchar(x, "width") :
## invalid multibyte string, element 1
```

```
read_lines("data/bp.csv")
```

```
## [1] "\xd0\xf2\xba\xc5,\xca\xd5\xcb\xf5 " "1,145"
## [3] "5,110"                              "6,  \xb2\xe2"
## [5] "9,150"                              "10, \xbe\xf8"
## [7] "15,115"
```

```
## [1] "\xd0\xf2 \xc5,\xca\xd5\xcb\xf5 " "1,145"
## [3] "5,110"                           "6,  \xe2"
## [5] "9,150"                           "10,  \xf8"
## [7] "15,115"
```

为了读取 GBK(或 GB18030) 编码的文件,在 `read_csv()` 和 `read_lines()` 函数中加入 `locale=locale(encoding="GBK")` 选项，GBK 也可以写成 GB18030：

```
read_csv("data/bp.csv", locale=locale(encoding="GBK"))
```

```
## Rows: 6 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): 收缩压
```

```
## dbl (1): 序号
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## # A tibble: 6 x 2
##    序号 收缩压
##   <dbl> <chr>
## 1     1 145
## 2     5 110
## 3     6 未测
## 4     9 150
## 5    10 拒绝
## 6    15 115
```

```
read_lines("data/bp.csv", locale=locale(encoding="GBK"))
```

```
## [1] "序号,收缩压" "1,145"       "5,110"       "6, 未测"     "9,150"
## [6] "10, 拒绝"    "15,115"
```

15.6.3 输出文件的编码

`write.csv()`、`writeLines()` 生成的含有中文的文件的编码在当前 R 版本 (4.2 以后) 已经使用 UTF-8 编码，在较早的 R 版本中默认为操作系统的默认中文编码。

readr 的 `write_csv()`、`write_lines()` 函数生成的含有中文的文件的编码默认 UTF-8 无 BOM。如

```
write_csv(tibble(" 姓名"=c(" 张三", " 李四")), "tmp.csv")
```

结果生成的文件编码为 UTF-8 无 BOM，这样的文件可以被 R 的 `readr::read_csv()` 正确读取，但是不能被 MS Excel 软件正确读取。

`write_lines()` 输出的文件也是编码为 UTF-8 无 BOM。

`write_excel_csv()` 可以生成带有 UTF-8 有 BOM 的 CSV 文件，这样的文件可以被 MS Office 正确识别：

```
write_excel_csv(tibble(" 姓名"=c(" 张三", " 李四")), "tmp2.csv")
```

15.7 目录和文件管理

目录和文件管理函数:

- `getwd()`—返回当前工作目录。
- `setwd(path)`—设置当前工作目录。
- `list.files()` 或 `dir()`—查看目录中内容。`list.files(pattern='.*[.]r$')` 可以列出所有以 ".r" 结尾的文件。
- `file.path()`—把目录和文件名组合得到文件路径。
- `file.info(filenames)`—显示文件的详细信息。
- `file.exists()`—查看文件是否存在。
- `file.access()`—考察文件的访问权限。
- `create.dir()`—新建目录。
- `file.create()`—生成文件。
- `file.remove()` 或 `unlink()`—删除文件。`unlink()` 可以删除目录。
- `file.rename()`—为文件改名。
- `file.append()`—把两个文件相连。
- `file.copy()`—复制文件。
- `basename()` 和 `dirname()`— 从一个全路径文件名获取文件名和目录。

Chapter 16

程序控制结构

16.1 表达式

R 是一个表达式语言, 其任何一个语句都可以看成是一个表达式。表达式之间以分号分隔或用换行分隔。表达式可以续行, 只要前一行不是完整表达式 (比如末尾是加减乘除等运算符, 或有未配对的括号) 则下一行为上一行的继续。若干个表达式可以放在一起组成一个复合表达式, 作为一个表达式使用，复合表达式的值为最后一个表达式的值，组合用大括号表示, 如：

```
{
  x <- 15
  x
}
```

16.2 分支结构

分支结构包括 if 结构：

> if (条件) 表达式 1

或

> if (条件) 表达式 1 else 表达式 2

其中的 “条件” 为一个标量的真或假值, 不允许取缺失值，表达式可以是用大括号包围的复合表达式。如

```
if(is.na(lambda)) lambda <- 0.5
```

又如

```
if(x>1) {
  y <- 2.5
} else {
  y <- -y
}
```

多个分支，可以在中间增加 `else if`，如：

```
x <- c(0.05, 0.6, 0.3, 0.9)
for(i in seq(along=x)){
  if(x[i] <= 0.2){
    cat("Small\n")
  } else if(x[i] <= 0.8){
    cat("Medium\n")
  } else {
    cat("Large\n")
  }
}
```

请注意这里的多重 if-else 结构的程序缩进和排列方法，这是一种易读且不容易引起误解的写法。

在多重 if-else 结构中，后面的判断，一定是在前面的判断为假的前提下进行判断的，所以上例中间的`"Medium"` 本应写条件为 `x[i]>0.2 & x[i]<=0.8`，但因为是第二个分支，所以前提就是第一个分支的 `x[i]<=0.2` 条件已经被否定。

16.2.1 用逻辑下标代替分支结构

R 是向量化语言，尽可能少用标量运算。比如，x 为一个向量，要定义 y 与 x 等长，且 y 的每一个元素当且仅当 x 的对应元素为正数时等于 1，否则等于零。

这样是错误的：

```
if(x>0) y <- 1 else y <- 0
```

正解为:

```
y <- numeric(length(x))
y[x>0] <- 1
y
```

16.2.2 ifelse 函数

函数 ifelse() 可以根据一个逻辑向量中的多个条件，分别选择不同结果。如

```
x <- c(-2, 0, 1)
y <- ifelse(x >=0, 1, 0); print(y)
```

```
## [1] 0 1 1
```

函数 ifelse(test, yes, no) 中的 test 是逻辑向量，yes 和 no 是向量，test、yes 和 no 的配合符合向量化原则，如果有长度为 1 的或者长度较短但其倍数等于最长一个的长度的，短的一个自动从头循环使用。如：

```
ifelse((1:6) >= 3, 1:2, c(-1,-2))
```

```
## [1] -1 -2  1  2  1  2
```

当然，最常见的还是 yes、no 为标量的情形。

不同于 if 语句，ifelse 的 test 中运行有缺失值，对应结果也是缺失值。

dplyr 包的 if_else 函数与基本 R 的 ifelse 功能基本相同，但允许指定条件为缺失值时的不同处理方法。

16.2.3 switch 函数

函数 switch() 可以建立多分枝结构, 可以根据一个整数表达式或者一个字符串表达式的值选择返还相对应的值。

例如，某校将百分制转换到等级制的规则如下：

- 98-100: A+
- 95-97: A
- 90-94: A-
- 85-89: B+
- 80-84: B
- 77-79: B-
- 73-76: C+
- 70-72: C
- 67-69: C-

- 63-66: D+
- 60-62: D
- 0-59: F

从百分制转换为等级制可以用 `if-else if-else` 完成。从等级制转换回百分制，用区间平均值，可以写成 `switch()` 的写法：

```
gr_v <- c("A+", "A", "A-", "B+", "B", "B-",
  "C+", "C", "C-", "D+", "F", "EX")
for(gr in gr_v){
  switch(gr,
    "A+" = 99,
    "A" = 96,
    "A-" = 92,
    "B+" = 87,
    "B" = 82,
    "B-" = 78,
    "C+" = 75,
    "C" = 71,
    "C-" = 68,
    "D+" = 65,
    "F" = 50,
    NA)
}
```

`switch()` 中的最后一个选择通常用于所有前面选择都不匹配时的一个例外情况处理，这里的例子对这种情况返回缺失值。上例中用了 `for` 循环，见下一节的说明。

`switch()` 中允许对多个值采取同一行动或者返回同一结果，如：

```
for(ch in c("a", "i", "c")){
  cat(ch, ":", switch(ch,
    "a"=, "e"=, "i"=, "o"=, "u" = "Vower",
    "Consonant"), "\n")
}
```

```
## a : Vower
## i : Vower
## c : Consonant
```

在上例中，a, e, i, o, u 返回相同的值`"Vower"`，而其它值则返回`"Consonant"`。

如果要建立 $1:n$ 到 $x_1 \sim x_n$ 的映射，可以用整数下标的方法；如果要建立 $s_1 \sim s_n$(字符串向量）到 $x_1 \sim x_n$ 的映射，可以用有名向量的方法。`switch()` 则可以处理有例外项的问题。更一般的分支，尤其是不同分支执行不同命令的情况，应该使用 `if-else if-else` 结构。

dplyr 包的 `case_when` 函数可以 `switch()` 的向量化推广，看成是 `ifelse` 的多分支推广，或看成 `if-else if-else` 语句的向量化。输入条件是向量化的，根据输入条件返回条件对应的结果。所有条件都不满足的元素的输出结果为 NA‘。

如：

```
v <- c("A+", "A", "A-", "B+", "B", "B-",
  "C+", "C", "C-", "D+", "F", "EX")
dplyr::case_when(
    v == "A+" ~ 99,
    v == "A" ~ 96,
    v == "A-" ~ 92,
    v == "B+" ~ 87,
    v == "B" ~ 82,
    v == "B-" ~ 78,
    v == "C+" ~ 75,
    v == "C" ~ 71,
    v == "C-" ~ 68,
    v == "D+" ~ 65,
    v == "F" ~ 50)
```

```
##  [1] 99 96 92 87 82 78 75 71 68 65 50 NA
```

```
x <- c("a", "i", "c")
dplyr::case_when(
  x %in% c("a", "e", "i", "o", "u")
  ~ "Vowel",
  TRUE
  ~ "Consonant"
)
```

```
## [1] "Vowel"     "Vowel"     "Consonant"
```

16.3 循环结构

16.3.1 计数循环

为了对向量每个元素、矩阵每行、矩阵每列循环处理，语法为

for(循环变量 in 序列) 语句

其中的语句一般是复合语句。如：

```
set.seed(101); x <- rnorm(5)
y <- numeric(length(x))
for(i in 1:5){
  if(x[i]>=0) y[i] <- 1 else y[i] <- 0
}
print(y)
```

```
## [1] 0 1 0 1 1
```

其中 `rnorm(5)` 会生成 5 个标准正态分布随机数。`numeric(n)` 生成有 n 个 0 的数值型向量（基础类型为 double）。

如果需要对某个向量 `x` 按照下标循环，获得所有下标序列的标准写法是 `seq_along(x)`, 而不用 `1:n` 的写法，因为在特殊情况下 `n` 可能等于零，这会导致错误下标，而 `seq_along(x)` 在 `x` 长度为零时返回零长度的下标。

例如，设序列 x_n 满足 $x_0 = 0$, $x_n = 2x_{n-1} + 1$, 求 $S_n = \sum_{i=1}^{n} x_n$:

```
x <- 0.0; s <- 0; n <- 5
for(i in 1:n){
  x <- 2*x + 1
  s <- s + x
}
print(s)
```

```
## [1] 57
```

在 R 中应尽量避免 for 循环：其速度比向量化版本慢一个数量级以上，而且写出的程序不够典雅。比如，前面那个示性函数例子实际上可以简单地写成

```
set.seed(101); x <- rnorm(5)
y <- ifelse(x >= 0, 1, 0)
print(y)
```

```
## [1] 0 1 0 1 1
```

许多计数循环都可以用 `lapply`、`vapply`、`sapply`、`apply`、`map`、`Map` 等函数替代，详见18.3。

用 `break` 语句退出所在的循环。用 `next` 语句进入所在循环的下一轮。

使用 for 循环的注意事项：

- 如果对向量每个元素遍历并保存结果，应在循环之前先将结果变量产生等长的存储，在循环内为已经分配好存储空间的输出向量的元素赋值。为了产生长度为 n 的数值型向量，用 numeric(n)；为了产生长度为 n 的列表，用 vector("list", n)。
- 对一个向量元素遍历时如果用下标访问，需要用 seq_along(x) 的做法而不是 1:length(x) 的做法。
- 如果直接对向量元素遍历，这有可能会丢失向量的属性（如日期型），用下标访问则不存在此问题。

如：

```
x <- as.POSIXct(c("1981-05-31", "2020-02-22"))
for(xi in x){print(xi)}
```

```
## [1] 360086400
## [1] 1582300800
```

```
for(i in seq_along(x)){print(x[i])}
```

```
## [1] "1981-05-31 CST"
## [1] "2020-02-22 CST"
```

16.3.2 while 循环和 repeat 循环

用

> while(循环继续条件) 语句

进行当型循环。其中的语句一般是复合语句。仅当条件成立时才继续循环，而且如果第一次条件就已经不成立就一次也不执行循环内的语句。

用

> repeat 语句

进行无条件循环（一般在循环体内用 if 与 break 退出）。其中的语句一般是复合语句。如下的写法可以制作一个直到型循环：

```
repeat{
  ...
  if(循环退出条件) break
}
```

直到型循环至少执行一次，每次先执行... 代表的循环体语句，然后判断是否满足循环退出条件，满足条件就退出循环。

例如，常量 e 的值可以用泰勒展开式表示为

$$e = 1 + \sum_{k=1}^{\infty} \frac{1}{k!}$$

R 函数 `exp(1)` 可以计算 `e` 的为了计算 e 的值，下面用泰勒展开逼近计算 `e` 的值：

```
e0 <- exp(1.0)
s <- 1.0
x <- 1
k <- 0
repeat{
  k <- k+1
  x <- x/k
  s <- s + x

  if(x < .Machine$double.eps) break
}
err <- s - e0
cat("k=", k, " s=", s, " e=", e0, " 误差 =", err, "\n")
```

```
## k= 18  s= 2.718282  e= 2.718282  误差= 4.440892e-16
```

其中`.Machine$double.eps` 称为机器 ε，是最小的加 1 之后可以使得结果大于 1 的正双精度数，小于此数的正双精度数加 1 结果还等于 1。用泰勒展开公式计算的结果与 `exp(1)` 得到的结果误差在 10^{-16} 左右。

16.4 R 中判断条件

if 语句和 while 语句中用到条件。条件必须是标量值，而且必须为 TRUE 或 FALSE，不能为 NA 或零长度。这是 R 编程时比较容易出错的地方。

16.5 管道控制

数据处理中经常会对同一个变量（特别是数据框）进行多个步骤的操作，比如，先筛选部分有用的变量，再定义若干新变量，再排序。R 从 4.1.0 版本开始提供了一个 `|>` 运算符实现这样的操作流程，R 的 magrittr 包提供了一个`%>%` 运算符执行类似功能。比如，变量 `x` 先用函数 `f(x)` 进行变换，再用函数 `g(x)` 进行变换，一般应该写成 `g(f(x))`，用 `|>` 运算符，可以表示成 `x |> f() |> g()`。更多的处理，如 `h(g(f(x)))` 可以写成 `x |> f() |> g() |> h()`。这样的表达更符合处理发生的次序，而且插入一个处理步骤也很容易。

处理用的函数也可以带有其它自变量，在管道控制中不要写第一个自变量。

例如：

```
2 |> sqrt() |> exp()
```

```
## [1] 4.11325
```

结果为 $e^{\sqrt{2}}$。

又如：

```
asin(0.5) |> round(digits=4)
```

```
## [1] 0.5236
```

这等价于 `round(asin(0.5), digits=4)`。

在 `|>` 表示的管道中，上一步的输出通过管道作为下一步的第一个自变量。如果需要当作下一步的其它自变量，可以用自变量名 `= _` 的特殊格式，这里 `_` 代表管道中上一步的结果。如：

```
6 |> round(pi,  digits = _)
```

```
## [1] 3.141593
```

这等价于 `round(pi, digits=6)`。

tibble 类型的数据框尤其适用于如此的管道操作，在23中有大量使用管道进行多步骤操作的例子。

magrittr 包定义了`%T%` 运算符，`x %T% f()` 返回 `x` 本身而不是用 `f()` 修改后的返回值 `f(x)`，这在中间步骤需要显示或者绘图但是需要进一步对输入数据进行处理时有用。

magrittr 包定义了`%$%` 运算符，此运算符的作用是将左运算元的各个变量（这时左运算元是数据框或列表）暴露出来，可以直接在右边调用其中的变量，类似于 `with()` 函数的作用。

magrittr 包定义了`%<>%` 运算符，用在管道链的第一个连接，可以将处理结果存入最开始的变量中，类似于 C 语言的 `+=` 运算符。

如果一个操作是给变量加 `b`，可以写成 `add(b)`，给变量乘 `b`，可以写成 `multiply_by(b)`。

Chapter 17

函数

17.1 函数基础

17.1.1 介绍

在现代的编程语言中使用自定义函数，优点是代码复用、模块化设计。

如果一段程序需要在多处使用，就应该将其写成一个函数，然后在多处调用。需要修改程序执行功能时，仅需要修改函数而不需要修改各处调用。如果不使用函数而是将相同的代码在多处复制粘贴，修改时就需要修改多处，很容易漏掉一处。

在编程时，把编程任务分解成小的模块，每个模块用一个函数实现，便于理解每个模块的作用，降低了程序复杂性，使得程序容易管理。

函数的自变量是只读的，函数中定义的局部变量只在函数运行时起作用，不会与外部或其它函数中同名变量混杂，所以使用函数还减少在很长的程序中变量互相混淆出错的危险。

函数返回一个对象作为输出，如果需要返回多个变量，可以用列表进行包装。

17.1.2 函数定义

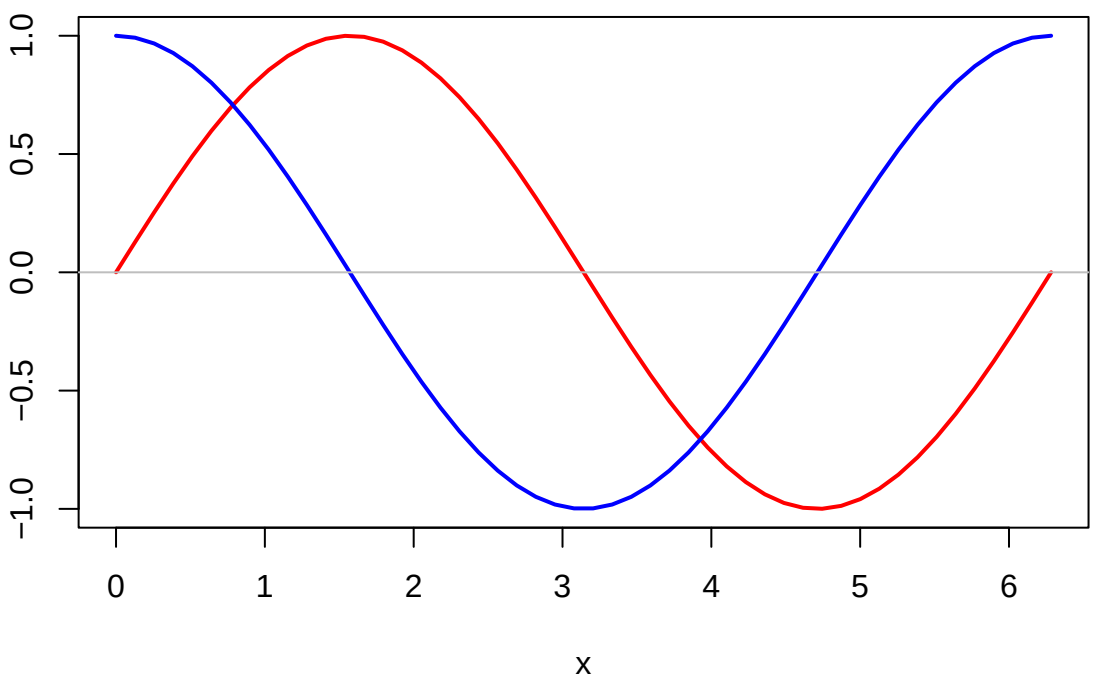
函数定义使用 `function` 关键字，一般格式为

函数名 `<-` function(形式参数表) 函数体

函数体是一个表达式或复合表达式（复合语句），以复合表达式中最后一个表达式为返回值，也可以用 `return(x)` 返回 `x` 的值。如果函数需要返回多个结果，可以打包在一个列表（list）中返回。形式参数表相当于函数自变量，可以是空的，形式参数可以有缺省值，R 的函数在调用时都可以用“形式参数名 = 实际参数”的格式输入自变量值。

下面的例子没有参数，仅画一个示例图：

```
f <- function() {
  x <- seq(0, 2*pi, length=50)
  y1 <- sin(x)
  y2 <- cos(x)
  plot(x, y1, type="l", lwd=2, col="red",
       xlab="x", ylab="")
  lines(x, y2, lwd=2, col="blue")
  abline(h=0, col="gray")
}
f()
```



注意此自定义函数虽然没有参数，但是在定义与调用时都不能省略圆括号。

自定义函数也可以是简单的一元函数，与数学中一元函数基本相同，例如

```
f <- function(x) 1/sqrt(1 + x^2)
```

从 R 4.1.0 版本开始，简单的 R 函数还可以写成`\(x)` 表达式的形式，所以上面的例子可以简写为

```
f <- \(x) 1/sqrt(1 + x^2)
```

这个函数基本与数学函数 $f(x) = 1/\sqrt{1+x^2}$ 相对应。定义中的自变量 `x` 叫做形式参数或形参 (formal arguments)。函数调用时，形式参数得到实际值，叫做实参 (actual arguments)。

R 函数有一个向量化的好处，在上述函数调用时，如果形式参数 `x` 的实参是一个向量，则结果也是向量，结果元素为实参向量中对应元素的变换值。如

```
f(0)
```

```
## [1] 1
```

```
f(c(-1, 0, 1, 2))
```

```
## [1] 0.7071068 1.0000000 0.7071068 0.4472136
```

第一次调用时，形式参数 `x` 得到实参 0，第二次调用时，形式参数 `x` 得到向量实参 `c(-1, 0, 1, 2)`。

函数实参是向量时，函数体中也可以计算对向量元素进行汇总统计的结果。例如，设 $x_1, x_2, \ldots, x_n$ 是一个总体的简单随机样本，其样本偏度统计量定义如下：

$$\hat{w} = \frac{n}{(n-1)(n-2)} \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{S} \right)^3$$

其中 $\bar{x}$ 与 S 分别是样本均值与样本标准差。如下的 R 函数可以把观测样本的值保存在一个向量中输入，计算并输出其样本偏度统计量值：

```
skewness <- function(x) {
  n <- length(x)
  xbar <- mean(x)
  S <- sd(x)
  n/(n-1)/(n-2)*sum( (x - xbar)^3 ) / S^3
}
```

函数体的最后一个表达式是函数返回值。

除了用函数体的最后一个表达式作为返回值，还可以用 `return(y)` 的方式在函数体的任何位置退出函数并返回 `y` 的值。为了返回多个变量值，将这些变量打包为一个列表返回即可；R 的统计建模函数的返回值大多数都是列表。

函数可以返回一个 `invisible(y)`，这表示其返回的值仍是 `y` 的值，但直接在 R 命令行调用此函数时不自动显示返回值，`print()` 或 `cat()` 显式地要求才显示。当预期返回的结果显示无意义或者显示在命令行会产生大量的输出时可以用此方法。

上面计算样本偏度的程序中函数体最后一个表达式中巧妙地利用了 R 的向量化运算（`(x - xbar)^3`）与内建函数（`sum`）。这比用 `for` 循环计算效率高得多，计算速度相差几十倍。

关于程序效率，请比较如下两个表达式：

```
  n/(n-1)/(n-2)*sum( (x - xbar)^3 ) / S^3
  n/(n-1)/(n-2)*sum( ((x - xbar)/S)^3 )
```

这两个表达式的值相同。表面上看，第二个表达式更贴近原始数学公式，但是在编程时，需要考虑计算效率问题，第一个表达式关于 S 只需要除一次，而第二个表达关于 S 除了 n 次，所以第一个表达式效率更高。一个函数如果仅仅用几次，这些细微的效率问题不重要，但是如果要编写一个 R 扩展包提供给许多人使用，程序效率就是重要的问题。参见18。

函数定义中的形式参数可以有多个，还可以指定缺省值。例如

```
fsub <- function(x, y=0){
  cat("x=", x, " y=", y, "\n")
  x - y
}
```

这里 x, y 是形式参数，其中 y 指定了缺省值为 0，有缺省值的形式参数在调用时可以省略对应的实参，省略时取缺省值。

一个自定义 R 函数由三个部分组成：

- 函数体 body()，即要函数定义内部要执行的代码；
- formals()，即函数的形式参数表以及可能存在的缺省值，这是一个列表对象；
- environment()，是函数定义时所处的环境，这会影响到参数表中缺省值与函数体中非局部变量的的查找。

注意，函数名并不是函数对象的必要组成部分。提取三个部分如

```
body(fsub)
```

```
## {
##     cat("x=", x, " y=", y, "\n")
##     x - y
## }
```

```
formals(fsub)
```

```
## $x
##
##
## $y
## [1] 0
```

```
environment(fsub)
```

```
## <environment: R_GlobalEnv>
```

“环境”是 R 语言比较复杂的概念，对于没有嵌套定义在函数内的函数，环境一般是 R 的全局工作空间（全局环境）；嵌套定义的函数则会有一个私有的环境，而且对于利用“函数工厂”生成的函数，还可以将其私有环境与函数对象一起保存下来，生成带有状态的函数。关于这些用法后面再详细解释。

实际上，“`function(参数表) 函数体`”这样的结构本身也是一个表达式，其结果是一个函数对象。在通常的函数定义中，函数名只不过是被赋值为某个函数对象，或者说是“绑定”(bind) 到某个函数对象上面。同一个函数对象可以被多个函数名绑定。函数是普通 R 对象，在编程语言术语中称这样的函数为**第一级函数** (first class functions)，或函数是**第一级对象** (first class objects)，即函数在 R 语言中与其他普通数值型对象、字符型对象有相同的地位。

因为函数也是 R 对象，也可以拥有属性。所谓对象，就是 R 的变量所指向的各种不同类型的统称。

可以将多个函数存放在一个列表中。例如，在用随机模拟比较不同的统计模型时，常常将要对一组数据采用的多个并行的建模函数存放在列表中，对许多组模拟数据的每一组用循环的方法应用列表中的每一个建模函数分别得到结果。

17.1.3 函数调用

函数调用时最基本的调用方式是把实参与形式参数按位置对准，这与我们在数学中使用多元函数的习惯类似。例如

```
fsub(3, 1)
## x= 3  y= 1
## [1] 2
```

相当于以 `x=3, y=1` 调用。

调用时可选参数可以省略实参，如

```
fsub(3)
## x= 3  y= 0
## [1] 3
```

相当于以 `x=3, y=0` 调用。

R 函数调用时全部或部分形参对应的实参可以用“形式参数名 = 实参”的格式给出，这样格式给出的实参不用考虑次序，不带形式参数名的则按先后位置对准。

例：

```
fsub(x=3, y=1)
## x= 3  y= 1
## [1] 2
fsub(y=1, x=3)
## x= 3  y= 1
## [1] 2
fsub(x=3)
## x= 3  y= 0
## [1] 3
fsub(3, y=1)
## x= 3  y= 1
## [1] 2
fsub(1, x=3)
## x= 3  y= 1
## [1] 2
fsub(x=3, 1)
## x= 3  y= 1
## [1] 2
```

在调用函数时，如果以“形参名 = 实参值”的格式输入参数，则“形参名”与定义时的形参名完全匹配时最优先采用；如果“形参名”是定义时的形参名的前一部分子串，即部分匹配，这时调用表中如果没有其它部分匹配，也可以输入到对应的完整形参名的参数中；按位置匹配是最后才进行的。有缺省值的形参在调用时可省略。

形参的部分匹配虽然可以节省一丁点儿键入工作量，但是很不严谨，容易出错，所以应避免使用。现在不能关闭这种语法，可以用

```
options(warnPartialMatchArgs = TRUE)
```

在使用了这种调用方法时发出警告信息。

作为好的程序习惯，调用 R 函数时，如果既有按位置对应的参数又有带名参数，应仅有一个或两个是按位置对应的，按位置对应的参数都写在前面，带名参数写在后面，按位置对应的参数在参数表中的位置应与定义时的位置一致。在定义函数时，没有缺省值的参数写在前面，有缺省值的参数写在后面。不遵守这样的约定容易使得程序被误读，有时会在运行时匹配错位。

R 的形参、实参对应关系可以写成一个列表，如 `fsub(3, y=1)` 中的对应关系可以写成列表 `list(3, y=1)`，如果调用函数的形参、实参对应关系保存在列表中，可以用函数 `do.call()` 来表示函数调用，如

```
do.call(fsub, list(3, y=1))
```

与

```
fsub(3, y=1)
```

效果相同。

17.1.4 函数复合调用与管道运算符

函数的复合调用，如 $\sin\sqrt{x}$，可以写成 `sin(sqrt(x))`。R 在 4.1.0 版本以后定义了一个管道运算符 "`|>`"，在 magrittr 包中也定义了类似的运算符 "`%>%`"，可以使得这种复合调用按照正常的执行次序来写，变成

```
x |>
  sqrt() |>
  sin()
```

复合调用的函数也可以带有选项，比如 $\sum_{k=1}^{n}\log_b x_k$，用普通复合方法写成

```
sum(log(x, b))
```

可以用管道运算符写成：

```
x |>
  log(base = b) |>
  sum()
```

在自变量比较简单时这种写法的优势不明显，在对数据框或 tibble 进行多步骤的筛选、变换时，总是将变换的中间结果数据框传递给下一个步骤，这种写法能清楚地展示处理是如何一步一步执行的。

如果需要传递给下一个函数的自变量不是第一个自变量，可以用 `_` 指代上一步传入的自变量值，如 $\sum_{k=1}^{n}\log_{x_k} a$，复合函数写法为 `sum(log(a, base=x))`，用管道运算符写成：

```
x |>
  log(a, base = _) |>
  sum()
```

管道运算在表示对一个输入使用多个步骤逐次进行处理时，如果每一个步骤都是函数调用，就比较方便编程也容易理解。但是，如果中间某一步没有现成的函数，就要写成无名函数格式，程序会显得略为晦涩难懂。

比如，这样的步骤：

```
x <- c(2,1,3,5,4)
x1 <- diff(x)
x2 <- x1 > 0
```

```
x3 <- as.numeric(x2)
x4 <- c(0, x3); x4
## [1] 0 0 1 1 0
```

改用管道运算符表示，则 x2 和 x4 这两步不是现成的函数，需要写成无名函数调用格式：

```
x <- c(2,1,3,5,4)
x |>
  diff() |>
  (\(x) x > 0)() |>
  as.numeric() |>
  (\(x) c(0, x))()
## [1] 0 0 1 1 0
```

17.1.5 递归调用

在函数内调用自己叫做递归调用。递归调用可以使得许多程序变得简单，但是往往导致程序效率很低，需谨慎使用。

R 中在递归调用时，最好用 Recall 代表调用自身，这样保证函数即使被改名（在 R 中函数是一个对象，改名后仍然有效）递归调用仍指向原来定义。

斐波那契数列是如下递推定义的数列：

$$x_0 = 0, \quad x_1 = 1$$
$$x_n = x_{n-2} + x_{n-1}$$

这个数列可以用如下递归程序自然地实现：

```
fib1 <- function(n){
  if(n == 0) return(0)
  else if(n == 1) return(1)
  else if(n >=2 ) {
    return(Recall(n-1) + Recall(n-2))
  }
}
for(i in 0:10) cat("i =", i, " x[i] =", fib1(i), "\n")
```

```
## i = 0  x[i] = 0
## i = 1  x[i] = 1
## i = 2  x[i] = 1
## i = 3  x[i] = 2
## i = 4  x[i] = 3
## i = 5  x[i] = 5
```

```
## i = 6  x[i] = 8
## i = 7  x[i] = 13
## i = 8  x[i] = 21
## i = 9  x[i] = 34
## i = 10  x[i] = 55
```

17.1.6 向量化

自定义的函数，如果其中的计算都是向量化的，那么函数自动地可以接受向量作为输入，结果输出向量。比如，将每个元素都变成原来的平方的函数:

```
f <- function(x){
  x^2
}
```

如果输入一个向量，结果也是向量，输出的每个元素是输入的对应元素的相应的平方值。

但是，如下的分段函数：

$$g(x) = \begin{cases} x^2, & |x| \leq 1, \\ 1, & |x| > 1 \end{cases}$$

其一元函数版本可以写成

```
g <- function(x){
  if(abs(x) <= 1) {
    y <- x^2
  } else {
    y <- 1
  }

  y
}
```

但是这个函数不能处理向量输入，因为 `if` 语句的条件必须是标量条件。一个容易想到的修改是

```
gv <- function(x){
  y <- numeric(length(x))
  sele <- abs(x) <= 1
  y[sele] <- x[sele]^2
  y[!sele] <- 1.0
```

```
  y
}
```

或者

```
gv <- function(x){
  ifelse(abs(x) <= 1, x^2, 1)
}
```

对于没有这样简单做法的问题，可以将原来的逻辑包在循环中，如

```
gv <- function(x){
  y <- numeric(length(x))
  for(i in seq(along=x)){
    if(abs(x[i]) <= 1) {
      y[i] <- x[i]^2
    } else {
      y[i] <- 1
    }
  }

  y
}
```

函数 `Vectorize` 可以将这样的操作自动化。如

```
g <- function(x){
  if(abs(x) <= 1) {
    y <- x^2
  } else {
    y <- 1
  }

  y
}
gv <- Vectorize(g)
gv(c(-2, -0.5, 0, 0.5, 1, 1.5))
```

```
## [1] 1.00 0.25 0.00 0.25 1.00 1.00
```

还可以使用 `purrr::map()` 或基本 R 的 `lapply()` 等泛函实现对各个元素的函数变换。参见25。

17.1.7 无名函数

R 允许使用没有函数名的函数对象, `lapply` 类的函数经常使用无名的函数对象作为输入。例如：

```
vapply(iris[,1:4], function(x) max(x) - min(x), 0.0)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##          3.6          2.4          5.9          2.4
```

```
integrate(function(x) sin(x)^2, 0, pi)
```

```
## 1.570796 with absolute error < 1.7e-14
```

iris 是 R 中的一个例子数据框，有 150 个观测，前 4 个变量是数值型的，最后一个变量 `Species` 是有三个水平的因子。第一个语句对前 4 列分别计算极差。第二个语句用数值积分方法计算 $\int_0^\pi \sin^2(x)\,dx$。两个语句都用到了无名函数。

`vapply` 这样的函数称为“泛函”，参见25。

R 为无名函数提供了`\(x)` 函数体这样的简写格式，如：

```
vapply(iris[,1:4], \(x) max(x) - min(x), 0.0)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##          3.6          2.4          5.9          2.4
```

```
integrate(\(x) sin(x)^2, 0, pi)
```

```
## 1.570796 with absolute error < 1.7e-14
```

17.2 变量作用域

函数中变量的作用域，是指在函数中用到的变量名如何找到对应的值，以及如何确定某个变量绑定适用的范围。

17.2.1 全局变量和工作空间

在所有函数外面（如 R 命令行）定义的变量是全局变量。在命令行定义的所有变量都保存在工作空间（workspace），也称为全局环境中。

在 RStudio 的 Environment 窗格可以看到“Global Environment” 的内容，分为数据框 (Data)、其它变量 (Values) 和函数 (Functions) 三类。

在命令行可以用 `ls()` 查看工作空间内容。`ls()` 中加上 `pattern` 选项可以指定只显示符合一定命名模式的变量，如

```
ls(pattern="^tmp[.]")
```

显示所有以 `tmp.` 开头的变量。用 `object.size()` 函数查看变量占用存储大小，单位为字节。

因为 R 的函数调用时可以读取工作空间中的全局变量值，工作空间中过多的变量会引起莫名其妙的程序错误。用 `rm()` 函数删除指定的变量，比如

```
rm(x, z41)
```

`rm()` 中还可以用 `list` 参数指定一个要删除的变量名表。如

```
rm(list=ls(pattern="^tmp[.]"))
```

用 `save()` 函数保存工作空间中选择的某些变量；用 `load()` 函数载入保存在文件中的变量。如

```
save(my.large.data,
     file="my-large-data.RData")
load("my-large-data.RData")
```

实际上，R 的工作空间是 R 的变量搜索路径中的一层，大体相当于全局变量空间。R 的已启用的软件包中的变量以及用 `attach()` 命令引入的变量也在这个搜索路径中。用 `search()` 返回当前的搜索路径。

17.2.2 局部变量

在一般计算机语言中，“变量” 实际是计算机内存中的一段存储空间，但是 R 中略微复杂一些，R 的变量实际是指向 R 对象的引用，称为 “绑定”。在较简单的函数定义中大体上可以将 R 变量看成是对应的存储空间。

函数的参数（自变量）在定义函数时并没有对应的存储空间，所以也称函数定义中的参数为 “形式参数”。

函数的形式参数在调用时被赋值为实参值（这是一般情形），形参变量和函数体内被赋值的变量都是局部的。这一点符合函数式编程 (functional programming) 的要求。所谓局部变量，就是仅在函数运行时才存在，一旦函数运行结束就不存在的变量。

17.2.2.1 自变量的局部性

在函数被调用时，形式参数（自变量）被赋值为实际的值（称为实参），如果实参是变量，形式参数可以看作实参的一个副本，除了极少数的特例（如环境），在函数内部对形式参数作任何修改在函数运行完成后都不影响原来的实参变量，而且函数运行完毕后形式参数不再与实际的存储空间联系。

在下例中，在命令行定义了全局变量 `xv`, `xl`, 然后作为函数 `f()` 的自变量值（实参）输入到函数中，函数中对两个形式参数作了修改，函数结束后实参变量 `xv`, `xl` 并未被修改，形参变量也消失了。例子程序如下：

```
xv <- c(1,2,3)
xl <- list(a=11:15, b="James")
if(exists("x")) rm(x)
f <- function(x, y){
  cat(" 输入的 x=", x, "\n")
  x[2] <- -1
  cat(" 函数中修改后的 x=", x, "\n")
  cat(" 输入的 y 为:\n"); print(y)
  y[[2]] <- "Mary"
  cat(" 函数中修改过的 y 为:\n"); print(y)
}
f(xv, xl)
## 输入的 x= 1 2 3
## 函数中修改后的 x= 1 -1 3
## 输入的 y 为:
## $a
## [1] 11 12 13 14 15
##
## $b
## [1] "James"
##
## 函数中修改过的 y 为:
## $a
## [1] 11 12 13 14 15
##
## $b
## [1] "Mary"
##
cat(" 函数运行完毕后原来变量 xv 不变: ", xv, "\n")
## 函数运行完毕后原来变量 xv 不变:  1 2 3
cat(" 函数运行完毕后原来变量 xl 不变: :\n"); print(xl)
## 函数运行完毕后原来变量 xl 不变: :
## $a
## [1] 11 12 13 14 15
##
## $b
## [1] "James"
##
cat(" 函数运行完毕后形式参数 x 不存在: :\n"); print(x)
```

```
## 函数运行完毕后形式参数 x 不存在：：
## Error in print(x) : object "x" not found
```

R 语言的这种特点对于传递超大的数据是不利的，所以 R 中会容纳超大数据的类型往往设计成修改副本时不占用不必要的额外存储空间，比如，tibble 类型就有这样的特点。

17.2.2.2 修改自变量

为了修改某个自变量，在函数内修改其值并将其作为函数返回值，赋值给原变量。

比如定义了如下函数：

```
f <- function(x, inc=1){
  x <- x + inc
  x
}
```

调用如

```
x <- 100
cat(" 原始 x=", x, "\n")
```

```
## 原始 x= 100
```

```
x <- f(x)
cat(" 修改后 x=", x, "\n")
```

```
## 修改后 x= 101
```

17.2.2.3 函数内的局部变量

在函数内部用赋值定义的变量都是局部变量，即使在工作空间中有同名的变量，此变量在函数内部被赋值时就变成了局部变量，原来的全局变量不能被修改。这种规则称为**掩藏** (masking)。之所以如此，是因为赋值本质上是“绑定”，函数内部属于与工作空间（全局环境）不同的另一个环境，一个变量一旦在函数内被重新绑定，它就变成了仅在函数内部能被访问的变量，即局部变量，与原来在工作空间中同名的变量也没有关系了。局部变量在函数运行结束后就会消失。如

```
if("x" %in% ls()) rm(x)
f <- function(){
  x <- 123
```

```
  cat(" 函数内: x = ", x, "\n")
}
f()
cat(" 函数运行完毕后: x=", x, "\n")
## 函数内: x =  123
> cat(" 函数运行完毕后: x=", x, "\n")
## Error in cat(" 函数运行完毕后: x=", x, "\n") : object "x" not found
```

再比如，下面的函数试图知道自己被调用了多少次，但是因为每次函数调用完毕局部变量就消失，再次调用时的局部变量已经对应到全新的存储空间，所以如下的程序不能达到目的：

```
f <- function(){
  if(!exists("runTimes")){
    runTimes <- 1
  } else {
    runTimes <- runTimes + 1
  }
  print(runTimes)
}
f()
```

```
## [1] 1
```

```
f()
```

```
## [1] 1
```

虽然这个问题可以利用将调用次数保存在全局变量中解决，但是应尽可能少用全局变量；用 R 的闭包 (closure) 可以比较完美地解决这样的问题。

17.2.3 在函数内访问全局变量

函数内部可以读取全局变量的值，但一般不能修改全局变量的值。在现代编程指导思想中，全局变量容易造成不易察觉的错误，应谨慎使用，当然，也不是禁止使用，有些应用中不使用全局变量会使得程序更复杂且低效。

在下面的例子中，在命令行定义了全局变量 `x.g`，在函数 `f()` 读取了全局变量的值，但是在函数内给这样的变量赋值，结果得到的变量就变成了局部变量，全局变量本身不被修改：

```
x.g <- 9999
f <- function(x){
  cat(" 函数内读取: 全局变量 x.g = ", x.g, "\n")
  x.g <- -1
  cat(" 函数内对与全局变量同名的变量赋值:  x.g = ", x.g, "\n")
}
f()
```

```
## 函数内读取: 全局变量 x.g =  9999
## 函数内对与全局变量同名的变量赋值:  x.g =  -1
```

```
cat(" 退出函数后原来的全局变量不变:  x.g =", x.g, "\n")
```

```
## 退出函数后原来的全局变量不变:  x.g = 9999
```

在函数内部如果要修改全局变量的值，用 <<-代替 <-进行赋值。如

```
x.g <- 9999
f <- function(x){
  cat(" 函数内读取: 全局变量 x.g = ", x.g, "\n")
  x.g <<- -1
  cat(" 函数内用"<<-" 对全局变量变量赋值:  x.g = ", x.g, "\n")
}
f()
```

```
## 函数内读取: 全局变量 x.g =  9999
## 对全局变量变量赋值:  x.g =  -1
```

```
cat(" 退出函数后原来的全局变量被修改了:  x.g =", x.g, "\n")
```

```
## 退出函数后原来的全局变量被修改了:  x.g = -1
```

后面将进一步解释函数在嵌套定义时 <<-的不同含义。

Chapter 18

R 程序效率

18.1 R 的运行效率

R 是解释型语言，在执行单个运算时，效率与编译代码相近；在执行迭代循环时，效率较低，与编译代码的速度可能相差几十倍。在循环中对变量进行修改尤其低效，因为 R 在修改某些数据类型的子集时会复制整个数据对象。R 以向量、矩阵为基础运算单元，在进行向量、矩阵运算时效率很高，应尽量采用向量化编程。

另外，R 语言的设计为了方便进行数据分析和统计建模，有意地使语言特别灵活，比如，变量为动态类型而且内容与类型均可修改，变量查找在当前作用域查找不到可以向上层以及扩展包中查找，函数调用时自变量仅在使用其值时才求值（懒惰求值），这样的设计都为运行带来了额外的负担，使得运行变慢。

在计算总和、元素乘积或者每个向量元素的函数变换时，应使用相应的函数，如 `sum`, `prod`, `sqrt`, `log` 等。

对于从其它编程语言转移到 R 语言的学生，如果不细究 R 特有的编程模式，编制的程序可能效率比正常 R 程序慢上几十倍，而且繁琐冗长。

为了提高 R 程序的运行效率，需要尽可能利用 R 的向量化特点，尽可能使用已有的高效函数，必要时可以把运行速度瓶颈部分改用 C++、FORTRAN 等编译语言实现，可以用 R 的 profiler 工具查找运行瓶颈。对于大量数据的长时间计算，可以借助于现代的并行计算工具。

如何决定是否要修改程序以提高效率？对已有的程序，仅在运行速度不满意时才需要进行改进，否则没必要花费宝贵的时间用来节省几秒钟的计算机运行时间。

要改善运行速度，首先要找到运行的瓶颈，这可以用专门的性能分析（profiling）功能实现。R 软件中的 `Rprof()` 函数可以执行性能分析的数据收集工作，收集到的性能数据用 `summaryRprof()` 函数可以显示运行最慢的函数。如果使用 RStudio 软件，可以用 `Profile` 菜单执行性能数据收集与分析，可以在图形界面中显示程序中哪些部分运行花费时间最多。

在改进已有程序的效率时，第一要注意的就是不要把原来的正确算法改成一个速度更快但是结果错误的算法。这个问题可以通过建立试验套装，用原算法与新算法同时试验看结果是否一致来避免。多种解决方案的正确性都可以这样保证，也可以比较多种解决方案的效率。

本章后面部分描述常用的改善性能的方法。对于涉及到大量迭代的算法，如果用 R 实现性能太差不能满足要求，可以改成 C++ 编码，用 Rcpp 扩展包连接到 R 中。Rcpp 扩展包的使用将单独讲授。

R 的运行效率也受到内存的影响，占用内存过多的算法有可能受到物理内存大小限制无法运行，过多复制也会影响效率。

如果要实现一个比较单纯的不需要利用 R 已有功能的算法，发现用 R 计算速度很慢的时候，也可以考虑先用 Julia 语言实现。Julia 语言设计比 R 更先进，运算速度快得多，运算速度经常能与编译代码相比，缺点是刚刚诞生几年的时间，可用的软件包还比较少。

18.2 性能比较和向量化编程示例

充分利用 R 的向量化编程和内建函数可以提高程序效率。R 函数 `system.time()` 可以测量某个表达式的运行时间，`proc.time()` 可以用作会话期间的计时时钟，扩展包 bench 可以用来比较不同表达式的运行时间。

例 18.1. 假设要计算如下的统计量：

$$w = \frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{m}|,$$

其中 $x_1, x_2, \ldots, x_n$ 是某总体的样本，$\hat{m}$ 是样本中位数。比较不同的编程方法。

用传统的编程风格，把这个统计量的计算变成一个 R 函数，可能会写成：

```
mad_f1 <- function(x){
  n <- length(x)
  mhat <- median(x)
  s <- 0.0
  for(i in 1:n){
    s <- s + abs(x[i] - mhat)
  }
  s <- s/n
  return(s)
}
```

用 R 的向量化编程，函数体只需要一个表达式：

```
mad_f2 <- function(x) mean( abs(x - median(x)) )
```

其中 `x - median(x)` 利用了向量与标量运算结果是向量每个元素与标量运算的规则，`abs(x - median(x))` 利用了 `abs()` 这样的一元函数如果以向量为输入就输出每个元素的函数值组成的向量的规则，`mean(...)` 避免了求和再除以 `n` 的循环也不需要定义多余的变量 `n`。

显然，第二种做法的程序比第一种做法简洁的多，如果多次重复调用，第二种做法的计算速度比第一种要快几十倍甚至上百倍。在 R 中，用 `system.time()` 函数可以求某个表达式的计算时间，返回结果的第 3 项是流逝时间。下面对 `x` 采用 10000 个随机数，并重复计算 1000 次，比较两个程序的效率：

```
nrep <- 1000
x <- runif(10000)
y1 <- numeric(nrep); y2 <- y1
system.time(for(i in 1:nrep) y1[i] <- mad_f1(x) )[3]
## elapsed
##    1.19
system.time(for(i in 1:nrep) y1[i] <- mad_f2(x) )[3]
## elapsed
##    0.22
```

速度相差数倍。

用 `system.time()` 或者 `proc.time()` 测量运行时间会受到后台正在运行的其它任务的干扰。有一个 R 扩展包 bench 可以多次运行表达式，从而比较两个或多个表达式的运行时间（类似的包还有 microbenchmark）。如:

```
x <- runif(10000)
bench::mark(
  mad_f1(x),
  mad_f2(x)
)
## Unit: microseconds
##   expr      min       lq      mean   median        uq      max neval cld
##  f1(x) 1321.802 1519.151 1642.8350 1555.001 1590.5510 5826.800   100   b
##  f2(x)  275.801  312.951  330.2541  322.501  326.9515  781.001   100  a
```

因为测试运行受到操作系统中同时运行的其它任务的干扰，所以我们不应该看平均时间，而应该看中位数，或者最小时间（最快可能）。就运行时间中位数（单位：毫秒）来看，`f2()` 比 `f1()` 快大约 4 倍。

例 18.2. 假设要编写函数计算

$$f(x) = \begin{cases} 1 & x \geq 0, \\ 0 & \text{其它} \end{cases}$$

如何向量化编程？

利用传统思维，程序写成

```
pos_f1 <- function(x){
    n <- length(x)
    y <- numeric(n)

    for(i in seq_along(x)){
        if(x[i] >= 0) y[i] <- 1
```

```
        else y[i] <- 0
    }

    y
}
```

实际上，`y <- numeric(n)` 使得 `y` 的每个元素都初始化为 0，所以程序中 `else y[i] <- 0` 可以去掉。

利用向量化与逻辑下标，程序可以写成:

```
pos_f2 <- function(x){
    n <- length(x)
    y <- numeric(n)
    y[x >= 0] <- 1

    y
}
```

但是，利用 R 中内建函数 `ifelse()`，可以把函数体压缩到仅用一个语句：

```
pos_f3 <- function(x) ifelse(x >= 0, 1, 0)
```

例 18.3. 考虑一个班的学生存在生日相同的概率。假设一共有 365 个生日 (只考虑月、日)。设一个班有 n 个人，当 n 大于 365 时 {至少两个人有生日相同} 是必然事件 (概率等于 1)。

当 n 小于等于 365 时：

$$
\begin{aligned}
&P(\text{至少有两人同生日})\\
=&1-P(n\text{ 个人生日彼此不同})\\
=&1-\frac{365\times 364\times\cdots\times(365-(n-1))}{365^n}\\
=&1-\frac{365-0}{365}\cdot\frac{365-1}{365}\cdots\frac{365-(n-1)}{365}
\end{aligned}
$$

对 $n=1,2,\ldots,365$ 来计算对应的概率。

完全用循环（两重循环），程序写成：

```
birth_f1 <- function(){
  ny <- 365
  x <- numeric(ny)
  for(n in 1:ny){
    s <- 1
    for(j in 0:(n-1)){
```

```
      s <- s * (365-j)/365
    }
    x[n] <- 1 - s
  }
  x
}
```

注意，不能先计算 $365 \times 364 \times \cdots \times (365-(n-1))$ 再和 365^n 相除，这会造成数值溢出。

用 `prod()` 函数可以向量化内层循环：

```
birth_f2 <- function(){
  ny <- 365
  x <- numeric(ny)
  for(n in 1:ny){
    x[n] <- 1 - prod((365:(365-n+1))/365)
  }
  x
}
```

程序利用了向量与标量的除法，以及内建函数 `prod()`。

把程序用 `cumprod()` 函数改写，可以完全避免循环：

```
birth_f3 <- function(){
  ny <- 365
  x <- 1 - cumprod((ny:1)/ny)
  x
}
```

用 bench 包比较:

```
bench::mark(
  birth_f1(),
  birth_f2(),
  birth_f3()
)
## Unit: microseconds
##  expr      min        lq       mean    median       uq      max neval cld
##  f1() 2467.000 2792.1515 2963.27806 2979.9515 3033.202 8058.001   100   c
##  f2()  330.100  387.1005  674.42992  413.3505  438.301 8445.601   100  b
##  f3()    1.901    2.3515   27.51003    2.8510    3.651 2322.500   100 a
```

`birth_f2()` 比 `birth_f1()` 快大约 6 倍，`birth_f3()` 比 `birth_f2()` 又快了大约 140 倍倍，`birth_f3()` 比 `birth_f1()` 快了 1000 倍！

18.3 减少显式循环

显式循环是 R 运行速度较慢的部分，有循环的程序也比较冗长，与 R 的向量化简洁风格不太匹配。另外，在循环内修改数据子集，例如数据框子集，可能会先制作副本再修改，这当然会损失很多效率。R 3.1.0 版本以后列表元素在修改时不制作副本，但数据框还会制作副本。

前面已经指出，利用 R 的向量化运算可以减少很多循环程序。

R 中的有些运算可以用内建函数完成，如 `sum`, `prod`, `cumsum`, `cumprod`, `mean`, `var`, `sd` 等。这些函数以编译程序的速度运行，不存在效率损失。

R 的 `sin`, `sqrt`, `log` 等函数都是向量化的，可以直接对输入向量的每个元素进行变换。

对矩阵，用 `apply` 函数汇总矩阵每行或每列。`colMeans`, `rowMeans` 可以计算矩阵列平均和行平均，`colSums`, `rowSums` 可以计算矩阵列和与行和。

apply 类函数有多个，包括 apply, sapply, lapply, tapply, vapply, replicate 等。这些函数不一定能提高程序运行速度，但是使用这些函数更符合 R 的程序设计风格，使程序变得简洁，当然，程序更简洁并不等同于程序更容易理解，要理解这样的程序，需要更多学习与实践。参见25。

18.3.1 `replicate()` 函数

`replicate()` 函数用来执行某段程序若干次，类似于 `for()` 循环但是没有计数变量。常用于随机模拟。`replicate()` 的缺省设置会把重复结果尽可能整齐排列成一个多维数组输出。

语法为

> `replicate`(重复次数, 要重复的表达式)

其中的表达式可以是复合语句, 也可以是执行一次模拟的函数。

下面举一个简单模拟例子。设总体 X 为 $\mathrm{N}(0,1)$, 取样本量 $n=5$，重复地生成模拟样本共 $B=6$ 组，输出每组样本的样本均值和样本标准差。模拟可以用如下的 `replicate()` 实现：

```
set.seed(1)
replicate(6, {
  x <- rnorm(5, 0, 1);
  c(mean(x), sd(x)) })
```

```
##           [,1]      [,2]       [,3]      [,4]       [,5]       [,6]
## [1,] 0.1292699 0.1351357 0.03812297 0.4595670 0.08123054 -0.3485770
## [2,] 0.9610394 0.6688342 1.49887443 0.4648177 1.20109623  0.7046822
```

结果是一个矩阵，矩阵行数与每次模拟的结果（均值、标准差）个数相同，这里第一行都是均值，第二行都是标准差；矩阵每一列对应于一次模拟。这样结果存储应该是来源于 R 的数组元素次序为列优先次序。

18.4 避免制作副本

类似于 `x <- c(x, y)` 这样的累积结果每次运行都会制作一个 `x` 的副本，在 `x` 存储量较大或者重复修改次数很多时会减慢程序。

例 18.4. 执行 10000 次模拟，每次模拟求 10 个 U(0,1) 均匀随机数的极差，保存每次的极差，求平均极差。

用 `x <- c(x, y)` 这样的写法：

```
set.seed(101)
system.time({
  M <- 1E5
  x <- c()
  for(i in seq(M)){
    x <- c(x, diff(range(runif(10))))
  }
  mean(x)
})
## 用户 系统 流逝
## 22.81  2.58 25.49
```

上面的程序不仅是低效率的做法，也没有预先精心设计用来保存结果的数据结构。数据建模或随机模拟的程序都应该事先分配好用来保存结果的数据结构，在每次循环中填入相应结果。如：

```
set.seed(101)
system.time({
  M <- 1E5
  x <- numeric(M)
  for(i in seq(M)){
    x[[i]] <- diff(range(runif(10)))
  }
  mean(x)
})
## 用户 系统 流逝
## 0.96 0.01 0.96
```

这样的程序结构更清晰，效率更高，而且循环次数越多，比 `x <- c(x, ...)` 这样的做法的优势越大。

例 18.5. 在循环内修改数据框的值也会制作数据框副本，当数据框很大或者循环次数很多时会使得程序很慢。下面给出示例。

```
set.seed(101)
m <- 2E4; n <- 100
x <- as.data.frame(matrix(
  runif(n*m), nrow=n, ncol=m))
system.time({
  for(j in seq(m)){
    x[[j]] <- x[[j]] + 1
  }
})
## 用户 系统 流逝
## 11.53  0.03 11.55
```

在循环内修改列表元素就不会制作副本，从而可以避免这样的效率问题，如：

```
set.seed(101)
m <- 2E4; n <- 100
x <- replicate(m,
  runif(n),
  simplify=FALSE)
system.time({
  for(j in seq(m)){
    x[[j]] <- x[[j]] + 1
  }
})
## 用户 系统 流逝
## 0.02 0.02 0.04
x <- as.data.frame(x)
```

`replicate()` 函数中用 `simplify=FALSE` 使结果总是返回列表。要注意的是，上面第二个程序中的 `as.data.frame(x)` 也是效率较差的。将数据保存在列表中比保存在数据框中访问效率高，但数据框提供的功能更丰富。

18.5 用性能分析工具找到程序效率的瓶颈

R 的性能分析工具可以按较高频率（如每隔几毫秒）抽查正在被调用的函数，因为函数可以嵌套调用，所以会记录下来正在嵌套调用的各个函数。因为抽查的速度很快，所以会得到大量的被调用函数的抽样数据，这样就可以进行概括，得知哪些函数被调用最频繁，调用的途径是怎样的。基本 R 的 `utils::Rprof()` 可以收集程序运行的性能分析用数据，而 `utils::summaryRprof()` 则可以用文本格式提供性能分析的概括。

在 RStudio 软件中，借助于 profvis 扩展包，我们可以从 Profile 菜单对选定行进行性能分析。但是，对比较复杂的程序，应该将要分析的程序保存为一个 R 源文件，并用 `source()` 的方法将要运行的函数载入，并用 `profvis()` 函数调用该函数并显示运行后性能分析结果。

例 18.6. 以18.4的程序为例。我们将第一个程序包装为一个函数 `bad_copy()` 并存入文件 `bad_copy.R` 中，然后用 profvis 进行性能分析。

`bad_copy.R` 文件内容为：

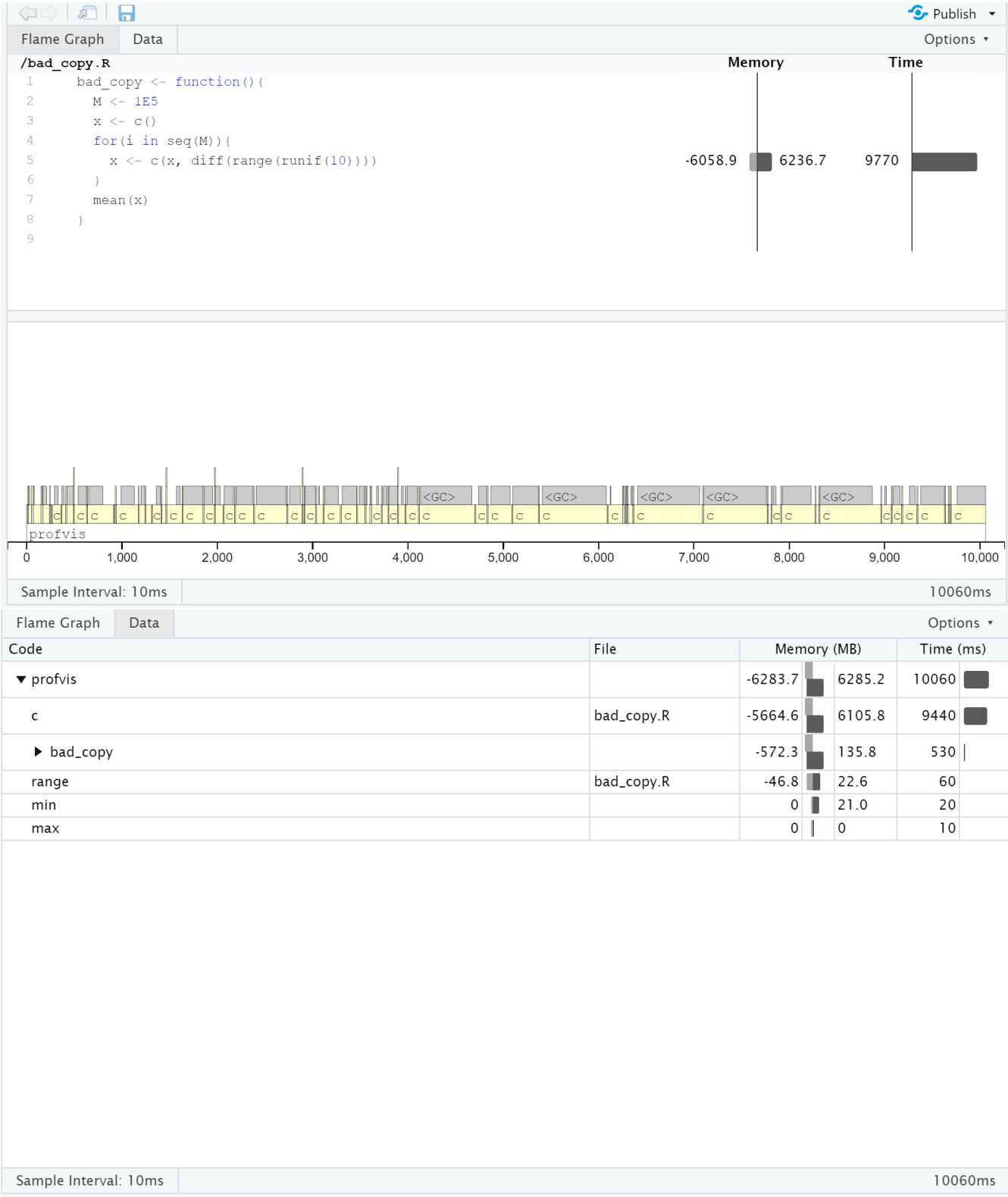
```
bad_copy <- function(){
  M <- 1E5
  x <- c()
  for(i in seq(M)){
    x <- c(x, diff(range(runif(10))))
  }
  mean(x)
}
```

安装好 profvis 扩展包后，运行如下的性能分析程序：

```
library(profvis)
profvis(bad_copy())
```

结果在 RStudio 中将显示如下的两个窗格内容：

p-prof-profiler-bc01.png



在 Flame Graph 窗格中，下方显示了函数调用所占用的时间，当鼠标光标移动到下方的某一个函数调用的色块时可以显示该函数的调用路径和耗时等信息。可以看出是 `c()` 函数和 `GC()` 函数占用了绝大多数时间，而 `GC()` 是被 `c()` 调用的。反复调用内存垃圾回收函数 `GC()` 一般说明程序有反复分配内存的问题，从该图上半部分的 Memery

图表和 Time 图表也可以看出这一点，Memery 图表中左侧为释放内存量，右侧为分配内存量；Time 图表看出程序运行集中在 `x` 被重复赋值的那一行。

在 Data 窗格中，点击 Profvis 后才展开详细的汇总统计图表。可以看出耗时最多的是 `c()` 函数调用，但要注意的是，这里的耗时也包括 `c()` 函数内部的 `diff()`, `range()`, `runif()` 函数的调用时间。

18.6 R 的计算函数

R 中提供了大量的数学函数、统计函数和特殊函数，可以打开 R 的 HTML 帮助页面，进入 "Search Enging & Keywords" 链接，查看其中与算术、数学、优化、线性代数等有关的专题。

这里简单列出一部分常用函数，并对数学计算中常用函数 `filter`, `fft`, `convolve` 进行较详细说明。

18.6.1 数学函数

常用数学函数包括 `abs`, `sign`, `log`, `log10`, `sqrt`, `exp`, `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `atan2`, `sinh`, `cosh`, `tanh`。还有 `gamma`, `lgamma`(伽玛函数的自然对数)。

用于取整的函数有 `ceiling`, `floor`, `round`, `trunc`, `signif`, `as.integer` 等。这些函数是向量化的一元函数。

`choose(n,k)` 返回从 n 中取 k 的组合数。`factorial(x)` 返回 $x!$ 结果。`combn(x,m)` 返回从集合 x 中每次取出不计次序的 m 个的所有不同取法，结果为一个矩阵，矩阵每列为一种取法的 m 个元素值。

18.6.2 概括函数

`sum` 对向量求和, `prod` 求乘积。

`cumsum` 和 `cumprod` 计算累计，得到和输入等长的向量结果。

`diff` 计算前后两项的差分（后一项减去前一项）。

`mean` 计算均值，`var` 计算样本方差或协方差矩阵，`sd` 计算样本标准差, `median` 计算中位数，`quantile` 计算样本分位数。`cor` 计算相关系数。

`colSums`, `colMeans`, `rowSums`, `rowMeans` 对矩阵的每列或每行计算总和或者平均值，效率比用 `apply` 函数要高。

`rle` 和 `inverse.rle` 用来计算数列中 "连" 长度及其逆向恢复，"连" 经常用在统计学的随机性检验中。

18.6.3 最值

`max` 和 `min` 求最大和最小，`cummax` 和 `cummin` 累进计算。

`range` 返回最小值和最大值两个元素。

对于 `max`, `min`, `range`，如果有多个自变量可以把这些自变量连接起来后计算。

pmax(x1,x2,...) 对若干个等长向量计算对应元素的最大值,不等长时短的被重复使用。pmin 类似。比如,pmax(0, pmin(1,x)) 把 x 限制到 $[0,1]$ 内。

18.6.4 排序

sort 返回排序结果。可以用 decreasing=TRUE 选项进行降序排序。sort 可以有一个 partial= 选项，这样只保证结果中 partial= 指定的下标位置是正确的。比如:

```
sort(c(3,1,4,2,5), partial=3)
```

```
## [1] 2 1 3 4 5
```

只保证结果的第三个元素正确。可以用来计算样本分位数估计。

在 sort() 中用选项 na.last 指定缺失值的处理，取 NA 则删去缺失值，取 TRUE 则把缺失值排在最后面，取 FALSE 则把缺失值排在最前面。

order 返回排序用的下标序列, 它可以有多个自变量，按这些自变量的字典序排序。可以用 decreasing=TRUE 选项进行降序排序。如果指定选项 method="radix", 则 decreasing 选项可以输入一个逻辑向量，对每个参与排序的自变量分别指定是否降序排列。

sort.list 函数与 order 功能相同但只支持按一个变量排序。

rank 计算秩统计量，可以用 ties.method 指定同名次处理方法，如 ties.method=min 取最小秩。

order, sort.list, rank 也可以有 na.last 选项，只能为 TRUE 或 FALSE。

unique() 返回去掉重复元素的结果，duplicated() 对每个元素用一个逻辑值表示是否与前面某个元素重复。如

```
unique(c(1,2,2,3,1))
```

```
## [1] 1 2 3
```

```
duplicated(c(1,2,2,3,1))
```

```
## [1] FALSE FALSE  TRUE FALSE  TRUE
```

rev 反转序列。

18.6.5 一元定积分 integrate

integrate(f, lower, upper) 对一元函数 f 计算从 lower 到 upper 的定积分。使用自适应算法保证精度。如:

```
integrate(sin, 0, pi)
## 2 with absolute error < 2.2e-14
```

函数的返回值不仅仅包含定积分数值，还包含精度等信息。

允许使用-Inf 和 Inf 作为积分边界，如：

```
integrate(dnorm, -Inf, Inf)
## 1 with absolute error < 9.4e-05
```

18.6.6 一元函数求根 uniroot

uniroot(f, interval) 对函数 f 在给定区间内求一个根，interval 为区间的两个端点。要求 f 在两个区间端点的值异号。即使有多个根也只能给出一个。如 $x(x-1)(x+1)$ 函数在 $[-2,2]$ 的根：

```
uniroot(\(x) x*(x-1)*(x+1), c(-2, 2))
```

```
## $root
## [1] 0
##
## $f.root
## [1] 0
##
## $iter
## [1] 1
##
## $init.it
## [1] NA
##
## $estim.prec
## [1] 2
```

对于多项式，可以用 polyroot 函数求出所有的复根，输入为升幂排列的多项式系数组成的向量。如多项式 x^3-x 和 x^2+1 的所有复根：

```
polyroot(c(0, -1, 0, 1))
## [1]  0+0i  1+0i -1+0i
polyroot(c(1, 0, 1))
## [1] 0+1i 0-1i
```

18.6.7 离散傅立叶变换 fft

R 中 fft 函数使用快速傅立叶变换算法计算离散傅立叶变换。设 x 为长度 n 的向量，y=fft(x)，则

```
y[k] = sum(x * complex(
    argument = -2*pi * (0:(n-1)) * (k-1)/n))
```

即

$$y_{k+1} = \sum_{j=0}^{n-1} x_{j+1} \exp\left(-i2\pi \frac{kj}{n}\right),\ k = 0, 1, \dots, n-1.$$

注意没有除以 n，结果是复数向量。

另外，若 y=fft(x), z=fft(y, inverse=TRUE), 则 x == z/length(x)。

快速傅立叶变换是数值计算中十分常用的工具，R 软件包 fftw 可以进行优化的快速傅立叶变换。

18.6.8 用 filter 函数作迭代

R 在遇到向量自身迭代时很难用向量化编程解决，filter 函数可以解决其中部分问题。filter 函数可以进行卷积型或自回归型的迭代。语法为

```
filter(x, filter,
       method = c("convolution", "recursive"),
       sides=2, circular =FALSE, init)
```

下面用例子演示此函数的用途。

18.6.8.1 示例 1：双侧滤波

对输入序列 $x_t, t = 1, 2, \dots, n$，希望进行如下滤波计算:

$$y_t = \sum_{j=-k}^{k} a_j x_{t-j},\ k+1 \le t \le n-k-1,$$

其中 $(a_{-k}, \dots, a_0, \dots, a_k)$ 是长度为 $2k+1$ 的向量。注意公式中 a_j 与 x_{t-j} 对应。

假设 x 保存在向量 x 中，$(a_{-k}, \dots, a_0, \dots, a_k)$ 保存在向量 f 中，$y_{k+1}, \dots, y_{n-k}$ 保存在向量 y 中, 无定义部分取 NA, 程序可以写成

```
y <- filter(x, f, method="convolution", sides=2)
```

比如，设 $x = (1, 3, 7, 12, 17, 23)$, $(a_{-1}, a_0, a_1) = (0.1, 0.5, 0.4)$, 则

$$y_t = 0.1 \times x_{t+1} + 0.5 \times x_t + 0.4 \times x_{t-1},\ t = 2, 3, \dots, 5$$

用 filter() 函数计算，程序为:

```
y <- filter(c(1,3,7,12,17,23), c(0.1, 0.5, 0.4),
            method="convolution", sides=2)
y
## Time Series:
## Start = 1
## End = 6
## Frequency = 1
## [1]   NA  2.6  5.9 10.5 15.6   NA
```

filter 的结果为一个时间序列对象，可以用 as.vector() 函数转换为普通数值型向量。

18.6.8.2 示例 2: 单侧滤波

对输入序列 $x_t, t = 1, 2, \dots, n$，希望进行如下滤波计算:

$$y_t = \sum_{j=0}^{k} a_j x_{t-j},\ k+1 \le t \le n,$$

其中 $(a_0, \dots, a_k)$ 是长度为 $k+1$ 的向量。注意公式中 a_j 与 x_{t-j} 对应。

假设 x 保存在向量 x 中，$(a_0, \dots, a_k)$ 保存在向量 f 中，$y_{k+1}, \dots, y_n$ 保存在向量 y 中, 无定义部分取 NA, 程序可以写成

```
y <- filter(x, f, method="convolution", sides=1)
```

比如，设 $x = (1, 3, 7, 12, 17, 23)$, $(a_0, a_1, a_2) = (0.1, 0.5, 0.4)$, 则

$$y_t = 0.1 \times x_t + 0.5 \times x_{t-1} + 0.4 \times x_{t-2},\ t = 3, 4, \dots, 6$$

程序为

```
y <- filter(c(1,3,7,12,17,23), c(0.1, 0.5, 0.4),
            method="convolution", sides=1)
y
## Time Series:
## Start = 1
## End = 6
## Frequency = 1
## [1]   NA   NA  2.6  5.9 10.5 15.6
```

18.6.8.3 示例 3: 自回归迭代

设输入 $e_t, t = 1, 2, \dots, n$，要计算

$$y_t = \sum_{j=1}^{k} a_j y_{t-j} + e_t,\ t = 1, 2, \dots, n,$$

其中 $(a_1, \dots, a_k)$ 是 k 个实数，$(y_{-k+1}, \dots, y_0)$ 已知。

设 x 保存在向量 `x` 中，$(a_1, \dots, a_k)$ 保存在向量 `a` 中，$(y_1, \dots, y_n)$ 保存在向量 y 中。

如果 $(y_{-k+1}, \dots, y_0)$ 都等于零，可以用如下程序计算 $y_1, y_2, \dots, y_n$:

```
filter(x, a, method="recursive")
```

如果 $(y_0, \dots, y_{-k+1})$ 保存在向量 b 中 (注意与时间顺序相反)，可以用如下程序计算 $y_1, y_2, \dots, y_n$:

```
filter(x, a, method="recursive", init=b)
```

比如，设 $e = (0.1, -0.2, -0.1, 0.2, 0.3, -0.2)$, $(a_1, a_2) = (0.9, 0.1)$, $y_{-1} = y_0 = 0$, 则

$$y_t = 0.9 \times y_{t-1} + 0.1 \times y_{t-2} + e_t,\ t = 1, 2, \dots, 6$$

迭代程序和结果为

```
y <- filter(c(0.1, -0.2, -0.1, 0.2, 0.3, -0.2),
            c(0.9, 0.1), method="recursive")
print(y, digits=3)
## Time Series:
## Start = 1
## End = 6
## Frequency = 1
## [1]  0.1000 -0.1100 -0.1890  0.0189  0.2981  0.0702
```

这个例子中，如果已知 $y_0 = 200, y_{-1} = 100$, 迭代程序和结果为：

```
y <- filter(c(0.1, -0.2, -0.1, 0.2, 0.3, -0.2),
            c(0.9, 0.1), init=c(200, 100),
            method="recursive")
print(y, digits=6)
## Time Series:
## Start = 1
## End = 6
## Frequency = 1
## [1] 190.100 190.890 190.711 190.929 191.207 190.979
```

18.7 并行计算

现代桌面电脑和笔记本电脑的 CPU 通常有多个核心或虚拟核心（线程），如 2 核心或 4 虚拟核心。通常 R 运行并不能利用全部的 CPU 能力，仅能利用其中的一个虚拟核心。使用特制的 BLAS 库（非 R 原有）可以并发运行多个线程，一些 R 扩展包也可以利用多个线程。利用多台计算机、多个 CPU、CPU 中的多核心和多线程同时完成一个计算任务称为并行计算。

想要充分利用多个电脑、多个 CPU 和 CPU 内的虚拟核心，技术上比较复杂，涉及到计算机之间与进程之间的通讯问题，在要交流的数据量比较大时会造成并行计算的瓶颈。

实际上，有些问题可以很容易地进行简单地并行计算。比如，在一个统计研究中，需要对 100 组参数组合进行模拟，评估不同参数组合下模型的性能。假设研究人员有两台安装了 R 软件的计算机，就可以在两台计算机上进行各自 50 组参数组合的模拟，最后汇总在一起就可以了。

R 的 parallel 包提供了一种比较简单的利用 CPU 多核心的功能，思路与上面的例子类似，如果有多个任务互相之间没有互相依赖，就可以分解到多个计算机、多个 CPU、多个虚拟核心中并行计算。最简单的情形是一台具有单个 CPU、多个虚拟核心的台式电脑或者笔记本电脑。但是，统计计算中最常见耗时计算任务是随机模拟，随机模拟要设法避免不同进程的随机数序列的重复可能，以及同一进程中不同线程的随机数序列的重复可能。

parallel 包提供了 `parLapply()`、`parSapply()`、`parApply()` 函数，作为 `lapply()`、`sapply()`、`apply()` 函数的并行版本，与非并行版本相比，需要用一个临时集群对象作为第一自变量。

并行计算的工具在不断进步，这部分内容的做法具有一定的参考价值，但所用工具不一定是最新的，欢迎读者批评指正。

18.7.1 例 1：完全不互相依赖的并行运算

考虑如下计算问题:

$$S_{k,n} = \sum_{i=1}^{n} \frac{1}{i^k}$$

下面的程序取 `n` 为一百万，`k` 为 2 到 21，循环地用单线程计算。

```
f10 <- function(k=2, n=1000){
  s <- 0.0
  for(i in seq(n)) s <- s + 1/i^k
  s
}
f11 <- function(n=1000000){
  nk <- 20
  v <- sapply(2:(nk+1), function(k) f10(k, n))
  v
}
system.time(f11())[3]
```

```
## elapsed
##    1.95
```

因为对不同的 k，f0(k) 计算互相不依赖，也不涉及到随机数序列，所以可以简单地并行计算而没有任何风险。先查看本计算机的虚拟核心（线程）数：

```
library(parallel)
detectCores()
## [1] 8
```

用 makeCluster() 建立临时的有 8 个节点的单机集群：

```
nNodes <- 8
cpucl <- makeCluster(nNodes)
```

用 parSapply() 或者 parLapply() 关于 k 并行地循环：

```
f12 <- function(n=1000000){
  f10 <- function(k=2, n=1000){
    s <- 0.0
    for(i in seq(n)) s <- s + 1/i^k
    s
  }

  nk <- 20
  v <- parSapply(cpucl, 2:(nk+1), function(k) f10(k, n))
  v
}
system.time(f12())[3]
## elapsed
##    1.2
```

并行版本速度提高了 60% 左右。

并行执行结束后，需要解散临时的集群，否则可能会有内存泄漏：

```
stopCluster(cpucl)
```

注意并行版本的程序还需要一些在每个计算节点上的初始化，比如调入扩展包，定义函数，初始化不同的随机数序列等。parallel 包的并行执行用的是不同的进程，所以传送给每个节点的计算函数要包括所有的依赖内容。比如，f12() 中内嵌了 f10() 的定义，如果不将 f10() 定义在 f12() 内部，就需要预先将 f10() 的定义传递给每个节点。

parallel 包的 `clusterExport()` 函数可以用来把计算所依赖的对象预先传送到每个节点。比如，上面的 `f12()` 可以不包含 `f10()` 的定义，而是用 `clusterExport()` 预先传递：

```
cpucl <- makeCluster(nNodes)
clusterExport(cpucl, c("f10"))
f13 <- function(n=1000000){
  nk <- 20
  v <- parSapply(cpucl, 2:(nk+1), function(k) f10(k, n))
  v
}
system.time(f13())[3]
## elapsed
##   1.16
stopCluster(cpucl)
```

如果需要在每个节点预先执行一些语句，可以用 `clusterEvalQ()` 函数执行，如

```
clusterEvalQ(cpucl, library(dplyr))
```

18.7.2 例 2：使用相同随机数序列的并行计算

为了估计总体中某个比例 p 的置信区间，调查了一组样本，在 n 个受访者中选“是”的比例为 $\hat{p}$。令 λ 为标准正态分布的双侧 α 分位数，参数 p 的近似 $1-\alpha$ 置信区间为

$$\frac{\hat{p}+\frac{\lambda^2}{2n}}{1+\frac{\lambda^2}{n}} \pm \frac{\lambda}{\sqrt{n}} \frac{\sqrt{\hat{p}(1-\hat{p})+\frac{\lambda^2}{4n}}}{1+\frac{\lambda^2}{n}}$$

称为 Wilson 置信区间。

假设要估计不同 $1-\alpha, n, p$ 情况下，置信区间的覆盖率（即置信区间包含真实参数 p 的概率）。可以将这些参数组合定义成一个列表，列表中每一项是一种参数组合，对每一组合分别进行随机模拟，估计覆盖率。因为不同参数组合之间没有互相依赖的关系，随机数序列完全可以使用同一个序列。

不并行计算的程序示例：

```
wilson <- function(n, x, conf){
  hatp <- x/n
  lam <- qnorm((conf+1)/2)
  lam2 <- lam^2 / n
  p1 <- (hatp + lam2/2)/(1 + lam2)
  delta <- lam / sqrt(n) * sqrt(hatp*(1-hatp) + lam2/4) / (1 + lam2)
  c(p1-delta, p1+delta)
```

```
}
f20 <- function(cpar){
  set.seed(101)
  conf <- cpar[1]
  n <- cpar[2]
  p0 <- cpar[3]
  nsim <- 100000
  cover <- 0
  for(i in seq(nsim)){
    x <- rbinom(1, n, p0)
    cf <- wilson(n, x, conf)
    if(p0 >= cf[1] && p0 <= cf[2]) cover <- cover+1
  }
  cover/nsim
}
f21 <- function(){
  lp <- expand.grid(c(0.8, 0.9), c(30, 100), c(0.5, 0.1)) |>
    as.matrix() |>
    t() |>
    as.data.frame() |>
    as.list()
  res <- sapply(lp, f20)
  res
}
system.time(f21())[3]
## elapsed
##   6.52
```

约运行 6.52 秒。

改为并行版本：

```
library(parallel)
nNodes <- 8
cpucl <- makeCluster(nNodes)
clusterExport(cpucl, c("f20", "wilson"))
f22 <- function(){
  lp <- expand.grid(c(0.8, 0.9), c(30, 100), c(0.5, 0.1)) |>
    as.matrix() |>
    t() |>
```

```
    as.data.frame() |>
    as.list()
  res <- parSapply(cpucl, lp, f20)
  res
}
system.time(f22())[3]
## elapsed
##   2.83
stopCluster(cpucl)
```

运行约 2.83 秒，速度提高一倍以上。这里模拟了 8 种参数组合，每种参数组合模拟了十万次，每种参数组合模拟所用的随机数序列是相同的。

18.7.3 例 3：使用独立随机数序列的并行计算

大量的耗时的统计计算是随机模拟，有时需要并行计算的部分必须使用独立的随机数序列。比如，需要进行一千万次重复模拟，每次使用不同的随机数序列，可以将其分解为 10 组模拟，每组模拟一百万次，这就要求这 10 组模拟使用的随机数序列不重复。

R 中实现了 `L'Ecuyer` 的多步迭代复合随机数发生器，此随机数发生器周期很长，而且很容易将发生器的状态前进指定的步数。parallel 包的 `nextRNGStream()` 函数可以将该发生器前进到下一段的开始，每一段都足够长，可以用于一个节点。

以 Wilson 置信区间的模拟为例。设 $n = 30$, $p = 0.01$, $1 - \alpha = 0.95$，取重复模拟次数为 1 千万次，估计 Wilson 置信区间的覆盖率。单线程版本为：

```
f31 <- function(nsim=1E7){
  set.seed(101)
  n <- 30; p0 <- 0.01; conf <- 0.95
  cover <- 0
  for(i in seq(nsim)){
    x <- rbinom(1, n, p0)
    cf <- wilson(n, x, conf)
    if(p0 >= cf[1] && p0 <= cf[2]) cover <- cover+1
  }
  cover/nsim
}
system.time(cvg1 <- f31())[3]
## elapsed
##  78.11
```

单线程版本运行了大约 78 秒。

改成并行版本。比例 2 多出的部分是为每个节点分别计算一个随机数种子将不同的种子传给不同节点。parallel 包的 `clusterApply()` 函数为临时集群的每个节点分别执行同一函数，但对每个节点分别使用列表的不同元素作为函数的自变量。

```
library(parallel)
nNodes <- 8
cpucl <- makeCluster(nNodes)
each.seed <- function(s){
  assign(".Random.seed", s, envir = .GlobalEnv)
}
RNGkind("L'Ecuyer-CMRG")
set.seed(101)
seed0 <- .Random.seed
seeds <- as.list(1:nNodes)
for(i in 1:nNodes){ # 给每个节点制作不同的种子
  seed0 <- nextRNGStream(seed0)
  seeds[[i]] <- seed0
}
## 给每个节点传送不同种子:
junk <- clusterApply(cpucl, seeds, each.seed)

f32 <- function(isim, nsimsub=10000){
  n <- 30; p0 <- 0.01; conf <- 0.95
  cover <- 0
  for(i in seq(nsimsub)){
    x <- rbinom(1, n, p0)
    cf <- wilson(n, x, conf)
    if(p0 >= cf[1] && p0 <= cf[2]) cover <- cover+1
  }
  cover
}
clusterExport(cpucl, c("f32", "wilson"))

f33 <- function(nsim=1E7){
  nbatch <- 40
  nsimsub <- nsim / nbatch
  cvs <- parSapply(cpucl, 1:nbatch, f32, nsimsub=nsimsub)
  sum(cvs)/(nsim*nbatch)
}
```

```
system.time(cvg2 <- f33())[3]
## elapsed
##  30.72
stopCluster(cpucl)
```

并行版本运行了大约 41 秒，速度提高一倍以上。从两个版本各自一千万次重复模拟结果来看，用随机模拟方法得到的覆盖率估计的精度大约为 3 位有效数字。

更大规模的随机模拟问题，可以考虑使用多 CPU 的计算工作站或者服务器，或用多台计算机通过高速局域网组成并行计算集群。

还有一种选择是租用云计算服务。

Chapter 19

函数进阶

19.1 函数调用的各种形式

在 R 语言中，有两条简明的理解 R 程序的原则：

- 任何成分都是 R 的对象（变量、函数等等）；
- 任何活动都是调用函数（求子集、四则运算、比较、函数调用等）。

函数调用有四种方式：

一、**前缀形式**。这也是一般的格式，如 `fsub(5, 2)`。

二、**中缀形式**。二元运算符实际上都是函数，`5 - 2` 的写法是中缀形式，等同于`` `-`(5, 2) ``。因为-不是合法的 R 变量名（函数名），所以在写成前缀形式时要用反向单撇号`保护。这样，在 `lapply` 等泛函中可以使用`` `+` ``这样的四则运算作为输入的操作。如

```
5 - 2
```

```
## [1] 3
```

```
`-`(5, 2)
```

```
## [1] 3
```

为了给 `1:5` 每个元素减去 2，可以写成

```
sapply(1:5, `-`, 2)
```

```
## [1] -1  0  1  2  3
```

用户也可以自己定义函数名如%x% 这样的中缀函数，可以用中缀格式调用。如：

```
`%+%` <- function(x, y) paste0(x, y)
"xyz" %+% "123"
```

```
## [1] "xyz123"
```

三、**替换形式**。对属性的修改经常这样写，如

```
x <- 1:2
names(x) <- c("a", "b")
x
```

```
## a b
## 1 2
```

看起来是在对一个函数的输出结果赋值，这很不合理，但实际相当于前缀形式的

```
x <- 1:2
`*tmp*` <- x
x <- `names<-`(x, c("a", "b"))
rm(`*tmp*`)
x
```

即制作了 x 的副本，调用 names<-函数，将 x 重新绑定到 names<-函数的返回值。

四、**特殊形式**。x[1], x[[1]] 这些取子集或元素以及修改，(), {}，if 结构、for 循环等本质上也是函数调用，只不过用了特殊的语法。这些函数在 R 中都是**初等函数** (primitive functions)。初等函数仅在基本 R 中定义，是由 C 代码实现的，用户无法访问其三个部分。

取子集的特殊函数例如：

```
x <- 1:5
x[1]
```

```
## [1] 1
```

```
`[`(x, 1)
```

```
## [1] 1
```

```
x[1] <- 999
x
```

```
## [1] 999   2   3   4   5
```

```
x <- 1:5
x <- `[<-`(x, 1, 999)
x
```

```
## [1] 999   2   3   4   5
```

注意上面的 `x[1] <- 999` 的替代写法中，调用`` `[<-`(x, 1, 999) `` 是将其返回值（一个向量对象）重新绑定给变量 `x`，才达到了修改 `x` 的目的。

`for` 循环也是函数调用，如 `for(i in 1:3) print(i)` 可以等价地写成`` `for`(i, 1:3, print(i)) ``。

19.2 嵌套定义与句法作用域 (lexical scoping)

R 语言允许在函数体内定义函数。比如，

```
x <- -1
f0 <- function(x){
  f1 <- function(){
    x + 100
  }
  f1()
}
```

其中内嵌的函数 `f1()` 称为一个 closure(闭包)。

内嵌的函数体内在读取某个变量值时，如果此变量在函数体内还没有被赋值，它就不是局部的，会向定义的外面一层查找，外层一层找不到，就继续向外查找。上面例子 `f1()` 定义中的变量 `x` 不是局部变量，就向外一层查找，找到的会是 `f0` 的自变量 `x`，而不是全局空间中 `x`。如

```
f0(1)
```

```
## [1] 101
```

最后 `x+100` 中 `x` 取的是 `f0` 的实参值 `x=1`，而不是全局变量 `x=-1`。

这样的变量查找规则叫做**句法作用域** (lexical scoping)，即函数运行中需要使用某个变量时，从其**定义时**的环境向外层逐层查找，而**不是**在调用时的环境中查找。

例如，

```
f0 <- function(){
  f1 <- function(){
    x <- -1
    f2 <- function(){
      x + 100
    }
    f2()
  }
  x <- 1000
  f1()
}
f0()
```

```
## [1] 99
```

其中 `f2()` 运行时，用到的 `x` 是 `f1()` 函数体内的局部变量 `x=-1`，而不是被调用时 `f0()` 函数体内的局部变量 `x=1000`，所以结果是`-1 + 100 = 99`。

“句法作用域” 指的是函数调用时查找变量是查找其定义时的变量对应的**存储空间**，而**不是**定义时变量所取的历史值。函数运行时在找到某个变量对应的存储空间后，会使用该变量的**当前值**，而不是函数定义的时候该变量的历史值。这种规则称为动态查找 (dynamic lookup)。例如

```
f0 <- function(){
  x <- -1
  f1 <- function(){
    x + 100
  }
  x <- 1000
  f1()
}
f0()
```

```
## [1] 1100
```

结果为什么不是`-1 + 100 = 99` 而是 `1000 + 100 = 1100`? 这是因为，`f1()` 在调用时，使用的 `x` 是 `f0` 函数体内局部变量 `x` 的值，但是要注意的是程序运行时会访问该变量的**当前值**，即 1000，而不是函数定义的时候 `x` 的历史值-1。句法作用域与动态查找一个说的是如何查找某个变量对应的存储空间，一个说的是使用该存储空间何时的存储值，程序运行时两个规则需要联合使用。

句法作用域不仅适用于查找变量，也适用于函数体内调用别的函数时查找函数。查找函数的规则与查找变量规则相同。

19.3 辅助嵌套函数

有时内嵌函数仅仅是函数内用来实现模块化的一种工具，和正常的函数作用相同，没有任何特殊作用。例如，如下的程序在自变量 x 中输入一元二次方程 $ax^2 + bx + c = 0$ 的三个系数, 输出解：

```
solve.sqe <- function(x){
  fd <- function(a, b, c) b^2 - 4*a*c
  d <- fd(x[1], x[2], x[3])
  if(d >= 0){
    return( (-x[2] + c(1,-1)*sqrt(d))/(2*x[1]) )
  } else {
    return( complex(real=-x[2], imag=c(1,-1)*sqrt(-d))/(2*x[1]) )
  }
}
```

在这个函数中内嵌的函数 fd 仅起到一个计算二次判别式的作用，没有用到任何的闭包特性，其中的形参变量 a, b, c 都是局部变量。运行如

```
solve.sqe(c(1, -2, 1))
```

```
## [1] 1 1
```

```
solve.sqe(c(1, -2, 0))
```

```
## [1] 2 0
```

```
solve.sqe(c(1, -2, 2))
```

```
## [1] 1+1i 1-1i
```

这样的内嵌函数与直接在全局空间中定义的函数区别不大，只有一条区别：只能在定义它的函数内运行，不能被直接调用，可以看成是函数内的私有函数，可以避免名字冲突。

19.4 懒惰求值

R 函数在调用执行时，除非用到某个形式变量的值才求出其对应实参的值。这一点在实参是常数时无所谓，但是如果实参是表达式就不一样了。形参缺省值也是只有在函数运行时用到该形参的值时才求值。

例如，

```
f <- function(x, y=ifelse(x>0, TRUE, FALSE)){
  x <- -111
  if(y) x*2 else x*10
}
f(5)
```

```
## [1] -1110
```

可以看出，虽然形参 x 输入的实参值为 5, 但是这时形参 y 并没按 x=5 被赋值为 TRUE, 而是到函数体中第二个语句才被求值，这时 x 的值已经变成了-111，故 y 的值是 FALSE。

另外要注意的是，懒惰求值使得缺省值在初次访问时，是在函数内的环境（局部变量作用域）内求值的，不是在其调用处的环境内求值。

在函数内部，用 missing(x) 对形参 x 判断用户是否没有提供对应的实参，对位置形参和有缺省值的形参都适用。

19.5 程序调试

19.5.1 基本调试策略

自己编写的某些涉及到复杂的算法的程序可能一开始并不能给出期望的结果。这包括如下的情况：

- 程序报错，需要找到出错的地方加以纠正；
- 程序正常运行，输出了结果，但是结果明显有错；
- 最糟糕的是，程序结果也看起来很正常，但实际结果是错误的。

以上这三种情况是依次越来越糟糕的。直接运行出错的情况一般是比较容易处理的。

为了尽可能保证程序结果正确，在自己编写新算法时，要运用模块化思想，将问题分解为若干个小问题，使得每一个小问题都比较容易验证结果是否正确，将每一个小问题编写成一个单独的函数，这样也可以避免一段很长的程序中许多变量混杂在一起。

在划分模块并编写好程序后，应该编写一系列的测试函数，对每个函数进行测试，保证其在各种情况下的结果是正确的。最好采纳 R 的规则化的测试策略进行自动化测试，在编写 R 扩展包时就推荐同时提供测试代码。

如果程序还是有错误，首先可以求助于搜索引擎、用户社区等。如果这个问题是常见问题，往往这样就可以解决问题。

如果问题没有解决，需要将问题最小化：减少引起错误的程序的复杂程度，将不必要的代码尽可能用固定的输入数据代替，使得出错程序很短，而且错误可重复。有时会有不可重复的错误，这样的错误往往很难解决，超出了一般 R 用户的能力。

在将问题程序简化并且错误可重复以后，就要试图定位错误。一般编程语言都有如下的一些一般性查错 (debugging) 方法：

- 在程序中适当位置加上输出命令（语句），输出怀疑有错的变量值。
- 某些变成语言提供了跟踪以及条件跟踪命令，可以在程序运行到某个语句或者触发了某个条件时程序中止，但允许用户控制逐行运行程序并随时查看变量值，称为跟踪调试 (tracing)。跟踪调试可以是命令行工具，也可以是集成在 RStudio 这样的集成编程环境中的图形界面工具。

在查错时，科学研究思维照样起作用：根据错误表现提出可能的原因假设，制作测试输入数据验证假设，记录相应输出并分析是否与假设的错误原因吻合。如此反复直到找到出错原因并加以纠正。

查错并纠正就可能会破坏了成熟的代码，造成新的错误，所以最好能有自动化的测试策略，在每次修改程序后都重新测试程序是否保持输出正确。

19.5.2 找到出错的函数

在较复杂的程序出错时，需要首先将错误定位到某一函数调用。如：

```
f1 <- function(x) f2(x)
f2 <- function(x) 1/x
f1("abc")
## Error in 1/x : 二进列运算符中有非数值参数
```

为了在多层次函数调用中找到出错的函数，可以用如下命令：

```
traceback()
## 2: f2(x) at #1
## 1: f1("abc")
```

结果是所谓的反向追踪 (traceback)，一般编程语言中称为调用栈 (calling stack)。这个输出是从下向上堆叠显示，下层是调用方，上层是被调用方。

在 RStudio 中运行时，出错程序的右端可以显示 “Show Traceback” 以及 “Rerun with Debug” 快捷图标，点击 “Show Traceback” 图标也可以显示反向追踪结果。如果是一个源文件用 source 命名或图标运行时出错，在显示反向追踪结果时还可以显示调用的各命令所在的程序行号。点击 “Rerun with Debug” 可以进入跟踪调试状态，显示出错时的运行环境中的变量值。

19.5.3 跟踪调试

R 和 RStudio 提供了很好的跟踪运行程序的能力。R 的 `browser()` 命令可以用在程序中，命令进入跟踪调试；RStudio 的源文件显示界面可以用鼠标点击定义跟踪调试位置。

函数定义一般都包含多行，所以一般不在命令行定义函数，而是把函数定义以及较长的程序写在源程序文件中，用 `source` 命令运行。用 `source` 命令调入运行的程序与在命令行运行的效果基本相同，这样定义的变量也是全局变量。

考虑如下函数定义:

```
f <- function(x){
  for(i in 1:n){
    s <- s + x[i]
  }
}
```

这个函数定义有许多问题。用一个测试输入调用 f，发现有错误:

```
print(f(1:5))
## Error in 1:n : NA/NaN 参数
```

简单的函数可以直接仔细检查发现错误，用 cat, print 等输出中间结果查找错误。R 提供了一个 browser() 函数，在程序中插入对 browser() 函数的调用，可以进入跟踪调试状态，可以实时地查看甚至修改运行时变量的值。

在 RStudio 的编辑窗口中打开.R 源程序文件，在某一程序行行号左端的空白处用鼠标单击，就可以设定某一行为断点，在用 source 命令运行到该处时就可以进入跟踪调试状态。

加入 browser() 命令后的程序如：

```
f <- function(x){
  browser()
  for(i in 1:n){
    s <- s + x[i]
  }
}
```

程序运行遇到 browser() 函数或设定的断点时程序进入跟踪调试状态，命令行的提示符变成 “Browse[1]>”。这个命令行的环境一般不再是全局环境，而是断点所在的函数的运行环境，可以看到函数的局部变量。可以在调试环境中用命令去查看当前定义的变量值、逐个命令地运行，但是用 RStudio 则可以更方便地执行这些操作。

在调试命令行，可以使用如下命令：

- 输入变量名查看变量值；
- 用 n 命令或者换行键逐句运行；
- 用 s 命令跟踪进调用的函数内部逐句运行；
- 用 f 命令快速执行到循环末尾或者函数末尾；
- 用 c 命令恢复正常运行，不再跟踪；
- 用 Q 命令强行终止程序运行。

进入调试状态后，RStudio 界面提供了相应的支持。这时 RStudio 的命令行窗格 (Console) 将会显示用于控制运行的图标，包括执行下一语句（Next）、跟踪进入要调用的函数运行 (Step into)、执行到函数末尾或者循环末尾

(Finish)、不再跟踪继续正常运行（Continue）、终止运行（Stop）。同时，在 RStudio 的 Environment 窗格中会显示当前执行的命令所在的运行环境的内容，包括函数内的局部变量；如果点击其中的下拉菜单还可以显示函数的各层父环境。在同一窗格中还会显示向后追踪 (Traceback)，即函数是如何被调用的。

为调试如上函数 `f` 的程序，在定义中插入对 `browser()` 的调用如:

```
f <- function(x){
  browser()
  for(i in 1:n){
    s <- s + x[i]
  }
}
```

当在 RStudio 中调试运行时，程序编辑窗口将显示当前要运行的程序行，用命令行窗口 (Console) 的 Next 快捷图标可以运行到下一行。命令行的显示如：

```
> print(f(1:5))
Called from: eval(expr, p)
Browse[1]> n
debug在D:/disk/projects/Rbookweb/tmp2.R#2: for (i in 1:n) {
    s <- s + x[i]
}
Browse[2]>
```

继续用 “Next” 图标运行，命令行结果如：

```
Browse[2]> n
Error in 1:n : NA/NaN参数
```

发现是在 `for(i in 1:n)` 行遇到未定义的变量 `n`。

在源文件中把出错行改为 `for(i in 1:length(x))`，再次运行, 发现在运行 `s <- s + x[i]` 行时，遇到 “错误: 找不到对象’s’ ”。这是忘记初始化引起的。在 `for` 语句前添加 `s <- 0` 语句，函数定义变成：

```
f <- function(x){
  browser()
  s <- 0
  for(i in 1:length(x)){
    s <- s + x[i]
  }
}
```

再次运行，在跟踪到循环时，为了避免繁琐的跟踪过程，可以用“执行到函数末尾或者循环末尾”快捷图标或命令行的 `f` 命令，或者“Continue”快捷图标或命令行的 `c` 命令。程序不显示错误但是也没有显示结果为 `NULL` 而不是我们期望得输入元素之和。检查可以看出错误是忘记把函数返回值写在函数定义最后。

在函数定义最后添加 `s` 一行，再次运行，程序结果与手工验算结果一致。函数变成

```
f <- function(x){
  browser()
  n <- length(x)
  s <- 0
  for(i in 1:n){
    s <- s + x[i]
  }
  s
}
```

自定义函数应该用各种不同输入测试其正确性和稳定性。比如，上面的函数当自变量 x 为零长度向量时应该返回 0 才合适，但是上面的写法会返回一个 `numeric(0)` 结果，这个结果表示长度为零的向量：

```
f(numeric(0))
## Called from: f(numeric(0))
## Browse[1]> c
## numeric(0)
```

程序输入了零长度自变量，我们期望其输出为零而不是 `numeric(0)`。在自变量 x 为零长度时，函数中 `for(i in 1:length(x)` 应该一次都不进入循环，跟踪运行可以发现实际对 i=1 和 i=0 共运行了两轮循环。把这里的 `1:length(x)` 改成 `seq_along(x)` 解决了问题，`seq_along(x)` 生成 x 的下标序列，如果 x 是零长度的则下标序列为零长度向量。

函数不需要修改后，可以把对 `browser()` 的调用删除或注释掉，在 RStudio 中关闭断点。函数最终修改为：

```
f <- function(x){
  s <- 0
  for(i in seq_along(x)){
    s <- s + x[i]
  }
  s
}
```

这里只是用这个简单函数演示如何调试程序，求向量元素和这个问题本身是不需要我们去定义新函数的，`sum` 函数本来就是完成这样的功能。实际上，许多我们认为需要自己编写程序做的事情，在 R 网站都能找到别人已经完成的扩展包。

19.5.4 条件断点

用 `browser()` 函数与 R 的 `if` 结构配合可以制作条件断点。比如，在调试带有循环的程序时，发现错误发生在循环内，如果从循环开始就跟踪运行，会浪费许多时间。设已知错误发生在循环变量 `i` 等于 501 的时候，就可以在循环内插入：

```
if(i == 501) browser()
```

这样就可以在更接近出错的位置进入跟踪运行。

19.5.5 开启对一个函数的调试

可以用 `debug(f)` 命令对函数 `f` 开启跟踪运行，这时每次调用 `f()` 时都会自动进入跟踪运行状态。用 `undebug(f)` 取消对 `f` 的这种操作。

19.5.6 出错调试选项

比较长的程序在调试时如果从开头就跟踪，比较耗时。可以设置成出错后自动进入跟踪模式，检查出错时的变量值。只要进行如下设置：

```
options(error=recover)
```

则在出错后可以选择进入出错的某一层函数内部, 在 browser 环境中跟踪运行。

在 RStudio 中调试某一源程序文件时，可以选择 “Debug–On Error” 菜单，并选择 “Break in Code”，就可以在出错时自动在出错处进入跟踪状态。

例如，用 `options()` 函数进行设置后，前面那个求向量元素和的例子程序按最初的定义，运行时出现如下的选择：

```
## Error in f(1:5) : object 'n' not found
##
## Enter a frame number, or 0 to exit
##
## 1: f(1:5)
##
## Selection: f(1:5)
##
## Selection: 1
## Called from: top level
## Browse[1]>
```

在 Selection 后面输入了 1，就进入了函数内部跟踪。用 Q 终止运行并退出整个 browser 跟踪。当函数调用函数时可以选择进入哪一个函数进行跟踪。如果在 RStudio 中设置了 “Break in Code”，会自动在出错处进入跟踪运行状态。

19.5.7 stop()、warning()、message()

编写程序时应尽可能提前发现不合法的输入和错误的状态。发现错误时，可以用 stop(s) 使程序运行出错停止，其中 s 是一个字符型对象，用来作为显示的出错信息。

发现某些问题后如果不严重，可以不停止程序运行，但用 warning(s) 提交一个警告信息，其中 s 是字符型的警告信息。警告信息的显示可能比实际运行要延迟一些。

有些警告信息实际是错误，用 options() 的 warn 参数可以设置警告级别，如设置 warn=2 则所有警告当作错误处理。设置如

```
options(warn=2)
```

其中 warn=0 是默认做法，warn=1 表示不延迟显示。

函数 message() 与 stop()、warning() 类似，不算是错误或者警告，是提示性的信息输出。message() 不会像 warning() 那样延迟显示。比如，长时间的等待之前给出一个提示，正在读写文件或者网络时给出提示，等等。与 cat() 等相比较，cat() 是用户要求的输出，而 message() 是程序员对用户的提示。

19.5.8 预防性设计

在编写自定义函数时，可以检查自变量输入以确保输入符合要求。函数 stopifnot 可以指定自变量的若干个条件，当自变量不符合条件时自动出错停止。

例如，函数 f() 需要输入两个数值型向量 x, y, 需要长度相等，可以用如下的程序

```
f <- function(x, y){
  stopifnot(is.numeric(x),
            is.numeric(y),
            length(x)==length(y))
  ## 函数体程序语句...
}
```

19.5.9 出错处理机制

R 运行可以用 stop() 产生错误状态，停止运行；用 warning() 产生警告，用 message() 产生提示。基本 R 提供了 tryCatch() 函数，用来保护可能出错的代码，并可以指定出错时用的恢复或诊断代码。try() 函数也可以保护可能出错的代码，使得出错时错误信息照样显示但不中断运行。

扩展包 rlang 提供了一些对出错处理机制的增强功能。详见 (Wickham, 2019) 第 8 章：Conditions。

19.6 单元测试

现代程序设计的最重要的思想是模块化设计。将整个任务分解为功能明确的多个小模块，在 R 中用函数表示，确保每个函数的功能正确就可以使得整个程序正确。

在程序的反复修改调试过程中，程序员会使用许多调试代码测试函数是否正确运行，这些测试代码也是大的编程项目的重要内容，所以应该将这些测试代码收集起来放到一个或若干个程序文件中，每次程序修改后都重新运行这些测试代码，看结果是否改变了。

人工地去检查每个测试的结果是否正确过于繁琐了，没有很好地发挥计算机的自动执行繁琐任务的能力。所以，应该将测试时调用每个函数的正确结果与测试代码放在一起，直接判断函数的输出是否符合预期。R 的 testthat 扩展包提供了这样的自动化测试的功能。这个功能主要用于 R 扩展包的开发测试，但是较复杂的一般 R 程序也可以使用。

testthat 包用 `test_that()` 执行测试，用 `expect_equal()` 等函数规定所测试的函数（代码）应有的行为。

例如，自定义了如下的计算简单收益率的函数：

```
simple.return <- function(x){
  x <- as.vector(x)
  c(NA, diff(x) / x[1:(length(x)-1)])
}
```

单独写一个包含测试的 R 文件，如 `test-return.R`，内容如：

```
library(testthat)
test_that("simple.return calculates simple return", {
  expect_equal(simple.return(c(1, 2, 3)),
    c(NA, 1, 0.5))
  expect_equal(simple.return(ts(c(1, 2, 3))),
    c(NA, 1, 0.5))
  expect_equal(simple.return(ts(c(1, 2, 3),
    start=c(2000,1), frequency=12)),
    c(NA, 1, 0.5))
})
```

结果显示 `Test passed`。有多个测试时会显示有多少个通过，多少个失败，多少个被跳过，失败的测试会给出出错程序位置。在 RStudio 中运行时还用不同颜色区分通过、失败和跳过。一个测试文件中可以多个 `test_that()` 调用，每个调用测试一个特定的功能，每个 `test_that()` 调用中有若干个 `exepect_xxx()` 调用进行测试。除了 `expect_equal()` 以外，还有 `expect_match()`, `expect_true()`，`expect_output()` 等测试。详见 (Wickham & Bryan, 2022) 第 12 章：Testing。

19.7 环境

环境是 R 语言比较困难的概念，一般用户也不需要了解环境就能很好地使用 R，也不影响自定义函数。环境是支持变量作用域、命名空间、R6 类型等功能的数据结构，了解环境有助于更好地理解作用域等概念。

这部分内容主要来自 (Wickham, 2019) 相应章节。

19.7.1 基本概念

19.7.1.1 基本认识

环境作为一个数据结构与有名的列表相似，但是其中的名字必须都互不相同，且没有次序（类似集合），环境都有一个父环境，修改环境内容时都不制作副本。

rlang 扩展包可以比较方便地操作 R 的语法内容。可以用 `rlang::env()` 生成新的环境，这类似于 `list()` 函数的用法，如：

```
e1 <- rlang::env(
  a = FALSE,
  b = "a",
  c = 2.3,
  d = 1:3)
```

环境的作用是将一系列的名字（变量名、函数名等）与 R 对象绑定起来，即建立从名字到对象的对应关系，不计次序。对环境的修改是直接进行而不制作副本的。如：

```
e1$e <- list(x=1, y="abcd")
```

显示环境，只会显示一个地址信息，对用户没有什么用处：

```
e1
## <environment: 0x000001d185e88860>
```

rlang 包的 `env_print()` 函数可以给出较多的信息：

```
rlang::env_print(e1)
## <environment: 000001D185E88860>
## parent: <environment: global>
## bindings:
##  * a: <lgl>
##  * b: <chr>
##  * c: <dbl>
```

```
##  * d: <int>
##  * e: <list>
```

rlang::env_names() 可以获得环境中绑定的名字组成的字符型向量：

```
rlang::env_names(e1)
## [1] "a" "b" "c" "d" "e"
```

19.7.1.2 重要环境

rlang::current_env() 或基本 R 的 environment() 返回调用代码时所在的环境，比如，在命令行调用时，返回 <environment: R_GlobalEnv>。

rlang::global_env() 和基本 R 的 globalenv() 返回全局环境，这是另一个重要环境，也称为 "工作空间"，是在命令行运行时所处的环境。

为了比较两个环境是否同一个，需要用 indentical(e1, e2)，而不能用 == 比较。

19.7.1.3 父环境

每个环境都有一个父环境，这样在按句法规则查找变量的绑定对象时，就依次查找环境本身、其父环境、父环境的父环境，等等。内嵌函数的父环境是定义它的函数的内部环境。

rlang 的 env() 在生成环境时，可以用第一个无名的参数输入父环境，如：

```
e1 <- rlang::env(a=1, b=2)
e2 <- rlang::env(e1, c=3, d=4)
```

当 rlang::env() 没有输入父环境时，父环境就设为调用时的环境。用 rlang::env_parent() 获得父环境，用 rlang::env_parents() 获得各层父环境，如：

```
rlang::env_print(e1)
## <environment: 000001D185DDE7D0>
## parent: <environment: global>
## bindings:
##  * a: <dbl>
##  * b: <dbl>
rlang::env_parent(e2)
## <environment: 0x000001d185dde7d0>
rlang::env_parents(e2)
## [[1]]   <env: 000001D185DDE7D0>
## [[2]] $ <env: global>
```

对于以全局环境为父环境的环境，`env_parents()` 的输出截止到全局环境为止；但是，全局环境的上层还有加载的各个扩展包的环境，这也是查找变量的较后面的搜索路径。

为了制造一个上层不包含全局环境的环境，可以用 `rlang::empty_env()` 作为父环境，这个环境称为空环境，记为 `R_EmptyEnv`。

19.7.1.4 在上层环境中赋值

用“`<<-`”在各级父环境中赋值，最先在那一层父环境中找到变量就在那一层中赋值，如果直到全局环境都没有找到变量，就在全局环境中新建一个变量。全局变量应谨慎使用，它使得程序之间的数据输入输出变得不明晰。在使用闭包时常常需要使用这种赋值方式保存并修改一个闭包的状态。

如：

```
f0 <- function(){
  x <- 0
  f1 <- function(){
    f2 <- function(){
      x <<- x+1
      x
    }
    f2()
  }
  f1
}
f01 <- f0()
f01()
```

```
## [1] 1
```

```
f01()
```

```
## [1] 2
```

在上面的例子中，`<<-`首先在 `f1` 环境内查找 `x`，没有找到就继续向上在 `f0` 的环境内查找 `x`。

19.7.1.5 环境中名字的访问

类似于列表元素访问，用“环境 `$` 名字”格式或者“环境 `[["` 名字`"]]`”读取环境的元素，不存在时返回 `NULL`。如：

```
e1 <- rlang::env(rlang::empty_env(), x=1, y=2)
e2 <- rlang::env(e1, a=3, b=4)
e2$a
## [1] 3
e2[["b"]]
## [1] 4
e2$x
## NULL
```

从上例也可以看出不能直接读取父环境中的变量。

如果希望在找不到变量时出错，可以用 rlang::env_get(环境名，" 名字")，如：

```
rlang::env_get(e2, "x")
## Error in rlang::env_get(e2, "x") : 找不到对象'x'
```

可以设置 env_get() 在查找不到时的缺省值，如：

```
rlang::env_get(e2, "x", default=NA)
## [1] NA
```

为了在环境中增加绑定或重新绑定,可以用 $ 或 [[格式直接赋值,可以用 rlang::env_poke() 或 rlang::env_bind()，rlang::env_bind() 运行同时进行多个绑定，如：

```
e1 <- rlang::env(x=1, y=2)
e1$z <- 3
rlang::env_poke(e1, "a", 11)
rlang::env_bind(e1, b=12, c=13)
rlang::env_names(e1)
## [1] "x" "y" "z" "a" "b" "c"
```

用 rlang::env_has() 检查某个环境中是否绑定了指定的名字，如：

```
rlang::env_has(e1, c("x", "c", "f"))
##     x     c     f
##  TRUE  TRUE FALSE
```

为了在环境中删除一个名字的绑定，需要用 rlang::env_unbind()，如：

```
rlang::env_unbind(e1, c("z", "c"))
rlang::env_names(e1)
## [1] "x" "y" "a" "b"
```

注意 `env_unbind()` 只是解除了绑定，原来的对象并不会马上被删除，如果没有其它名字引用该对象，R 的垃圾收集器会随后删除该对象。

基本 R 的 `get()`,`assign()`, `exists()`,`rm()` 等函数起到与 rlang 包中的 `env_get()`,`env_poke()`, `env_has()`, `env_unbind()` 类似的功能，但是这些函数通常都针对调用时的当前环境，不容易处理其他环境，另外它们都有一个 `inherits` 选项默认为 `TRUE`，可以自动搜索父环境，所以不如 rlang 包的函数功能明确。

19.7.1.6 两种较少使用的特殊环境

`rlang::env_bind_lazy()` 可以创造延迟的绑定，就是类似于 R 函数形参缺省值的懒惰求值那样，第一次使用其值的时候才进行绑定。利用这种技术，可以实现类似 `autoload()` 的功能，`autoload()` 可以使得用到某个扩展包中指定的名字时才自动载入该扩展包，利用延迟绑定，可以使得数据框看起来像是已经在内存中，但实际是用到该数据框时才中硬盘中读入。基本 R 中提供了类似的 `delayedAssign()` 函数。

`rlang::env_bind_acitive()` 可以制造一个环境，每次访问环境中的名字都重新求值并绑定一次。基本 R 中提供了类似的 `makeActiveBinding()` 函数。

19.7.2 逐层向上访问环境

要逐层向上访问环境，可以利用 R 的递归函数。下面写一个逐层向上查找指定的名字所在的环境的自定义函数。(来自 Advanced R)。

```
where <- function(name, env = rlang::caller_env()) {
  if (identical(env, empty_env())) {
    # 找到了顶层都没有找到
    stop(" 找不到 ", name, call. = FALSE)
  } else if (rlang::env_has(env, name)) {
    # 在当前的 env 环境中找到了，返回找到时的环境
    env
  } else {
    # 利用递归向上层查找
    Recall(name, rlang::env_parent(env))
  }
}
```

自变量 `name` 是要查找的名字，`env` 是从那个环境开始逐层向上查找，`env` 的缺省值是调用 `where()` 函数时的环境。定义中分了三种情况：到顶层（空环境）都没有找到，出错停止；在向上逐层查找中在某个环境中找到了，返回找到时的环境；否则就利用递归向上层查找。

这个例子可以用作环境逐层向上遍历的模板。

19.7.3 特殊环境

实际上，一般用户都不会用到 `rlang::env()` 生成的环境；但是用户都会接触到使用 R 语言时自然产生的各种环境，只不过许多用户可能没有认识到自己是在使用环境。

R 语言使用中涉及到环境的情景有：

- 扩展包的环境和搜索路径；
- 函数环境；
- 命名空间；
- 运行环境。

19.7.3.1 扩展包环境

每次用 `library()` 或 `require()` 命令载入一个扩展包，它定义的变量和函数就构成一个环境，这个环境变成全局环境的父环境。这样，全局环境的各层父环境包括用户载入的扩展包和启动 R 会话时会自动载入的扩展包（如 stats 等），载入最晚的一个扩展包的环境是全局环境的父环境，载入越早的扩展包的环境距离全局环境的层次越远。实际上，在查找某个名字时，在当前环境没有找到时会逐层向父环境查找，这些父环境一般就包括全局环境和全局环境上层的各个加载了的扩展包形成的环境，这种搜索次序称为当前搜索路径，`search()` 返回当前搜索路径，如：

```
search()
## [1] ".GlobalEnv"        "tools:rstudio"     "package:stats"
## [4] "package:graphics"  "package:grDevices" "package:utils"
## [7] "package:datasets"  "package:methods"   "Autoloads"
## [10]"package:base"
```

`rlang::search_envs()` 返回以环境为元素的搜索路径。

搜索路径中除了全局环境之外，最后两个比较特殊：

- `Autoloads` 用类似于函数缺省值懒惰求值的方法在需要用到某个变量时才从磁盘将其载入到内存中，适用于占用存储空间很大的数据框之类的对象。
- `package:base` 是基本 R 的环境，必须先载入这一环境才能加载其它环境。可以直接用 `rlang::base_env()` 返回这一环境。

19.7.3.2 函数内部的环境

自定义函数包括形参表、函数体和定义时绑定的环境三个部分，非内嵌的也不在扩展包中定义的函数一般都与全局环境绑定，这样的函数的绑定环境没有什么用处，即使不了解环境部分也能够很好地使用这样的函数。

内嵌在函数内定义的函数称为闭包，闭包的绑定的环境是定义它的函数的内部环境，如果这个闭包作为定义它的函数的输出，闭包对象带有一个私有环境，即定义它的函数的内部环境，可以用来保存闭包函数的状态。在前面的讲到函数工厂时给出了闭包的例子。

用 `rlang::fn_env(f)` 可以求函数 `f` 的绑定环境。如：

```
f1 <- function(x) 2*x
rlang::fn_env(f1)
## <environment: R_GlobalEnv>
```

又如闭包：

```
f1 <- function(){
  times <- 0
  f2 <- function(){
    times <<- times + 1
    cat("NO. ", times, "\n", sep="")
  }
  print(rlang::fn_env(f2))
  f2
}
f2b <- f1()
## <environment: 0x00000201dc5096d8>
print(rlang::fn_env(f2b))
## <environment: 0x00000201dc5096d8>
f2b()
## NO. 1
f2b()
## NO. 2
```

这个例子显示的 `f2` 和 `f2b` 的环境都是 `f1` 内部的环境，在现实 `f2b` 的环境时虽然 `f1()` 已经结束运行，但是闭包可以保存其定义时的环境。

19.7.3.3 命名空间

变量名和函数名的搜索路径中包含了已载入的扩展包的环境，这就造成一个问题：后载入的扩展包中的函数会遮盖住先载入的扩展包中的同名函数，变量也是如此。所以，应该仅载入必要的扩展包，尽可能用“扩展包名:: 函数名”的格式调用。

这些问题是用户可控的，还有一个本质性的问题：假设扩展包 A 中的函数 `f1` 要用到扩展包 B 中的函数 `f11`，先载入了扩展包 B，然后载入了扩展包 A，这时调用 A 中的 `f1()` 没有问题。现在假设随后又调入了一个扩展 C，

扩展包 C 中也定义了一个 `f11` 函数，那么，现在调用 A 中的 `f1` 时，会调用 B 中的 `f11` 还是 C 中的 `f11`? 如果调用 C 中的 `f11` 就是会程序出错或给出错误结果。

为了避免这样的不可控的错误发生，R 语言的扩展包开发进行了严格的规定。R 的扩展包与两个环境有关，一个就是扩展包的环境，这实际是用户能看到的 R 扩展包提供的变量和函数，在载入扩展包时会插入到搜索路径中。另一个环境是命名空间环境，这是扩展包私有的一个环境，其中的变量和函数有一些对包的用户不可见，扩展包环境中那些用户可见的变量和函数也在命名空间环境中。R 扩展包在设计时都会利用命名空间严格限定包内部调用的其它包中的函数，不至于引起歧义。

每个扩展包的命名空间环境都有如下的一套上层环境：

- imports 环境，其中包含所有的用到的其它扩展包的函数，这是由扩展包的开发者确定的，所以不会错误调用错误的包；
- imports 环境的父环境是基本 R 环境对应的命名空间环境，但其父环境与基本 R 环境的父环境不同；
- 基本 R 命名空间环境的父环境是全局环境。注意基本 R 环境的父环境是空环境。

所以，扩展包内调用其它扩展包的函数是需要开发者明确地加入到 imports 环境中的，不受用户调用时载入了那些扩展包和载入次序影响。扩展包环境（针对用户的）和扩展包命名空间环境（包开发者自用）这 ‘两个环境不发生直接的引用联系，可以通过函数环境逐层向上变量发生联系。

19.7.3.4 运行环境

函数在调用执行时自动生成一个运行环境，其父环境为函数定义时的环境，比如，设 `f` 是在命令行定义的函数，调用 `f()` 时自动生成一个 `f` 的运行环境，相当于 `f` 的局部变量和形参的环境，其父环境为 `f` 定义时的环境，即全局环境。

设函数 `f2` 在函数工厂 `f1` 中定义并被 `f1` 输出为一个闭包 `f2b`，则调用 `f2b` 时自动生成一个 `f2b` 的运行环境，相当于 `f2b` 的局部变量和形参组成的环境，此运行环境的父环境是定义时的环境，即 `f2` 函数内部的环境。函数执行结束则运行环境消失。

为了能够保留下来运行环境，一种办法是将运行环境在运行时用 `rlang::current_env()` 获取并作为函数的返回值保存到变量中，另一种办法是像函数工厂那样输出一个闭包，闭包的环境就是函数工厂的运行环境。

19.7.4 调用栈

函数在被调用时，还涉及到调用它的环境，可以用 `rlang::caller_env()` 获得。调用环境与调用时的实参计算有关，需要了解调用栈 (call stack) 概念，调用栈由若干个分层的框架 (frames) 组成。R 运行出错时会显示一个 `traceback()` 结果，就是调用栈的各个框架。如：

```
f1 <- function(x) {
  f2(x = 2)
}
```

```
f2 <- function(x) {
  f3(x = 3)
}
f3 <- function(x) {
  stop()
}
f1()

##  Error in f3(x = 3) :
## 4. stop()
## 3. f3(x = 3)
## 2. f2(x = 2)
## 1. f1()
```

上面例子在用 `stop()` 产生出错信号时显示调用栈，下面的函数调用了上面的函数。

可以用 `lobstr::cst()` 显示函数的调用栈。

当调用时有懒惰求值时，调用栈就可能有多个分支。

调用栈中的每一次调用称为一个框架，或求值上下文 (evaluation context)。框架是支持 R 语言的重要成分，R 程序仅能对框架数据结构作很少的操作。每个框架有三个构成部分：

- 一个表示调用函数的表达式 `expr`。
- 一个环境，通常是调用的函数的运行环境。但是，全局框架的环境还是全局环境，使用 `eval()` 会造出一个框架，其环境则是可以由用户干预的。
- 父框架，即调用它的框架。

框架还会保存函数的一些非正常执行路径，如 `on.exit()` 指定的操作，出错时的处理。

R 采用句法作用域，即由定义决定变量作用域，有少数语言如 Lisp 采用动态作用域 (dynamic scoping)，即在调用栈上查找变量值。

19.7.5 将环境用作一般数据结构

环境中的变量都是引用，或者绑定，不需要制作环境副本，这使得环境可以当作一种高级的数据类型使用。

将环境作为一般数据结构使用，可以用在如下一些方面：

- 因为不需要制作副本，所以可以节省内存空间。但是直接使用环境不够友好，可以使用 R6 类型的数据，R6 类型是建立在环境的基础上的。
- 在自己建立的扩展包中，用环境保存包的状态。这样一个包中的函数多次调用时，可以在多次调用之间传递一些状态信息。可以在包中用 `get.xxx()` 函数和 `set.xxx()` 函数提供包用户访问状态的接口函数。
- 环境可以当作一个杂凑表用，杂凑表可以在常数时间将名字对应到值。

Part IV

研究项目管理和研究报告

Chapter 20

研究项目管理

执行一个数据分析研究项目，必须遵循一定的管理规则，才能事半功倍。

20.1 可重复式科学研究

现代科学研究之所以能被大众接受，成为主流世界观，很大程度上要归功于“可重复结果的试验”。现在生物、物理、化学等领域的科研越来越复杂，重复试验越来越难，但是不能重复的试验就备受争议。

数据分析项目也是这样，你的研究结果，一定要能够被第三方研究者所重复，要尽可能地提供详尽的代码和数据使得别人能够重复研究。

§20.6所述的“文学式编程”是一类详尽记录操作过程和代码、结果的工具。如果使用 R 作为计算工具，为了结果可重复，还应该记录下研究所用的操作系统、R 版本、调用的扩展包版本。`devtools::session_info()` 可以显示这些信息：

```
devtools::session_info()
```

R 扩展包 renv 支持给每个研究项目设置私有的扩展包集合，减轻了使用不同 R 扩展包版本的麻烦，见 §1.4.6的说明。

为了谨慎起见，可以将研究项目（包括数据、代码、分析报告、计算结果）复制到一个虚拟机中，在虚拟机中重复整个计算过程，然后将虚拟机整体备份。这样可以保证在操作系统、R 软件、R 扩展包升级或迁移后，仍可以重现原来的研究结果。

20.2 备份

数据科学研究人员的工作成果就保存在源程序代码、整理好的数据、研究报告中。如果不注意对这些成果定期备份，一旦发生硬件故障、软件崩溃，就不得不依赖于一些数据恢复工具，而这不一定有效。

现在存储资源十分便宜，所以备份只是一个好的习惯，没有实际操作的困难。常用的备份方式如：

- 使用闪存盘或者移动硬盘定期复制；
- 使用公开或私有的网盘服务，如百度网盘；
- 使用网络存储装置，或者自己架设 FTP 服务器用来上传备份；
- 使用 github、git 等源代码控制软件对源文件进行增量备份，等等。

如果使用 MS Windows 系统，备份某个子目录到闪存盘的典型命令行命令如：

```
xcopy d:\proj G:\proj /E/D/R/Y
```

其中 `d:\proj` 是要备份的子目录，`G:` 是闪存盘的驱动器。`xcopy` 命令的好处是可以备份下级子目录，而且可以要求仅复制有更新的文件而自动跳过没有变化的文件。可以将这样的命令保存为一个 `backup.bat` 文件，放在桌面上双击就可以执行备份任务。

20.3 源代码管理

源代码管理软件可以跟踪记录源文件的每一个更改，必要时回退到某一个时间节点的状态，在开发的软件系统达到一定成熟度后进行“快照”，发布一个可用的软件，然后再继续改进。另一个主要功能时在软件开发团队内进行协作，允许将不同程序员独立进行某个模块的开发并上传到团队公共空间，也可以让不同程序员对同一源文件进行更改。还有一些错误列表、待完成工作等功能。

一个有代表性的源代码管理软件是 github 网站服务。用户或者企业也可以架设内部的 git 服务器。

对于只有一个人的小规模开发项目，也可以不使用这些专用软件，而是定期将源文件复制为加日期标签的备份，如 `mysrc-2023-04-14.R`。

20.4 单元测试

数据科学研究项目会涉及大量代码编程，如果使用了特殊的新算法，就需要确保新算法的程序实现能在各种情况下正常工作。应采用模块化程序设计思想，将复杂任务分解为许多较简单的模块，以 R 函数的形式实现。为了保证结果正确，应对每个模块进行多种测试，确保各种输入产生的输出符合该模块的目标要求。R 的 testthat 扩展包和 RUnit 扩展包可以帮助管理这些测试代码，可以批量执行测试并汇总测试结果。

20.5 阶段任务管理

一个研究任务通常可以分解为若干个步骤，这些步骤有的有依赖关系，需要按一定次序执行，有的则可以并行执行。例如，步骤 A 是数据输入和清理，步骤 B 是数据筛选和变换，步骤 C1 是用逻辑斯谛回归建模，步骤 C2 是用随机森林建模，步骤 D 是比较两种建模结果并做出生成结果报告。这些步骤都应该写成 R 函数形式。对于较大规模

的数据或者比较复杂的计算密集型算法，每个步骤的计算运行时间可能会很长，比如，几分钟、几小时甚至几天。需要进行适当的管理，避免不必要的重复运行。但是，如果某个数据或程序的修改使得后续步骤结果过时，就必须重新执行后续步骤，否则整个项目结果就处于错误的状态，称为不同步的状态。比如，步骤 C2 的参数进行了调整，则步骤 A、B、C1 不需要重复执行，但是步骤 D 就需要重新执行。

对于比较简单的研究任务，代码和研究笔记（或研究报告）都可以写在一个 Quarto 文件中，Quarto 支持代码段执行结果的缓存，使得已经运行过的程序段除非程序被修改或者依赖的输入被修改都不必重复运行，也能够规定不同步骤之间的依赖关系，见22.14。

对于更一般的研究任务，应该使用一个依赖关系分析和控制执行软件来管理各个步骤的执行，使得整个项目处于同步状态。这样软件的典型代表是用于软件编译管理的 make 软件，make 软件根据软件开发项目规定的目标和源文件的依赖关系，有选择地仅编译有更新的源文件。R 的targets扩展包可以很好地管理用 R 作为主要计算软件的数据分析项目的各个步骤的同步执行。

下面举一个简单例子演示 target 的使用。设输入数据文件为 `heart.csv`，包含了若干病人的一些指标和有误心脏病 (AHD) 的标签。要分别用逻辑斯谛回归和随机森林方法进行拟合比较。

设项目的 `R/`目录中包含如下的各个步骤的函数：

```
get_data <- function(infile){
  read_csv(infile) |>
    select(-`...1`) |>
    mutate(
      AHD = ifelse(AHD == "Yes", 1, 0) ) |>
    na.omit()
}

fit_logistic_model <- function(data){
  glm(AHD ~ .,
      family = binomial,
      data = data) |>
    fitted() |>
    (\(prob) ifelse(prob > 0.5, 1, 0))()
}

fit_rf_model <- function(data){
  data |>
    mutate(AHD = factor(AHD)) |>
  randomForest(
    AHD ~ .,
    data = _) |> predict()
}
```

```
show_summary <- function(
  data, logistic_model, rf_model){
  cat("Logistic 回归结果: \n")
  table(data$AHD, logistic_model)
  cat(" 随机森林结果: \n")
  table(data$AHD, rf_model)
}
```

注意这些函数都是指定输入，以函数的返回值为输出，不需要人为地调用并保存函数结果，targets 包会管理这些调用和结果存储问题。

在项目目录中包含如下的 _targets.R 文件用于管理设置：

```
# _targets.R

library(targets)
source("R/functions.R")
tar_option_set(packages = c(
  "tidyverse", "randomForest"))

list(
  tar_target(file, "heart.csv", format = "file"),
  tar_target(data, get_data(file)),
  tar_target(logistic_model, fit_logistic_model(data)),
  tar_target(rf_model, fit_rf_model(data)),
  tar_target(summary, show_summary(
    data, logistic_model, rf_model))
)
```

在 _targets.R 文件中用 source() 载入了各个数据分析步骤对应的函数的源程序文件，用 tar_option_set() 设置了一些选项，比如要提前载入的 R 扩展包。在这个文件末尾，用 tar_target() 设置了每一个步骤（目标，target）。比如，名称为 file 的目标，仅仅是说明了 file 目标依赖于 heart.csv 这个文件，如果这个文件更改了，则依赖于 file 目标的其它步骤也需要重新运行。名为 data 的目标，则需要针对 file 目标调用 get_data() 函数，结果自动保存，并可以作为 logistic_model 步骤和 rf_model 步骤的输入。最后一个目标名为 summary。

用 tar_manifest() 可以在实际执行这些分析步骤之前，罗列出各个目标（步骤）：

```
> tar_manifest(fields = all_of("command"))
# A tibble: 5 × 2
  name           command
  <chr>          <chr>
```

```
1 file           "\"heart.csv\""
2 data           "get_data(file)"
3 logistic_model "fit_logistic_model(data)"
4 rf_model       "fit_rf_model(data)"
5 summary        "show_summary(data, logistic_model, rf_model)"
```

用 `tar_visnetwork()` 可以用图形显示各个目标之间的关系，见图20.1。

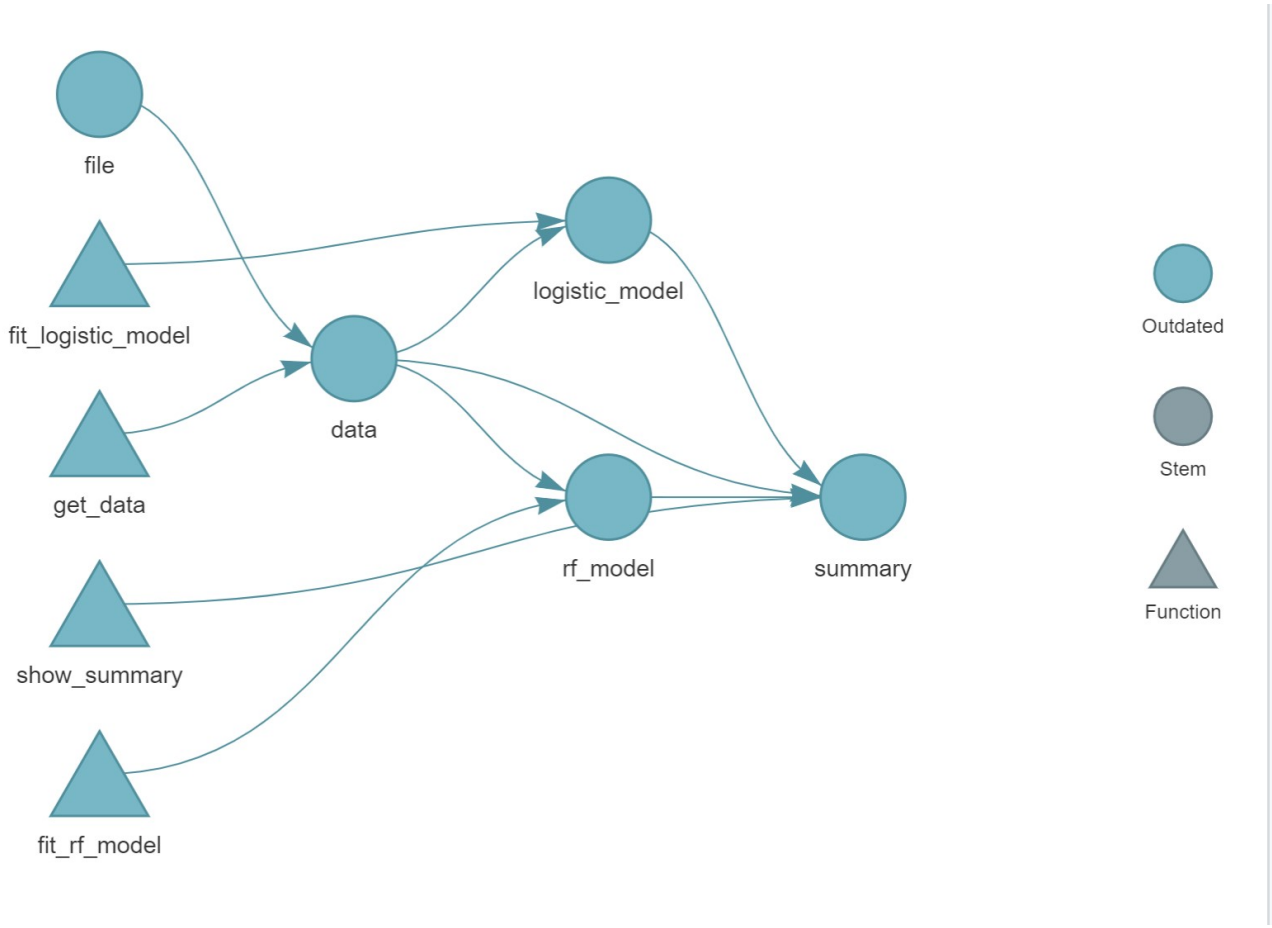


图 20.1: 各个步骤之间的关系，未执行时

在图20.1中，圆形代表一个节点（目标），三角形代表一个处理步骤（Function），用 Outdated 对应的颜色，代表某一个目标尚未生成或者生成后其依赖的目标或函数已经更新，需要重新生成。

用 `tar_make()` 执行所有步骤。如果某些目标已经是最新的，则相应的步骤不会执行。如 (输出信息有删减)：

- `start target file`
- `built target file [0.9 seconds]`
- `start target data`
- `built target data [0.25 seconds]`
- `start target logistic_model`

```
• built target logistic_model [0.01 seconds]
• start target rf_model
Logistic回归结果:
• built target rf_model [0.12 seconds]
随机森林结果:
• start target summary
• built target summary [0 seconds]
• end pipeline [1.44 seconds]
```

这时发现 show_summary() 函数定义有小错误，不能显示两个模型的结果。将该函数修改如下：

```
show_summary <- function(
    data, logistic_model, rf_model){
  cat("Logistic 回归结果: \n")
  table(data$AHD, logistic_model) |>
    print()
  cat(" 随机森林结果: \n")
  table(data$AHD, rf_model) |>
    print()
}
```

再用 tar_visnetwork() 作网络图显示，见图20.2。

图20.2，多个节点和函数已经是 “Up to date” 状态，只有节点 summary 因为其依赖的函数 show_summary 被修改而呈现出过时状态。

重新运行 tar_make()，这时将仅运行 show_summary 函数：

```
  skip target file
  skip target data
  skip target logistic_model
  skip target rf_model
• start target summary
Logistic回归结果:
   logistic_model
       0   1
  0 146  14
  1  27 110
随机森林结果:
   rf_model
       0   1
  0 138  22
```

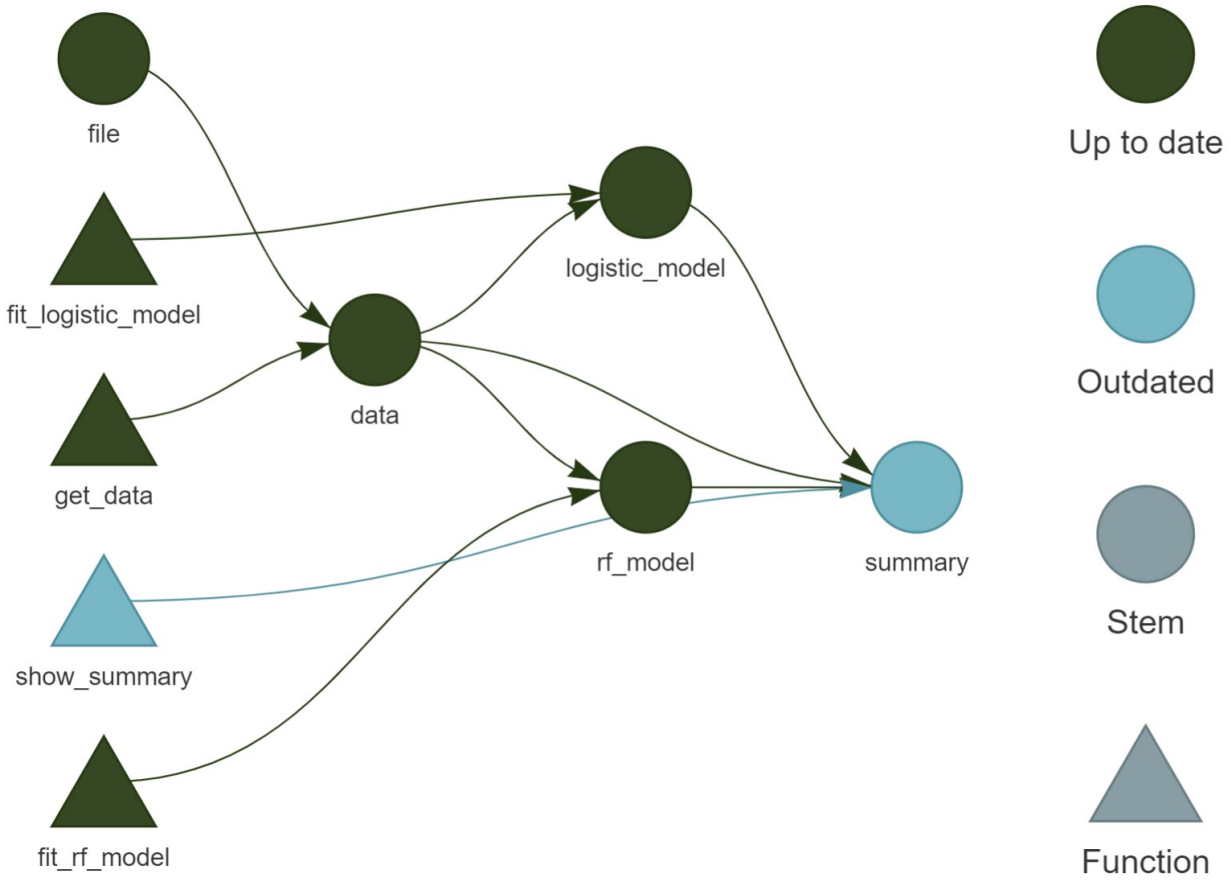


图 20.2: 各个步骤之间的关系，仅 summary 目标过时

```
  1  29 108
• built target summary [0 seconds]
• end pipeline [1.01 seconds]
```

这时再显示各步骤的网络图，就都是 “Up to date” 状态了，见图20.3。

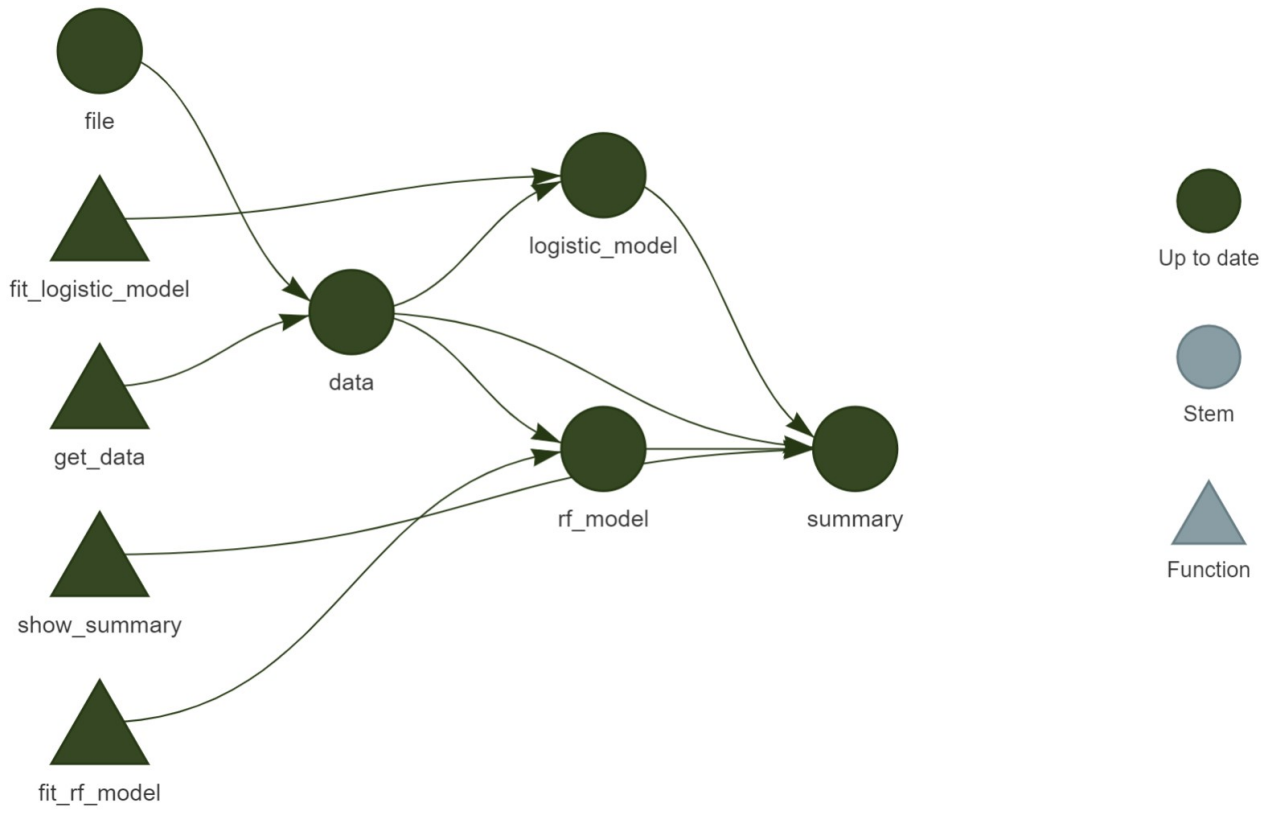


图 20.3: 各个步骤之间的关系，全都同步

如果原始数据文件被修改了，targets 也能发现这些修改，这时所有的目标都需要重新计算。

20.6 文学式编程

一个统计或数据分析的科研项目，都会产生一个或多个研究报告。因为使用统计与数据分析不可避免地有很多计算涉及在内，这里假设使用 R 软件做了计算。科研是一个不断改进的过程，所以每一次重新做了计算，研究报告中的汇总表格、图形都要更新。这样的任务比较繁琐，也容易出错。

“文学式编程”(literate programming, (Knuth, 1984)) 是这样一种思想，把撰写报告与计算程序有机地结合在一起，用一个源文件既包含报告内容，又包含计算程序。每次产生研究报告时，先运行源文件中的计算程序得到计算结果，这些结果包括文字性内容与图形，然后利用适当软件自动地把这些原始文字、计算结果组合成最终的报告。利用这样的思想，可以自动生成重复的例行报告，还可以作为 “可重复科学研究” 的载体。

这种做法也适用于将自己的科学研究过程保存并记录下来。可以在这样的源文件中记录自己的想法，将测试用的源程序保存在内，可以记录成功的试验和失败的试验，逐步地改进，起到一个能进行科学计算的笔记本的功能。将这样的思考过程记录下来并分享给别人，也能帮助别人了解你的研究过程。注意，失败的研究过程也不要从中删除，而是说明失败的原因，这可以让自己和别的读者避免类似的错误。

上述的源文件一般是文本文件，格式可以是 Markdown 格式，也可以是 LaTeX、Jupyter 笔记本等许多格式。R 的 knitr、rmarkdown、bookdown 等扩展包就是用来支持这样的编程思想。因为这些包主要与 R 挂钩，不利用推广到其它程序语言用户，所以 RStudio 的团队开发了开源软件 Quarto，可以将含有源程序的 Markdown 格式转换为各种包含程序和结果的各种输出格式。

Markdown 是一种很简单的文本文件格式，通常保存为.md 扩展名，利用 Quarto 支持的版本保存为.qmd 扩展名。

Markdown 文件里面有一些简单的格式标注方法，比如两个星号之间的文字会转化为斜体，缩进四个空格或一个制表符的内容会看成代码。从 markdown 格式比较适合转换为 html(网页) 格式，也可以转换为 MS Word 的 docx 格式，通过 LaTeX 编译器可以转换为 PDF 格式，也可以将 docx 转存为 PDF 格式，或者用输出到 PDF 文件的打印机将网页格式转换为 PDF 格式。

Jupyter 笔记本也是当前使用很多的一种包含文字说明、程序、结果的文件格式，扩展名为`.ipynb`，其主要格式也是 markdown，目前支持 Python、Julia、R 等程序语言。Quarto 软件可以将这样的笔记本文件转换为各种输出格式。

LaTeX 是一个文档排版系统，功能强大，结果美观，设计合理。缺点是需要学习类似于 HTML 的另一种语言。LaTeX 源文件主要是编译为 PDF。

R 扩展包 knitr 支持在 Markdown 格式、LaTeX 格式等类型的文件中插入 R 代码，经过转换，文中的 R 代码可以变成代码的结果文字、表格、图形，与原有报告文字有机地结合在一起。Jupyter 笔记本软件与 Quarto 软件配合也能实现类似功能。

插入了 R 代码的 markdowng 格式的文件一般以`.qmd` 为扩展名，R 的 knitr 包与 Quarto 软件一起为`.qmd` 提供了支持。在 RStudio 中对编辑、转换 quarto 格式有很好的支持。

Quarto 软件也支持生成学术类型的图书、论文，公式、定理可以自动编号并支持文献、公式、定理、图表、章节的引用和链接，支持生成 PDF、多文件互相链接的 HTML、Word 等输出。

统计数据分析、数据科学的研究报告和论文，通常含有如下的内容：

- 说明文字；
- 表格、图形；
- 分析源代码，以及运行结果；
- 数学公式。

虽然我们可以用 Word、LaTeX、Jupyter 笔记本完成这些功能，但各自有其缺点：

- Word 有很好的排版功能，容易学习和使用，但是数学公式的输入依靠鼠标菜单，速度比较慢，程序结果只能复制粘贴到文档内，重新运行程序后必须重新复制粘贴结果；

- LaTeX 学习起来比较困难，数学公式能力强，特别适用于科技论文、报告、书籍，但是程序结果的使用比 Word 还麻烦；
- Jupyter 笔记本对源程序和程序结果支持很好，可以使用 LaTeX 格式的数学公式，但是对写作科学论文用的公式编号、图表编号、引用链接等支持不够好。与 Quarto 软件配合可以弥补这些缺点。

所以，有写科技论文、报告要求的读者，应该学习使用 Quarto 软件。

下面两章先介绍基础的 markdown 格式，这是在.qmd 文件和 Jupyter 笔记本文件中文字部分使用的基本格式，然后介绍 Quarto 对 markdown 的补充。参考：

- https://quarto.org/
- (Xie et al., 2019)
- (Xie, 2017)

利用.qmd 格式、Jupyter 笔记本格式和 Quarto 软件可以执行如下的任务：

- 将单一的`.qmd` 文件或者`.ipynb` 文件编译为不同的格式，如：
  - HTML;
  - PDF;
  - MS Word。
- 在 RStudio 软件内制作笔记本文档，可以在其中包含说明文字与 R 代码，可以在笔记本文档内交互地运行 R 代码并能将结果交互地显示在笔记本内。
- 生成演示幻灯片，可以是基于 HTML5 的，也可以是基于 LaTeX beamer 扩展包的，或者 MS Powerpoint。
- 编写多章节组成的书籍。
- 编写期刊论文。
- 制作网站或者博客。
- 制作商业智能仪表盘 (dashboards)，即数据可视化展示。

Chapter 21

Markdown 格式

21.1 介绍

Markdown 是一种很简单的文本文件格式，通常保存为.md 扩展名。Mardown 中文内容应该使用 UTF-8 编码。Markdown 文件里面有一些简单的格式标注方法，比如两个星号之间的文字会转化为斜体，缩进四个空格或一个制表符的内容会看成代码，用井号开头的行表示标题，井号越多标题级别越低。

Markdown 适用于比较简单的文章、源程序说明等,不太适用于复杂的含有大量数学公式、图表的文章,从 markdown 格式比较适合转换为 html(网页) 格式，也可以转换为 MS Word 的 docx 格式，通过 docx 格式可以转存为 PDF 格式。Pandoc 软件、Quarto 软件、Jupyter 笔记本软件、R 的 knitr 等扩展包大大扩展了 markdown 格式的适用范围。

这里介绍的 markdown 格式是 Pandoc 软件进行了功能扩充后的版本。

21.2 Markdown 格式文件的应用

Markdown 格式用在一些微博、论坛中作为缺省格式,用户在网络浏览器软件的输入框中按照 markdown 格式输入，网站自动将其转换为 html 富文本内容显示出来。科学计算常用的 Jupyter 笔记本的文件格式也是基于 markdown 格式。

因为 markdown 格式就是纯文本，而且其格式十分简单，所以可以仅仅用普通文本编辑器编写 markdown 格式的文件，不需要转换为其它格式。

有一些独立运行的编辑软件，在其中输入了 markdown 格式文件后，可以并列地显示转化为 html 富文本后的结果，很多还支持自动备份到云服务中。但是，这些独立运行的编辑器大都有商业因素。比如，国内的印象笔记软件是提供了云存储的笔记软件，在其 PC 端就支持创建 markdown 格式的笔记，可以在输入时即时显示 html 效果。

RStudio 软件不是专用的 Markdown 编辑器，它是一个 R 程序的集成编辑、运行环境，对个人用户免费，在 R Studio 中可以编辑 Markdown 文件和含有 R 代码的 Markdown 文件，可以一键将其转化成 HTML、MS Word docx 文件，在单独安装的 LaTeX 编译软件支持下还可以直接编译成 PDF。RStudio 支持下的增强的 Mardown

格式，称为 RMarkdown 格式，以`.Rmd`为扩展名，支持大多数的 LaTeX 公式，在 bookdown 扩展包支持下还支持公式、定理、图表等自动编号和引用、链接。这个格式的升级版本称为 Quarto 格式，以`.qmd`为扩展名。

21.3 markdown 格式说明

这部分内容参考了 markdown 手册和 Rstudio 的文档，以及 (Xie et al., 2019)。

21.3.1 概述

Markdown 格式是 John Gruber 于 2004 年创造的，Markdown 的目标是实现“易读易写”。Markdown 定义了一种简单好用的文本文件格式，作为单独的文本文件，此格式没有什么多余的标签，又可以转化为很多其它的格式。

Markdown 的语法全由一些符号所组成，这些符号经过精挑细选，其作用一目了然。比如：在文字两旁加上星号，看起来就像*强调*。Markdown 的列表看起来就像我们平常在邮件中写一个列表的方法。Markdown 的区块引用看起来就真的像是引用一段文字，就像你曾在电子邮件中见过的那样。

需要时，可以直接在 markdown 中写 HTML 标记内容。markdown 能实现的功能是 HTML 的一部分，但是比 HTML 内容更干净，没有掺杂过多的与要表达的意思无关的标签。Markdown 的理念是，能让文档更容易读、写和随意改。

21.3.2 段落

一个段落由一行或连续的多行组成。段落之间以空行分隔。同一段落内的不同行在转换成 HTML 或 docx 等格式后会重新排列，原来的段内换行被当成了空格，这样的规定与 LaTeX 类似。普通段落不该用空格或制表符来缩进，不应在行尾留有空格。

为了在段内换行并且转化后仍保持段内换行，输入时在前面行的末尾输入两个或两个以上空格。例如：

```
白日依山尽，黄河入海流。
欲穷千里目，更上一层楼。
```

显示结果为：

白日依山尽，黄河入海流。
欲穷千里目，更上一层楼。

这样做的缺点是末尾的空格时不可见的。可以使用 HTML 的 `<br>` 标签在段内换行，如：

```
白日依山尽，黄河入海流。<br>
欲穷千里目，更上一层楼。
```

结果为

白日依山尽，黄河入海流。欲穷千里目，更上一层楼。

21.3.3 段内文字格式

在一段内，用星号或下划线包围的内容如 `* 强调 *` 是*强调*格式。用双星号或双下划线包围的内容如 `** 加重 **` 是**加重**格式。星号、下划线与要强调或加重的内容之间不要空开，否则会当作普通星号或下划线解释，在行首还会当作列表。为了插入普通的星号或下划线，可以使用反斜杠保护，或者写成段内代码格式。

可以用一对 `~` 作为界定符给出下标，如 `HO~2~` 会变成 HO_2。可以用一对 `^` 作为界定符给出上标，如 `Cu^2+^` 会变成 Cu^{2+}。但是，数学公式一般还是应该使用 LaTeX 数学公式形式（见22.10）。

在普通段落内一部分内容希望显示成代码，对其中的特殊字符不进行解释，只要包在两个反向单撇号内。如`` `if(_x_ > 0) y=1;` ``会变成 `if(_x_ > 0) y=1;`。如果内容本身就包含反向单撇号，可以在两边使用更多个数的反向单撇号，如`` `` `x` `` ``会变成`x`。

在 Markdown 文件中，为了使得某些有特殊意义的字符不作特殊解释，可以在该字符前面加上反斜杠\，取消其特殊含义。

为了原样显示反向单撇号，可以在两边用双写的反向单撇号界定字符串。

21.3.4 标题和分隔线

以一个井号 (“#”) 开始的行是一级标题，以两个井号 (“##”) 开始的行是二级标题，……，以六个井号开始的行是六级标题。标题行前面应该空一行，否则可能把某些偶然出现在行首的 “#” 号误认为标题行的标志。

对一级标题，也可以用标题内容下面输入一行等于号 “=” 表示上一行内容是一级标题。对二级标题，可以用标题内容下面输入一行减号 “-” 表示上一行内容是二级标题。等于号和减号的个数不限。

用三个或三个以上连续的星号组成的行，可以转换成分隔线。下面是一个分隔线：

21.3.5 引用段落

可以用类似 Email 的回复包含原始邮件内容的办法输入引用段落，即，在段落的每行前面加一个大于号。比如下面的诗：

```
> 白日依山尽，黄河入海流。
> 欲穷千里目，更上一层楼。
```

转换成

> 白日依山尽，黄河入海流。欲穷千里目，更上一层楼。

注意引用也是段落模式，内容中的换行不起作用，空行导致分段。

引用段落也可以仅在段落第一行写大于号，其它行顶格写，例如下面的两段引用：

```
> 远上寒山石径斜，
白云生处有人家。
>
> 停车坐爱枫林晚，
霜叶红于二月花。
```

转换成

> 远上寒山石径斜，白云生处有人家。
>
> 停车坐爱枫林晚，霜叶红于二月花。

引用也可以嵌套，如：

```
> 张三说：李四这样说过
>
>> 不想当将军的木匠不是好厨子。
>
```

转换成

> 张三说：李四这样说过
>
> > 不想当将军的木匠不是好厨子。

注意嵌套内容前后都有空的引用行，否则不能实现嵌套引用。

引用内也可以嵌套其它的 Markdown 格式如标题、列表等。引用前后应该有空行把引用内容与其他内容分隔开。

为了在引用中换行，就需要加引用空行。为了排版诗、词之类的内容，希望人为控制换行和引导空格，可以将引用中的 > 替换成 |，如：

```
| 白日依山尽，
| 黄河入海流。
| 欲穷千里目，
| 更上一层楼。
```

转换成

白日依山尽，
黄河入海流。
欲穷千里目，
更上一层楼。

```
| 枯藤老树昏鸦，
|   小桥流水人家，
|   古道西风瘦马。
| 夕阳西下，
|   断肠人在天涯。
```

转换成：

枯藤老树昏鸦，
小桥流水人家，
古道西风瘦马。
夕阳西下，
断肠人在天涯。

其中不用 | 开头的行仍会当作上一行的续行，不会强行换行。

21.3.6 列表

列表分为不编号的列表和编号的列表。不编号的列表转化后通常显示圆点开头的列表项。

在 Markdown 中，用星号表示一个不编号的列表项。星号也可以替换成加号或减号，后面必须有一个或多个空格。每个列表项可以输入多行，各行的内容最好左对齐，左对齐在使用文本格式时较易阅读，但不是必须的。两个列表项之间不要空行。例如：

```
* 白日依山尽，
  黄河入海流。
* 欲穷千里目，
  更上一层楼。
```

转换为

- 白日依山尽，黄河入海流。
- 欲穷千里目，更上一层楼。

段落项头的数字加句点和空格表示编号列表，两个列表项之间尽量不要空行。例如：

```
1. 第一种解决方法，
   收买敌人的高官。
2. 第二种解决方法，
   尽可能拖延。
```

转换为

1. 第一种解决方法，收买敌人的高官。
2. 第二种解决方法，尽可能拖延。

标准的 markdown 编号列表不能自己定义数字的显示格式，不允许开始值不等于 1。pandoc 支持更自由的列表，允许输入时有括号或右括号，允许使用字母和罗马数字，但是括号会被去掉。如

```
(1) 顶层一；
    a) 内层二；
    b) 内层三；
(2) 顶层二。
```

转换为

(1) 顶层一；

    a) 内层二；
    b) 内层三；

(2) 顶层二。

为了避免错误地产生非本意的编号列表，在行首写数字加句点和空格时，可以在句点前加反斜杠，或者在句点前加空格。例如，下面是一个年号：

```
2016\.
```

如上输入将不会错误地解释为有序列表。

如果列表项目中有多个段落，这时两个列表项之间应该以空行分隔，每个项目除了第一行外，输入的每行内容都应该缩进 4 个空格或者一个制表符。例如：

```
*   R语言第一个版本开发于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”
    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。

*   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。
```

```
    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。
```

转换为

- R 语言第一个版本开发于 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”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。

- 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。

  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。

列表项目内如果有引用段落，需要都缩进 4 个空格。如果有程序代码，需要缩进 4 个空格后用三个反单撇号表示开始与结束。

列表可以嵌套，嵌套的列表需要缩进 4 个空格，中间不需要空行。例如：

```
1.  第一类工作包括：
    + 技术服务；
    + 咨询服务。
2.  其它工作略。
```

转换为

1. 第一类工作包括：
   - 技术服务；
   - 咨询服务。
2. 其它工作略。

如果需要把每个列表项当作段落排版，可以在每个列表项后空行。

21.3.7 源程序

为了让源程序能够自动显示成源程序的样式，而不至于自动分行、特殊字符解释，最初的 markdown 的规定是用空行把源程序与其它内容隔开，并把源程序行都缩进 4 个空格（或以上）或者一个制表符。源程序格式持续到不缩进 4 个空格的地方为止。

例如，下面的输入：

```
    f <- function(x){
        n <- length(x)
        y <- numeric(n)
        y[x >= 0] <- 1
        ##y[x < 0] <- 0

        y
    }
```

转换为

```
f <- function(x){
    n <- length(x)
    y <- numeric(n)
    y[x >= 0] <- 1
    ##y[x < 0] <- 0

    y
}
```

更适当的做法是用三个连续的反向单撇号表示代码开头与代码结束，中间就会当作源程序代码处理。例如下面的输入

```
```
> x <- rnorm(100)
> hist(x)
```
```

转换为

```
> x <- rnorm(100)
> hist(x)
```

R 的 knitr 包在 Markdown 格式的文件中插入 R 可执行代码时，就用了这样的方法。而且，R Markdown 格式的代码块不需要用空行与前后分隔开。如果代码内本身含有反向单撇号行，只要使得开头与结尾标志中的反向单撇号个数更多就可以了。这种做法称为 “栅栏式” 代码段。

使用栅栏式代码段时可以在开始行的行尾写大括号，在大括号内写选项。其中一种选项是要求按照某种编程语言对结果进行彩色语法显示，如.cpp 表示 C++，.c 表示 C，.r 表示 R，.python 表示 python 等。选项`.numberLines`要求该代码行编号。如:

```
```{.cpp .numberLines startFrom=101}
#include <math.h>
double sqr(double x){
  return(x*x);
}
```
```

转换为

```
#include <math.h>
double sqr(double x){
  return(x*x);
}
```

21.3.8 链接

最简单的链接是原样显示的可点击的链接，只要把链接地址用小于号和大于号包在中间，两边用空格和其它内容隔开。如果是网页，需要加 `http://`，如果是邮箱，需要加 `mailto:`。例如，如下代码:

```
北京大学的网页地址是： <http://www.pku.edu.cn/> 。
```

显示为

北京大学的网页地址是：http://www.pku.edu.cn/ 。

除此之外，Markdown 还支持两种形式的链接语法：行内式和引用式两种形式。不管是哪一种，链接的显示文字都是用方括号 `[...]` 来标记。

要建立一个行内式的链接，只要在方括号内写链接的显示文字，右方括号后面紧接着圆括号，并在圆括号中间插入网址链接即可。方括号部分与圆括号部分之间不能断开。例如：

```
请参考：[李东风的教学主页](http://www.math.pku.edu.cn/teachers/lidf/course/index.htm)
```

变成了

请参考：李东风的教学主页

在圆括号中，链接后面还可以包含用双撇号包围的标题文字，与链接之间用空格分开。

引用式的链接，需要在某处（比如文章结尾）定义一些链接的标识符，然后用方括号包围链接的显示文字，后面紧接着方括号包围着链接的标识符。

为了定义链接标识符，用方括号包围链接标识符，前面可以有至多 3 个空格缩进，右方括号后面紧接着冒号和一个空格，空格后写链接地址，然后空格，在两个双撇号中间写一个链接地址的标题。例如

```
[baidu]: http://baidu.com/ "百度"
[pku]: http://www.pku.edu.cn/ "北大"
```

这时，在文章中就可以用标识符调用链接，如 `[北京大学主页][pku]` 将变成链接北京大学主页。

使用引用式的链接，有些像论文中把所有参考文献排列在文章末尾，文中用到某一篇文献只要提及其序号。

还有一种链接是内部链接，用于文内跳转。在各级标题行的末尾，可以添加 `{# 自定义标签}` 这样的内容，其中“自定义标签”是自己写的一个标识符，标识符仅使用英文字母、数字、下划线、减号，用来区分不同的位置。在 Quarto 文件中需要以“`#sec-`”开头比如，本文第一节“介绍”添加了 `sec-markdown-intro` 为标签，就可以用“`[回到介绍](#sec-markdown-intro)`”产生链接回到介绍。

21.3.9 插入图形

图形只能用链接形式，不可能保存到一个纯文本文件内。图形文件可以存在于远程服务器上，也可以是与生成的 HTML 文件在同一目录结构中的文件。语法仍使用行内式和参考式两种形式。转化成 HTML、PDF、Word 格式后可以把图形内嵌在输出文件内部。

行内式的图片链接，是普通行内链接格式前面添加了一个叹号，惊叹后面紧接着方括号，方括号内写图片的标题，标题可以空缺，在右方括号后面紧接着圆括号，圆括号内写图片的链接。

例如，下面的代码可以插入百度的一个 logo，使用的是网上的资源:

```
![](http://www.baidu.com/img/baidu_jgylogo3.gif)
```

结果为

为了插入保存在本地的与 Markdown 源文件在同一子目录或下级子目录的图形，只要在圆括号中写图片文件名（如果与 Markdown 源文件在同一子目录）或相对路径。例如，在 `D:\work\figs` 下的图形文件 `baidu_logo.gif` 在本 Markdown 源文件所在目录的 figs 子目录中, 可以用代码

```
![](figs/bd_logo.png)
```

结果为

这样含有网上图片和本地图片的 Markdown 源文件转化为 HTML 和 docx 格式，都可以正常显示插入的图片。

与链接类似，也可以在文章某处（比如末尾）定义图片的标识符，然后把行内图片引用中图片地址替换成图片标识符即可。

在使用 RStudio 的可视化方式编辑 Quarto 格式文件时，可以直接将图形文件复制粘贴入正在编辑的文件中，相应的图形文件自动复制保存在项目目录中，这可以简化插入图片的操作。Quarto 中有选项可以调整图片大小、显示标题等属性。

21.3.10 表格

Markdown 文本格式的表格就像是用减号、等号、竖线画的文本格式表格一样，转化为 HTML、docx 等格式后就变成了富文本的表格。有如下几种表格：

- 管道表
- 简单表
- 换行表
- 有格表

21.3.10.1 管道表

管道表在两列之间用竖线分开，在列标题下面用减号画横线，用如下方法指定各对齐方式：

- 在列标题下的横线开始加冒号，表示左对齐；
- 在列标题下的横线末尾加冒号，表示右对齐；
- 在列标题下的横线两端加冒号，表示居中对齐；

- 列标题下面仅有横线没有冒号，表示缺省对齐方式，一般是左对齐。

在表格内容后面空一行后写用 `Table:` 开头的表格说明。

这种方法不需要输入内容上下对齐，适用于中文内容。后面所讲的简单表、换行表、有格表需要能够输入内容对齐，对于中英文混合内容很难做到对齐，所以仅管道表比较适合中文内容。

例如:

```
| 姓名    |  收入       |    职业           | 颜色偏好  |
|:------|-----------:|:---------------:|-----------|
| 赵四海  |    123456  |   业务经理       |   红      |
| 刘英    |        50   |     无          |   蓝      |
| 钱德里  |      3200   |    保洁         |    灰     |

Table: 管道表示例
```

结果为

表 21.1: 管道表示例

姓名	收入	职业	颜色偏好
赵四海	123456	业务经理	红
刘英	50	无	蓝
钱德里	3200	保洁	灰

管道表不允许输入单元格换行，单元格内容太宽时转换结果可能自动换行，自动换行时列宽度与输入的列标题下横线宽度成比例。

21.3.10.2 简单表

简单表的格式是，第一行是各列标题，第二行是各标题下面用减号组成的表格线，同一行的不同列要用空格分开，从第三行开始是内容。

在表格前或表格后用空行隔开的以 `Table:` 开头的行是表格说明或标题。

为了确定表格每列单元格内容如何对齐，用列标题下的表格线给出提示：

- 表格线与列标题右对齐，表示该列右对齐；
- 表格线与列标题左对齐，表示该列左对齐；
- 列标题在表格线中间，表示该列居中对齐；
- 列标题左右都与表格线对齐，表示该列为缺省对齐方式，一般是左对齐。

一定要使用一个等宽字体来编辑这样的表格，否则对齐与否无法准确分辨。单元格内容不能超出表格线左端。经过试验发现，中文内容很难按这种方法对齐。

例如：

```
Name        Income          Job              Color
------    --------     ------------------    -----
Jane        123456     Research Assistant    red
John           50        N/A                 blue
William      3200      Cleaner               blue

Table: 一个简单表的例子
```

结果为：

表 21.2: 一个简单表的例子

Name	Income	Job	Color
Jane	123456	Research Assistant	red
John	50	N/A	blue
William	3200	Cleaner	blue

21.3.10.3 换行表

换行表在输入列标题和单元格内容时，允许输入内容拆分行，但是转化后并不拆分行。这样的表以一行减号开始，以一行减号结束，中间的表格用空行分开实际的不同行。例如：

```
-------------------------------------------------------
Name
of
Subject     Income        Job                color
------    --------     ------------------    -----
Jane        123456       Research            red
Ayer                     Assistant

John           50        N/A                 blue
Tukey

William      3200      Cleaner               blue
Tale
```

```
-----------------------------------------------------

Table: 一个换行表的例子
```

结果为：

表 21.3: 一个换行表的例子

Name of Subject	Income	Job	color
Jane Ayer	123456	Research Assistant	red
John Tukey	50	N/A	blue
William Tale	3200	Cleaner	blue

换行表输入时各列的输入宽度是有作用的，输入较宽的列结果也较宽。

21.3.10.4 有格表

完全用减号、竖线、等于号、加号画出表格线。这样的表在文本格式下呈现出很好的表格形状。转化后不能指定对齐方式。

例如：

```
Table: 有格表示例

+---------------+---------------+--------------------+
| Fruit         | Price         | Advantages         |
+===============+===============+====================+
| Bananas       | $1.34         | - built-in wrapper |
|               |               | - bright color     |
+---------------+---------------+--------------------+
| Oranges       | $2.10         | - cures scurvy     |
|               |               | - tasty            |
+---------------+---------------+--------------------+
```

结果为

表 21.4: 有格表示例

Fruit	Price	Advantages
Bananas	$1.34	• built-in wrapper • bright color
Oranges	$2.10	• cures scurvy • tasty

21.4 附录：pandoc 软件介绍

pandoc是一个杰出的开源软件，作者为 John MacFarlane。RStudio 软件中已经包含了这个软件。此软件在操作系统命令行运行，可以用来在多种不同的文件格式之间进行转换，输入格式一般是文本格式，比如 markdown、LaTeX、MediaWiki、reStructured Text、HTML、DocBook，但是也可以输入 MS Word docx、Open Office ODT、EPUB 格式。输出可以是这些文本格式，也可以是 MS Word docx、Open Office ODT、EPUB、LaTeX 等格式，在安装了 LaTeX 编译系统如 MikTeX 时可以输出为 PDF。

可以用普通文本编辑器编写 markdown 格式的文件，用 pandoc 转换为 HTML 或 MS Word docx 等格式。因为 pandoc 需要 UTF-8 编码的输入文件，所以应该把 markdown 文件保存为 UTF-8 格式，MS Windows 下的 Notepad++ 软件可以很容易地编辑文本文件并在各种编码之间转换。

首先，从 pandoc 网站下载并安装 pandoc。安装程序很奇怪地安装到了 `C:\Users\`登录用户名`\AppData\Local\Pandoc` 中，请将此子目录复制到一个合适的位置，比如 `C:\Pandoc`。如果把此路径加入到 Windows 系统的可执行文件搜索路径中（在 “控制面板-系统和安全-系统-高级系统设置-环境变量” 中，为系统变量的 Path 添加一个分号分开的路径 `C:\Pandoc` 即可以不用全路径访问 pandoc.exe 可执行文件。

为了用 pandoc 转化某个 markdown 文件，首先在该文件所在子目录打开一个 Windows 命令行窗口，在 MS Win10 系统中只要在文件管理器的 “文件” 菜单选择 “打开 Windows PowerShell” 即可。比如，文件名是 `test.md`, 用如下 pandoc 命令可以转换为.docx 文件 (MS Word 文件的新版本):

```
C:\Pandoc\pandoc -o test.docx test.md
```

用如下 pandoc 命令可以转换为.html 文件:

```
C:\Pandoc\pandoc -s -o --mathjax test.html test.md
```

如果把 pandoc.exe 加入了 Windows 操作系统的 Path 环境变量中，上面的 `C:\Pandoc\pandoc` 可以简写为 `pandoc`。

如果使用 RStudio 编辑 markdown 文件、Quarto 或者 R Markdown 文件，它会自动调用内置的 pandoc 程序。

Chapter 22

Quarto 格式文件

22.1 介绍

Quarto是 RStudio 团队开发的一个开源软件，可以将包含 R、Python、Julia、Observable JS 源程序的 markdown 文件转换为各种输出格式，这些源文件可以是普通的包含程序代码的 markdown 文件 (扩展名为`.qmd`)，也可以是 Jupyter 笔记本文件 (扩展名为`.ipynb`)。支持 HTML、MS Word、LaTeX PDF、ePub、网站、幻灯片等多种输出格式。

软件基于 Pandoc 的扩展的 markdown 文件转换功能。对于其中的程序代码运行的支持，可以在 R 的 knitr 包支持下运行并插入运行结果，也可以在 Jupyter 软件的支持下运行并插入运行结果。

有另外一种较早的 “R Markdown” 格式，与 quarto 格式相似，但是 R markdown 主要支持针对 R 语言用户，而 quarto 软件则可以支持多种编程语言，可以适用于更广泛的程序语言用户。

现在 qmd 格式还有一些不完善的地方，比如多行公式编号还没有实现，仅能对一个公式使用统一编号。

22.2 安装

Quarto 软件有一个单独的可执行程序，为了实现所有功能应安装此可执行程序；如果仅在 RStudio 中使用其功能，而不需要使用 Jupyter 格式，可以不必安装此程序。

为了在 R 和 RStudio 中使用 Quarto 格式，应同时安装 rmarkdown 扩展包，最好也安装 quarto 扩展包。

RStudio 支持使用 “可见即所得” 方式编辑 qmd 格式的文件，所以是 Quarto 软件很好的伴随软件。即使使用 Python 或 Julia 作为主要编程语言，也可以用 RStudio 来进行 qmd 文档的编辑和转换。RStudio 可以使用已安装的独立的 Quarto 软件，也可以使用内置的 Quarto 功能。

为了能够将 Quarto 源文件通过 LaTeX 转换为 PDF，需要安装 TinyTeX 软件，这是一个 LaTeX 编译系统软件，安装方法参见 §22.19。

22.3 简单样例文件

Quarto 可以用来生成多个源文件组成的图书、网站，但最简单的用法是用单个源文件生成单个的网页、Word、PDF 文件结果。在 RStudio 中，可以用 “File – New File – Quarto Document” 菜单生成一个包含模板内容的空文件。在文件开头，一般有用三个减号组成的行界定的一部分，称为 YAML 属性设置。后面是正文内容。一个简单的文件如：

```
---
title: " 演示基本功能"
format: html
---


## 格式介绍

Quarto格式基于markdown格式与pandoc软件，
可以转换“.qmd”和“.ipynb”(Jupyter笔记本)格式的文件，
参见<https://quarto.org>.

## R代码

可以用如下方法插入R代码段：

```{r}
1 + 1
```

可以用“`#|`”添加特殊注释，
作为代码段属性，如：

```{r}
#| echo: false
2 * 2
```
```

22.4 RStudio 的可视化编辑

RStudio 支持 visual（可视化）编辑模式和 source（源代码）编辑模式。可视化编辑对初学者比较友好，所以习惯了用 Word 的用户可以从可视化编辑方法开始使用 Quarto 的功能。在可视化编辑方式中，可以直接将图形粘贴入编辑界面，这时 RStudio 会将该图形的副本复制保存进入项目目录中，也可以修改图形的大小、标题等内容。

可视化编辑的一个缺点是会自动调整换行，这使得切换回到源代码编辑模式时感觉自己输入的内容一片混乱，提供的各种设置都不能解决这个问题。习惯使用源代码的用户不要切换到可视化编辑模式！如果偶尔不小心切换过去了，可以切换回到源代码模式然后用 Windows 的 `Ctrl+Z` 快捷键恢复编辑内容。

因为可视化编辑模式也有很大的优势，对于习惯源代码方式的人，可以打开一个小的临时文件使用可视化编辑方法编辑，然后将生成的源代码剪切进入主要的源文件。

22.5 生成输出文件

22.5.1 使用 RStudio

Quarto 格式可以生成 HTML、Word、PDF 等输出格式。输出 PDF 格式需要安装 TinyTeX 软件，见22.19。

只要在 RStudio 的文件编辑窗口点击 “Render” 快捷图标就可以转换为指定的输出文件。如果使用 Preview 快捷图标，则支持在修改文件后自动更新结果。

还可以在 R 中安装 quarto 扩展包，并用如下 R 命令转换：

```
library(quarto)
quarto_render("index.qmd")
```

可以加选项 `output_format = "pdf"` 或 `output_format = "docx"` 转换为 PDF 或 Word 格式。

22.5.2 使用 Quarto 可执行程序

也可以不借助于 RStudio，而是单独安装 Quarto 的可执行程序，在操作系统命令行使用如下命令转换某个文件：

```
quarto render index.qmd
```

这默认生成 HTML 结果，在上述命令末尾空格并添加`--to pdf` 转换为 PDF 格式（需要预先安装 TinyTeX 软件），添加`--to docx` 转换为 MS Word 格式。

Quarto 可以使用 R 的 knitr 扩展包或者 Jupyter 软件执行 qmd 文件中的代码。可以用 YAML 选项 `engine: knitr` 选用 knitr, `engine: jupyter` 选用 Jupyter。

22.5.3 Jupyter 笔记本源文件的转换

在操作系统中运行如下命令，可以建立 Jupyter 笔记本对 HTML 格式输出结果的预览：

```
quarto preview mydoc.ipynb
```

这个预览会自动更新，也就是说，每次笔记本保存时，预览会自动刷新。如果要生成最后结果，不需要自动刷新，可以用 `render` 命令，如：

```
quarto render mydoc.ipynb
```

为了生成其它格式的输出，可以加`--to` 选项，如：

```
quarto render mydoc.ipynb --to docx
```

生成 Word 格式，`--to pdf` 生成 PDF 格式。

Jupyter 笔记本文件如果需要一些关于 Quarto 的设置，仍可以在文件头部添加 YAML 选项，办法是在文件头部添加一个单元格，将其类型转换为 `raw`(原样使用的单元格)，并在其中添加需要的 YAML 内容，如：

```
---
title: 测试 Python 和 Quarto
author: XXX
date: 2022-11-27
---
```

Jupyter 笔记本不自动运行其中的程序 (Python、Julia、R 等)，转换前应确保笔记本中的程序结果是最新的，可以在转换前重新运行一遍笔记本，还有一种办法是在 `quarto` 命令后面加`--execute` 选项。

对于.qmd 格式的文件，如 `mydoc.qmd`，在操作系统命令行界面中用如下命令转换为 Jupyter 笔记本：

```
quarto convert mydoc.qmd
```

有些程序计算运行时间很长，可以安装 Jupyter 笔记本的缓存功能模块 `jupyter-cache`，使用缓存。见https://quarto.org/docs/computations/python.html。

22.5.4 编译管理

在 RStudio 中可以用 `Preview Book` 或 `Render Book` 将整本书编译为输出格式。当项目较大或者其中的程序运行时间较长时，可以应用一些技巧减少花费的时间。

22.5.4.1 增量转换

用 `quarto` 命令转换时，可以仅指定一个文件或一个子目录进行转换。这时，项目中的其它文件使用原来已经转换过的结果。

在 RStudio 中，可以用 `Preview Book` 打开一个动态更新的预览窗口，这样，在用 `Render` 重新转换单个文件时可以自动更新结果。如果选择了 “Knit on Save” 也可以在保存文件时自动更新结果。

22.5.4.2 冻结版本

可以使用 YAML 选项 `execute -- freeze: true` 要求在编译整个项目时，不运行文件中的源程序。将 `true` 改为 `auto`，则意味着仅在源文件改变时才运行其中的源程序。而编译单个文件或单个子目录时还是会运行文件中的源程序。

使用冻结功能要谨慎，这个功能与使用缓存类似，可能会使得输出结果使用旧版本的数据或程序。这些冻结结果保存在 `_freeze` 子目录中，可以删除这个子目录，强迫重新计算结果。

22.5.4.3 缓存功能

可以用 YAML 选项 `execute -- cache: true` 设置每个代码段结果缓存，在代码段中可以用 `#| cache: true` 对特定代码段设置缓存。当代码段的源代码被修改时，缓存会自动更新，但是有可能有一些不能察觉的依赖内容改变，这时缓存并不会自动更新，对单个文件，可以用 quarto 命令如下刷新缓存：

```
quarto render file1.qmd --cache-refresh
```

对于整个项目，刷新缓存命令为

```
quarto render --cache-refresh
```

22.6 插入 R 程序代码

如果仅是插入 R 程序代码，而不需执行，只要写成`` ```r ... ``` ``的格式，以`` ```r ``行开头，以`` ``` ``行结尾。这会自动对代码进行彩色加亮显示。

为了执行，以`` ```{r} ``行开头，以`` ``` ``行结尾。在 RStudio 中，可以用 “Ctrl+Alt+I” 快捷键插入代码段。在代码内容中可以用 “`#|`” 开头的特殊注释作为代码段选项，“`#| label: #mylabel`” 用来给代码段一个可引用标签，可引用标签需要在整本书（整篇文章）中都唯一。如：

```
```{r}
#| label: myaddex
1+2
```
```

在 “`#|`” 选项中使用 YAML 格式，如 “`#| warning: false`”。为了在 “`#|`” 选项中使用 R 代码，使用 “`!expr` 代码” 的格式，如 “`#| warning: !expr preview`”，其中 “`preview`” 是在前面的 R 代码段中已定义的逻辑变量。

在 RStudio 软件中使用时，这些 R 代码块都可以点击代码块右上角的 “Run current chunk” 图标（向右的三角形图标）执行代码块，并将代码块输出结果直接显示在文档内部，代码块下方。

还有一种用法是在段内使用 R 代码，称为行内代码，行内代码的结果插入到一个段落中间，代码以`` `r ``开头，以`` ` ``结尾，如`` `r sin(pi/2)` ``在结果中会显示为 1。为了原样显示一个反向单撇号，可以在两边用双反向单撇号界定并用空格隔开内部的内容。

22.7 代码段选项

22.7.1 标签

在代码段内用“`#| label: mylabel`”添加一个标签，在 RStudio 中编辑时可以很容易地按标签导航到某个代码段，代码段如果生成图形，可以按标签（图形标签以 `fig-`开头）在别处交叉引用。在使用缓存功能时，可以按代码段标签表示代码段之间的依赖关系。

标签可以包含字母、数字、减号和下划线，生成图形的代码段标签必须以 `fig-`开头。标签必须在文件中唯一，如果是多个源文件的图书、网站项目，标签在不同文件之间也必须唯一。

22.7.2 代码段执行选项

代码段可以设置各种执行选项。可以设置在整个项目的 `_quarto.qmd` 文件或单个 qmd 文件的顶层 `execute` 下面，如：

```yaml
execute:
  echo: false
  warning: false
  message: false
```

也可以设置在单个代码段中，如 `#| echo: false`，这时不需要 `execute` 这个父节点。

可用选项：

- `eval`: true 或 false，表示是否执行;
- `echo`: true 或 false，表示是否显示程序代码;
- `output`: true, false 或 asis，表示显示为经整理的格式、不显示、不进行整理地显示;
- `fig-show`: 选 `false` 表示不显示生成的图形。
- `message`, `warning`: true 或 false，表示是否显示提示或警告信息;
- `error`: 选 `true` 表示在出错时仍继续运行，缺省是 `false`。
- `include`: true 或 false, 表示代码与结果是否都包含，或都不包含，但不影响程序代码执行。

在用 `library()` 载入扩展包时，经常有大量的版本信息、函数名冲突信息，这时只要设置 `output: false` 就可以抑制这些信息显示到排版结果中。

22.8 插图

22.8.1 插图方法

在.qmd 文件中插图，使用本地文件，仍可以用 markdown 语法，即“`![代替显示文字](图片地址)`”，如“`![百度](figs/baidu01.png)`”

在使用 RStudio 编辑文件时，可以打开可视化编辑模式 (Visual 模式)，选快捷图标 Figure/Image，选择要插入的图片，并选择一些选项。也可以将本项目目录或子目录内的图形文件直接拖曳到要插入的位置。还可以复制图片，然后在可视化编辑模式中的文档中粘贴图片，这会在项目文件夹中生成图片的副本。

可以使用 R 程序插图，如：

```
```{r}
#| label: fig-mytestimg
#| fig-cap: "百度图标"
#| echo: false
knitr::include_graphics("figs/baidu01.png")
```
```

22.8.2 插图有关代码段选项

可以用选项修改插入的图形属性。这些选项适用于用代码段程序生成或者插入的图形。

代码段中设置“`#| fig-cap: 图形说明`”和“`#| label: fig-mylabel`”后自动生成图形编号，生成的图形文件名会利用标签，并可用“`[@fig-mylabel]`”格式引用图形，显示如“图 2.1”。如果仅需要生成编号数字，可以写成“`[-@fig-mylabel]`”。设置 `fig-cap` 和 `lable` 后使得图形变成“浮动”图形，在 PDF 输出中可以调整出现的位置。

图形大小有默认值。可以在源文件或者整个项目中每个输出格式的设置下面，用 `fig-width` 和 `fig-height` 设置图形的英寸为单位的宽度和高度，也可以作为代码段选项设置。

代码段关于图形大小、分辨率的选项包括：

- `fig-width`, `fig-height`: 以英寸为单位的图形宽度、高度。
- `fig-asp`：高宽比，如 `fig-asp: 0.75`。这样可以仅规定 `fig-width`，用 `fig-asp` 推算高度。
- `out-width`: 图片占实际行宽的比例，如 `out-width: 80%`。同时可以设置 `fig-align: center` 使得图片在行宽范围内居中放置。如果有并排放置的图片，可以用 `out-width: 45%`，并设置 `fig-align: default`。
- `fig-format`: 可设置为 `png`。当输出为 PDF 格式时程序生成的图形文件默认使用 `pdf` 格式，PDF 格式是矢量格式，具有放大不失真的优点，但是如果图形中数据量很大会造成文件过大。

`out-width` 设置了显示宽度，而 `fig-width` 实际会影响到图片分辨率。在用程序生成的图形中包含较多文字内容时，可以设置较大的 `fig-width` 使得实际分辨率较高，这样文字内容不至于混在一起。

22.9 插入表格

仍可以用 markdown 的方法插入表格，参见 §21.3.10。在表格的标题行末尾空格后添加如“`{#tbl-mytable}`”这样的标签，就可以产生可交叉引用的编号表格，用“`[@tbl-mytable]`”这样的格式引用，显示如“表 2.1”。如果仅需要产生引用数字，可用格式“`[-@tbl-mytable]`”。

如果使用 RStudio 编辑，可以打开可视化编辑模式 (Visual 模式)，选择插入表格，并用可见即所得方式输入表格。

可以将表格输入为 R 数据框，然后用 `knitr::kable()` 函数插入表格。在插入表格的代码段中，定义 `label` 选项为一个标签，以 `tbl-`开头，如 `tbl-mytable`，定义 `tbl-cap` 选项为表格要显示的说明文字，就可以生成可交叉引用的表格，引用格式如 `[@tbl-mytable]`，并且在 PDF 输出中是浮动表格。

插入的代码段例子如：

```
```{r, echo=FALSE}
#| label: tbl-mytable
#| tbl-cap: "汽车数据"
knitr::kable(mccars[1:10, 1:5])
```
```

`knitr::kable()` 函数本身有一些选项，如 `digits` 规定小数点后的位数。

支持并排放置的表格，详见 Quarto 的文档。

gt是一个制表用的 R 扩展包，支持从数据框转换生成比较复杂的表格，支持 HTML、LaTeX、RTF 输出。

22.10 数学公式

22.10.1 在 Markdown 中输入数学公式

原始的 Markdown 格式并不支持数学公式。Pandoc 扩展的 markdown 格式提供了对数学公式的支持，可以在 Markdown 文件中插入 LaTeX 格式的数学公式。虽然不能提供所有的 LaTeX 公式能力，但是常用的数学公式还是能做得很好，转换到 HTML、docx 都可以得到正常显示的公式。

用 RStudio 软件编译 Markdown 文件，可以在其中插入 LaTeX 格式的数学公式，数学公式可以在编辑器内部显示预览，编译成 HTML 或者 docx 格式后都可以正常显示数学公式, HTML 结果可以直接利用 RStudio 内部的浏览器预览，在另外安装的 LaTeX 编译器的支持下也可以将.qmd 格式编译 LaTeX 格式然后再转换为 PDF 格式，这种基于 LaTeX 的方法对数学公式的支持会更完善。

`.qmd` 和`.ipynb` 文件在 Quarto 软件支持下都可以使用 LaTeX 数学公式。

22.10.2 数学公式类别

数学公式公式分为行内公式和独立公式。行内公式和段落的文字混排，写在两个美元符号 `$` 中间，或者`\(`和`\)` 之间。例如 `$f(x)=\frac{1}{2} \int_0^1 \sin^2 (t x) dt$` 变成 $f(x) = \frac{1}{2}\int_0^1 \sin^2(tx)dt$。开头的 `$` 后面不能紧跟着空格，结尾的 `$` 不能紧跟在空格后面。

独立公式写在成对的美元符号中间，或者`\[`和`\]` 之间。例如：

```
$$
  f(x) = \frac{1}{2} \sum_{j=1}^\infty \int_0^1 \sin^2(j t x) dt .
$$
```

显示为

$$f(x) = \frac{1}{2} \sum_{j=1}^\infty \int_0^1 \sin^2(jtx) dt.$$

22.10.3 基本功能

在数学公式中，用下划线表示下标，比如 `$x_1$` 结果为 x_1。用 `^` 表示上标，如 `$x^2$` 结果为 x^2。上下标都有，如 `$x_1^2$` 结果为 x_1^2。

公式中用大括号表示一个整体，比如 `$x^(1)$` 不能得到 $x^{(1)}$，需要用 `$x^{(1)}$`。

公式中用反斜杠开始一个命令，命令仅包含字母而不能包含数字，数字只能作为参数。如 `$\frac{1}{2}$` 和 `$\frac12$` 都可以生成 $\frac{1}{2}$。类似的命令如 `$\sqrt{2}$` 为 $\sqrt{2}$。

希腊字母都有对应的命令，如 `$\alpha$` 为 α，`$\Sigma$` 为 Σ。

常用数学函数有自己的命令，如 `$\sin x$` 变成 $\sin x$，`$\exp (x)$` 为 $\exp(x)$。

圆括号与方括号可以直接使用，绝对值符号用 `|`，如 `$|x|$` 变成 $|x|$。但是，大括号必须写成`\{`和`\}` 的形式，如 `$\exp\{-\frac12 x^2\}$` 变成 $\exp\{-\frac{1}{2}x^2\}$。

为了使得括号能够与括号内部的内容等高，可以用`\left` 和`\right` 命令进行修饰，如

```
$$
\phi(x) = \frac{1}{\sqrt{2\pi}} \exp\left\{ \frac{1}{2} x^2 \right\}
$$
```

变成

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \exp\left\{ \frac{1}{2} x^2 \right\}$$

注意`\left` 与`\right` 必须成对使用，否则出错。

求和如 `$\sum_{i=1}^n x_i$` 变成 $\sum_{i=1}^n x_i$。乘积如 `$\prod_{i=1}^n x_i$` 变成 $\prod_{i=1}^n x_i$。积分如 `$\int_0^1 f(x) dx$` 变成 $\int_0^1 f(x)dx$。

22.10.4 修饰符

`$f'(x)$` 显示为 $f'(x)$，表示导数；`$f''(x)$` 显示为 $f''(x)$，表示二阶导数。顺便说一句，偏导数写法如 `$\frac{\partial f(x,t)}{\partial x}$`，显示为 $\frac{\partial f(x,t)}{\partial x}$。

$\bar{x}$ 的写法是 `$\bar{x}$`。$\overline{\text{span}}$ 的写法是 `$\overline{\text{span}}$`。$\hat{x}$ 的写法是 `$\hat{x}$`。$\tilde{x}$ 的写法是 `$\tilde{x}$`。$\vec{x}$ 的写法是 `$\vec{x}$`。

22.10.5 对齐与矩阵

为了产生对齐的公式，在独立公式中使用 aligned 环境。公式中的环境以`\begin{环境名}` 开始，以`\end{环境名}` 结束，用`\\`表示换行，用 `&` 表示一个上下对齐位置。如

```
$$\begin{aligned}
  f(x) =& \sum_{k=0}^\infty \frac{1}{k!} x^k \\
  =& e^x
\end{aligned}$$
```

转化成

$$\begin{aligned}
f(x) =& \sum_{k=0}^\infty \frac{1}{k!} x^k \\
=& e^x
\end{aligned}$$

可以用 pmatrix 环境制作写在圆括号中的矩阵，用 bmatrix 环境制作写在方括号中的矩阵，用 vmatrix 制作写在绝对值号中的矩阵，如

```
$$
\begin{pmatrix}
x_{11} & x_{12} \\
x_{21} & x_{22}
\end{pmatrix}
$$
```

变成

$$\begin{pmatrix}
x_{11} & x_{12} \\
x_{21} & x_{22}
\end{pmatrix}$$

列向量、行向量也可以用这种办法制作。

22.10.6 特殊字体

数学公式中的单词、文字必须用`\text{...}` 保护，比如

```
$$
\text{CV} = \frac{S}{\bar x} \times 100 \%
$$
```

变成

$$\text{CV} = \frac{S}{\bar x} \times 100 \%$$

上例中如果不用`\text{}` 保护，就会显示为

$$CV = \frac{S}{\bar{x}} \times 100\%$$

这里的 CV 通常理解为 C 乘以 V。

有时用粗体表示向量或者矩阵，用`\boldsymbol{...}` 说明。如

```
$$
\boldsymbol{v} = (v_1, v_2)^T
$$
```

变成

$$\boldsymbol{v} = (v_1, v_2)^T$$

美术体英文字母用`\mathcal{...}`，如 $\mathcal{A}, \mathcal{B}, \mathcal{C}$ 写法为 `$\mathcal{A}, \mathcal{B}, \mathcal{C}$`。

手写花体字母用`\mathscr{...}`，如 $\mathscr{B}, \mathscr{C}, \mathscr{F}, \mathscr{G}$ 写法为 `$\mathscr{B}, \mathscr{C}, \mathscr{F}, \mathscr{G}$`。

空心字体用`\mathbb{...}`，如 $\mathbb{R}, \mathbb{C}$ 写法为 `$\mathbb{R}, \mathbb{C}$`

22.11 设置

22.11.1 使用单个源文件

只有一个源文件时，所有的设置都放在文件头部的 YAML 块中，如：

```yaml
---
title: " 作业框架"
author: " 李东风"
date: "2023-04-26"
lang: zh
format:
  html:
    toc: true
    toc-location: body
    toc-depth: 3
    number-sections: true
    html-math-method: katex
  docx:
    toc: true
    toc-depth: 3
```

```
    number-sections: true
  pdf:
    documentclass: article
    toc: true
    toc-depth: 3
    include-in-header:
      text: |
        \usepackage{ctex}
        \usepackage{amsthm,mathrsfs}
---
```

22.11.2 生成图书

如果用多个源文件同时构成一本书或网站，这时应该在项目目录中添加一个“`_quarto.yml`”文件，将关于书（网站）的设置都放在这个文件中。

为了用 Quarto 格式写书，应该在 RStudio 中用新建项目方式自动生成新文件夹，并选择 Quarto book 格式。

会自动生成文件夹和主要文件框架，其中 `index.qmd` 是主要入口，`_quarto.yml` 是主要的设置文件。`_quarto.yml` 中会包括如下设置：

```
project:
  type: book
```

这说明整个项目的目标是生成一本书，书的输出结果一般包括网站形式和 PDF 形式。

仍用 `title`, `author`, `date` 给出书名、作者、日期，如：

```
title: " 统计计算"
author: " 李东风"
date: "2022-11-28"
```

其中日期如果想动态填入编译时的日期，可以用：

```
title: " 统计计算"
author: " 李东风"
date: !r lubridate::today()
```

22.11.3 图书章节结构

章节结构最简单的情况是仅有若干章，这时在 `_quarto.qml` 中，写在 `book:` 下面如下的内容：

```yaml
book:
  chapters:
    - index.qmd
    - intro.qmd
    - summary.qmd
    - references.qmd
```

如果有部分，有附录，结构如：

```yaml
book:
  chapters:
    - index.qmd
    - preface.qmd
    - part: " 基础知识"
      chapters:
        - elem.qmd
        - linear.qmd
    - part: " 进阶知识"
      chapters:
        - nonlin.qmd
        - solve.qmd
    - appendices
      - summary.qmd
      - references.qmd
```

注意“部分”下面的章需要用 `chapters` 作为父节点，而 `appendices` 下面则直接列出各个附录文件。

22.11.4 多种输出格式

在单个源文件或者图书的 `_quarto.yml` 文件中，可以设置 `format` 项，在其下层列出要生成的输出格式类型，以及专用于该输出格式的选项。比如在 `_quarto.yml` 中：

```yaml
format:
  html:
    toc-float: true
    html-math-method: katex
  pdf:
    documentclass: scrbook
  docx: default
```

其中 `default` 表示使用默认设置，但是不能仅列出输出格式而没有任何设置。

22.11.5 目录

在 `_quarto.qml` 中用 `toc: true` 指定对所有输出格式都自动生成目录。用 `toc-depth: 3` 指定目录中显示内容的层级，3 是显示到小节层次。对于 HTML 输出，用 `toc-location: right` 指定目录单独放在正文右侧，用 `toc-location: body` 指定目录单独放在正文内部开头处。

在 `_quarto.qml` 中用 `number-sections: true` 使得所有输出格式都有章节编号数字。不需要编号的章节，在章节标题内容后面空格后写 `{.unnumbered}`。

22.11.6 中文支持

中文内容，需要指定在 `_quarto.qml` 中指定 `lang: zh`。默认的中文翻译有些不尽合理，可以从https://github.com/quarto-dev/quarto-cli/tree/main/src/resources/language/下载 `_language-zh.yml`，修改成需要的样子，改名为 `_language.yml`，放在自己的项目目录中。这时不需要使用 `lang: zh` 选项。

为了使得 LaTeX 输出的 PDF 支持中文，必须在 YAML 的 `format--pdf--include-in-header` 中指定`\usepakage{ctex}`，如：

```yaml
format:
  pdf:
    documentclass: scrbook
    include-in-header:
      text: |
        \usepackage{ctex}
        \usepackage{amsthm,mathrsfs}
```

注意其中的 `text:` 项，这用来指定具体的插入内容，而不是插入文件。也可以将这些 LaTeX 头文件中需要的内容写在一个 `preamble.tex` 文件中，这时就可以写成 `include-in-header: preamble.tex`。

另外一种办法是直接指定使用的中文字体，这种方法因为仅指定一种字体，效果不太美观。在安装了 LaTeX 软件如 TinyTeX 后，在 Windows 的命令行窗口 (cmd)，运行如下命令：

```
fc-list :lang=zh > tmp.txt
```

则产生一个 `tmp.txt` 文件，其中包含了可用字体的列表，比如 `NSimSun` 代表“新宋体”。在 YAML 中设置 `format -- pdf -- mainfont: "NSimSun"` 可以使得通过 LaTeX 生成的 PDF 支持中文内容。

22.11.7 图、表、交叉引用定制

在引用图、表、公式、章节时，默认的引用格式不一定合适，可以用上述修改 `_language.yml` 的方法，也可以在 `_quarto.yml` 中进行设置，如：

```yaml
crossref:
  eq-prefix: ""
  sec-prefix: "§"
```

这里 eq-prefix 设置了在引用公式时，(@eq-mylabel) 的编译结果如 (1.1)，而不是 (公式 1.1)。sec-prefix 使得在引用某一节时，使用适当的格式如 “§1.1”。

22.11.8 参考文献数据库设置

在在 _quarto.yml 中可以设置一个或多个文献数据库，会自动生成参考文献列表，如：

```yaml
bibliography: ["myref1.bib", "myref2.bib"]
```

这些.bib 文件目前需要放在项目目录中，不支持用绝对路径或相对路径。

如果使用 RStudio 对 Quarto 源文件的可视化编辑模式，还支持利用 DOI 等方法加入参考文献引用，这时 RStudio 自动添加一个名为 “bibliography.bib”，将这些引用保存在里面。

关于.bib 文件的格式，可以参考一些讲 LaTeX 用法的书。一个典型的.bib 文件如：

```bibtex
@Book{MWP06-HighStat,
  author    = {茆诗松 and 王静龙 and 濮晓龙},
  title     = {高等数理统计},
  year      = {2006},
  edition   = {第二版},
  publisher = {高等教育出版社}
}
@ARTICLE{Ansley90,
  author = {Ansley, C.F. and Kohn, R.},
  title = {Filtering and smoothing in state space
  models with partially diffuse
    initial conditions},
  journal = {J. Time Series Analysis},
  year = {1990},
  volume = {11},
  pages = {275--93}
}
```

引用方法如 “[@Ansley90]”。多个文献引用如 “[参见 @Ansly90; @MWP06-HighStat pp 101-103]”。注意 “@文献标签” 前后不允许直接使用汉字和中文标点。

22.11.9 其它选项

22.11.9.1 编辑模式选择

为了在 RStudio 中设置总是使用源代码编辑模式，可以在 `_quarto.yml` 中加入：

```
editor: source
```

22.11.9.2 HTML 自包含

生成 HTML 结果时，会有图片、CSS 等文件伴随 HTML 文件，如果需要放在网站，可以将这些文件同时放在网站上。如果是单个源文件，希望生成的 HTML 文件不要依赖于外部文件，可以在源文件的 YAML 内容中添加如下内容：

```
format:
  html:
    embed-resources: true
```

22.12 定理类

定理:使用`:::{#thm-label} ... :::`的方案,其中`...`可以使用各种内容段落。定理自动在章内编号,引用如参见`[@thm-label]`，显示如“参见定理 2.1”。如果引用时仅希望生成引用数字，可用如“参见定理 `[-@thm-label]`”。

可以加一个 `name="..."` 定理名称。因为不同章在不同文件中，但是标签是统一的，所以不同的章也需要使用不同的标签。

定理的标题可以用二级标题在内容中表示。例如：

```
:::{#thm-lebesgue01 name=" 勒贝格定理"}
一元函数黎曼可积，
当且仅当其不连续点的集合为零测集。
:::

引用如：见[@thm-lebesgue01]。
```

表 22.1: 定理类段落

环境名	默认显示名	标签前缀
theorem	Theorem	#thm-
lemma	Lemma	#lem-

环境名	默认显示名	标签前缀
corollary	Corollary	#cor-
proposition	Proposition	#prp-
conjecture	Conjecture	#cnj-
definition	Definition	#def-
example	Example	#exm-
exercise	Exercise	#exr-

另外还提供了 proof, remark, solution 环境，这些环境不提供编号和交叉索引功能，没有必要使用。使用的格式如`::: {.proof} ... :::`。

22.13 引用

22.13.1 引用章节

为了引用章节，在章节标题空格后写 `{#sec-label}`，其中 `label` 是全书都唯一的标签。引用如 “参见 `[@sec-label]`”。如果仅希望生成引用数字，可用如“参见第 `[-@sec-label]` 章”。

22.13.2 可引用插图和表格

可引用插图示例：

```
#| label: fig-test-fig01
#| fig-cap: " 测试图形编号 "

plot(1:10)
```

参见`[@fig-test-fig01]`。

注意代码开始注释不要忘记井号后面的竖线符号。`label` 的内容必须以 `fig-`开头。后面的选项是 `fig-cap` 而不是 `fig-caption`。

22.13.3 数学公式引用

目前不支持多行公式中的多个编号。在独立公式的 `$$` 后面，用 `{#eq-mylabel}` 的方式加公式标签，引用时用 “`(@eq-mylabel)`” 格式引用。应在 YAML 中加选项：

```
crossref:
  eq-prefix: ""
```

使得公式引用仅包含编号，如 (2.1)，否则可能会变成 (方程式 2.1)。也将引用写成如“(-@eq-mylabel)”。

22.14 代码段缓存

R 代码段支持缓存，使得已经花比较长时间运行得到的结果不必每次都重复运行程序。为了所有文件中的 R 代码段都缓存，可以在 _quarto.qmd 中顶层设置“execute -- cache”为“true”，如：

```
execute:
  cache: true
```

为了使得某一段可缓存，可以在代码段开头设置“#| cache: true”。

设置了缓存的代码段，其运行结果会被保存下来，仅该代码段的程序被修改时才重新运行。但是，如果 A 代码段依赖于前面的 B 代码段的结果，而 B 代码段被修改了，则 A 也应该重新运行，这并不能自动识别。这可以靠 dependson 选项和 cache-extra 选项建立依赖关系。

例如，代码段 B 读入原始数据：

```
```{r}
#| label: code-B
da0 <- read.csv("myinput.csv")
```
```

代码段 A 处理读入的数据产生建模用数据，使用缓存保存生成的数据，用 dependson 选项令其依赖于代码段 B：

```
```{r}
#| label: code-A
#| cache: true
#| dependson: code-B
da <- da0[,c("weight", "height")]
```
```

这样，当代码段 B 的程序被修改时，代码段 A 也会重新运行。但是，如果代码段 B 输入的原始数据“myinput.csv”的内容被修改了，这里还是无法识别这种修改，代码段 A 不会重新运行，导致数据 da 是过时的版本。这时，可以在代码段 B 中增加 cache-extra 选项，指定当输入文件的信息改变时就对依赖于代码 B 的代码段发出缓存更新提示。如：

```
```{r}
#| label: code-B
#| cache-extra: file-info("myinput.csv")
da0 <- read.csv("myinput.csv")
```
```

人为地管理这些依赖关系是比较繁琐易错的，所以不必要时不要使用这种缓存的办法。当怀疑缓存陈旧时，可以在 R 中运行如下命令清除缓存：

```
knitr::clean_cache()
```

22.15 制作演示文稿

Quarto 格式可以转换为演示文稿，支持如下形式的演示文稿：

- revealjs: 这是网页格式的演示文稿，功能比较强；
- pptx: 这是 MS Power Point 格式的演示文稿；
- beamer: 这是 LaTeX 的 Beamer 演示文稿。

演示文稿分为多个帧 (frame)，每帧用二级标题作为标志并以其为标题。用一级标题作为单独的分节帧，将单独显示在一帧中。

帧也可以没有标题，比如仅有照片的帧，这时，用三个或三个以上的减号连在一起标识新帧的开始。

演示文稿中的文字内容常用列表格式，也支持 R 程序代码段。

一个简单的 revealjs 格式演示文稿源文件 example-rj.qmd，内容如：

```
---
title: "Quarto revealjs 演示文稿样例"
author: " 李东风"
date: "2022-12-31"
format: revealjs
---
```

```
# 用 Quarto 的 revealjs 格式输出制作演示

## 幻灯片结构

- 用二级标题标志一个页面开始
- 用一级标题制作单独的分节页面
- 用三个或三个以上减号标志没有标题的页面开始
```

```
- 每个页面一般用 markdown 列表显示若干个项目

## 幻灯片编辑和编译

- 用 RStudio 编辑内容
- 用 RStudio 的 Render 按钮生成结果

-----

![一个演示画面](figs/bd_logo.png)


## 数学公式

$$\begin{aligned}
  \sum_{k=1}^n k^2 =& \frac{1}{6} n (n+1) (2n+1) , \\
  \sum_{k=1}^n k^3 =& \left(\frac{1}{2} n (n+1) \right)^2 .
\end{aligned}$$


## 代码段和结果

```{r}
x <- 1:10
y <- x^2
plot(x, y)
```
```

在播放时，可以用向下或向右方向键或空格键翻到下一帧，用向上或向左方向键退回到上一帧，可以点击左下角的图表展开目录或者菜单，可以用 F 键切换成全屏显示，在全屏显示时用 Esc 键退回到正常显示。

如果需要一帧中的列表逐项显示（每按一次向下光标键显示一项），可以写成如下格式：

```
## 幻灯片结构

::: {.incremental}

- 用二级标题标志一个页面开始
- 用一级标题制作单独的分节页面
- 用三个或三个以上减号标志没有标题的页面开始
```

```
- 每个页面一般用 markdown 列表显示若干个项目


:::
```

如果希望每一帧的列表都逐项显示，可以在源文件开头的 YAML 设置中，增加：

```
format:
  revealjs:
    incremental: true
```

为了增加 PowerPoint 格式的输出，可以将 YAML 的 format 部分改为：

```
format:
  revealjs: default
  pptx: default
```

22.16 运行 Julia 程序

Qmd 文件中除了 R 代码段以外，还可以插入 Rcpp、Python、Julia、SQL 等许多编程语言的代码段，常用编程语言还可以与 R 代码段进行信息交换。在 Jupyter 软件支持下直接支持将.ipynb 格式的文件转换为各种输出格式。

为了在.qmd 文件中运行 Julia 程序并使用其结果，需要在操作系统 PATH 环境变量中正确设置所用 Julia 可执行的路径。

22.17 用作科学计算笔记本

Quarto 的.qmd 格式与 Jupyter 笔记本格式，都很适用于当作一个科学研究时的计算分析笔记本使用。这个笔记本可以记录自己的想法，研究过程，成功与失败，其中的计算，包括计算程序和结果。

当作计算分析笔记本时，有如下建议：

- 应该包括一个有意义的标题，并使用有意义的文件名保存。
- 应该用日期属性表明写作日期或者最后修改日期。
- 内容的第一段应该对研究进行简要说明。
- 研究中失败的研究内容也不要删除，而是说明原因，避免自己和读者再次犯类似错误。
- 一般不要将数据包含在笔记本源文件中，仅在数据量很小时，可以写在代码段中。
- 在一个研究项目中，如果数据有错误，一般不应该直接修改数据文件，而是在笔记本源文件中用程序修改并注明。
- 使用一个笔记本源文件连续进行研究时，最好每天结束时清除缓存，用 render 功能生成一个版本。

- 持续较长时间的研究，可能会受到 R 扩展包版本变化的影响。较正规的解决思路是使用 renv 扩展包，将需要的扩展包变成项目私有扩展包。
- 因为 Quarto 很好用，你的多个研究项目都可以使用，随着时间累积出许多笔记本源文件。最好将这些研究用“项目”表达，并使用合适的文件夹名，进行合理的层次组织。

22.18 问题

- 参考文献只能放在项目文件夹而不能使用相对或绝对路径。
- 在定理环境中，使用列表会使得定理本身进入列表。
- RStudio 中设置为 visual 编辑方式后，会自动调整 qmd 文件的空白，使得切换回 sourse 编辑方式后内容编排很不习惯。这主要是有数学公式时不方便，如果仅文字内容倒是没有关系。

22.19 附录：TinyTex 的安装使用

22.19.1 基于 Quarto 可执行程序

如果不使用 Rmarkdown、bookdown，仅使用 Quarto 格式，而且安装了单独的 Quarto 可执行程序，可以在操作系统命令行运行如下安装 TinyTeX 的命令：

```
quarto install tool tinytex
```

这种安装方式不更新操作系统的 PATH 环境变量，仅供 Quarto 本身使用，从而不会影响到操作系统中已经安装的其它 LaTeX 编译环境。Quarto 在编译 PDF 时遇到缺少的格式文件可以自动下载安装。

22.19.2 基于 RStudio

如果要在 rmarkdown、bookdown 中使用 PDF 输出功能，可以在在 R 中安装 tinytex 扩展包并安装 TinyTeX 编译软件：

```
install.packages('tinytex')
tinytex::tlmgr_repo('http://mirrors.tuna.tsinghua.edu.cn/CTAN/')
tinytex::install_tinytex()
```

其中上面第一行命令安装 R 的 tinytex 扩展包，第二行将下载 LaTeX 编译程序的服务器设置为清华大学 tuna 镜像站，第三行安装 LaTeX 编译程序。

如果安装成功,TinyTeX 软件包在 MS Windows 系统中一般会安装在 `C:\Users\用户名\AppData\Roaming\TinyTeX` 目录中，其中“用户名”应替换成系统当前用户名。如果需要删除 TinyTeX 软件包，只要直接删除那个子目录就可以。

为了判断 TinyTeX 是否安装成功，在 RStudio 中运行

```
tinytex::is_tinytex()
```

结果应为 `TRUE`, 出错或者结果为 `FALSE` 都说明安装不成功。

当用户使用 RMarkdown 和 tinytex 包转换 latex 并编译为 PDF 时，如果缺少某些 latex 宏包，tinytex 会自动安装缺少的宏包。

Part V

数据整理与汇总

Chapter 23

数据整理

23.1 tidyverse 系统

tidyverse 是一系列用于数据输入输出、数据整理和数据汇总的 R 扩展包集合，使用这些包遵循相近的编程风格，比直接使用基本 R 编程要更直观、容易理解。其中 readr 包用于读入数据，tidyr 包用于进行长、宽表转换，dplyr 包用于数据整理与汇总，purr 包进行 map-reduce 类操作，等等。

假设数据以 tibble 格式保存。数据集如果用于统计与绘图，需要满足一定的格式要求，(Wickham, 2014) 称之为整洁数据 (tidy data)，基本要求为：

- 每行一个观测，代表一个样本点（个体），每列一个变量，代表个体的一个属性，每个单元格（每行、每列交叉的位置）恰好有一个数据值。
- 这些变量应该是不同的属性，而不是同一属性在不同年、月等时间的值分别放到单独的列。

数据集经常需要选行子集、选列子集、排序、定义新变量、横向合并、长宽转换等操作，而且经常会用若干个连续的操作分步处理，R 的管道运算符 `|>` 和 magrittr 包的`%>%` 特别适用于这种分步处理。

dplyr 包和 tidyr 包定义了一系列 “动词”，可以用比较自然的方式进行数据整理。较复杂的分组操作还可以利用 purrr 包的 `map` 类函数。

为了使用这些功能，可以载入 `tidyverse` 包，则 magrittr 包，readr 包，dplyr 包和 tidyr 包都会被自动载入:

```
library(tidyverse)
```

下面的例子中用如下的一个班的学生数据作为例子，保存在如下`data/class.csv`文件中：

```
name,sex,age,height,weight
Alice,F,13,56.5,84
Becka,F,13,65.3,98
```

```
Gail,F,14,64.3,90
Karen,F,12,56.3,77
Kathy,F,12,59.8,84.5
Mary,F,15,66.5,112
Sandy,F,11,51.3,50.5
Sharon,F,15,62.5,112.5
Tammy,F,14,62.8,102.5
Alfred,M,14,69,112.5
Duke,M,14,63.5,102.5
Guido,M,15,67,133
James,M,12,57.3,83
Jeffrey,M,13,62.5,84
John,M,12,59,99.5
Philip,M,16,72,150
Robert,M,12,64.8,128
Thomas,M,11,57.5,85
William,M,15,66.5,112
```

读入为 tibble:

```
d.class <- read_csv(
  "data/class.csv",
  col_types=cols(
  .default = col_double(),
  name=col_character(),
  sex=col_factor(levels=c("M", "F"))
))
```

这个数据框有 19 个观测，有如下 5 个变量：

- name
- sex
- age
- height
- weight

另一个例子数据集是 R 的 NHANES 扩展包提供的 NHANES，这是一个规模更大的示例数据框，可以看作是美国扣除住院病人以外的人群的一个随机样本，有 10000 个观测，有 76 个变量，主题是个人的健康与营养方面的信息。仅作为教学使用而不足以作为严谨的科研用数据。原始数据的情况详见http://www.cdc.gov/nchs/nhanes.htm。载入 NHANES 数据框：

```
library(NHANES)
data(NHANES)
```

```
print(dim(NHANES))
```

```
## [1] 10000    76
```

```
print(names(NHANES))
```

```
##  [1] "ID"               "SurveyYr"         "Gender"           "Age"
##  [5] "AgeDecade"        "AgeMonths"        "Race1"            "Race3"
##  [9] "Education"        "MaritalStatus"    "HHIncome"         "HHIncomeMid"
## [13] "Poverty"          "HomeRooms"        "HomeOwn"          "Work"
## [17] "Weight"           "Length"           "HeadCirc"         "Height"
## [21] "BMI"              "BMICatUnder20yrs" "BMI_WHO"          "Pulse"
## [25] "BPSysAve"         "BPDiaAve"         "BPSys1"           "BPDia1"
## [29] "BPSys2"           "BPDia2"           "BPSys3"           "BPDia3"
## [33] "Testosterone"     "DirectChol"       "TotChol"          "UrineVol1"
## [37] "UrineFlow1"       "UrineVol2"        "UrineFlow2"       "Diabetes"
## [41] "DiabetesAge"      "HealthGen"        "DaysPhysHlthBad"  "DaysMentHlthBad"
## [45] "LittleInterest"   "Depressed"        "nPregnancies"     "nBabies"
## [49] "Age1stBaby"       "SleepHrsNight"    "SleepTrouble"     "PhysActive"
## [53] "PhysActiveDays"   "TVHrsDay"         "CompHrsDay"       "TVHrsDayChild"
## [57] "CompHrsDayChild"  "Alcohol12PlusYr"  "AlcoholDay"       "AlcoholYear"
## [61] "SmokeNow"         "Smoke100"         "Smoke100n"        "SmokeAge"
## [65] "Marijuana"        "AgeFirstMarij"    "RegularMarij"     "AgeRegMarij"
## [69] "HardDrugs"        "SexEver"          "SexAge"           "SexNumPartnLife"
## [73] "SexNumPartYear"   "SameSex"          "SexOrientation"   "PregnantNow"
```

变量 ID 是受试者编号，SurveyYr 是调查年份，同一受试者可能在多个调查年份中有数据。变量中包括性别、年龄、种族、收入等人口学数据，包括体重、身高、脉搏、血压等基本体检数据，以及是否糖尿病、是否抑郁、是否怀孕、已生产子女数等更详细的健康数据，运动习惯、饮酒、性生活等行为方面的数据。这个教学用数据集最初的使用者是 Cashmere 高中的 Michelle Dalrymple 和新西兰奥克兰大学的 Chris Wild。

23.2 查看数据框一般信息

对一个数据框（包括 tibble）d，可以用 `dim(d)` 获得行、列数，用 `names(d)` 获得各列的变量名，用 `str(d)` 显示类型、大小、各列变量名以及类型、前几个值，以及其他属性。`dplyr::glimpse(d)` 则可以显示数据框的大小、各列变量名、类型等，如：

```
glimpse(d.class)
```

```
## Rows: 19
## Columns: 5
## $ name   <chr> "Alice", "Becka", "Gail", "Karen", "Kathy", "Mary", "Sandy", "S~
## $ sex    <fct> F, F, F, F, F, F, F, F, F, M, M, M, M, M, M, M, M, M, M
## $ age    <dbl> 13, 13, 14, 12, 12, 15, 11, 15, 14, 14, 14, 15, 12, 13, 12, 16,~
## $ height <dbl> 56.5, 65.3, 64.3, 56.3, 59.8, 66.5, 51.3, 62.5, 62.8, 69.0, 63.~
## $ weight <dbl> 84.0, 98.0, 90.0, 77.0, 84.5, 112.0, 50.5, 112.5, 102.5, 112.5,~
```

23.3 用 filter() 选择行子集

数据框的任何行子集仍为数据框，即使只有一行而且都是数值也是如此。行子集可以用行下标选取，如 `d.class[8:12,]`。函数 `head()` 取出数据框的前面若干行，`tail()` 取出数据框的最后若干行。

dplyr 包的 `filter()` 函数可以按条件选出符合条件的行组成的子集。下例从 `d.class` 中选出年龄在 13 岁和 13 岁以下的女生：

```
d.class |>
  filter(sex=="F", age<=13) |>
  knitr::kable()
```

name	sex	age	height	weight
Alice	F	13	56.5	84.0
Becka	F	13	65.3	98.0
Karen	F	12	56.3	77.0
Kathy	F	12	59.8	84.5
Sandy	F	11	51.3	50.5

`filter()` 函数第一个参数是要选择的数据框，后续的参数是条件，这些条件是需要同时满足的，另外，条件中取缺失值的观测自动放弃，这一点与直接在数据框的行下标中用逻辑下标有所不同，逻辑下标中有缺失值会在结果中产生缺失值。

`filter()` 会自动舍弃行名，如果需要行名只能将其转换成数据框的一列。

`filter()` 的结果为行子集数据框。用在管道操作当中的时候第一自变量省略（是管道传递下来的）。

23.4 按行序号选择行子集

基本 R 的 utils 包的函数 `head(x, n)` 可以用来选择数据框 `x` 前面 `n` 行，`tail(x, n)` 可以用来选择数据框 `x` 后面 `n` 行，如：

```
d.class |>
  head(n=5) |>
  knitr::kable()
```

name	sex	age	height	weight
Alice	F	13	56.5	84.0
Becka	F	13	65.3	98.0
Gail	F	14	64.3	90.0
Karen	F	12	56.3	77.0
Kathy	F	12	59.8	84.5

dplyr 包的函数 `slice(.data, ...)` 可以用来选择指定序号的行子集，正的序号表示保留，负的序号表示排除。如：

```
d.class |>
  slice(3:5) |>
  knitr::kable()
```

name	sex	age	height	weight
Gail	F	14	64.3	90.0
Karen	F	12	56.3	77.0
Kathy	F	12	59.8	84.5

还有几个 `slice_xxx()` 方便函数：

- `slice_head(n=5)` 提取前 n 行；
- `slice_tail(n=5)` 提取最后 n 行；
- `slice_min(x, n=1)` 提取 `x` 值最小的 n 行；
- `slice_max(x, n=1)` 提取 `x` 值最大的 n 行；
- `slice_sample(n=5)` 随机无放回抽取 `n` 行，也可以用 `sample_n(size = n)` 函数。

这些函数也可以用 `prop=0.1` 取代 `n=5` 这样的写法，输入一个要提取的行数的比例。

23.5 用 sample_n() 对观测随机抽样

dplyr 包的 `sample_n(tbl, size)` 函数可以从数据集 `tbl` 中随机无放回抽取 `size` 行，如：

```
d.class |>
  sample_n(size = 3) |>
  knitr::kable()
```

name	sex	age	height	weight
Philip	M	16	72.0	150
Jeffrey	M	13	62.5	84
Mary	F	15	66.5	112

`sample_n()` 中加选项 `replace=TRUE` 可以变成有放回抽样。可以用 `weight` 选项指定数据框中的一列作为抽样权重，进行不等概抽样。

23.6 用 distinct() 去除重复行

有时我们希望得到一个或若干个变量组合的所有不同值。dplyr 包的 `distinct()` 函数可以对数据框指定若干变量，然后筛选出所有不同值，每组不同值仅保留一行。指定变量名时不是写成字符串形式而是直接写变量名，这是 dplyr 和 tidyr 包的特点。例如，筛选出性别与年龄的所有不同组合：

```
d.class |>
  distinct(sex, age) |>
  knitr::kable()
```

sex	age
F	13
F	14
F	12
F	15
F	11
M	14
M	15
M	12
M	13
M	16
M	11

如果希望保留数据框中其它变量，可以加选项`.keep_all=TRUE`。

下面的程序查看 NHANES 数据框中 ID 与 SurveyYr 的组合的不同值的个数：

```
NHANES |>
  distinct(ID, SurveyYr) |>
  nrow()
```

```
## [1] 6779
```

这个结果提示有些人在某一调查年中有多个观测。

23.7 用 drop_na() 去除指定的变量有缺失值的行

在进行统计建模时，通常需要用到的因变量和自变量都不包含缺失值。tidyr 包的 `drop_na()` 函数可以对数据框指定一到多个变量，删去指定的变量有缺失值的行。不指定变量时有任何变量缺失的行都会被删去。

例如，将 NHANES 中所有存在缺失值的行删去后数出保留的行数，原来有 10000 行：

```
NHANES |>
  drop_na() |>
  nrow()
```

```
## [1] 0
```

可见所有行都有缺失值。下面仅剔除 AlcoholDay 缺失的观测并计数：

```
NHANES |>
  drop_na(AlcoholDay) |>
  nrow()
```

```
## [1] 4914
```

基本 stats 包的 `complete.cases` 函数返回是否无缺失值的逻辑向量，`na.omit` 函数则返回无缺失值的观测的子集。

23.8 用 select() 选择列子集

dplyr 包的 `select()` 选择列子集，并返回列子集结果。

可以指定变量名，如

```
d.class |>
  select(name, age) |>
  head(n=3) |>
  knitr::kable()
```

name	age
Alice	13
Becka	13
Gail	14

可以用冒号表示列范围，如

```
d.class |>
  select(age:weight) |>
  head(n=3) |>
  knitr::kable()
```

age	height	weight
13	56.5	84
13	65.3	98
14	64.3	90

可以用数字序号表示列范围，如

```
d.class |>
  select(3:5) |>
  head(n=3) |>
  knitr::kable()
```

age	height	weight
13	56.5	84
13	65.3	98
14	64.3	90

参数中前面写负号表示扣除，如

```
d.class |>
  select(-name, -age) |>
  head(n=3) |>
  knitr::kable()
```

sex	height	weight
F	56.5	84
F	65.3	98
F	64.3	90

可以用 `where(示性函数)` 用输入的示性函数选择满足条件的列，比如，选择所有数据型列：

```
d.class |>
  select(where(is.numeric)) |>
  head(n=3) |>
  knitr::kable()
```

age	height	weight
13	56.5	84
13	65.3	98
14	64.3	90

如果要选择的变量名已经保存为一个字符型向量，可以用 `all_of()` 函数引入，如

```
vars <- c("name", "sex")
d.class |>
  select(all_of(vars)) |>
  head(n=3) |>
  knitr::kable()
```

name	sex
Alice	F
Becka	F
Gail	F

R 的字符串函数（如 `paste()`）和正则表达式函数可以用来生成变量名子集，然后在 `select` 中配合 `all_of()` 使用。`all_of()` 要求指定的所有变量名都是数据框中存在的；如果指定的变量有些可能是不存在的，想将确实存在的那些变量选取进来，应使用 `any_of()`。

`select()` 有若干个配套函数可以按名字的模式选择变量列，如

- `starts_with("se")`: 选择名字以 “se”‘开头的变量列；
- `ends_with("ght")`: 选择名字以 “ght”‘结尾的变量列；
- `contains("no")`: 选择名字中含有子串 “no”‘的变量列；
- `matches("^[[:alpha:]]+[[:digit:]]+$")`，选择列名匹配某个正则表达式模式的变量列，这里匹配前一部分是字母，后一部分是数字的变量名，如 `abc12`。
- `num_range("x", 1:3)`，选择 `x1`, `x2`, `x3`。
- `everything()`: 代指所有选中的变量，这可以用来将指定的变量次序提前，其它变量排在后面。

选择变量时，可以用 “新变量 = 老变量” 的格式同时改名，如：

```
d.class |>
  select(id = name, gender=sex) |>
  head(n=3) |>
  knitr::kable()
```

id	gender
Alice	F
Becka	F
Gail	F

R 函数 subset 也能对数据框选取列子集和行子集。

23.9 取出单个变量为向量

如果需要选择单个变量并使得结果为普通向量，可以用 dplyr 包的 `pull()` 函数，如：

```
d.class |>
  head(n=3) |>
  pull(name) |>
  paste(collapse=":")
```

```
## [1] "Alice:Becka:Gail"
```

pull() 可以指定单个变量名，也可以指定变量序号，负的变量序号从最后一个变量数起。缺省变量名和序号时取出最后一个变量。

如果要取出的变量名保存在一个字符型变量 varname 中，可以用 pull(.data, !!sym(varname)) 这种格式；如果 varname 是函数的自变量，可以用 pull(.data, {{ varname }}) 这种格式。或者，先选择仅有一个变量的子数据框再用 pull()，如：

```
varname <- "name"
d.class |>
  head(n=3) |>
  select(one_of(varname)) |>
  pull() |>
  paste(collapse=":")
```

```
## [1] "Alice:Becka:Gail"
```

基于基本 R，也可以用 d.class[["name"]] 这种格式取出一列为普通变量，如果 varname 保存了变量名，可以用 d.class[[varname]] 这种格式。

不能用 d.class[,"name"] 这种方法，对于 tibble 类型，其结果仍是一个子数据框；用 d.class["name"] 这种格式，结果也是一个子数据框。

23.10 用 arrange() 排序

dplyr 包的 arrange() 按照数据框的某一列或某几列排序，返回排序后的结果，如

```
d.class |>
  arrange(sex, age) |>
  knitr::kable()
```

name	sex	age	height	weight
Thomas	M	11	57.5	85.0
James	M	12	57.3	83.0
John	M	12	59.0	99.5
Robert	M	12	64.8	128.0
Jeffrey	M	13	62.5	84.0
Alfred	M	14	69.0	112.5
Duke	M	14	63.5	102.5
Guido	M	15	67.0	133.0
William	M	15	66.5	112.0
Philip	M	16	72.0	150.0
Sandy	F	11	51.3	50.5
Karen	F	12	56.3	77.0
Kathy	F	12	59.8	84.5
Alice	F	13	56.5	84.0
Becka	F	13	65.3	98.0
Gail	F	14	64.3	90.0
Tammy	F	14	62.8	102.5
Mary	F	15	66.5	112.0
Sharon	F	15	62.5	112.5

用 `desc()` 包裹想要降序排列的变量，如

```
d.class |>
  arrange(sex, desc(age)) |>
  knitr::kable()
```

name	sex	age	height	weight
Philip	M	16	72.0	150.0
Guido	M	15	67.0	133.0
William	M	15	66.5	112.0
Alfred	M	14	69.0	112.5
Duke	M	14	63.5	102.5
Jeffrey	M	13	62.5	84.0
James	M	12	57.3	83.0
John	M	12	59.0	99.5
Robert	M	12	64.8	128.0
Thomas	M	11	57.5	85.0
Mary	F	15	66.5	112.0
Sharon	F	15	62.5	112.5
Gail	F	14	64.3	90.0
Tammy	F	14	62.8	102.5
Alice	F	13	56.5	84.0
Becka	F	13	65.3	98.0
Karen	F	12	56.3	77.0
Kathy	F	12	59.8	84.5
Sandy	F	11	51.3	50.5

排序时不论升序还是降序，所有的缺失值都自动排到末尾。

R 函数 `order()` 可以用来给出数据框的排序次序，然后以其输出为数据框行下标，可以将数据框排序。

23.11 用 `rename()` 修改变量名

在 dplyr 包的 `rename()` 中用 “新名字 = 旧名字” 格式修改变量名，如

```
d2.class <- d.class |>
  dplyr::rename(h=height, w=weight)
```

注意这样改名字不是对原始数据框修改而是返回改了名字后的新数据框。也可以利用赋值运算符`->` 写成：

```
d.class |>
  dplyr::rename(h=height, w=weight) ->
  d2.class
```

`rename()` 这个函数可能出现在其它包中，保险起见写成 `dplyr::rename()`。

如果数据框中有大批变量名需要按某种规范统一修改，可以考虑使用`janitor::clean_names()`。

23.12 用 relocate() 调整变量次序

relocate(df, var1, var2) 将指定的变量调整到最前面。如：

```
d.class |>
  relocate(height, weight) |>
  head(n=3) |>
  knitr::kable()
```

height	weight	name	sex	age
56.5	84	Alice	F	13
65.3	98	Becka	F	13
64.3	90	Gail	F	14

可以用.before 指定一个变量名或变量序号，使得指定的变量调整到此变量前面，用.after 指定一个变量，使得指定的变量调整到此变量后面。如：

```
d.class |>
  relocate(height, weight, .after=name) |>
  head(n=3) |>
  knitr::kable()
```

name	height	weight	sex	age
Alice	56.5	84	F	13
Becka	65.3	98	F	13
Gail	64.3	90	F	14

23.13 用 mutate() 计算新变量

dplyr 包的 mutate() 可以为数据框计算新变量，返回含有新变量以及原变量的新数据框。如

```
d.class |>
  mutate(
    rwh=weight/height,
    sexc=if_else(sex=="F", " 女", " 男")) |>
  head(n=3) |>
  knitr::kable()
```

name	sex	age	height	weight	rwh	sexc
Alice	F	13	56.5	84	1.486726	女
Becka	F	13	65.3	98	1.500766	女
Gail	F	14	64.3	90	1.399689	女

上面程序中的 `if_else()` 是 dplyr 包对基本 R 函数 `ifelse()` 的一个改进版本，能够好处理缺失值和结果类属的问题。

用 `mutate()` 计算新变量时如果计算比较复杂，也可以用多个语句组成复合语句，如：

```
d.class |>
  mutate(
    sexc = {
      x <- rep(" 男", length(sex))
      x[!is.na(sex) & sex == "F"] <- " 女"
      x
    }  ) |>
  head(n=3) |>
  knitr::kable()
```

name	sex	age	height	weight	sexc
Alice	F	13	56.5	84	女
Becka	F	13	65.3	98	女
Gail	F	14	64.3	90	女

注意这样生成新变量不是在原来的数据框中添加，原来的数据框没有被修改，而是返回添加了新变量的新数据框。R 软件的巧妙设计保证了这样虽然是生成了新数据框，但是与原来数据框重复的列并不会重复保存。

计算公式中可以包含对数据框中变量的统计函数结果，如

```
d.class |>
  mutate(
    cheight = height - mean(height)) |>
  knitr::kable()
```

新变量可以与老变量名相同，这样就在输出中修改了老变量。

可以加`.before=1` 选项，使得新定义的变量添加到原有变量的最前面。可以用`.after=` 变量名使得新添加的变量放在指定的变量后面。可以用选项`.keep="used"` 仅保留新变量以及计算新变量时用到的变量。

dplyr 定义了一些新的函数：

- `row_number()` 返回每个观测的行号。
- `min_rank(x)`，计算秩统计量，冲突时按最小值，类似函数有 `dense_rank(x)`，`percent_rank(x)`，`cume_rank(x)` 等。基本 R 的 `rank(x)` 函数配合 `ties.method` 选项实现类似功能。
- `dplyr::lag(x)` 将 `x` 看成一个时间为 `1:n` 的时间序列，然后在第 i 个时间点返回第 $i-1$ 个时间点的值，如 `dplyr::lag(c(2,3,5,7))` 返回 `NA, 2, 3, 5`。`dplyr::lead(x)` 则在时间点 i 返回时间点 $i+1$ 的值，如 `dplyr::lead(c(2,3,5,7))` 返回 `3,5,7,NA`。注意 `stats::lag()` 函数有很大的差别，它主要针对时间序列数据，对一般向量反而不好用。

23.14 用 tranmute() 生成新变量的数据框

函数 transmute() 用法与 mutate() 类似，但是仅保留新定义的变量，不保留原来的所有变量。如：

```
d.class |>
  transmute(
    height_cm = round(height*2.54),
    weight_kg = round(weight*0.4535924),
    bmi =  weight_kg / (height_cm / 100)^2) |>
  head(n=3) |>
  knitr::kable()
```

height_cm	weight_kg	bmi
144	38	18.32562
166	44	15.96748
163	41	15.43152

可见结果中仅保留了新定义的变量。

定义新变量也可以直接为数据框的新变量赋值：

```
d.class[["rwh"]] <- d.class[["weight"]] / d.class[["height"]]
```

这样的做法与 mutate() 的区别是这样不会生成新数据框，新变量是在原数据框中增加的。

给数据框中某个变量赋值为 NULL 可以修改数据框，从数据框中删去该变量。

23.15 缺失值填补

从统计角度出发考虑如何填补数据中的缺失值是比较复杂的。这里仅给出从编程考虑的做法。

有些非整洁的数据在某变量值持续多行保持不变时，往往仅输入第一个值，随后重复的值就输入为空白，在读入为 R 数据框时自动变成缺失值。对于这样的缺失值，可以用 tidyr::fill() 函数，可以指定需要填补的变量。如：

A	B	n
1	1	12
NA	2	10
NA	3	NA
2	1	15
NA	2	NA
NA	3	14

填补变量 A 的缺失值：

```
d.miss |>
  fill(A)
```

```
## # A tibble: 6 x 3
##       A     B     n
##   <dbl> <dbl> <dbl>
## 1     1     1    12
## 2     1     2    10
## 3     1     3    NA
## 4     2     1    15
## 5     2     2    NA
## 6     2     3    14
```

有时某个变量的缺失值实际上有含义，比如取值为 0。这时，可以用 `dplyr::coalesce(x, default)` 使 `x` 的缺失值用 `default` 取代。如：

```
d.miss |>
  mutate(
    x = coalesce(n, 0)
  )
```

```
## # A tibble: 6 x 4
##       A     B     n     x
##   <dbl> <dbl> <dbl> <dbl>
## 1     1     1    12    12
## 2    NA     2    10    10
## 3    NA     3    NA     0
## 4     2     1    15    15
## 5    NA     2    NA     0
## 6    NA     3    14    14
```

有时，有可能缺失值被输入成了一个特殊数值，比如，`x` 的值仅取正数，输入了`-1` 就表示缺失值。在读入 CSV 文件时，如果所有缺失值都用`-1` 表示，可以在 `read_csv()` 中加选项 `na = -1`。如果不同列有不同的缺失值编码，可以用 `na_if(x, na_value)` 将 `x` 中等于 `na_value` 的值改为缺失值，可以用在 `mutate()` 中，如 `mutate(x = na_if(x, -1))`。

在涉及到交叉分组的数据时，有可能某些交叉分组没有出现在数据中，这可能对某些分析程序造成影响。可以用 `complete(A, B)` 这样的方法将数据框中 `A, B` 的交叉分类补全。如果这样不能包含所有可取值，也可以生成一个包含所有组合的小数据框，用 `dplyr::full_join()` 作外连接。另外，`dplyr::anti_join()` 可以用变量取值全集的数据框来比对，发现没有出现的变量取值。

在使用 `dplyr::count()` 对因子进行频数统计时，数据中不出现的水平或水平组合默认在结果中也不出现，但可以加选项`.drop = FALSE` 使得这些水平或水平组合以频数 0 出现。用 `dplyr::group_by()` 时，也可以加`.drop = FALSE` 使得没有观测的分组（但因子水平存在）也出现在结果中。但更好的办法是在分组汇总完以后，用 `complete(A, B)` 这样的办法补全汇总结果中的因子水平或水平组合。

23.16 用管道连接多次操作

管道运算符特别适用于对同一数据集进行多次操作。例如，对 `d.class` 数据，先选出所有女生，再去掉性别和 age 变量：

```
d.class |>
  filter(sex=="F") |>
  select(-sex, -age) |>
  knitr::kable()
```

name	height	weight
Alice	56.5	84.0
Becka	65.3	98.0
Gail	64.3	90.0
Karen	56.3	77.0
Kathy	59.8	84.5
Mary	66.5	112.0
Sandy	51.3	50.5
Sharon	62.5	112.5
Tammy	62.8	102.5

管道操作的结果可以保存为新的 tibble，如:

```
class_F <- d.class |>
  filter(sex=="F") |>
  select(-sex, -age)
```

也可以将赋值用`->` 写在最后，如：

```
d.class |>
  filter(sex=="F") |>
  select(-sex, -age) -> class_F
```

如果管道传递的变量在下一层调用中不是第一自变量，可以用 `_` 代表，如：

```
d.class |>
  lm(weight ~ height, data = _) |>
  coef()
```

```
## (Intercept)      height
##  -143.02692     3.89903
```

23.17 expand_grid() 函数

在进行有多个因素的试验设计时，往往需要生成多个因素完全搭配并重复的表格。tidyr 包的函数 `expand_grid()` 可以生成这样的重复模式。基本 R 的 `expand.grid()` 功能类似。基本 R 的 gl 函数提供了更简单的功能。

比如，下面的例子：

```
tidyr::expand_grid(
  group=1:3,
  subgroup=1:2,
  obs=1:2) |>
  knitr::kable()
```

group	subgroup	obs
1	1	1
1	1	2
1	2	1
1	2	2
2	1	1
2	1	2
2	2	1
2	2	2
3	1	1
3	1	2
3	2	1
3	2	2

结果的数据框 d 有三个变量: group 是大组，共分 3 个大组，每组 4 个观测；subgroup 是子组，在每个大组内分为 2 个子组，每个子组 2 个观测。共有 $3 \times 2 \times 2 = 12$ 个观测（行）。

23.18 宽表转换为长表

“整洁数据”的要求是每行为一个观测，每列为一个变量（属性），如果有同一个体在不同时间的测量值应该放在不同的观测中，如果同一个体在同一时间的不同属性应该放在同一观测中。满足这些要求的数据框才容易作为作图、汇总、统计建模函数的输入使用。

实际数据经常不满足上述的要求，所以需要将数据框进行长宽格式的转换。以典型的纵向数据为例，每个受试者有多次随访的记录值，如果每个受试者的所有随访记录值存放在一个观测中，就称为宽表，这不符合整洁数据要求；如果每个受试者的随访记录值同一时间的多个属性用了多个观测保存，称为长表，也不符合整洁数据要求。

有些数据将受试者的每个属性放在一行，每一列代表一个受试者，这也不符合整洁数据要求。

tidyr 的 `pivot_longer()` 可以将宽表转换成整洁数据，`pivot_wider()` 可以将长表转换成整洁数据。基本 R 的 `reshape()` 函数也可以进行长宽格式的转换。reshape2 包也提供了较丰富的长宽表转换功能。建议优先使用 tidyr 包的功能。

23.18.1 `pivot_longer` 函数

tidyr 的 `pivot_longer()` 函数可以将横向的多次观测堆叠在一列中。例如，下面的数据：

```
knitr::kable(dwide1)
```

subject	1	2	3	4
1	1	NA	NA	NA
2	NA	7	NA	4
3	5	10	NA	NA
4	NA	NA	9	NA

subject 是受试者编号，每个受试者有 4 次随访，NA 表示缺失。数据分析和绘图用的函数一般不能直接使用这样的数据，一般需要将 4 次测量合并在一列中作为分析变量，将随访序号单独放在另外一列中。用 `pivot_longer()` 函数实现：

```
dwide1 |>
  pivot_longer(`1`:`4`,
     names_to = "time",
     values_to = "response") |>
  knitr::kable()
```

subject	time	response
1	1	1
1	2	NA
1	3	NA
1	4	NA
2	1	NA
2	2	7
2	3	NA
2	4	4
3	1	5
3	2	10
3	3	NA
3	4	NA
4	1	NA
4	2	NA
4	3	9
4	4	NA

选项 `names_to` 指定一个新变量名，将原来的列标题转换为该变量的值；选项 `values_to` 指定一个新变量名，将原来的各个列对应的测量值保存在该变量名的列中。

注意原来的变量名不是合法 R 变量名，所以在 `pivot_longer()` 中用反单撇号保护，并用了冒号来表示变量范围，也可以仿照 `select` 函数中指定变量名的方法将程序中的`` `1`:`4` ``替换为:

- `c("1", "2", "3", "4")`
- `c(`1`, `2`, `3`, `4`)`
- `-subject`
- `cols = all_of(vars)`, 其中 `vars` 被赋值为 `c("1", "2", "3", "4")`。

如果转换结果中不希望保留那些 NA，可以加 `values_drop_na=TRUE`:

```
dwide1 |>
  pivot_longer(`1`:`4`,
     names_to = "time",
     values_to = "response",
     values_drop_na = TRUE) |>
  knitr::kable()
```

subject	time	response
1	1	1
2	2	7
2	4	4
3	1	5
3	2	10
4	3	9

23.18.2 从列名中提取数值

有时要合并的列名中带有数值，需要将这些数值部分提取出来，这时可以用 `names_prefix` 指定要去掉的非数值前缀，用 `names_transform` 指定将列名转换为值时结果的类型，`names_transform` 是一个列表，实现列表元素名到转换函数的映射。例如，上述的 dwide1 数据框变成这样：

subject	FU1	FU2	FU3	FU4
1	1	NA	NA	NA
2	NA	7	NA	4
3	5	10	NA	NA
4	NA	NA	9	NA

可以用如下程序将随访编号变成整数值存入一列：

```
dwide2 |>
  pivot_longer(cols = starts_with("FU"),
     names_to = "time",
     values_to = "response",
     names_prefix = "FU",
     names_transform = list(time = as.integer),
     values_drop_na = TRUE) |>
  knitr::kable()
```

subject	time	response
1	1	1
2	2	7
2	4	4
3	1	5
3	2	10
4	3	9

其中的 `cols = starts_with("FU")` 也可以写成 `cols = paste0("FU", 1:4)`。

考虑 nlmeU 扩展包的 armd.wide 数据框。这个数据框中有 240 个受试者的信息，每行为一个受试者，包括 5 个时间点：visual0, visual4, visual12, visual24, visual52。数据如：

```
library(nlmeU)
```

```
##
## Attaching package: 'nlmeU'

## The following object is masked from 'package:stats':
##
##     sigma
```

```
data(armd.wide)
knitr::kable(head(armd.wide, 3))
```

subject	lesion	line0	visual0	visual4	visual12	visual24	visual52	treat.f	miss.pat
1	3	12	59	55	45	NA	NA	Active	–XX
2	1	13	65	70	65	65	55	Active	—-
3	4	8	40	40	37	17	NA	Placebo	—X

将其转换为长表格式并提取时间信息的程序如下：

```
armd.wide |>
  pivot_longer(
    cols = starts_with("visual"),
    names_to = "time",
    values_to = "visual",
    names_prefix = "visual",
    names_transform = list(time = as.integer),
    values_drop_na = TRUE) |>
  head(10) |>
  knitr::kable()
```

subject	lesion	line0	treat.f	miss.pat	time	visual
1	3	12	Active	–XX	0	59
1	3	12	Active	–XX	4	55
1	3	12	Active	–XX	12	45
2	1	13	Active	—-	0	65
2	1	13	Active	—-	4	70
2	1	13	Active	—-	12	65
2	1	13	Active	—-	24	65
2	1	13	Active	—-	52	55
3	4	8	Placebo	—X	0	40
3	4	8	Placebo	—X	4	40

在转换的长表中，希望增加基线测量值到每个观测中，并希望将时间 0, 4, 12, 24, 52 增加一个时间序号 0, 1, 2, 3, 4。程序修改为：

```
armd.wide |>
  mutate(base = visual0) |>
  pivot_longer(
    cols = starts_with("visual"),
    names_to = "time",
    values_to = "visual",
    names_prefix = "visual",
    names_transform = list(time = as.integer),
    values_drop_na = TRUE) |>
  mutate(timep = as.integer(factor(time, levels=c(0, 4, 12, 24, 52))) - 1) |>
  head(10) |>
  knitr::kable()
```

subject	lesion	line0	treat.f	miss.pat	base	time	visual	timep
1	3	12	Active	–XX	59	0	59	0
1	3	12	Active	–XX	59	4	55	1
1	3	12	Active	–XX	59	12	45	2
2	1	13	Active	—-	65	0	65	0
2	1	13	Active	—-	65	4	70	1
2	1	13	Active	—-	65	12	65	2
2	1	13	Active	—-	65	24	65	3
2	1	13	Active	—-	65	52	55	4
3	4	8	Placebo	—X	40	0	40	0
3	4	8	Placebo	—X	40	4	40	1

23.18.3 从列名中提取多个分类变量值

上面的 dwide2 数据集的 FU1 到 FU4 变量中包含了随访次数这一个变量的值。有些数据集在列名中用编码形式保存了不止一个变量的信息，假设那些列保存的数值仍属于同一属性。例如，下面的数据：

unit	F_1	F_2	M_1	M_2
1	55	52	64	60
2	98	93	120	116
3	40	38	44	40

假设这是对某个问题的赞成或反对意见的某个抽样调查的频数表，unit 是不同的抽样子集，其它四列都是频数，`F_1` 代表女性中赞成人数，`F_2` 代表女性中反对人数，`M_1` 代表男性中赞成人数，`M_2` 代表男性中反对人数。

为了利用这样的数据，需要将不同性别和两种意见的人数都合并到一列中，增加性别和意见列。这时，`names_to` 指

定多个变量名，即在变量名中包含的多个变量的新变量名，用 `names_sep = "_"` 指定在各个分类值直接的分隔符号是下划线。程序如：

```
dwide3 |>
  pivot_longer(
    cols = F_1:M_2,
    names_to = c("gender", "response"),
    values_to = "freq",
    names_sep = "_",
    names_ptypes = list(
      gender = factor(
        levels = c("F", "M")),
      response = factor(
        levels = c("1", "2"))
    )
  ) |>
  knitr::kable()
```

unit	gender	response	freq
1	F	1	55
1	F	2	52
1	M	1	64
1	M	2	60
2	F	1	98
2	F	2	93
2	M	1	120
2	M	2	116
3	F	1	40
3	F	2	38
3	M	1	44
3	M	2	40

如果多个分类变量值编码组合列名时没有用分隔符分隔，就需要使用正则表达式的方式将有变量值的部分用正则表达式的捕获子集标记出来，关于正则表达式详见42。比如，若变量名为 `F1`, `M1`, `F2`, `M2`，则应将上面程序中的 `names_sep = "_"` 修改为 `names_pattern = "(F|M)(1|2)"`。其中的`"(F|M)(1|2)"` 就是正则表达式，`F|M` 表示 F 或者 M，`1|2` 表示 1 或者 2，圆括号表示分组，前后两组分别表示两个分类变量 `gender` 和 `response`。

23.18.4 一行中有多个属性的多次观测的情形

设有多个属性的多次测量用编号的列名保存在了同一观测中，例如，基本 R 软件中的 anscombe 数据集的一部分行：

```
dwide4 <- anscombe |>
  slice(1:3) |>
  mutate(id = 1:3) |>
  select(id, x1, x2, y1, y2)
knitr::kable(dwide4)
```

id	x1	x2	y1	y2
1	10	10	8.04	9.14
2	8	8	6.95	8.14
3	13	13	7.58	8.74

这可以看成是每个受试者的 x, y 两个变量的 2 次随访的值保存在了一个观测中。用 `names_pattern` 指定切分变量名和随访号的模式，在对应的 `names_to` 中用特殊的`".value"` 名字表示切分出来的那一部分实际是变量名，这时不需要 `values_to` 选项。程序如下：

```
dwide4 |>
  pivot_longer(
    -id,
    names_pattern = "(x|y)([[:digit:]])",
    names_to = c(".value", "time")
  ) |>
  knitr::kable()
```

id	time	x	y
1	1	10	8.04
1	2	10	9.14
2	1	8	6.95
2	2	8	8.14
3	1	13	7.58
3	2	13	8.74

这里“`(x|y)([[:digit:]])`”是正则表达式，“`x|y`”意思是匹配 x 或 y，“`[[:digit:]]`”意思是匹配阿拉伯数字，圆括号用来分组。

23.19 长表转换为宽表

23.19.1 将多个混在一起的变量拆开

tidyr 包的 `pivot_wider` 函数可以将长表变成宽表。这适用于将多个变量保存到了一列的情况。例如，下面的长表将变量 x 和 y 放在了同一列中：

id	variable	value
1	x	11
1	y	23
2	x	10
2	y	20
3	x	15
3	y	28

这样的数据也不利于进行统计分析，我们用 `pivot_wider` 函数将两个变量放到各自的列中，用 `names_from` 选项指定区分不同变量的列，用 `values_from` 指定保存实际变量值的列：

```
dlong1 |>
  pivot_wider(
    names_from = "variable",
    values_from = "value"  ) |>
  knitr::kable()
```

id	x	y
1	11	23
2	10	20
3	15	28

例子数据框 dlong1 除了分组用的变量 id 和要展开的列 variable、value 以外没有额外的变量，如果有额外的变量，就无法自动识别哪一列用来区分不同个体，这时应该用 `id_cols` 指定用来区分个体的一列或多列，如：

```
dlong1 |>
  pivot_wider(
    id_cols = c("id"),
    names_from = "variable",
    values_from = "value"  ) |>
  knitr::kable()
```

id	x	y
1	11	23
2	10	20
3	15	28

在这样拆分列时，有可能某些变量值不存在，例如：

id	variable	value
1	x	11
1	y	23
2	x	10
3	y	28

这里 2 号 id 缺少 y，3 号 id 缺少 x。直接转换为宽表：

```
dlong2 |>
  pivot_wider(
    id_cols = c("id"),
    names_from = variable,
    values_from = value  ) |>
  knitr::kable()
```

id	x	y
1	11	23
2	10	NA
3	NA	28

产生了缺失值。如果知道缺失值实际等于 0，可以用选项 `values_fill=` 选项指定，如：

```
dlong2 |>
  pivot_wider(
    id_cols = c("id"),
    names_from = variable,
    values_from = value,
    values_fill = list(
      value = 0)  ) |>
  knitr::kable()
```

id	x	y
1	11	23
2	10	0
3	0	28

23.19.2 将多个类别合并到一个观测

在某些情况下也可能需要将数据转换为非整洁形态。比如，为了给读者用列联表形式显示数据，为了某个特定的多元分析程序使用，等等。

设 3 个受试者的 2 次测量值放在变量 x 中，用 `time` 区分 2 次测量值：

id	time	x
1	1	11
1	2	10
2	1	15
2	2	13
3	1	18
3	2	16

下面的程序将 x 的两次测量变成变量 x1 和 x2：

```
dlong3 |>
  pivot_wider(
    id_cols = c("id"),
    names_from = time,
    values_from = x,
    names_prefix = "x") |>
  knitr::kable()
```

id	x1	x2
1	11	10
2	15	13
3	18	16

23.19.3 将交叉类别合并到一个观测

这个例子也是将数据转换为非整洁形态。考虑如下的频数表数据：

year	sex	type	count
2018	F	Benign	4
2018	F	Malignant	9
2018	M	Benign	18
2018	M	Malignant	3
2019	F	Benign	6
2019	F	Malignant	10
2019	M	Benign	20
2019	M	Malignant	5

下面的程序将每年的数据合并到一行中：

```
dlong4 |>
  pivot_wider(
    id_cols = c("year"),
    names_from = c("sex", "type"),
    values_from = "count"
  ) |>
  knitr::kable()
```

year	F_Benign	F_Malignant	M_Benign	M_Malignant
2018	4	9	18	3
2019	6	10	20	5

23.19.4 多个变量的多种值

设有如下的 x 变量和 y 变量的分组汇总统计数据：

group	variable	avg	sd
1	x	1.2	0.5
1	y	-5.1	0.4
2	x	1.4	0.5
2	y	-4.9	0.8
3	x	1.3	0.7
3	y	-4.3	0.9

下面将 x 和 y 的两种统计量都放到同一行中：

```
dlong5 |>
  pivot_wider(
    id_cols = c("group"),
    names_from = "variable",
    values_from = c("avg", "sd")
  ) |>
  knitr::kable()
```

group	avg_x	avg_y	sd_x	sd_y
1	1.2	-5.1	0.5	0.4
2	1.4	-4.9	0.5	0.8
3	1.3	-4.3	0.7	0.9

23.19.5 长宽转换混合使用

有时数据需要使用两个方向的转换才能达到可用的程度，比如下面的数据：

```
knitr::kable(dlong6)
```

id	variable	2018	2019
1	x	1.2	1.3
1	y	-5.1	-5.4
2	x	1.4	1.6
2	y	-4.9	-4.2
3	x	1.3	1.5
3	y	-4.3	-4.1

这个数据的问题是 x, y 应该放在两列中却合并成一个了，2018 和 2019 应该放在一列中却分成了两列。先合并 2018 和 2019 这两列，然后再拆分 x 和 y：

```
dlong6 |>
  pivot_longer(
    `2018`:`2019`,
    names_to = "year",
    values_to = "value"
    ) |>
  pivot_wider(
    id_cols = c("id", "year"),
    names_from = "variable",
    values_from = "value"
    ) |>
  knitr::kable()
```

id	year	x	y
1	2018	1.2	-5.1
1	2019	1.3	-5.4
2	2018	1.4	-4.9
2	2019	1.6	-4.2
3	2018	1.3	-4.3
3	2019	1.5	-4.1

23.20 拆分数据列

23.20.1 separate 函数

有时应该放在不同列的数据用分隔符分隔后放在同一列中了。比如，下面数据集中 “succ/total” 列存放了用 “/” 分隔开的成功数与试验数：

```
d.sep <- read_csv(
"testid, succ/total
1, 1/10
2, 3/5
3, 2/8
")
```

```
## Rows: 3 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): succ/total
```

```
## dbl (1): testid
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
knitr::kable(d.sep)
```

testid	succ/total
1	1/10
2	3/5
3	2/8

用 `tidyr::separate()` 可以将这样的列拆分为各自的变量列，如

```
d.sep |>
  separate(
    `succ/total`,
    into=c("succ", "total"),
    sep="/",
    convert=TRUE) |>
  knitr::kable()
```

testid	succ	total
1	1	10
2	3	5
3	2	8

其中 `into` 指定拆分后新变量名，`sep` 指定分隔符，`convert=TRUE` 要求自动将分割后的值转换为适当的类型。`sep` 还可以指定取子串的字符位置，按位置拆分各个子串。

选项 `extra` 指出拆分时有多余内容的处理方法，选项 `fill` 指出有不足内容的处理方法。

拆分的也可以是变量名和因子，比如，变量包括血压的高压和低压，分男女计算了平均值，结果表格可能为如下格式：

```
knitr::kable(dbpa)
```

var	avg
male:systolicbp	118
male:diastolicbp	85
female:systolicbp	115
female:diastolicbp	83

用 `separate()` 函数将变量名和性别值分开：

```
dbpa2 <- dbpa |>
  separate(var, into = c("sex", "var"), sep=":")
knitr::kable(dbpa2)
```

sex	var	avg
male	systolicbp	118
male	diastolicbp	85
female	systolicbp	115
female	diastolicbp	83

实际上，这个数据集可能还需要将高压和低压变成两列，用 `pivot_wider()` 函数：

```
dbpa3 <- dbpa2 |>
  pivot_wider(
    names_from = "var", values_from = "avg")
knitr::kable(dbpa3)
```

sex	systolicbp	diastolicbp
male	118	85
female	115	83

也可以拆分出更多列，如：

```
knitr::kable(dth)
```

id	sm
1	1,2,3
2	4,5,6
3	7,8,9

```
dth |>
  separate(
    sm,
    into=c("x1", "x2", "x3"),
    sep=",",
    convert=TRUE)
```

```
## # A tibble: 3 x 4
##      id    x1    x2    x3
##   <int> <int> <int> <int>
## 1     1     1     2     3
## 2     2     4     5     6
## 3     3     7     8     9
```

23.20.2 extract 函数

函数 `extract()` 可以按照某种正则表达式表示的模式从指定列拆分出对应于正则表达式中捕获组的一列或多列内容。例如，下面的数据中 factors 水平 AA, AB, BA, BB 实际是两个因子的组合，将其拆分出来：

```
dexp <- tibble(
  design = c("AA", "AB", "BA", "BB"),
  response = c(120, 110, 105, 95))
knitr::kable(dexp)
```

design	response
AA	120
AB	110
BA	105
BB	95

```
dexp |>
  extract(
    design,
    into = c("fac1", "fac2"),
    regex = "(.)(.)"
  ) |>
  knitr::kable()
```

fac1	fac2	response
A	A	120
A	B	110
B	A	105
B	B	95

23.20.3 separate_wider_delim 函数

`tidyr::separate_wider_delim(x, delim, names)` 将用分隔符分隔的内容拆分为多列，新的列名用 `names` 给出。相当于 `separate()` 的一个简化版本。

`separate_wider_position(x, width)` 用指定宽度从字符串中提取多列，`width` 用一个有名向量格式指定，输入的元素名作为提取的列名。

23.20.4 separate_longer_delim 函数

`tidyr::separate_longer_delim()` 将某列中用分隔符分开的值拆分出来并堆叠在同一列中，如：

```
d.tmp <- tibble(
  xmixed=c("1,2", "3,4,5"))
d.tmp |>
  mutate(
    x = separate_longer_delim(xmixed, sep=",")  )
```

类似地，separate_longer_position(x, width) 可以按指定宽度拆分字符串中的内容并堆叠到一列中。

23.21 合并数据列

tidyr::unite() 函数可以将同一行的两列或多列的内容合并成一列。这是 separate() 的反向操作，如：

```
d.sep |>
  separate(`succ/total`, into=c("succ", "total"),
           sep="/", convert=TRUE) |>
  unite(ratio, succ, total, sep=":") |>
  knitr::kable()
```

testid	ratio
1	1:10
2	3:5
3	2:8

unite() 的第一个参数是要修改的数据框，这里用管道 |> 传递进来，第二个参数是合并后的变量名（ratio 变量），其它参数是要合并的变量名，sep 指定分隔符。实际上用 mutate()、paste() 或者 sprintf() 也能完成合并。

23.22 数据框纵向合并

矩阵或数据框要纵向合并，使用 rbind 函数即可。dplyr 包的 bind_rows() 函数也可以对两个或多个数据框纵向合并。要求变量集合是相同的，变量次序可以不同。

比如，有如下两个分开男生、女生的数据框：

```
d3.class <- d.class |>
  select(name, sex, age) |>
  filter(sex=="M")
d4.class <- d.class |>
  select(name, sex, age) |>
  filter(sex=="F")
```

合并行如下:

```
d3.class |>
  bind_rows(d4.class) |>
  knitr::kable()
```

name	sex	age
Alfred	M	14
Duke	M	14
Guido	M	15
James	M	12
Jeffrey	M	13
John	M	12
Philip	M	16
Robert	M	12
Thomas	M	11
William	M	15
Alice	F	13
Becka	F	13
Gail	F	14
Karen	F	12
Kathy	F	12
Mary	F	15
Sandy	F	11
Sharon	F	15
Tammy	F	14

将下面的数据框的变量列次序打乱，合并不受影响：

```
d3.class |>
  select(age, name, sex) |>
  bind_rows(d4.class) |>
  knitr::kable()
```

age	name	sex
14	Alfred	M
14	Duke	M
15	Guido	M
12	James	M
13	Jeffrey	M
12	John	M
16	Philip	M
12	Robert	M
11	Thomas	M
15	William	M
13	Alice	F
13	Becka	F
14	Gail	F
12	Karen	F
12	Kathy	F
15	Mary	F
11	Sandy	F
15	Sharon	F
14	Tammy	F

23.23 横向合并

为了将两个行数相同的数据框按行号对齐合并，可以用基本 R 的 `cbind()` 函数或者 dplyr 包的 `bind_cols()` 函数。

23.23.1 连接介绍

实际数据往往没有存放在单一的表中，需要从多个表查找数据。多个表之间的连接，一般靠关键列（key）对准来连接。连接可以是一对一的，一对多的。多对多连接应用较少，因为多对多连接是所有两两组合。

在规范的数据库中，每个表都应该有主键，这可以是一列，也可以是多列的组合。为了确定某列是主键，可以用 `count()` 和 `filter()`，如

```
d.class |>
  count(name) |>
  filter(n>1) |>
  nrow()
```

```
## [1] 0
```

没有发现重复出现的 name，说明 d.class 中 name 可以作为主键。

23.23.2 一对一内连接

为了演示一对一的横向连接，我们将 d.class 取 11 和 12 岁的子集，然后拆分为两个数据集 d1.class 和 d2.class，两个数据集都有主键 name，d1.class 包含变量 name, sex, d2.class 包含变量 name, age, height, weight, 并删去某些观测：

```
d1.class <- d.class |>
  filter(age <= 12) |>
  select(name, sex) |>
  filter(!(name %in%  "Sandy"))
d2.class <- d.class |>
  filter(age <= 12) |>
  select(name, age, height, weight)
```

用 dplyr 包的 inner_join() 函数将两个数据框按键值横向合并，仅保留能匹配的观测。因为 d1.class 中丢失了 Sandy 的观测，所以合并后的数据框中也没有 Sandy 的观测：

```
d1.class |>
  inner_join(d2.class) |>
  knitr::kable()
```

```
## Joining, by = "name"
```

name	sex	age	height	weight
Karen	F	12	56.3	77.0
Kathy	F	12	59.8	84.5
James	M	12	57.3	83.0
John	M	12	59.0	99.5
Robert	M	12	64.8	128.0
Thomas	M	11	57.5	85.0

横向连接自动找到了共同的变量 name 作为连接的键值，可以在 inner_join() 中用 join_by(键) 指定键值变量名，如果有不同的键名，可以用 join_by(左键 == 右键) 的格式指定，这里用了 == 是因为这类似于 SQL 中左右连接时的连接条件，这里是相等情况下的连接，而 R 中用两个等于号表示是否相等的判断。

如果你确信左右表的连接都是一对一的，如果出现重复匹配意味着数据错误，可以在 inner_join() 中加选项 multiple = "error"，其默认值是"warn"，即会显示警告信息。如果在 inner_join() 或 left_join() 中使用 multiple = "all"，则意味着允许左表的一个观测可以和右表的多个观测匹配。

如果左右表的一对一连接不应该有遗漏，如果有任何不匹配意味着数据有错，可以在 inner_join() 中加选项 unmatched = "error"，其默认值是"drop"。

23.23.3 多对一左连接

两个表的横向连接，经常是多对一连接，这用于从右表向左表中添加一些额外的变量。

例如，d.stu 中有学生学号、班级号、姓名、性别，d.cl 中有班级号、班主任名、年级，可以通过班级号将两个表连接起来：

```
d.stu <- tibble(
  sid=c(1,2,3,4,5,6),
  cid=c(1,2,1,2,1,2),
  sname=c("John", "Mary", "James", "Kitty", "Jasmine", "Kim"),
  sex=c("M", "F", "M", "F", "F", "M"))
knitr::kable(d.stu)
```

sid	cid	sname	sex
1	1	John	M
2	2	Mary	F
3	1	James	M
4	2	Kitty	F
5	1	Jasmine	F
6	2	Kim	M

```
d.cl <- tibble(
  cid=c(1,2),
  tname=c("Philip", "Joane"),
  grade=c("2017", "2016")
)
knitr::kable(d.cl)
```

cid	tname	grade
1	Philip	2017
2	Joane	2016

```
d.stu |>
  left_join(d.cl, join_by(cid)) |>
  knitr::kable()
```

sid	cid	sname	sex	tname	grade
1	1	John	M	Philip	2017
2	2	Mary	F	Joane	2016
3	1	James	M	Philip	2017
4	2	Kitty	F	Joane	2016
5	1	Jasmine	F	Philip	2017
6	2	Kim	M	Joane	2016

`left_join()` 按照 `join_by()` 指定的关键列匹配观测，左数据集所有观测不论匹配与否全部保留，右数据集仅使用与左数据集能匹配的观测。所以，结果数据集总是与左数据集行数相同，其观测也域左数据集的观测一一对应，仅用来添加部分右数据集中出现的变量。不指定关键列时，使用左、右数据集的共同列作为关键列。如果左右数据集关键列变量名不同，可以用 `by=c(" 左名"=" 右名")` 的格式。

类似地，`right_join()` 保留右数据集的所有观测，而仅保留左数据集中能匹配的观测。`full_join()` 保留所有观测。`inner_join()` 仅保留能匹配的观测。

23.23.4 利用第二个数据集筛选

`left_join()` 将右表中与左表匹配的观测的额外的列添加到左表中。如果希望按照右表筛选左表的观测，在左表中仅保留键值与右表键值匹配的观测，可以用 `semi_join()`，

函数 `anti_join()` 则与 `semi_join()` 相反，是要求从左表中删去与右表匹配的观测，仅保留与右表不匹配的观测。

这两种情况中右表仅使用关键列，右表中其它变量不起作用。

23.23.5 非相等条件的连接

前面的例子都是左、右表键值相等的连接。也可以将 `join_by(左键 == 右键)`，改成其它条件，如 `join_by(左键 <= 右键)`。

也可以要求左、右表不按照某个关键列连接，而是生成所有两两组合，这样，比如左表有 5 行，右表有 3 行，则结果有所有的 5×3 行两两组合。其写法是 `full_join()` 中加选项 `join_by()`，注意 `join_by()` 中没有关键列。

dplyr 包还支持其它的一些更复杂的连接，比如，满足条件的最近观测的连接（用 `join_by(closest(连接条件))`），还提供了进行区间判断的方便连接函数。

复杂的连接可以考虑安装一个支持 SQL 语言的扩展包，如 sqlite 扩展包，直接使用 SQL 语言解决问题。

23.24 数据集的集合操作

R 的 `intersect()`，`union()`, `setdiff()` 本来是以向量作为集合进行集合操作。dplyr 包也提供了这些函数，但是将两个 tibble 的各行作为元素进行集合操作。

23.25 标准化

设 x 是各列都为数值的列表 (包括数据框和 tibble) 或数值型矩阵，用 `scale(x)` 可以把每一列都标准化，即每一列都减去该列的平均值，然后除以该列的样本标准差。用 `scale(x, center=TRUE, scale=FALSE)` 仅中心化而不标准化。如

```
d.class |>
  select(height, weight) |>
  scale()
```

```
##            height      weight
##  [1,] -1.13843504 -0.70371312
##  [2,]  0.57794313 -0.08897522
##  [3,]  0.38290015 -0.44025402
##  [4,] -1.17744363 -1.01108207
##  [5,] -0.49479323 -0.68175819
##  [6,]  0.81199469  0.52576268
##  [7,] -2.15265850 -2.17469309
##  [8,]  0.03182280  0.54771760
##  [9,]  0.09033569  0.10861910
## [10,]  1.29960213  0.54771760
## [11,]  0.22686577  0.10861910
## [12,]  0.90951618  1.44786952
## [13,] -0.98240066 -0.74762297
## [14,]  0.03182280 -0.70371312
## [15,] -0.65082761 -0.02311045
## [16,]  1.88473105  2.19433697
## [17,]  0.48042164  1.22832027
## [18,] -0.94339207 -0.65980327
## [19,]  0.81199469  0.52576268
## attr(,"scaled:center")
##    height    weight
##  62.33684 100.02632
## attr(,"scaled:scale")
##    height    weight
##  5.127075 22.773933
```

为了把 `x` 的每列变到 $[0,1]$ 内，可以用如下的方法：

```
d.class %>%
  select(height, weight) %>%
  scale(center=apply(., 2, min),
      scale=apply(., 2, max) - apply(., 2, min))
```

其中的. 在%>% 管道操作中表示被传递处理的变量（一般是数据框）。也可以写一个自定义的进行零一标准化的函数：

```
scale01 <- function(x){
  mind <- apply(x, 2, min)
  maxd <- apply(x, 2, max)
  scale(x, center=mind, scale=maxd-mind)
}
d.class |>
  select(height, weight) |>
  scale01()
```

```
##          height    weight
##  [1,] 0.2512077 0.3366834
##  [2,] 0.6763285 0.4773869
##  [3,] 0.6280193 0.3969849
##  [4,] 0.2415459 0.2663317
##  [5,] 0.4106280 0.3417085
##  [6,] 0.7342995 0.6180905
##  [7,] 0.0000000 0.0000000
##  [8,] 0.5410628 0.6231156
##  [9,] 0.5555556 0.5226131
## [10,] 0.8550725 0.6231156
## [11,] 0.5893720 0.5226131
## [12,] 0.7584541 0.8291457
## [13,] 0.2898551 0.3266332
## [14,] 0.5410628 0.3366834
## [15,] 0.3719807 0.4924623
## [16,] 1.0000000 1.0000000
## [17,] 0.6521739 0.7788945
## [18,] 0.2995169 0.3467337
## [19,] 0.7342995 0.6180905
## attr(,"scaled:center")
## height weight
##   51.3   50.5
```

```
## attr(,"scaled:scale")
## height weight
##   20.7   99.5
```

函数 sweep() 可以执行对每列更一般的变换。

23.26 读入多个文件

设同类的数据放置在了多个文件中，这时可以利用多种技术，将多个文件读入并纵向合并，需要时增加文件名代表的变量信息。

作为举例，我们将 d.class 分为男生和女生两个数据框，分别保存到 data 子目录中：

```
d.class |>
  filter(sex == "F") |>
  select(-sex) |>
  write_csv(file = "data/class-F.csv")
d.class |>
  filter(sex == "M") |>
  select(-sex) |>
  write_csv(file = "data/class-M.csv")
```

如果要读入这两个文件并合并为一个大数据框，当然可以使用重复的写法：

```
dc1 <- read_csv(
  "data/class-F.csv",
  show_col_types = FALSE)
dc1[["sex"]] <- "F"
dc2 <- read_csv(
  "data/class-M.csv",
  show_col_types = FALSE)
dc2[["sex"]] <- "M"
dc <- rbind(dc1, dc2)
dc[c(1:2, 11:12), ]
```

```
## # A tibble: 4 x 5
##   name    age height weight sex
##   <chr> <dbl>  <dbl>  <dbl> <chr>
## 1 Alice    13   56.5    84  F
## 2 Becka    13   65.3    98  F
```

```
## 3 Duke     14   63.5   102. M
## 4 Guido    15   67     133  M
```

如果有许多个文件要合并，这样的硬编码方式就不够简洁，也容易出错。为此，可以联合使用许多技巧。

`list.files()` 函数可以指定某种文件名模式，获取所有要读入的文件名，如：

```
flis <- list.files(
  "data", pattern = "class-[[:alnum:]]+[.]csv")
flis
```

```
## [1] "class-F.csv" "class-M.csv"
```

其中的 `pattern` 选项输入了一个模式（正则表达式），见第42章。

可以用 `purrr::map()`（见 §25.2.1）将所有文件读入为一个数据框的列表，用 `purrr::list_rbind()` 函数将作为列表元素的数据框纵向合并：

```
dca <- file.path("data", flis) |>
  map(\(file) read_csv(file, show_col_types = FALSE)) |>
  list_rbind()
dca |>
  slice(c(1:2, 11:12))
```

```
## # A tibble: 4 x 4
##   name    age height weight
##   <chr> <dbl>  <dbl>  <dbl>
## 1 Alice    13   56.5    84
## 2 Becka    13   65.3    98
## 3 Duke     14   63.5   102.
## 4 Guido    15   67     133
```

这个程序的缺点是丢失了性别变量，这个变量的值保存在文件名内。为此，可以用 `purrr::set_names()` 给各个文件名指定元素名，这个函数的第二自变量可以直接输入元素名向量，也可以指定一个函数，从元素值产生元素名。用 `map()` 读入了数据框列表后，列表元素名会沿用文件名向量的元素名，然后在 `list_rbind()` 中可以用 `names_to` 选项，将数据框列表的元素名转换为合并后大数据框的一列：

```
dca2 <- file.path("data", flis) |>
  set_names(basename) |>
  map(\(file) read_csv(file, show_col_types = FALSE)) |>
  list_rbind(names_to = "filename")
dca2 |>
  slice(c(1:2, 11:12))
```

```
## # A tibble: 4 x 5
##   filename    name    age height weight
##   <chr>       <chr> <dbl>  <dbl>  <dbl>
## 1 class-F.csv Alice    13   56.5    84
## 2 class-F.csv Becka    13   65.3    98
## 3 class-M.csv Duke     14   63.5   102.
## 4 class-M.csv Guido    15   67     133
```

可以用适当的字符串处理函数提取文件名中的变量值，如：

```
dca3 <- file.path("data", flis) |>
  set_names(basename) |>
  map(\(file) read_csv(file, show_col_types = FALSE)) |>
  list_rbind(names_to = "filename") |>
  mutate(sex = str_sub(filename, 7,7)) |>
  select(-filename)
dca3 |>
  slice(c(1:2, 11:12))
```

```
## # A tibble: 4 x 5
##   name    age height weight sex
##   <chr> <dbl>  <dbl>  <dbl> <chr>
## 1 Alice    13   56.5    84  F
## 2 Becka    13   65.3    98  F
## 3 Duke     14   63.5   102. M
## 4 Guido    15   67     133  M
```

这里用了 `str_sub` 取子串的方法提取变量值，这适用于文件名中的变量值占据固定的位置的情形。更复杂的文件名，可以用正则表达式，或者 `tidyr::separate_wider_delim()` 指定分隔符分割成多列。

如：

```
dca4 <- file.path("data", flis) |>
  set_names(basename) |>
  map(\(file) read_csv(file, show_col_types = FALSE)) |>
  list_rbind(names_to = "filename") |>
  separate_wider_delim(
    filename, delim="-", names=c(NA, "sexext")) |>
  separate_wider_delim(
    sexext, delim=".", names=c("sex", NA))
dca4 |>
  slice(c(1:2, 11:12))
```

上面的程序先用 "-" 分割成两部分，再用 "." 将第二部分分割成两部分。

也可以直接使用正则表达式，如：

```
dca5 <- file.path("data", flis) |>
  set_names(basename) |>
  map(\(file) read_csv(file, show_col_types = FALSE)) |>
  list_rbind(names_to = "filename") |>
  mutate(
    sex = str_replace(
      filename, "class-([[:alnum:]]+)[.]csv", "\\1")  ) |>
  select(-filename)
dca5 |>
  slice(c(1:2, 11:12))
```

23.27 数据库访问与 SQL

R 支持与多种数据库连接，比如基于内存的 sqlite, duckdb, 基于本地文件的 arrow 数据库，基于客户/服务器的 MySQL，Oracle 等。见44。

SQL 是数据库的一种专用语言，可以很容易地实现数据查询、多个数据集连接等操作，见44.5。

Chapter 24

数据汇总

24.1 用 dplyr 作数据汇总

dplyr 包的 `summarise()` 函数可以对数据框计算统计量。

以肺癌病人化疗数据`cancer.csv`为例，有 34 个肺癌病人的数据：

```
d.cancer <- read_csv(
  "data/cancer.csv", locale=locale(encoding="GBK"))
```

```
## Rows: 34 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): sex, type
## dbl (4): id, age, v0, v1
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
knitr::kable(d.cancer)
```

id	age	sex	type	v0	v1
1	70	F	腺癌	26.51	2.91
2	70	F	腺癌	135.48	35.08
3	69	F	腺癌	209.74	74.44
4	68	M	腺癌	61.00	34.97
5	67	M	鳞癌	237.75	128.34
6	75	F	腺癌	330.24	112.34
7	52	M	鳞癌	104.90	32.10
8	71	M	鳞癌	85.15	29.15
9	68	M	鳞癌	101.65	22.15
10	79	M	鳞癌	65.54	21.94
11	55	M	腺癌	125.31	12.33
12	54	M	鳞癌	224.36	99.44
13	55	F	腺癌	12.93	2.30
14	75	M	腺癌	40.21	23.96
15	61	F	腺癌	12.58	7.39
16	76	M	鳞癌	231.04	112.58
17	65	M	鳞癌	172.13	91.62
18	66	M	鳞癌	39.26	13.95
19	NA	F	腺癌	32.91	9.45
20	63	F	腺癌	161.00	122.45
21	67	M	鳞癌	105.26	68.35
22	51	M	鳞癌	13.25	5.25
23	49	M	鳞癌	18.70	3.34
24	49	M	鳞癌	60.23	50.36
25	NA	F	鳞癌	223.00	25.59
26	NA	M	鳞癌	145.70	35.36
27	NA	M	鳞癌	138.44	11.30
28	NA	M	鳞癌	83.71	76.45
29	NA	M	鳞癌	74.48	23.66
30	NA	F	腺癌	42.70	5.97
31	NA	M	鳞癌	142.48	68.46
32	NA	F	鳞癌	46.97	27.32
33	NA	F	鳞癌	170.63	74.76
34	NA	F	鳞癌	67.37	54.52

求年龄 (age) 的观测个数、非缺失观测个数、平均值、标准差：

```
d.cancer |>
  summarise(
    nobs = n(),
    n = sum(!is.na(age)),
    mean.age=mean(age, na.rm=TRUE),
    sd.age=sd(age, na.rm=TRUE)) |>
  knitr::kable()
```

nobs	n	mean.age	sd.age
34	23	64.13043	9.161701

其中 `n()` 计算观测数（行数）。为了计算某列的非缺失值个数，用 `sum(!is.na(x))`。

`summarise` 中常用的汇总函数有：

- 位置度量：`mean()`, `median()`。
- 分散程度（变异性）度量：`sd()`, `IQR()`, `mad()`。
- 分位数：`min()`, `max()`, `quantile()`。
- 按下标查询，如 `first(x)` 取出 `x[1]`，`last(x)` 取出 `x` 的最后一个元素，`nth(x,2)` 取出 `x[2]`。可以提供一个缺省值以防某个下标位置不存在。
- 计数：`n()` 给出某个组的观测数，`sum(!is.na(x))` 统计 `x` 的非缺失值个数，`n_distinct(x)` 统计 `x` 的不同值个数 (缺失值也算一个值)，也可以计算多个变量的不同组合值个数。
- `count(x)` 给出 `x` 的每个不同值的个数（类似于 `table()` 函数），但 `count(x)` 只能单独作为管道运算的一个步骤，不能用在 `summarise` 函数中。

这里有些函数是 dplyr 包提供的，仅适用于 tibble 类型。

`count()` 函数允许用 `wt` 添加一个权重，实际是重复观测数。例如，有如下的汇总数据：

```
dwt <- tribble(
  ~age, ~n, ~meanheight,
  10, 20, 110,
  10, 10, 112,
  11, 25, 120
)
knitr::kable(dwt)
```

age	n	meanheight
10	20	110
10	10	112
11	25	120

其中 `n` 表示该观测代表的人数。汇总两个年龄的人数：

```
dwt |> count(age, wt=n)
```

```
## # A tibble: 2 x 2
##     age     n
##   <dbl> <dbl>
## 1    10    30
## 2    11    25
```

计算所有人的总数和平均身高：

```
dwt |>
  summarise(
    nobs = sum(n),
    meanheight = sum(meanheight*n)/sum(n)
  )
```

```
## # A tibble: 1 x 2
##    nobs meanheight
##   <dbl>      <dbl>
## 1    55       115.
```

```
## 114.9091
```

基本 R 和 dplyr 没有提供对加权计算的一般支持。

24.2 多个变量的汇总

在有多个变量需要汇总时，summarise 的格式就会比较罗嗦。比如，需要对 cancer 数据集中 v0 和 v1 两个变量同时计算平均值和标准差：

```
d.cancer |>
  summarise(
    mean_v0=mean(v0, na.rm=TRUE),
    sd_v0 = sd(v0, na.rm=TRUE),
    mean_1=mean(v1, na.rm=TRUE),
    sd_v1 = sd(v1, na.rm=TRUE)) |>
  knitr::kable(digits=2)
```

mean_v0	sd_v0	mean_1	sd_v1
110.08	79.52	44.69	38.44

显然，如果有许多变量要计算不止一个统计量，就需要人为地将每一个变量的每一个统计量单独命名。dplyr 包的 `across()` 函数可以指定一批变量名与一批统计函数，自动命名结果变量。如果仅计算一个统计量，只要在 `across` 中给出要分析的变量集合与一个统计函数名，如：

```
d.cancer |>
  summarise(
    across(c("v0", "v1", "age"), mean)) |>
  knitr::kable(digits=2)
```

v0	v1	age
110.08	44.69	NA

其中 `age` 的平均值得到了缺失值。在指定函数时，可以用 R 的无名函数方法，增加选项：

```
d.cancer |>
  summarise(
    across(
      c("v0", "v1", "age"),
      \(x) mean(x, na.rm=TRUE))) |>
  knitr::kable(digits=2)
```

v0	v1	age
110.08	44.69	64.13

可以看出，这样仅指定函数名或无名函数的方法，仅限于一种统计量，而且统计量结果名称也沿用了变量名本身。调用 `across` 较好的方法，是将要计算的统计量写在一个有名列表中，列表的元素名将出现在结果的变量名中作为统计量名部分，列表的元素是函数名或无名函数。如：

```
d.cancer |>
  summarise(
    across(c("v0", "v1", "age"),
           list(
             avg = \(x) mean(x, na.rm=TRUE),
             std = \(x) sd(x, na.rm=TRUE)))) |>
  knitr::kable(digits=2)
```

v0_avg	v0_std	v1_avg	v1_std	age_avg	age_std
110.08	79.52	44.69	38.44	64.13	9.16

`across()` 的第一自变量是要分析的变量集合，可以使用 `dplyr::select()` 的变量格式，如 `v0`, `c(v0, v1)`, `c("v0", "v1")`, `starts_with("v")`, `5:6` 等。

`across()` 的第二自变量：

- 可以是一个函数，如 mean；
- 可以是一个如\(x) mean(x, na.rm=TRUE) 这样的自定义无名函数；
- 函数或者自定义无名函数的有名列表，可以计算所有变量的所有统计函数值并对结果自动命名，结果统计量自动命名为“变量名 _ 统计量名”的格式。

如果要对所有数值型变量计算某些统计量，可以用

```
summarise(across(
  where(is.numeric),
  list(
    变量后缀 = \(x) 统计函数名 (x, <...>),
    <...>)))
```

这样的格式。如：

```
d.cancer |>
  summarise(
    across(where(is.numeric),
    list(avg = \(x) mean(x, na.rm=TRUE),
         std = \(x) sd(x, na.rm=TRUE)))) |>
  knitr::kable(digits=2)
```

id_avg	id_std	age_avg	age_std	v0_avg	v0_std	v1_avg	v1_std
17.5	9.96	64.13	9.16	110.08	79.52	44.69	38.44

其中的 is.numeric 用来筛选需要进行统计的变量子集，可以替换成其它的示性函数（对每列返回一个逻辑值的函数），如 is.character, is.Date, is.POSIXct, is.logical，is.factor。

为了对所有列都计算相同的统计量，可以在 across() 中指定变量集合为 everything()，如

```
d.cancer |>
  select(v0, v1) |>
  summarise(across(everything(),
    list(avg = \(x) mean(x, na.rm=TRUE),
         std = \(x) sd(x, na.rm=TRUE))) ) |>
  knitr::kable(digits=2)
```

v0_avg	v0_std	v1_avg	v1_std
110.08	79.52	44.69	38.44

可以用 tidyr 包将上面的变量与统计量拆开，将变量放到不同的观测中，如：

```
d.cancer |>
  select(v0, v1) |>
  summarise(across(everything(),
    list(avg = \(x) mean(x, na.rm=TRUE),
         std = \(x) sd(x, na.rm=TRUE))) ) |>
  pivot_longer(
    everything(),
    names_sep ="_",
    names_to = c("variable", ".value")
    ) |>
  knitr::kable(digits=2)
```

variable	avg	std
v0	110.08	79.52
v1	44.69	38.44

24.2.1 修改多个变量

除了对多个变量进行汇总以外，有时需要对多个变量进行修改。例如，某个数据集中所有的数值型变量都用-999 表示缺失值，需要将其改为 NA，如果在 mutate 中逐个写出修改，就太麻烦，这时可以在 mutate() 中用 across() 指定要修改的列和修改方法，如：

```
d.999.orig <- tribble(
  ~id, ~a, ~b, ~c,
  1, 5, 10, 6,
  2, -999, 12, 8,
  3, 4, -999, -999,
  4, 7, 9, 10
)
d.999.mod <- d.999.orig |>
  mutate(across(
    a:c,
    \(x) ifelse(x == -999, NA, x)
  ))
d.999.mod
```

```
## # A tibble: 4 x 4
##      id     a     b     c
##   <dbl> <dbl> <dbl> <dbl>
## 1     1     5    10     6
```

```
## 2     2    NA    12     8
## 3     3     4    NA    NA
## 4     4     7     9    10
```

修改结果直接替换了原来的变量。如果希望修改结果用新变量保存，可以在 across 中用.names 指定一个新变量命名模式，如

```
d.999 <- d.999.orig |>
  mutate(across(
    a:c,
    \(x) ifelse(x == -999, NA, x),
    .names = "{.col}_na"
  ))
d.999
```

```
## # A tibble: 4 x 7
##      id     a     b     c  a_na  b_na  c_na
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     1     5    10     6     5    10     6
## 2     2  -999    12     8    NA    12     8
## 3     3     4  -999  -999     4    NA    NA
## 4     4     7     9    10     7     9    10
```

其中.names 的取值是一个模式字符串，模式中的 {.col} 特指原始变量名。

下面的函数 expand_dates()，输入一个数据框，选择其中的所有日期列，将其每一个都拆分为年、月、日：

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
expand_dates <- function(df){
  df |>
    mutate(across(
      where(is.Date),
      list(year = year, month = month, day = mday)
```

```
    ))
}
```

```
d.dates <- read_csv(
  'data/dates.csv',
  locale=locale(encoding="GBK"),
  col_types=cols(
    `序号`=col_integer(),
    `出生日期`=col_date(format="%Y/%m/%d"),
    `发病日期`=col_date(format="%Y/%m/%d")
  ))
knitr::kable(d.dates[1:4,])
```

序号	出生日期	发病日期
1	1941-03-08	2007-01-01
2	1972-01-24	2007-01-01
3	1932-06-01	2007-01-01
4	1947-05-17	2007-01-01

```
d.dates2 <- expand_dates(d.dates)
knitr::kable(d.dates2[1:4,])
```

序号	出生日期	发病日期	出生日期 _year	出生日期 _month	出生日期 _day	发病日期 _year	发病日期 _m
1	1941-03-08	2007-01-01	1941	3	8	2007	
2	1972-01-24	2007-01-01	1972	1	24	2007	
3	1932-06-01	2007-01-01	1932	6	1	2007	
4	1947-05-17	2007-01-01	1947	5	17	2007	

24.2.2 针对多个变量进行筛选

across 的类似函数也可以用在 filter 函数中，同时指定多个变量进行筛选。if_any 指定是任一变量满足条件就选择该观测，if_all 指定所有变量满足该该条件才选择该观测。如：

```
d.999 |>
  filter(if_any(a_na:c_na, is.na))
```

```
## # A tibble: 2 x 7
##      id     a     b     c  a_na  b_na  c_na
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     2  -999    12     8    NA    12     8
## 2     3     4  -999  -999     4    NA    NA
```

```
d.999 |>
  filter(if_all(a_na:c_na, \(x) !is.na(x)))
```

```
## # A tibble: 2 x 7
##      id     a     b     c  a_na  b_na  c_na
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     1     5    10     6     5    10     6
## 2     4     7     9    10     7     9    10
```

24.3 用 dplyr 作数据分组汇总

数据汇总问题更常见的是作分组汇总。dplyr 包的 `group_by()` 函数对数据框（或 tibble）分组，随后的 `summarise()` 将按照分组汇总。`group_by()` 对数据框增加分组信息，使得后续的管道操作针对每一组内进行操作，但 `group_by()` 并不修改数据框内容。

24.3.1 简单汇总

比如，按不同性别分组计算人数与年龄平均值：

```
d.cancer |>
  group_by(sex) |>
  summarise(
    count = n(),
    mean.age = mean(age, na.rm=TRUE)) |>
  knitr::kable(digits=2)
```

sex	count	mean.age
F	13	66.14
M	21	63.25

按不同性别分组计算 v0 和 v1 的人数、平均值、标准差：

```
d.cancer |>
  group_by(sex) |>
  summarise(across(
    c("v0", "v1"),
    list(
      count = ~n(),
      avg = \(x) mean(x, na.rm=TRUE),
```

```
      std = \(x) sd(x, na.rm=TRUE))) ) |>
  knitr::kable(digits=2)
```

sex	v0_count	v0_avg	v0_std	v1_count	v1_avg	v1_std
F	13	113.24	100.07	13	42.66	41.72
M	21	108.12	66.45	21	45.96	37.28

上面的程序应该将 v0_count 和 v1_count 合并，但是 summarise 在有多个变量和多个统计量时总会将变量两两搭配，确实需要时可以分别统计再合并，结果程序会略繁琐一些：

```
bind_cols(
  d.cancer |>
    group_by(sex) |>
    summarise(count = n() ),
  d.cancer |>
    group_by(sex) |>
    summarise(
      across(c("v0", "v1"),
      list(
        avg = \(x) mean(x, na.rm=TRUE),
        std = \(x) sd(x, na.rm=TRUE)))) |>
    select(-sex)
  ) |>
  knitr::kable(digits=2)
```

sex	count	v0_avg	v0_std	v1_avg	v1_std
F	13	113.24	100.07	42.66	41.72
M	21	108.12	66.45	45.96	37.28

24.3.2 满足条件汇总

为了查询数值型变量取值满足某种条件的个数和比例，可以将该条件用 sum() 和 mean() 函数统计，比如，对男女两组分别计算年龄在 60 岁以上的人数和比例：

```
d.cancer |>
  group_by(sex) |>
  summarise(
    nold = sum(age >= 60, na.rm=TRUE),
    pold = nold / n()) |>
  knitr::kable(digits=4)
```

sex	nold	pold
F	6	0.4615
M	10	0.4762

24.3.3 分组筛选

用 `group_by()` 分组后除了可以分组汇总，还可以分组筛选：

```
d.cancer |>
  group_by(sex) |>
  filter(rank(desc(v0)) <= 2) |>
  arrange(sex, desc(v0)) |>
  knitr::kable()
```

id	age	sex	type	v0	v1
6	75	F	腺癌	330.24	112.34
25	NA	F	鳞癌	223.00	25.59
5	67	M	鳞癌	237.75	128.34
16	76	M	鳞癌	231.04	112.58

以上程序按性别分组后，在每组中找出疗前体积排名在前两名的。

24.3.4 分组计算新变量

在分组后也可以根据每组的统计量用 `mutate()` 定义新变量。比如，每组的 v0, v1 除以本组的最大值（结果略）：

```
d.cancer |>
  group_by(sex) |>
  mutate(
    v0s = v0 / max(v0),
    v1s = v1 / max(v1))
```

24.3.5 按前后变化分组

有时我们的数据中行次序是有意义的，可能需要根据某个变量的变化来分组。如：

```
d.cid <- tribble(
  ~state, ~y,
  10, 1.5,
  10, 1.6,
  10, 1.8,
```

```
  20, 2.3,
  20, 2.2,
  10, 1.1,
  10, 1.2
)
knitr::kable(d.cid)
```

state	y
10	1.5
10	1.6
10	1.8
20	2.3
20	2.2
10	1.1
10	1.2

按 state 的变化生成一个分组变量：

```
d.cid |>
  mutate(
    grp = consecutive_id(state)) |>
  knitr::kable()
```

可以直接按临时生成的分组变量分组汇总，如：

```
d.cid |>
  group_by(grp = consecutive_id(state)) |>
  slice_head(n=1) |>
  knitr::kable()
```

24.4 交叉分类的汇总

24.4.1 简单例子

下面的程序对 d.cancer 数据框分性别与病理类型分别统计人数：

```
d.cancer |>
  group_by(sex, type) |>
  summarise(freq=n()) |>
  knitr::kable()
```

```
## `summarise()` has grouped output by 'sex'. You can override using the `.groups`
## argument.
```

sex	type	freq
F	鳞癌	4
F	腺癌	9
M	鳞癌	18
M	腺癌	3

如果仅需要计算交叉分类频数，不需要用 `group_by()`，可以用 dplyr 的 `count()` 函数，如：

```
d.cancer |>
  count(sex, type) |>
  knitr::kable()
```

sex	type	n
F	鳞癌	4
F	腺癌	9
M	鳞癌	18
M	腺癌	3

下面的程序数出 NHANES 数据框中 ID(受访者编码) 与 SurveyYr(考察年份) 每一对组合的出现次数, 筛选出二次及以上者，并降序排列，仅显示前 10 行结果：

```
NHANES |>
  count(ID, SurveyYr) |>
  filter(n >= 2 ) |>
  arrange(desc(n)) |>
  slice_head(n = 10) |>
  knitr::kable()
```

ID	SurveyYr	n
70324	2011_12	8
62927	2011_12	7
63297	2011_12	7
69626	2011_12	7
60566	2009_10	6
61442	2009_10	6
63163	2011_12	6
63330	2011_12	6
63390	2011_12	6
63744	2011_12	6

24.4.2 分组信息残留问题

用 group_by() 交叉分组汇总后的结果不是普通的 tibble，总是带有外层分组信息，最内层的分组信息不再有效。如果有更多的分组变量，每次用 summarise() 汇总都会去掉最内层的分组信息，但保留其它层的分组信息。不注意这种规定在后续的使用中可能会产生问题，为此，可以在 summarise() 函数中加选项.groups="drop" 要求汇总后不保留任何分组信息，或者汇总后再用 ungroup() 函数取消所有分组。

例如，希望在用 group_by() 按照性别和病理类别交叉分类计算频数后求所有病人的总人数，用了如下程序：

```
d.cancer |>
  group_by(sex, type) |>
  summarise(freq=n()) |>
  summarise(ntotal=sum(freq)) |>
  knitr::kable()
```

```
## `summarise()` has grouped output by 'sex'. You can override using the `.groups`
## argument.
```

sex	ntotal
F	13
M	21

可以看出并没有能够计算总人数，而是按原来交叉分类的外层分类即性别分组计算了总人数。这是因为交叉分组计算频数后的结果仍按照外层分类变量 sex 分组，所以 summarise(ntotal=sum(freq)) 也是分两组进行的。在第一个 summarise() 中加.groups = "drop" 可以不保留分组信息，如：

```
d.cancer |>
  group_by(sex, type) |>
  summarise(freq=n(), .groups="drop") |>
  summarise(ntotal=sum(freq)) |>
  knitr::kable()
```

ntotal
34

得到了需要的结果。

为了避免在汇总后仍保留分组信息，在 summarise() 中加选项.groups="drop"，这样可以在汇总结果中去掉残留的分组信息。默认的选项是.groups="drop_last"，即仅去掉最内层分组信息。还可以要求.groups="keep"，这时汇总结果中包含所有的分组信息，包括最内层分组。

用 ungroup() 去掉所有分组信息的例子如下：

```
d.cancer |>
  group_by(sex, type) |>
  summarise(freq=n()) |>
  ungroup() |>
  summarise(ntotal=sum(freq)) |>
  knitr::kable()
```

```
## `summarise()` has grouped output by 'sex'. You can override using the `.groups`
## argument.
```

ntotal
34

24.5 在自定义函数中使用 dplyr 功能

如果对数据框有一些通用的操作，可以编写函数，以数据框为输入和输出，指定针对的变量集合与要进行的操作。

因为在 `mutate` 和 `summarise` 中需要使用直接写的变量名，以字符型变量形式输入的变量名无法直接使用。

实际上，dplyr 和 tidyr 关于变量和变量集合使用两种写法类似、底层不同的做法，即数据掩码 (data masking) 和整洁选择 (tidy select)。`mutate()`, `filter()`, `arrange()`, `count()`, `group_by()`, `summarise()` 在底层使用数据掩码方法。`select()`, `relocate()`, `rename()`, `pull()`, `across()` 在底层使用整洁选择方法。

在使用数据掩码方法的函数中，如果针对的数据框变量是原样（不是字符串或者字符型变量）输入的，可以用 `{{ var_names }}` 的写法表示。比如，输入一个分组变量和一个分析变量，计算均值和最大值：

```
group_summ_1 <- function(df, group_var, ana_var){
  df |>
    group_by({{ group_var }}) |>
    summarise(
      mean = mean({{ana_var}}, na.rm=TRUE),
      max = max({{ana_var}}, na.rm=TRUE))
}
group_summ_1(d.class, sex, height)
```

```
## # A tibble: 2 x 3
##   sex    mean   max
##   <fct> <dbl> <dbl>
## 1 M      63.9  72
## 2 F      60.6  66.5
```

上面的 group_by 和 summarise 都是数据掩码方式指定变量的，所以当调用自定义函数时如果原样写分组变量和分析变量名，而不是写成字符串或者输入为字符型变量，就只要用 {{ var_name }} 这种格式。但是，如果输入为字符串则不行：

```
group_summ_1(d.class, "sex", "height")
## # A tibble: 1 × 3
##   `"sex"`  mean max
##   <chr>   <dbl> <chr>
## 1 sex        NA height
```

这时，可以用.data[[var_name]] 这种方式访问，如：

```
group_summ_2 <- function(df, group_var, ana_var){
  df |>
    group_by(.data[[group_var]]) |>
    summarise(
      mean = mean(.data[[ana_var]], na.rm=TRUE),
      max = max(.data[[ana_var]], na.rm=TRUE))
}
group_summ_2(d.class, "sex", "height")
```

```
## # A tibble: 2 x 3
##   sex    mean   max
##   <fct> <dbl> <dbl>
## 1 M      63.9  72
## 2 F      60.6  66.5
```

基于数据掩码的函数在指定多个变量时，为了能够使用整洁选择时可用的方法，需要用 pick() 函数进行说明。比如，group_by() 是基于数据掩码的，如果要指定多个变量，无法直接使用如 c(sex, age) 这样的方法，可以在自定义函数中写成 pick({{ group_var }})，如：

```
group_summ_3 <- function(df, group_var, ana_var){
  df |>
    group_by(pick({{ group_var }})) |>
    summarise(
      mean = mean({{ ana_var }}, na.rm=TRUE),
      max = max({{ ana_var }}, na.rm=TRUE))
}
group_summ_3(d.class, c(sex, age), height)
```

select, across 等函数在指定变量时，使用整洁选择方式。如果变量是用整洁选择函数或者原样变量名方式输入的，仍可以用 {{ var_names }} 这种格式。

例如，下面的函数输入一个数据框，输入一个分组用的变量名，输入要计算汇总统计量的数值型变量，输出分组汇总结果的数据框：

```
group_summ_4 <- function(df, group_var, ana_vars){
  df |>
    group_by({{ group_var }}) |>
    summarise(across(
      {{ ana_vars }},
      list(avg = ~ mean(., na.rm=TRUE),
           max = ~ max(., na.rm=TRUE))))
}
group_summ_4(d.class, sex, c(height, weight))
```

```
## # A tibble: 2 x 5
##   sex   height_avg height_max weight_avg weight_max
##   <fct>      <dbl>      <dbl>      <dbl>      <dbl>
## 1 M           63.9       72        109.        150
## 2 F           60.6       66.5       90.1       112.
```

其中的 group_by 是基于数据掩码的，across 是基于整洁选择的。在 across 中如果输入字符型变量，则不能用 {{ var_names }} 这种方法，而是用 all_of(var_names)，如：

```
group_summ_5 <- function(df, group_var, ana_vars){
  df |>
    group_by({{ group_var }}) |>
    summarise(across(
      all_of(ana_vars),
      list(avg = ~ mean(., na.rm=TRUE),
           max = ~ max(., na.rm=TRUE))))
}
ana_vars <- c("height", "weight")
group_summ_5(d.class, sex, ana_vars)
```

```
## # A tibble: 2 x 5
##   sex   height_avg height_max weight_avg weight_max
##   <fct>      <dbl>      <dbl>      <dbl>      <dbl>
## 1 M           63.9       72        109.        150
## 2 F           60.6       66.5       90.1       112.
```

在自定义函数中使用 dplyr 功能，可参考Programming with dplyr。在自定义函数中使用 tidyr 功能，可参考programming with tidyr。

24.6 数据分析流程控制

一个完整的数据分析过程，经常需要反复地修改程序，更新数据。如果每次都重新运行整个的数据整理、概括、建模、结果呈现的代码，会花费很多时间，中间某一步骤出错时如果修改程序重新运行，就使得前面步骤白作，如果人为控制重新运行哪些代码，容易犯错使得利用了旧的结果。

解决这样的问题的工具是 make 类的软件，这些软件规定各个任务之间的依赖关系，当任务依赖的数据和代码更新时自动管理需要重新运行哪些程序。R 提供了 targets 包可以实现这样的任务管理功能，规定了数据、代码、结果的依赖关系后可以仅运行受到影响的部分。

参见 https://docs.ropensci.org/targets/。

24.7 基本 R 的汇总功能

24.7.1 summary() 函数

对数值型向量 `x`，用 `summary(x)` 可以获得变量的平均值、中位数、最小值、最大值、四分之一和四分之三分位数。如

```
summary(d.cancer[["v0"]])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   12.58   43.77   93.40  110.08  157.18  330.24
```

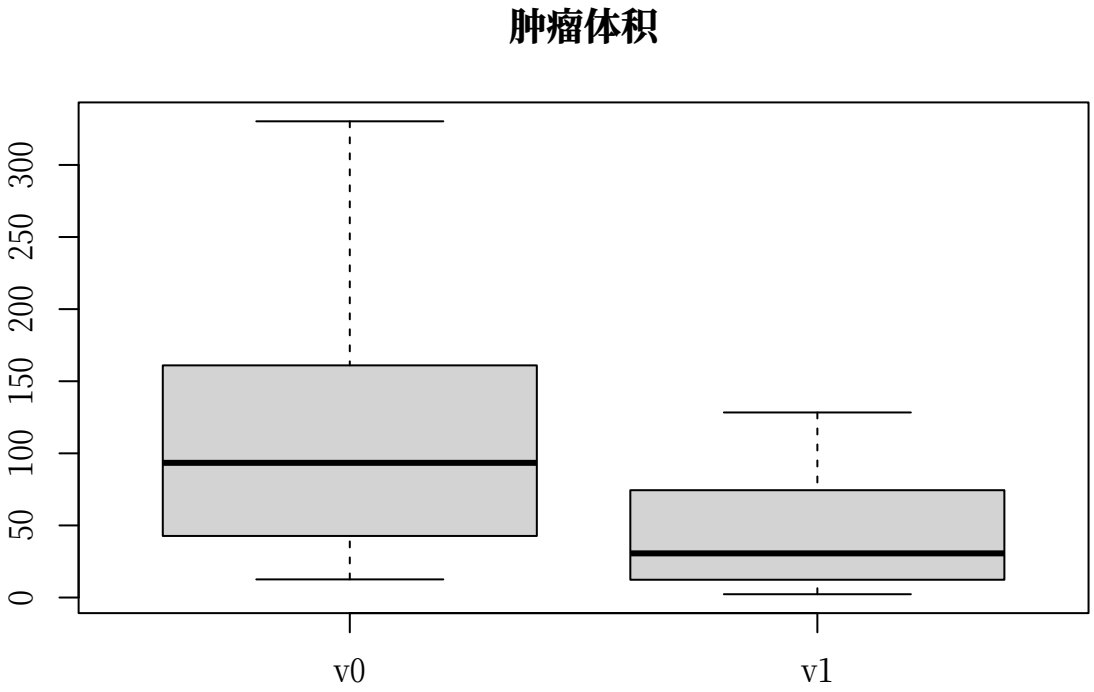
```
summary(d.cancer[["v1"]])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.30   12.73   30.62   44.69   72.94  128.34
```

v0 是放疗前的肿瘤体积，v1 是放疗后的体积，可以看出放疗后体积减小了很多。

可以用盒形图表现类似的信息，如

```
boxplot(d.cancer[,c("v0", "v1")], main=" 肿瘤体积")
```



对一个数据框 d，用 summary(d) 可以获得每个连续型变量的基本统计量，和每个离散取值变量的频率。如

```
summary(d.cancer)
```

```
##        id             age            sex                type
##  Min.   : 1.00   Min.   :49.00   Length:34          Length:34
##  1st Qu.: 9.25   1st Qu.:55.00   Class :character   Class :character
##  Median :17.50   Median :67.00   Mode  :character   Mode  :character
##  Mean   :17.50   Mean   :64.13
##  3rd Qu.:25.75   3rd Qu.:70.00
##  Max.   :34.00   Max.   :79.00
##                  NA's   :11
##        v0               v1
##  Min.   : 12.58   Min.   :  2.30
##  1st Qu.: 43.77   1st Qu.: 12.73
##  Median : 93.40   Median : 30.62
##  Mean   :110.08   Mean   : 44.69
##  3rd Qu.:157.18   3rd Qu.: 72.94
##  Max.   :330.24   Max.   :128.34
##
```

对数据框 d，用 str(d) 可以获得各个变量的类型和取值样例。如

```
str(d.cancer)
```

```
## spec_tbl_df [34 x 6] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ id  : num [1:34] 1 2 3 4 5 6 7 8 9 10 ...
##  $ age : num [1:34] 70 70 69 68 67 75 52 71 68 79 ...
##  $ sex : chr [1:34] "F" "F" "F" "M" ...
##  $ type: chr [1:34] "腺癌" "腺癌" "腺癌" "腺癌" ...
##  $ v0  : num [1:34] 26.5 135.5 209.7 61 237.8 ...
##  $ v1  : num [1:34] 2.91 35.08 74.44 34.97 128.34 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   id = col_double(),
##   ..   age = col_double(),
##   ..   sex = col_character(),
##   ..   type = col_character(),
##   ..   v0 = col_double(),
##   ..   v1 = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

用 head(d) 可以返回数据框（或向量、矩阵）的前几行，用 tail(d) 可以返回数据框的后几行。

24.7.2 连续型变量概括函数

对连续取值的变量 x，可以用 mean, std, var, sum, prod, min, max 等函数获取基本统计量。加 na.rm=TRUE 选项可以仅对非缺失值计算。

sort(x) 返回排序后的结果。rev(x) 把 x 所有元素次序颠倒后返回。quantile(x, c(0.05, 0.95)) 可以求 x 的样本分位数。rank(x) 对 x 求秩得分（即名次，但从最小到最大排列）。

24.7.3 分类变量概括

分类变量一般输入为因子。对因子或其它向量 x，table(x) 返回 x 的每个不同值的频率（出现次数），结果为一个类（class）为 table 的一维数组。每个元素有对应的元素名，为 x 的各水平值。如

```
res <- table(d.cancer[["sex"]]); res
```

```
##
##  F  M
## 13 21
```

```
res["F"]
```

```
##  F
## 13
```

对单个分类变量, table 结果是一个有元素名的向量。用 as.data.frame() 函数把 table 的结果转为数据框:

```
as.data.frame(res)
```

```
##   Var1 Freq
## 1    F   13
## 2    M   21
```

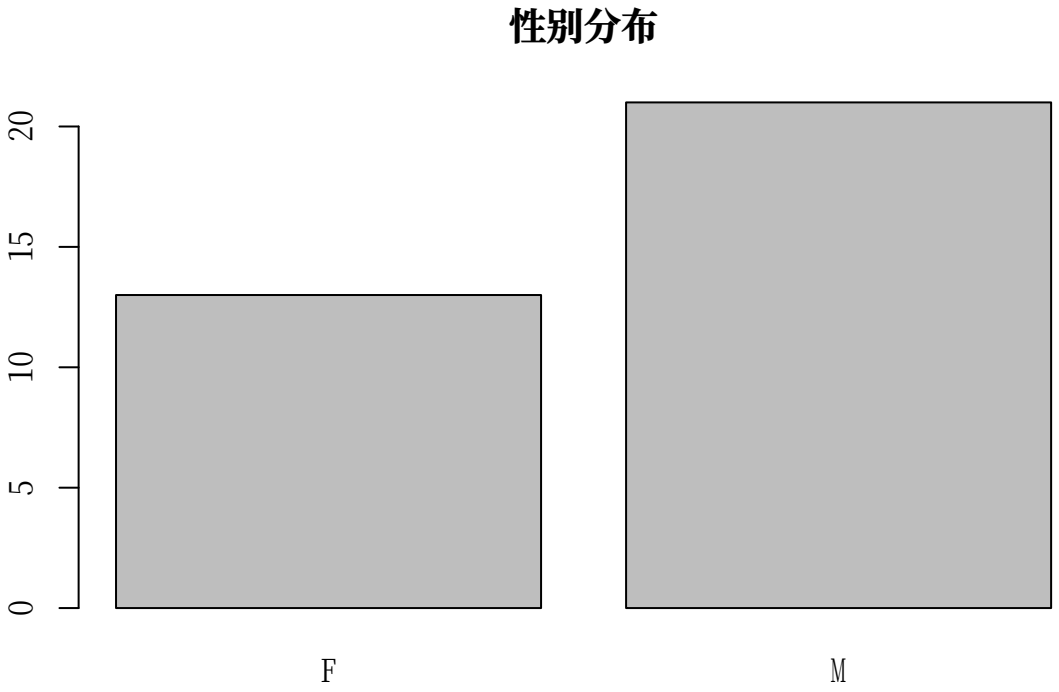
用 prop.table() 将频数转换成百分比:

```
prop.table(res)
```

```
##
##         F         M
## 0.3823529 0.6176471
```

table 作的单变量频数表可以用 barplot 表现为图形，如:

```
barplot(res, main=" 性别分布")
```



对两个分类变量 x1 和 x2，其每个组合的出现次数可以用 table(x1,x2) 函数统计，结果叫做列联表。如

```
res2 <- with(d.cancer, table(sex, type)); res2
```

```
##    type
## sex 鳞癌 腺癌
##   F    4    9
##   M   18    3
```

结果是一个类为 table 的二维数组（矩阵），每行以第一个变量 x1 的各水平值为行名，每列以第二个变量 x2 的各水平值为列名。这里用了 with() 函数引入一个数据框，后续的参数中的表达式可以直接使用数据框的变量。

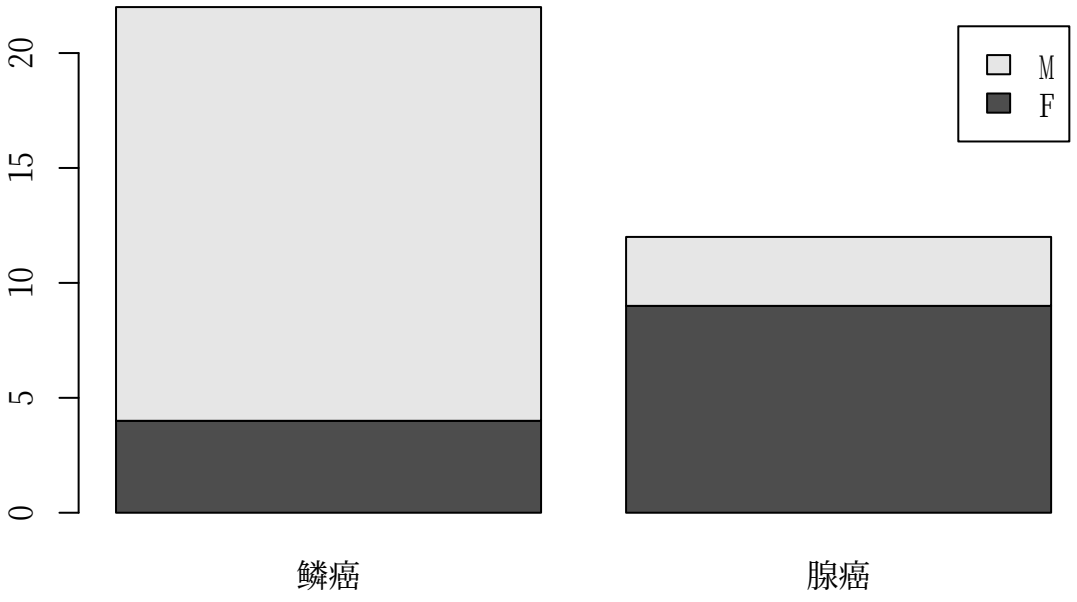
对两个分类变量, table 结果是一个矩阵。用 as.data.frame 函数把 table 的结果转为数据框:

```
as.data.frame(res2)
```

```
##   sex type Freq
## 1   F 鳞癌    4
## 2   M 鳞癌   18
## 3   F 腺癌    9
## 4   M 腺癌    3
```

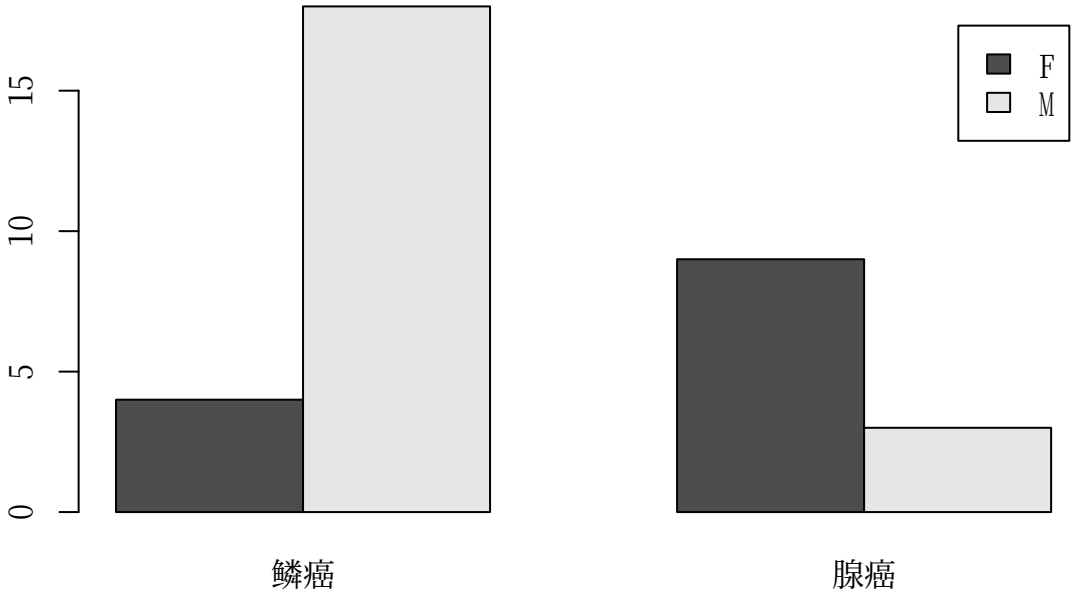
列联表的结果可以用条形图表示。如

```
barplot(res2, legend=TRUE)
```



或

```
barplot(res2, legend=TRUE, beside=TRUE)
```



对于 table() 的结果列联表，可以用 addmargins() 函数增加行和与列和。如

```
addmargins(res2)
```

```
##      type
## sex   鳞癌 腺癌 Sum
##   F      4    9  13
##   M     18    3  21
##   Sum   22   12  34
```

用 margin.table() 可以计算列联表行或列的和并返回，如

```
margin.table(res2, 1)
```

```
## sex
##  F  M
## 13 21
```

```
margin.table(res2, 2)
```

```
## type
## 鳞癌 腺癌
##   22   12
```

用 prop.table(r) 把一个列联表 r 转换成百分比表。如

```
prop.table(res2)
```

```
##    type
## sex       鳞癌       腺癌
##   F 0.11764706 0.26470588
##   M 0.52941176 0.08823529
```

用 prop.table(res,1) 把列联表 res 转换成行百分比表。用 prop.table(res,2) 把列联表 res 转换成列百分比表。如

```
prop.table(res2, 1)
```

```
##    type
## sex      鳞癌      腺癌
##   F 0.3076923 0.6923077
##   M 0.8571429 0.1428571
```

```
prop.table(res2, 2)
```

```
##    type
## sex      鳞癌      腺癌
##   F 0.1818182 0.7500000
##   M 0.8181818 0.2500000
```

在有多个分类变量时，用 as.data.frame(table(x1, x2, x3)) 形成多个分类变量交叉分类的频数统计数据框。dplyr 包的 count() 功能与 table() 类似。

24.7.4 数据框概括

用 colMeans() 对数据框或矩阵的每列计算均值，用 colSums() 对数据框或矩阵的每列计算总和。用 rowMeans() 和 rowSums() 对矩阵的每行计算均值或总和。

数据框与矩阵有区别，某些适用于矩阵的计算对数据框不适用，例如矩阵乘法。用 `as.matrix()` 把数据框的数值子集转换成矩阵。

对矩阵，用 `apply(x, 1, FUN)` 对矩阵 x 的每一行使用函数 FUN 计算结果，用 `apply(x, 2, FUN)` 对矩阵 x 的每一列使用函数 FUN 计算结果。

如果 `apply(x,1,FUN)` 中的 FUN 对每个行变量得到多个 m 结果，结果将是一个矩阵，行数为 m，列数等于 nrow(x)。如果 `apply(x,2,FUN)` 中的 FUN 对每个列变量得到多个 m 结果，结果将是一个矩阵，行数为 m，列数等于 ncol(x)。例如：

```
apply(as.matrix(iris[,1:4]), 2,
        function(x)
        c(n=sum(!is.na(x)),
          mean=mean(x, na.rm=TRUE),
          sd=sd(x, na.rm=TRUE)))
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## n     150.0000000 150.0000000   150.000000 150.0000000
## mean    5.8433333   3.0573333     3.758000   1.1993333
## sd      0.8280661   0.4358663     1.765298   0.7622377
```

24.8 用基本 R 作分类汇总

24.8.1 用 tapply() 分组汇总

用 `tapply()` 函数进行分组概括, 格式为：

```
  tapply(X, INDEX, FUN)
```

其中 `X` 是一个向量，`INDEX` 是一个分类变量，`FUN` 是概括统计函数。

比如，下面的程序分性别组计算疗前体积的均值：

```
with(
  d.cancer,
  tapply(v0, sex, mean))
```

```
##        F        M
## 113.2354 108.1214
```

24.8.2 用 aggregate() 分组概括数据框

aggregate 函数对输入的数据框用指定的分组变量（或交叉分组）分组进行概括统计。例如，下面的程序按性别分组计算年龄、疗前体积、疗后体积的平均值:

```
aggregate(
  d.cancer[, c("age", "v0", "v1")],
  list(sex=d.cancer[["sex"]]),
  mean, na.rm=TRUE)
```

```
##   sex      age       v0       v1
## 1   F 66.14286 113.2354 42.65538
## 2   M 63.25000 108.1214 45.95524
```

aggregate() 第一个参数是数据框，第二个参数是列表，列表元素是用来分组或交叉分组的变量，第三个参数是概括用的函数，概括用的函数的选项可以在后面给出。

可以交叉分组后概括，如

```
with(
  d.cancer,
  aggregate(
    cbind(v0, v1),
    list(sex=sex, type=type),
    mean))
```

```
##   sex type        v0       v1
## 1   F 鳞癌 126.99250 45.54750
## 2   M 鳞癌 113.55722 49.65556
## 3   F 腺癌 107.12111 41.37000
## 4   M 腺癌  75.50667 23.75333
```

24.8.3 用 split() 函数分组后概括

split 函数可以把数据框的各行按照一个或几个分组变量分为子集的列表，然后可以用 sapply() 或 vapply() 对每组进行概括。如

```
sp <- split(d.cancer[,c("v0","v1")], d.cancer[["sex"]])
sapply(sp, colMeans)
```

```
##             F         M
```

```
## v0 113.23538 108.12143
## v1  42.65538  45.95524
```

返回矩阵，行为变量 v0, v1，列为不同性别，值为相应的变量在各性别组的平均值。当 `sapply()` 对列表每项的操作返回一个向量时，总是列表每项的输出保存为结果的一列。`colMeans` 函数计算分组后数据框子集每列的平均值。

24.9 用 plyr 包进行分类概括

plyr 是一个专注于分组后分别分析然后将分析结果尽可能合理地合并的扩展包, 功能强大，dplyr 包仅针对数据框，使用更方便，但是对于复杂情况功能不如 plyr 包强。plyr 包已经被 dplyr、purrr 包代替，不再继续开发新版本。这部分内容仅作为备忘，读者可以跳过。

plyr 的输入支持数组、数据框、列表，输出支持数组、数据框、列表或无输出。分组分析的函数输出格式需要与指定的输出格式一致。

这里主要介绍从数据框分组概括并将结果保存为数据框的方法，使用 plyr 包的 `ddply()` 函数。实际上，dplyr 包的这种功能更方便。plyr 包的优点是可以自定义概括函数，使得结果表格符合用户的预期，处理多个变量时程序更简洁。

plyr 包与 dplyr 包的函数名冲突比较大，所以需要先卸载 dplyr 包再调用 plyr 包：

```
if("dplyr" %in% .packages()) detach("package:dplyr")
library(plyr)
```

`ddply()` 函数第一自变量是要分组的数据框，第二自变量是分组用的变量名，第三自变量是一个概括函数，此概括函数以一个数据框子集（数据类型是数据框）为输入，输出是一个数值、一个数值型向量或者一个数据框, 但最好是数据框。例如，按性别分组，计算 v0 的平均值：

```
ddply(d.cancer, "sex",
      function(d) c(mean.v0 = mean(d[["v0"]], na.rm=TRUE)))
```

下面的程序按性别分组，分别计算 v0 与 v1 的平均值：

```
ddply(d.cancer, "sex",
      function(d) colMeans(d[,c("v0", "v1")]))
```

下面的程序按性别分组，计算 v0 和 v1 的平均值、标准差：

```
f1 <- function(dsub){
  tab <- tibble(
    " 变量"=c("v0", "v1"),
    " 均值"=c(mean(dsub[,"v0"], na.rm=TRUE),
```

```
            mean(dsub[,"v1"], na.rm=TRUE)),
    " 标准差"=c(sd(dsub[,"v0"], na.rm=TRUE),
             sd(dsub[,"v1"], na.rm=TRUE)))
  tab
}
ddply(d.cancer, "sex", f1)
```

注意 f1() 结果是一个数据框。程序有些重复内容，对每个变量和每个统计量都需要分别写出，如果这样用 plyr 包就不如直接用 dplyr::summarise() 了。下面用 vapply() 简化程序。

按性别分组，然后 v0、v1 各自一行结果，计算非缺失值个数、均值、标准差、中位数：

```
f2 <- function(d, variables){
  d1 <- d[,variables,drop=FALSE]
  nnotmiss <- vapply(d1, function(x) sum(!is.na(x)), 1L)
  xm <- vapply(d1, mean, 0.0, na.rm=TRUE)
  xsd <- vapply(d1, sd, 1.0, na.rm=TRUE)
  xmed <- vapply(d1, median, 0.0, na.rm=TRUE)
  data.frame(variable=variables,
             n=nnotmiss,
             mean=xm,
             sd=xsd,
             median=xmed)
}
ddply(d.cancer, "sex", f2, variables = c("v0", "v1"))
```

f2() 函数针对分组后的数据框子集，这样的函数可以先在一个子集上试验。在 f2() 函数中，设输入的数据子集为 d，要分析的变量组成的数据框为 d1，用 vapply() 函数对 d1 的每一列计算一种统计量，然后将每种统计量作为结果数据框的一列。vapply() 函数类似于 lapply() 和 sapply()，但是用第三个自变量表示要应用的变换函数的返回值类型和个数，用举例的方法给出。

ddply() 也可以对交叉分类后每个类分别汇总，例如按照性别与病理类型交叉分组后汇总 v0、v1：

```
ddply(d.cancer, c("sex", "type"), f2, variables=c("v0", "v1"))
```

上面的程序写法适用于已知要分析的变量名的情况。如果想对每个数值型变量都分析，而且想把要计算的统计量用统一的格式调用，可以写成：

```
f3 <- function(d){
  ff <- function(x){
    c(n=sum(!is.na(x)),
```

```
      each(mean, sd, median)(x, na.rm=TRUE))
  }
  ldply(Filter(is.numeric, d), ff)
}
ddply(d.cancer, "sex", f3)
```

ff() 函数对输入的一个数值型向量计算 4 种统计量，返回一个长度为 4 的数值型向量，用来对分组后的数据子集中的一列计算 4 种统计量。plyr 包的 each() 函数接受多个函数，返回一个函数可以同时得到这几个函数的结果，结果中各元素用输入的函数名命名。f3() 函数中的 Filter 函数用于从列表或数据框中取出满足条件的项，这里取出输入的数据子集 d 中所有的数值型列。f3() 函数中的 ldply() 函数接受一个列表或看成列表的一个数据框，对数据框的每列应用 ff() 函数计算 4 种统计量，然后合并所有各列的统计量为一个数据框，结果数据框的每行对应于 d 中的一列。程序中的 ddply() 函数接受一个数据框，第二自变量指定用来将数据框分组的变量，第三自变量 f3() 是对分组后的数据框子集进行分析的函数，此函数接受一个数据框，输出一个数据框。

上面的程序也可以利用无名函数写成：

```
f4 <- function(x){
  c(n=sum(!is.na(x)),
    each(mean, sd, median)(x, na.rm=TRUE))
}
ddply(d.cancer, "sex",
      function(d)
        ldply(Filter(is.numeric, d), f4))
```

24.10 练习

- 把patients.csv读入 “d.patients” 中，并计算发病年龄、发病年、发病月、发病年月（格式如 “200702” 表示 2007 年 2 月份）。
- 把 “现住地址国标” 作为字符型，去掉最后两位，仅保留前 6 位数字，保存到变量 “地址编码” 中。
- 按照地址编码和发病年月交叉分类汇总发病人数，保存到数据框 d.pas1 中，然后保存为 CSV 文件 “分区分年月统计.csv” 中。要求结果有三列：“地址编码”、“发病年月”、“发病人数”。
- 按照地址编码和发病月分类汇总发病人数，保存到数据框 d.pas2 中，然后保存为 CSV 文件 “分区分月统计.csv” 中。要求每个地址编码占一行，各列为地址编码以及 1、2、……、12 各月份，每行为同一地址编码各月份的发病数。
- 按发病年月和性别汇总发病人数，并计算同年月不分性别的发病总人数。结果保存到数据框 d.pas3 中，然后保存到 CSV 文件 “分年月分性别统计.csv” 中。要求每个不同年月占一行，变量包括年月、男性发病数、女性发病数、总计。

- 分析病人的职业分布，保存到数据框 d.pas4 中，然后保存到 CSV 文件“职业构成.csv”中。要求各列为职业、发病人数、百分比（结果乘以 100 并保留一位小数）。
- 把年龄分成 0—9, 11—19, ……, 70 以上各段，保存为“年龄段”变量。用年龄段和性别交叉汇总发病人数和百分比 (结果乘以 100 并保留一位小数)，保存到“年龄性别分布.csv”中。要求将每个年龄段的男性发病人数、发病率、女性发病人数、发病率存为一行。

Chapter 25

函数式编程和数据框列表列

25.1 函数式编程介绍

R 支持类 (class) 和方法 (method)，实际提供了适用于多种自变量的通用函数 (generic function，或称泛型函数)，不同自变量类型调用该类特有的方法，但函数名可以保持不变。这可以支持一定的面向对象编程方式。

R 也支持函数式编程，但不是专门的函数式编程语言。R 语言的设计主要用函数求值来进行运算；R 的用户主要使用函数调用来访问 R 的功能。

按照函数式编程的要求，函数应该是“第一级对象”，可以将函数对象绑定到变量名上面，可以在列表等结构中保存多个函数，可以在函数内定义函数，可以用函数作为函数的自变量，R 函数满足这样的要求。

函数式编程的目的是提供可理解、可证明正确的软件。R 虽然带有函数式编程语言特点，但并不强求使用函数式编程规范。典型的函数式编程语言如 Haskel, Lisp 的运行与 R 的显式的、顺序的执行方式相差很大。

25.1.1 纯函数

函数式编程要求每个函数必须功能清晰、定义确切，最好是所谓“纯函数”。R 并不是专门的函数式编程语言，专门的函数式编程语言提供了定义纯函数的功能。纯函数需要满足如下条件：

- 没有副作用。调用一个函数对后续运算没有影响，不管是再次调用此函数还是调用其它函数。这样，用全局变量在函数之间传递信息就是不允许的。其它副作用包括写文件、打印、绘图等，这样的副作用对函数式要求破坏不大。函数返回值包含了函数执行的所有效果。
- 不受外部影响。函数返回值只依赖于其自变量及函数的定义。函数定义仅由对所有可能的自变量值确定返回值来确定，不依赖于任何外部信息（也就不能依赖于全局变量与系统设置值）。在专用的函数式编程语言中，函数定义返回值的方式是隐含地遍历所有可能的参数值给出返回值，而不是用过程式的计算来修改对象的值。
- 不受赋值影响。函数定义不需要反复对内部对象（所谓“状态变量”）赋值或修改。

R 的函数一般不能修改实参的值，这有助于实现纯函数的要求。但是，如果多个函数之间用全局变量传递信息，就不能算是纯函数。像 options() 函数这样修改全局运行环境的功能会破坏函数式要求。尽可能让自己的函数不依赖于 options() 中的参数。

如果函数对相同的输入可以有不同的输出当然不是纯函数，例如 R 中的随机数函数 (sample(), runif(), rnorm 等)。

与具体硬件、软件环境有关的一些因素也破坏纯函数要求，如不同的硬件常数、精度等。调用操作系统的功能对函数式要求破坏较大。减少赋值主要需要减少循环，可以用 R 的向量化方法解决。

一个 R 函数是否满足纯函数要求不仅要看函数本身，还要看函数内部调用的其它函数是否纯函数。

R 不是专用的函数式编程语言，但可以采用函数式编程的范式，将大多数函数写成纯函数，将有副作用的单独写在少数的函数中。

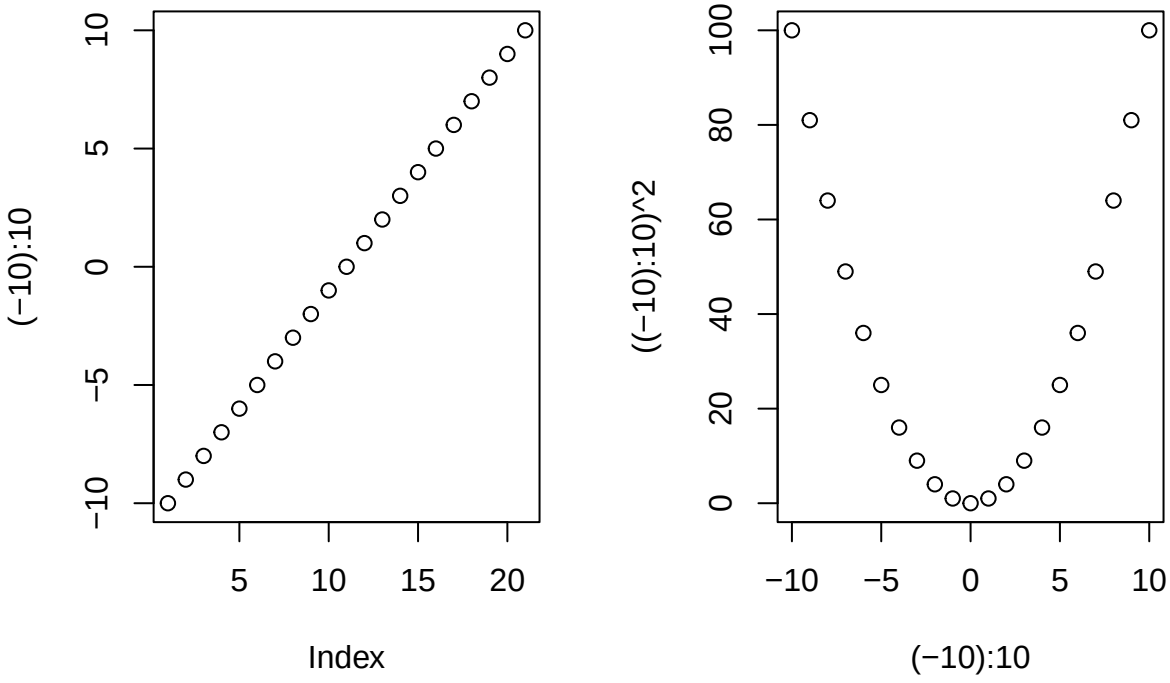
25.1.2 副作用和运行环境恢复

如果函数除了输出之外还在其它方面影响了运行环境，这样的函数就不是纯函数。所有画图函数 (plot 等)、输出函数 (cat, print, save 等) 都是这样的函数。这些对运行环境的改变叫做**副作用**（side effects）。又比如，library() 函数会引入新的函数和变量，setwd(), Sys.setenv(), Sys.setlocale() 会改变 R 运行环境，options(), par() 会改变 R 全局设置。自定义 R 函数中如果调用了非纯函数也就变成了非纯函数。编程中要尽量控制副作用而且要意识到副作用的影响，尤其是全局设置与全局变量的影响。

有些函数不可避免地要修改运行环境，比如当前工作目录 (用 setwd()）、R 运行选项 (用 options()）、绘图参数 (用 par()) 等，如果可能的话，在函数结束运行前，应该恢复对运行环境的修改。为此，可以在函数体的前面部分调用 on.exit() 函数，此函数的参数是在函数退出前要执行的表达式或复合表达式。此函数可以多次调用，一般做法是函数体内部每次更改运行环境处调用一次 on.exit()，预先设定退出前恢复。这需要使用 add=TRUE 选项。

例如，绘图的函数中经常需要用 par() 修改绘图参数，这会使得后续程序出错。为此，可以在函数开头保存原始的绘图参数，函数结束时恢复到原始的绘图参数。如

```
f <- function(){
  opar <- par(mfrow=c(1,2))
  on.exit(par(opar), add=TRUE)
  plot((-10):10)
  plot((-10):10, ((-10):10)^2)
}
f()
```



如果需要指定越晚添加的恢复动作越先执行，在 `on.exit()` 中还要加上 `after=FALSE` 选项。

25.1.3 R 的函数式编程功能

R 语言不是专用的函数式编程语言，但支持使用函数式编程的一些常见做法。R 函数是第一级对象，支持内嵌函数，并可以输入函数作为函数的自变量，称这样的函数为**泛函** (functionals)，如 `lapply` 类函数；可以输出函数作为函数结果，称这样的函数为**函数工厂**；可以输入函数，进行一定修改后输出函数，称这样的函数为**函数算子** (function operators)。这些功能都为函数式编程风格提供了有力的支持。

利用 R 的 purrr 扩展包，可以用统一的风格进行函数式编程，比基本 R 的 lapply 类函数、Map、Reduce 等更容易使用。

下面讲解泛函、函数工厂和函数算子。

25.2 泛函

许多函数需要用函数作为参数，称这样的函数为**泛函** (functionals)。典型的泛函是 `lapply` 类函数。这样的函数具有很好的通用性，因为需要进行的操作可以输入一个函数来规定，用输入的函数规定要进行什么样的操作。

25.2.1 purrr::map 函数

设我们要对列表或向量 x 的每个元素 x[[i]] 调用函数 f()，将结果保存成一个列表。这样做的一个程序框架是：

```
y <- vector(mode = "list", length = length(x))
for(i in seq_along(x)){
  y[[i]] <- f(x[[i]])
}
names(y) <- names(x)
```

其中的输入 x 是任意的，函数 f 是任意的。purrr 包的 map() 函数可以用一条命令完成上述任务：

```
y <- map(x, f)
```

这个函数与基本 R 的 lapply 功能基本相同，对数据框每列、列表每项进行计算或操作时最为适用。

25.2.1.1 map 数据框处理示例

下面举例说明 map 函数对数据框处理的应用。

typeof() 函数求变量的存储类型，如

```
typeof(d.class[["height"]])
```

```
## [1] "double"
```

这里 d.class 是数据框，数据框也是列表，每个列表元素是数据框的一列。

如下程序使用 purrr::map() 求每一列的存储类型，map 的结果总是列表，每个列表元素对应于输入的一个元素，如:

```
library(purrr)
map(d.class, typeof)
```

```
## $name
## [1] "character"
##
## $sex
## [1] "integer"
##
## $age
## [1] "double"
```

```
##
## $height
## [1] "double"
##
## $weight
## [1] "double"
```

当结果比较简单时，保存为列表不够方便，函数 `unlist()` 可以将比较简单的列表转换为基本类型的向量，如：

```
map(d.class, typeof) |> unlist()
```

```
##        name         sex         age      height      weight
## "character"   "integer"    "double"    "double"    "double"
```

实际上，关于一个数据框的结构，用 `str()` 函数或者 `dplyr::glimpse()` 函数可以得到更为详细的信息：

```
str(d.class)
```

```
## spc_tbl_ [19 x 5] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ name  : chr [1:19] "Alice" "Becka" "Gail" "Karen" ...
##  $ sex   : Factor w/ 2 levels "M","F": 2 2 2 2 2 2 2 2 2 1 ...
##  $ age   : num [1:19] 13 13 14 12 12 15 11 15 14 14 ...
##  $ height: num [1:19] 56.5 65.3 64.3 56.3 59.8 66.5 51.3 62.5 62.8 69 ...
##  $ weight: num [1:19] 84 98 90 77 84.5 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   .default = col_double(),
##   ..   name = col_character(),
##   ..   sex = col_factor(levels = c("M", "F"), ordered = FALSE, include_na = FALSE),
##   ..   age = col_double(),
##   ..   height = col_double(),
##   ..   weight = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
dplyr::glimpse(d.class)
```

```
## Rows: 19
## Columns: 5
## $ name   <chr> "Alice", "Becka", "Gail", "Karen", "Kathy", "Mary", "Sandy", "S~
```

```
## $ sex    <fct> F, F, F, F, F, F, F, F, F, M, M, M, M, M, M, M, M, M, M
## $ age    <dbl> 13, 13, 14, 12, 12, 15, 11, 15, 14, 14, 14, 15, 12, 13, 12, 16,~
## $ height <dbl> 56.5, 65.3, 64.3, 56.3, 59.8, 66.5, 51.3, 62.5, 62.8, 69.0, 63.~
## $ weight <dbl> 84.0, 98.0, 90.0, 77.0, 84.5, 112.0, 50.5, 112.5, 102.5, 112.5,~
```

25.2.1.2 map 返回基本类型向量

purrr::map() 总是返回列表。如果确知其调用的函数总是返回某种类型的标量值，可以用 map 的变种：

- map_lgl()：返回逻辑向量；
- map_int()：返回整型向量；
- map_dbl(): 返回双精度浮点型向量 (double 类型)；
- map_chr(): 返回字符型向量。

比如，求 mtcars 各列类型，因为确知 typeof() 函数对每列返回一个标量字符串，所以可以写成：

```
map_chr(d.class, typeof)
```

```
##        name         sex         age      height      weight
## "character"   "integer"    "double"    "double"    "double"
```

用 map_lgl() 返回每一列是否数值型列：

```
map_lgl(d.class, is.numeric)
```

```
##   name    sex    age height weight
##  FALSE  FALSE   TRUE   TRUE   TRUE
```

25.2.1.3 ... 形参

在 R 函数的形参中，允许有一个特殊的... 形参（三个小数点），这在调用泛函类型的函数时起到重要作用。在调用泛函时，所有没有形参与之匹配的实参，不论是带有名字还是不带有名字的，都自动归入这个参数，将会由泛函传递给作为其自变量的函数。... 参数的类型相当于一个列表，列表元素可以部分有名部分无名，用 list(...) 可以将其转换成列表再访问。

例如，函数 mean() 可以计算去掉部分最低、最高值之后的平均值，用选项 trim= 指定一个两边分别舍弃的值的个数比例。为了将 d.cancer 的 v0, v1 列计算上下各自扣除 10% 的平均值，需要利用 map_dbl() 函数的... 参数输入 trim 选项值，如：

```
map_dbl(d.cancer[,c("v0", "v1")],
  mean, trim=0.10)
```

```
##        v0        v1
## 103.74357  40.98786
```

用 tidyverse 方式：

```
d.cancer |>
  summarise(across(c(v0, v1),
    ~ mean(., trim = 0.10)))
```

```
## # A tibble: 1 x 2
##      v0    v1
##   <dbl> <dbl>
## 1  104.  41.0
```

选择数值型列，然后用 `map_double` 计算截尾的均值：

```
d.cancer[, map_lgl(d.cancer, is.numeric)] |>
  map_dbl(mean, trim=0.10, na.rm = TRUE)
```

```
##        id       age        v0        v1
##  17.50000  64.31579 103.74357  40.98786
```

purrr 包提供了一个 `keep` 函数，可以专门用来选择数据框各列或列表元素中满足某种条件的子集，这个条件用一个返回逻辑值的函数来给出。如：

```
d.cancer |>
  keep(is.numeric) |>
  map_dbl(mean, trim = 0.10, na.rm = TRUE)
```

```
##        id       age        v0        v1
##  17.50000  64.31579 103.74357  40.98786
```

也可以用 tidyverse 解决：

```
d.cancer |>
  summarise(across(
    where(is.numeric),
    ~ mean(., trim = 0.10, na.rm = TRUE) ))
```

```
## # A tibble: 1 x 4
##      id   age    v0    v1
##   <dbl> <dbl> <dbl> <dbl>
## 1  17.5  64.3  104.  41.0
```

需要注意的是，在 map 类泛函中... 仅用来将额外的选项传递给要调用的函数，不支持向量化，如果需要对两个或多个自变量的对应元素作变换，需用用 purrr 包的 map2 等函数。如果泛函中调用的是无名函数，则... 参数会造成变量作用域理解困难。

25.2.1.4 用 map 处理 strsplit 函数结果示例

假设有 4 个学生的 3 次小测验成绩，每个学生的成绩记录到了一个以逗号分隔的字符串中，如：

```
s <- c('10, 8, 7',
      '5, 2, 2',
      '3, 7, 8',
      '8, 8, 9')
```

对单个学生，可以用 strsplit() 函数把三个成绩拆分，如：

```
strsplit(s[1], ',', fixed=TRUE)[[1]]
```

```
## [1] "10" " 8" " 7"
```

注意这里 strsplit() 的结果是仅有一个元素的列表，用了“[[...]]”格式取出列表元素。拆分的结果可以用 as.numeric() 转换为有三个元素的数值型向量：

```
strsplit(s[1], ',', fixed=TRUE)[[1]] |> as.numeric()
```

```
## [1] 10  8  7
```

还可以求三次小测验的总分：

```
strsplit(s[1], ',', fixed=TRUE)[[1]] |>
  as.numeric() |>
  sum()
```

```
## [1] 25
```

用 strsplit() 处理有 4 个字符串的字符型向量 s, 结果是长度为 4 的列表：

```
tmpr <- strsplit(s, ',', fixed=TRUE); tmpr
```

```
## [[1]]
## [1] "10" " 8" " 7"
##
## [[2]]
## [1] "5"  " 2" " 2"
##
## [[3]]
## [1] "3"  " 7" " 8"
##
## [[4]]
## [1] "8"  " 8" " 9"
```

用 map() 和 as.numeric() 可以把列表中所有字符型转为数值型，输出为一个列表，然后再对各个列表元素中的向量求和。使用管道运算符表达逐步的操作：

```
s |>
  strsplit(split=",", fixed=TRUE) |>
  map(as.numeric) |>
  map_dbl(sum)
```

```
## [1] 25  9 18 25
```

25.2.1.5 在 map 中使用无名函数以及简写方法

map() 中调用的函数可以是在 map() 中直接现场定义的无名函数。

仍考虑上面的问题，有 4 个学生，每个学生有三门成绩，成绩之间用逗号分隔。将每个学生的成绩拆分为三个字符串后，就可以对每个学生调用一个统一的无名函数，将字符型转换为数值型以后求和：

```
s |>
  strsplit(split=",", fixed=TRUE) |>
  map_dbl(\(x) sum(as.numeric(x)))
```

```
## [1] 25  9 18 25
```

使用无名函数格式比较复杂，purrr 包为在 map() 等泛函中使用无名函数提供了简化的写法，将无名函数写成“~表达式”格式，表达式就是无名函数定义，用.x 表示只有一个自变量时的自变量名，用.x 和.y 表示只有两个自变量时的自变量名，用..1、..2、..3 这样的名字表示有多个自变量时的自变量名。如：

```
s |>
  strsplit(split=",", fixed=TRUE) |>
  map_dbl(~ sum(as.numeric(.x)))
```

```
## [1] 25  9 18 25
```

需要注意的是，如果 map() 等泛函中的无名函数需要访问其它变量的话，需要理解其变量作用域或访问环境。另外，无名函数中的其它变量在每次被 map() 应用到输入列表的元素时都会重新计算求值。建议这样的情况改用有名函数，这样其中访问其它变量时作用域规则比较容易掌控，也不会重复求值。

25.2.1.6 在 map 中提取列表元素成员的简写

较为复杂的数据，有时表现为列表的列表，每个列表元素都是列表或者向量。JSON、YAML 等格式转换为 R 对象就经常具有这种嵌套结构。

例如，有如下的嵌套格式数据，这样的数据不利于用数据框格式保存：

```
od <- list(
  list(
    101, name=" 李明", age=15,
    hobbies=c(" 绘画", " 音乐")),
  list(
    102, name=" 张聪", age=17,
    hobbies=c(" 足球"),
    birth="2002-10-01")
)
```

为了取出每个列表元素的第一项，本来应该写成：

```
map_dbl(od, \(x) x[[1]])
```

```
## [1] 101 102
```

或：

```
map_dbl(od, ~ .x[[1]])
```

```
## [1] 101 102
```

purrr 包提供了进一步的简化写法，在需要一个函数或者一个“~ 表达式”的地方，可以用整数下标值表示对每个列表元素提取其中的指定成分，如：

```
map_dbl(od, 1)
```

```
## [1] 101 102
```

类似地，可以在需要函数的地方写一个成员名，提取每个列表元素中该成员，如：

```
map_chr(od, "name")
```

```
## [1] "李明" "张聪"
```

在应该用函数的地方还可以提供一个列表，列表元素为成员序号或者成员名，进行逐层挖掘，如：

```
map_chr(od, list("hobbies", 1))
```

```
## [1] "绘画" "足球"
```

表示取出每个列表元素的 hobbies 成员的第一个元素（每人的第一个业余爱好）。

取出不存在的成员会出错，但可以用一个.default 选项指定查找不到成员时的选项，如：

```
map_chr(od, "birth", .default=NA)
```

```
## [1] NA           "2002-10-01"
```

25.2.1.7 数据框分组处理示例

对 d.class 数据框，希望分成男女生两个组，每组内建立用身高预测体重的一元线性回归模型，提取各模型的斜率项。基本 R 的 split 函数可以按数据框的某列将数据框分成若干个子数据框，结果为子数据框的列表。借助于 purrr 包的 map 类函数和管道运算符，可以将分组建模过程写成：

```
d.class |>
  split(d.class[["sex"]]) |>
  map(~ lm(weight ~ height, data=.x)) |>
  # 这里的“.”是 map 需要输入的无名函数的自变量
  map(coef) |>
  map_dbl(2)
```

```
##        M        F
## 3.912549 3.424405
```

这些步骤用基本 R 的 lapply 或者 for 循环也能够完成，但会更难读懂，或者需要生成许多中间临时变量，不如上面例子这样步骤清晰而且不需要产生中间结果。上述问题使用 tidyr 的 nest 和 unnest 可以将分组结果、回归结果保存为 tibble 的列表列，管理起来更为直观，见25.4。

dplyr 包和 plyr 包与这个例子的思想类似，只不过更有针对性。

25.2.2 purrr 包中 map 函数的变种

purrr 包的 map 函数输入一个数据自变量和一个函数，输出为列表；map_dbl() 等将输出转化为基础类型的向量。

purrr 包还提供了与 map 目的类似，但输入输出类型有变化的一些函数，包括：

- modify()，输入一个数据自变量和一个函数，输出与输入数据同类型的结果；
- map2() 可以输入两个数据自变量和一个函数，将两个自变量相同下标的元素用函数进行变换，输出列表；
- imap() 根据一个下标遍历；
- walk() 输入一个数据自变量和一个函数，不返回任何结果，仅利用输入的函数的副作用；
- pmap() 输入若干个数据自变量和一个函数，对数据自变量相同下标的元素用函数进行变换。

将这些 map 变种按输入类型分为：

- 一个数据自变量，代表为 map()；
- 两个自变量，代表为 map2()；
- 一个自变量和一个下标变量，代表为 imap()；
- 多个自变量，代表为 pmap()。

将这些 map 变种按输出结果类型分为：

- 列表，代表为 map();
- 基础类型的向量，如 map_dbl(), map_chr() 等；
- 与输入数据类型相同的输出，代表为 modify()；
- 不输出结果，代表为 walk()。

输入类型和输出类型两两搭配，purrr 包提供了 27 种 map 类函数。

25.2.2.1 输入输出类型相同的 modify 函数

purrr 的 modify 函数与 map 函数作用类似，并不会原地修改输入数据，而是制作修改后的副本，输出的结果类型与输入数据的结果类型相同，所以可以用来修改数据框各列生成一个副本数据框。

比如，对 d.class 中的三个数值型列，都减去列中位数，其它列保持不变：

```
d1 <- modify(d.class, ~ if(is.numeric(.x)) .x - median(.x) else .x )
```

purrr 包还提供了一个 modify_if() 函数，可以对满足条件的列进行修改，如：

```
d2 <- modify_if(d.class, is.numeric, ~ .x - median(.x))
```

x1	x2
106	101
108	112
103	107
110	105

x1	x2
104	102
111	114
112	105
109	107

x1	x2
0.0210773	0.0070588

25.2.2.2 对两个自变量的相同下标元素调用函数

map() 函数仅支持一个输入数据的列表或向量。map2() 函数支持两个输入数据的列表或向量，map2(x, y, f, ...) 对每个下标 i 调用 f(x[[i]], y[[i]], ...)，结果返回一个列表。如果知道函数 f() 会返回类型确定的标量值，可以用 map2_dbl() 等变种。

例如，d1 是某市 2001 年四个季度的若干项经济指标，d2 是 2002 年的对应指标，计算每项指标年度总和的同比增幅：

```
d1 <- tibble(
  x1 = c(106, 108, 103, 110),
  x2 = c(101, 112, 107, 105) )
d2 <- tibble(
  x1 = c(104, 111, 112, 109),
  x2 = c(102, 114, 105, 107) )
d3 <- map2_dbl(d1, d2, ~ (sum(.y) - sum(.x)) / sum(.x))
knitr::kable(list(d1=d1, d2=d2, d3=as_tibble(rbind(d3))))
```

上述 map2_dbl() 调用等价于：

```
d3 <- numeric(2)
names(d3) <- names(d1)
for(j in seq_along(d1)){
  d3[[j]] <- (sum(d2[[j]]) - sum(d1[[j]]))/sum(d1[[j]])
}
d3
```

```
##          x1          x2
## 0.021077283 0.007058824
```

如果计算结果与两个输入数据类型相同，可以用 modify2()。比如，上面的例子数据计算每个指标的同比增幅：

```
modify2(d1, d2, ~ (.y - .x) / .x)
```

```
## # A tibble: 4 x 2
```

```
##         x1       x2
##      <dbl>    <dbl>
## 1 -0.0189   0.00990
## 2  0.0278   0.0179
## 3  0.0874  -0.0187
## 4 -0.00909  0.0190
```

注意和 modify() 一样，上述程序并不会修改 d1 和 d2 的值，而是返回与其类型和大小相同的一个数据框。

map2() 允许输入的 x 和 y 两个列表其中一个长度为 1，这时长度为 1 的列表的元素被重复利用。如：

```
d1b <- d1[,1,drop=FALSE]
map2_dbl(d1b, d2, ~ (sum(.y) - sum(.x)) / sum(.x))
```

```
##         x1         x1
## 0.02107728 0.00234192
```

基本 R 的 Map() 函数起到与 map2() 和 pmap() 类似的作用。

25.2.2.3 不产生输出的 walk 类函数

有时仅需要遍历一个数据结构调用函数进行一些显示、绘图，这称为函数的副作用，不需要返回结果。purrr 的 walk 函数针对这种情形。

例如，显示数据框中每个变量的类别：

```
walk(d.class, ~ cat(typeof(.x), "\n"))
```

```
## character
## integer
## double
## double
## double
```

上面这个例子缺点是没有显示对应的变量名。

walk2() 函数可以接受两个数据自变量，类似于 map2()。例如，需要对一组数据分别保存到文件中，就可以将数据列表与保存文件名的字符型向量作为 walk2() 的两个数据自变量。下面的程序将 d.class 分成男女生两个子集，保存到两个 csv 文件中：

```
dl <- split(d.class, d.class[["sex"]])
walk2(dl, paste0("class-", names(dl), ".csv"),
      ~ write.csv(.x, file=.y))
```

改用管道运算符：

```
d.class |>
  split(d.class[["sex"]]) |>
  walk2(paste0("class-", names(.), ".csv"),
        ~ write.csv(.x, file=.y))
```

事实上，walk、walk2 并不是没有输出，它们返回不显示的第一个自变量，所以也适合用在管道运算的中间使得管道不至于中断。

基本 R 没有提供类似 walk 的功能。

25.2.2.4 可同时访问下标或元素名与元素值的 imap 类函数

在前面用 walk 函数显示数据框各列类型的例子中，没有能够同时显示变量名。如果 x 有元素名，imap(x, f) 相当于 map2(x, names(x), f)；如果 x 没有元素名，imap(x, f) 相当于 map2(x, seq_along(x), f)。iwalk() 与 imap() 类似但不返回信息。f 是对数据每一项要调用的函数，输入的函数的第二个自变量或者无名函数的.y 自变量会获得输入数据的元素名或者元素下标。imap_chr() 等是固定返回类型的变种。

例如，显示数据框各列的变量名：

```
iwalk(d.class, ~ cat(.y, ": ", typeof(.x), "\n"))
```

```
## name :  character
## sex :  integer
## age :  double
## height :  double
## weight :  double
```

返回字符型向量的写法：

```
imap_chr(d.class, ~ paste0(.y, " ==> ", typeof(.x))) |>
  unname()
```

```
## [1] "name ==> character" "sex ==> integer"    "age ==> double"
## [4] "height ==> double"  "weight ==> double"
```

输入数据没有元素名的演示：

```
dl <- list(1:5, 101:103)
iwalk(dl, ~ cat("NO. ", .y, ": ", .x[[1]], "\n"))
```

```
## NO.  1 :  1
## NO.  2 :  101
```

显示了每个列表元素的第一项。

基本 R 没有提供类似 `imap` 的功能。

25.2.2.5 多个数据自变量的 pmap 类函数

R 的函数调用时支持向量化，可以很好地处理各个自变量是向量的情形，但是当自变量是列表、数据框等复杂类型时不能自动进行向量化处理。purrr 包的 `pmap` 类函数支持对多个列表、数据框、向量等进行向量化处理。`pmap` 不是将多个列表等作为多个自变量，而是将它们打包为一个列表。所以，`map2(x, y, f)` 用 `pmap()` 表示为 `pmap(list(x, y), f)`。

在确知输出类型时可以用 `pmap_chr()`, `pmap_dbl()` 等变种，在不需要输出结果时可以用 `pwalk()`。

比如，将三个列表中的对应项用 `c()` 函数连接：

```
x <- list(101, name=" 李明")
y <- list(102, name=" 张聪")
z <- list(103, name=" 王国")
pmap(list(x, y, z), c)
```

```
## [[1]]
## [1] 101 102 103
##
## $name
## [1] "李明" "张聪" "王国"
```

因为数据框是列表，所以 `pmap()` 也可以输入一个数据框和要并行执行的函数，对数据框的每一行执行该函数。例如：

```
d <- tibble::tibble(
  x = 101:103,
  y=c(" 李明", " 张聪", " 王国"))
knitr::kable(d)
```

x	y
101	李明
102	张聪
103	王国

```
pmap_chr(d, \(...) paste(..., sep=":"))
```

```
## [1] "101:李明" "102:张聪" "103:王国"
```

pmap() 和其它的 map() 类函数有一个区别是，因为将输入数据打包在一个列表中，而列表元素是有变量名的，这样就可以将列表变量名取为要调用的函数的自变量名，使得对输入列表中各元素的每个成员调用函数时，可以带有对应的形参名调用。

例如，mean() 函数可以计算去掉最小、最大一部分后的平均值，mean(x, trim) 用 trim 选项控制两端分别去掉的比例，但是 trim 选项必须是标量。用 map_dbl() 解决方法如下：

```
set.seed(101)
x <- rcauchy(1000)
trims <- c(0.05, 0.1, 0.2, 0.3, 0.4)
map_dbl(trims, ~ mean(x=x, trims=.x))
```

```
## [1] 0.7271278 0.7271278 0.7271278 0.7271278 0.7271278
```

可以用 pmap() 的列表元素名自动对应到调用函数形参名的方法：

```
pmap_dbl(list(trims = trims), mean, x=x)
```

```
## [1] 0.7271278 0.7271278 0.7271278 0.7271278 0.7271278
```

或：

```
pmap_dbl(list(trims = trims), ~ mean(x))
```

```
## [1] 0.7271278 0.7271278 0.7271278 0.7271278 0.7271278
```

基本 R 的 Map() 函数提供了类似的功能，但是不允许多个自变量中有长度为 1 的；基本 R 的 mapply() 函数与 Map() 类似，但是会像 sapply() 函数那样试图自动找到最简化的输出数据结构，这在通用程序中会使得结果不可控。

25.2.3 purrr 包中 reduce 类函数

25.2.3.1 reduce 函数

许多二元运算符如加法、乘法，可以很自然地推广到多个运算元之间的运算，变成连加、连乘积等等。某些常用的操作已经变成了 R 函数，比如 sum()、prod()，但是其它一些运算，包括用函数表示的运算，也需要推广到对多个进行，比如 intersect(x, y) 求两个集合的交集，希望能推广到求多个集合的交集。

purrr 包的 reduce 函数把输入列表（或向量）的元素逐次地用给定的函数进行合并计算。比如，设 f(x,y) 是一个二元函数，设 z 是有 4 个元素的列表，则 reduce(z, f) 表示

```
f(f(f(z[[1]], z[[2]]), z[[3]]), z[[4]])
```

例如，

```
reduce(1:4, `+`)
```

```
## [1] 10
```

实际执行的是 $(((1+2)+3)+4)$。当然，求 $1:4$ 的和只需要 sum(1:4)，但是 reduce 可以对元素为复杂类型的列表进行逐项合并计算。

考虑多个集合的交集的问题。下面的例子产生了 4 个集合，然后反复调用 intersect() 求出了交集：

```
x <- list(
  c(2, 3, 1, 3, 1),
  c(1, 5, 3, 3, 2),
  c(5, 4, 2, 5, 3),
  c(1, 4, 3, 2, 5))
intersect(intersect(intersect(x[[1]], x[[2]]), x[[3]]), x[[4]])
```

```
## [1] 2 3
```

也可以用管道运算写成:

```
x[[1]] |> intersect(x[[2]]) |> intersect(x[[3]]) |> intersect(x[[4]])
```

```
## [1] 2 3
```

还可以写成循环:

```
y <- x[[1]]
for(i in 2:4) y <- intersect(y, x[[i]])
y
```

```
## [1] 2 3
```

都比较繁琐。

利用 purrr 包的 reduce 函数，只要写成

```
reduce(x, intersect)
```

```
## [1] 2 3
```

泛函的好处是需要进行的变换或计算是作为参数输入的，只要输入其它函数就可以改变要做的计算，比如，变成求各个集合的并集：

```
reduce(x, union)
```

```
## [1] 2 3 1 5 4
```

reduce() 支持... 参数，所以可以给要调用的函数额外的自变量或选项。

reduce 函数对多个输入默认从左向右计算，可以用.dir = "backward" 选项改为从右向左合并。

可以用选项.init 给出合并初值，在通用程序中使用 reduce() 时应该提供此选项，这样如果输入了零长度数据，可以有一个默认的返回值；输入非零长度数据时，此初值作为第一个元素之前的值参与计算，所以一定要取为与要进行的运算一致的值，比如连加的初始值自然为 0，连乘积的初始值自然为 1，多个集合交集的初始值为全集（所有参与运算的各个集合应为此初值的子集），等等。

基本 R 的 Reduce() 函数提供了类似 purrr::reduce() 的功能，不支持... 参数。

25.2.3.2 reduce2 函数

reduce2(x, y, f) 中的 x 是要进行连续运算的数据列表或向量，而 y 是给这些运算提供不同的参数。如果没有.init 初始值，f 仅需调用 length(x)-1 次，所以 y 仅需要有 length(x)-1 个元素；如果有.init 初始值，f 需要调用 length(x) 次，y 也需要与 x 等长。

25.2.3.3 accumulate 函数

对于加法，R 的 sum() 函数可以计算连加，而 cumsum() 函数可以计算逐步的连加。如：

```
sum(1:4)
```

```
## [1] 10
```

```
cumsum(1:4)
```

```
## [1]  1  3  6 10
```

purrr::reduce() 将连加推广到了其它的二元运算，purrr::accumulate() 则类似 cumsum() 的推广。

例如，对前面例子中的 4 个集合，计算逐步的并集，结果的第一项保持原来的第一项不变：

```
accumulate(x, union)
```

```
## [[1]]
## [1] 2 3 1 3 1
##
## [[2]]
## [1] 2 3 1 5
##
## [[3]]
## [1] 2 3 1 5 4
##
## [[4]]
## [1] 2 3 1 5 4
```

将上述结果简化显示：

```
accumulate(x, union) |>
  map(~ sort(unique(.x)))
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] 1 2 3 5
##
## [[3]]
## [1] 1 2 3 4 5
##
## [[4]]
## [1] 1 2 3 4 5
```

25.2.3.4 Map-reduce 算法

Map-reduce 是大数据技术中的重要算法，在 Hadoop 分布式数据库中主要使用此算法思想。将数据分散存储在不同计算节点中，将需要的操作先映射到每台计算节点，进行信息提取压缩，最后用 reduce 的思想将不同节点的信息整合在一起。

25.2.4 purrr 包中使用示性函数的泛函

返回逻辑向量的函数称为示性函数，R 中有许多 `is.xxx` 函数都是示性函数 (predicate functions)。示性函数本身不是泛函，但是它们可以作为泛函的输入。

purrr 包提供了如下的以示性函数函数为输入的泛函：

- `some(.x, .p)`，对数据列表或向量`.x` 的每一个元素用`.p` 判断，只要至少有一个为真，结果就为真；`every(.x, .p)` 与 `some` 类似，但需要所有元素的结果都为真结果才为真。这些函数与 `any(map_lgl(.x, .p))` 和 `all(map_lgl(.x, .p))` 类似，但是只要在遍历过程中能提前确定返回值就提前结束计算，比如 `some` 只要遇到一个真值就不再继续判断，`every` 只要遇到一个假值就不再继续判断。
- `detect(.x, .p)` 返回数据`.x` 的元素中第一个用`.p` 判断为真的元素值，而 `detect_index(.x, .p)` 返回第一个为真的下标值。
- `keep(.x, .p)` 选取数据`.x` 的元素中用`.p` 判断为真的元素的子集；`discard(.x, .p)` 返回不满足条件的元素子集。

例如，判断数据框中有无因子类型的列：

```
some(d.class, is.factor)
```

```
## [1] TRUE
```

判断数据框是否完全由数值型列组成：

```
every(d.class, is.numeric)
```

```
## [1] FALSE
```

返回向量中的第一个超过 100 的元素的值：

```
detect(c(1, 5, 77, 105, 99, 123), ~ .x >= 100)
```

```
## [1] 105
```

返回向量中的第一个超过 100 的元素的下标：

```
detect_index(c(1, 5, 77, 105, 99, 123),
  ~ .x >= 100)
```

```
## [1] 4
```

对于上一个例子，`which(x >= 100)` 可以返回所有满足条件的元素的下标。

下面的例子筛选出数据框的数值型列，并用 `map_dbl` 求每列的平方和：

```
d.class |>
  keep(is.numeric) |>
  map_dbl(~ sum(.x ^ 2))
```

```
##       age    height    weight
##   3409.00  74304.92 199435.75
```

用 tidyverse 的方法：

```
d.class |>
  summarise(across(
    where(is.numeric),
    \(x) sum(x^2)  ))
```

```
## # A tibble: 1 x 3
##     age height  weight
##   <dbl>  <dbl>   <dbl>
## 1  3409 74305. 199436.
```

从数据框（或列表）中选一部分满足某种条件的子集进行变换是常用的做法，所以 `map` 提供了 `map_if()` 和 `modify_if()` 变种，允许输入一个示性函数，对满足条件的子集才应用输入的变换函数进行处理，输入数据中其它元素原样返回。`map_if` 返回列表，`modify_if` 返回与输入数据相同类型的输出。例如，将数据框中数值型列除以 100，其它列保持不变：

```
modify_if(d.class, is.numeric, `/`, 100) |> head()
```

```
## # A tibble: 6 x 5
##   name  sex     age height weight
##   <chr> <fct> <dbl>  <dbl>  <dbl>
## 1 Alice F      0.13  0.565  0.84
## 2 Becka F      0.13  0.653  0.98
```

```
## 3 Gail  F      0.14  0.643  0.9
## 4 Karen F      0.12  0.563  0.77
## 5 Kathy F      0.12  0.598  0.845
## 6 Mary  F      0.15  0.665  1.12
```

基本 R 的 `Find` 函数与 `detect` 作用类似，`Position` 与 `detect_index` 作用类似，`Filter` 函数与 `keep` 作用类似。

25.2.5 基本 R 的函数式编程支持

使用 purrr 包的泛函的好处是用法风格一致，有许多方便功能。对于少量的使用泛函的需求，在不想使用 purrr 包的情况下，可以使用基本 R 中的类似功能。

基本 R 的 `apply` 函数可以对矩阵的每行或每列进行计算，或对多维向量的某个维度进行计算。参见12.6。

基本 R 中的 `integrate`、`uniroot`、`optim`、`omptimize` 等函数也需要输入函数，但主要用于数学计算，与一般的函数式编程关系不大。

基本 R`lapply` 函数用输入的函数对数据的每个元素进行变换，格式为

```
lapply(X, FUN, ...)
```

其中 `X` 是一个列表或向量，`FUN` 是一个函数（可以是有名或无名函数），结果也总是一个列表，结果列表的第 i 个元素是将 `X` 的第 i 个元素输入到 `FUN` 中的返回结果。`...` 参数会输入到 `FUN` 中。这与 `purrr::map()` 功能类似。

`sapply` 与 `lapply` 函数类似，但是 `sapply` 试图简化输出结果为向量或矩阵，在不可行时才和 `lapply` 返回列表结果。如果 `X` 长度为零，结果是长度为零的列表；如果 `FUN(X[i])` 都是长度为 1 的结果，`sapply()` 结果是一个向量；如果 `FUN(X[i])` 都是长度相同且长度大于 1 的向量，`sapply()` 结果是一个矩阵，矩阵的第 i 列保存 `FUN(X[i])` 的结果。因为 `sapply()` 的结果类型的不确定性，在自定义函数中应慎用。

`vapply()` 函数与 `sapply()` 函数类似，但是它需要第三个参数即函数返回值类型的例子，格式为

```
vapply(X, FUN, FUN.VALUE, ...)
```

其中 `FUN.VALUE` 是每个 `FUN(X[i])` 的返回值的例子，要求所有 `FUN(X[i])` 结果类型和长度相同。

例如，求 `d.class` 每一列类型的问题，用 lapply，写成：

```
lapply(d.class, typeof)
```

```
## $name
## [1] "character"
##
## $sex
```

```
## [1] "integer"
##
## $age
## [1] "double"
##
## $height
## [1] "double"
##
## $weight
## [1] "double"
```

lapply 的结果总是列表。sapply 会尽可能将结果简化为向量或矩阵，如：

```
sapply(d.class, typeof)
```

```
##        name         sex         age      height      weight
## "character"   "integer"    "double"    "double"    "double"
```

或使用 vapply():

```
vapply(d.class, typeof, "")
```

```
##        name         sex         age      height      weight
## "character"   "integer"    "double"    "double"    "double"
```

vapply 可以处理遍历时每次调用的函数返回值不是标量的情形，结果为矩阵，purrr 包的 map_dbl 等只能处理调用的函数返回值是标量的情形。

R 提供了 Map, Reduce, Filter, Find, Negate, Position 等支持函数式编程的泛函。

Map() 与 purrr::map、purrr::pmap 功能类似，以一个函数作为参数，可以对其它参数的每一对应元素进行变换，结果为列表。

例如，对数据框 d，如下的程序可以计算每列的平方和：

```
d <- data.frame(
  x = c(1, 7, 2),
  y = c(3, 5, 9))
Map(function(x) sum(x^2), d)
```

```
## $x
## [1] 54
```

```
##
## $y
## [1] 115
```

实际上，这个例子也可以用 lapply() 改写成

```
lapply(d, function(x) sum(x^2))
```

```
## $x
## [1] 54
##
## $y
## [1] 115
```

Map() 比 lapply() 增强的地方在于它允许对多个列表的对应元素逐一处理。例如，为了求出 d 中每一行的最大值，可以用

```
Map(max, d$x, d$y)
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] 7
##
## [[3]]
## [1] 9
```

可以用 unlist() 函数将列表结果转换为向量，如

```
unlist(Map(max, d$x, d$y))
```

```
## [1] 3 7 9
```

mapply() 函数与 Map() 类似，但是可以自动简化结果类型，可以看成是 sapply() 推广到了可以对多个输入的对应元素逐项处理。mapply() 可以用参数 MoreArgs 指定逐项处理时一些共同的参数。如

```
mapply(max, d$x, d$y)
```

```
## [1] 3 7 9
```

当 d 数据框有多列时为了求每行的最大值，可以用 Reduce 函数将两两求最大值的运算推广到多个之间的运算。

Reduce 函数功能与 purrr::reduce 类似，把输入列表（或向量）的元素逐次地用给定的函数进行合并计算。

例如，求四个集合的交集：

```
set.seed(5)
x <- replicate(4, sample(
  1:5, size=5, replace=TRUE), simplify=FALSE); x
```

```
## [[1]]
## [1] 2 3 1 3 1
##
## [[2]]
## [1] 1 5 3 3 2
##
## [[3]]
## [1] 5 4 2 5 3
##
## [[4]]
## [1] 1 4 3 2 5
```

```
Reduce(intersect, x)
```

```
## [1] 2 3
```

Reduce 函数对多个输入默认从左向右计算，可以用 right 参数选择是否从右向左合并。参数 init 给出合并初值，参数 accumulate 要求保留每一步合并的结果（累计）。这个函数可以把很多仅适用于两个运算元的运算推广到多个参数的情形。

Filter(f, x) 与 purrr::keep 作用类似，用一个示性函数 f 作为筛选规则，从列表或向量 x 中筛选出用 f 作用后为真值的元素子集。例如

```
f <- function(x) x > 0 & x < 1
Filter(f, c(-0.5, 0.5, 0.8, 1))
```

```
## [1] 0.5 0.8
```

当然，这样的简单例子完全可以改写成：

```
x <- c(-0.5, 0.5, 0.8, 1)
x[x>0 & x < 1]
```

```
## [1] 0.5 0.8
```

但是，对于比较复杂类型的判断，比如当 x 是列表且其元素本身也是复合类型的时候，就需要把判断写成一个函数，然后可以用 Filter 比较简单地表达按照判断规则取子集的操作。

Find() 功能与 purrr::detect 类似，返回满足条件的第一个元素，也可以用参数 right=TRUE 要求返回满足条件的最后一个。

Position() 功能与 purrr::detect_index 类似，返回第一个满足条件的元素所在的下标位置。

25.2.6 自定义泛函

用户也可以自定义泛函。比如，希望对一个数据框中所有的数值型变量计算某些统计量，要计算的统计量由用户决定而不是由此自定义函数决定，输入的函数的结果总是数值，编写自定义的泛函为：

```
library(purrr)
summary.df.numeric <- function(df, FUN, ...){
  df |>
    keep(is.numeric) |>
    map_dbl(FUN, ...)
}
```

这里参数 FUN 是用来计算统计量的函数。例如对 d.cancer 中每个数值型变量计算最小值：

```
summary.df.numeric(d.cancer, min, na.rm=TRUE)
```

```
##    id   age    v0    v1
##  1.00 49.00 12.58  2.30
```

为了说明上面定义的泛函是如何对数据框进行处理的，我们对其进行如下的改写：

```
summary.df.numeric2 <- function(df, FUN, ...){
  res <- c()
  nd <- c()
  for(j in seq_along(df)){
    if(is.numeric(df[[j]])){
      resj <- FUN(df[[j]], ...)
      res <- cbind(res, resj)
```

```
      nd <- c(nd, names(df)[j])
    }
  }
  if(ncol(res)>0) {
    colnames(res) <- nd
    if(nrow(res) == 1) {
      res <- c(res)
      names(res) <- nd
    }
  }
  res
}
summary.df.numeric2(d.cancer, min, na.rm=TRUE)
```

```
##    id   age    v0    v1
##  1.00 49.00 12.58  2.30
```

也可以用基本 R 的 `Filter()`, `sapply()` 实现相同功能：

```
summary.df.numeric <- function(df, FUN, ...){
  Filter(is.numeric, df) |>
    sapply(FUN, ...)
}
```

25.3 函数工厂

函数的返回值可以是函数，为此只要在函数内部定义嵌套函数并以嵌套函数为返回值。返回函数的函数称为**函数工厂**，函数工厂的输出结果称为一个闭包 (closure)。因为函数由形参表、函数体和定义环境三个部分组成，函数工厂输出的闭包的定义环境是函数工厂的内部环境，即函数工厂运行时产生的运行环境，所以闭包包含了生产它的函数工厂的运行环境，可以将闭包的一些状态信息保存在该环境中，实现带有状态的函数。

基本 R 函数 `approxfun` 和 `splinefun` 就是以函数为输出的函数工厂。

25.3.1 闭包例子

利用函数工厂和闭包可以解决前面提出的记录函数已运行次数的问题。如

```
f.gen <- function(){
  runTimes <- 0
```

```
  function(){
    runTimes <<- runTimes + 1
    print(runTimes)
  }
}
f <- f.gen()
f()
```

```
## [1] 1
```

```
f()
```

```
## [1] 2
```

函数 `f.gen` 中定义了内嵌函数并以内嵌函数为输出，`f.gen` 是一个函数工厂, 其返回值是一个闭包，闭包也是一个 R 函数，这个返回值 “绑定”(bind) 到变量名 `f` 上，所以 `f` 是一个函数。

调用函数 `f` 时用到变量 `runTimes`，用了 `<<-`这种格式给这个变量赋值，这样赋值的含义是在定义时的环境中逐层向上 (向外，向父环境方向) 查找变量是否存在，在哪一层找到变量就给那里的变量赋值。这样查找的结果是变量 `runTimes` 在 `f.gen` 的运行环境中。调用 `f` 的时候 `f.gen` 已经结束运行了，一般说来 `f.gen` 的运行环境应该已经不存在了；但是，函数的定义环境是随函数本身一同保存的，因为函数工厂 `f.gen` 输出了函数 `f`，`f` 的定义环境是 `f.gen` 的运行环境，所以起到了把 `f.gen` 的运行环境保存在 `f` 中的效果，而 `f.gen` 运行环境中的变量值 `runTimes` 也就保存在了函数 `f` 中，可以持续被 `f` 访问，不像 `f` 的局部变量那样每次运行结束就会被清除掉。

注意，如果函数工厂生产出了两个闭包，这两个闭包的定义环境是不同的，因为生产时的运行环境是不同的。例如，生产两个计数器，这两个计数器是分别计数的：

```
c1 <- f.gen()
c2 <- f.gen()
c1()
```

```
## [1] 1
```

```
c1()
```

```
## [1] 2
```

```
c2()
```

```
## [1] 1
```

下面是一个类似的函数工厂例子, 产生的闭包可以显示从上次调用到下次调用之间经过的时间：

```
make_stop_watch <- function(){
  saved.time <- proc.time()[3]

  function(){
    t1 <- proc.time()[3]
    td <- t1 - saved.time
    saved.time <<- t1
    cat(" 流逝时间（秒）: ", td, "\n")
    invisible(td)
  }
}
ticker <- make_stop_watch()
ticker()
## 流逝时间（秒）:  0
for(i in 1:1000) sort(runif(10000))
ticker()
## 流逝时间（秒）:  1.53
```

其中 `proc.time()` 返回当前的 R 会话已运行的时间，结果在 MS Windows 系统中有三个值，分别是用户时间、系统时间、流逝时间，其中流逝时间比较客观。

25.3.2 动态查找和懒惰求值引起的问题

上面的两个函数工厂都没有使用任何选项。如果函数工厂有选项，其中的选项值会被保存到生产出的闭包函数中，但是因为懒惰求值规则的影响，有可能调用闭包函数时才对选项求值，如果保存选项的变量在生产和调用之间改变了值，就会发生错误。

比如，下面的函数工厂可以生产出进行幂变换的函数：

```
make.pf <- function(power){
  function(x) x^power
}
p <- 2
square <- make.pf(p)
p <- 3
square(2)
```

```
## [1] 8
```

在生产出 square 函数时，选项 p 的值是 2，所以函数 square 应该是做平方变换的函数，虽然在调用之前 p 的值被改成了 3，但是按理说不应该修改已经生产出来的 square 定义。程序结果说明调用 square 时用的是 p=3 的值，这是怎么回事？

R 函数有懒惰求值规则，在生产出 square 的那一步，因为并不需要实际计算 x^power，所以实参 p 的值并没有被计算，而是将 square 的定义环境中的 power 指向了全局空间的变量 p，调用 square(4) 的时候才实际需要 power 的值，这时 power 才求值，其值为 p 的**当前**值。

避免这样的问题的办法是在函数工厂内用 force() 函数命令输入的参数当场求值而不是懒惰求值。如：

```
make.pf <- function(power){
  force(power)
  function(x) x^power
}
p <- 2
square <- make.pf(p)
p <- 3
square(2)
```

```
## [1] 4
```

这个版本的程序结果正确。

25.3.3 函数工厂的内存负担

因为函数工厂生产出的闭包函数保存了函数工厂的运行环境，如果这个运行环境很大，就会造成较大的不必要的内存占用。所以，函数工厂内应尽量不要有占用大量内存的变量。可以在函数工厂内用 rm() 删除不再使用的变量。

25.3.4 函数算子

函数算子输入函数，输出函数，通常用来对输入函数的行为进行改进或做细微的修改。基本 R 的 Vectorize 函数输入一个函数，将其改造成支持向量化的版本。

下面的 dot_every 函数输入一个函数，将其改造为被循环调用时可以每调用一定次数就显示一个小数点，这样可以用来显示循环的进度，也适用于在 lapply() 或 purrr::map() 等函数调用时显示进度。purrr::map() 有一个.progress 选项可以显示进度。

```
dot_every <- function(f, n) {
  force(f)
  force(n)

  i <- 0
```

```
  function(...) {
    i <<- i + 1
    if (i %% n == 0) cat(".")
    f(...)
  }
}
sim <- function(i){
  x <- runif(1E6)
  invisible(sort(x))
}
walk(1:100, dot_every(sim, 10))
```

使用.progress 选项：

```
walk(1:100, sim, .progress = TRUE)
```

25.4 tibble 中的列表列

25.4.1 nest 和 unnest

dplyr 包的 group_by 与 summarise、summarise_at 等函数配合，可以对数据框分组计算各种概括统计量。

但是，如果分组以后希望进行更复杂的统计分析，比如分组回归建模，summarise 就不够用了。这时，可以用基本 R 的 split 函数将数据框按某个分类变量拆分为子数据框的列表，然后用 purrr 包的 map 类函数分类建模，最终将各个模型的结果合并为一个数据框。参见25.2.1.7。

上面的办法虽然可行，但是管理不够方便。tidyr 包（属于 tidyverse 系列，载入 tidyverse 时会自动载入）提供了 nest 和 unnest 函数，可以将子数据框保存在 tibble 中，可以将保存在 tibble 中的子数据框合并为一个大数据框。实际上，tibble 允许存在数据类型是列表 (list) 的列，子数据框就是以列表数据类型保存在 tibble 的一列中的。

25.4.2 group_by 与 nest 配合

对数据框用 group_by 分组后调用 nest 函数就可以生成每个组的子数据框。

例如，将 d.cancer 数据框按 type 分类拆分为 2 个子数据框，存入 tibble 的 data 列中：

```
d.cancer |>
  group_by(type) |>
  nest()
```

```
## # A tibble: 2 x 2
## # Groups:   type [2]
##   type  data
##   <chr> <list>
## 1 腺癌  <tibble [12 x 5]>
## 2 鳞癌  <tibble [22 x 5]>
```

现在 data 列是列表类型的，有 2 个元素，每个元素是一个子数据框。group_by() 中也可以用两个或多个分类变量构成交叉分组。

可以用 purrr 包的 map() 等函数对每个子数据框进行处理，结果可以用 mutate 保存为新的列表类型的列，如果结果是数值型标量也可以保存为普通的数据框列。

例如，下面先定义对子数据框回归建模的函数，然后用 purrr 包的 map 函数将回归建模的函数作用到 data 列的每个元素，用 mutate 保存到列表类型的 lmr 列中：

```
fmod <- function(subdf)  lm(v1 ~ v0, data = subdf)
mod.cancer <- d.cancer |>
  group_by(type) |>
  nest() |>
  mutate(lmr = map(data, fmod))
mod.cancer
```

```
## # A tibble: 2 x 3
## # Groups:   type [2]
##   type  data              lmr
##   <chr> <list>            <list>
## 1 腺癌  <tibble [12 x 5]> <lm>
## 2 鳞癌  <tibble [22 x 5]> <lm>
```

上面程序的 mutate 中的 map(data, fmod)，data 是用 nest() 生成的结果数据框中 data 列，此列为一个两个元素的列表，每个元素是一个子数据框，用 map(data, fmod) 对每个子数据框分别建立回归模型，得到的两个回归模型，作为两个列表元素组合成一个列表，存放在结果数据框的 lmr 列中，lmr 列的类型是列表。

下面写一个函数从一个回归模型及相应子数据框中，提取 R 方，将提取的结果保存为普通数值型列 r.squared：

```
frsqr <- function(mod){
  summary(mod)$r.squared
}
mod.cancer |>
  mutate(
    r.squared = map_dbl(lmr, frsqr)) |>
  select(-data, -lmr)
```

```
## # A tibble: 2 x 2
## # Groups:   type [2]
##   type  r.squared
##   <chr>     <dbl>
## 1 腺癌      0.710
## 2 鳞癌      0.520
```

map() 和 map_dbl() 中输入函数时可以用 purrr 包的无名函数写法，如：

```
d.cancer |>
  group_by(type) |>
  nest() |>
  mutate(
    lmr = map(data, ~ lm(v1 ~ v0, data = .x)),
    r.squared = map_dbl(lmr, ~ summary(.x)$r.squared)) |>
  select(-data, -lmr)
```

```
## # A tibble: 2 x 2
## # Groups:   type [2]
##   type  r.squared
##   <chr>     <dbl>
## 1 腺癌      0.710
## 2 鳞癌      0.520
```

也可以从每个模型提取多个值，这时，为了使得多个值在展开时能保存在同一行中，需要将每个子数据框的提取结果保存为一个一行的子数据框：

```
fextract <- function(mod){
  x1 <- coef(mod)
  tibble(
    intercept = x1[1],
    v0 = x1[2],
    r.squared = summary(mod)$r.squared)
}
mod.cancer |>
  mutate(
    outlist = map(lmr, fextract))
```

```
## # A tibble: 2 x 4
## # Groups:   type [2]
```

```
##   type  data              lmr    outlist
##   <chr> <list>            <list> <list>
## 1 腺癌  <tibble [12 x 5]> <lm>   <tibble [1 x 3]>
## 2 鳞癌  <tibble [22 x 5]> <lm>   <tibble [1 x 3]>
```

结果的 outlist 列是列表类型的，每个元素是一个 1×3 的 tibble。下面，就可以用 unnest() 将每个组提取的回归结果转换为普通的数据框列：

```
mod.cancer |>
  mutate(
    outlist = map(lmr, fextract)) |>
  select(-data, -lmr) |>
  unnest(outlist)
```

```
## # A tibble: 2 x 4
## # Groups:   type [2]
##   type  intercept    v0 r.squared
##   <chr>     <dbl> <dbl>     <dbl>
## 1 腺癌      0.225 0.370     0.710
## 2 鳞癌      5.51  0.374     0.520
```

提取的结果也可以是一个不止一行的子数据框，例如，提取回归结果中的系数估计、标准误差、t 统计量和检验 p 值组成的矩阵：

```
fcoefmat <- function(mod){
  as_tibble(
    summary(mod)$coefficients,
    rownames="term")
}
mod.cancer |>
  mutate(
    outlist = map(lmr, fcoefmat)) |>
  select(-data, - lmr) |>
  unnest(outlist)
```

```
## # A tibble: 4 x 6
## # Groups:   type [2]
##   type  term        Estimate `Std. Error` `t value` `Pr(>|t|)`
##   <chr> <chr>          <dbl>        <dbl>     <dbl>      <dbl>
## 1 腺癌  (Intercept)    0.225       10.2       0.0221   0.983
```

```
## 2 腺癌  v0            0.370       0.0749    4.94     0.000585
## 3 鳞癌  (Intercept)   5.51       10.8       0.510    0.616
## 4 鳞癌  v0            0.374       0.0803    4.66     0.000151
```

为了更好地提取统计模型的信息为规整的数据框格式，broom 扩展包提供了 `tidy` 函数，可以将统计模型的输出转换为数据框，这些功能与 tidyr 的 `nest`, `unnest` 配合，可以很好地提取统计模型的信息，如：

```
mod.cancer |>
  mutate(
    outlist = map(lmr, broom::tidy)) |>
  select(-data, -lmr) |>
  unnest(outlist)
```

```
## # A tibble: 4 x 6
## # Groups:   type [2]
##   type  term        estimate std.error statistic  p.value
##   <chr> <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 腺癌  (Intercept)   0.225   10.2       0.0221 0.983
## 2 腺癌  v0            0.370    0.0749    4.94   0.000585
## 3 鳞癌  (Intercept)   5.51    10.8       0.510  0.616
## 4 鳞癌  v0            0.374    0.0803    4.66   0.000151
```

`unnest` 提取出的信息也可以是一个向量，在展开时会展开到同一列中。例如，对每个组提取回归的拟合值：

```
mod.cancer |>
  mutate(
    v1hat = map(lmr, ~ fitted(.))) |>
  select(-lmr) |>
  unnest(c(data, v1hat))
```

```
## # A tibble: 34 x 7
## # Groups:   type [2]
##    type     id   age sex      v0     v1  v1hat
##    <chr> <dbl> <dbl> <chr> <dbl>  <dbl>  <dbl>
##  1 腺癌      1    70 F      26.5   2.91  10.0
##  2 腺癌      2    70 F     135.   35.1   50.4
##  3 腺癌      3    69 F     210.   74.4   77.9
##  4 腺癌      4    68 M      61    35.0   22.8
##  5 腺癌      6    75 F     330.  112.   123.
##  6 腺癌     11    55 M     125.   12.3   46.6
```

```
## 7 腺癌      13    55 F      12.9   2.3    5.01
## 8 腺癌      14    75 M      40.2  24.0   15.1
## 9 腺癌      15    61 F      12.6   7.39   4.88
## 10 腺癌     19    NA F      32.9   9.45  12.4
## # ... with 24 more rows
```

程序中的 unnest 将 data 列和 v1hat 列都释放为普通的数据框列了，data 列中释放出了多列原始数据，fitted 列中释放出了 v1 回归拟合值。

25.4.3 summarise 统计量用列表表示

实际上，summarise 等函数如果将结果用 list() 声明，汇总结果就可以保存为列表类型的列，结果可以包含多个值，unnest 可以将结果恢复成正常的数据框，如：

```
vnames <- expand_grid(
  var = c("v0", "v1"),
  stat = c("min", "max")) |>
  pmap_chr(paste, sep="_")

d.cancer |>
  group_by(type) |>
  summarise(
    stat = list(vnames),
    value = list(c(range(v0), range(v1)))  ) |>
  unnest(c(stat, value))
```

```
## # A tibble: 8 x 3
##   type  stat    value
##   <chr> <chr>   <dbl>
## 1 腺癌  v0_min  12.6
## 2 腺癌  v0_max 330.
## 3 腺癌  v1_min   2.3
## 4 腺癌  v1_max 122.
## 5 鳞癌  v0_min  13.2
## 6 鳞癌  v0_max 238.
## 7 鳞癌  v1_min   3.34
## 8 鳞癌  v1_max 128.
```

这个例子可以用长宽表转换方法变成每个统计量占一列：

```
d.cancer |>
  group_by(type) |>
  summarise(
    stat = list(vnames),
    value = list(c(range(v0), range(v1)))  ) |>
  unnest(c(stat, value)) |>
  separate(stat, into = c("variable", "stat"), sep="_") |>
  pivot_wider(
    names_from = "stat",
    values_from = "value"
  )
```

```
## # A tibble: 4 x 4
##   type  variable   min   max
##   <chr> <chr>    <dbl> <dbl>
## 1 腺癌  v0       12.6   330.
## 2 腺癌  v1        2.3   122.
## 3 鳞癌  v0       13.2   238.
## 4 鳞癌  v1        3.34  128.
```

25.4.4 unnest 的语法格式

unnest() 第一自变量为管道输入的数据框，第二自变量 cols 可以用如下格式：

- 单个变量名，不需要写成字符串形式；
- 多个变量名，写成 c(x, y, z) 这样的格式，不需要写成字符型向量；
- 保存在字符型向量中的变量名，用 all_of(vnames) 格式，其中 vnames 是保存了要释放的列名的字符型向量的变量名; 也可以写成 all_of(c("x", "y", "z")) 这样的格式。

25.4.5 直接生成列表类型的列

也可以直接生成列表类型的列，符合条件时可以用 unnest() 合并为大数据框。如：

```
d1 <- tibble(
  id = 1:2,
  df = vector("list", length=2))
d1[["df"]][1] <- list(
  tibble(x=1, y=2)
)
```

```
d1[["df"]][2] <- list(
  tibble(x=11:12, y=21:22)
)
d1 |>
  unnest(cols = c(df))
```

```
## # A tibble: 3 x 3
##      id     x     y
##   <int> <dbl> <dbl>
## 1     1     1     2
## 2     2    11    21
## 3     2    12    22
```

但是，不能直接写成 `d1[["df"]][[1]] <- tibble(x=1, y=2)`。这样写需要提前将列表 `d1$df` 的元素类型设定为 tibble 类型。

Part VI

绘图

Chapter 26

基本 R 绘图

R 语言的前身是 S 语言，S 语言的设计目的就是交互式数据分析、绘图。所以绘图是 R 的重要功能。

R 有最初的基本绘图，这是从 S 语言继承过来的，还有一些功能更易用、更强大的绘图系统，如 lattice、ggplot2。基本绘图使用简单，灵活性强，但是为了做出满意的图形需要比较多的调整。这里先讲解 R 语言的基本绘图功能。

R 的基本绘图功能有两类图形函数：高级图形函数，直接针对某一绘图任务作出完整图形；低级图形函数，在已有图形上添加内容。具备有限的与图形交互的能力（函数 `locator` 和 `identify`）。

26.1 常用高级图形

26.1.1 条形图

`d.cancer` 数据框包含了肺癌病人放疗的一些数据，从`data/cancer.csv`读入：

```
d.cancer <- readr::read_csv("data/cancer.csv",
    locale=locale(encoding="GBK"))
```

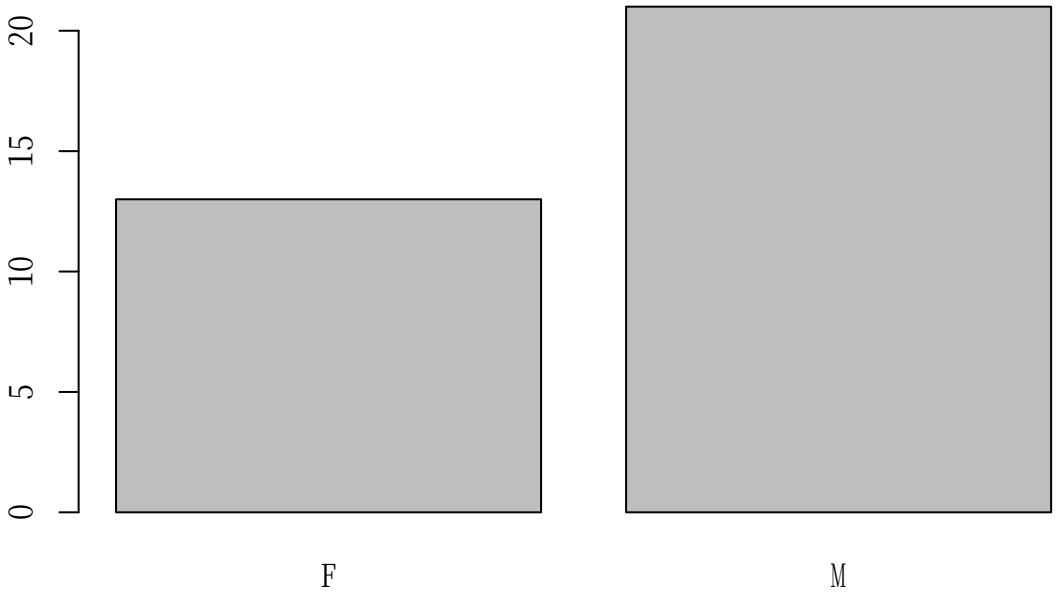
```
## Rows: 34 Columns: 6
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): sex, type
## dbl (4): id, age, v0, v1
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

统计男女个数并用条形图表示：

```
res1 <- table(d.cancer[,'sex']); print(res1)
```

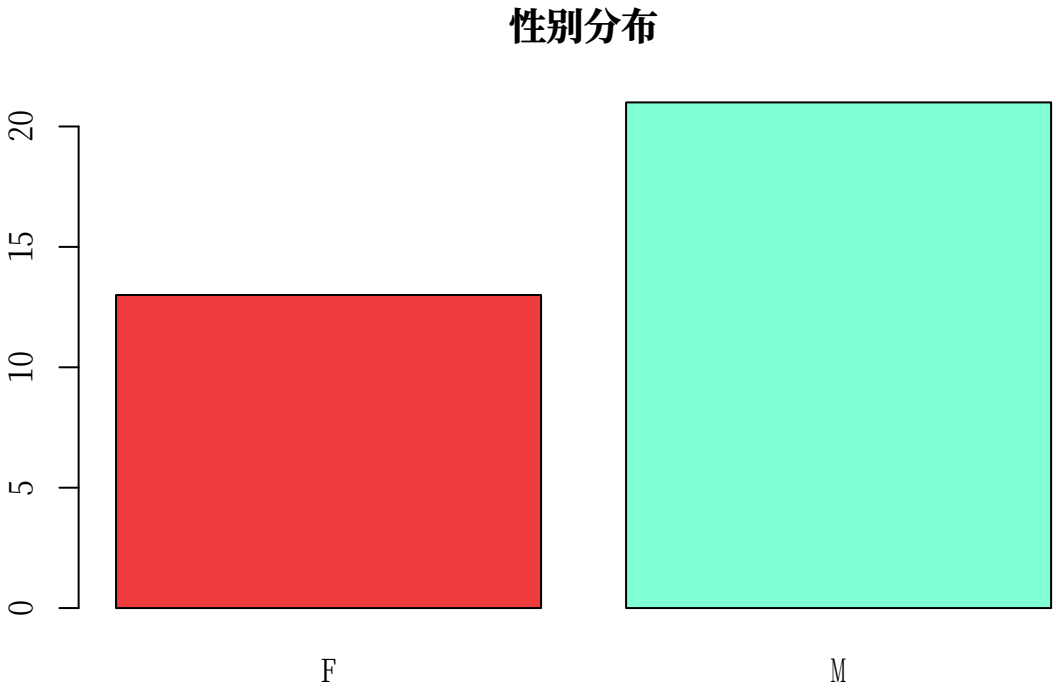
```
## sex
##  F  M
## 13 21
```

```
barplot(res1)
```



可以增加标题，采用不同的颜色：

```
barplot(res1, main=" 性别分布",
        col=c("brown2", "aquamarine1"))
```



R 函数 `colors()` 可以返回 R 中定义的用字符串表示的六百多种颜色名字。如

```
head(colors(), 6)
```

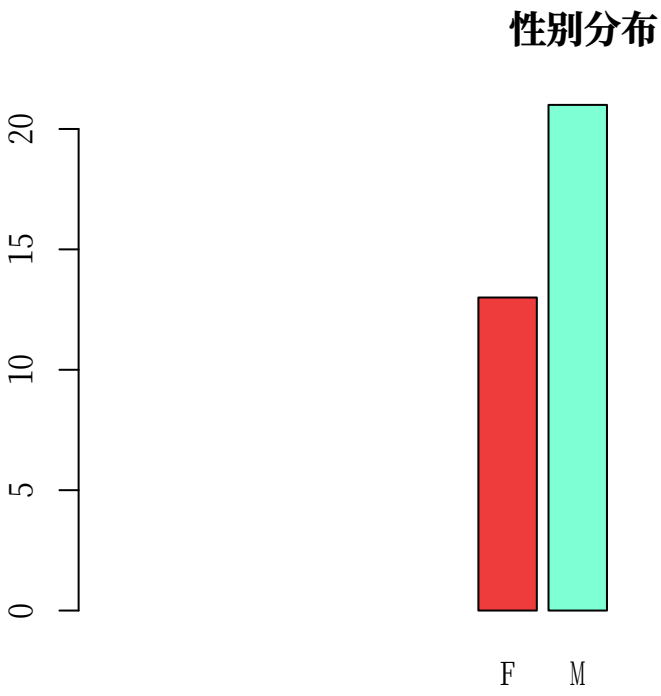
```
## [1] "white"         "aliceblue"     "antiquewhite"  "antiquewhite1"
## [5] "antiquewhite2" "antiquewhite3"
```

下面的函数可以用来挑选颜色，鼠标点击画出的颜色就可以挑选，结果返回挑选出的颜色名：

```
select.colors <- function(){
  nc <- length(colors())
  x <- rep(seq(26), 26)[1:nc]
  y <- rep(seq(26), each=26)[1:nc]
  cols <- colors()
  plot(x, y, type="p", pch=16, cex=2,
       col=cols)
  res <- cols[identify(x,y, labels=cols)]
  res
}
```

用 width 选项与 xlim 选项配合可以调整条形宽度，如

```
barplot(res1, width=0.5, xlim=c(-3, 5),
        main=" 性别分布",
        col=c("brown2", "aquamarine1"))
```



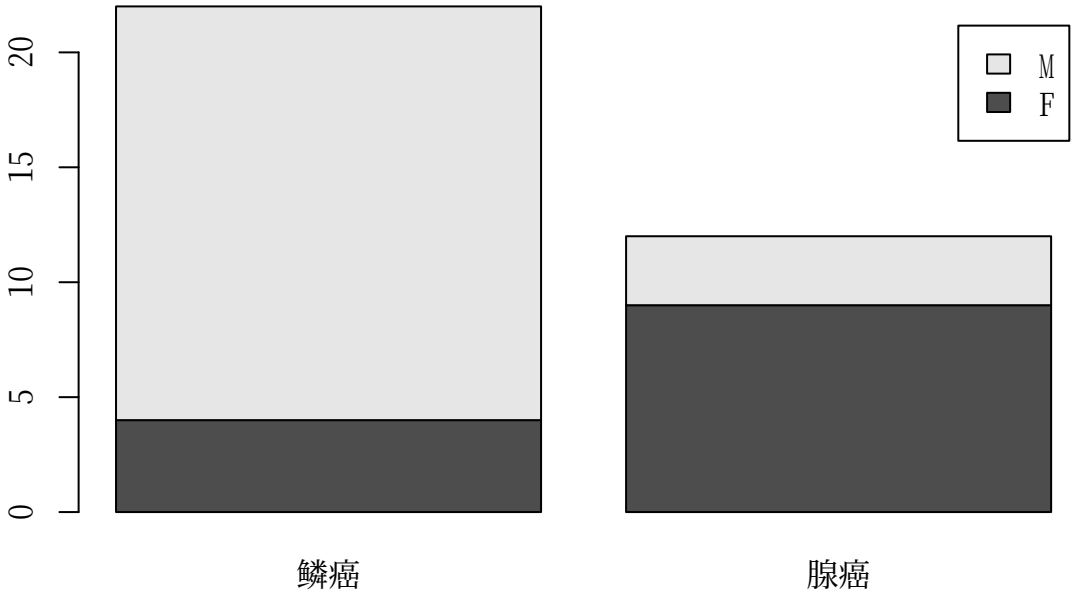
按性别与病理类型交叉分组后统计频数，结果称为列联表：

```
res2 <- with(d.cancer, table(sex, type)); res2
```

```
##    type
## sex 鳞癌 腺癌
##   F    4    9
##   M   18    3
```

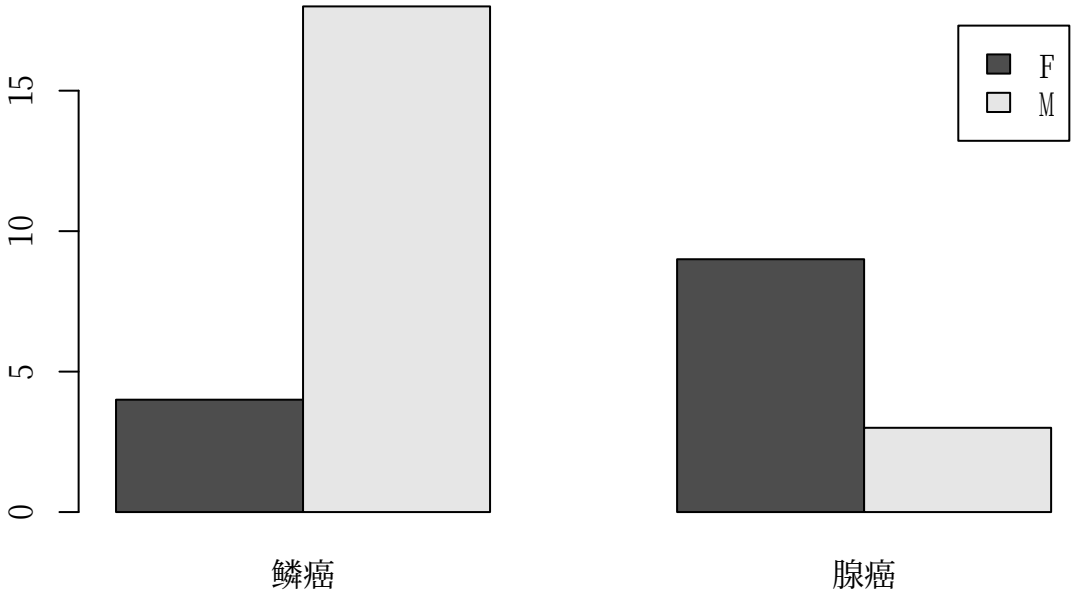
用分段条形图表现交叉分组频数，交叉频数表每列为一条：

```
barplot(res2, legend=TRUE)
```



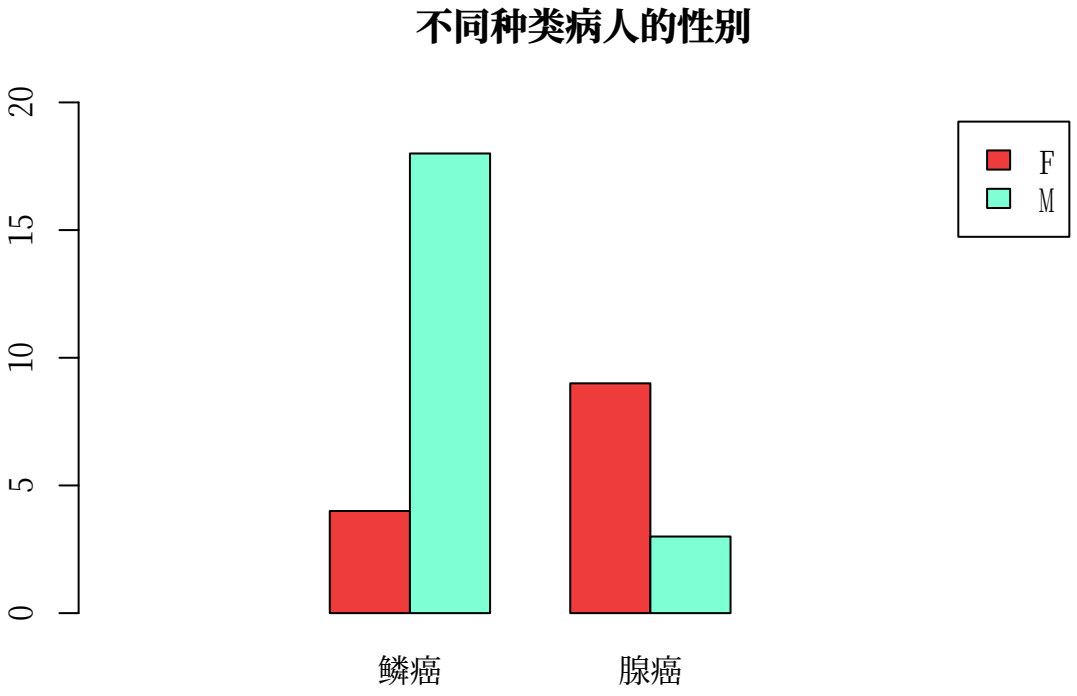
用并排条形图表现交叉分组频数，交叉频数表每列为一组：

```
barplot(res2, beside=TRUE, legend=TRUE)
```



增加标题，指定颜色，调整图例位置，调整条形宽度：

```
barplot(res2, beside=TRUE, legend=TRUE,
        main='不同种类病人的性别',
        ylim=c(0, 20),
        xlim=c(-1, 6), width=0.6,
        col=c("brown2", "aquamarine1"))
```



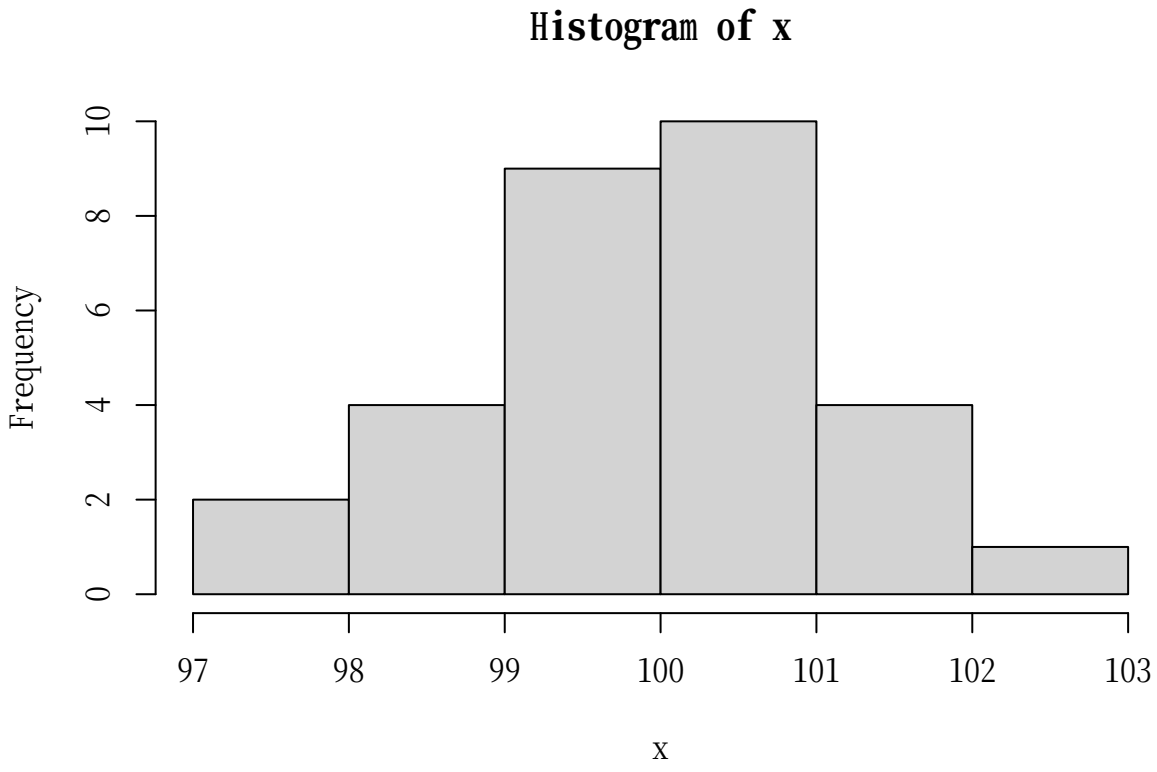
26.1.2 直方图和密度估计图

用 hist 作直方图以了解连续取值变量分布情况。例如，下面的程序模拟正态分布数据并做直方图：

```
x <- rnorm(30, mean=100, sd=1)
print(round(x,2))
```

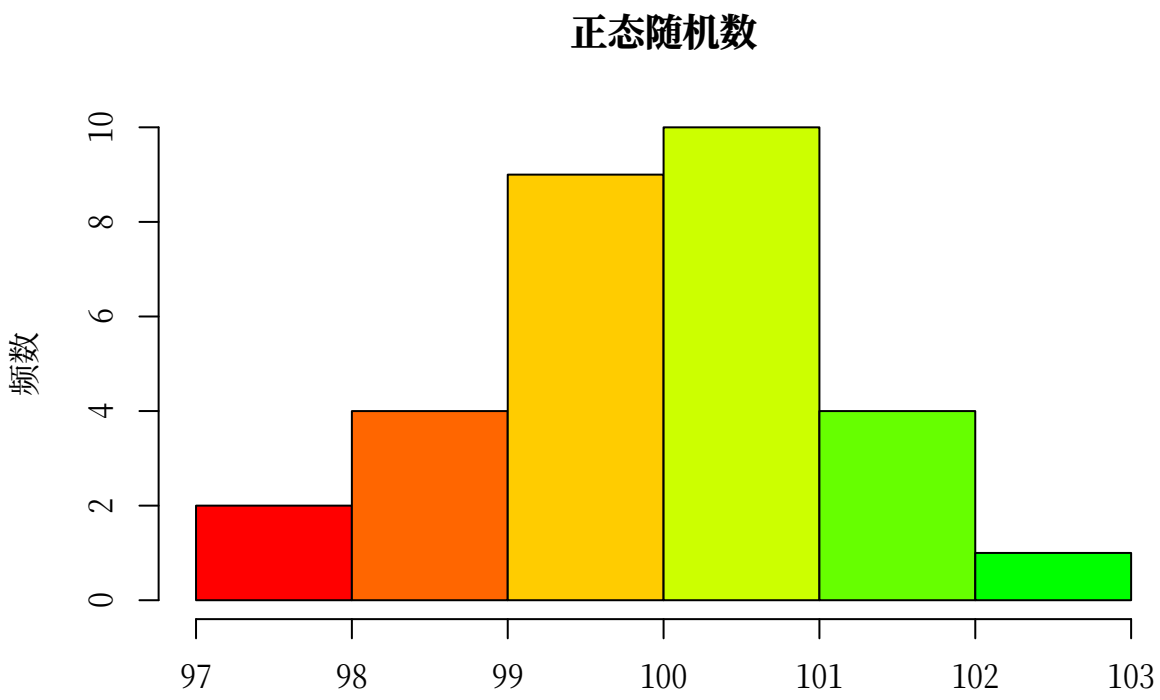
```
##  [1]  99.65  97.82 100.17 100.91 100.79  99.44 100.29  98.11  97.89  99.82
## [11] 100.46  99.18 100.72  98.84 101.25 101.26 101.31  98.90  99.75  98.62
## [21] 101.30  99.57 100.69 100.31 102.10  99.89  99.05 100.33 100.32  99.24
```

```
hist(x)
```



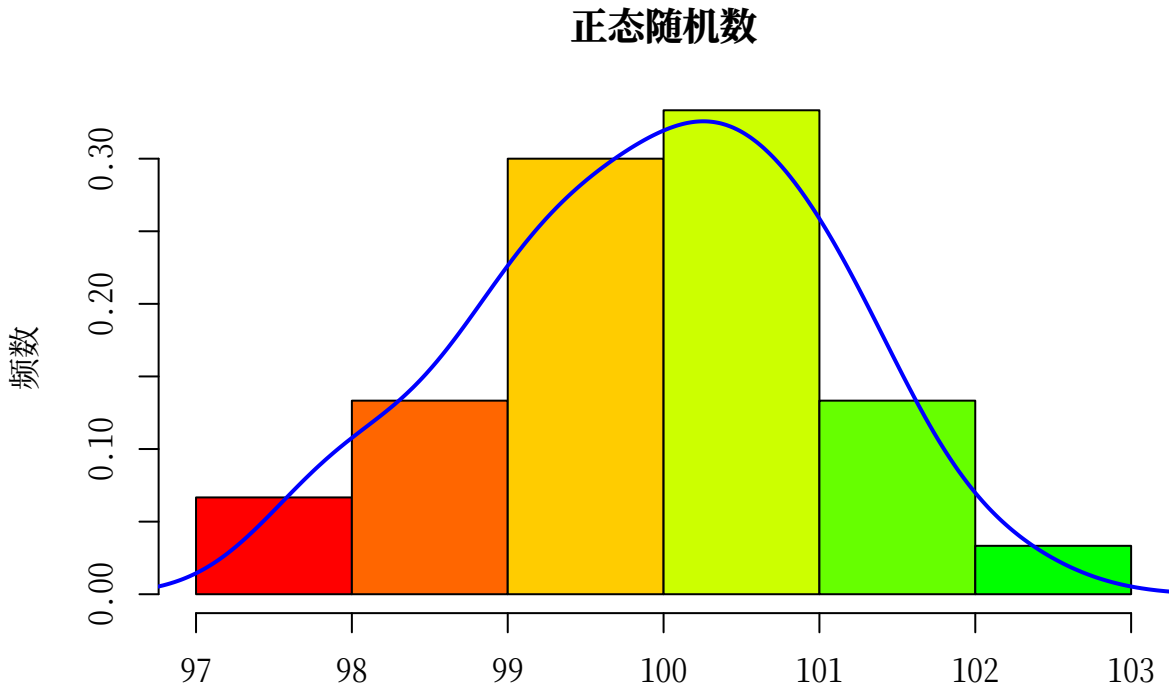
可以用 `main=`、`xlab=`、`ylab=` 等选项，可以用 `col=` 指定各个条形的颜色，如：

```
hist(x, col=rainbow(15),
     main='正态随机数', xlab='', ylab='频数')
```



函数 density() 估计核密度。下面的程序作直方图，并添加核密度曲线：

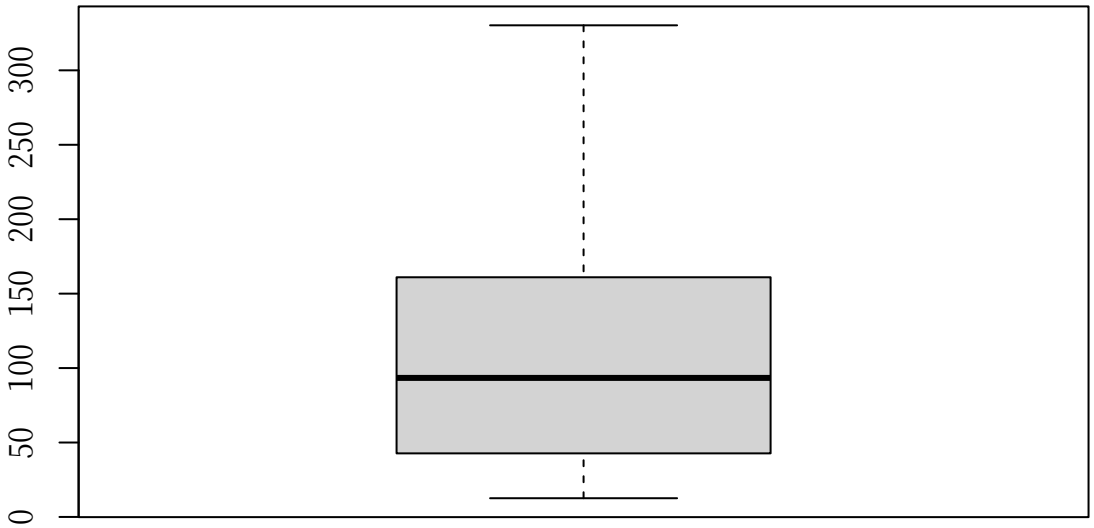
```
tmp.dens <- density(x)
hist(x, freq=FALSE,
     ylim=c(0,max(tmp.dens$y)),
     col=rainbow(15),
     main='正态随机数',
     xlab='', ylab='频数')
lines(tmp.dens, lwd=2, col='blue')
```



26.1.3 盒形图

盒形图可以简洁地表现变量分布，如

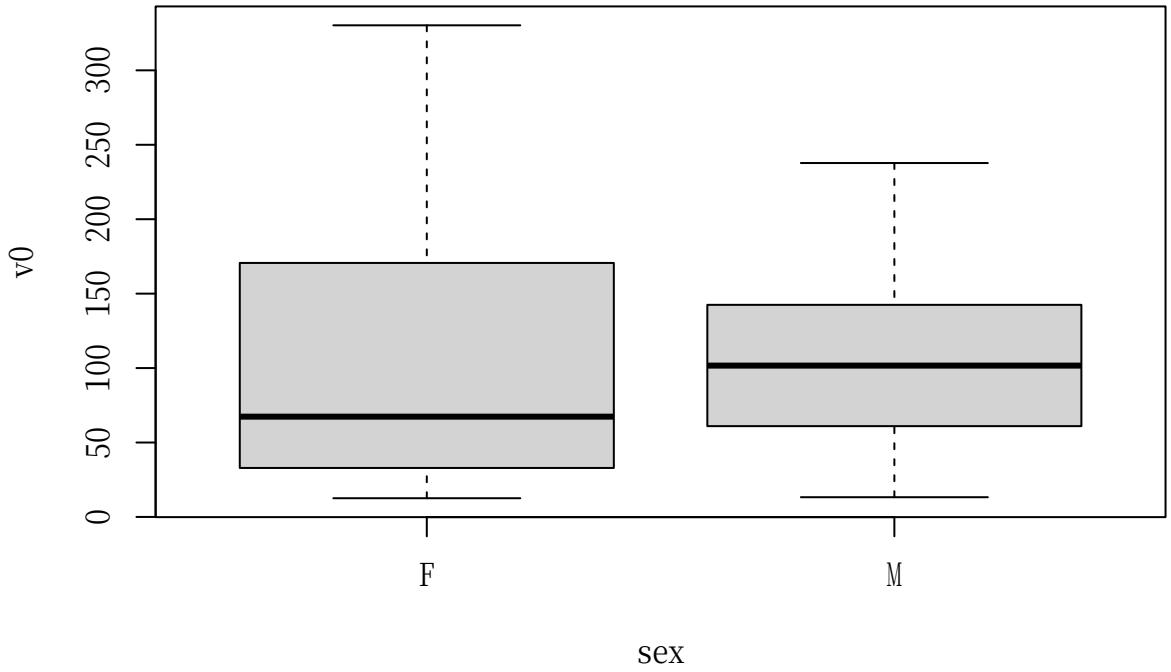
```
with(d.cancer, boxplot(v0))
```



其中中间粗线是中位数，盒子上下边缘是 $\frac{3}{4}$ 和 $\frac{1}{4}$ 分位数，两条触须线延伸到取值区域的边缘。

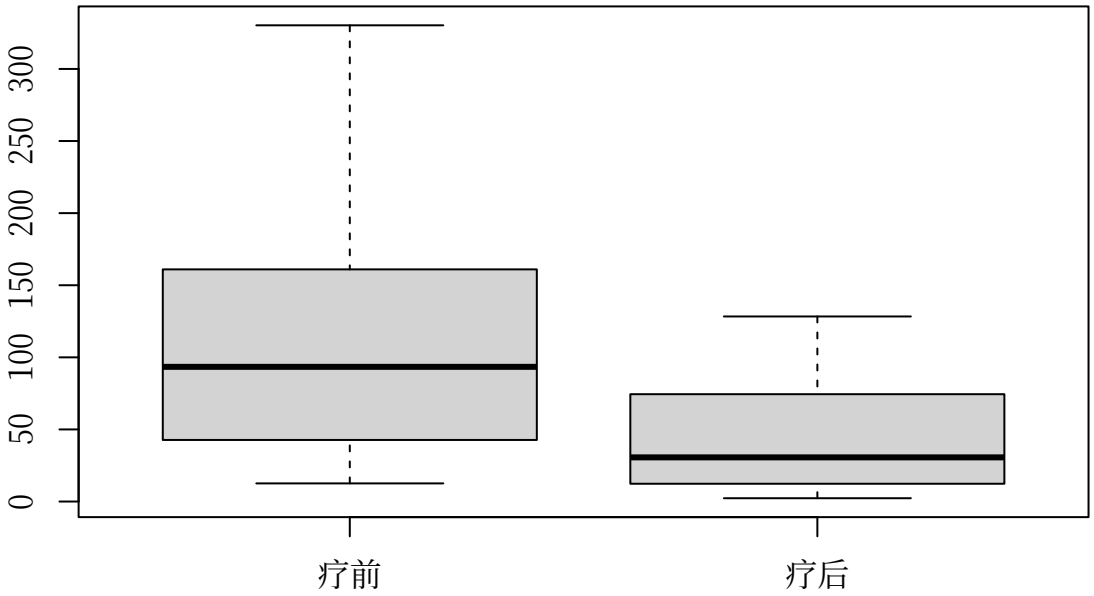
盒形图可以很容易地比较两组或多组，如

```
with(d.cancer, boxplot(v0 ~ sex))
```



也可以画若干个变量的并排盒形图，如

```
with(d.cancer,
     boxplot(list('疗前'=v0, '疗后'=v1)))
```



26.1.4 正态 QQ 图

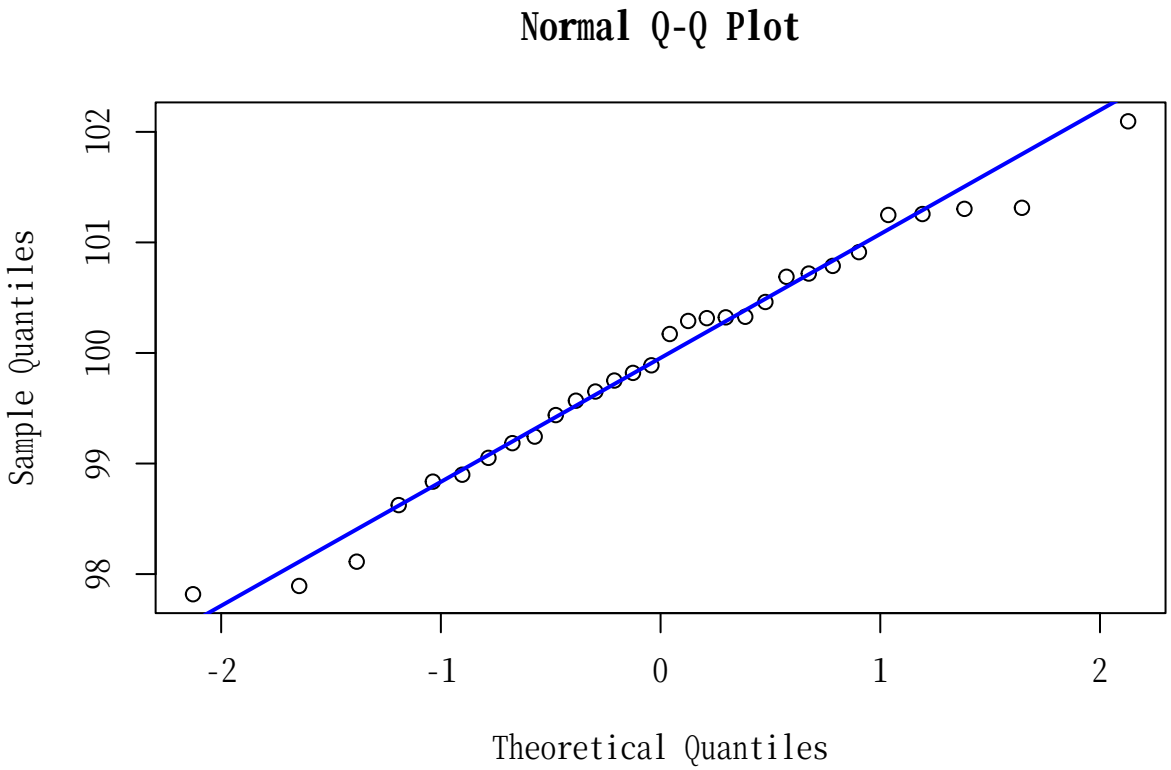
用 `qqnorm` 和 `qqline` 作正态 QQ 图。当变量样本来自正态分布总体时，正态 QQ 图的散点近似在一条直线周围。

QQ 图一般作法如下：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 $N(\mu,\sigma^2)$ 的分布函数为 $F(x)$，有 $y_i \approx F^{-1}(\frac{i}{n})$, $F(y_i)=\Phi(\frac{y_i-\mu}{\sigma}) \approx \frac{i}{n}$, $\frac{y_i-\mu}{\sigma} \approx \Phi^{-1}(\frac{i}{n}) = x_i$, $y_i \approx \mu+\sigma x_i$，于是用 (x_i, y_i) $(i=1,2,\dots,n)$ 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。

但是, 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, 相当于说总体分布不能超过 y_n, 这是不合理的。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 分位数而是对应于略调整的如 $(i-0.375)/(n+0.25)$ 这样的分位数, 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 $\frac{0.625}{n+0.25}$。

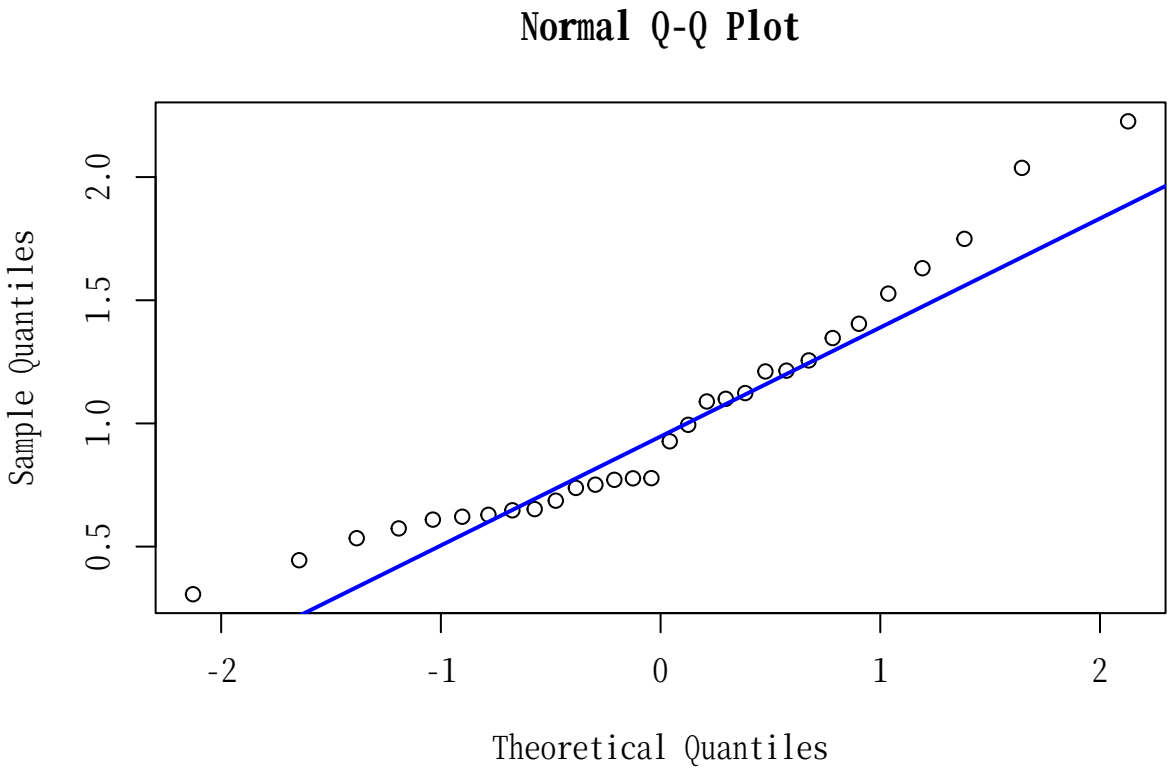
下面的程序模拟正态分布随机数，并作正态 QQ 图:

```
qqnorm(x)
qqline(x, lwd=2, col='blue')
```



下面的程序模拟对数正态数据，并作正态 QQ 图:

```
z <- 10^rnorm(30, mean=0, sd=0.2)
qqnorm(z)
qqline(z, lwd=2, col='blue')
```



26.1.5 散点图

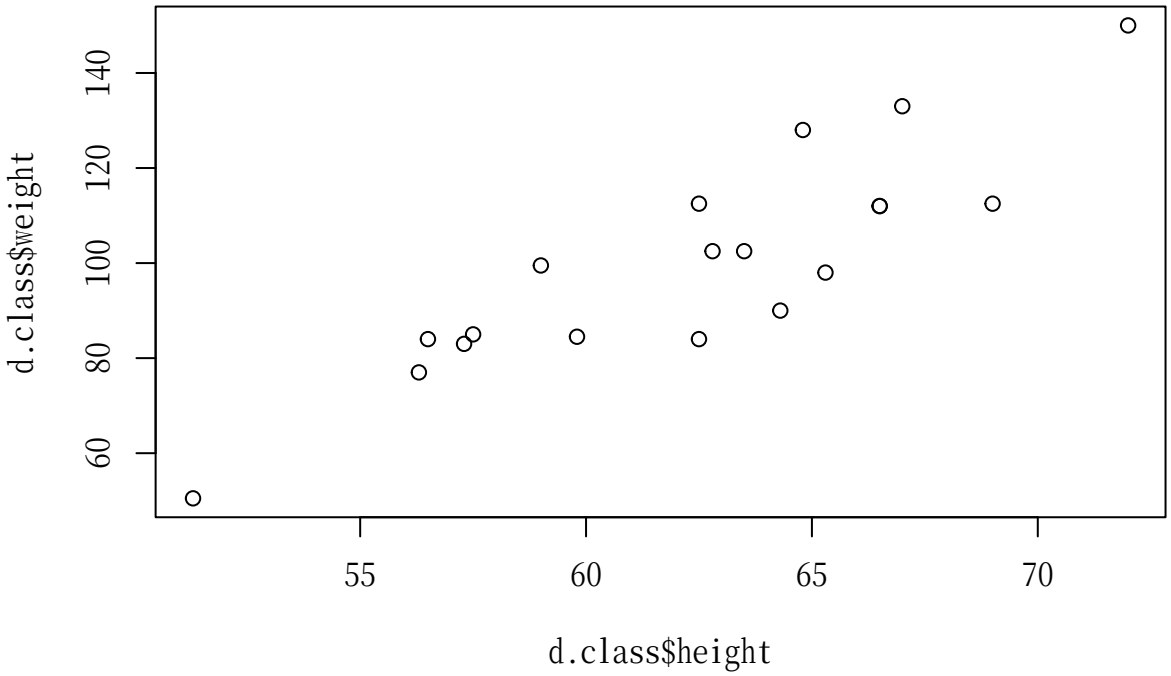
以 d.class 数据为例，有 name, sex, age, height, weight 等变量。从`data/class.csv`读入：

```
d.class <- read_csv("data/class.csv")
```

```
## Rows: 19 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): name, sex
## dbl (3): age, height, weight
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

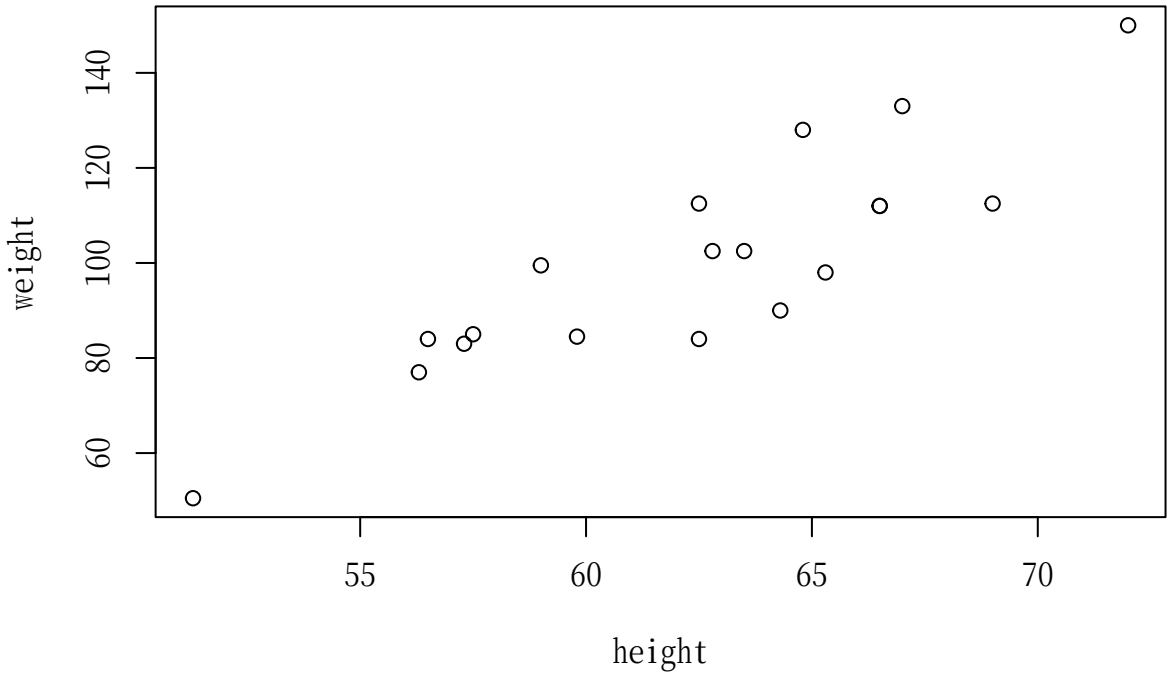
体重对身高的散点图：

```
plot(d.class$height, d.class$weight)
```



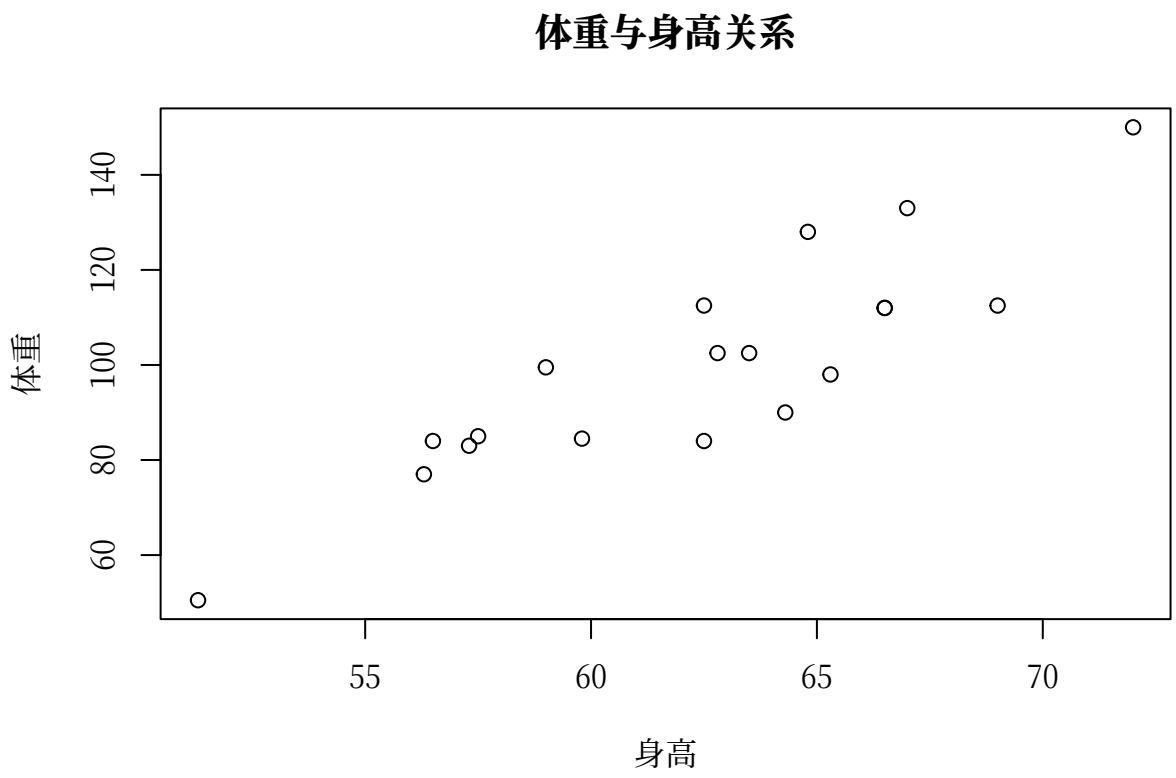
用 `with()` 函数简化数据框变量访问格式:

```
with(d.class,
     plot(height, weight))
```



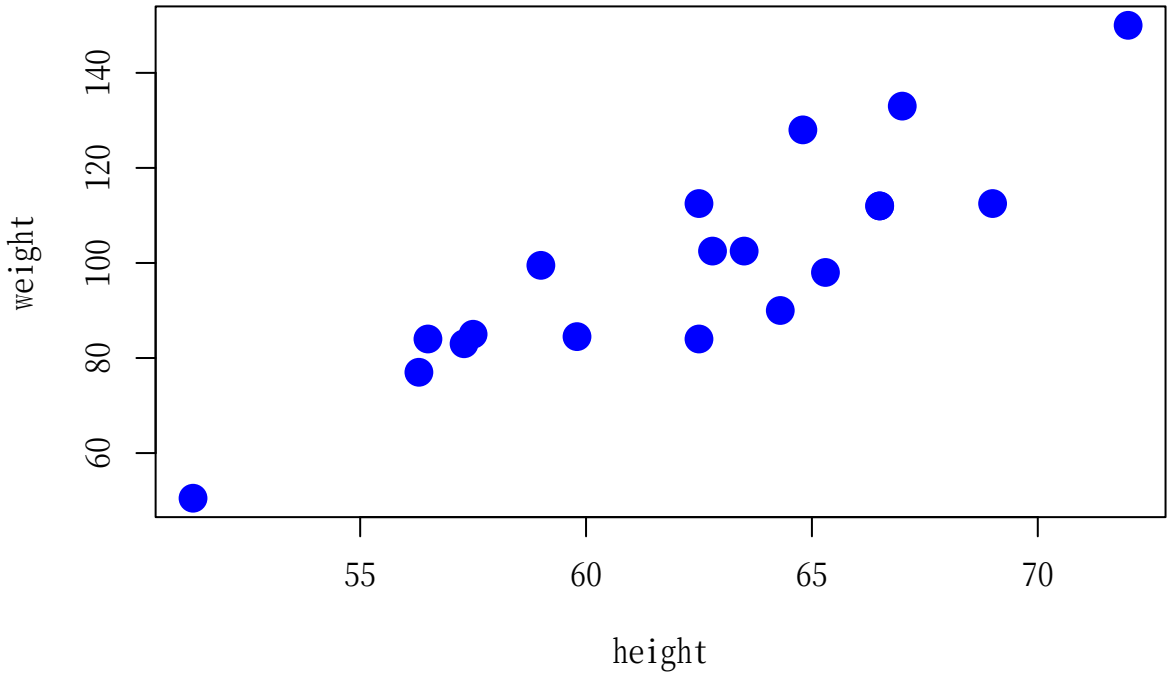
在 plot() 函数内用 main 参数增加标题，用 xlab 参数指定横轴标注，用 ylab 参数指定纵轴标注，如

```
with(d.class,
     plot(height, weight,
          main='体重与身高关系',
          xlab='身高', ylab='体重'))
```



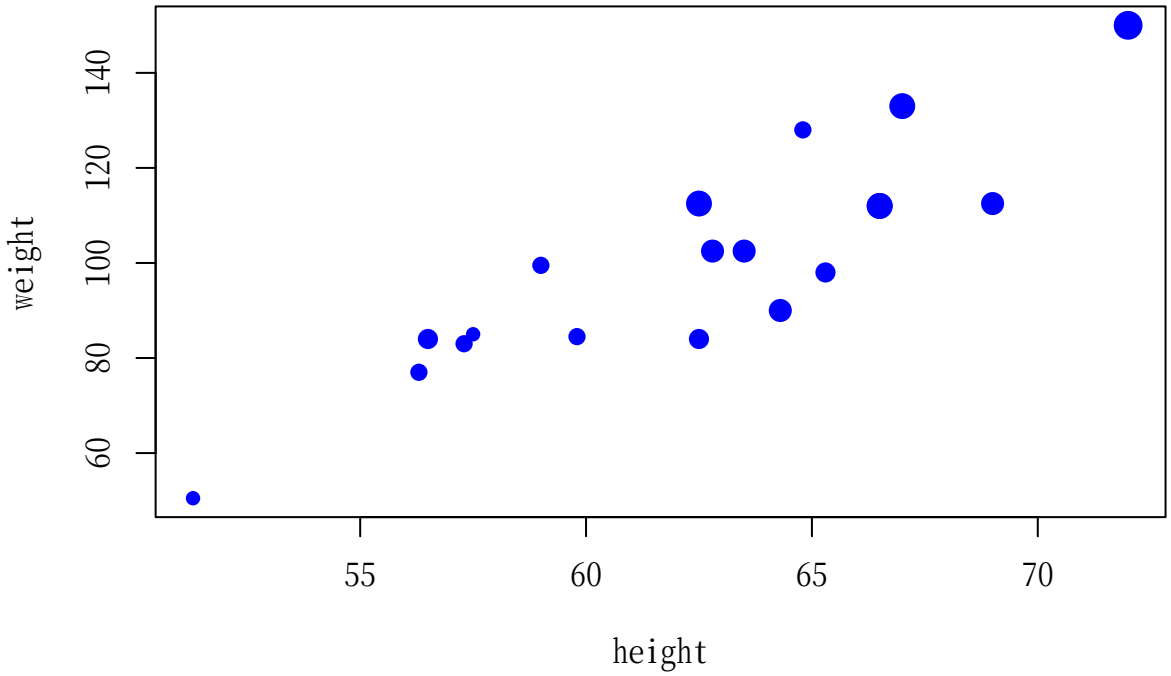
用 pch 参数指定不同散点形状，用 col 参数指定颜色，用 cex 参数指定大小，如：

```
with(d.class,
     plot(height, weight,
          pch=16, col='blue',
          cex=2))
```



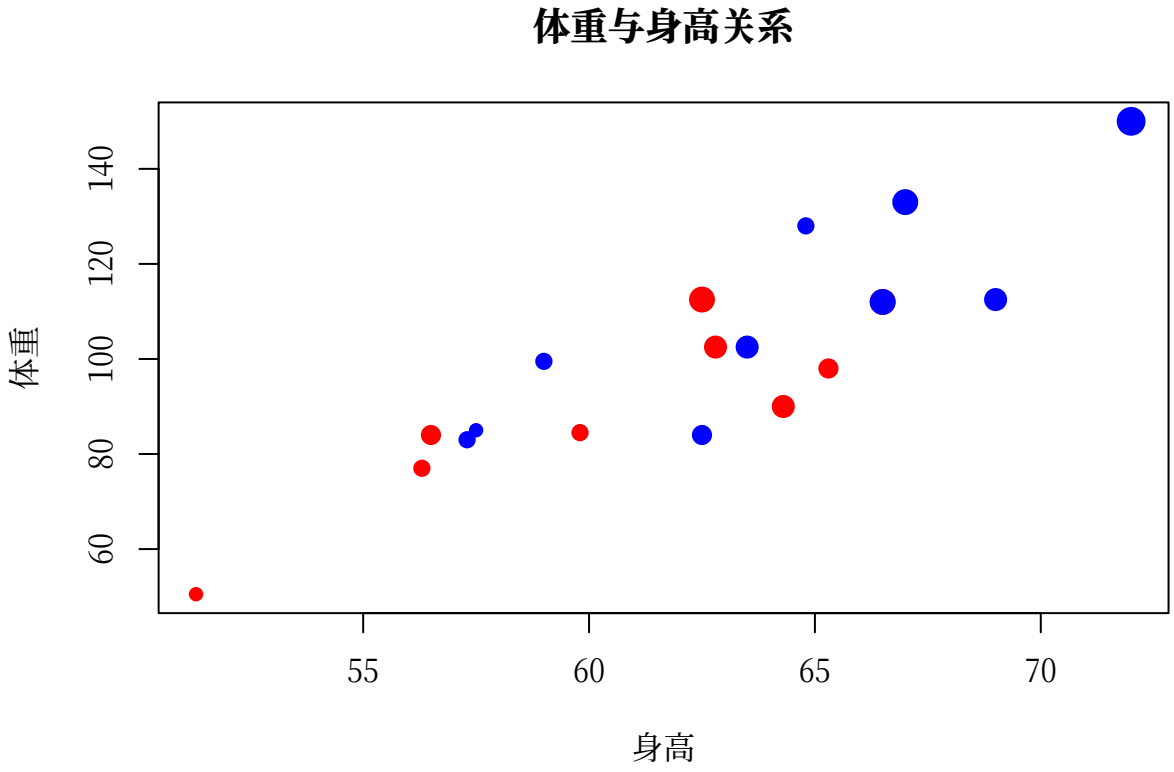
用气泡大小表现第三维（年龄）：

```
with(d.class,
     plot(height, weight,
          pch=16, col='blue',
          cex=1 + (age - min(age))/(max(age)-min(age))))
```



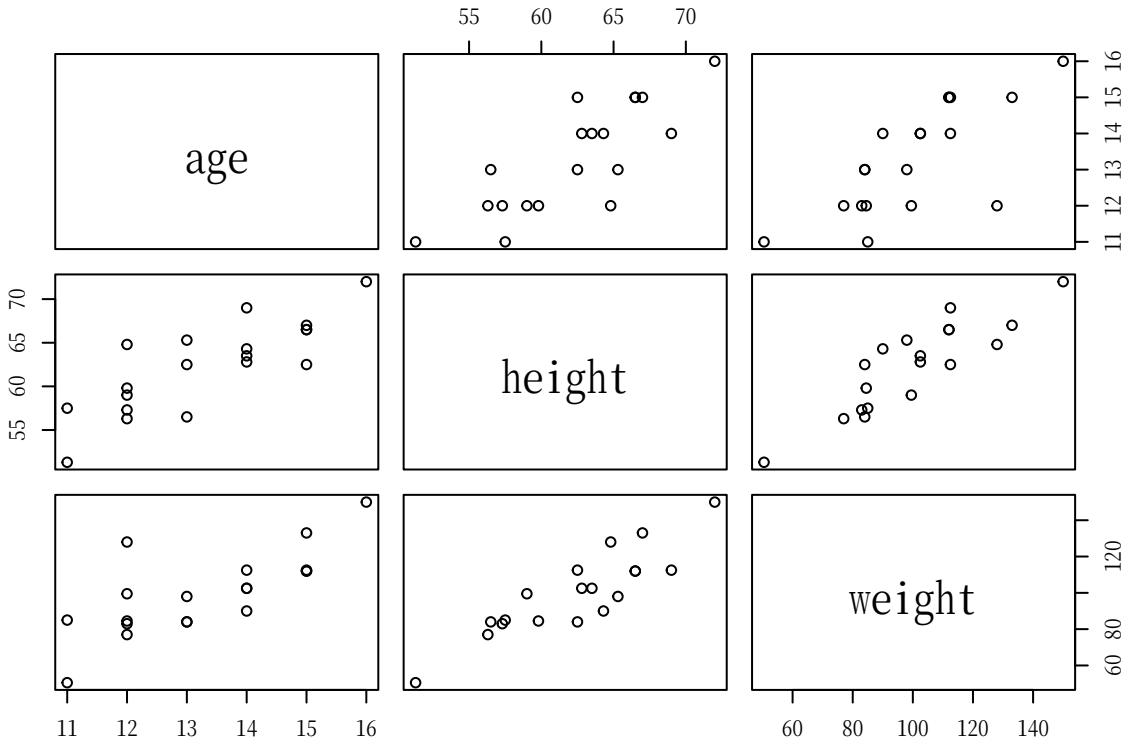
用气泡大小表现年龄，用颜色区分性别：

```
with(d.class,
        plot(height, weight,
          main='体重与身高关系',
          xlab='身高', ylab='体重',
          pch=16,
          col=ifelse(sex=='M', 'blue', 'red'),
          cex=1 + (age - min(age))
            /(max(age)-min(age))))
```



用 pairs() 函数可以做散点图矩阵：

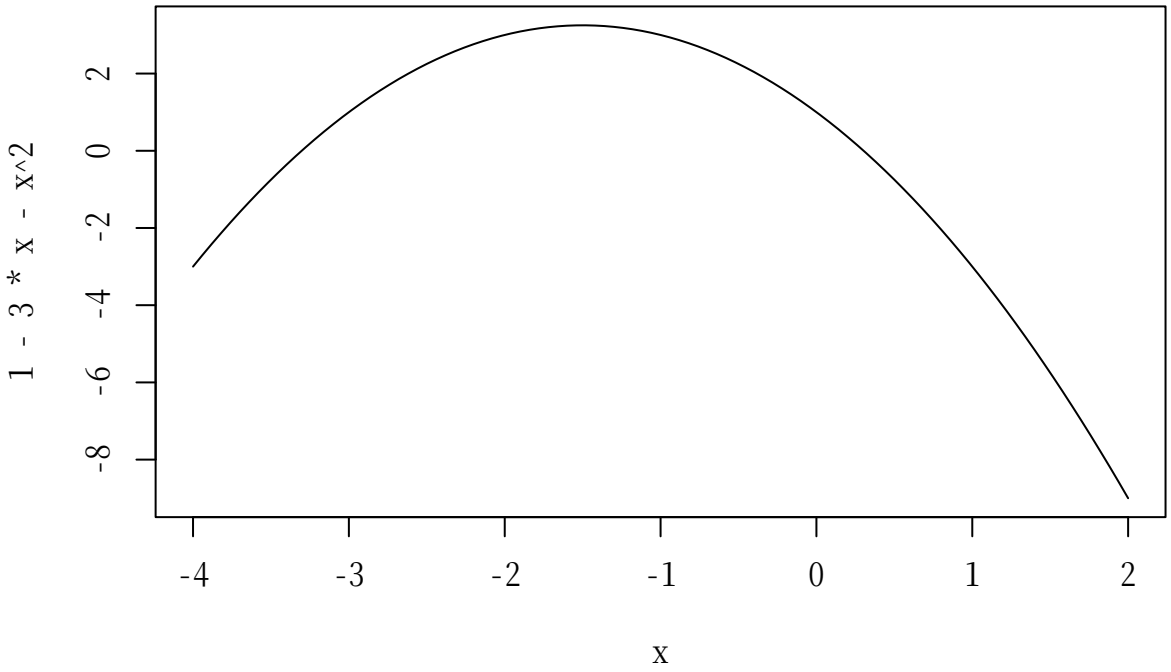
```
pairs(d.class[, c('age', 'height', 'weight')])
```



26.1.6 曲线图

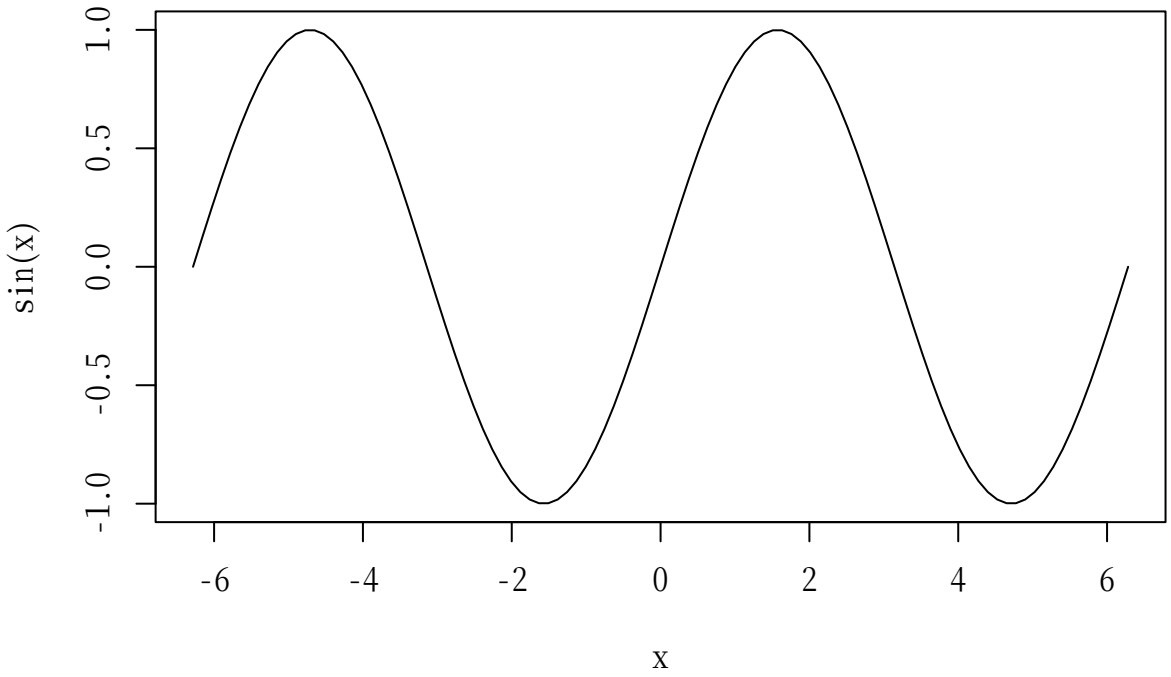
`curve()` 函数接受一个函数，或者一个以 `x` 为变量的表达式，以及曲线的自变量的左、右端点，绘制函数或者表达式的曲线图，如：

```
curve(1 - 3*x - x^2, -4, 2)
```



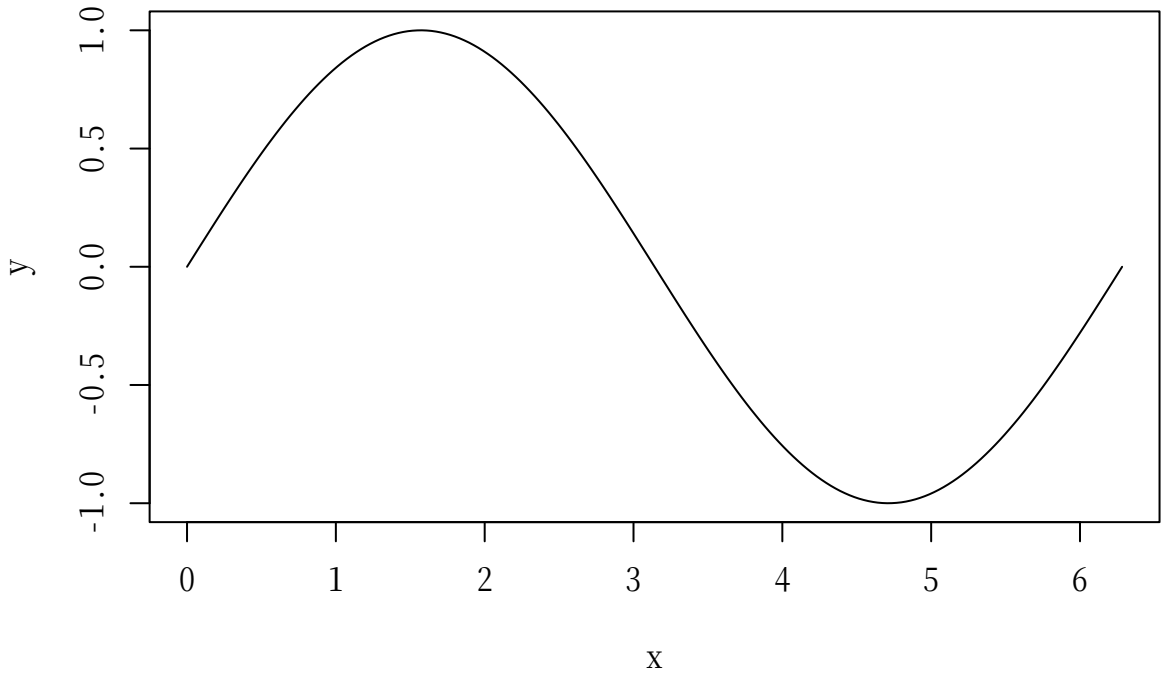
又如：

```
curve(sin, -2*pi, 2*pi)
```



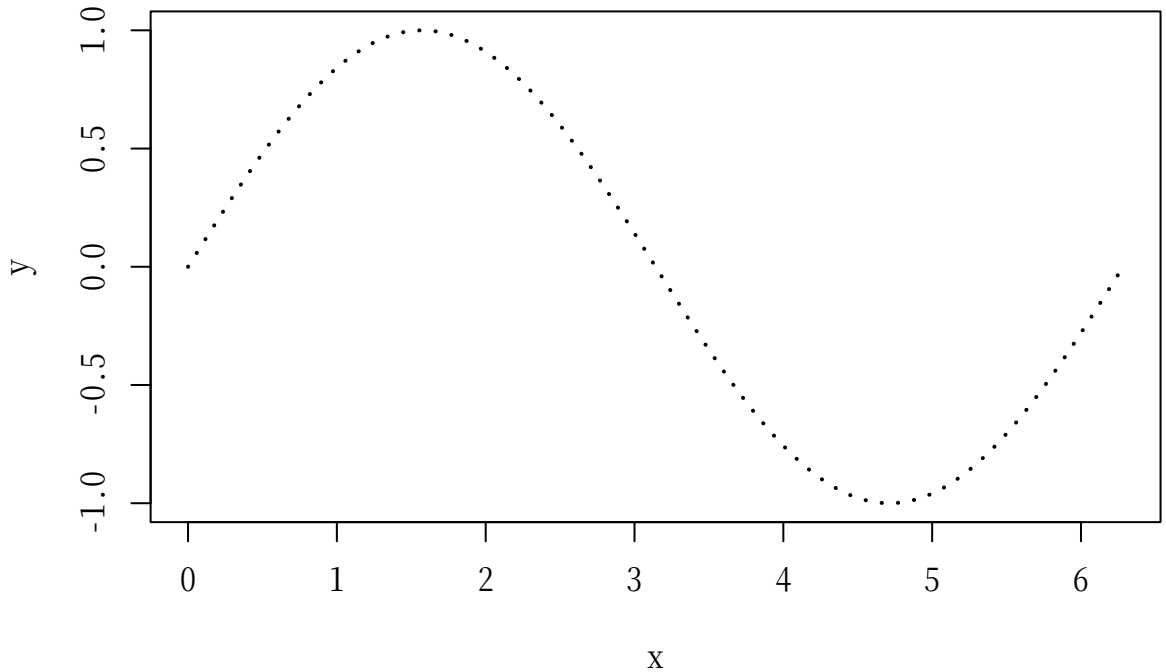
在 plot 函数中使用 type=’l’参数可以作曲线图，如

```
x <- seq(0, 2*pi, length=200)
y <- sin(x)
plot(x,y, type='l')
```



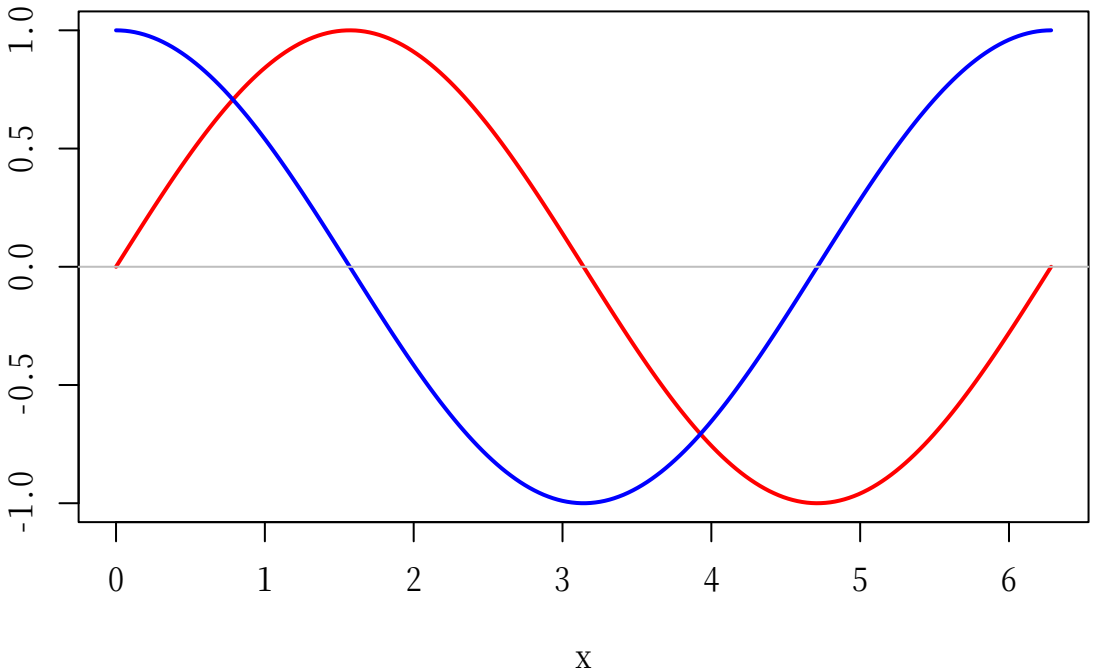
除了仍可以用 `main`, `xlab`, `ylab`, `col` 等参数外，还可以用 `lwd` 指定线宽度，`lty` 指定虚线，如

```
plot(x,y, type='l', lwd=2, lty=3)
```



多条曲线，可以用 `matplot()` 函数。例如

```
x <- seq(0, 2*pi, length=200)
y1 <- sin(x)
y2 <- cos(x)
matplot(x, cbind(y1, y2), type='l',
        lty=1, lwd=2, col=c("red", "blue"),
        xlab="x", ylab="")
abline(h=0, col='gray')
```



26.1.7 三维图

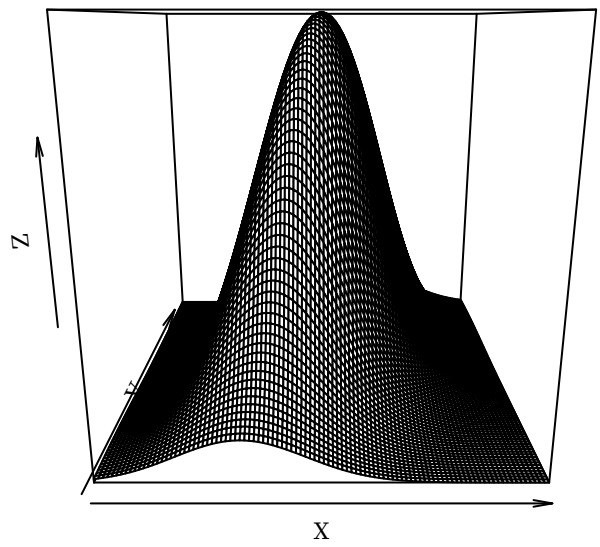
用 persp 函数作三维曲面图, contour 作等值线图，image 作色块图。坐标 x 和 y 构成一张平面网格，数据 z 是包含 z 坐标的矩阵，每行对应一个横坐标，每列对应一个纵坐标。

下面的程序生成二元正态分布密度曲面数据：

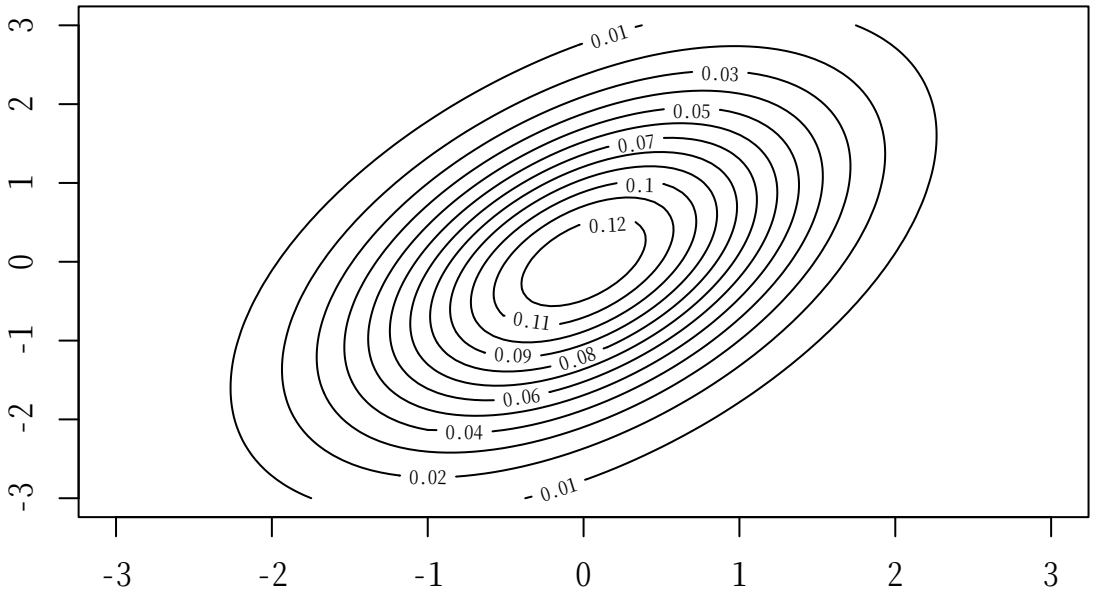
```
x <- seq(-3,3, length=100)
y <- x
f <- function(x,y,ssq1=1, ssq2=2, rho=0.5){
     det1 <- ssq1*ssq2*(1 - rho^2)
     s1 <- sqrt(ssq1)
     s2 <- sqrt(ssq2)
     1/(2*pi*sqrt(det1)) * exp(-0.5 / det1 * (
       ssq2*x^2 + ssq1*y^2 - 2*rho*s1*s2*x*y))
}
z <- outer(x, y, f)
```

作二元正态密度函数的三维曲面图、等高线图、色块图:

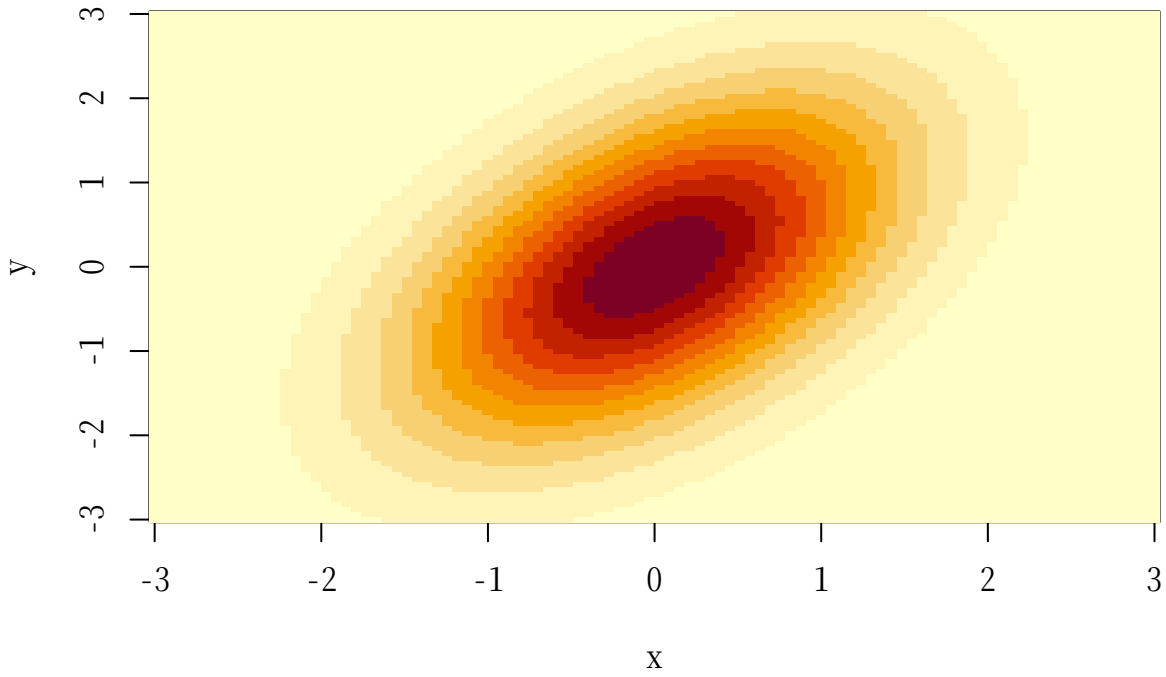
```
persp(x, y, z)
```




```
contour(x, y, z)
```




```
image(x, y, z)
```



26.1.8 动态三维图

rgl 包能制作动态的三维散点图与曲面图。

```
library(rgl)
```

iris 数据框包含了 3 种鸢尾花的各 50 个样品的测量值，测量值包括花萼长、宽，花瓣长、宽。用 rgl 的 `plot3d()` 作动态三维散点图如下：

```
with(iris, plot3d(
  Sepal.Length, Sepal.Width, Petal.Length,
  type="s", col=as.numeric(Species)))
```

这个图可以用鼠标拖动旋转。其中 `type="s"` 表示绘点符号是球体形状。还可选`"p"`(点)、`"l"`(连线)、`"h"`(向 z=0 连线)。可以用 `size=` 指定大小倍数（缺省值为 3）。

用 rgl 的 `persp3d()` 函数作曲面图。如二元正态分布密度曲面：

```
x <- seq(-3,3, length=100)
y <- x
f <- function(x,y,ssq1=1, ssq2=2, rho=0.5){
     det1 <- ssq1*ssq2*(1 - rho^2)
     s1 <- sqrt(ssq1)
     s2 <- sqrt(ssq2)
     1/(2*pi*sqrt(det1)) * exp(-0.5 / det1 * (
       ssq2*x^2 + ssq1*y^2 - 2*rho*s1*s2*x*y))
}
z <- outer(x, y, f)
persp3d(x=x, y=y, z=z, col='red')
```

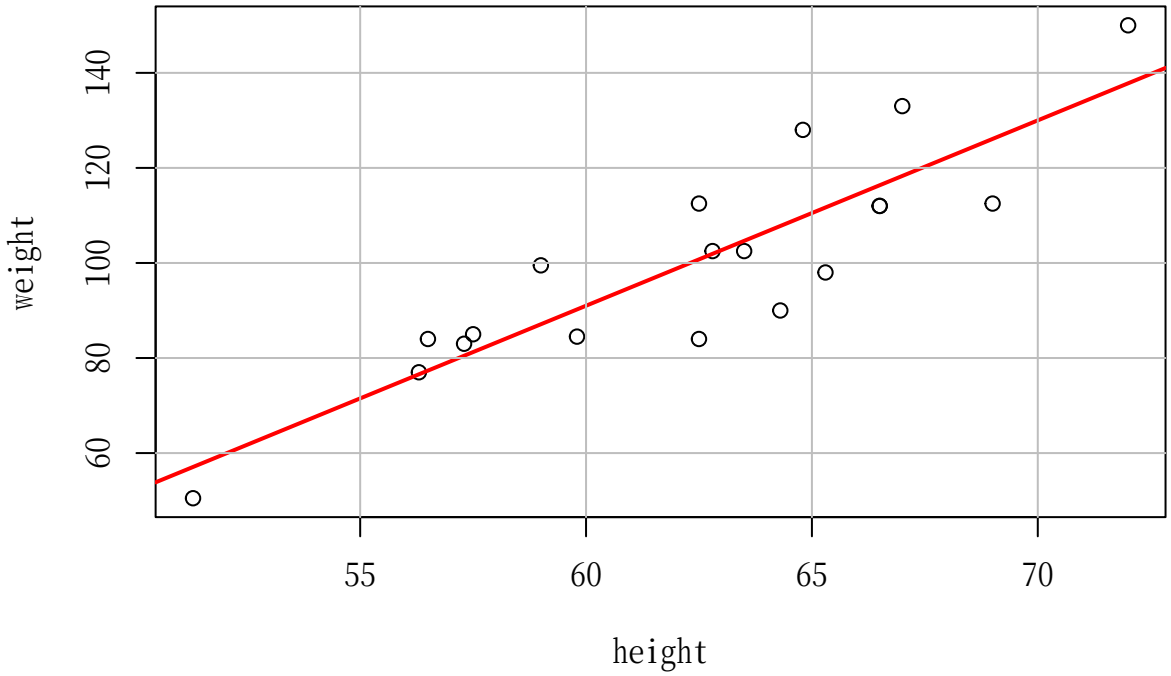
rgl 也有低级图形函数支持向已有图形添加物体、文字等，也支持并列多图。适当设置可以在 R Markdown 生成的 HTML 结果中动态显示三维图。

26.2 低级图形函数

26.2.1 abline()

用 abline 函数在图中增加直线。可以指定截距和斜率，或为竖线指定横坐标 (用参数 v)，为水平线指定纵坐标 (用参数 h)。如

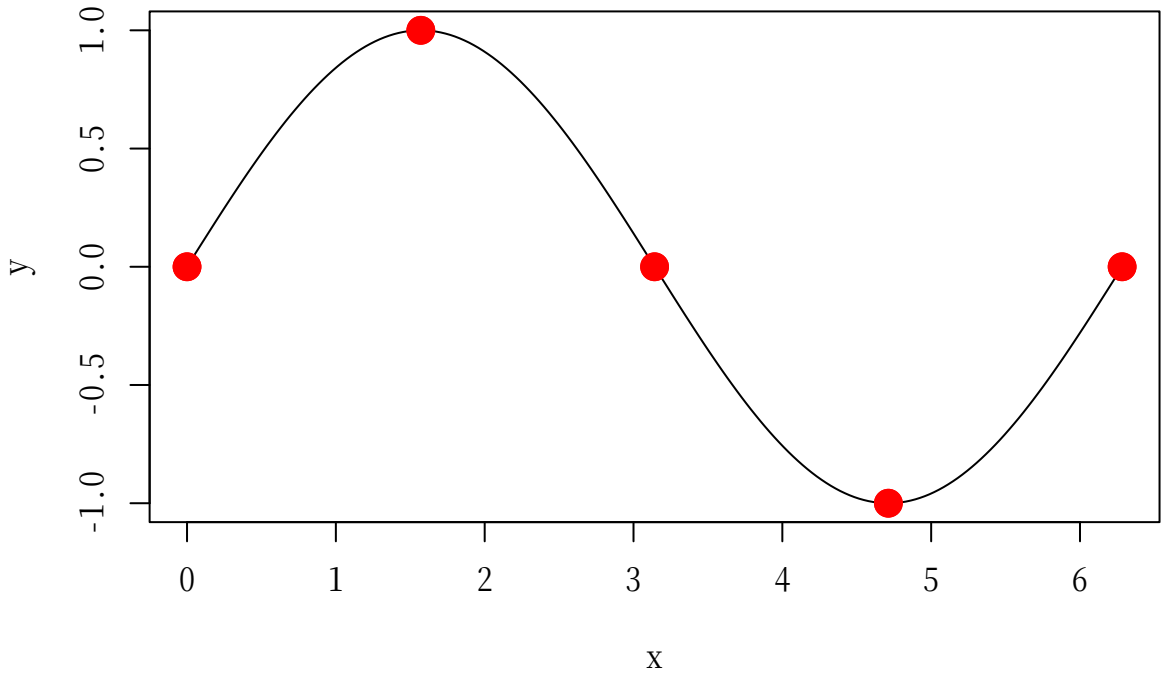
```
with(d.class, plot(height, weight))
abline(-143, 3.9, col="red", lwd=2)
abline(v=c(55,60,65,70), col="gray")
abline(h=c(60,80,100,120,140), col="gray")
```



26.2.2 points()

用 points 函数增加散点，如：

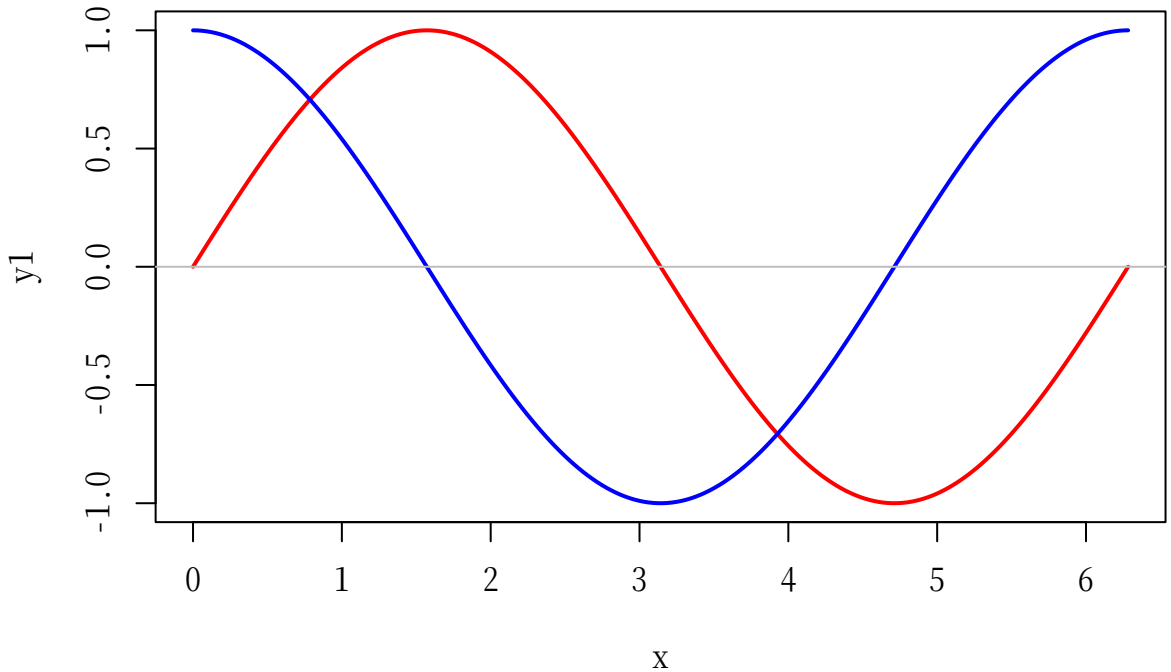
```
x <- seq(0, 2*pi, length=200)
y <- sin(x)
special <- list(x=(0:4)*pi/2, y=sin((0:4)*pi/2))
plot(x, y, type='l')
points(special$x, special$y,
       col="red", pch=16, cex=2)
points(special, col="red", pch=16, cex=2)
```



26.2.3 lines()

用 lines 函数增加曲线，如：

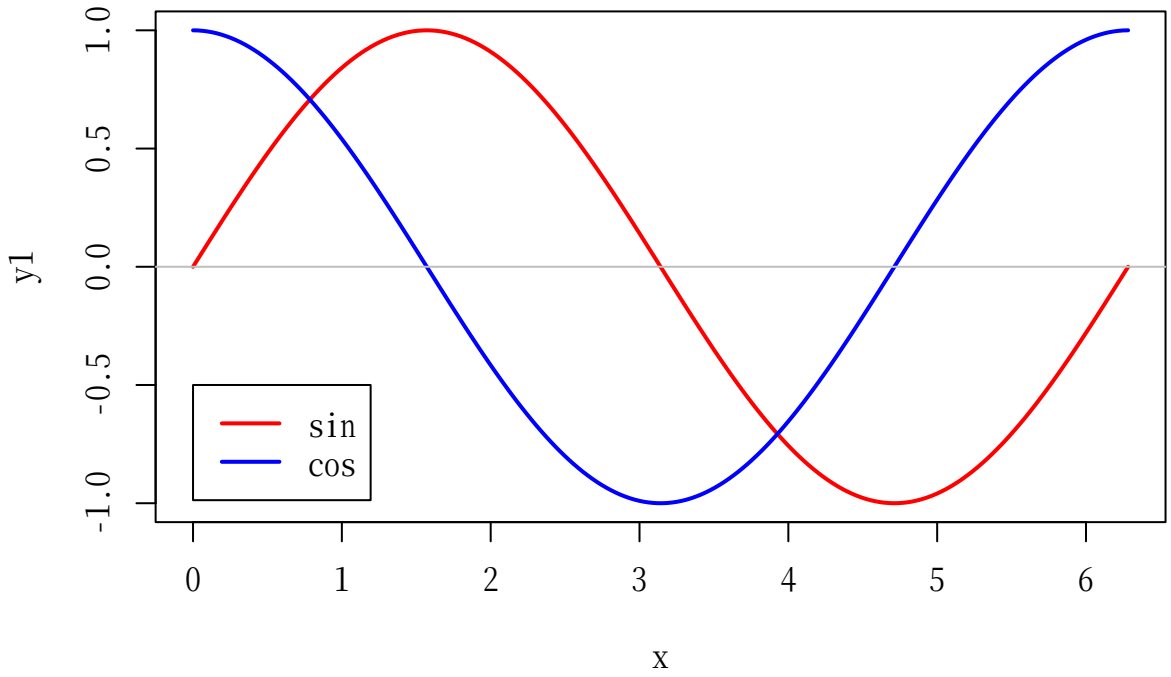
```
x <- seq(0, 2*pi, length=200)
y1 <- sin(x)
y2 <- cos(x)
plot(x, y1, type='l', lwd=2, col="red")
lines(x, y2, lwd=2, col="blue")
abline(h=0, col='gray')
```



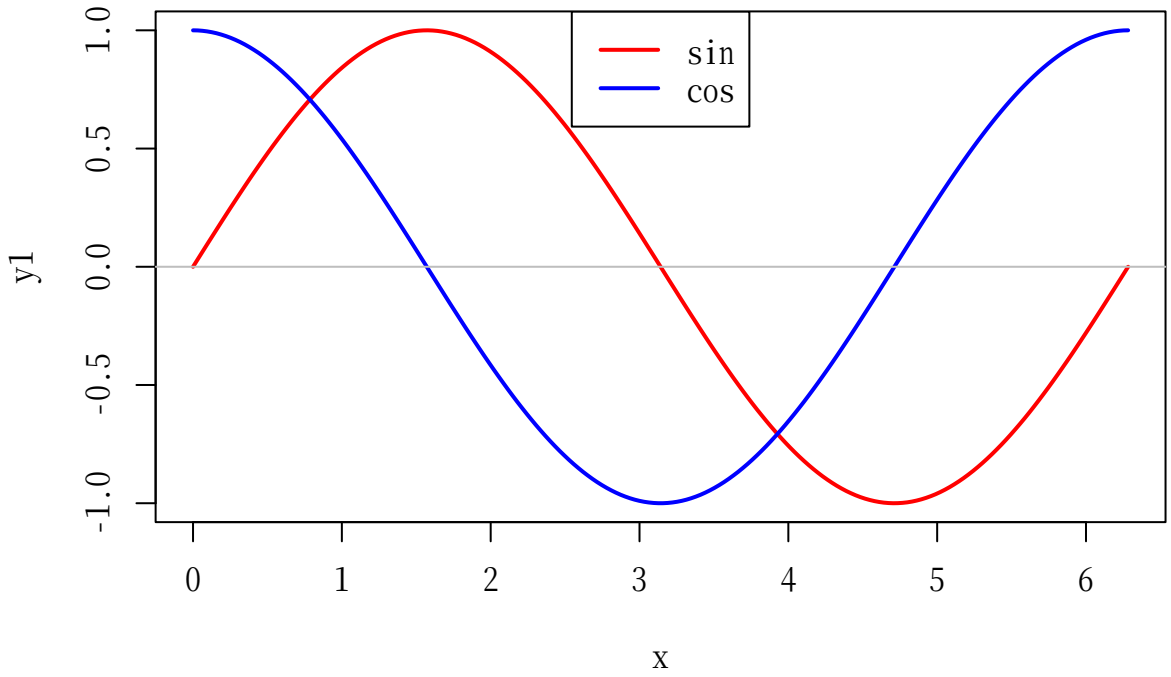
26.2.4 图例

可以用 `legend` 函数增加标注，如

```
x <- seq(0, 2*pi, length=200)
y1 <- sin(x)
y2 <- cos(x)
plot(x, y1, type='l', lwd=2, col="red")
lines(x, y2, lwd=2, col="blue")
abline(h=0, col='gray')
legend(0, -0.5, col=c("red", "blue"),
       lty=c(1,1), lwd=c(2,2),
       legend=c("sin", "cos"))
```



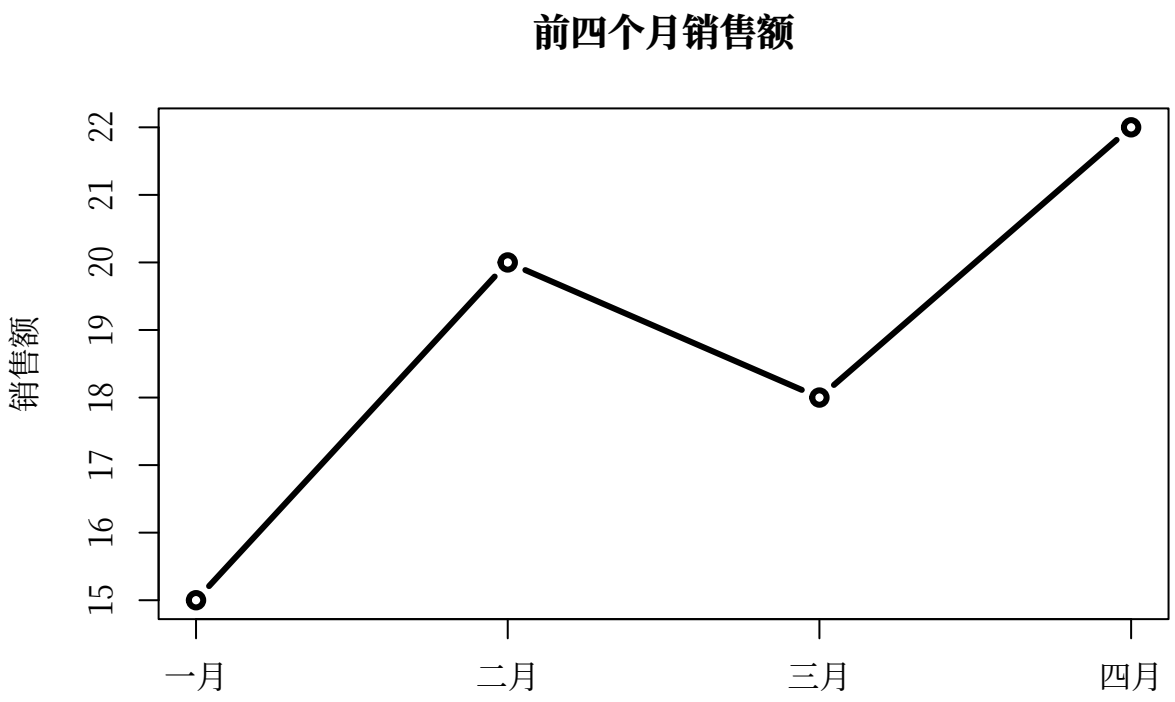

```
x <- seq(0, 2*pi, length=200)
y1 <- sin(x)
y2 <- cos(x)
plot(x, y1, type='l', lwd=2, col="red")
lines(x, y2, lwd=2, col="blue")
abline(h=0, col='gray')
legend('top', col=c("red", "blue"),
       lty=c(1,1), lwd=c(2,2),
       legend=c("sin", "cos"))
```



26.2.5 axis()

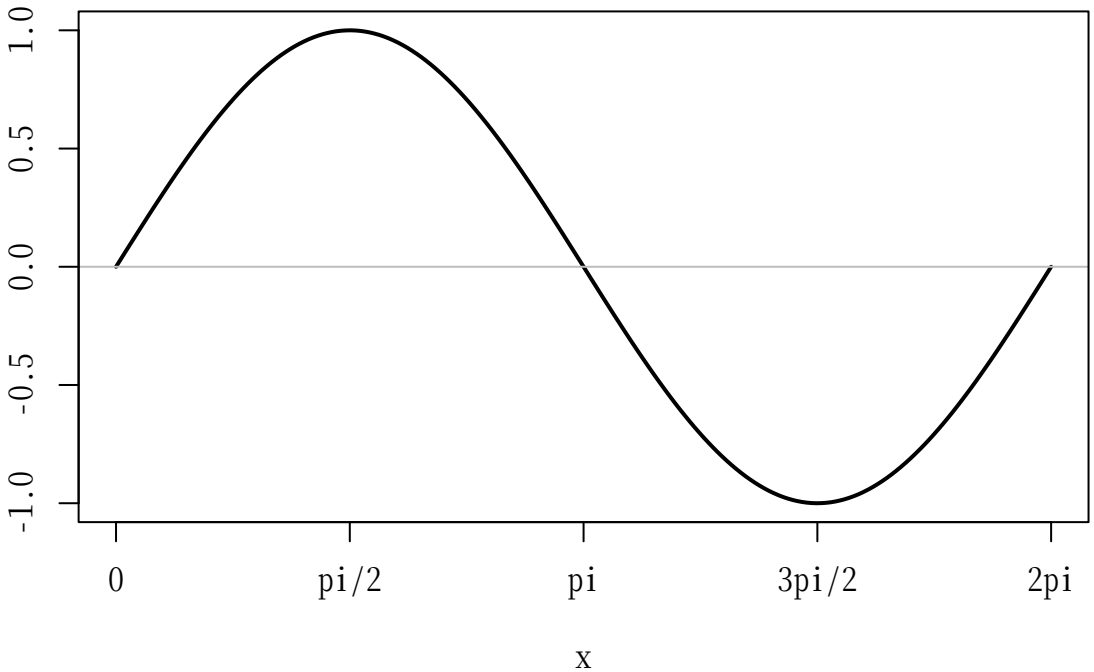
在 plot() 函数中用 axes=FALSE 可以取消自动的坐标轴。用 box() 函数画坐标边框。用 axis 函数单独绘制坐标轴。axis 的第一个参数取 1，2，3，4，分别表示横轴、纵轴、上方和右方。axis 的参数 at 为刻度线位置，labels 为标签。如

```
x <- c('一月'=15, '二月'=20,
       '三月'=18, '四月'=22)
plot(seq(along=x), x, axes=FALSE,
     type='b', lwd=3,
     main='前四个月销售额',
     xlab='', ylab='销售额')
box(); axis(2)
axis(1, at=seq(along=x), labels=names(x))
```



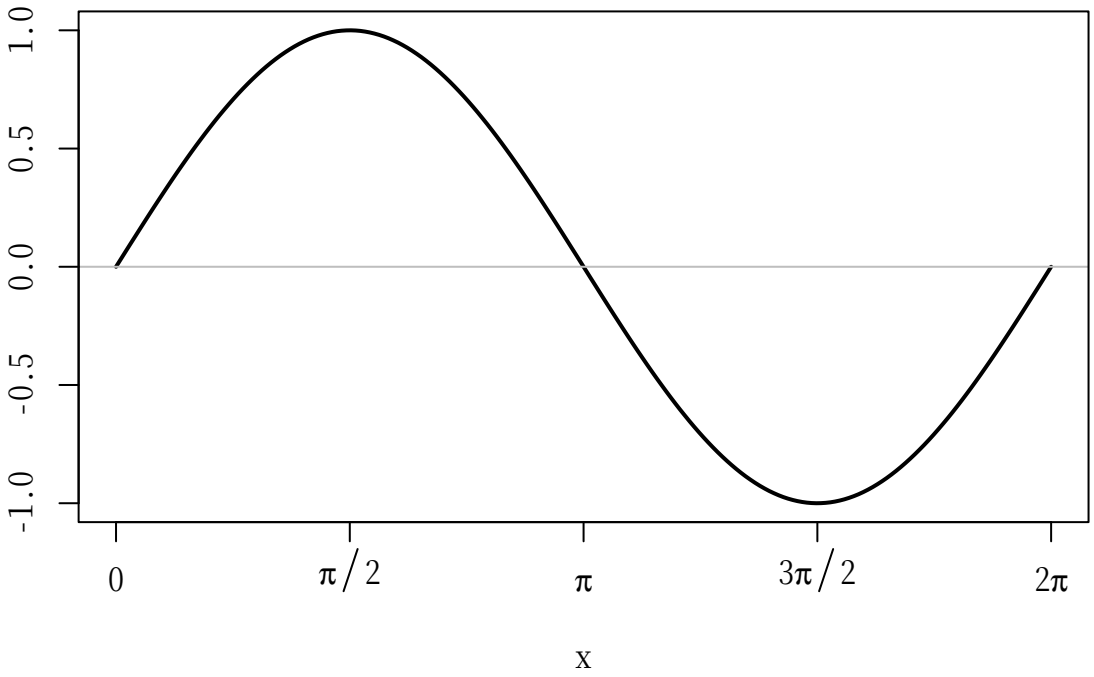
R 基本绘图支持少量的数学公式显示功能，如不用数学符号时：

```
x <- seq(0, 2*pi, length=200)
y1 <- sin(x)
plot(x, y1, type='l', lwd=2,
     axes=FALSE,
     xlab='x', ylab='')
abline(h=0, col='gray')
box()
axis(2)
axis(1, at=(0:4)/2*pi,
     labels=c('0', 'pi/2', 'pi', '3pi/2', '2pi'))
```



使用数学符号时：

```
x <- seq(0, 2*pi, length=200)
y1 <- sin(x)
plot(x, y1, type='l', lwd=2,
     axes=FALSE,
     xlab='x', ylab='')
abline(h=0, col='gray')
box()
axis(2)
axis(1, at=(0:4)/2*pi,
     labels=c(0, expression(pi/2),
     expression(pi), expression(3*pi/2),
     expression(2*pi)))
```



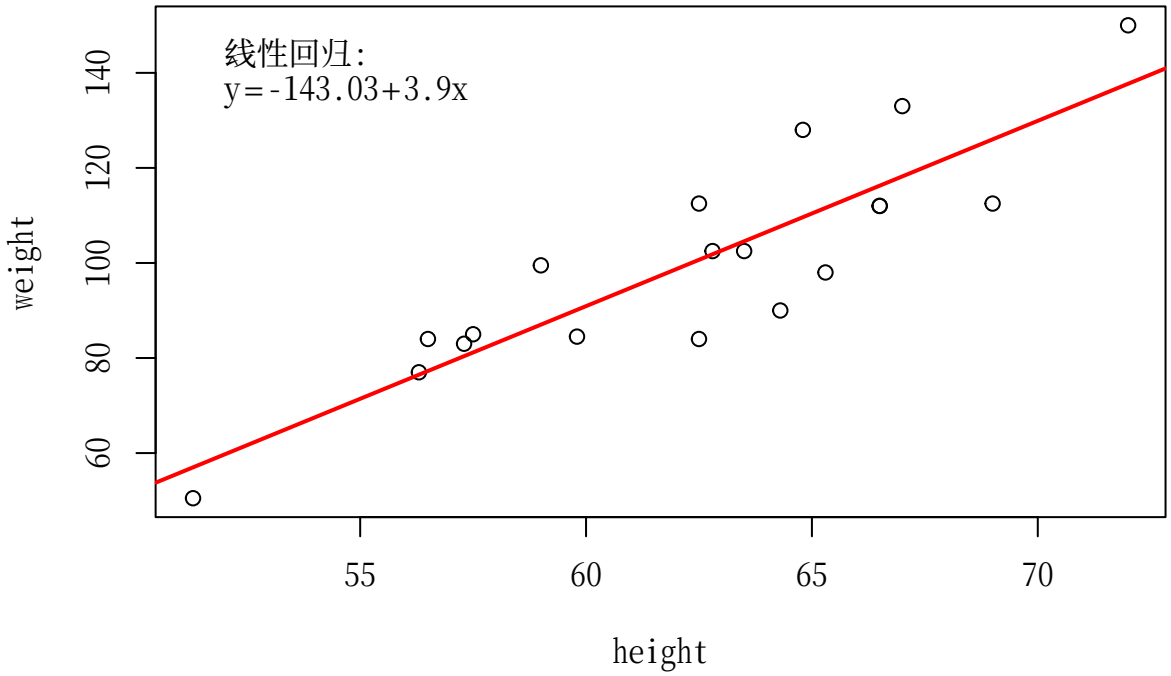
绘图中使用数学符号的演示：

```
demo(plotmath)
```

26.2.6 text()

text() 在坐标区域内添加文字。mtext() 在边空处添加文字。如

```
with(d.class, plot(height, weight))
lm1 <- lm(weight ~ height, data=d.class)
abline(lm1, col='red', lwd=2)
a <- coef(lm1)[1]
b <- coef(lm1)[2]
text(52, 145, adj=0, '线性回归:')
text(52, 140, adj=0,
    substitute(hat(y) == a + b*x,
    list(a=round(coef(lm1)[1], 2),
         b=round(coef(lm1)[2], 2))))
```



26.2.7 locator() 和 identify()

locator() 函数在执行时等待用户在图形的坐标区域内点击并返回点击处的坐标。可以用参数 n 指定要点击的点的个数。不指定个数则需要用右键菜单退出。这个函数也可以用来要求用户点击以进行到下一图形。如

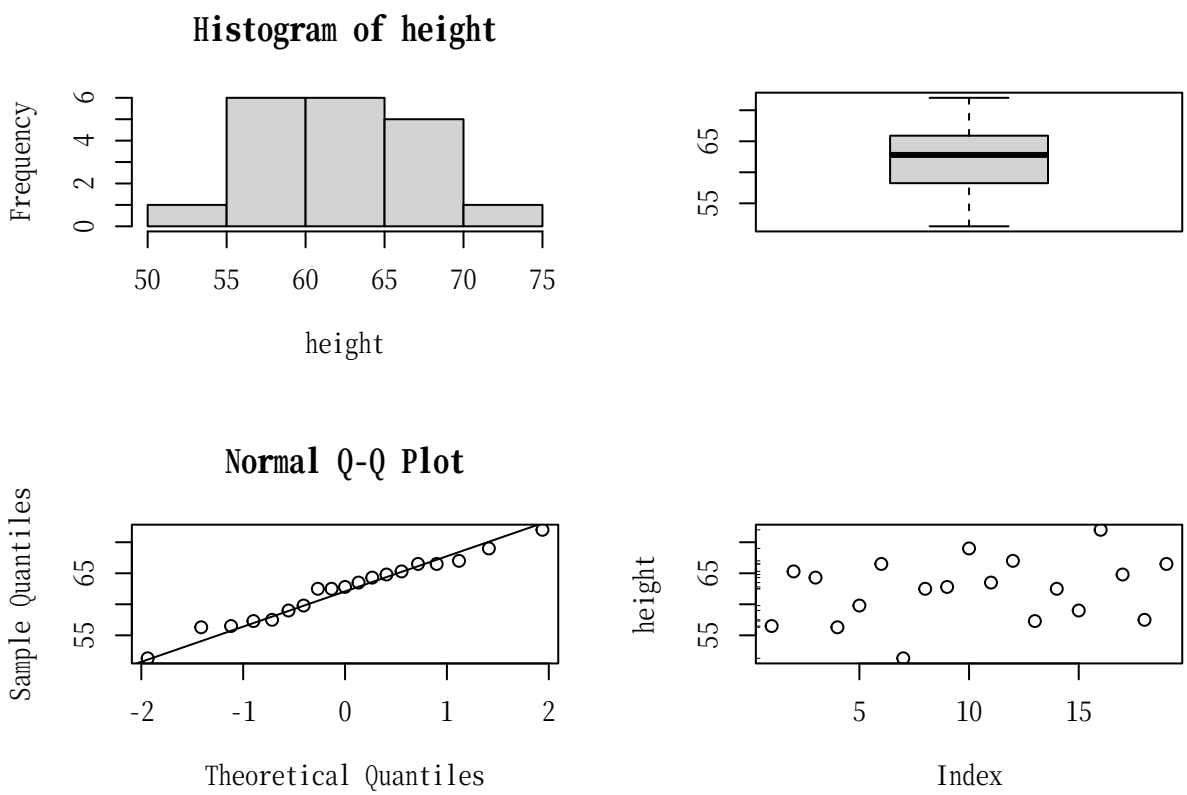
```
x <- seq(0, 2*pi, length=200)
y1 <- sin(x); y2 <- cos(x)
plot(x, y1, type='l',
     col="red")
lines(x, y2, col="blue")
legend(locator(1), col=c("red", "blue"),
       lty=c(1,1), legend=c("sin", "cos"))
```

identify() 可以识别点击处的点并标注标签，格式为 identify(x, y, labels)，其中 (x,y) 给出可点击的点的坐标，labels 是每个点对应的标签，点击那个点就在那个点旁边标对应的标签。

26.3 图形参数

用图形参数可以选择点的形状、颜色、线型、粗细、坐标轴做法、边空、一页多图等。有些参数直接用在绘图函数内，如 plot 函数可以用 pch、col、cex、lty、lwd 等参数。有些图形参数必须使用 par() 函数指定。par 函数指定图形参数并返回原来的参数值，所以在修改参数值作图后通常应该恢复原始参数值，做法如

```
opar <- par(mfrow=c(2,2))
with(d.class, {hist(height);
     boxplot(height);
     qqnorm(height); qqline(height);
     plot(height); rug(height,side=2)})
```

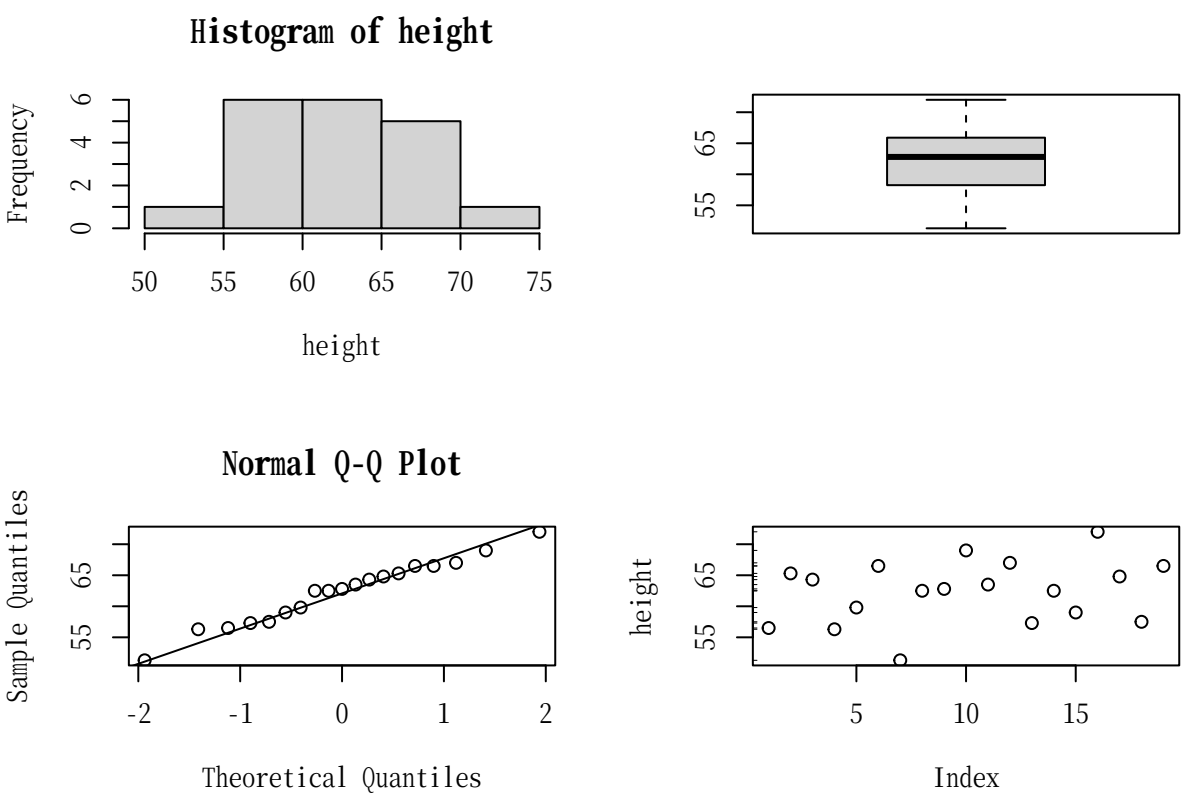



```
par(opar)
```

在函数内，可以在函数开头修改了图形参数后，用 on.exit() 函数将恢复原始图形参数作为函数退出前必须完成的任务，如

```
f <- function(){
    opar <- par(mfrow=c(2,2)); on.exit(par(opar))
```

```
    with(
        d.class,
        {hist(height);
         boxplot(height);
         qqnorm(height); qqline(height);
         plot(height); rug(height,side=2)
     })
}
f()
```

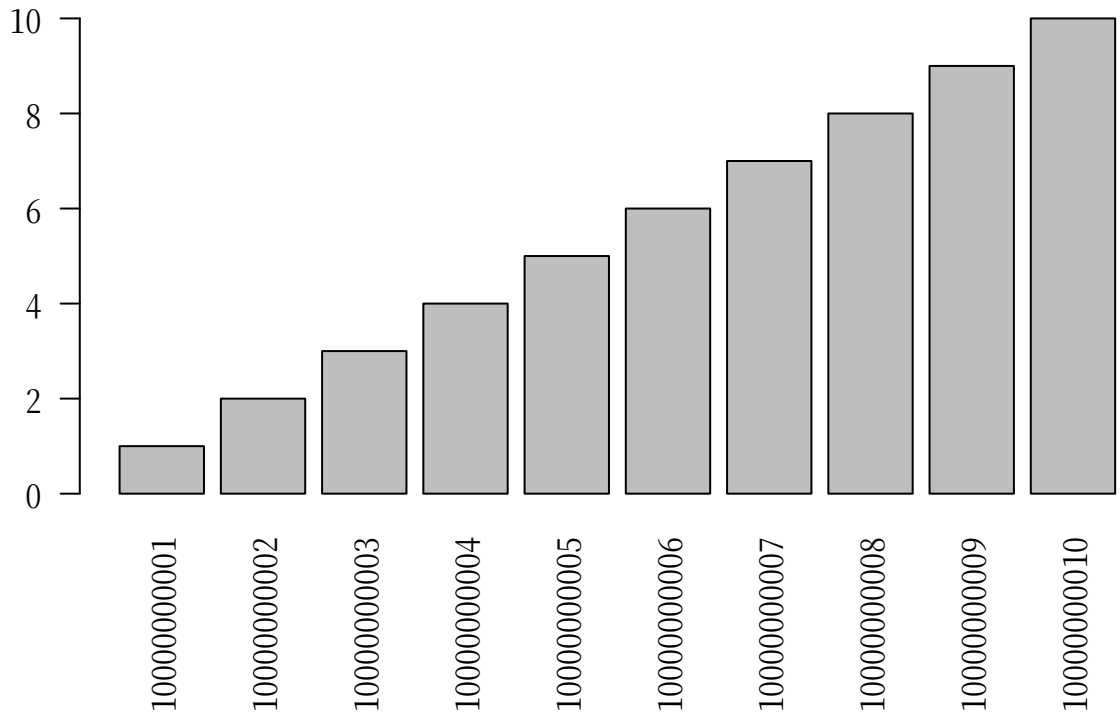


26.3.1 例子：用图形参数解决 barplot 图形横坐标值过宽

barplot 的横坐标标注太宽时，自动将某些标注省略。用 las=2 指定坐标轴刻度标签垂直于坐标轴，这样 x 轴的刻度值就变成了纵向的。注意使用 mar 参数增加横坐标边空大小。例如

```
f <- function(){
    opar <- par(mar=c(8, 4, 2, 0.5)); on.exit(par(opar))
    x <- 1:10
    names(x) <- paste(10000000000 + (1:10))
```

```
    barplot(x, las=2)
}
f()
```



图形参数可以分为如下四类

- 图形元素控制；
- 坐标轴与坐标刻度；
- 图形边空；
- 一页多图。

26.3.2 图形元素控制

- `pch=16` 参数。散点符号，取 0 ∼ 18 的数。
- `lty=2` 参数。线型，1 为实线，从 2 开始为各种虚线。
- `lwd=2` 参数，线的粗细，标准粗细为 1。
- `col="red"` 参数，颜色，可以是数字 1 ∼ 8，或颜色名字符串如`"red"`，`"blue"` 等。用 `colors()` 函数查询有名字的颜色。用 `rainbow(n)` 函数产生连续变化的颜色。

- `font=2` 参数，字体，一般 font=1 是正体,2 是粗体, 3 是斜体,4 是粗斜体。
- `adj=-0.1` 指定文本相对于给定坐标的对齐方式。取 0 表示左对齐, 取 1 表示右对齐, 取 0.5 表示居中。此参数的值实际代表的是出现在给定坐标左边的文本的比例。
- `cex=1.5` 绘点符号大小倍数，基本值为 1。

26.3.3 坐标轴与坐标刻度

- `mgp=c(3,1,0)` 坐标轴的标签、刻度值、坐标轴线到实际的坐标轴位置的距离，以行高为单位。经常用来缩小坐标轴所占的空间, 如 `mgp=c(1.5, 0.5, 0)`。
- `lab=c(5,7,12)` 提供刻度线多少的建议，第一个数为 x 轴刻度线个数，第二个数为 y 轴刻度线个数，第三个数是坐标刻度标签的字符宽度。
- `las=1` 坐标刻度标签的方向。0 表示总是平行于坐标轴, 1 表示总是水平, 2 表示总是垂直于坐标轴。
- `tck=0.01` 坐标轴刻度线长度，以绘图区域大小为单位 1。
- `xaxs="s"`, `yaxs="e"`: 控制 x 轴和 y 轴标刻度的方法。

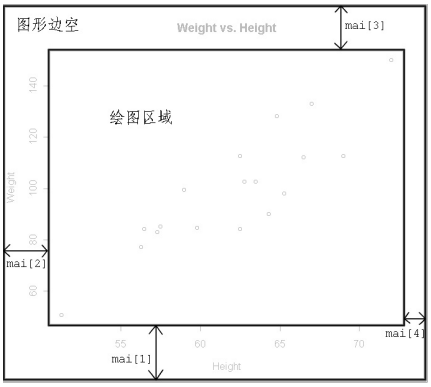
  取`"s"`(即 standard) 或`"e"`(即 extended) 的时候数据范围控制在最小刻度和最大刻度之间。取`"e"` 时如果有数据点十分靠近边缘轴的范围会略微扩大。

  取值为`"i"`（即 internal）或`"r"`（此为缺省）使得刻度线都落在数据范围内部, 而`"r"` 方式所留的边空较小。

  取值设为`"d"` 时会锁定此坐标轴, 后续的图形都使用与它完全相同的坐标轴, 这在要生成一系列可比较的图形的时候是有用的。要解除锁定需要把这个图形参数设为其它值。

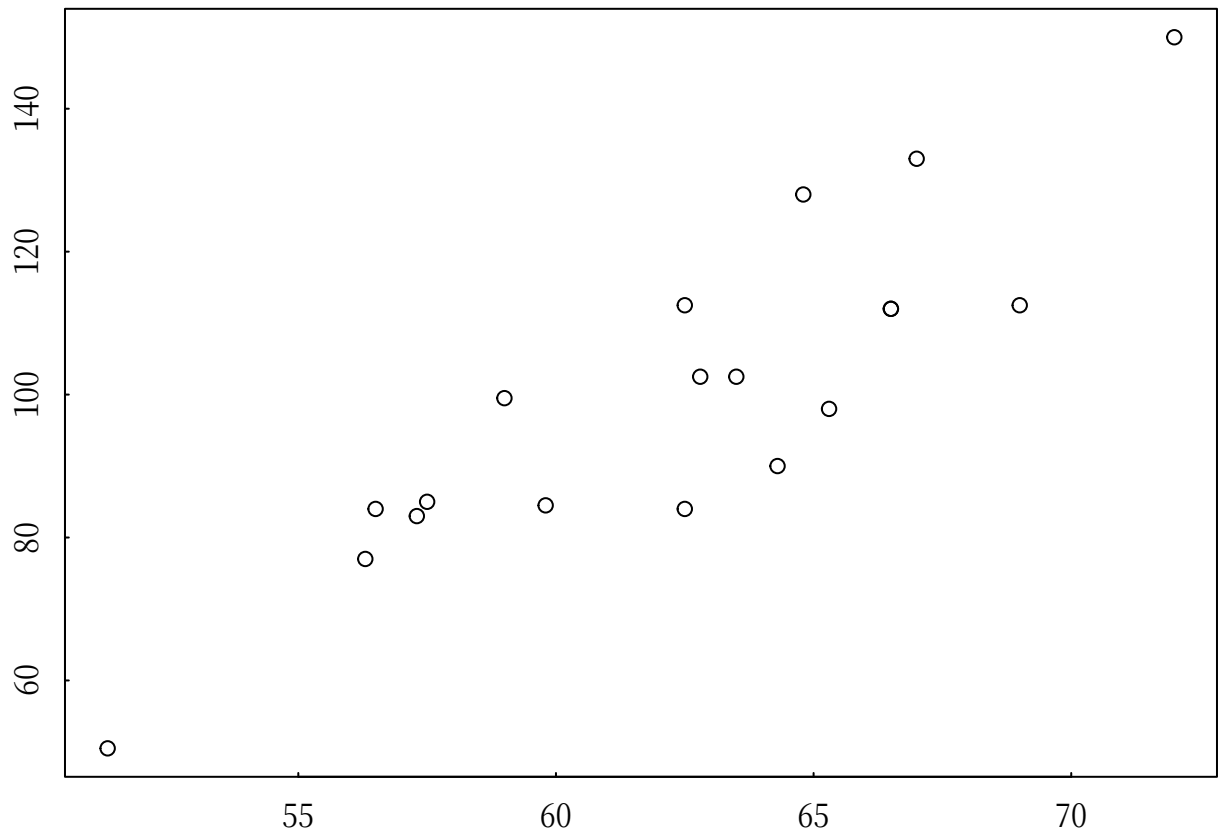
26.3.4 图形边空

一个单独的图由绘图区域 (绘图的点、线等画在这个区域中) 和包围绘图区域的边空组成, 边空中可以包含坐标轴标签、坐标轴刻度标签、标题、小标题等, 绘图区域一般被坐标轴包围。



边空的大小由 mai 参数或 mar 参数控制, 它们都是四个元素的向量, 分别规定下方、左方、上方、右方的边空大小, 其中 mai 取值的单位是英寸, 而 mar 的取值单位是文本行高度。例如：

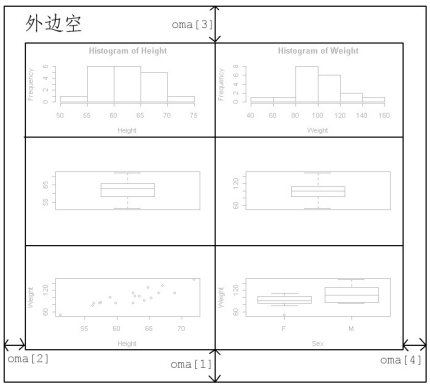
```
opar <- par(mar=c(2,2,0.5,0.5),
            mgp=c(0.5, 0.5, 0), tck=0.005)
with(d.class, plot(height, weight,
                   xlab='', ylab=''))
```




```
par(opar)
```

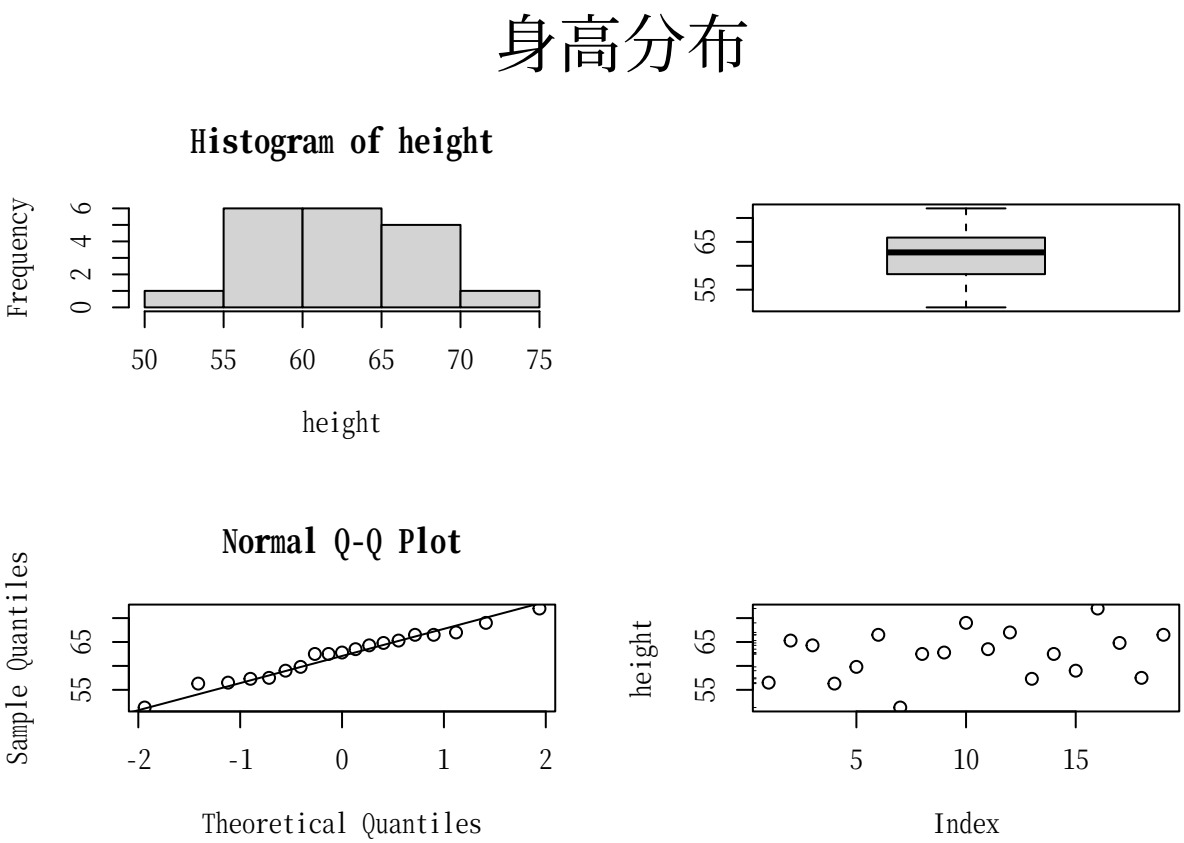
26.3.5 一页多图

R 可以在同一页面开若干个按行、列排列的窗格, 在每个窗格中可以作一幅图。每个图有自己的内边空, 而所有图的外面可以包一个 “外边空”。



一页多图用 `mfrow` 参数或 `mfcol` 参数规定。用 `oma` 指定四个外边空的行数。用 `mtext` 加 `outer=T` 指定在外边空添加文本。如果没有 `outer=T` 则在内边空添加文本。如

```
opar <- par(mfrow=c(2,2),
            oma=c(0,0,2,0))
with(d.class, {hist(height);
     boxplot(height);
     qqnorm(height); qqline(height);
     plot(height); rug(height,side=2)})
mtext(side=3, text='身高分布', cex=2, outer=T)
```



```
par(opar)
```

26.4 图形输出

只要启用了高级绘图函数会自动选用当前绘图设备，缺省为屏幕窗口。

26.4.1 PDF 输出

用 pdf 函数可以指定输出到 PDF 文件。如

```
pdf(file='fig-hw.pdf', height=10/2.54,
    width=10/2.54, family='GB1')
with(d.class, plot(height, weight,
                   main='体重与身高关系'))
dev.off()
```

用 dev.off() 关闭当前设备并生成输出文件（如果是屏幕窗口则没有保存结果）。

26.4.2 PNG 输出

```
png(file='fig-hw.png', height=1000, width=1000)
with(d.class, plot(height, weight,
                   main='体重与身高关系'))
dev.off()
```

类似地，用 jpeg() 函数启用 JPEG 图形设备, 用 bmp() 函数启用 BMP 图形设备, 用 postscript() 函数启用 PostScript 图形设备。

26.5 包含多种中文字体的图形

为了使用 MS Windows 系统字体，一种办法是安装 showtext 包。该包替换画图时的添加文本函数命令，把文本内容替换成多边形（PDF 或 PS 图）或点阵（点阵图）。

需要的工作：

- 查看 Windows 的 font 目录内容，看文件名与字体名的对应关系。下面的程序中的列表是我的中文 Windows 10 的部分中文字体。
- 找到自己希望使用的中文字体的文件名。

- 用 font.add() 命令，增加一套自定义字体 family，一套中可以指定四种：常规 (regular), 粗体 (bold), 斜体 (italic), 粗斜体 (bolditalic)
- 程序中调入 showtext 包并运行 showtext.auto() 命令，这个命令使得文本命令采用 showtext 包
- 用 par(family=) 指定自定义的字体 family。
- 作图（主要是 PDF）。关闭图形设备。

```
test.chinese <- function(){
    require(showtext); showtext.auto()

    ## 建立文件名到字体名对照表
    fmap <- c(
        'msyh'='微软雅黑常规',
        'msyhbd'='微软雅黑粗体',
        'msyhl'='微软雅黑细体',
        'simsun'='宋体',
        'simfang'='仿宋',
        'simkai'='楷体',
        'simhei'='黑体',
        'SIMLI'='隶书',
        'SIMYOU'='幼圆',
        'STSONG'='华文宋体',
        'STZHONGS'='华文中宋',
        'STFANGSO'='华文仿宋',
        'STKAITI'='华文楷体',
        'STXIHEI'='华文细黑',
        'STLITI'='华文隶书',
        'STXINGKA'='华文行楷',
        'STXINWEI'='华文新魏',
        'STCAIYUN'='华文彩云',
        'STHUPO'='华文琥珀'
        )
    fmapr <- names(fmap); names(fmapr) <- unname(fmap)
    cat('==== 字体文件名与字体名称对应:\n')
    print(fmap)
    cat('==== 字体名与字体文件名对应:\n')
    print(fmapr)

    ## 找到某个字体的字体文件
    ## font.name 是字体名称
    find.font <- function(font.name){
```

```
        fname <- fmapr[font.name]
        flist0 <- font.files()
        flist1 <- sapply(strsplit(flist0, '[.]'), function(it) it[1])
        flist0[flist1==fname]
    }
    ff1 <- find.font('宋体')
    ff2 <- find.font('黑体')
    ff3 <- find.font('仿宋')
    ff4 <- find.font('隶书');

    font.add('cjk4',
             regular=ff1,
             bold=ff2,
             italic=ff3,
             bolditalic=ff4)
    ##browser()

    pdf('test-chinese.pdf'); on.exit(dev.off())
    par(family='cjk4')

    plot(c(0,1), c(0,1), type='n',
         axes=FALSE, xlab='', ylab='')
    text(0.1, 0.9, '正体', font=1)
    text(0.1, 0.8, '粗体', font=2)
    text(0.1, 0.7, '斜体', font=3)
    text(0.1, 0.6, '粗斜体', font=4)
}
test.chinese()
```

注意：图形参数 font=1 表示正体，font=2 表示粗体, font=3 表示斜体, font=4 表示粗斜体。

26.6 其它图形

26.6.1 相关系数图

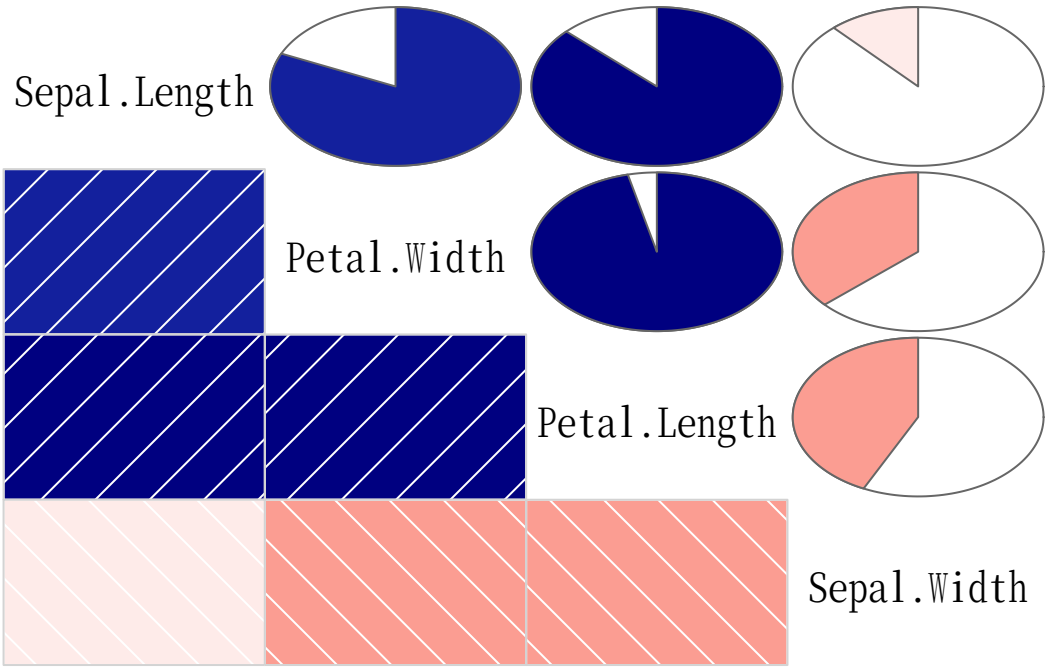
用 `cor(x)` 可以计算数据框 `x` 的各列的相关系数阵。corrgram 包的 `corrgram()` 函数可以将相关系数阵用图形表示，系数绝对值大小用色块颜色深浅表示，正负用两种颜色区分。

例如，计算 iris 数据框中四个测量值的相关系数并用矩阵表示：

```
library(corrgram)
R.iris <- cor(iris[,1:4])
print(round(R.iris, 2))
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length         1.00       -0.12         0.87        0.82
## Sepal.Width         -0.12        1.00        -0.43       -0.37
## Petal.Length         0.87       -0.43         1.00        0.96
## Petal.Width          0.82       -0.37         0.96        1.00
```

```
corrgram(
  R.iris, order=TRUE,
  lower.panel=panel.shade,
   upper.panel = panel.pie,
   text.panel = panel.txt
)
```



左下方用颜色代表正负相关，蓝色为正相关，红色为负相关，可以看出花萼长 (Sepal.Length)、花瓣宽 (Petal.Width) 和花瓣长 (Petal.Length) 相互为较强的正相关，但是花萼宽与其它三个变量为负相关。这个相关结果实际是虚假

的，因为样本不是单一总体而是来自三个总体。图形右上方用阴影部分大小和颜色深度代表相关系数绝对值，用颜色区分正负。

Chapter 27

ggplot 作图入门

27.1 介绍

Hadley Wickem 的 ggplot2 包是 R 的一个作图用的扩展包，它实现了“图形的语法”，将一个作图任务分解为若干个子任务，只要完成各个子任务就可以完成作图。在作常用的图形时，只需要两个步骤：首先将图形所展现的数据输入到 `ggplot()` 函数中，然后调用某个 `geom_xxx()` 函数，指定图形类型，如散点图、曲线图、盒形图等。

如果需要进一步控制图形细节，只要继续调用其它函数，就可以控制变量值的表现方式 (scale)、图例、配色等。这使得我们很容易做出基本的图形，在有需要时再深入学习，做出更为满意的图形。

与基本 R 中的作图系统相比，ggplot2 的作图有规律可循，作图结果直接达到出版印刷质量，除了可以按照一些既定模式做出常见种类的图形，也很容易将不同图形种类组合在一起，或者设计新颖的图形。基本 R 的作图结果通常不够美观，如果要将不同种类图形组合在一起比较困难，对设计新的图形类型支持也不够好。

ggplot2 的作图一般步骤为：

- 准备数据，一般为数据框，且一般为长表，即每个观测时间占一行，每个观测变量占一列。
- 将数据输入到 `ggplot()` 函数中，并指定参与作图的每个变量分别映射到哪些图形特性，比如映射为 x 坐标、y 坐标、颜色、形状等。这些映射称为 aesthetic mappings 或 aesthetics。
- 选择一个合适的图形类型，函数名以 `geom_` 开头，如 `geom_point()` 表示散点图。图形类型简称为 geom。将 `ggplot()` 部分与 `geom_xxx()` 部分用加号连接。到此已经可以作图，下面的步骤是进一步的细化设定。
- 设定适当的坐标系统，如 `coord_cartesian()`, `scale_x_log10()` 等。仍用加号连接。
- 设定标题和图例位置等，如 `labs()`。仍用加号连接。

这个流程的一个大致的模板为：

```
p <- ggplot(data=<输入数据框>,
  mapping=aes(<维度>=<变量名>,
    <维度>=<变量名>, <...>))
```

```
p + geom_<图形类型>(<...>) +
  scale_<映射>_<类型>(<...>) +
  coord_<类型>(<...>) +
  labs(<...>)
```

其中 `<...>` 表示额外的选项。变量 `p` 包含做出的图形的所有数据与设定，变量名可以任意取。

本章内容主要来自：

- Healy, Kieran (2018). Data Visualization: A Practical Introduction. Princeton University Press. https://socviz.co/index.html 这本书讲了 R 的 ggplot 的使用，也讲了一些可视化的一般性原则。
- Claus O. Wilke(2019). Fundamentals of Data Visualization. O'Reilly Media. https://serialmentor.com/dataviz/ 这本书虽然也使用 R 的 ggplot2 包，但正文中没有代码，主要讲作图有哪些考虑、各种图形类型。代码在 github 上可下载。
- Winston Chang(2018). R Graphics Cookbook. O'Relly Media. 网站：https://r-graphics.org/ 为第二版。讲了各种图的 R 程序。
- Wickham, Hadley (2016). Ggplot2: Elegant graphics for data analysis. New York: Springer.
- RStudio 的 ggplot2 概览：data-visualization-2.1.pdf

Wickham 的书主要需要安装 tidyverse 扩展包，安装时会自动安装其它一些有关扩展包。Healy 的的书需要通过如下程序安装 socviz 软件包：

```
devtools::install_github("kjhealy/socviz")
```

后续的例子中用到一些数据集:

- 来自 gapminder 扩展包的 gapminder 数据集，有若干个国家不同年份的一些数据，包括所属洲、期望寿命、人口数、人均 GDP。有 1704 个观测和 6 个变量。
- socviz 包的 `gss_sm` 数据集，是 2016 年美国一般社会调查数据的部分内容。有 2867 个观测，32 个变量。社会调查数据的变量主要取属性值，比如无序分类、有序分类、分组的数值、整数值等。
- socviz 包的 organdata 数据集，是 17 个 OECD 国家历年的器官捐献情况以及一些其它记录。
- socviz 扩展包的 `elections_historic` 数据集。包括美国历次总统大选当选人、所属党派、支持比例等。
- socviz 扩展包的 asasec 数据集。这是美国社会学学会 (ASA) 的各分会 2005 年到 2015 年的一些数据。
- ggplot2 包中的 `midwest` 数据集包含了美国中西部的一些县的统计数据，如面积等。
- 来自 ggplot2 包的钻石数据集。

gapminder 的头部：

```
library(gapminder)
head(gapminder, 20)
```

```
## # A tibble: 20 x 6
##    country     continent  year lifeExp      pop gdpPercap
##    <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
##  1 Afghanistan Asia       1952    28.8  8425333      779.
##  2 Afghanistan Asia       1957    30.3  9240934      821.
##  3 Afghanistan Asia       1962    32.0 10267083      853.
##  4 Afghanistan Asia       1967    34.0 11537966      836.
##  5 Afghanistan Asia       1972    36.1 13079460      740.
##  6 Afghanistan Asia       1977    38.4 14880372      786.
##  7 Afghanistan Asia       1982    39.9 12881816      978.
##  8 Afghanistan Asia       1987    40.8 13867957      852.
##  9 Afghanistan Asia       1992    41.7 16317921      649.
## 10 Afghanistan Asia       1997    41.8 22227415      635.
## 11 Afghanistan Asia       2002    42.1 25268405      727.
## 12 Afghanistan Asia       2007    43.8 31889923      975.
## 13 Albania     Europe     1952    55.2  1282697     1601.
## 14 Albania     Europe     1957    59.3  1476505     1942.
## 15 Albania     Europe     1962    64.8  1728137     2313.
## 16 Albania     Europe     1967    66.2  1984060     2760.
## 17 Albania     Europe     1972    67.7  2263554     3313.
## 18 Albania     Europe     1977    68.9  2509048     3533.
## 19 Albania     Europe     1982    70.4  2780097     3631.
## 20 Albania     Europe     1987    72    3075321     3739.
```

gss_sm 的头部：

```
head(gss_sm, 20)
```

```
## # A tibble: 20 x 32
##     year    id ballot   age childs sibs  degree race  sex   region incom~1 relig
##    <dbl> <dbl> <labe> <dbl>  <dbl> <lab> <fct>  <fct> <fct> <fct>  <fct>   <fct>
##  1  2016     1 1         47      3 2     Bache~ White Male  New E~ $17000~ None
##  2  2016     2 2         61      0 3     High ~ White Male  New E~ $50000~ None
##  3  2016     3 3         72      2 3     Bache~ White Male  New E~ $75000~ Cath~
##  4  2016     4 1         43      4 3     High ~ White Fema~ New E~ $17000~ Cath~
##  5  2016     5 3         55      2 2     Gradu~ White Fema~ New E~ $17000~ None
##  6  2016     6 2         53      2 2     Junio~ White Fema~ New E~ $60000~ None
##  7  2016     7 1         50      2 2     High ~ White Male  New E~ $17000~ None
##  8  2016     8 3         23      3 6     High ~ Other Fema~ Middl~ $30000~ Cath~
##  9  2016     9 1         45      3 5     High ~ Black Male  Middl~ $60000~ Prot~
```

```
## 10  2016    10 3         71      4 1     Junio~ White Male  Middl~ $60000~ None
## 11  2016    11 2         33      5 4     High ~ Black Fema~ Middl~ under ~ Prot~
## 12  2016    12 1         86      4 4     High ~ White Fema~ Middl~ under ~ Prot~
## 13  2016    13 2         32      3 3     High ~ Black Male  Middl~ $8 000~ Cath~
## 14  2016    14 3         60      5 6     High ~ Black Fema~ Middl~ $12500~ Prot~
## 15  2016    15 2         76      7 0     Lt Hi~ White Male  New E~ $40000~ Cath~
## 16  2016    16 3         33      2 1     High ~ White Fema~ New E~ $50000~ Other
## 17  2016    17 3         56      6 3     High ~ White Male  New E~ $50000~ Prot~
## 18  2016    18 2         62      5 8     Lt Hi~ Other Fema~ New E~ $5 000~ Prot~
## 19  2016    19 2         31      0 2     Gradu~ Black Male  New E~ $35000~ None
## 20  2016    20 1         43      2 0     High ~ Black Male  New E~ $25000~ Prot~
## # ... with 20 more variables: marital <fct>, padeg <fct>, madeg <fct>,
## #   partyid <fct>, polviews <fct>, happy <fct>, partners <fct>, grass <fct>,
## #   zodiac <fct>, pres12 <labelled>, wtssall <dbl>, income_rc <fct>,
## #   agegrp <fct>, ageq <fct>, siblings <fct>, kids <fct>, religion <fct>,
## #   bigregion <fct>, partners_rc <fct>, obama <dbl>, and abbreviated variable
## #   name 1: income16
```

27.2 作图的一般原则

关于什么是好的图形和坏的图形，William S. Cleveland，Edward R. Tufte 等人有很多的研究。

坏的图形可能有如下缺点：

- 坏的品味。统计图形应该用尽可能少的图形元素表示尽可能多的数据，从打印图形而言，即数据量与所用墨水比例越大越好。没有必要的颜色、三维形态经常会影响读者对图形的认读。这是 Edward R. Tufte 的观点，但是过于极端也不好。
- 坏的数据。即使图形本身的做法没有问题，选择了错误的或者不合适的数据也会误导读者，甚至于用错误数据做的很专业的图形会比粗陋的图形更能误导读者。
- 坏的感知。不好的颜色选择、三维形状、坐标轴范围、宽高比都有可能对读者的认知有影响。

作图时应考虑的一些因素：

- 数值型变量的不同值可以表示为：
  - 同一坐标轴上的不同位置、
  - 不同轴上的位置、
  - 不同长度、
  - 不同角度或者斜率、
  - 不同面积、

  - 三维空间中的不同位置、
  - 颜色的不同明暗度、
  - 不同颜色饱和度、
  - 曲线的不同曲率、
  - 三维体积，

  这些表示的选择项越往后越难以被读者正确辨识。使用颜色时，应该使用渐变的明暗度或者渐变色。

- 分类变量的不同值可以表示为：

  - 不同分组、
  - 不同颜色、
  - 三维动态、
  - 不同符号。

  这些表示的选择项越往后越难辨识。使用颜色时，应该使用明显不同的颜色而不应该使用渐变色。

- 对于最少是零的变量，是否应该以零作为坐标轴的最低值需要考虑，但没有一定的规则。同一组数据在不同的坐标范围或者长宽比下曲线的斜率会有很大差别。

27.3 散点图：ggplot 入门

27.3.1 基本的散点图

以 gapminder 数据集作为输入数据，做出简单的散点图，并逐步进行改善。这个数据集有多个国家在多个年份的期望寿命与人均 GDP 值，作期望寿命对人均 GDP 的散点图，每个国家的每个年份作为一个点。散点图最重要的映射是 x 轴与 y 轴两个维度。

首先调用 `ggplot()` 函数，指定数据集，将人均 GDP 映射到 x 轴，将期望寿命映射到 y 轴，结果保存为一个 R 变量：

```
p <- ggplot(data = gapminder,
  mapping = aes(
    x = gdpPercap,
    y = lifeExp))
```

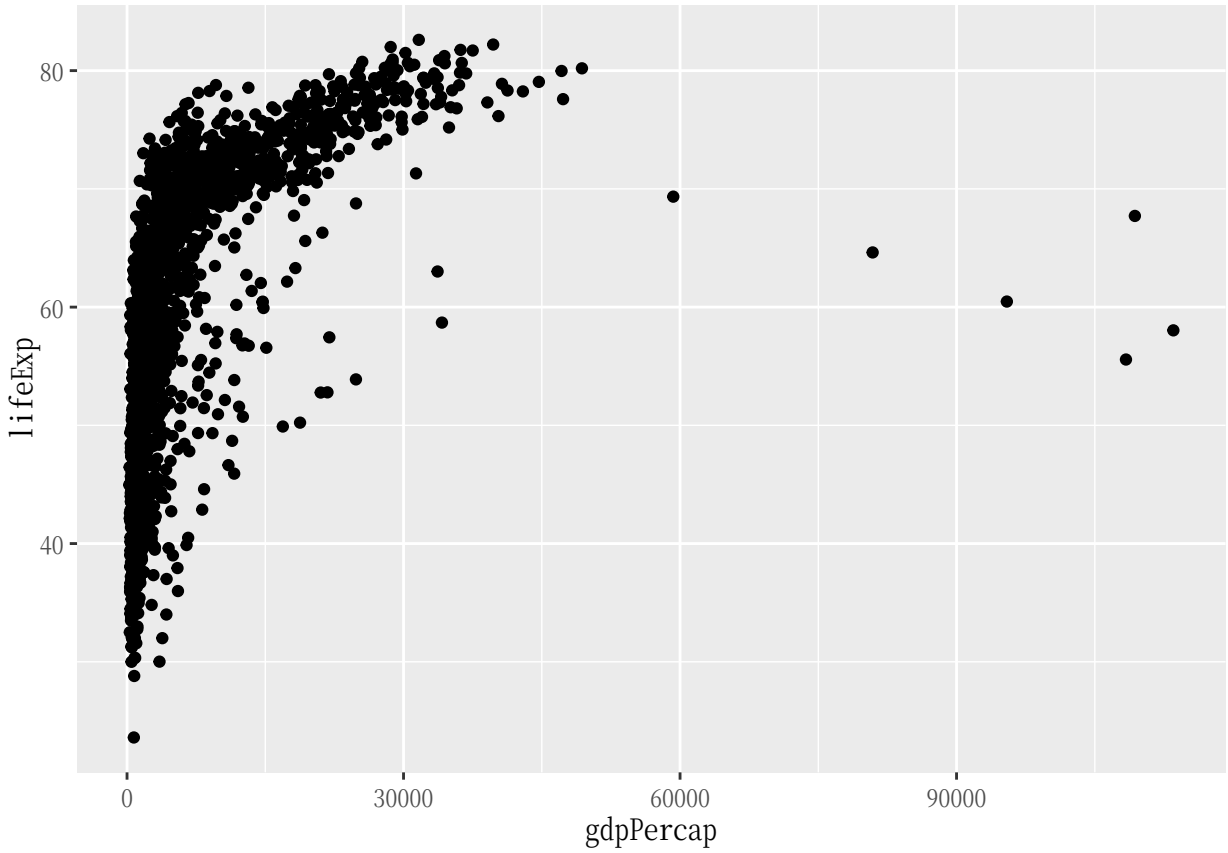
x、y 轴是最常见的映射，也可以将变量映射为颜色、符号、线型等，这时不需要指定具体的颜色、符号、线型，而是将变量映射为这些图形元素类型。

`ggplot()` 的调用中，可以省略 `data =`, `mapping =`, `x =`, `y =`，写成：

```
p <- ggplot(gapminder, aes(gdpPercap, lifeExp))
```

在如上指定了数据和映射后，只要用 geom_xxx() 指定一个图形类型，并与 ggplot() 的结果用加号连接就可以作图了，如：

```
p + geom_point()
```



实际上，上面的程序等同于调用 print(p + geom_point())。在 R 函数中或者在循环中需要显式地调用 print()，否则不会显示结果。当载入了 tidyverse 系统时可以写成 (p + geom_point()) |> print()。

显式地调用 print()，将上一程序写成：

```
p1 <- p + geom_point()
print(p1)
```

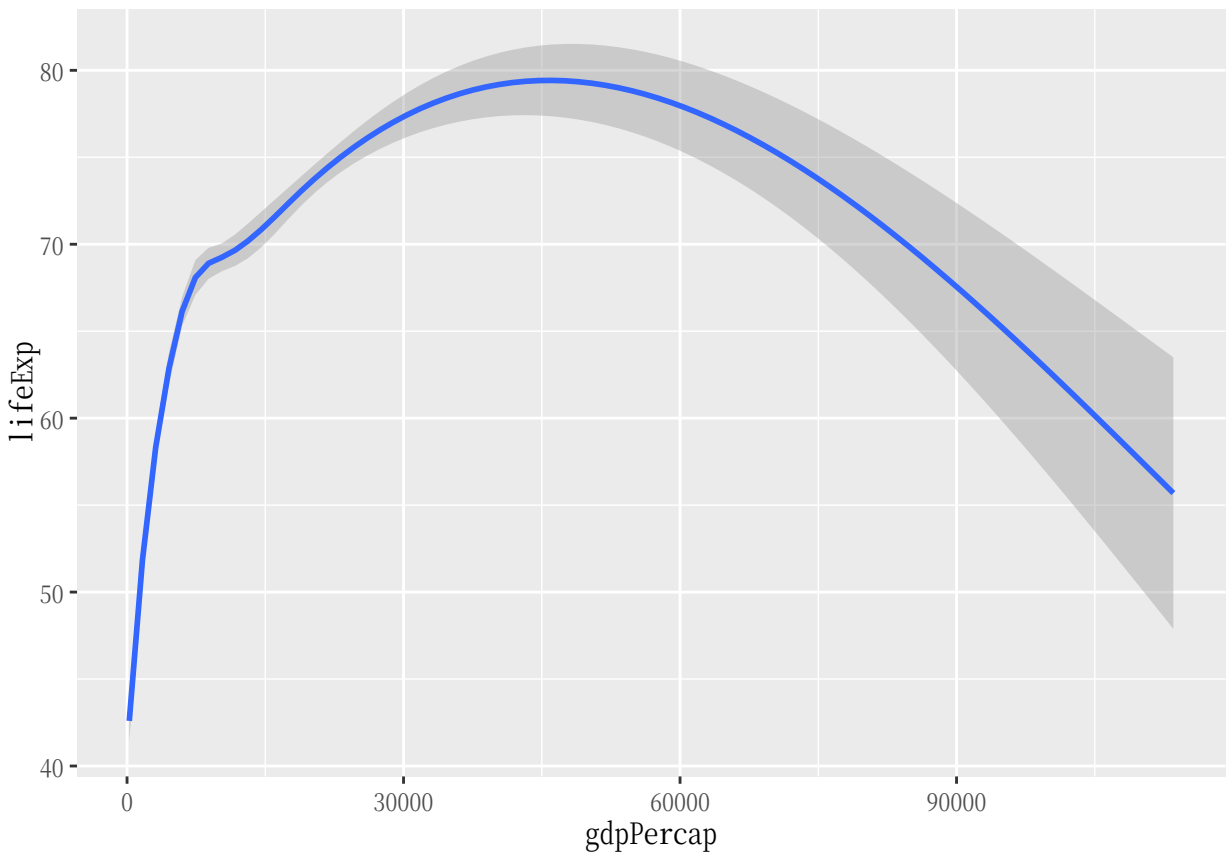
27.3.2 逐步改善

指定数据集、指定映射、选择适当的图形类型就可以做出基本的图形，随后可以逐步对坐标系、坐标系刻度、标签与图例、配色等进行改善。实际上，ggplot2 包已经提供了十分合理的预设值，用户只要进行一些必要的改动即可。

作图步骤之间用加号连接，这是 ggplot 包特有的语法。例如，用相同的映射做出拟合曲线图：

```
p + geom_smooth()
```

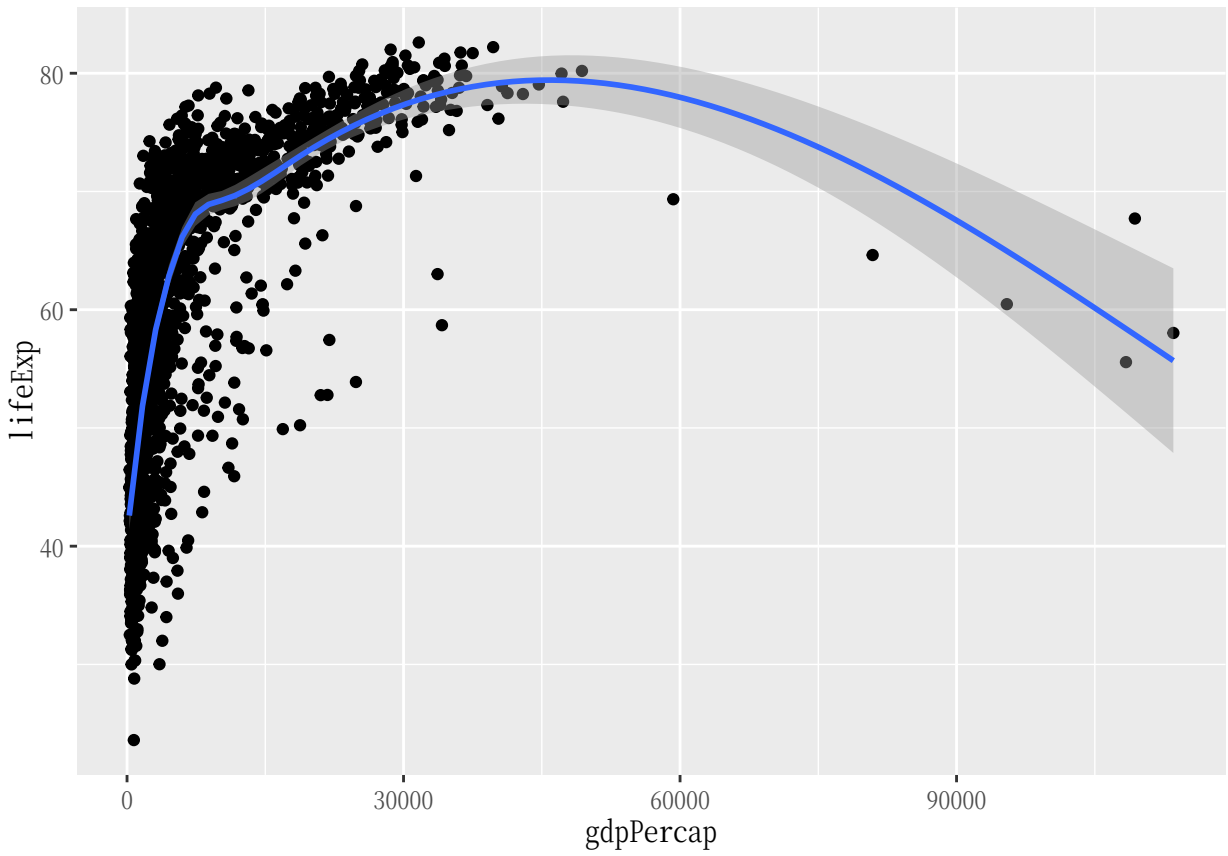
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



用相同的映射做出散点图并叠加拟合曲线图：

```
p + geom_point() + geom_smooth()
```

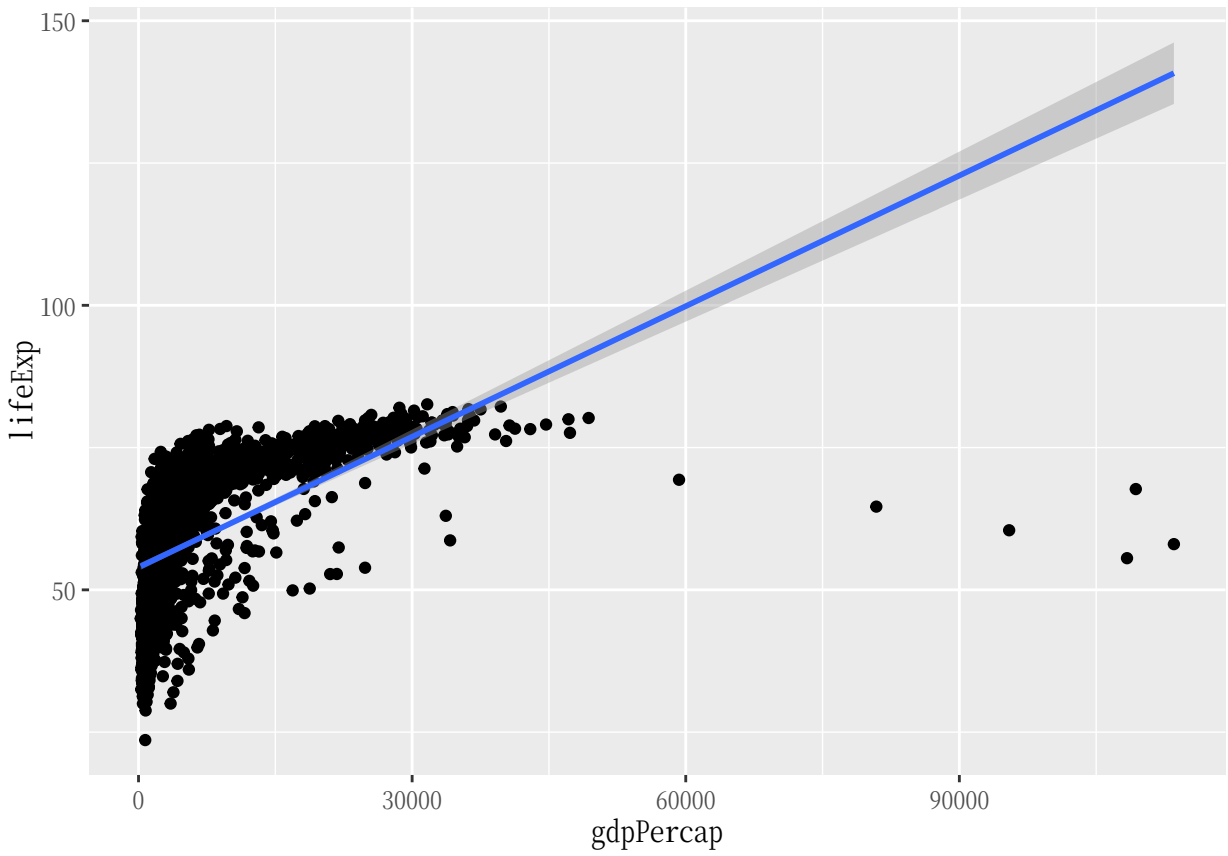
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



geom_smooth() 的默认设置调用了 gam() 函数来拟合曲线，可以用 geom_smooth() 的参数选择不同的拟合方法，如直线拟合：

```
p + geom_point() + geom_smooth(method="lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```

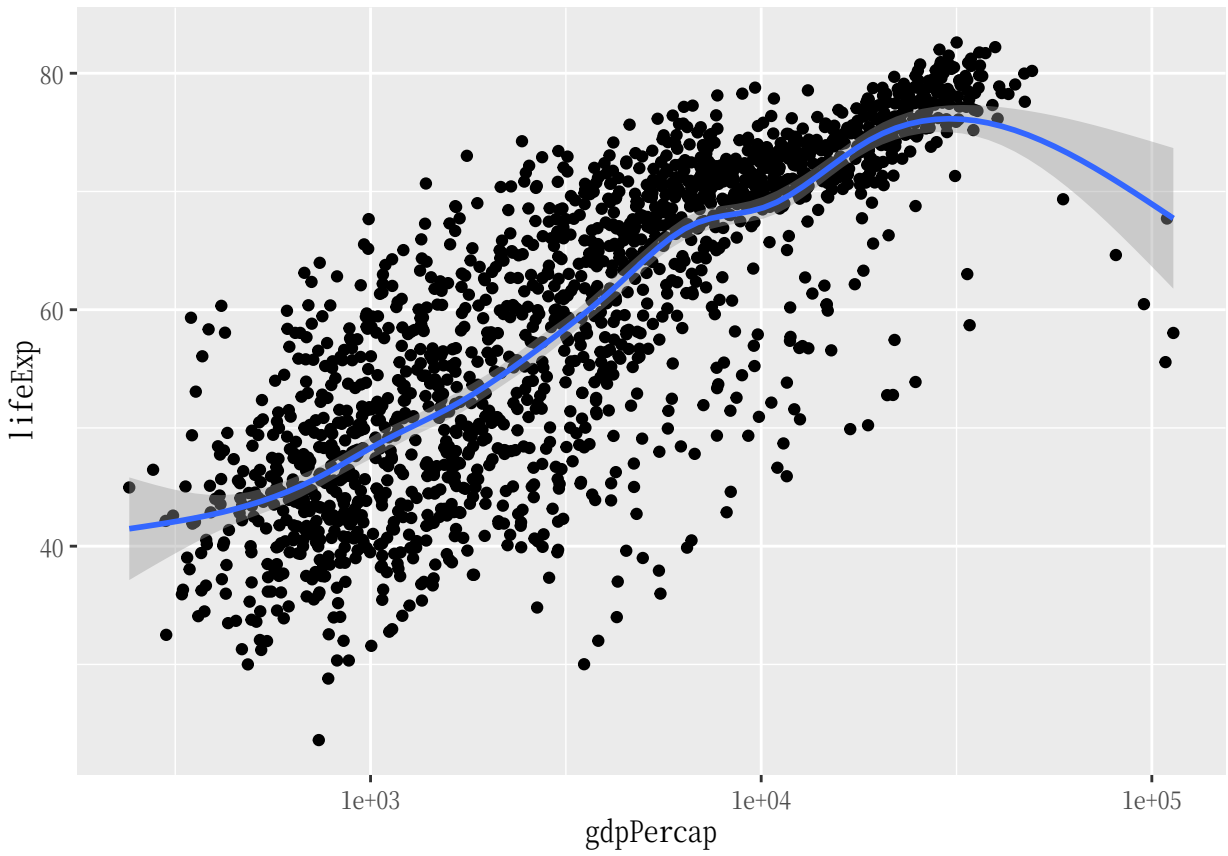


注意 `geom_xxx()` 函数计算所需的变量值是从 `ggplot()` 函数保存在变量 `p` 中的信息提取的。

在以上的所有图形中，x 轴变量（人均 GDP）分布非正态，严重右偏，使得大多数散点重叠地分布在直角坐标系的左下角。将 x 轴用对数刻度可以改善，函数为 `scale_x_log10()`:

```
p + geom_point() +
  geom_smooth(method="gam") +
  scale_x_log10()
```

```
## `geom_smooth()` using formula 'y ~ s(x, bs = "cs")'
```

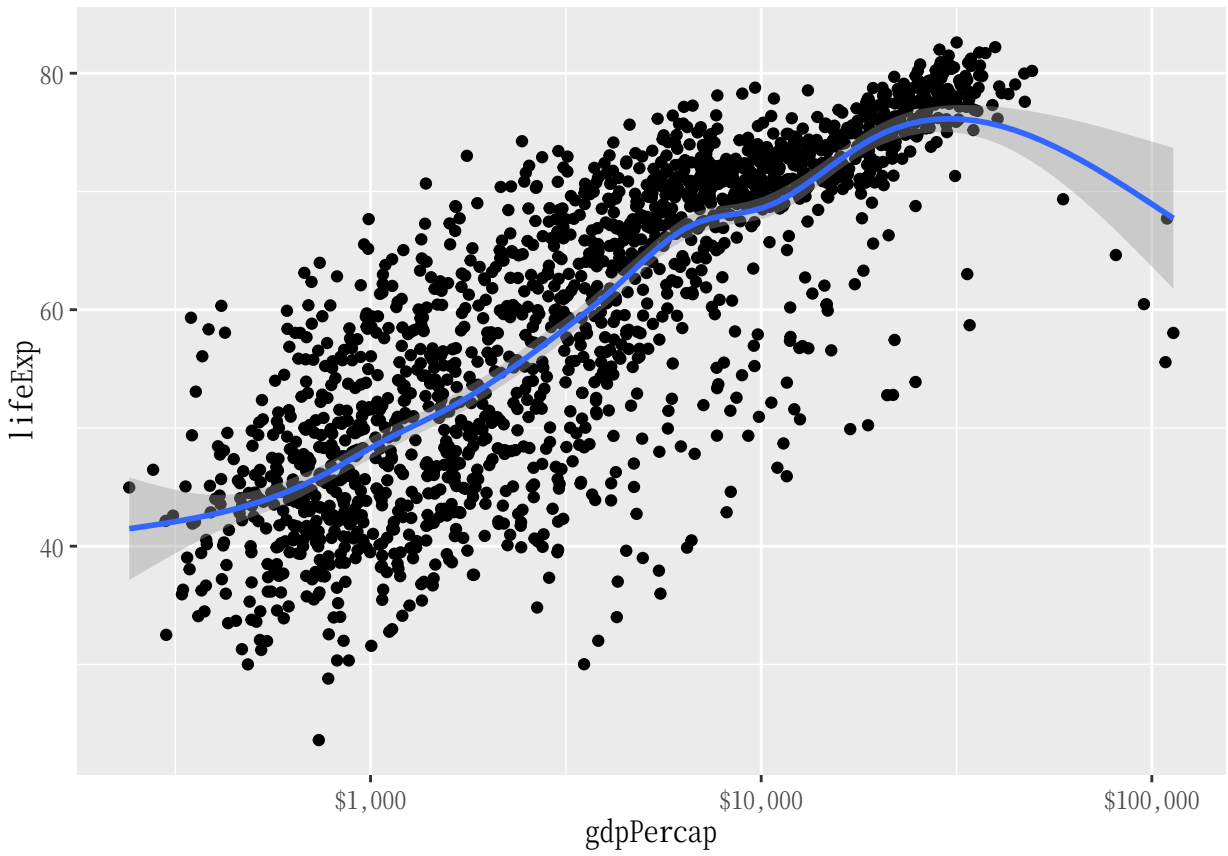


广义可加模型拟合的曲线基本是一条直线。注意，对数刻度实际上是对原始数据进行对数变换，而 `geom_smooth()` 的拟合计算是在对数变换之后进行的。

刚刚的图形的横坐标轴刻度不太友好，可以调用 scales 扩展包的适当函数进行改善，作为 `scale_x_log10()` 的 `labels` 选项：

```
p + geom_point() +
  geom_smooth(method="gam") +
  scale_x_log10(labels=scales::dollar)
```

```
## `geom_smooth()` using formula 'y ~ s(x, bs = "cs")'
```



`scale_xxx()` 的 `labels` 选项指定如何标出坐标刻度数字，参数值是一个函数对象，如果 scales 包中找不到适当的功能，可以自定义一个函数将数值转换为字符串。scales 包提供了 `comma`, `date`, `dollar`, `math`, `number`, `ordinal`, `pvalue`, `scientific`, `time` 等坐标刻度值转换函数。

27.3.3 颜色、符号、线型等映射

在 `ggplot()` 函数的 `mapping` 参数的 `aes()` 设定中将变量映射到 x、y 轴，颜色、符号、线型等图形元素类型，也可以作为图形设置将某些图形元素设置为固定值。

例如，用不同颜色表示不同大洲，就是将 `continent` 变量映射到 `color`:

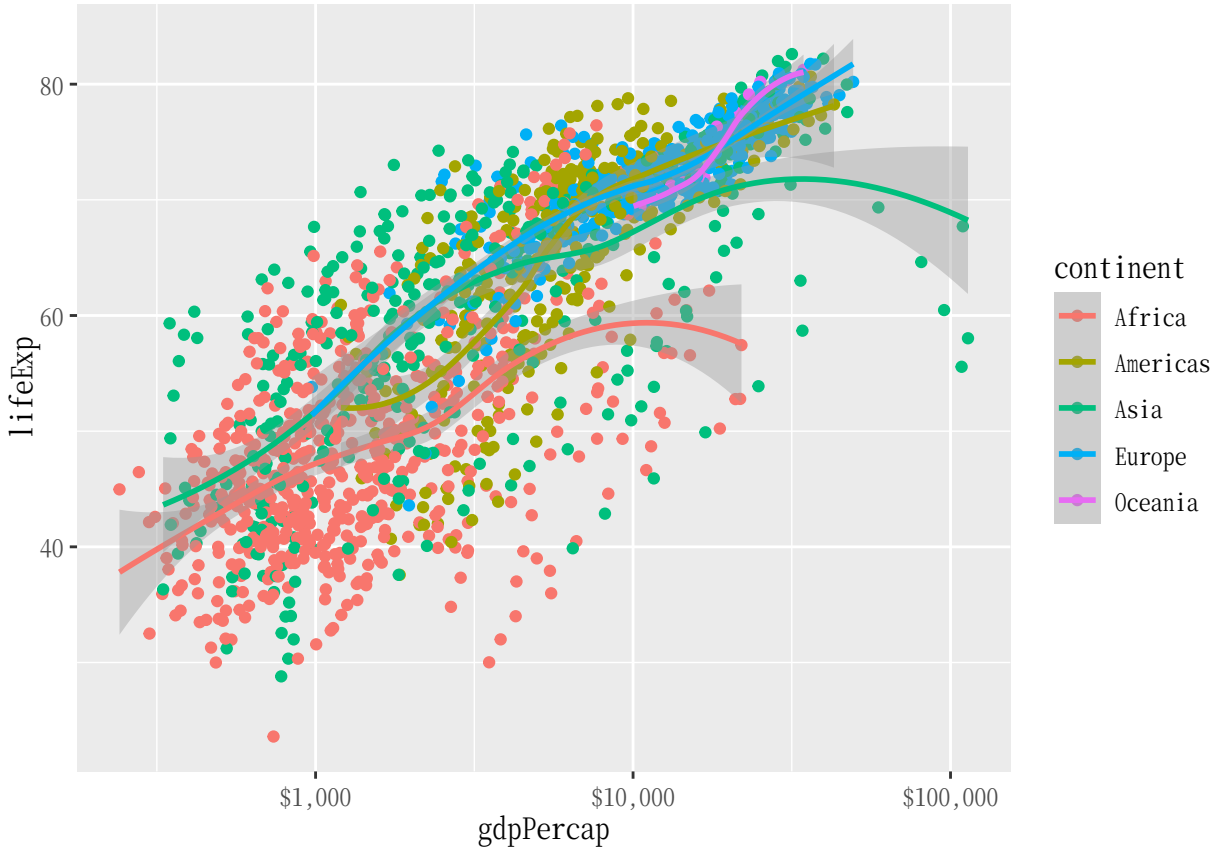
```
p <- ggplot(gapminder, aes(
    x = gdpPercap,
    y = lifeExp,
    color = continent))
```

程序中仅指定了将大洲映射到颜色维，并不具体指定所用的颜色。

作带有局部多项式曲线拟合的散点图：

```
p + geom_point() +
  geom_smooth(method="loess") +
  scale_x_log10(labels=scales::dollar)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

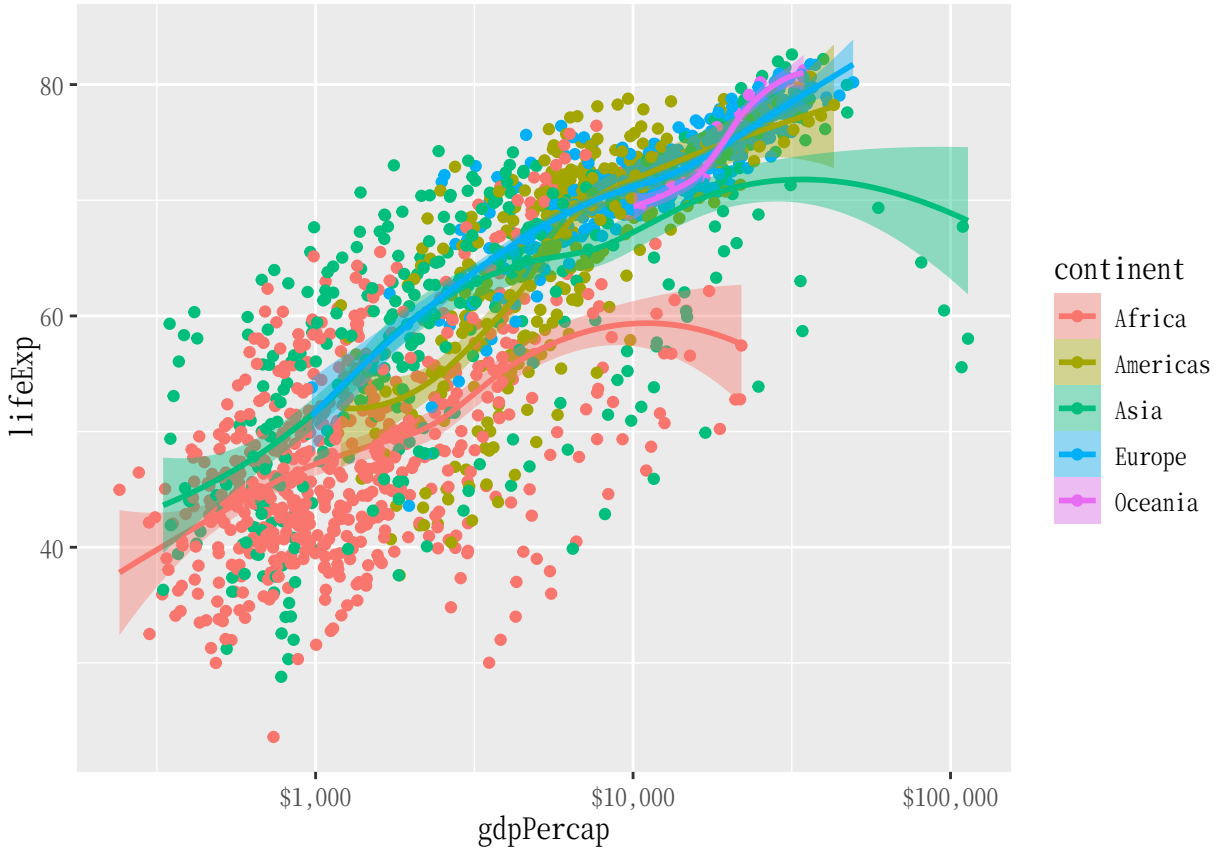


可以看出，不同散点用了不同颜色表示其 continent 变量的值，五个大洲分别进行了曲线拟合，曲线使用了不同颜色但置信域颜色相同，使得难以认读。在图形右侧自动生成了颜色与 continent 变量值的对应关系图例。

下面的图形仍分不同大洲作曲线拟合，并将置信区间阴影的颜色也用不同大洲区分，方法是在 `aes()` 中将 `color` 和 `fill` 都指定为变量 `continent`:

```
p <- ggplot(gapminder, mapping = aes(
    x = gdpPercap,
    y = lifeExp,
    color = continent,
    fill = continent))
p + geom_point() +
  geom_smooth(method="loess") +
  scale_x_log10(labels=scales::dollar)
```

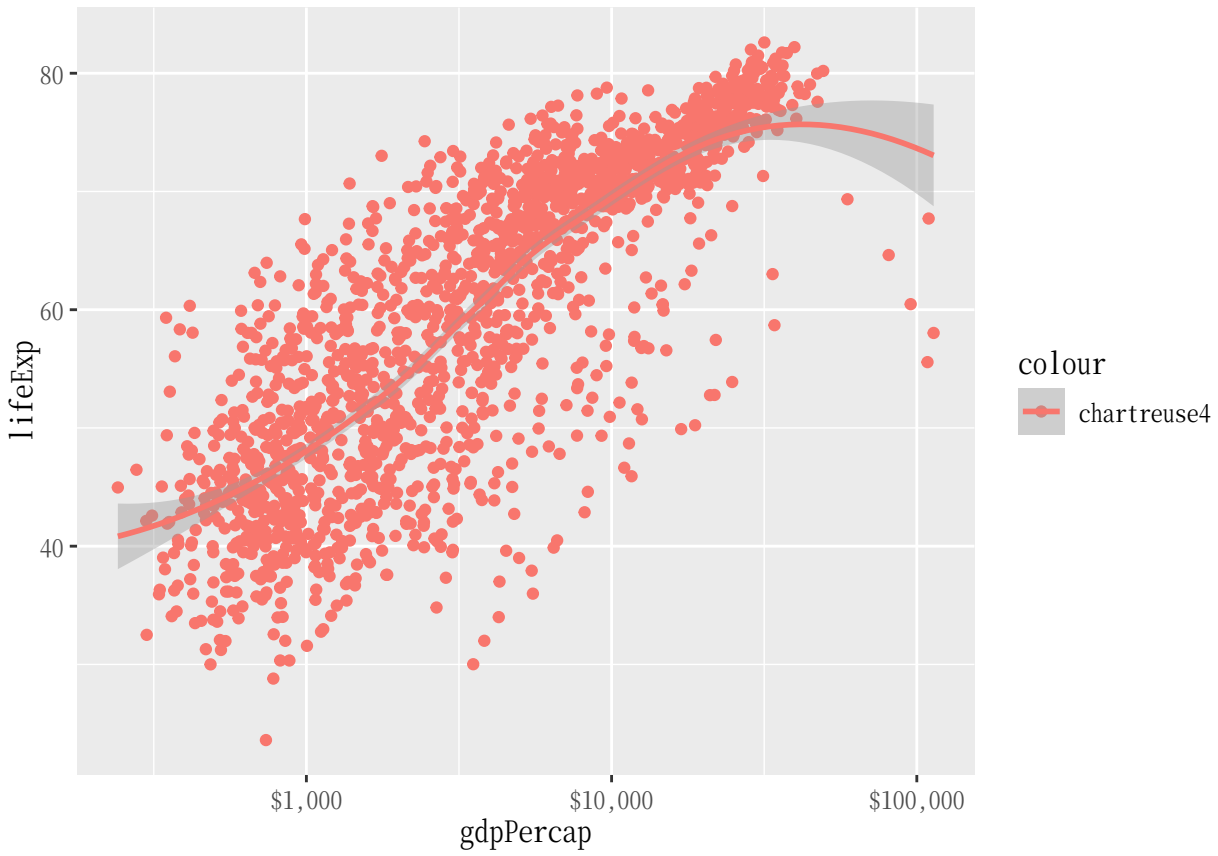
```
## `geom_smooth()` using formula 'y ~ x'
```



尝试将颜色指定为一个固定值，如：

```
p <- ggplot(gapminder, aes(
    x = gdpPercap,
    y = lifeExp,
    color = "chartreuse4"))
p + geom_point() +
  geom_smooth(method="loess") +
  scale_x_log10(labels=scales::dollar)
```

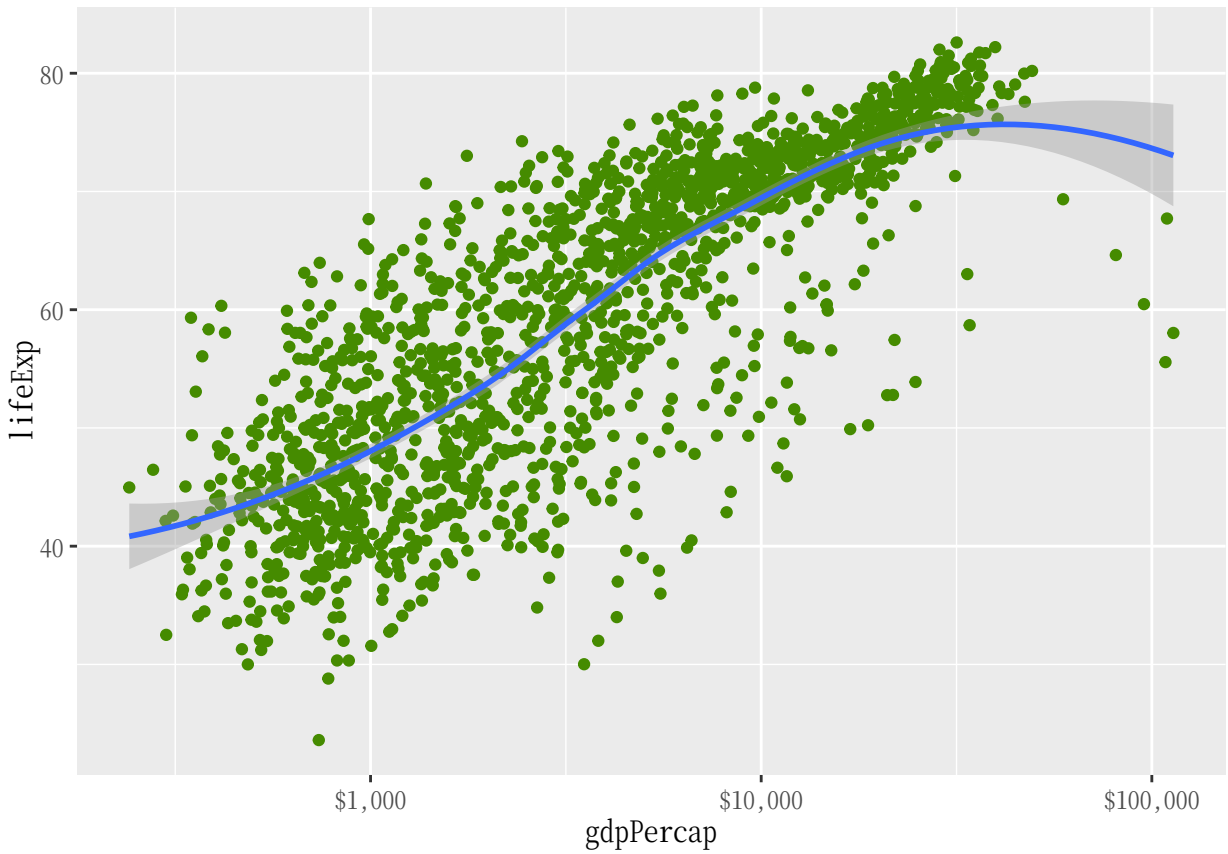
```
## `geom_smooth()` using formula 'y ~ x'
```



我们发现，散点并没有使用草绿色，而且图形右侧有一个 `chartreuse4` 图例。这是因为，`aes()` 仅用来指定变量与图形元素类型的映射，所以实际上是生成了一个仅有一个常数值`"chartreuse4"` 的新变量，用颜色表示这个新变量。为了指定固定颜色，应将 `color=` 作为 `geom_xxx()` 函数的选项，而不是放在 `aes()` 映射中，如：

```
p <- ggplot(gapminder, aes(
    x = gdpPercap,
    y = lifeExp))
p + geom_point(color="chartreuse4") +
  geom_smooth(method="loess") +
  scale_x_log10(labels=scales::dollar)
```

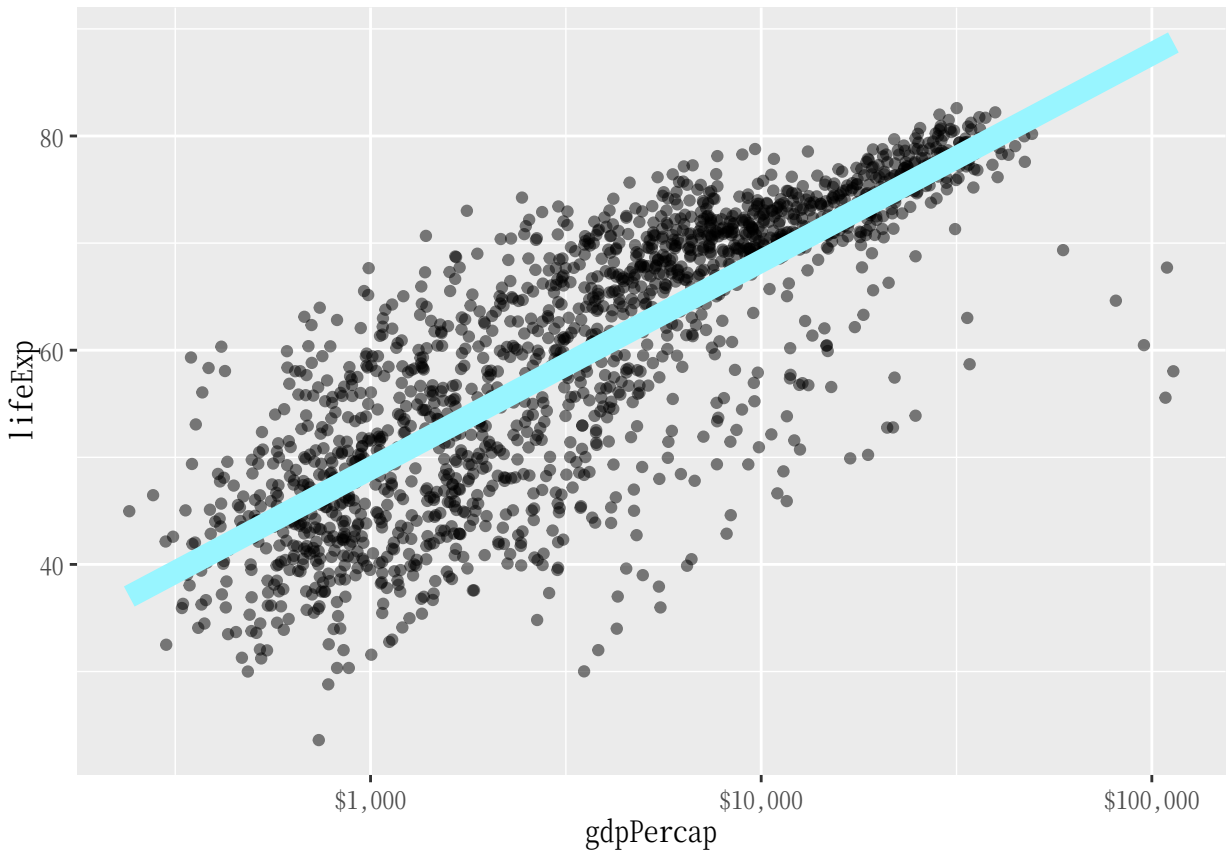
```
## `geom_smooth()` using formula 'y ~ x'
```



geom_xxx() 函数接受许多关于颜色、透明度、符号、线型的设置参数。比如，下面的程序指定了散点的透明度，以及拟合直线的粗细：

```
p + geom_point(alpha=0.5) +
  geom_smooth(method="lm", color="cadetblue1", se = FALSE, size = 4, alpha = 0.3) +
  scale_x_log10(labels=scales::dollar)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



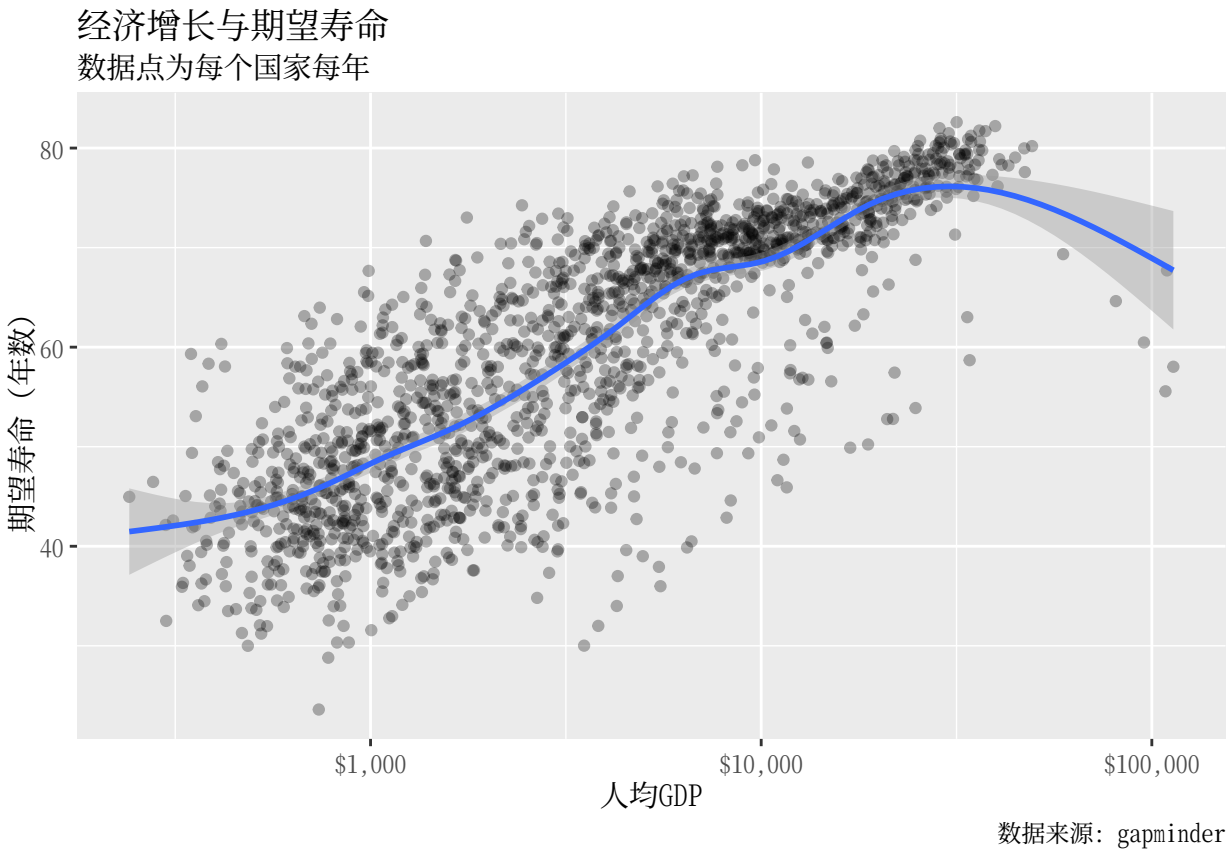
程序中 `size` 指定了线的以毫米为单位的粗细，`se = FALSE` 关闭了置信区间显示。用 `alpha =` 设置了透明度，取 0 和 1 之间的值，数值越小越透明。在有许多个点时适当设置透明度可以比较好地显示出重叠的点，重叠点越多点的颜色越深。虽然这里设置了固定的透明度，也可以在 `aes()` 中将透明度 `alpha` 映射到某个变量，使得该变量值大小用点的透明度表示。

画线时可以用 `linetype` 参数指定线型，0 表示实线，1 到 6 分别表示不同的虚线线型。

下面用 `labs()` 函数给图形加上适当的标题：

```
p <- ggplot(gapminder, aes(
    x = gdpPercap,
    y = lifeExp))
p + geom_point(alpha = 0.3) +
  geom_smooth(method="gam") +
  scale_x_log10(labels=scales::dollar) +
  labs(
    x = " 人均 GDP",
    y = " 期望寿命（年数）",
    title = " 经济增长与期望寿命",
    subtitle = " 数据点为每个国家每年",
    caption = " 数据来源: gapminder"  )
```

```
## `geom_smooth()` using formula 'y ~ s(x, bs = "cs")'
```



可以看出，`labs()` 规定了上方的标题、小标题，x 轴、y 轴的标题，右下方的标注 (caption)。坐标轴刻度数值的规定则需要在 `scale_xxx()` 函数中给出。

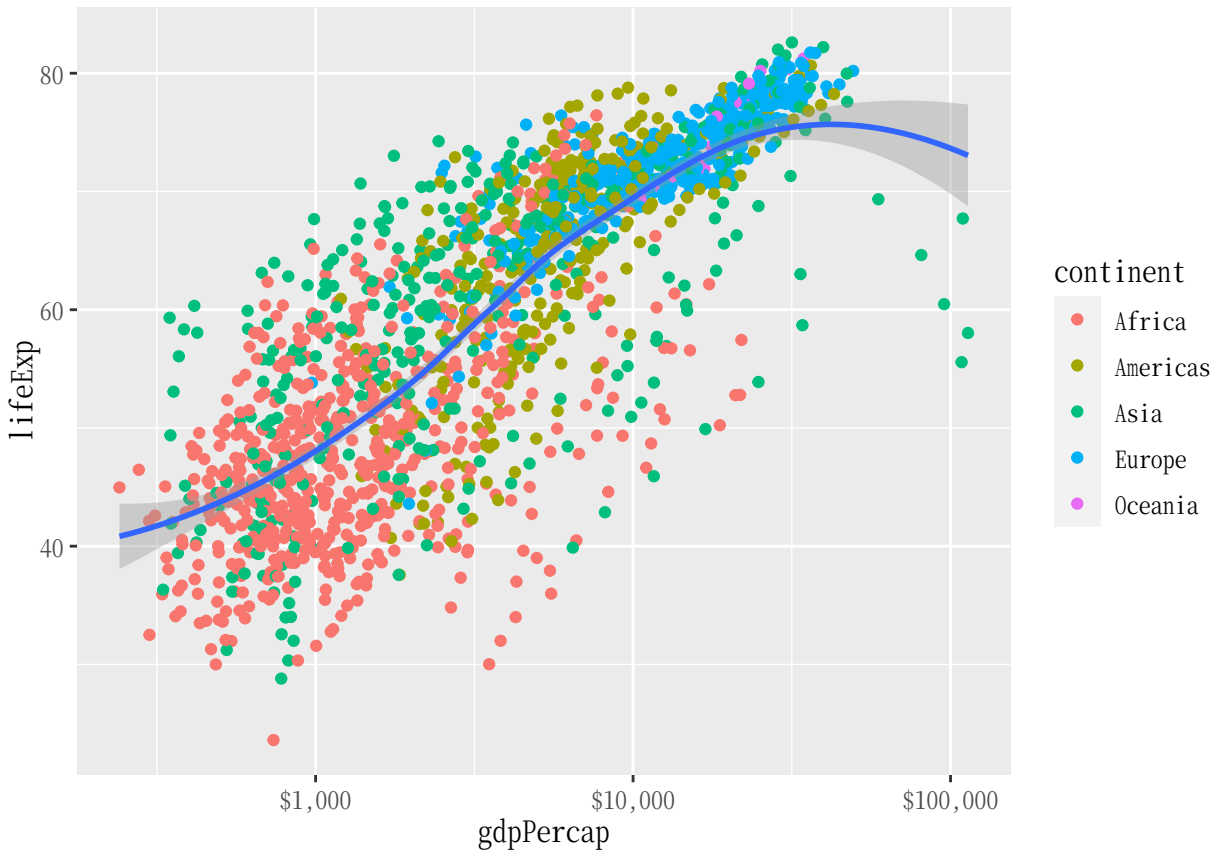
27.3.4 在 geom 函数中映射变量

在前面的一个例图中，在 `ggplot()` 函数中将 `color` 和 `fill` 映射到了 `continent` 变量，使得不仅散点颜色代表了不同大洲，还使得每个大洲单独拟合了曲线。如果希望所有大洲拟合同一条曲线怎么办？

在必要时，可以在 `geom_xxx()` 函数中用 `mapping = aes(<...>)` 单独指定变量映射。例如，下面的程序在 `geom_point()` 中将不同大洲映射为不同颜色，而不影响 `geom_smooth()` 中的颜色以及分组：

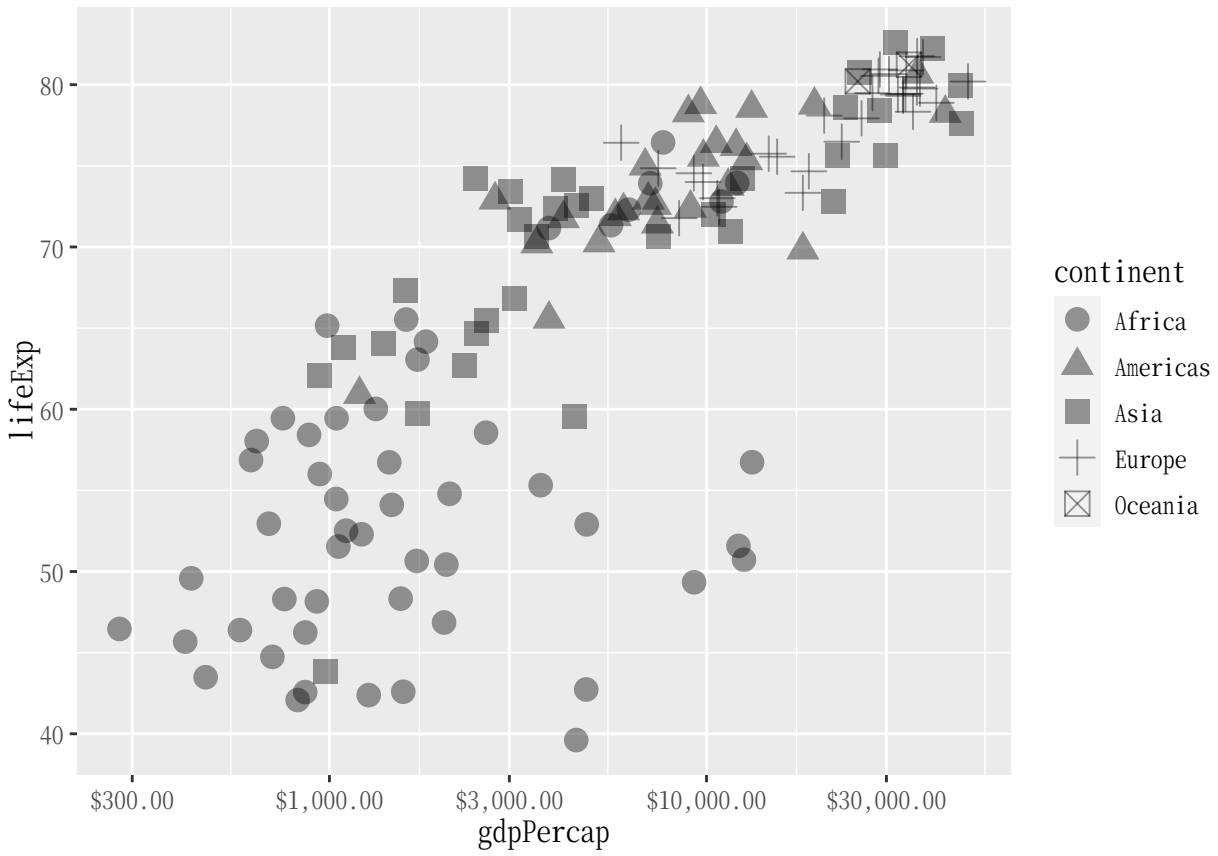
```
p <- ggplot(gapminder, aes(
    x = gdpPercap,
    y = lifeExp))
p + geom_point(mapping = aes(color = continent)) +
  geom_smooth(method="loess") +
  scale_x_log10(labels=scales::dollar)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



也可以将一个分类变量映射到不同绘图符号。例如，取 gapminder 2007 年数据子集，将大洲映射到符号 (shape)：

```
p <- ggplot(
  data = filter(gapminder, year == 2007),
  mapping = aes(
    x = gdpPercap,
    y = lifeExp,
    shape = continent))
p + geom_point(alpha = 0.4, size = 4) +
  scale_x_log10(labels=scales::dollar)
```



这种映射仅适用于点数比较少的情况，还用了 `size` 参数指定符号的大小（单位：毫米）。如果所有点使用同一符号并需要指定符号，可以在 `geom_point()` 中用 `shape` 参数指定，可以用 0 到 25 的整数值表示，比如 19 为实心点，也可以用字符串符号名称表示，如`"circle"` 表示实心点。参见 ggplot2 帮助目录中的 vignette ggplot2: ggplot2-specs。

注意，绘图时参与映射的分类变量会自动产生分类效果，`color` 映射与 `fill` 映射到分类变量时常常会起到与添加 `group` 维相同的作用，但为了逻辑清晰起见，需要分组时还应该显式地映射 `group` 维。

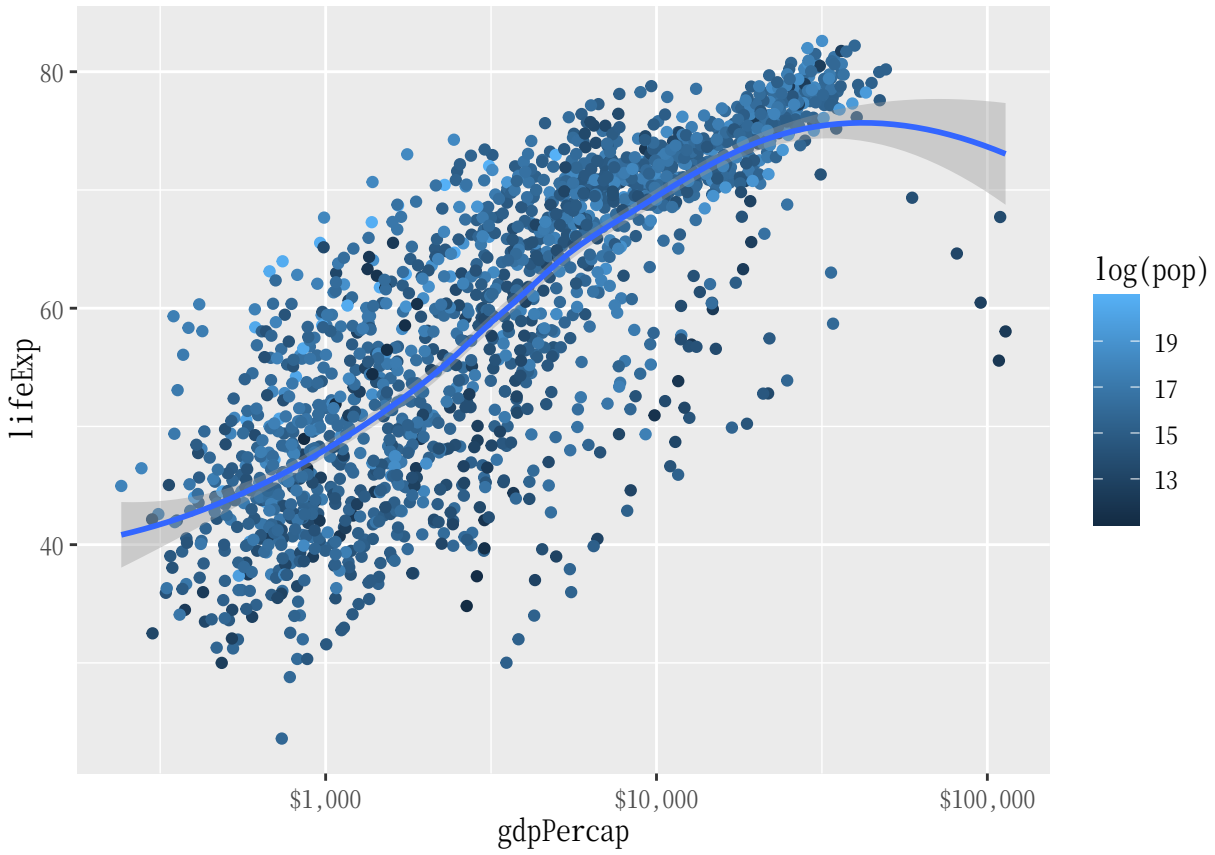
27.3.5 连续变量的颜色映射

也可以将连续变量映射为渐变色。除了表示二元函数的等值线图以外这种方法并不利于读者认读。

例如，将人口数取自然对数映射为渐变色：

```
p <- ggplot(gapminder, aes(
    x = gdpPercap,
    y = lifeExp,
    color = log(pop)))
p + geom_point() +
  geom_smooth(method="loess") +
  scale_x_log10(labels=scales::dollar)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



这里不同散点的颜色是连续变化的，右侧的图例仅显示了有限的一些代表值。

27.3.6 保存图像

如果使用 Rmarkdown 制作图文，图像会自动进入编译的结果（如 PDF、Word、HTML）中，图像大小、输出大小可以用 Rmarkdown 的设置调整。

为了将最近生成的图形保存为 PNG 格式，用命令如

```
ggsave(filename=" 文件名.png")
```

保存为 PDF 格式：

```
ggsave(filename=" 文件名.pdf")
```

可以将制作的图形保存到了一个 R 变量中，在 `ggsave()` 中可以用 `plot=` 参数指定，如

```
ggout01 <- p + geom_point()
ggsave(filename=" 文件名.pdf", plot=ggout01)
```

在 ggsave() 中可以用 scale = 指定放大比例，用 height = 指定高度，用 width = 指定宽度，用 units = 指定高度和宽度的单位，如：

```
ggsave(filename=" 文件名.pdf", plot=ggout01,
       height=12, width=8, units="cm")
```

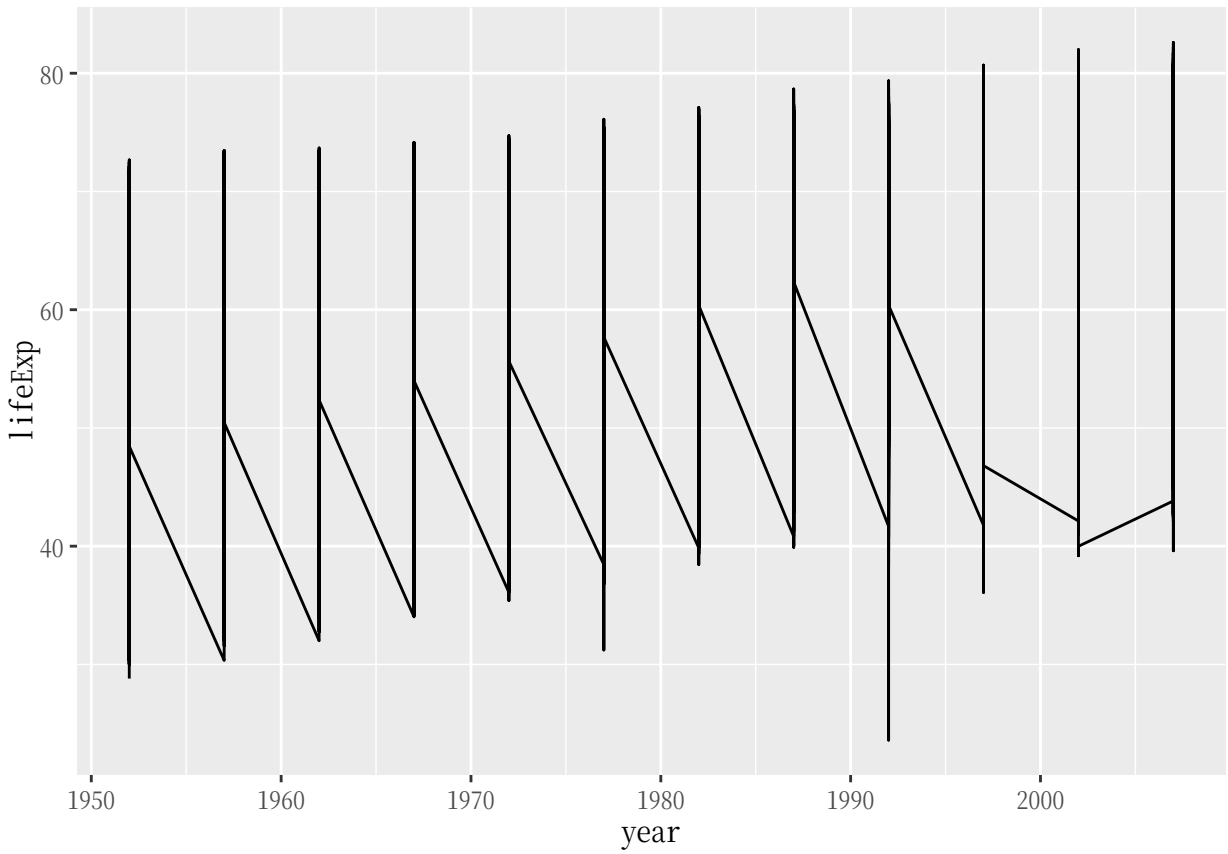
单位可以是 in, cm, mm。

27.4 折线图、分组、切片

27.4.1 图形中的分组和折线图

考虑 gapminder 数据集中每个国家的期望寿命随时间（年）的变化。用 geom_line() 可以画折线图。因为有许多国家，所以仅指定 x、y 变量无法得到所需图形，如：

```
p <- ggplot(gapminder, aes(
    x = year,
    y = lifeExp))
p + geom_line()
```

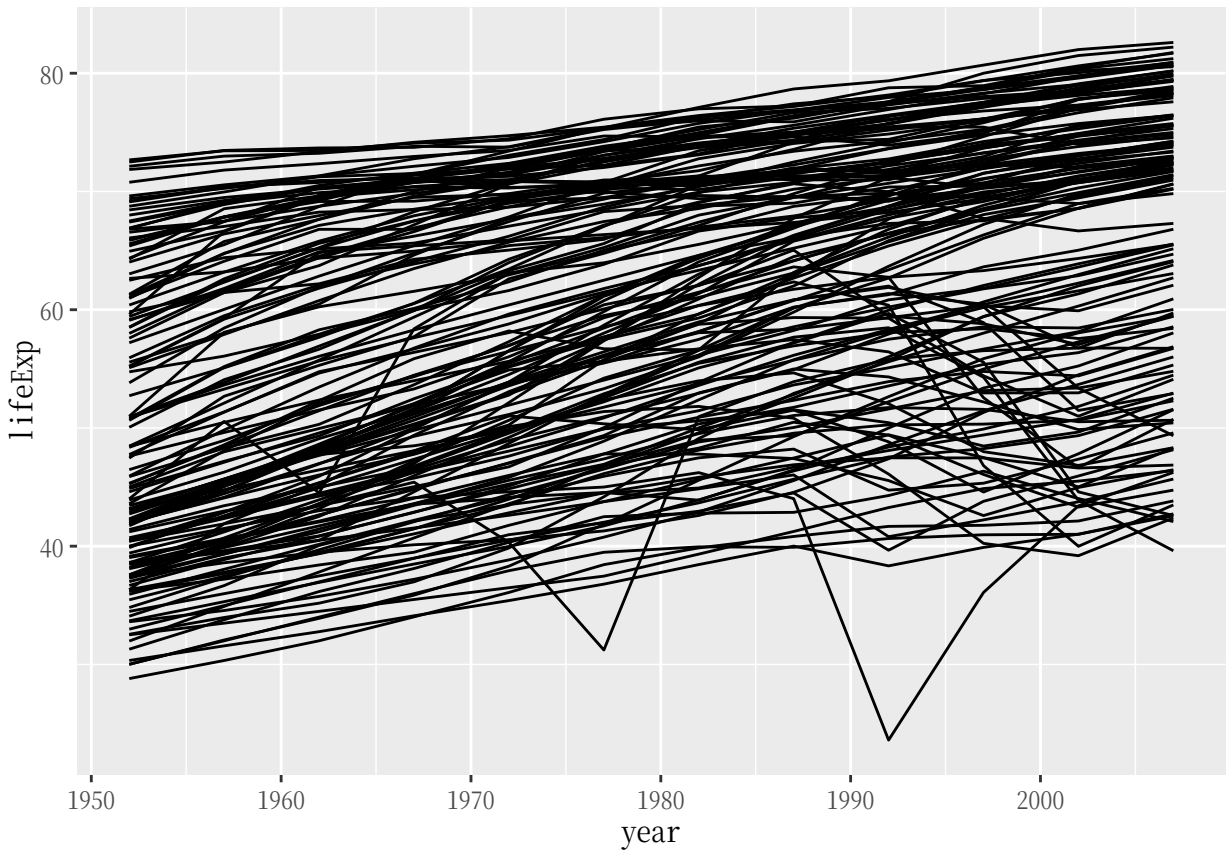


没有得出我们希望的每个国家一条曲线的效果。这是因为程序中没有指定需要按照国家分组，使得同一年的不同国家的坐标连成了一条竖线。

要注意的是，`geom_line()` 会自动将 x 坐标从小到大排序，然后再连接相邻的点。如果希望按输入数据的次序连接相邻的点，需要用 `geom_path()` 函数。

为了解决上图的问题，加入按照国家分组的设定。实际上，分组 (`group`) 与 `x`、`y`、`color`、`fill` 一样可以映射到一个变量，但仅能映射到分类变量。上述程序的改进如下：

```
p <- ggplot(gapminder, aes(
    x = year,
    y = lifeExp,
    group = country))
p + geom_line()
```



结果图形中每一条曲线对应一个国家。为了查探其中最下方的不稳定曲线是哪一个国家，使用筛选观测的功能：

```
gapminder |>
  filter(lifeExp < 30, year >= 1990)
```

```
## # A tibble: 1 x 6
##   country continent  year lifeExp     pop gdpPercap
##   <fct>   <fct>     <int>   <dbl>   <int>     <dbl>
## 1 Rwanda  Africa     1992    23.6 7290203      737.
```

该国家为 Rwanda。

如果需要按照两个或多个分类变量交叉分组，可以给 `group` 维指定 `interaction(...)`，其中`...` 是分类变量表。

`geom_line()` 用来绘制从左向右连接的折线。比如，仅绘制 `Rwanda` 的期望寿命时间序列：

```
gapminder |>
  filter(country == "Rwanda") |>
  ggplot(aes(
    x = year,
```

```
    y = lifeExp)) +
  geom_line()
```

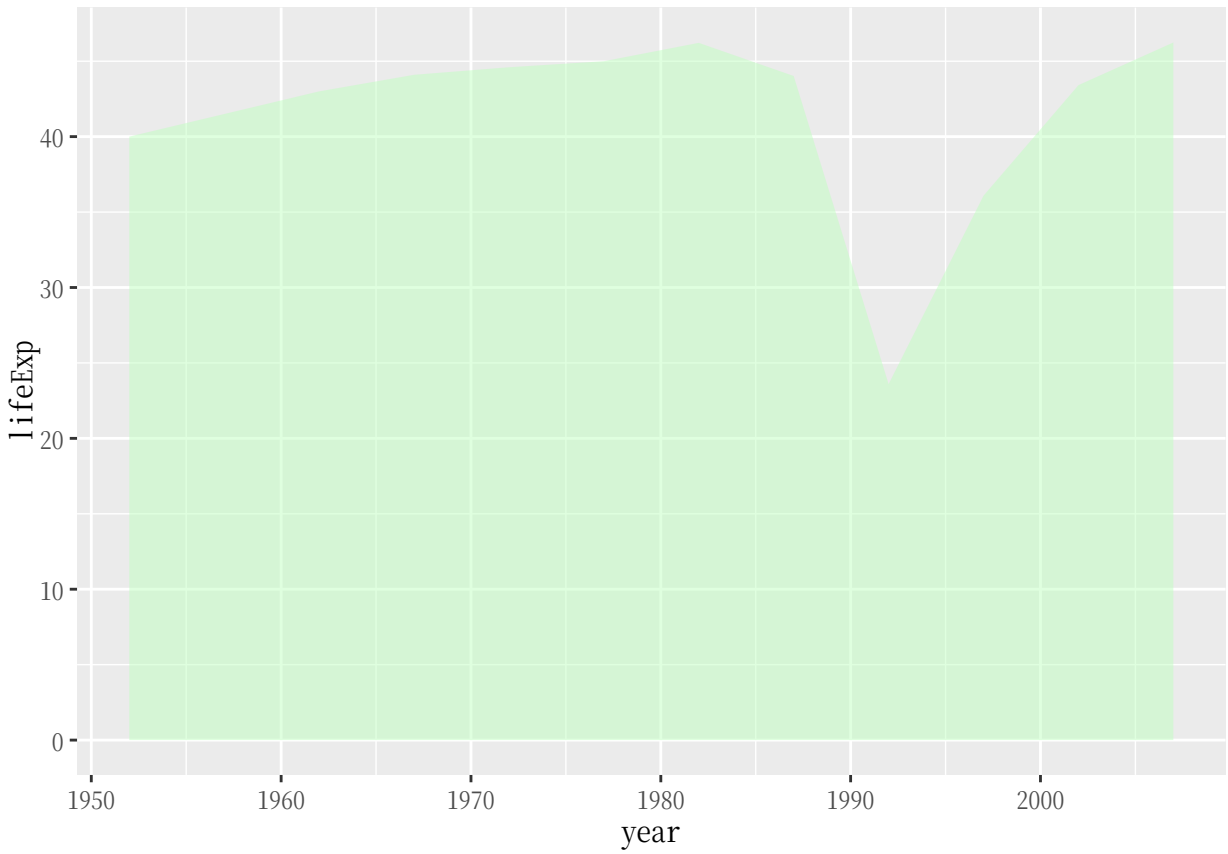
在上面的例子中，通过管道运算符 `|>` 将筛选行子集得到的数据框作为第一自变量传送给了 `ggplot()` 函数，然后就可以继续用 `+` 号添加进一步的作图命令。

如果需要每个点画出散点符号，可以同时使用 `geom_point()`，如：

```
gapminder |>
  filter(country == "Rwanda") |>
  ggplot(aes(
    x = year,
    y = lifeExp)) +
  geom_line() +
  geom_point()
```

用 geom_area() 作类似图形，但在折线下方填充颜色：

```
gapminder |>
  filter(country == "Rwanda") |>
  ggplot(aes(
    x = year,
    y = lifeExp)) +
  geom_area(fill = "darkseagreen1", alpha = 0.5)
```

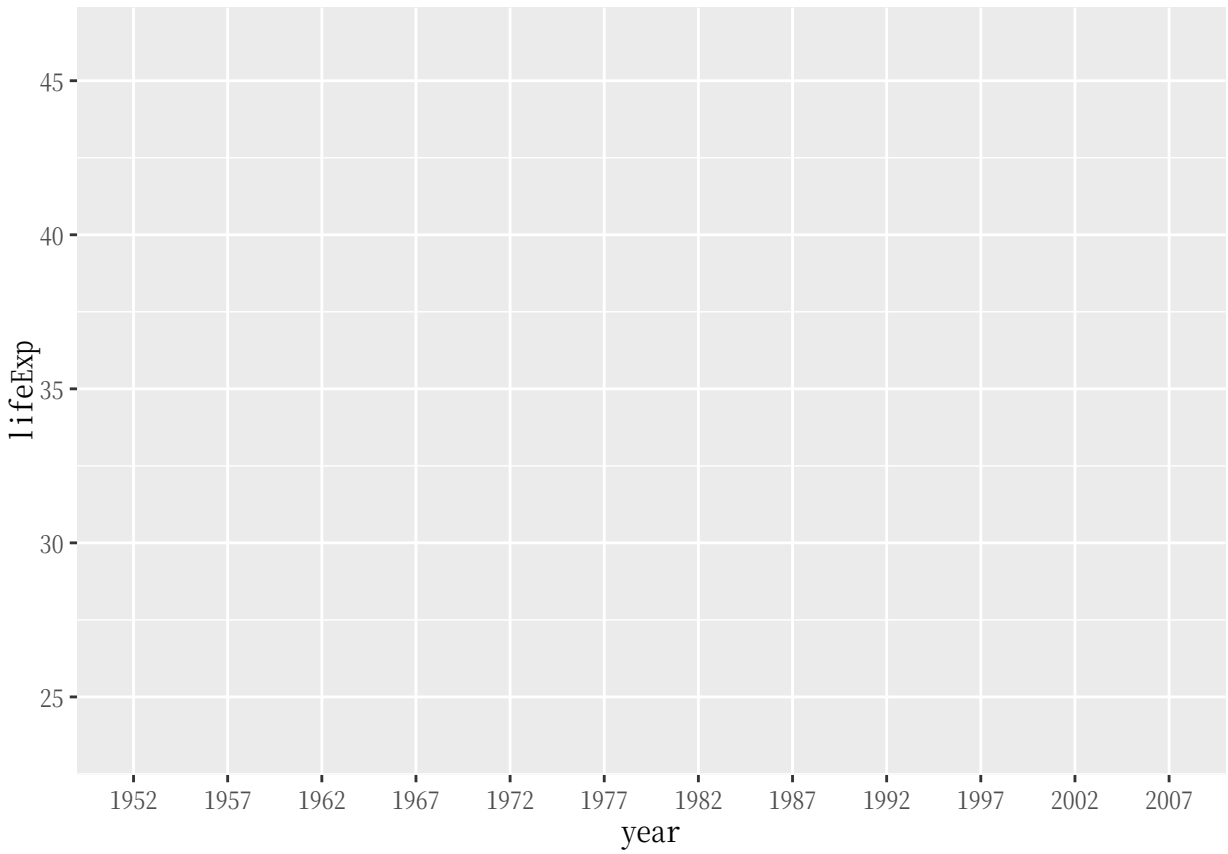


这种图形的纵坐标应该从 0 开始，使得阴影部分的大小与纵坐标值成比例，这也是 ggplot2 的默认做法。

连线图还有一个问题，就是如果 x 坐标不是数值型变量而是因子或者字符型，则两点之间不会相连。比如，将 gapminder 的 Rwanda 子集中的 year 转换成因子，再画折线图：

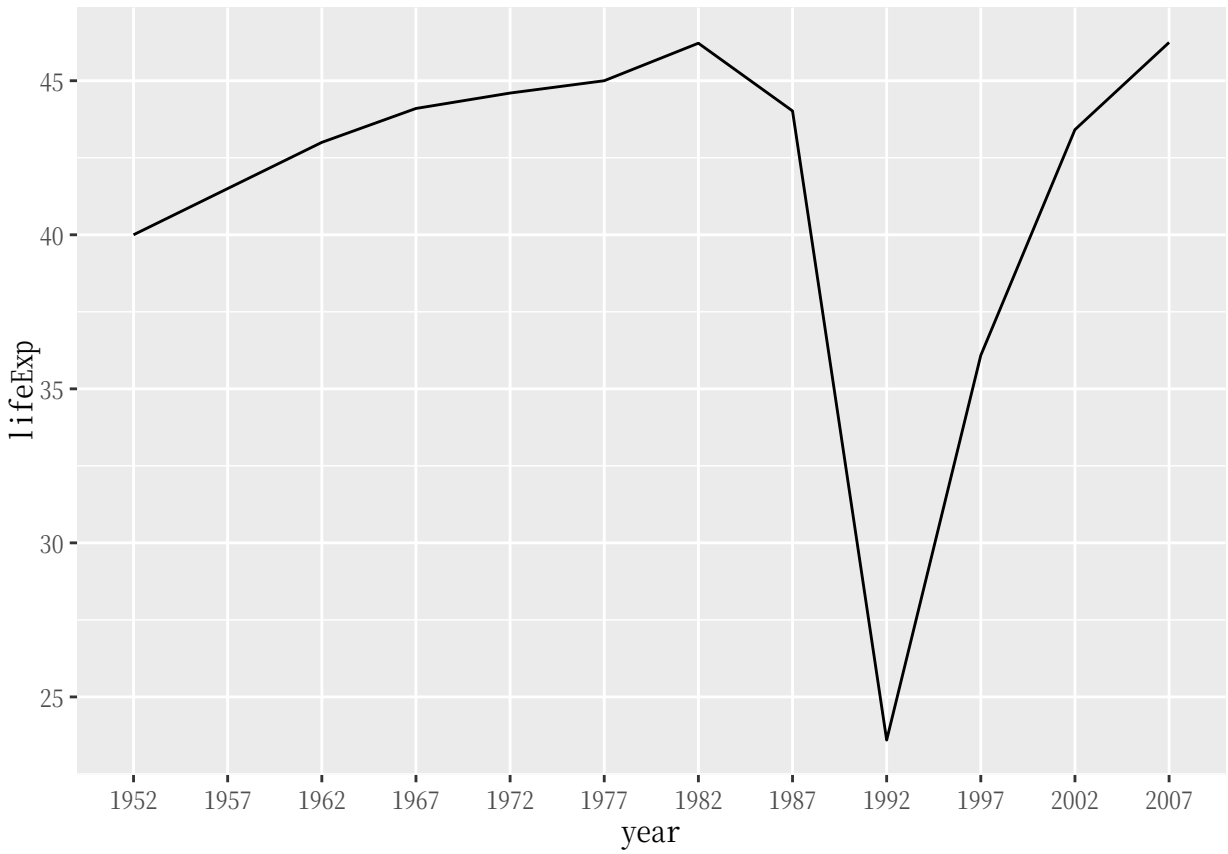
```
gapminder |>
  filter(country == "Rwanda") |>
  mutate(year = factor(
    year, levels = seq(1952, 2007, by=5))) |>
  ggplot(aes(
    x = year,
    y = lifeExp)) +
  geom_line()
```

```
## geom_path: Each group consists of only one observation. Do you need to adjust
## the group aesthetic?
```



没有得到应有的结果。这是因为因子 `year` 起到了分组作用，相当于每个年份为一组，连线只能在组内连，但每组仅有一个观测。这时，显式地指定 `group` 变量可以解决问题：

```
gapminder |>
  filter(country == "Rwanda") |>
  mutate(year = factor(
    year, levels = seq(1952, 2007, by=5))) |>
  ggplot(aes(
    x = year, y = lifeExp, group = country)) +
  geom_line()
```



对于折线图，可以在 `geom_line()` 函数中用 `color` 参数指定颜色，用 `linetype` 参数指定线型，用 `size` 参数指定以毫米为单位的粗细。线型包括：

- 0：不画线；
- 1：实线；
- 2：dashed；
- 3：dotted；
- 4：dotdash；
- 5：longdash；
- 6：twodash。

27.4.1.1 找出期望寿命增长不稳定的国家

我们编程找出寿命增长不稳定的国家。这部分内容与作图关系不大，可以跳过。对每个国家做线性回归，以期望寿命为因变量，以年份为自变量，找出残差方差很大的，以及直线斜率为负的国家。

```
extract_lmr <- function(lmr){
  summ = summary(lmr)
  tibble(
```

```
    sigma = summ$sigma,
    rate = lmr$coefficients[2]
  )
}

lmres <- gapminder |>
  select(country, year, lifeExp) |>
  group_by(country) |>
  nest(data = c(year, lifeExp)) |>
  mutate(lmr = map(
    data,
    ~ lm(lifeExp ~ year, data = .x))) |>
  mutate(info = map(lmr, extract_lmr)) |>
  select(-data, -lmr) |>
  unnest(info) |>
  arrange(desc(sigma))
summary(lmres)
```

```
##         country        sigma             rate
##  Afghanistan:  1   Min.   :0.2118   Min.   :-0.09302
##  Albania    :  1   1st Qu.:0.7111   1st Qu.: 0.20832
##  Algeria    :  1   Median :1.2843   Median : 0.32145
##  Angola     :  1   Mean   :1.6587   Mean   : 0.32590
##  Argentina  :  1   3rd Qu.:1.9272   3rd Qu.: 0.44948
##  Australia  :  1   Max.   :7.2054   Max.   : 0.77218
##  (Other)    :136
```

```
head(lmres, 10)
```

```
## # A tibble: 10 x 3
## # Groups:   country [10]
##    country      sigma    rate
##    <fct>        <dbl>   <dbl>
##  1 Zimbabwe      7.21 -0.0930
##  2 Swaziland     6.64  0.0951
##  3 Rwanda        6.56 -0.0458
##  4 Botswana      6.11  0.0607
##  5 Lesotho       5.93  0.0956
##  6 Cambodia      5.63  0.396
```

```
##  7 Namibia       4.96  0.231
##  8 South Africa  4.74  0.169
##  9 Zambia        4.53 -0.0604
## 10 Kenya         4.38  0.207
```

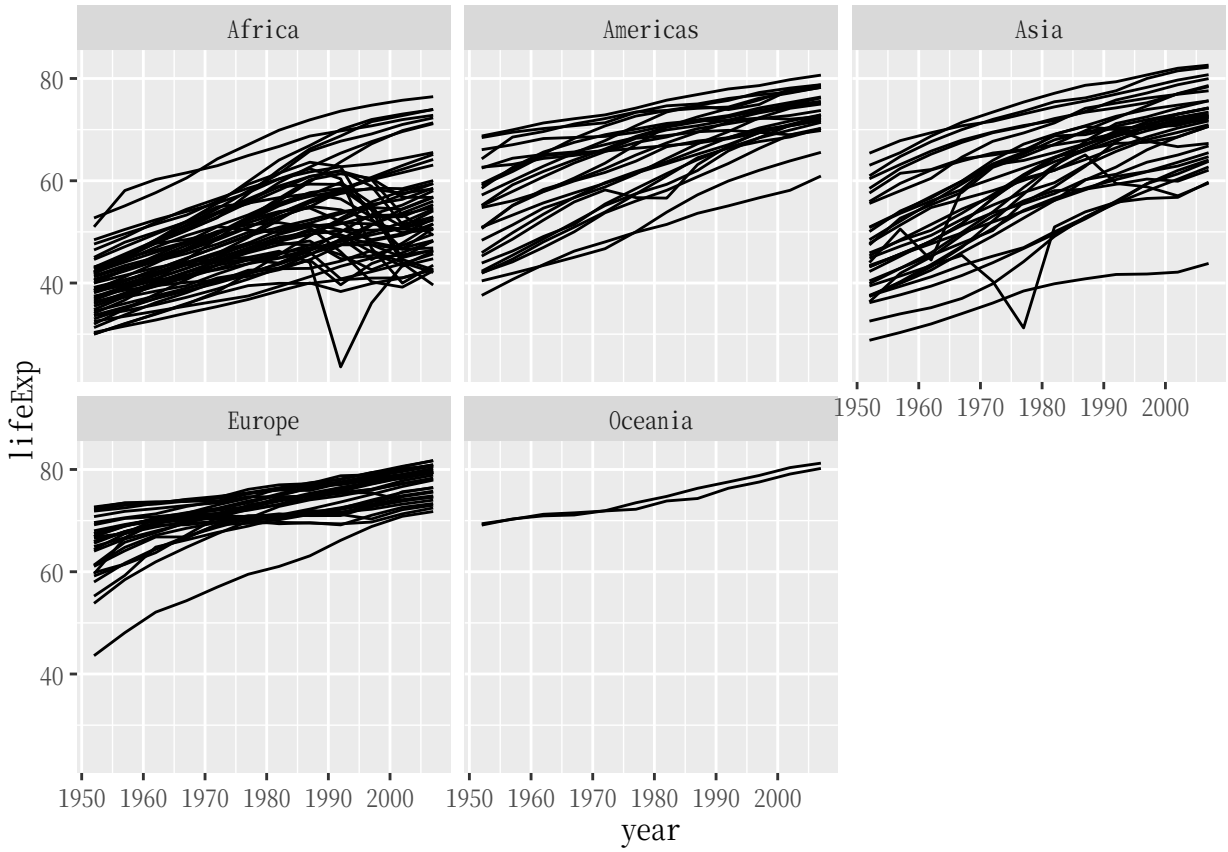
这些排名靠前的国家都是期望寿命增长不稳定的。

27.4.2 切片 (facet)

前面所有国家的图包含了过多的曲线，使得图形表现得很拥挤。可以将一个作图区域拆分成若干个小块，称为切片或者分面（facet），按照某一个或两个分类变量的不同值将数据分为若干个子集，每个数据子集分别在切片上作图。

对于上面的例子，可以将每个大洲的图形分别放置在一个切片上。切片不是一种变量映射，而是一种图形摆放方法，所以不设置在 `aes()` 函数内，而是用 `facet_wrap()` 函数规定。程序如：

```
ggplot(gapminder, aes(
    x = year,
    y = lifeExp,
    group = country)) +
  geom_line() +
  facet_wrap(~ continent)
```



区分不同切片的标签写在每个切片的上方。可以用 `facet_wrap()` 参数 `strip_position` 和参数 `switch` 调整标签的上下左右。

切片之间默认公用了横坐标和纵坐标且坐标范围保持一致。如果不保持一致，读者可能会有误解。但是 x 轴或 y 轴映射为分类变量且不同切片的分类完全不同时，可以令各切片中该轴的取值不统一。`facet_wrap()` 选项 `scales` 默认为`"fixed"`，即所有切片的 x 轴、y 轴都范围一致，取`"free_x"` 则允许各切片的 x 轴不统一，`"free_y"` 允许各切片的 y 轴不统一，`"free"` 允许各切片的 x 轴和 y 轴都不统一。

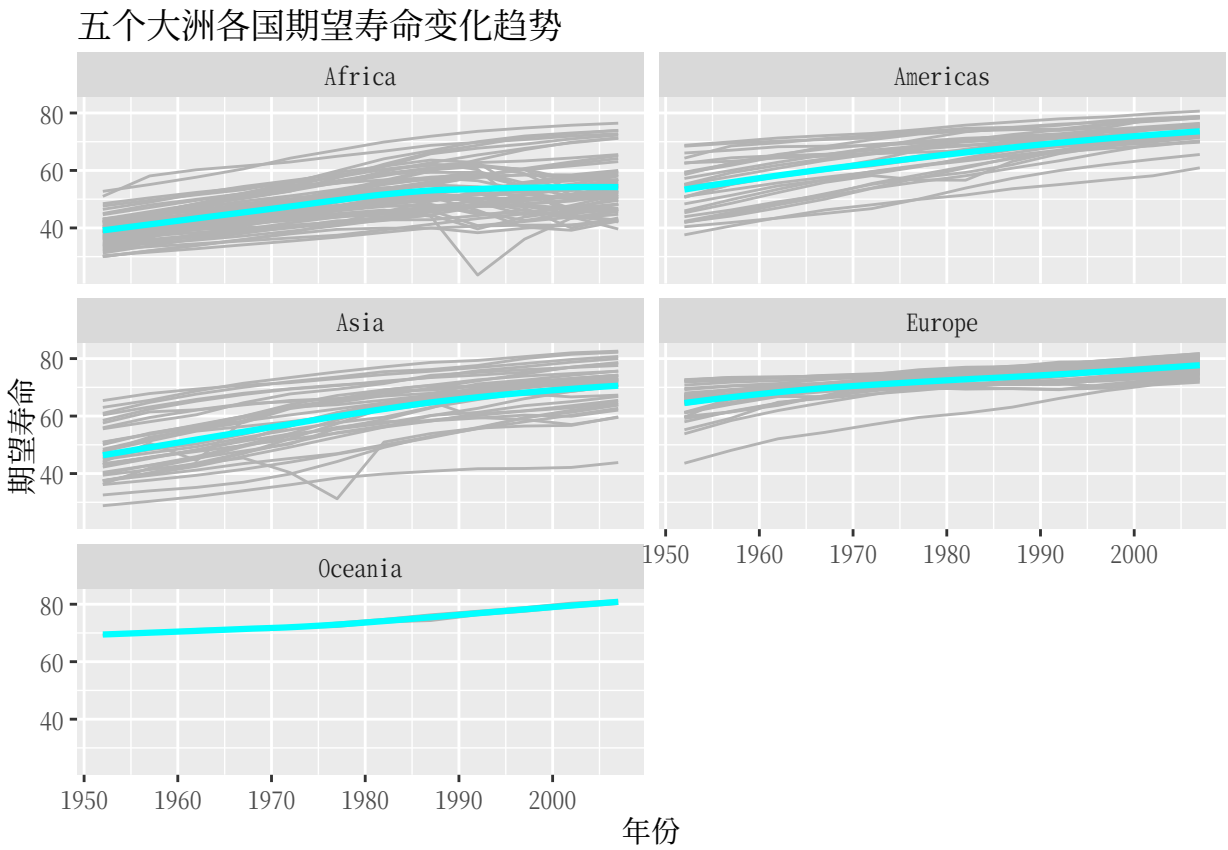
在 `facet_wrap()` 中可以用 `ncol` 参数指定切片的列数，用 `nrow` 指定切片的行数。各个切片的次序应该设定为一定的合理次序，比如用来分类的变量本身有序，或者令各切片中的数据值有一定的增减次序。

下面的程序将曲线颜色变浅，对每个大洲增加了拟合曲线，增加了适当的标题和坐标轴标签。注意，这时不能使用统一的 `group = country` 映射，否则拟合曲线就是对每个国家都单独有一条拟合曲线，而不是每幅切片中仅有一条拟合曲线。办法是仅在 `geom_line()` 中给出 `group = country` 的映射，但在 `geom_smooth()` 中则不用 `group` 维。程序如下：

```
ggplot(gapminder, aes(
    x = year,
    y = lifeExp)) +
  geom_line(
    mapping = aes(group = country),
```

```
    color = "gray70") +
  geom_smooth(
    method = "loess",
    color="cyan",
    se = FALSE,
    size = 1.1) +
  facet_wrap(~ continent, ncol = 2) +
  labs(
    x = " 年份",
    y = " 期望寿命",
    title = " 五个大洲各国期望寿命变化趋势")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



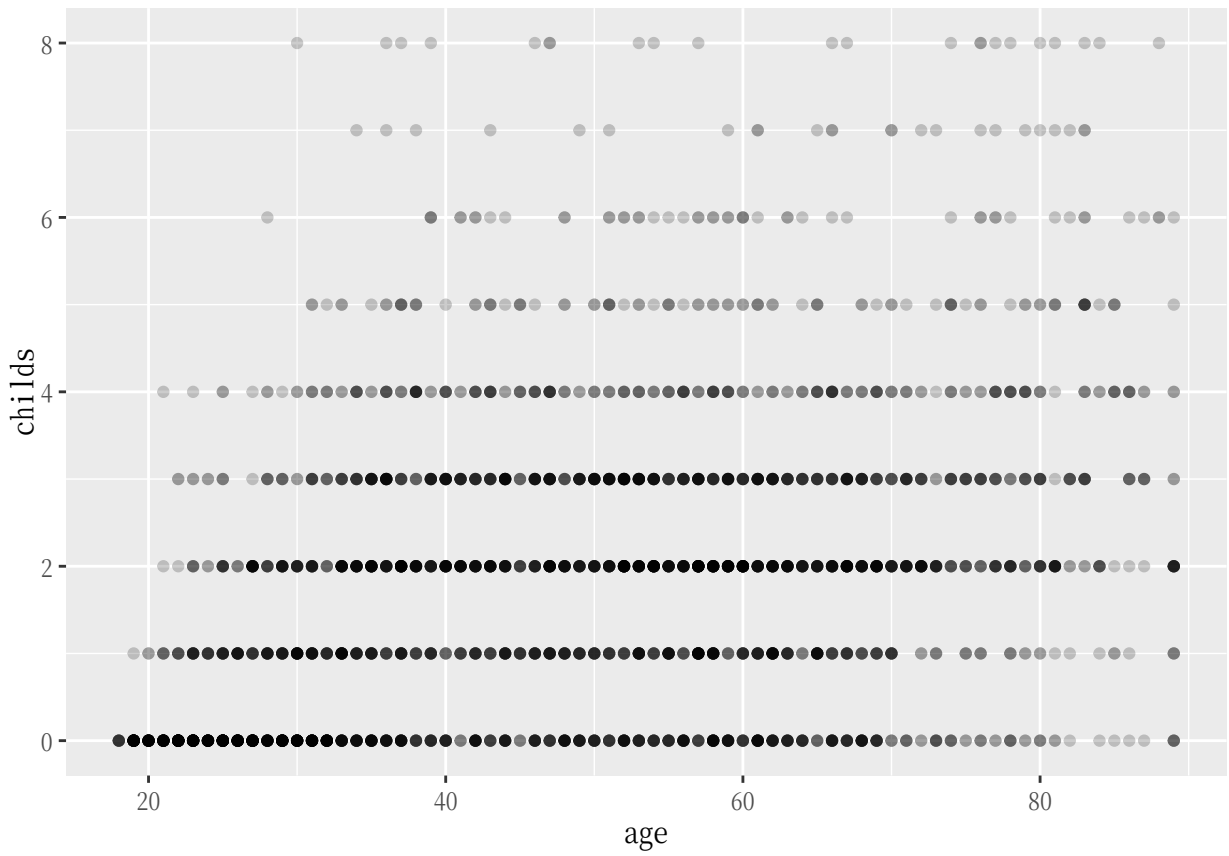
注意 `group = country` 的设置从 `ggplot()` 函数中转移到了 `geom_line()` 函数中，否则就意味着拟合线也需要按照国家分组，而不是按大洲分组。

`facet_wrap()` 主要适用于按照一个分类变量的值将不同观测在不同切片中表现，可以人为指定切片的行数和列数。如果需要按照两个分类变量交叉分组分配切片，可以用 `facet_grid()` 函数。

例如，对 `gss_sm` 数据集，作小孩个数对年龄的散点图：

```
p <- ggplot(gss_sm, aes(
    x = age,
    y = childs))
p + geom_point(alpha = 0.2)
```

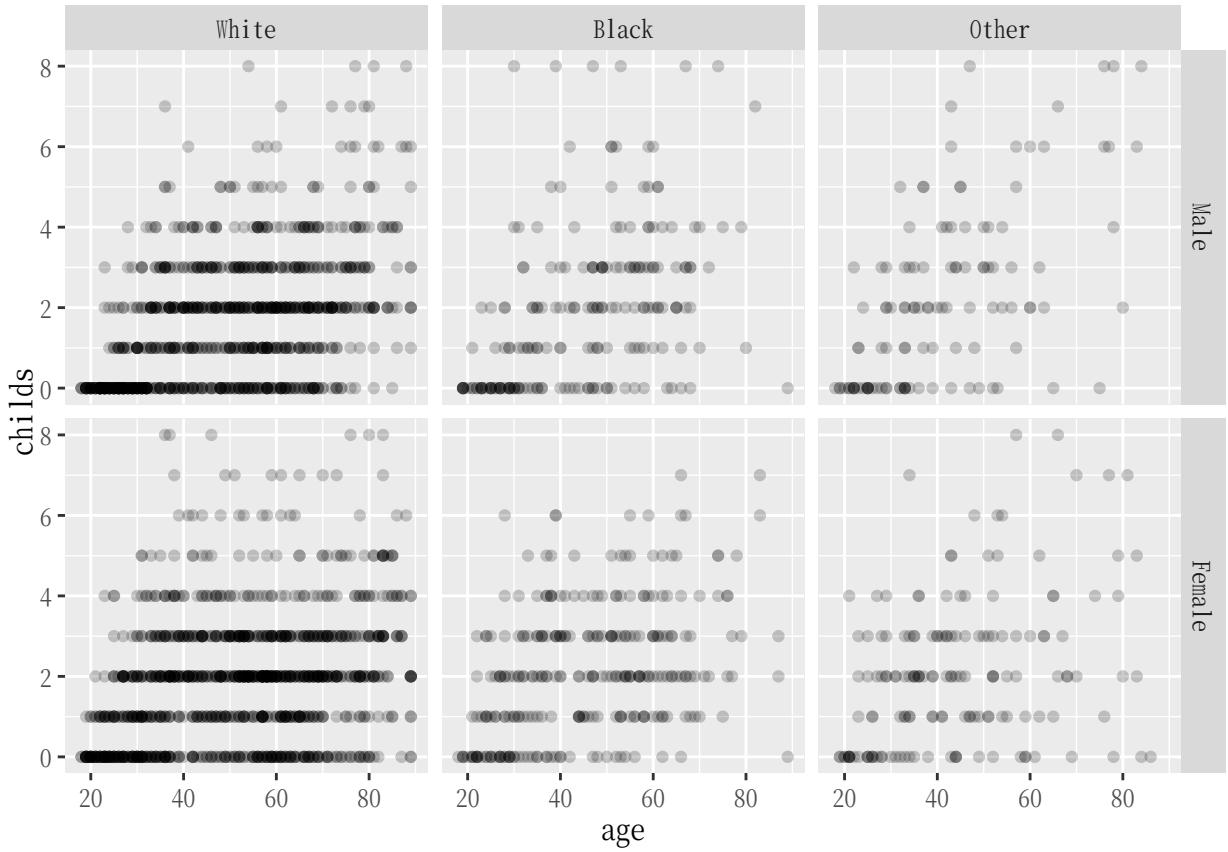
```
## Warning: Removed 18 rows containing missing values (geom_point).
```



有过多的重叠点。将观测按照性别 (sex) 和种族 (race) 交叉分组，分配到不同的切片上：

```
p + geom_point(alpha = 0.2) +
  facet_grid(sex ~ race)
```

```
## Warning: Removed 18 rows containing missing values (geom_point).
```



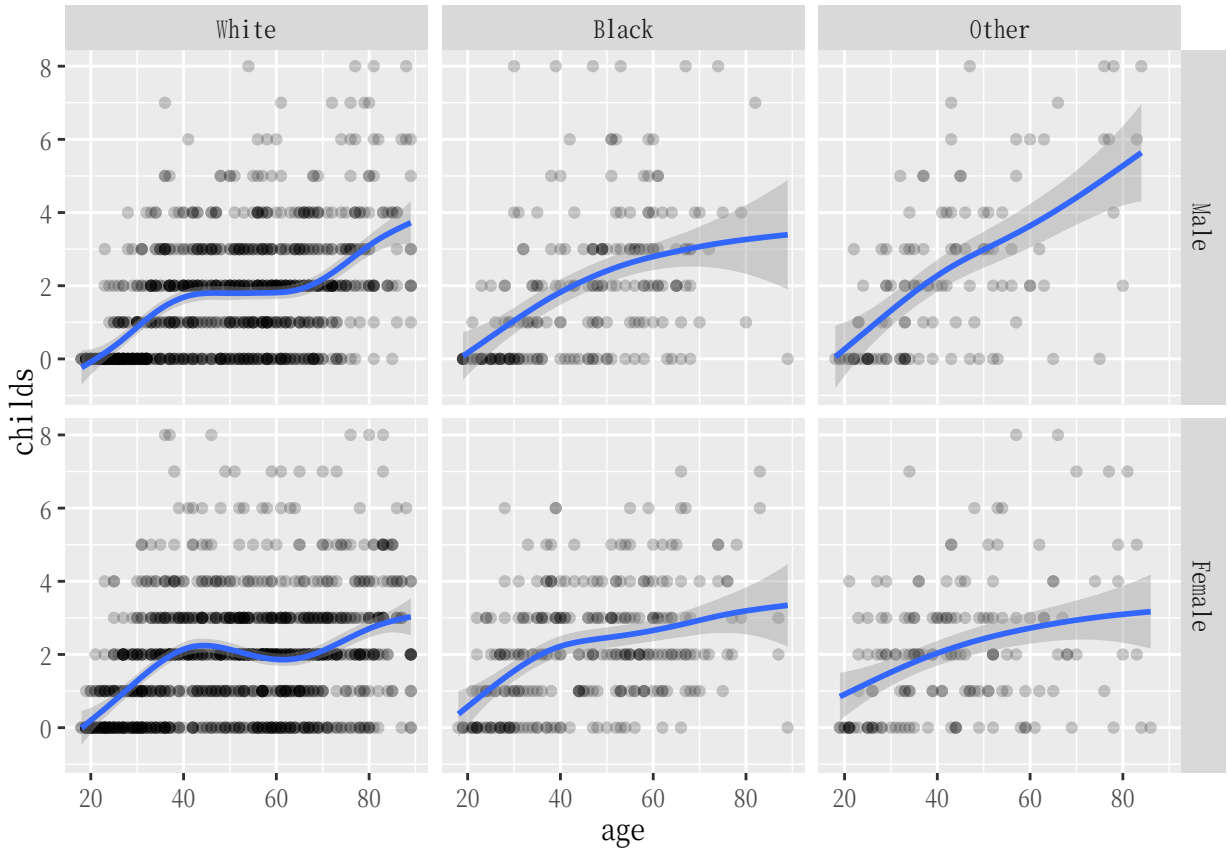
交叉分组时作切片时，`sex ~ race` 这种写法使得不同性别对应到不同行，不同种族对应到不同列。在图形中增加拟合曲线：

```
p + geom_point(alpha = 0.2) +
  geom_smooth() +
  facet_grid(sex ~ race)
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

```
## Warning: Removed 18 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 18 rows containing missing values (geom_point).
```



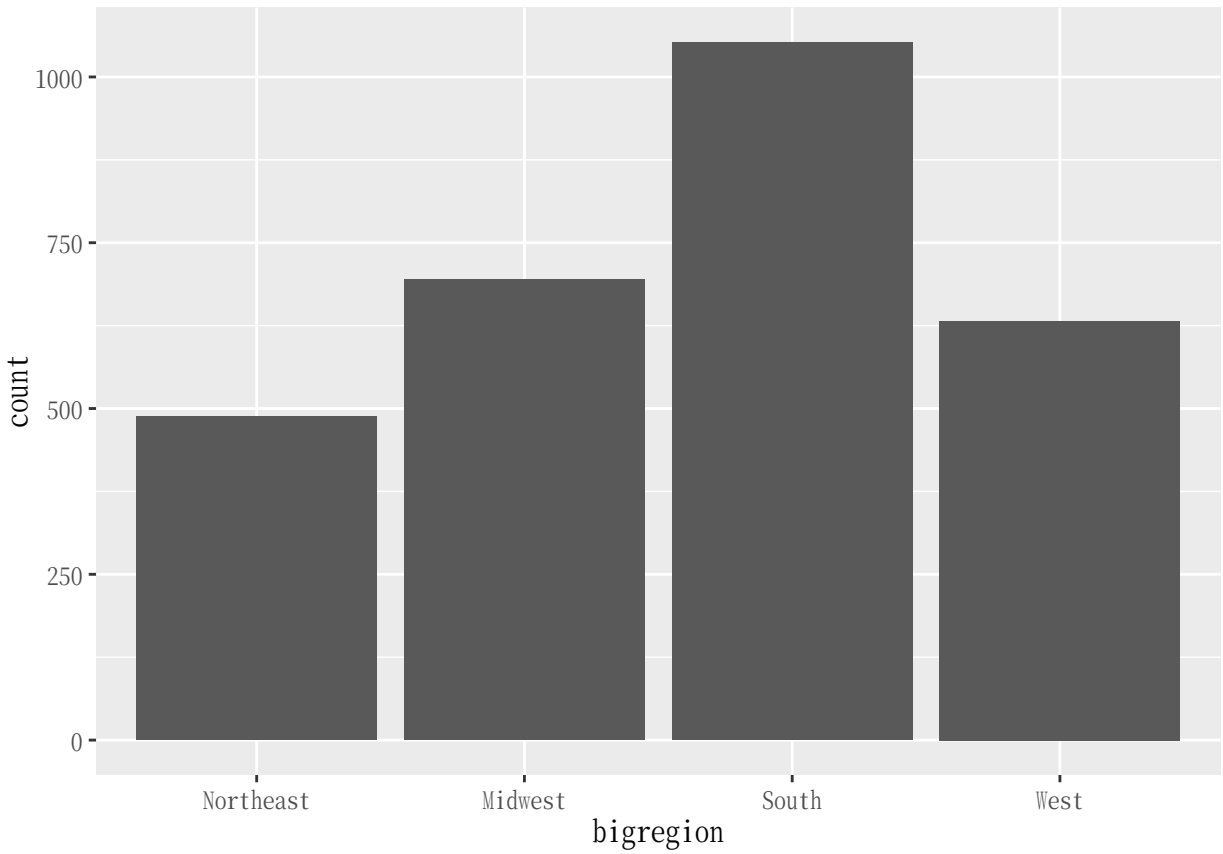
这里虽然没有映射 group 维，但还是按性别和种族对数据集分成了 6 个子集，每个切片中仅有一个自己的数据。

27.5 数据变换与条形图

有些 geom_xxx() 函数直接按照数据值作图，如 geom_point()、geom_line()，而 geom_smooth() 这样的函数则会按照某种算法计算并对计算结果作图。geom_xxx() 都有默认的 stat_xxx() 函数用来计算，也可以人为指定不同的统计规则。

考虑条形图的例子。ggplot2 中的条形图函数 geom_bar() 可以对一个分类变量自动统计频数，并作频数条形图。比如对 gss_sm 数据集的 bigregion 变量作频数条形图：

```
ggplot(gss_sm, aes(
  x = bigregion)) +
  geom_bar()
```

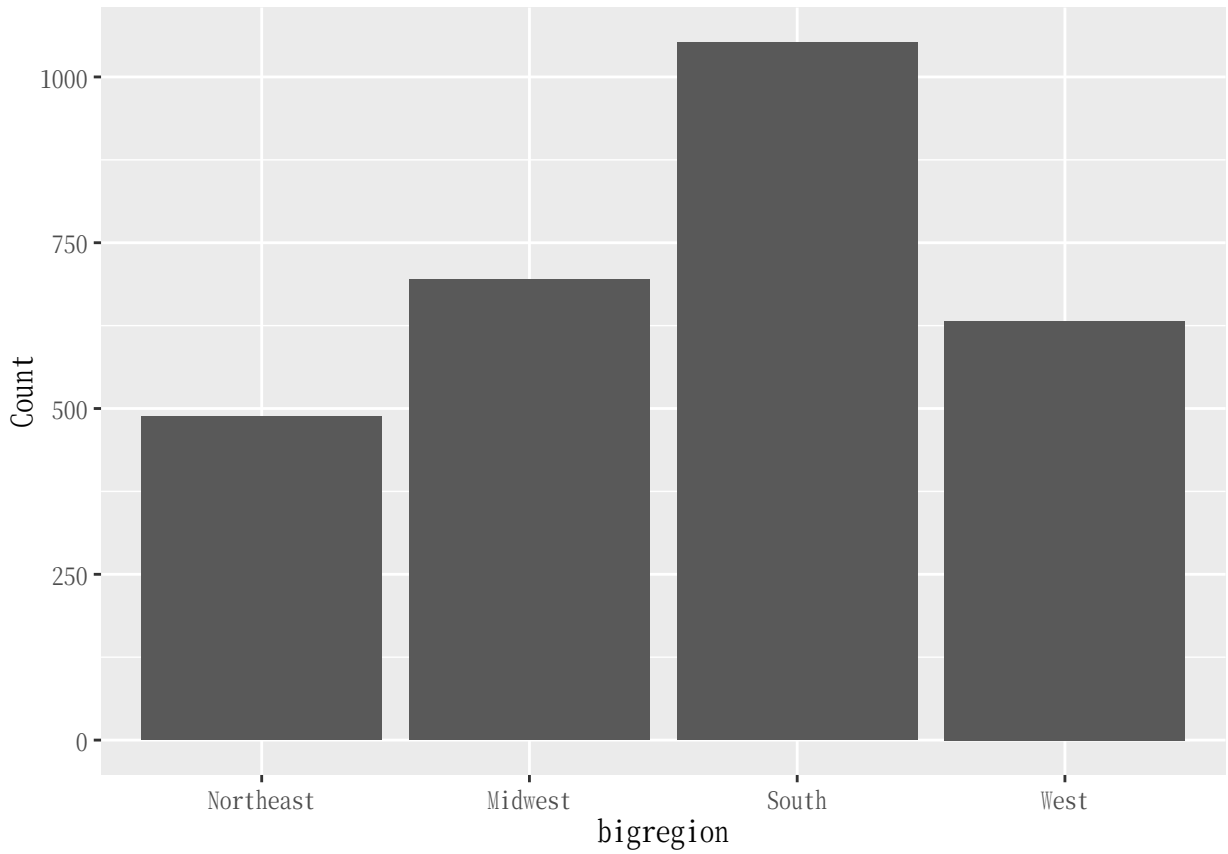


结果是每个大区的受访者人数的条形图。图形中 `x` 映射是用户指定的，而 `y` 轴则是自动计算的频数。实际上，`geom_bar()` 自动调用了统计函数 `stat_freq()` 对每个大区计算频数，生成新变量 `count` 和 `prop`。`geom_bar()` 默认使用 `count`(频数)。

虽然 ggplot2 能够自动统计频数，但最好还是预先统计好频数，仅用 ggplot2 绘图，这对做分段、并列条形图尤其有用。所以，上例可以用 tidyverse 的 `count` 和 ggplot2 的 `geom_col` 改写成：

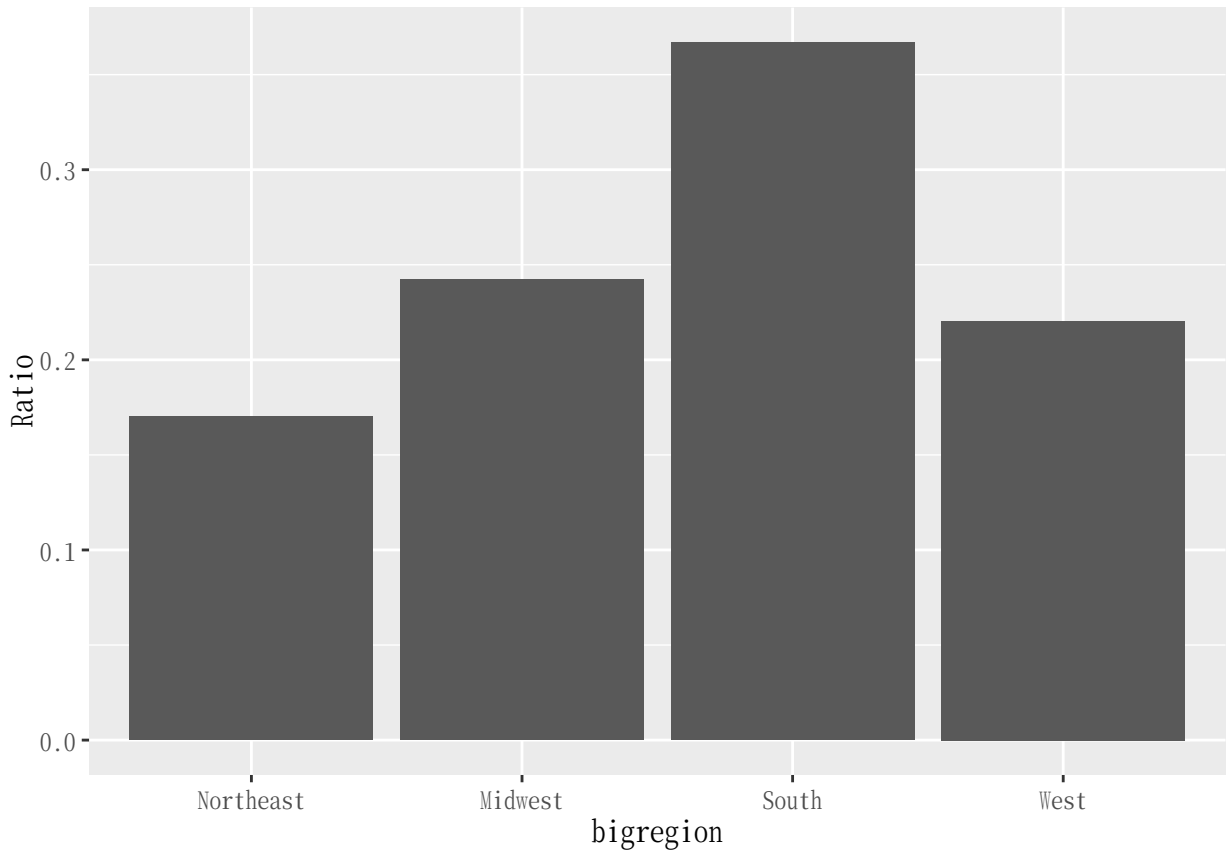
```
df1 <- gss_sm |>
  select(bigregion) |>
  count(bigregion) |>
  mutate(ratio = n / sum(n))

ggplot(df1, aes(
  x = bigregion,
  y = n)) +
  geom_col() +
  labs(y = "Count")
```



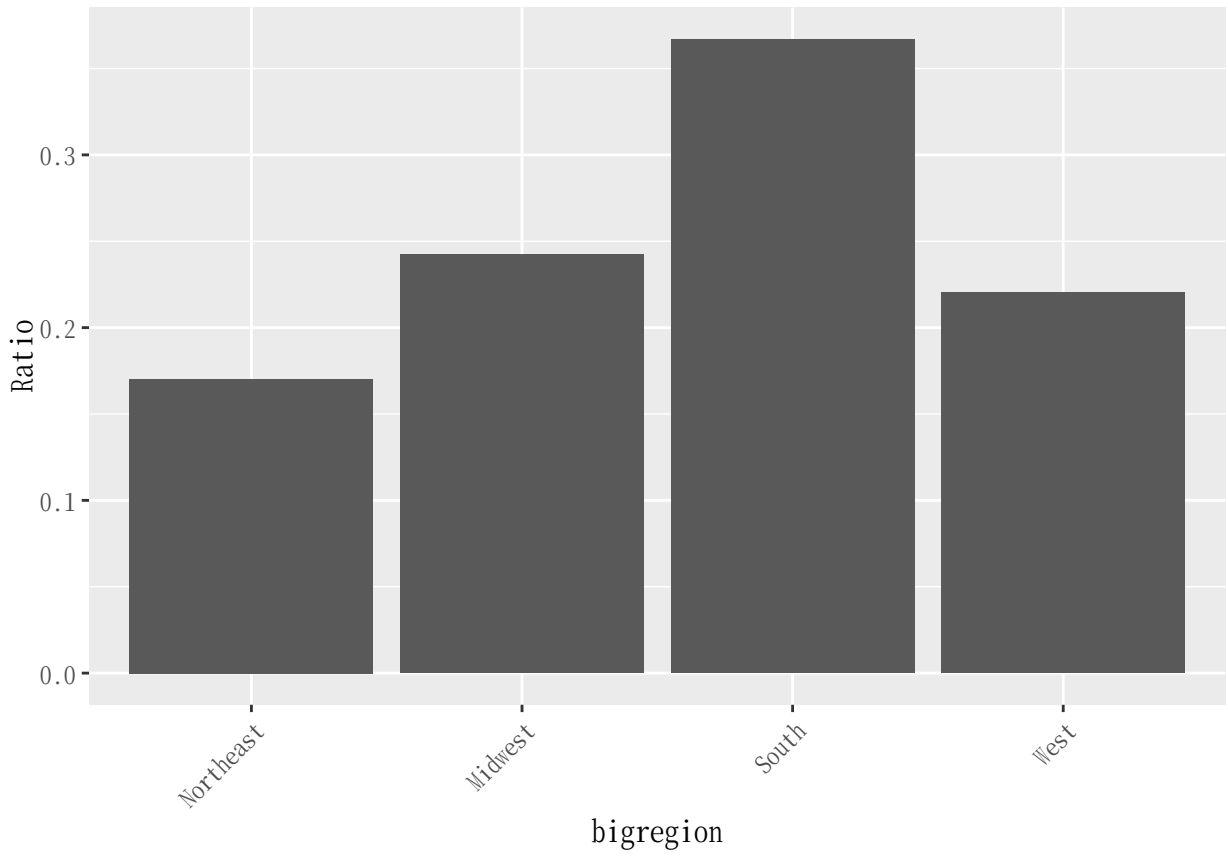
下面的程序将纵坐标改成了比例：

```
ggplot(df1, aes(
  x = bigregion,
  y = ratio)) +
  geom_col() +
  labs(y = "Ratio")
```



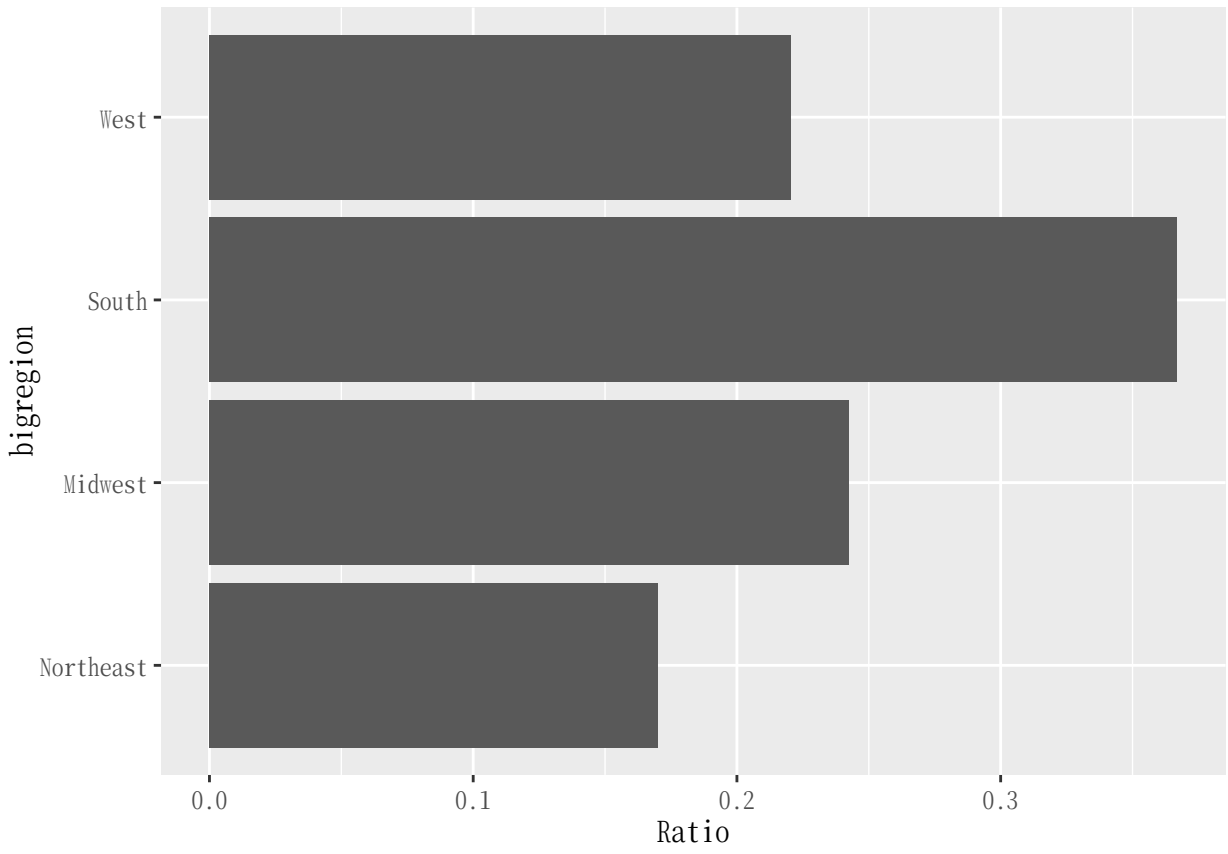
当 x 轴的分组较多时，区分各组的文字标签可能会重叠。为此，可以将文字标签旋转 45 度角：

```
ggplot(df1, aes(
  x = bigregion,
  y = ratio)) +
  geom_col() +
  labs(y = "Ratio") +
  theme(
    axis.text.x = element_text(
      angle = 45, vjust = 1, hjust = 1)  )
```



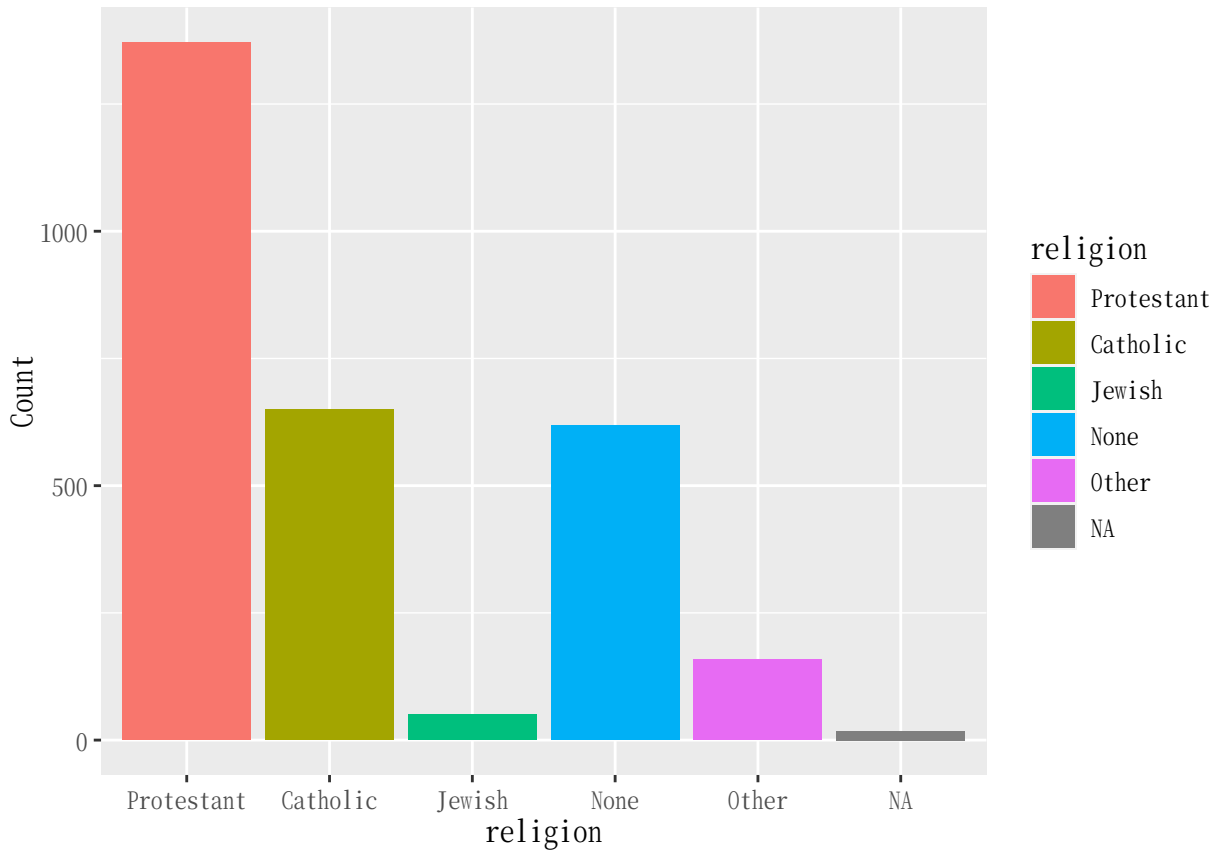
解决条形图类别标签重叠的另一常用做法是将纵向条形图改为横向条形图，ggplot2 的 `coord_flip()` 函数可以将 x 轴和 y 轴互换，这样类别标签就变到了纵轴，仍然横向显示，互相之间就不再重叠，如：

```
ggplot(df1, aes(
  x = bigregion,
  y = ratio)) +
  geom_col() +
  labs(y = "Ratio") +
  coord_flip()
```



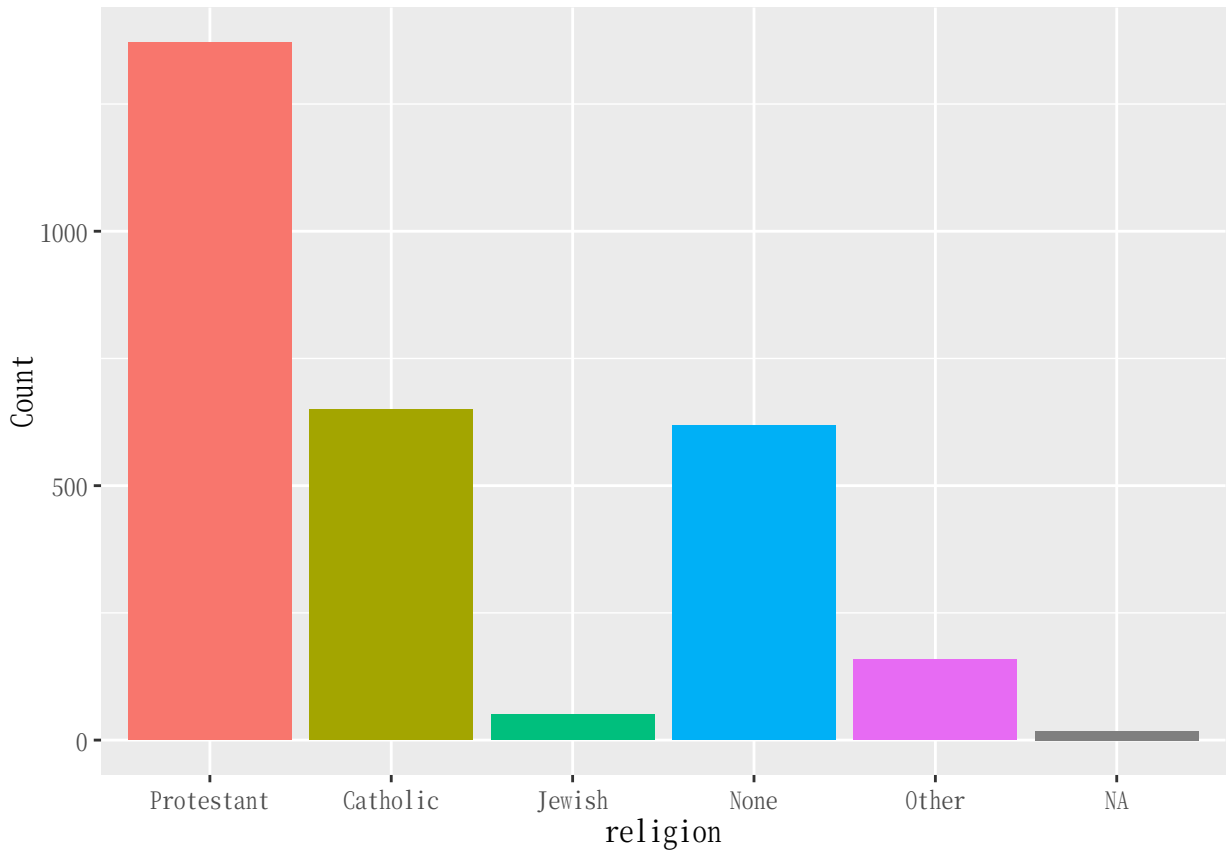
有时人们愿意看到色彩更丰富的图形。下面的例子作 `gss_sm` 数据集中 `religion` 变量的频数条形图，并给不同的条形自动分配不同的颜色，方法是指定 `fill = religion`：

```
df2 <- gss_sm |>
  select(religion) |>
  count(religion)
p <- ggplot(df2, aes(
    x = religion,
    y = n,
    fill = religion))
p + geom_col() +
  labs(y = "Count")
```



因为将 `religion` 同时映射到 `x` 维与 `fill` 维，所以对应 `fill` 维在图形右侧出现了图例，这是多余的。调用 `guides(fill = "none")` 可以人为指定不做关于填充色的图例：

```
p + geom_col() +
  guides(fill = "none") +
  labs(y = "Count")
```



从可视化理论的角度看，上图中的不同颜色是多余的，用同一颜色更能强调数据本身。

27.5.1 分段与并列条形图

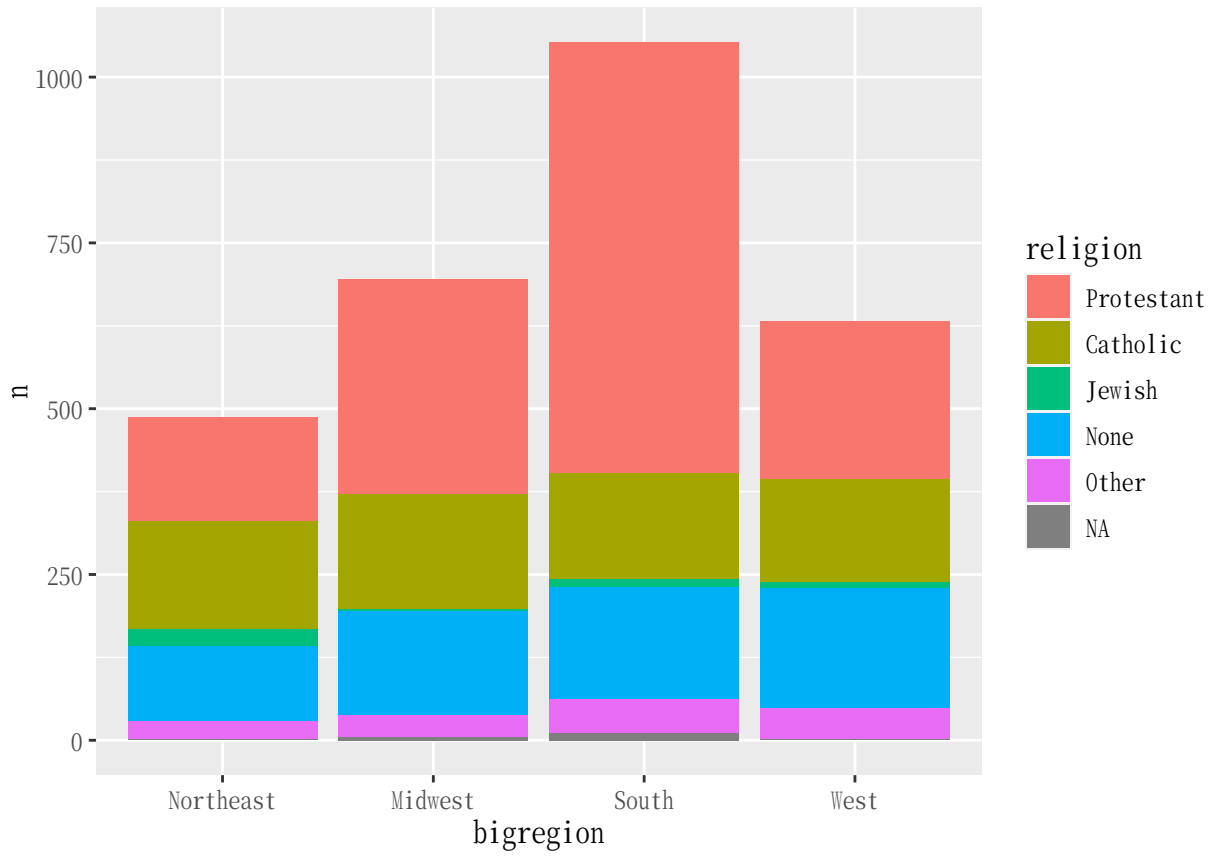
上面的条形图展现了单个分类变量的频数分布。两个分类变量的交叉频数分布可以用分段条形图或者并列条形图表现。

例如，对 `gss_sm` 数据集，按照 bigregion 分组计算频数，每组内再按照 religion 计算频数，并且计算每种 religion 在 bigregion 内部的比例，用 tidyverse 统计后作图如下：

```
df3 <- gss_sm |>
  select(bigregion, religion) |>
  group_by(bigregion, religion) |>
  summarise(n = n(), .groups="drop_last") |>
  mutate(ratio = n / sum(n))

p <- ggplot(df3, aes(
    x = bigregion,
    y = n,
```

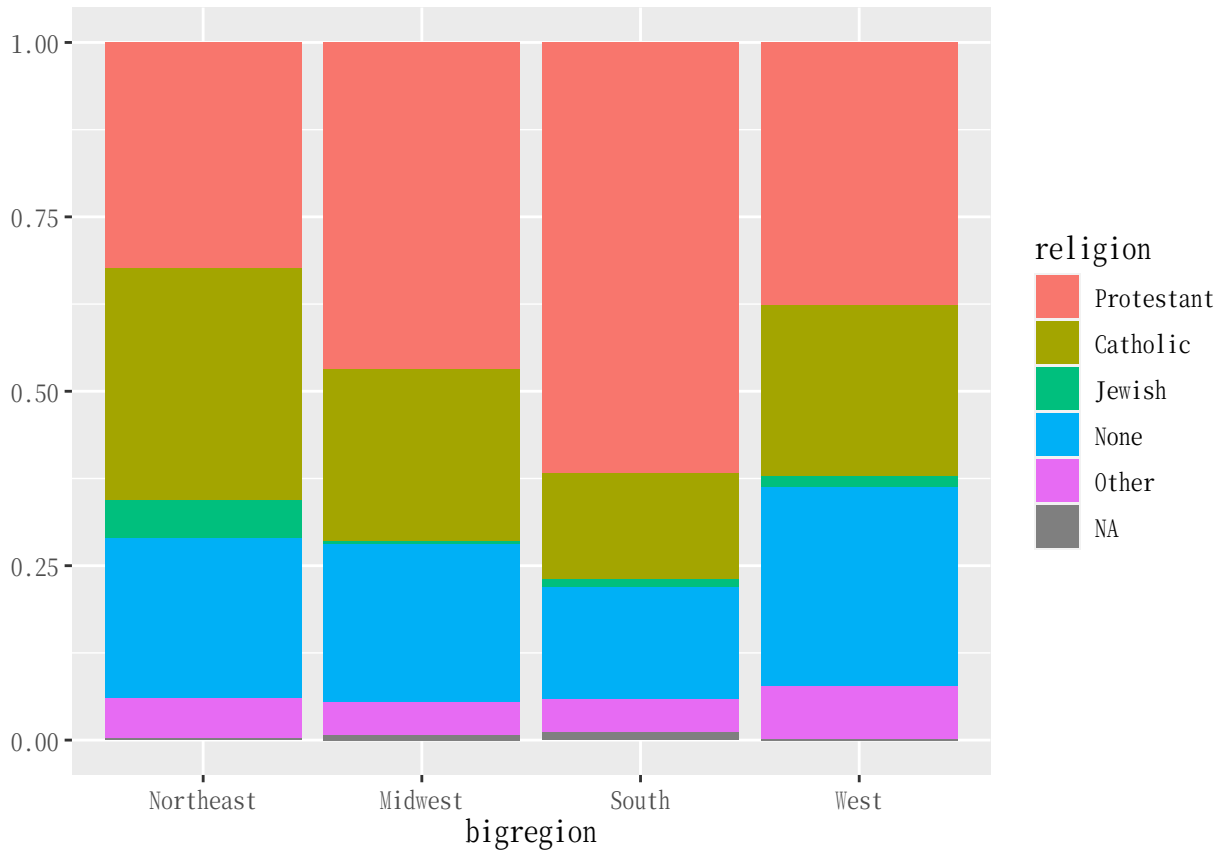
```
    fill = religion))
p + geom_col()
```



这样的图形可以很容易地比较大类 (这里是 bigregion) 的频数比例，但大类内的小类 (这里是 religion) 可以比较容易地在大类内部比较，但是在大类之间比较则较困难。如果仅有两个小类，则小类在大类之间的比较也没有问题。

另一种做法是将大类的高度拉平，图形仅表示每一大类内部小类的比例，没有大类频数信息，也不能比较两个大类之间的小类频数，但可以大致地在大类之间比较小类的比例：

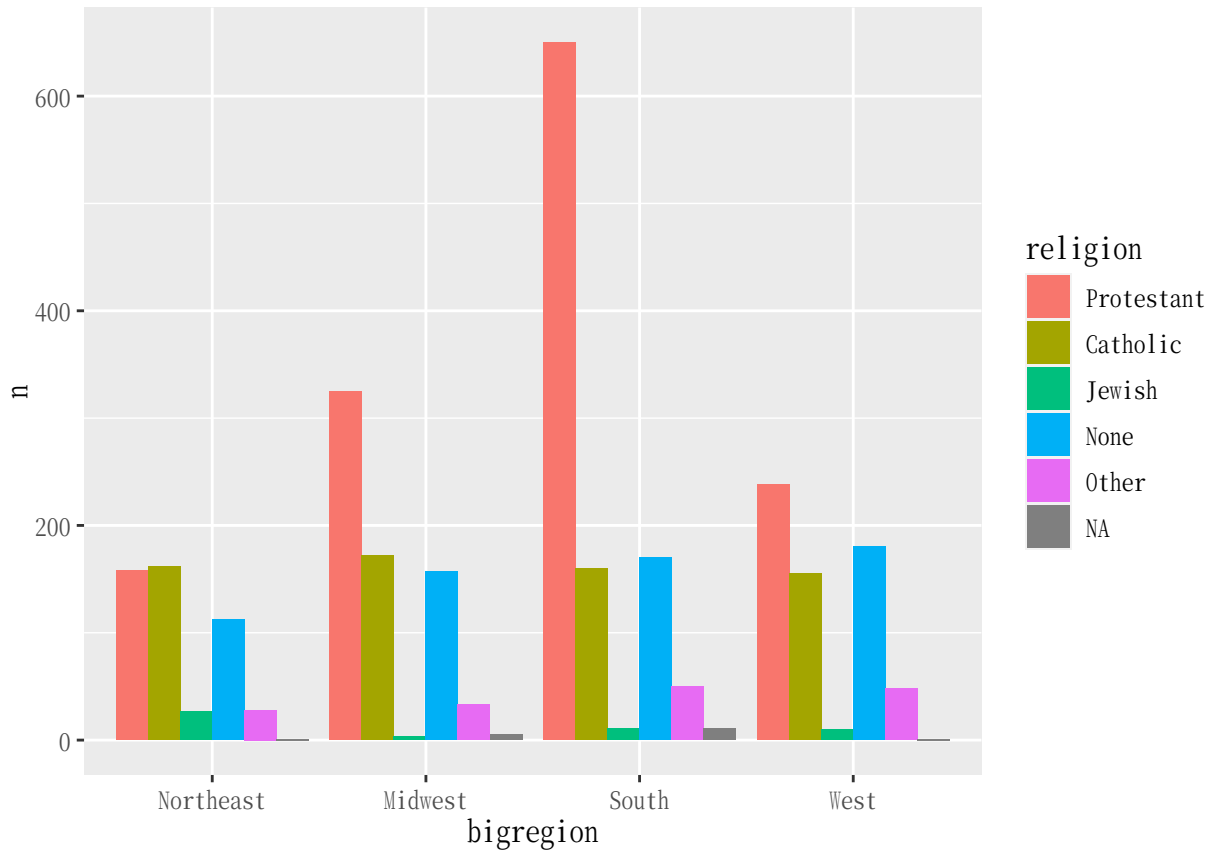
```
p + geom_col(position = "fill") +
  labs(y = NULL)
```



上面的程序在 `geom_col()` 中用了 `position = "fill"` 选项。

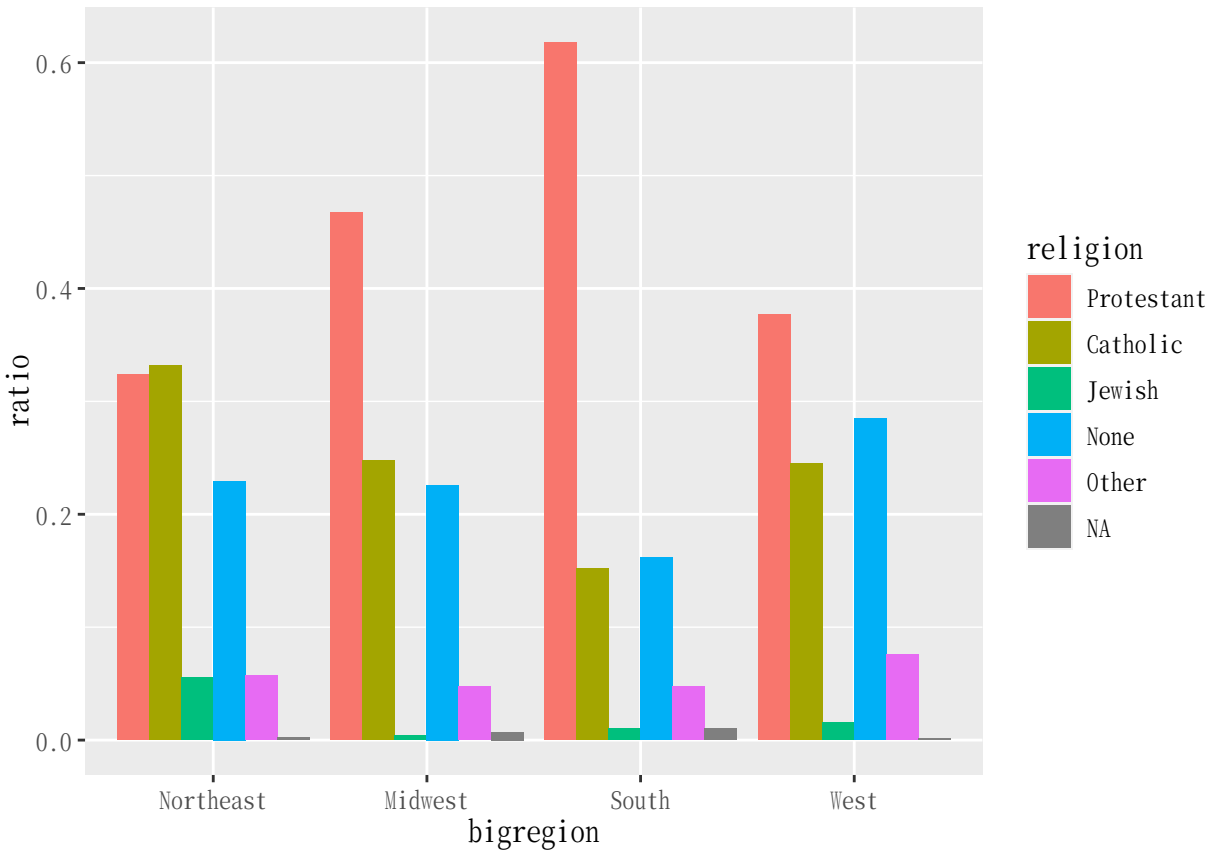
并排的条形图可以表现每个交叉类的频数，可以比较容易地比较每个大类内部的小类比例以及小类的频数，但是不容易比较大类的比例：

```
p + geom_col(position = "dodge")
```



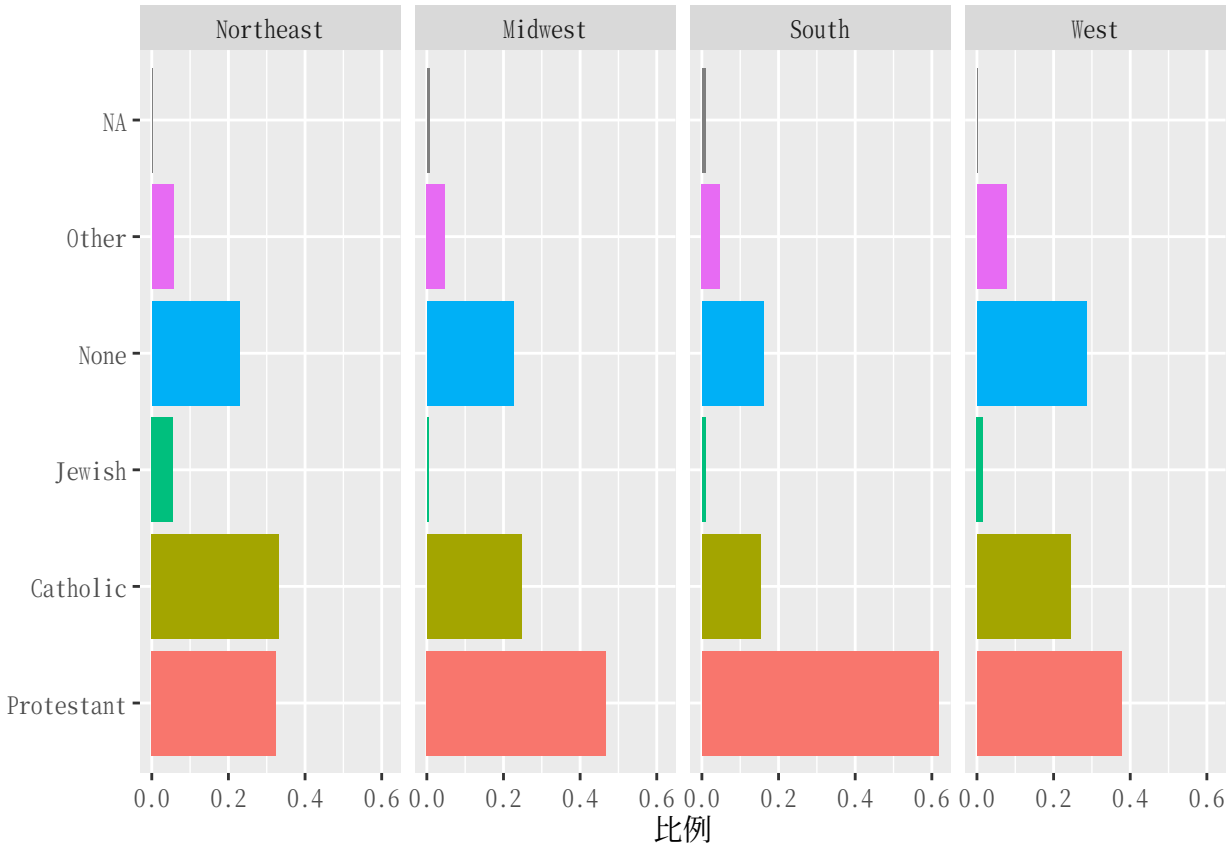
将上图中的纵轴改为大类内的比例（每个大区的比例之和等于 1）：

```
ggplot(df3, aes(
    x = bigregion,
    y = ratio,
    fill = religion)) +
  geom_col(position = "dodge")
```



为了在不同大区之间比较宗教比例分布，可以借助于切片，将每个大区分配到一个切片：

```
ggplot(df3, aes(
  x = religion, y = ratio, fill=religion)) +
  geom_col(position="dodge") +
  labs(x=NULL, y=" 比例") +
  coord_flip() +
  facet_grid(~ bigregion) +
  guides(fill="none")
```



有时用来绘图的数据已经是一个频数表，比如泰坦尼克号乘客生存与性别的频数表：

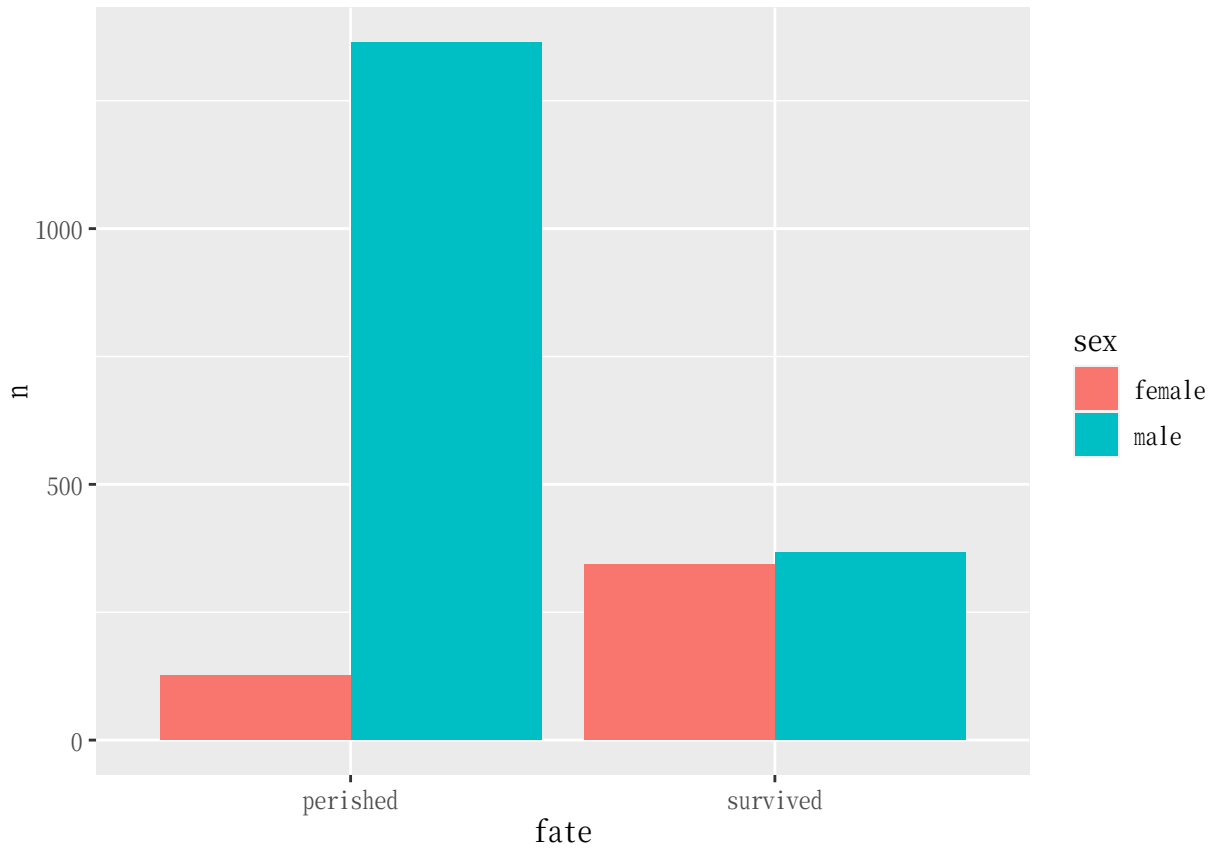
```
titanic
```

```
##       fate    sex    n percent
## 1 perished   male 1364    62.0
## 2 perished female  126     5.7
## 3 survived   male  367    16.7
## 4 survived female  344    15.6
```

这是一个长表格式的列联表，对于 `table()` 生成的列联表可以用 `as.data.frame` 将其转换为长表格式。

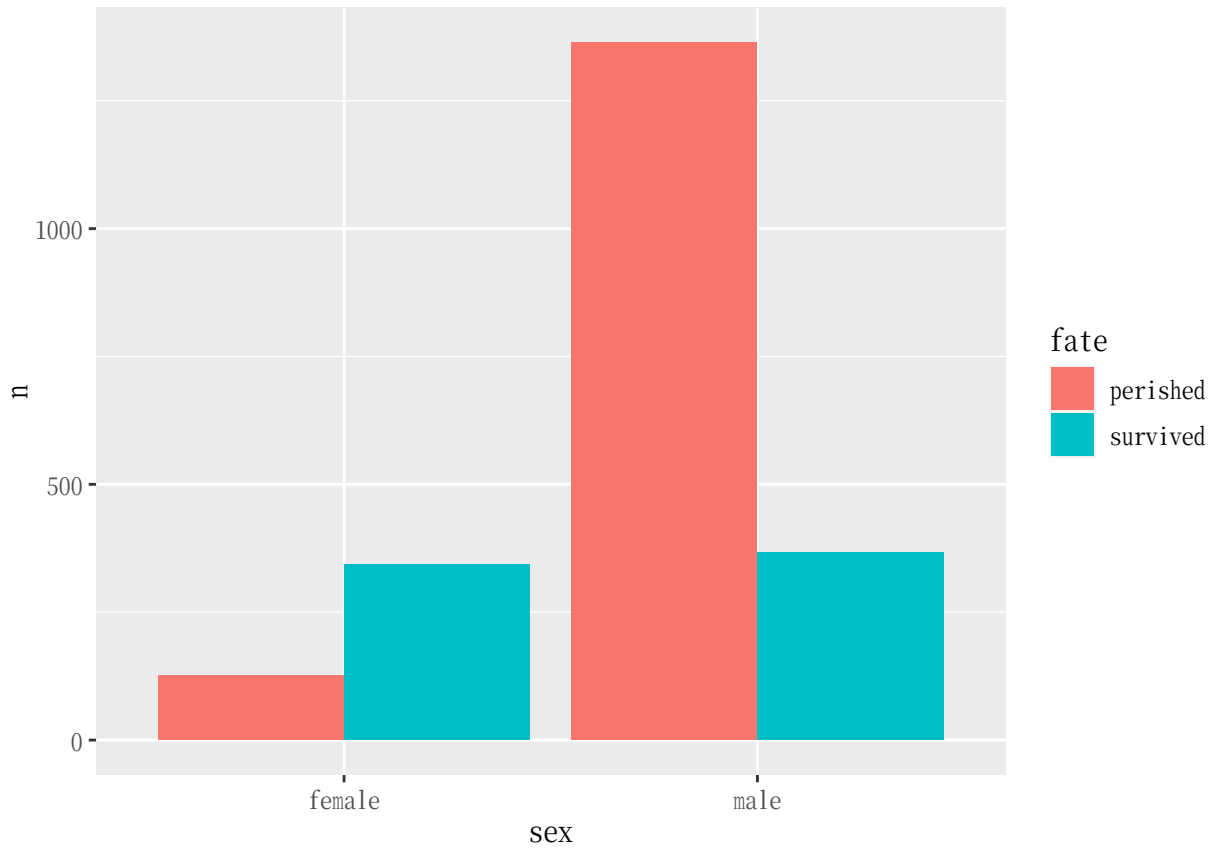
作 fate 为大组的 sex 的并排条形图：

```
ggplot(titanic, aes(
  x = fate, y = n, fill = sex)) +
  geom_col(position = "dodge")
```



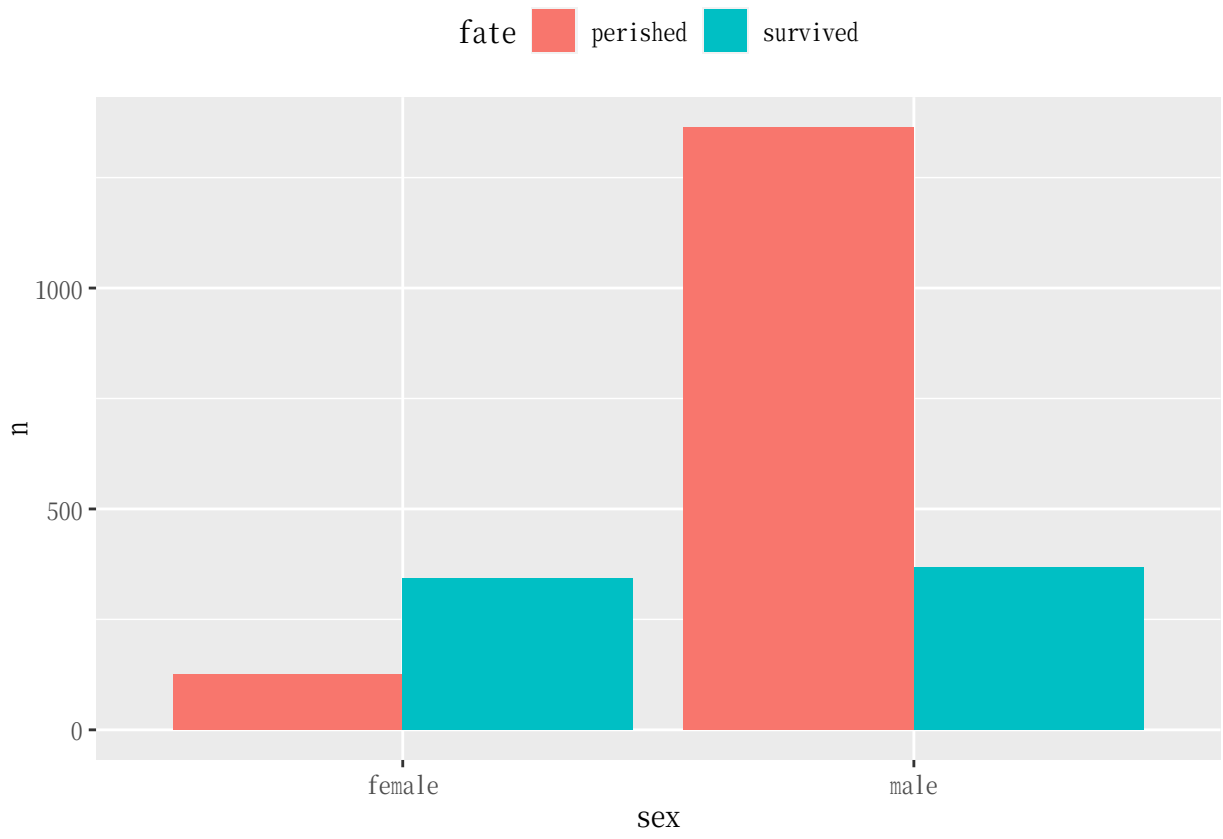
也可以按照性别分成大组：

```
p <- ggplot(titanic, aes(
  x = sex, y = n, fill = fate))
p + geom_col(position = "dodge")
```



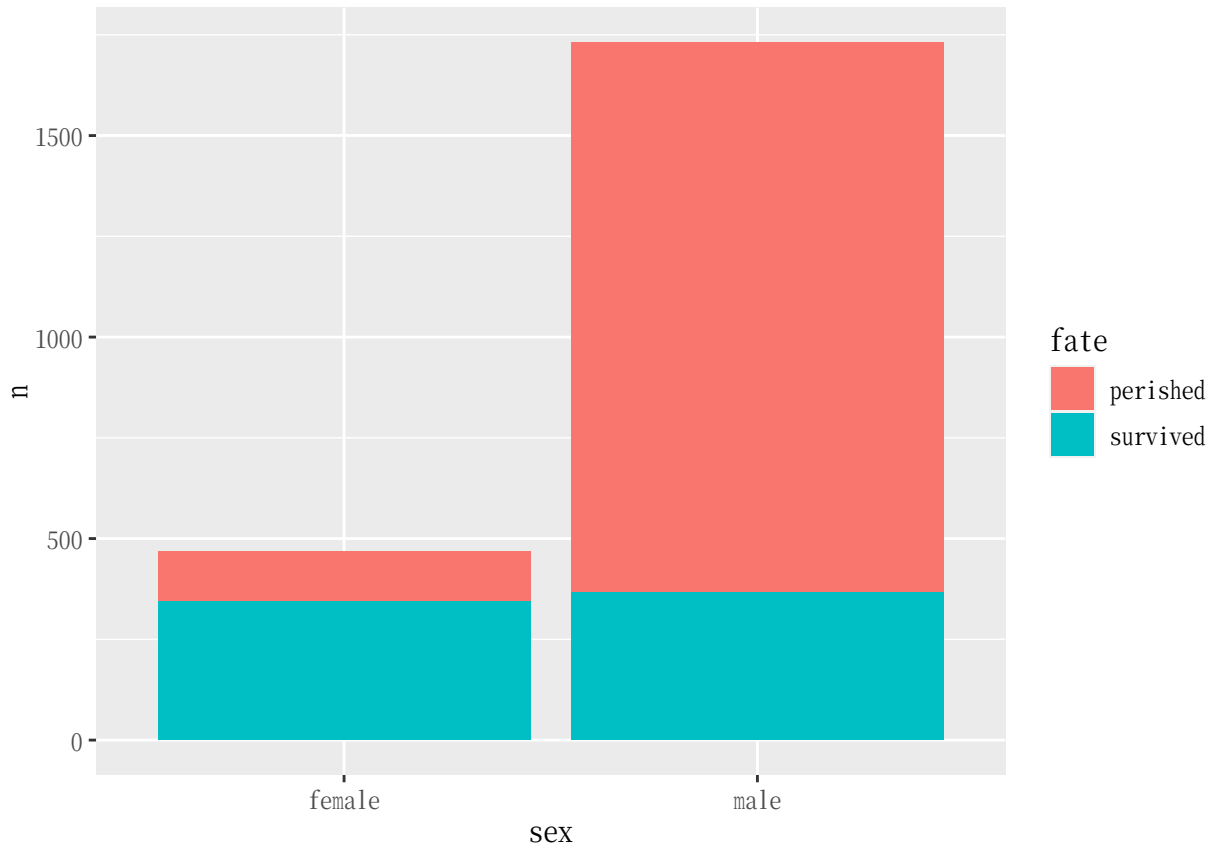
用 `theme()` 函数的 `legend.position` 参数可以指定图例的位置，如：

```
p + geom_col(position = "dodge") +
  theme(legend.position = "top")
```



做成堆叠形式：

```
p + geom_col(position = "stack")
```



`position = "stack"` 也是 `geom_col()` 函数的默认选项。

实际上，还可以用适当程序将存亡状态以及频数直接标在条形的色块内，`geom_text()` 函数可以在指定坐标位置标注指定的文字标签，见27.8.2。

datasets 包的 Titanic 数据集包含了泰坦尼克号乘客更详细的信息。我们按照存亡结果和舱位等级分切片作男女频数条形图：

```
titanic2 <- as.data.frame(Titanic) |>
  group_by(Class, Sex, Survived) |>
  summarise(n = sum(Freq), .groups = "drop_last") |>
  filter(Class != "Crew") |>
  mutate(Survived = factor(
    Survived, levels = c("Yes", "No"),
    labels = c("survived", "perished")))
ggplot(titanic2, aes(
  x = Sex, y = n, fill = Sex)) +
  geom_col() +
  facet_grid(Class ~ Survived) +
  guides(fill = "none")
```

这里将 `fill` 映射到了 Sex，使得表示男女的条形填充了不同的颜色。如果不满意上面的颜色，可以用 `scale_fill_manual()` 函数人为地指定颜色、对应离散值和图例标签。R 扩展包 colourpicker 提供了很好交互图形界面用来挑选颜色。

27.5.2 条形图的其它应用

`geom_col()` 不仅限于画频数或者比例的条形图，此函数可以将一般用折线图表现的内容画成条形图，但一定要注意一点：y 坐标轴必须从 0 开始，这也是 `geom_col()` 和 `geom_bar()` 函数默认的设置。如果坐标轴不从零开始，则条形的长度就不能正确表示对应的 y 变量数值。

举一个用条形图表示不是频数和比例的量的例子。socviz 包的 `oecd_sum` 数据集包含各年的美国以及 OECD 国家的期望寿命：

```
oecd_sum
```

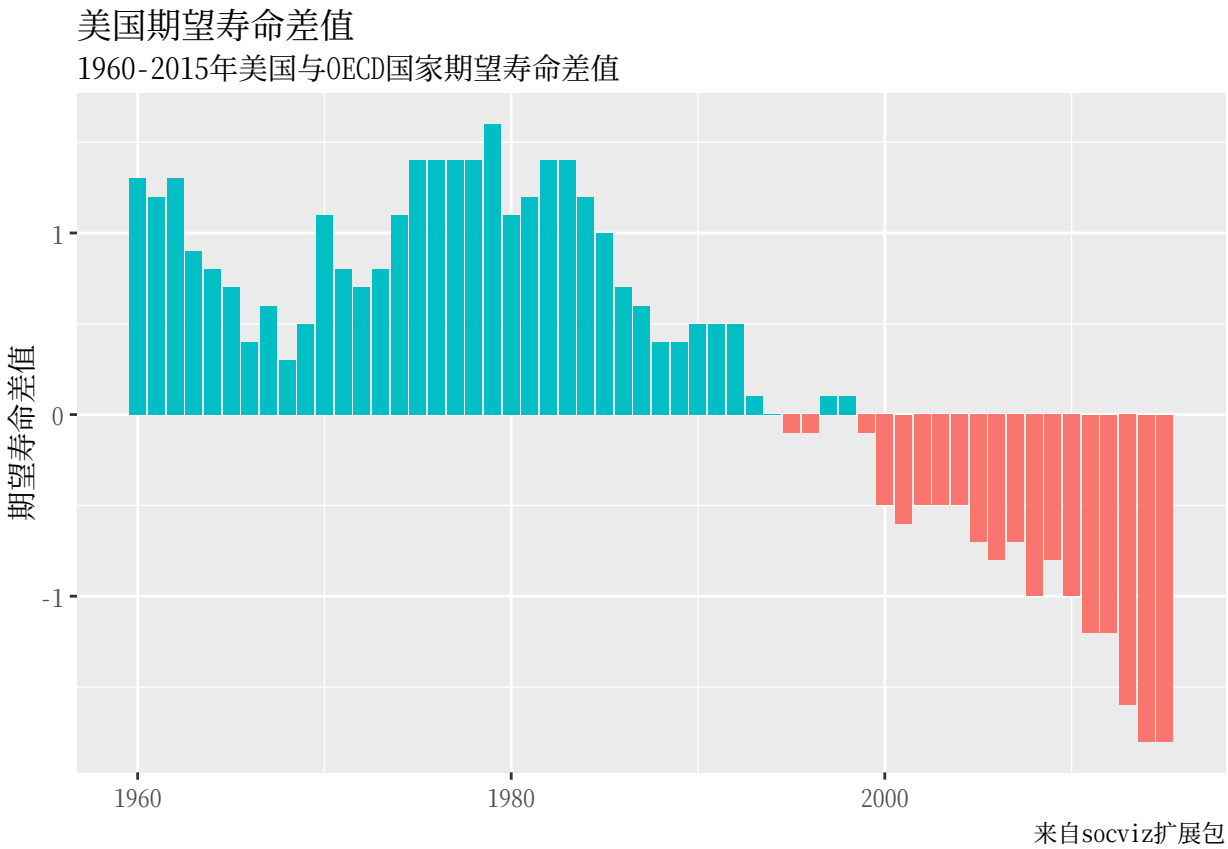
```
## # A tibble: 57 x 5
## # Groups:   year [57]
##     year other   usa  diff hi_lo
##    <int> <dbl> <dbl> <dbl> <chr>
##  1  1960  68.6  69.9 1.30  Below
```

```
## 2 1961 69.2 70.4 1.20 Below
## 3 1962 68.9 70.2 1.30 Below
## 4 1963 69.1 70 0.900 Below
## 5 1964 69.5 70.3 0.800 Below
## 6 1965 69.6 70.3 0.700 Below
## 7 1966 69.9 70.3 0.400 Below
## 8 1967 70.1 70.7 0.600 Below
## 9 1968 70.1 70.4 0.300 Below
## 10 1969 70.1 70.6 0.5 Below
## # ... with 47 more rows
```

我们用 geom_col() 作 diff 变量的条形图，并按照 hi_lo 变量对正负差值分别使用不同颜色：

```
ggplot(oecd_sum, aes(
      x = year,
      y = diff,
      fill = hi_lo)) +
  geom_col() +
  # 正负号的不同颜色不使用图例标注
  guides(fill = "none") +
  labs(
    x = NULL,
    y = " 期望寿命差值",
    title = " 美国期望寿命差值",
    subtitle = "1960-2015 年美国与 OECD 国家期望寿命差值",
    caption=" 来自 socviz 扩展包")
```

```
## Warning: Removed 1 rows containing missing values (position_stack).
```



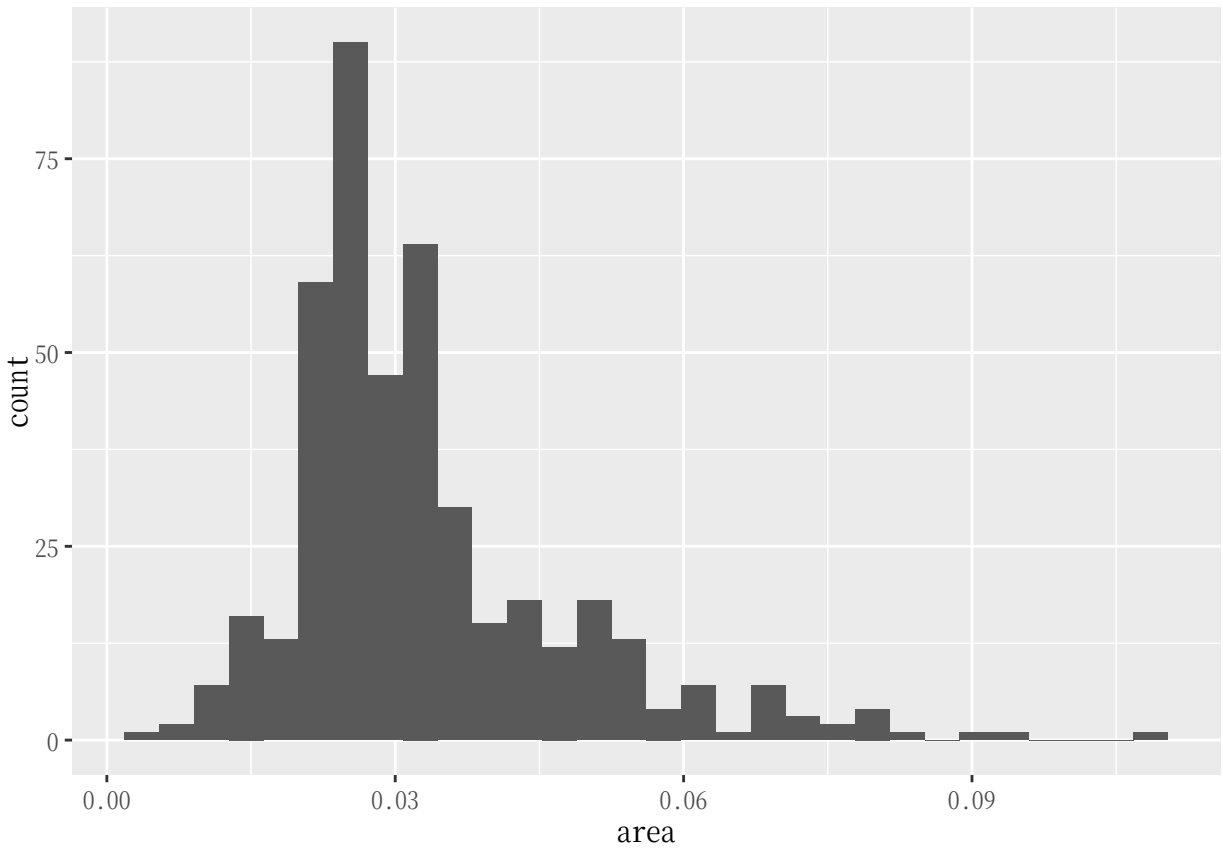
27.6 直方图与密度估计

条形图（barplot）反映分类变量的频数分布或者比例，直方图（histogram）反映连续取值的数值变量的分布。`geom_histogram()` 作直方图，可以自动选取合适的分组个数，也可以人为指定分组个数。

ggplot2 包中的 `midwest` 数据集包含了美国中西部的一些县的统计数据，如面积（单位：平方英里）。下面的程序对连续取值的数值型变量 area 作频数直方图，自动确定分组个数：

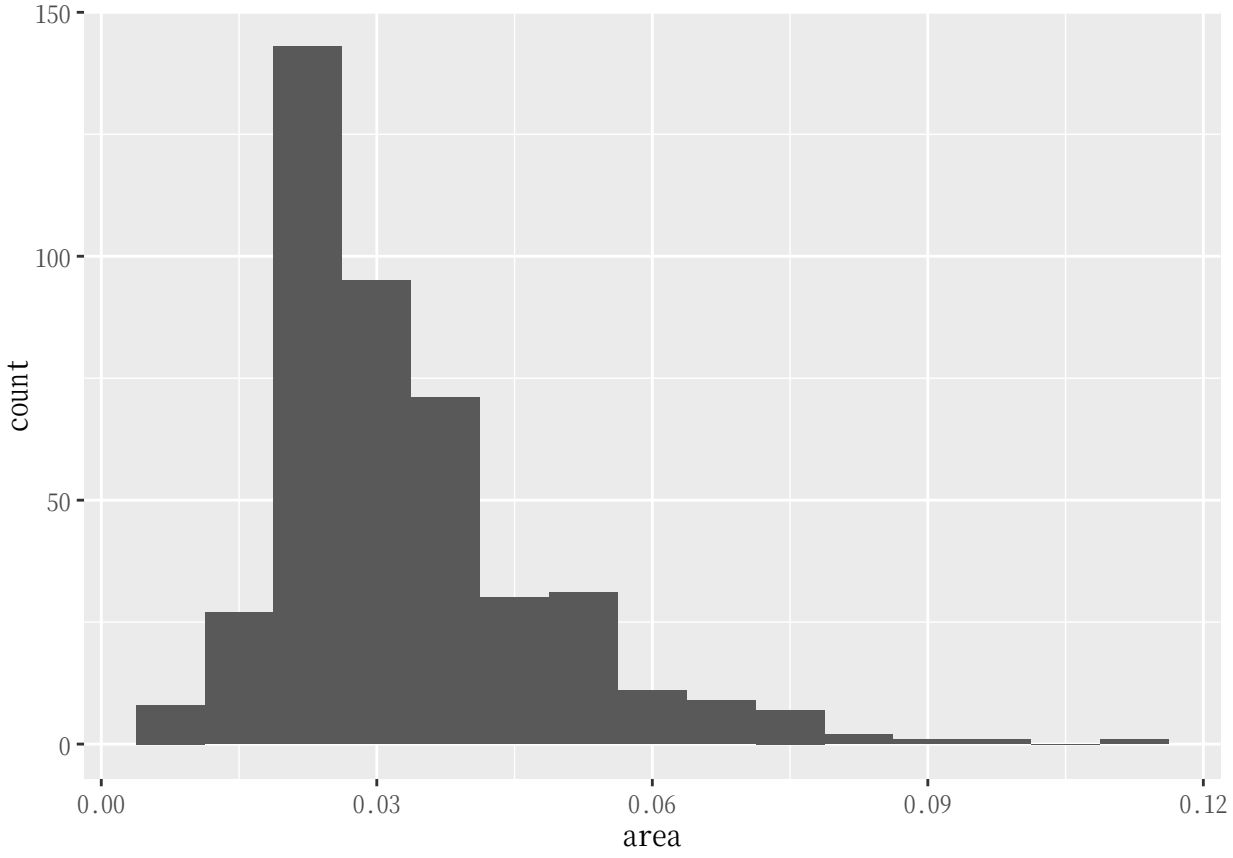
```
p <- ggplot(midwest, aes(x = area))
p + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



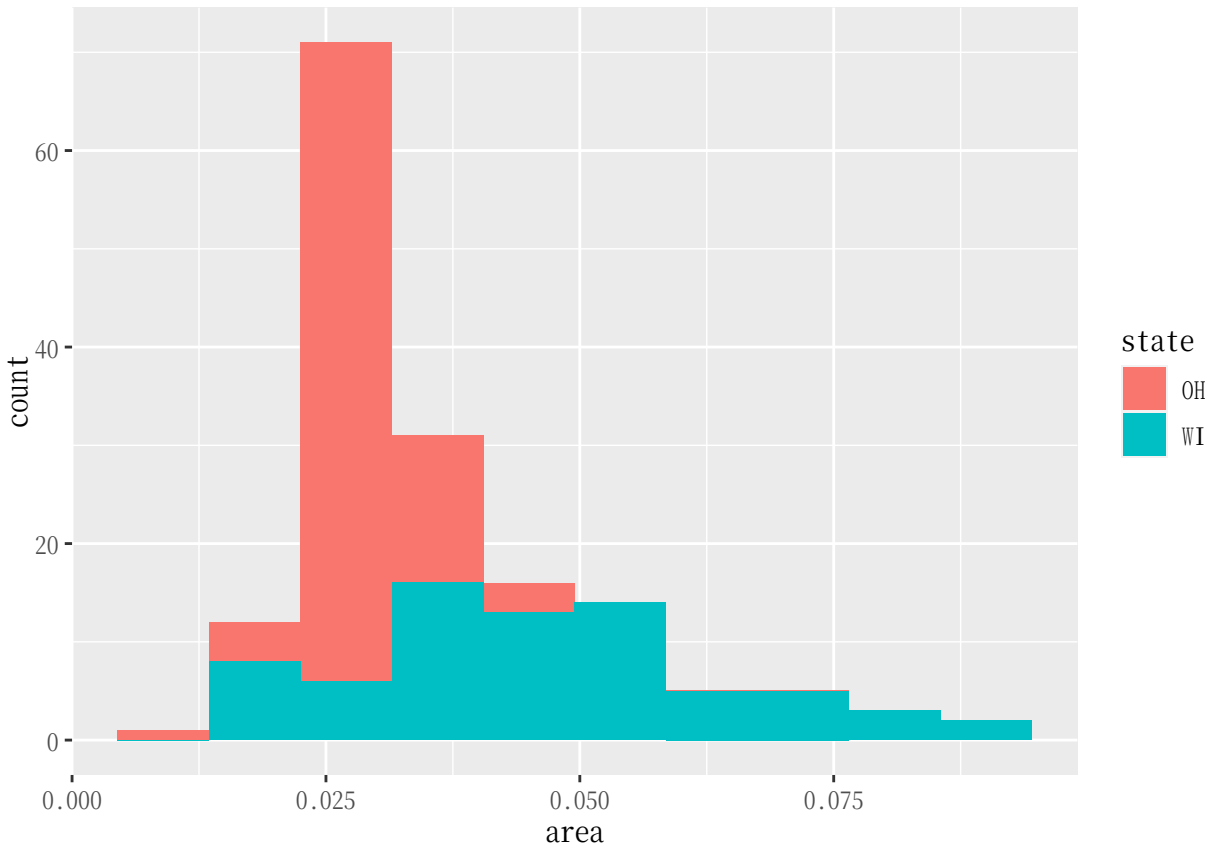
上面图形的纵坐标是频数 (count)，是每个组的频数。`geom_histogram()` 默认调用 `stat_bin()` 进行分组及频数统计。直方图的形状比较依赖于分组数与分组起始点位置，可以用 `bins` 参数控制分组数，用 `binwidth` 参数控制分组宽度，用 `center` 或者 `boundary` 参数控制组中心或者组边界对齐位置，如：

```
p + geom_histogram(bins = 15)
```



可以利用 `fill` 映射将构成直方图的观测按照某个分类变量分组，然后每个条形内部按照该分类变量的值分段染色，段内各颜色的长度代表该条形所在组某一类的频数，如：

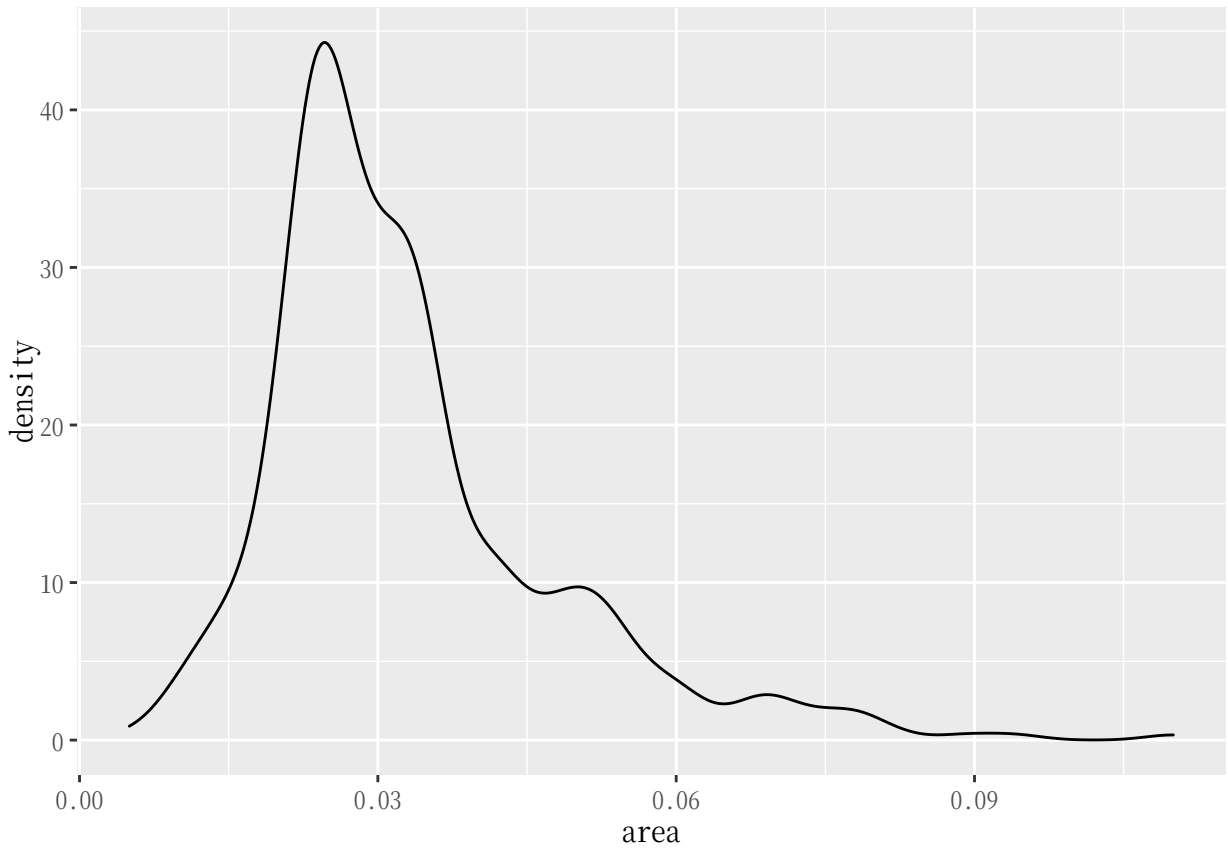
```
midwest_sub <- midwest |>
  filter(state %in% c("OH", "WI"))
ggplot(midwest_sub, aes(
  x = area, fill = state)) +
  geom_histogram(bins = 10)
```



可见面积较小的县主要来自 OH 州，面积较大的县主要来自 WI 州。

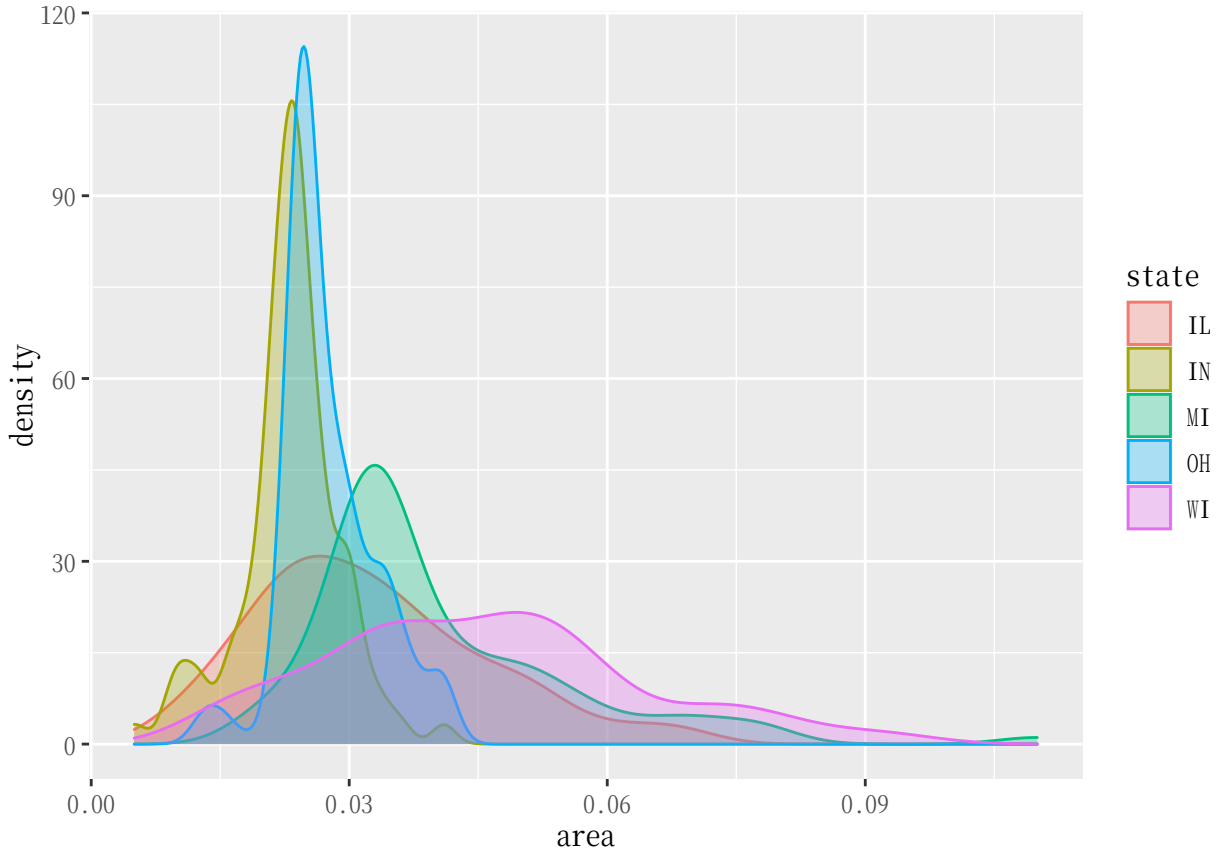
`geom_density()` 可以对连续变量绘制密度估计曲线，如：

```
ggplot(midwest, aes(x = area)) +
  geom_density()
```



下面的程序写法制作每个州的各县的面积密度估计，画在同一坐标系中：

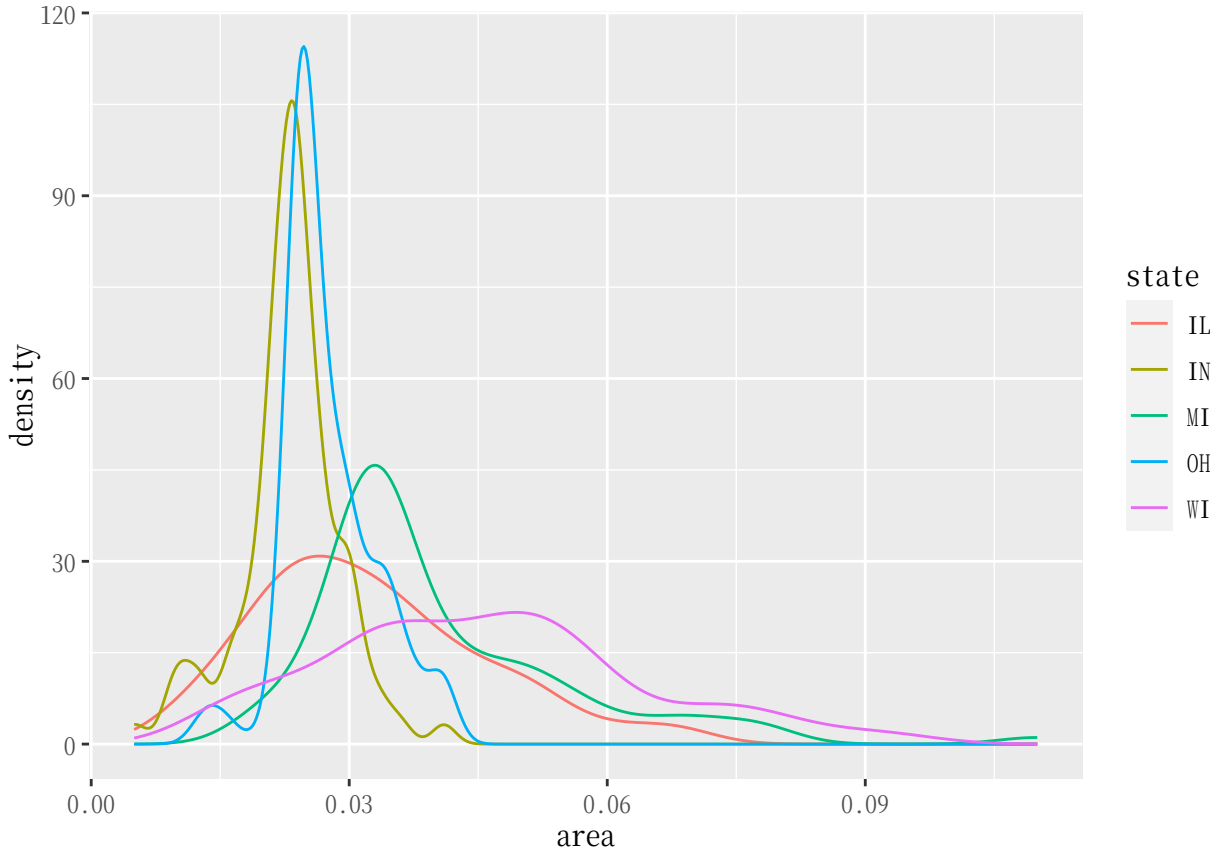
```
ggplot(midwest, aes(
      x = area,
      color = state,
      fill = state)) +
  geom_density(alpha = 0.3)
```



可以看出，IN 与 MI 州各县的面积偏小。WI 州各县的面积较大。

上面的图形可以借助于 `geom_line(stat = "density")` 改成仅有多条曲线：

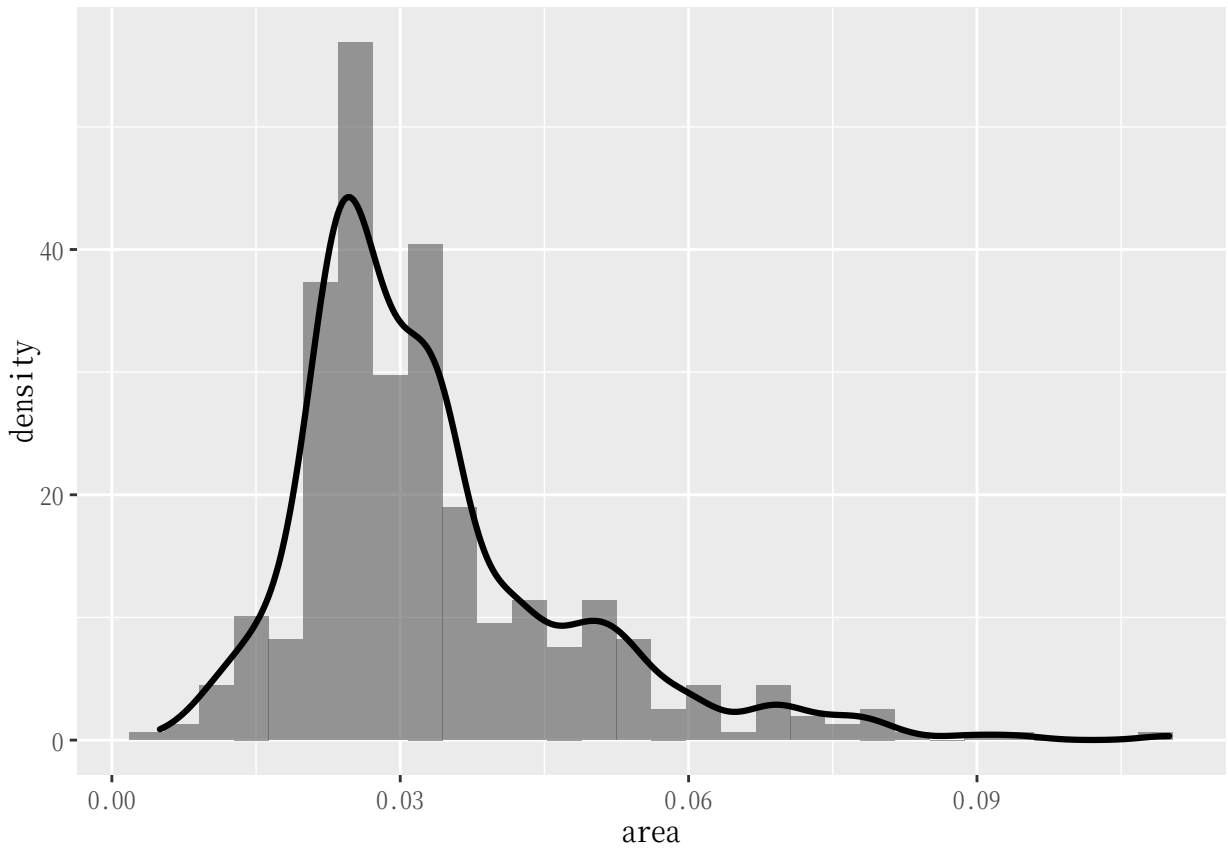
```
ggplot(midwest, aes(
       x = area,
       color = state)) +
  geom_line(stat = "density")
```



geom_density() 的纵轴是密度估计。为了能够将直方图与密度估计画在同一坐标系中，需要将直方图的纵轴也改为密度估计，如：

```
ggplot(midwest, aes(x = area)) +
  geom_histogram(
    mapping = aes(y = ..density..),
    alpha = 0.6) +
  geom_density(size = 1.1)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



进一步地，`geom_freqpoly()` 将直方图做成折线格式；`geom_bin2d()` 作二维的直方图，用不同颜色代表密度；`geom_density_2d()` 作二维密度估计等值线图。

27.7 绘图语法介绍

27.7.1 绘图语法

ggplot2 的不仅仅是能够做一些固定格式的图形，而是按照一种图形语法构建图形。切片功能可以将数据集分成若干子集作多幅切片，每幅切片中，又可以分层，每层有不同类型的图，各层叠加显示在一起。所以，分层语法作图结构如下：

- 有一个主要的数据集，以及从数据集变量到坐标位置、颜色、填充、大小、符号等的映射关系 (aesthetics);
- 有一到多个图层，比如散点图和平滑曲线图层，每个图层有几何对象、必要的统计变换、位置调整，还可以有额外的数据集以及额外的映射关系；
- 对每个映射关系有一个刻度 (scaling)，对 x、y 维，一般需要线性变换，偶尔用到对数变换之类的其它变换，颜色、填充等维度需要一些复杂的对应关系。无特殊需要时只要使用默认刻度；
- 有一个坐标系统，如直角坐标系、极坐标系、球面坐标系等，一般只要使用默认的坐标系统；
- 可以划分切片 (facetting)。

利用绘图语法既可以做出常见的统计图形，也可以做出各种新颖的图形，当然，就像语法正确的语句不一定有意义，用绘图语法做的新颖图形不一定有实际意义，还是要按照可视化的一般原则做出能说服读者的图形。

27.7.2 图形种类

`geom_xxx()` 提供了各种基本图形。列表如下：

- 基础图形：
  - `geom_blank()` 不画图，可以按映射的变量设定坐标范围；
  - `geom_point()` 每个观测为一个散点；
  - `geom_hline()`, `geom_vline()`, `geom_abline()` 画线；
  - `geom_path()` 每个观测提供 (x, y) 坐标，在相邻观测之间连线；
  - `geom_ribbon()` 需要 x 和 ymin, ymax 维，在从小到大排序后的相邻观测之间连接阴影区域；
  - `geom_segment()` 需要 x, y 和 xend, yend，为每个观测画一条线段；
  - `geom_rect()` 需要 xmin, xmax, ymin, ymax，为每个观测画一个长方形，可有填充色；
  - `geom_polygon()` 需要 x, y，将相邻观测连续并连接成一个闭合的多边形，中间填充颜色；
  - `geom_text()` 需要 x, y 和 lable，每个观测画一条文字标签。
- 单变量图层：
  - `geom_bar()`, `geom_col()` 作条形图；
  - `geom_histogram()` 对连续变量 x 作直方图；
  - `geom_density()` 对连续变量 x 作一元密度估计曲线；
  - `geom_dotplot()` 用原点作直方图；
  - `geom_freqpoly()` 用折线作直方图。
- 两变量图形：
  - 两个连续变量 x, y：
    * `geom_point()` 散点图；
    * `geom_quantile()` 拟合分位数回归曲线；
    * `geom_rug()` 在坐标轴处画数值对应的短须线；
    * `geom_smooth()` 画各种拟合曲线；
    * `geom_text()` 在指定的 x, y 位置画 label 给出的文字标签；
  - 显示二元分布:
    * `geom_bin2d()` 作长方形分块的二维直方图；
    * `geom_density2d()` 作二元密度估计等值线图；
    * `geom_hex()` 作正六边形分块的二维直方图。
  - 两个变量中有分类变量时：
    * `geom_count()`：重叠点越多画点越大；
    * `geom_jitter()`: 随机扰动散点位置避免重叠，数值变量有重叠时也可以用；

  - 一个连续变量和一个分类变量：
    * `geom_col()` 作条形图，对分类变量的每个值画一个条形，长度与连续变量值成比例；
    * `geom_boxplot()` 对每个类做一个盒形图；
    * `geom_violin()` 对每个类做一个小提琴图。
  - 一个时间变量和一个连续变量：
    * `geom_area()` 作阴影曲线图，曲线下方填充阴影色；
    * `geom_line()` 作折线图，在相邻两个时间之间连接线段；
    * `geom_step()` 作阶梯函数图，在相邻两个时间之间连接阶梯函数线。
  - 不确定性：
    * `geom_crossbar()` 对每个观测输入的 x, y, ymin, ymax 画中间有线的纵向条形；
    * `geom_errbar()` 对每个观测输入的 x, ymin, ymax 画纵向误差条；
    * `geom_linerange()` 对每个观测输入的 x, ymin, ymax 画一条竖线；
    * `geom_pointrnage()` 对每个观测输入的 x, y, ymin, ymax 画一条中间有点的竖线。
  - 地图：
    * `geom_map()`: 用区域边界坐标数据画边界线地图。

- 三个变量：
  - `geom_contour()`: 用输入的 x, y, z 数据画等值线图。
  - `geom_tile()` 用输入的 x, y 位置, width, height 大小和指定的 fill 维画长方形色块填充图。
  - `geom_raster()` 是 `geom_tile()` 的长方形大小相同时的快速版本。

27.8 更多图形种类

27.8.1 连续变量的分组图形

选用 socviz 包的 organdata 数据集，这是 17 个 OECD 国家历年的器官捐献情况以及一些其他变量的记录。其中前 6 列的一些抽样数据：

```
organdata |>
  select(1:6) |>
  sample_n(size = 10)
```

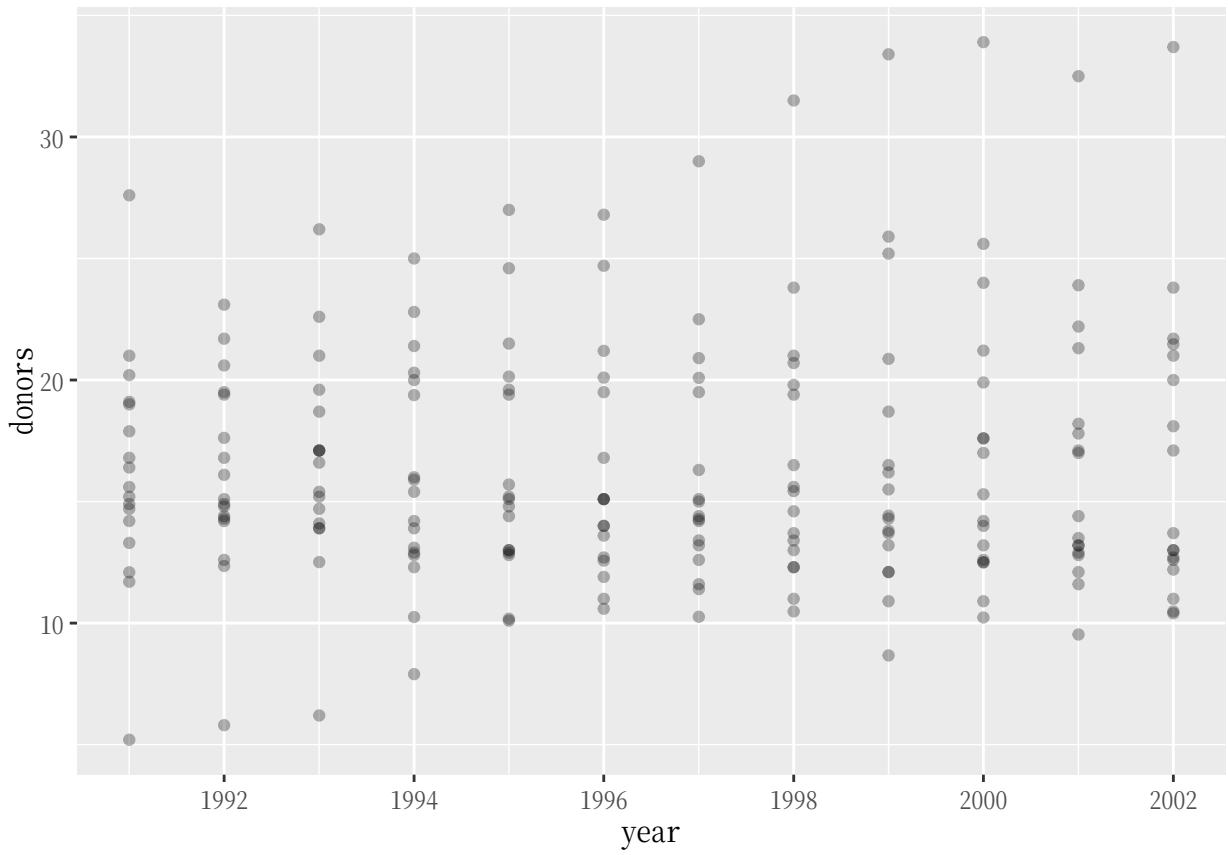
```
## # A tibble: 10 x 6
##    country       year       donors   pop pop_dens   gdp
##    <chr>         <date>      <dbl> <int>    <dbl> <int>
##  1 Australia     2002-01-01   10.5 19663    0.254 28168
##  2 Switzerland   1997-01-01   14.3  7089   17.2   27675
##  3 Spain         1994-01-01   25   39166    7.74  15024
```

```
##  4 Spain         NA          NA      NA   NA        NA
##  5 Austria       1991-01-01  27.6  7755    9.25  19860
##  6 Australia     NA          NA   17065    0.220 16774
##  7 France        1996-01-01  15.1 58026   10.5   21990
##  8 United States NA          NA      NA   NA        NA
##  9 Belgium       NA          NA      NA   NA        NA
## 10 Switzerland   1991-01-01  15.6  6797   16.5   24879
```

变量 donors 是每百万人中器官捐献数。作 donors 对 year 的散点图：

```
ggplot(organdata, aes(
      x = year,
      y = donors)) +
  geom_point(alpha=0.3)
```

```
## Warning: Removed 34 rows containing missing values (geom_point).
```

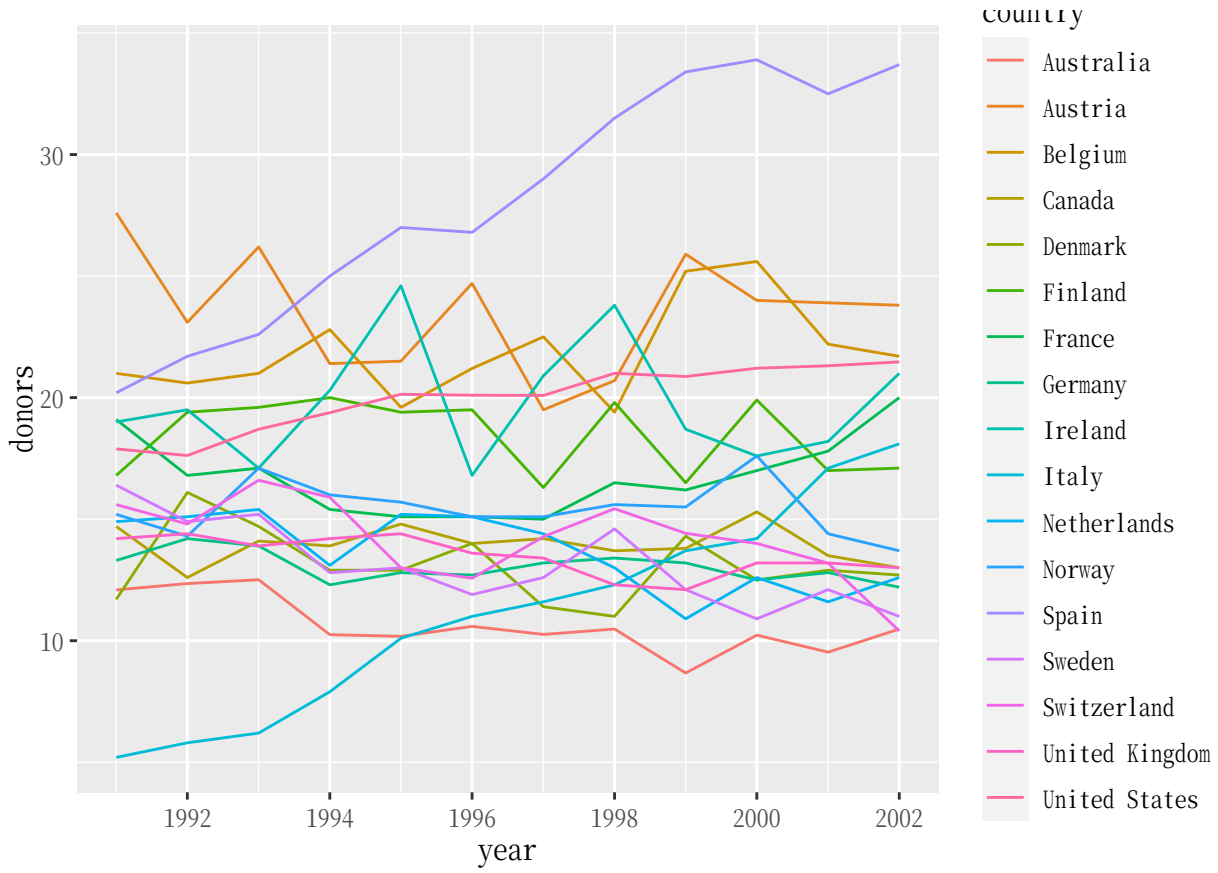


每年有多个数值，是不同国家的捐献数。这个图形不能反映一种时间趋势，不太有用。

可以对每个国家画一条折线图：

```
ggplot(organdata, aes(
      x = year,
      y = donors,
      color = country)) +
  geom_line()
```

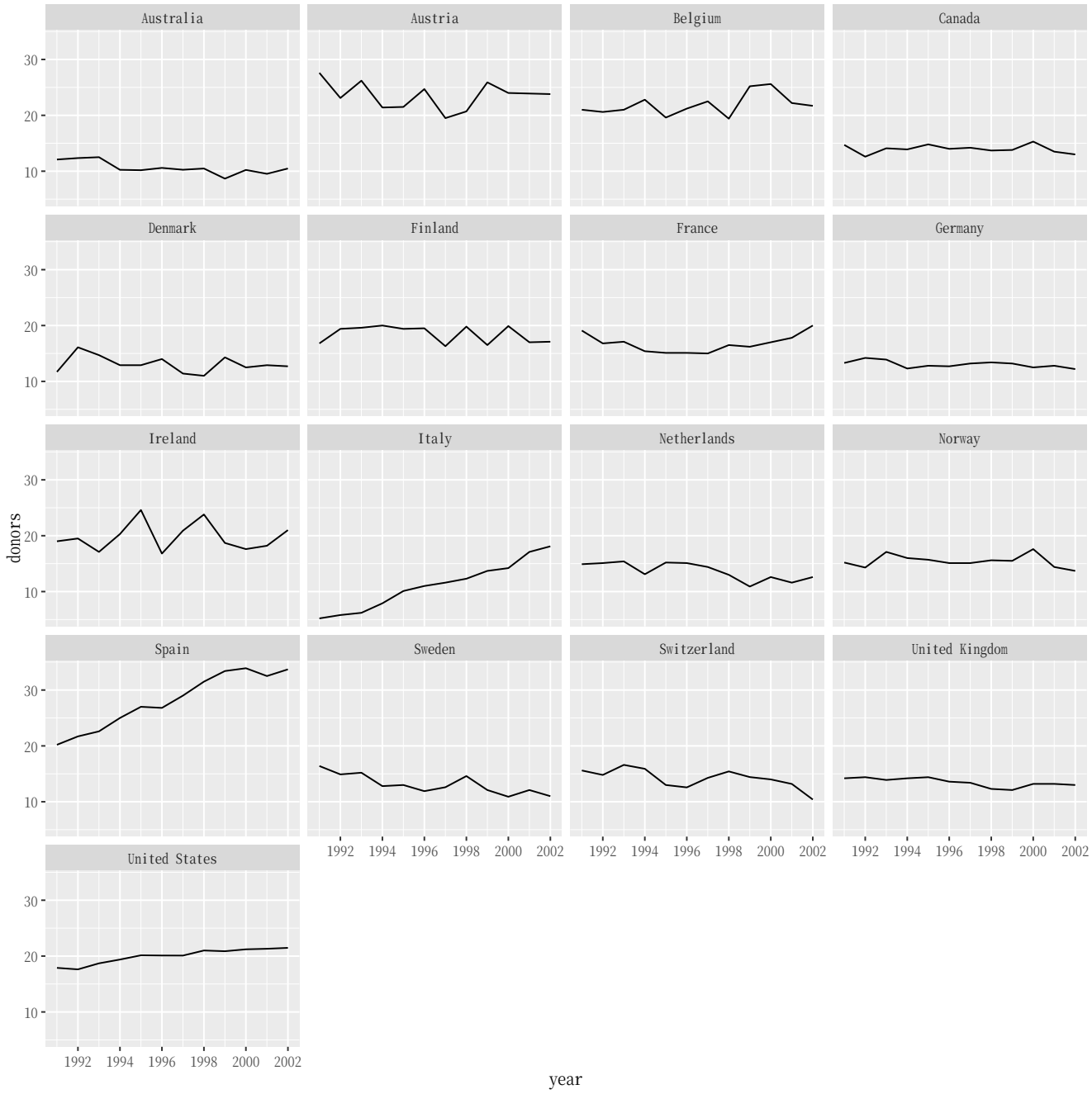
```
## Warning: Removed 34 row(s) containing missing values (geom_path).
```



共有 17 个国家，每个国家做了器官捐赠率随时间变化的折线图。用切片的方法将其分配到不同的切片：

```
ggplot(organdata, aes(
      x = year,
      y = donors)) +
  geom_line() +
  facet_wrap(~ country, ncol=4)
```

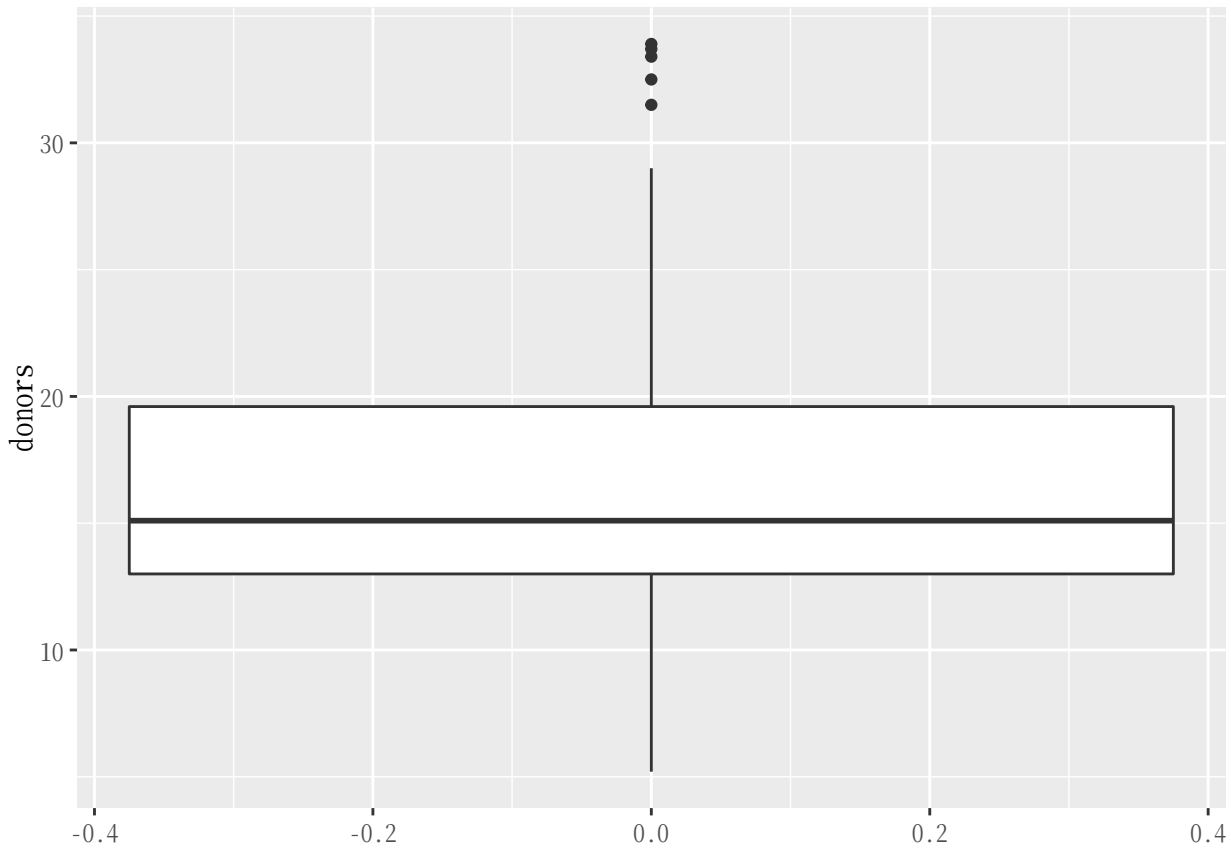
```
## Warning: Removed 2 row(s) containing missing values (geom_path).
```



用 `geom_boxplot()` 可以做盒形图，能够画出连续型变量的主要分位数，表现变量分布，如：

```
ggplot(organdata, aes(y = donors)) +
  geom_boxplot()
```

```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```

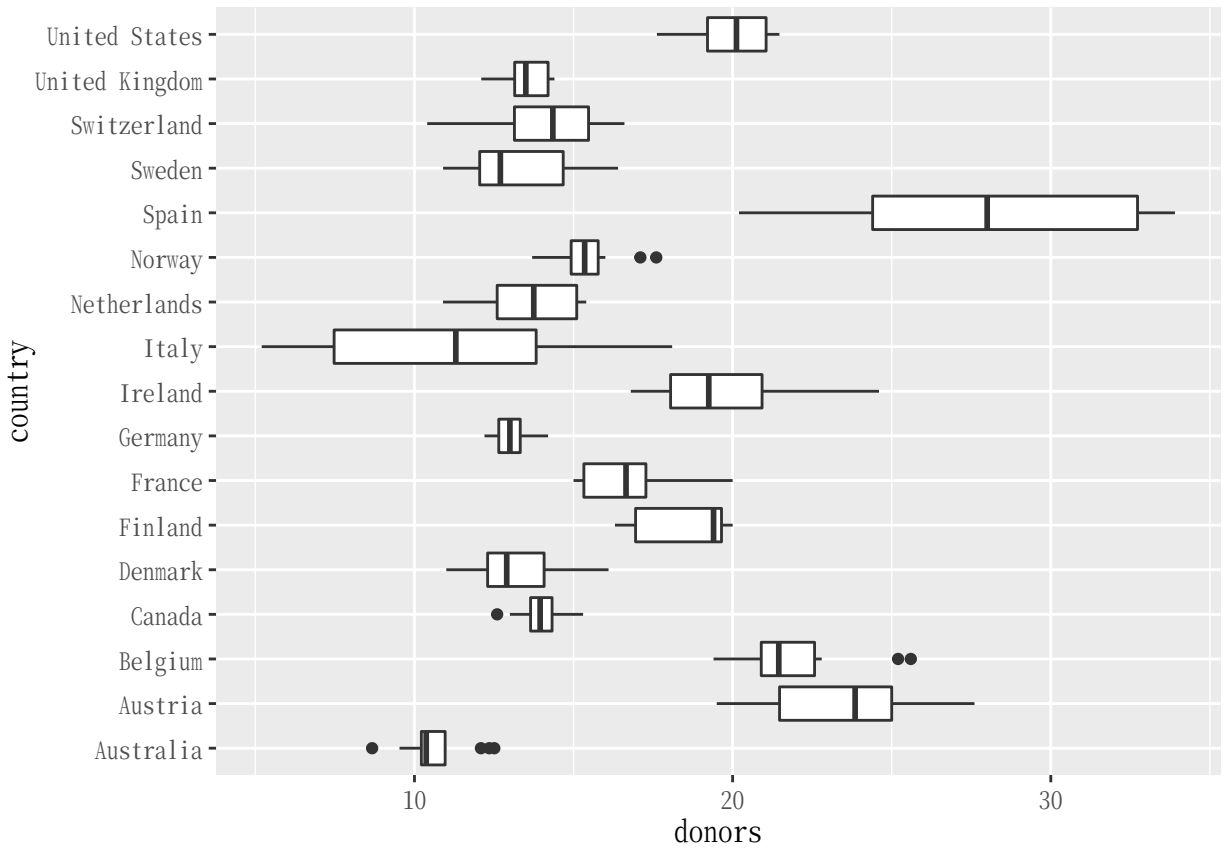


这是所有国家所有年的捐献率分布情况。类似函数还有 `geom_violin()`。

每个国家的捐献率单独做盒形图并且放在同一坐标系中：

```
ggplot(organdata, aes(y = donors, x = country)) +
  geom_boxplot() +
  coord_flip()
```

```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```

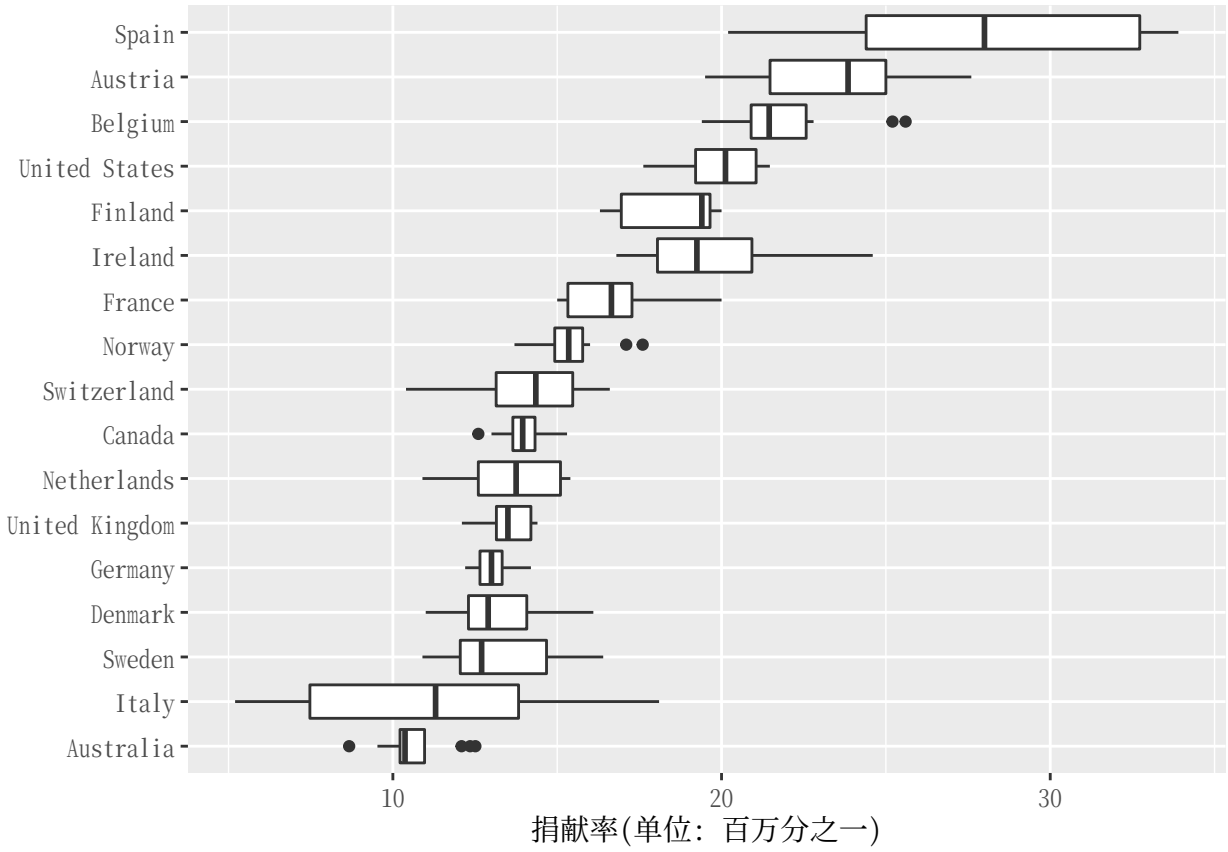


这里盒形图是横向的，如果仍然纵向作图，国家名称在横轴，许多个国家名称就会重叠在一起，只好仅显示其中一部分名称, 程序中用 `coord_flip()` 交换了 x 轴与 y 轴的作用。这个图形很好地比较了不同国家的历年捐献率的分布，比如，Spain 的捐献率最高。

为了将图形中的各个国家按照捐献率的某个统计量排序，可以使用 stats 包的 `reorder()` 函数，调整因子的水平次序，注意不是重排数据集中观测次序，而是重新定义因子的水平次序使得捐献率更高的国家的因子序号更小。如：

```
ggplot(organdata, aes(
      y = donors,
      x = reorder(
        country, donors, median, na.rm=TRUE))) +
  geom_boxplot() +
  labs(y = " 捐献率 (单位: 百万分之一)",
       x = NULL) +
  coord_flip()
```

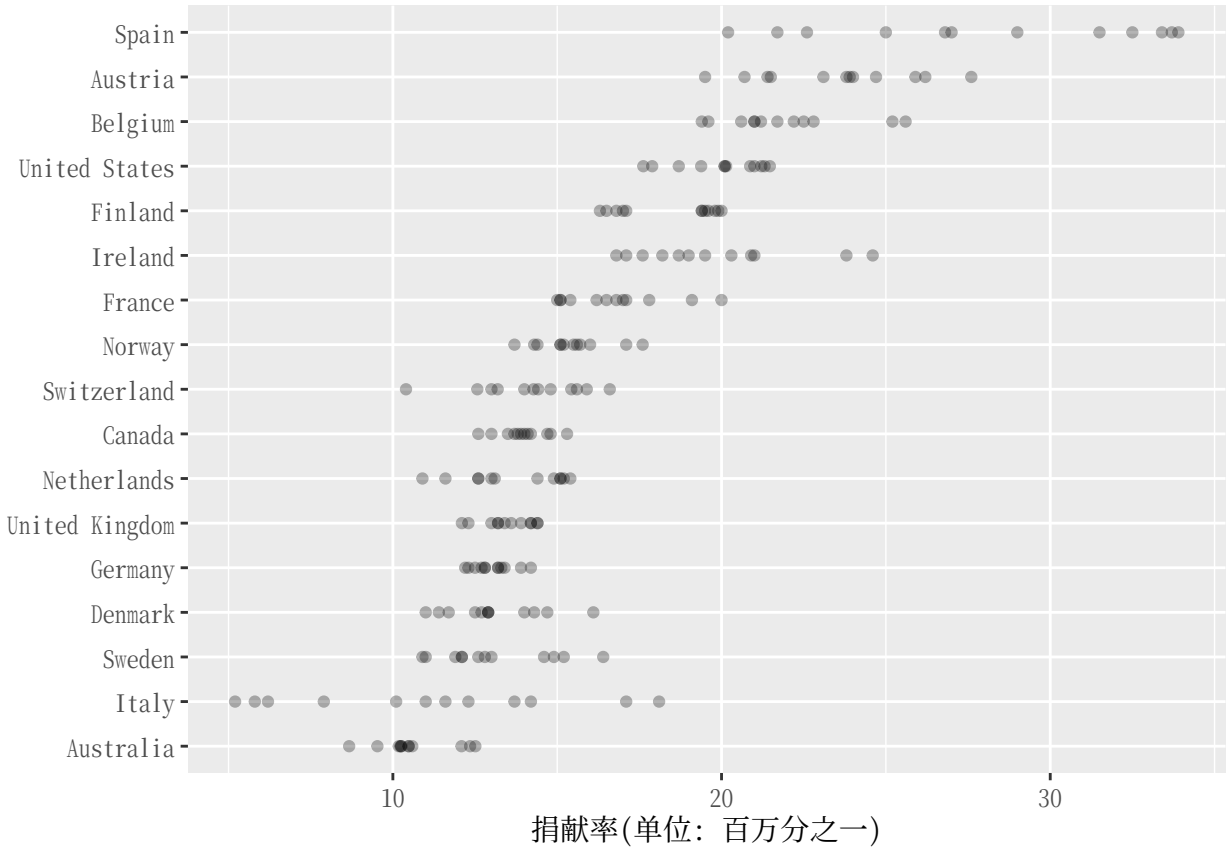
```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```



当每组（这里是每个国家）的观测个数很少时，也可以做成散点图，如：

```
p <- ggplot(organdata, aes(
      y = donors,
      x = reorder(
        country, donors, median, na.rm=TRUE)))
p + geom_point(alpha = 0.3) +
  labs(
    y = " 捐献率 (单位: 百万分之一)",
    x = NULL) +
  coord_flip()
```

```
## Warning: Removed 34 rows containing missing values (geom_point).
```

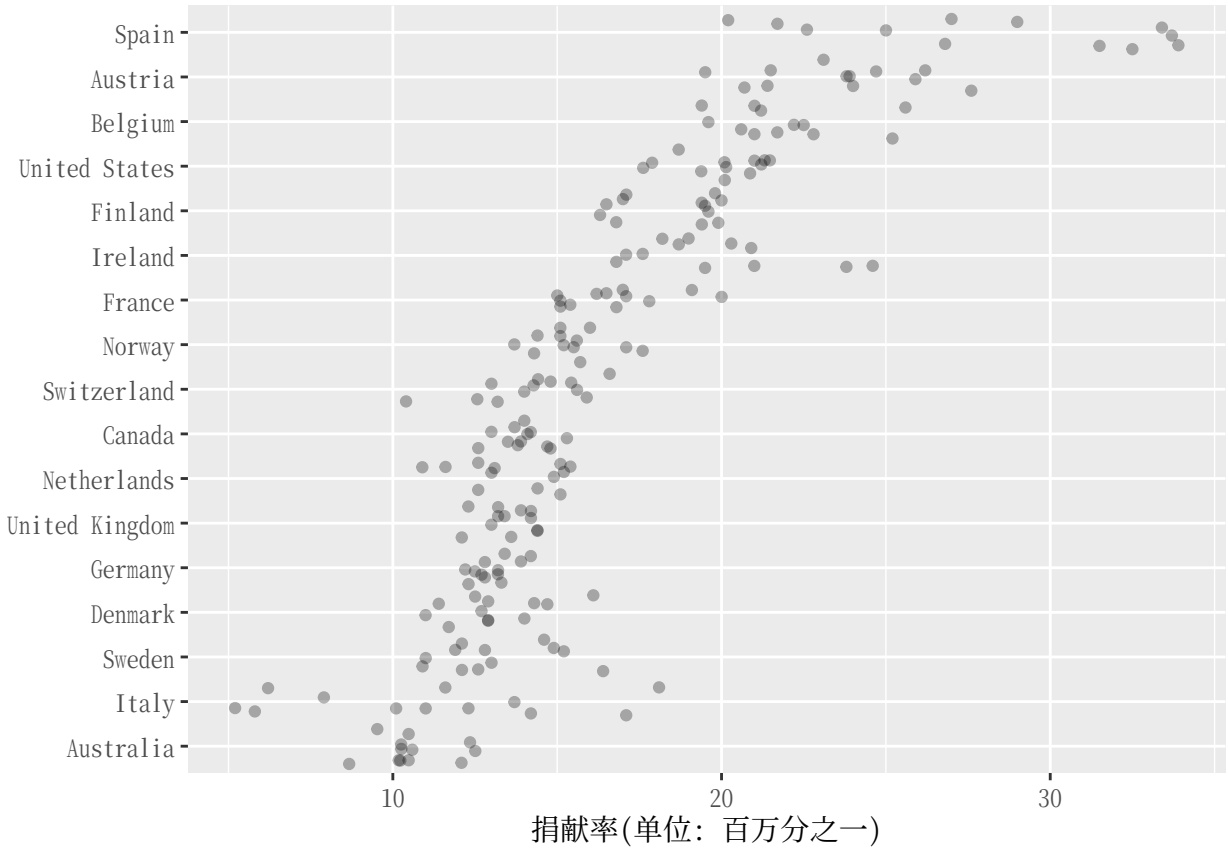


因为有的观测点完全重叠，所以用了 `alpha` 参数指定一定的透明度，重叠越多的点显示的颜色越深。但是，如果两个不同颜色的点完全重叠，半透明不能显示两个不同颜色的效果。

在作这样的散点图时，为了避免重叠的点，可以将 `geom_point()` 改为 `geom_jitter()`，如:

```
p + geom_jitter(alpha = 0.3) +
  labs(
    y = " 捐献率 (单位: 百万分之一)",
    x = NULL) +
  coord_flip()
```

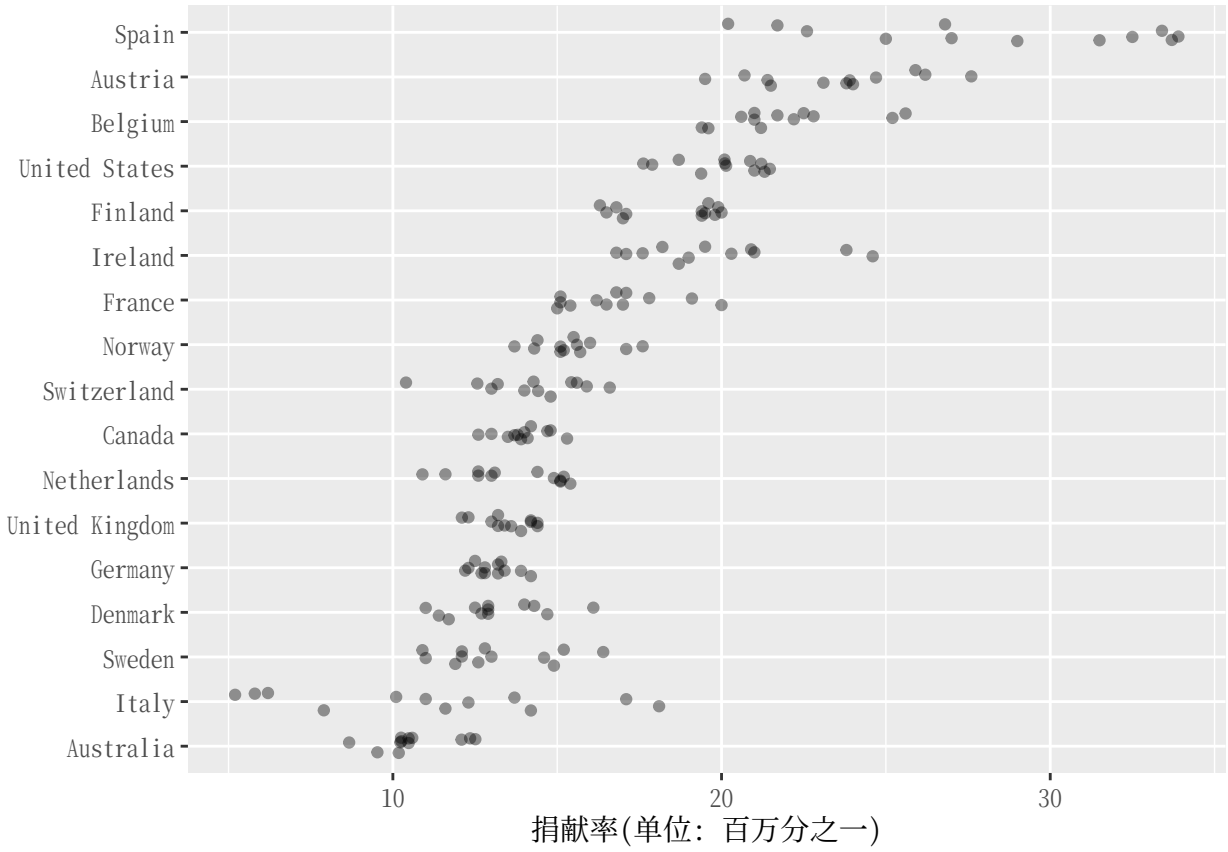
```
## Warning: Removed 34 rows containing missing values (geom_point).
```



上图的点的扰动过大了，使得不同国家的区分不明显了。作扰动的散点图时，可以用 `width` 指定左右扰动范围，用 `height` 指定上下扰动范围，这里只需要指定左右扰动范围，因为坐标轴对调所以就变成了上下扰动：

```
p + geom_jitter(alpha = 0.4, width = 0.2, height = 0) +
  labs(y = " 捐献率 (单位: 百万分之一)",
       x = NULL) +
  coord_flip()
```

```
## Warning: Removed 34 rows containing missing values (geom_point).
```

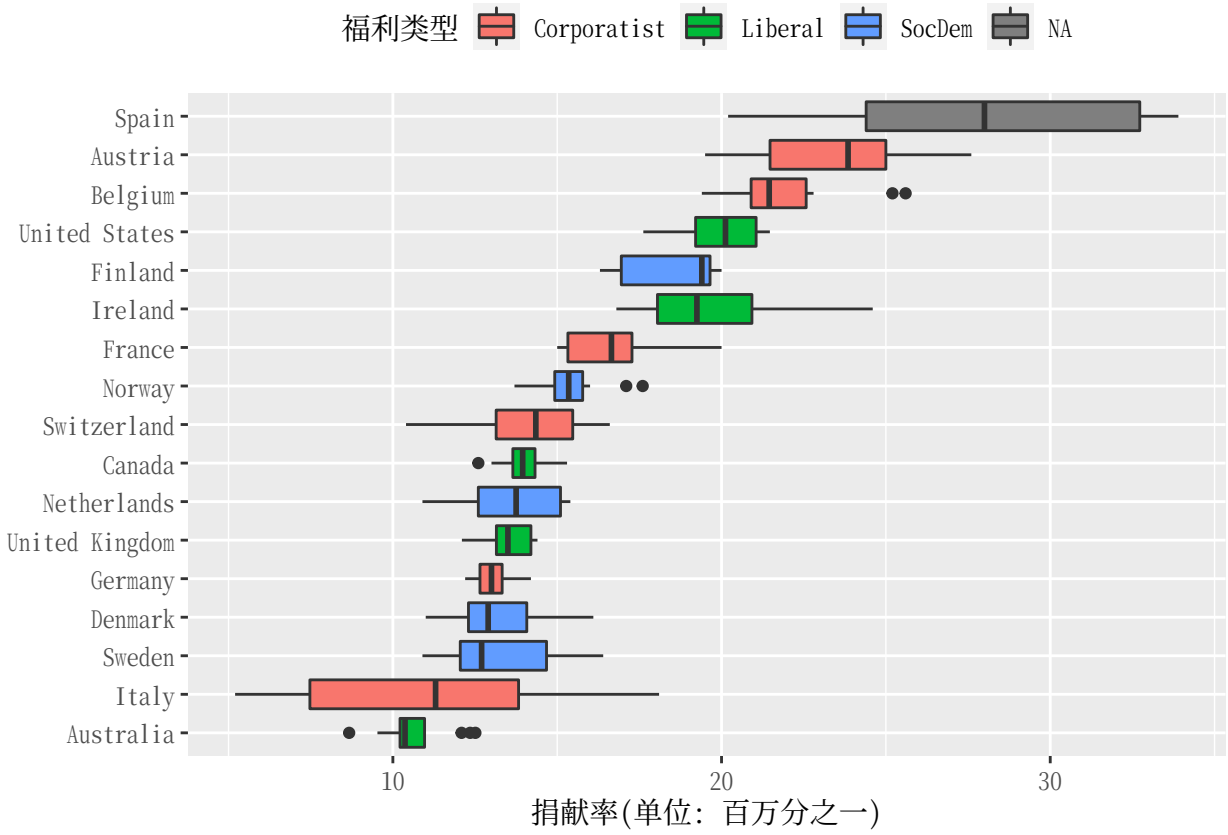


还可以做“蜂群图”，见下一章。

`geom_boxplot()` 也支持 `color`, `fill` 维度。organdata 中的变量 `world` 是一个国家的福利类型，用不同填充色表示 `world` 变量：

```
ggplot(organdata, aes(
      y = donors,
      x = reorder(
        country, donors, median, na.rm=TRUE),
      fill = world)) +
  geom_boxplot() +
  coord_flip() +
  labs(
    y = " 捐献率 (单位: 百万分之一)",
    x = NULL,
    fill = " 福利类型") +
  theme(legend.position = "top")
```

```
## Warning: Removed 34 rows containing non-finite values (stat_boxplot).
```



下面做不同国家的平均捐赠率的图形。首先得到统计数据：

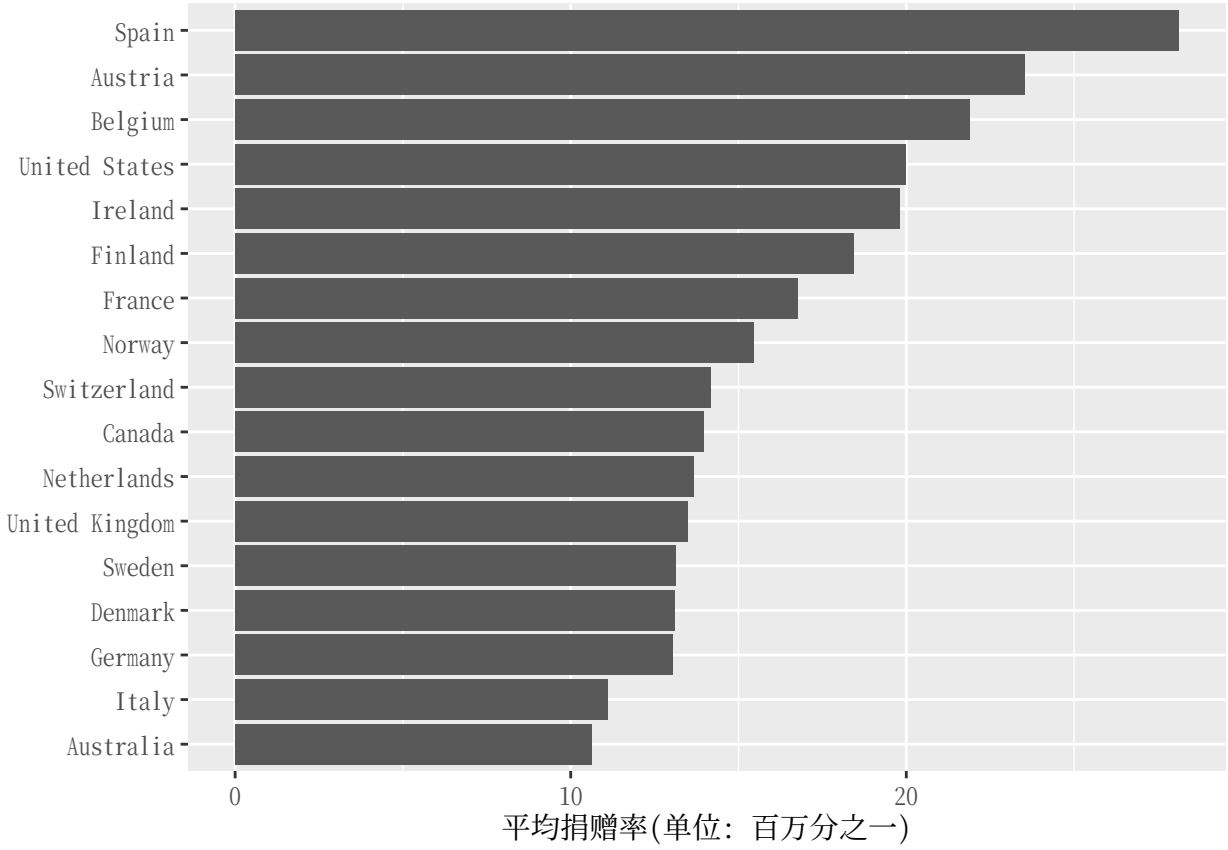
```
organdata2 <- organdata |>
  group_by(country) |>
  summarise(
    donors_n = sum(!is.na(donors)),
    donors_mean = mean(donors, na.rm=TRUE),
    donors_sd = sd(donors, na.rm=TRUE),
    donors_se = donors_sd / sqrt(donors_n))
organdata2
```

```
## # A tibble: 17 x 5
##    country        donors_n donors_mean donors_sd donors_se
##    <chr>             <int>       <dbl>     <dbl>     <dbl>
##  1 Australia            12        10.6      1.14      0.330
##  2 Austria              12        23.5      2.42      0.697
##  3 Belgium              12        21.9      1.94      0.559
##  4 Canada               12        14.0      0.751     0.217
##  5 Denmark              12        13.1      1.47      0.424
##  6 Finland              12        18.4      1.53      0.441
```

```
## 7 France              12        16.8     1.60      0.461
## 8 Germany             12        13.0     0.611     0.176
## 9 Ireland             12        19.8     2.48      0.715
## 10 Italy              12        11.1     4.28      1.23
## 11 Netherlands        12        13.7     1.55      0.448
## 12 Norway             12        15.4     1.11      0.320
## 13 Spain              12        28.1     4.96      1.43
## 14 Sweden             12        13.1     1.75      0.506
## 15 Switzerland        12        14.2     1.71      0.493
## 16 United Kingdom     12        13.5     0.775     0.224
## 17 United States      12        20.0     1.33      0.383
```

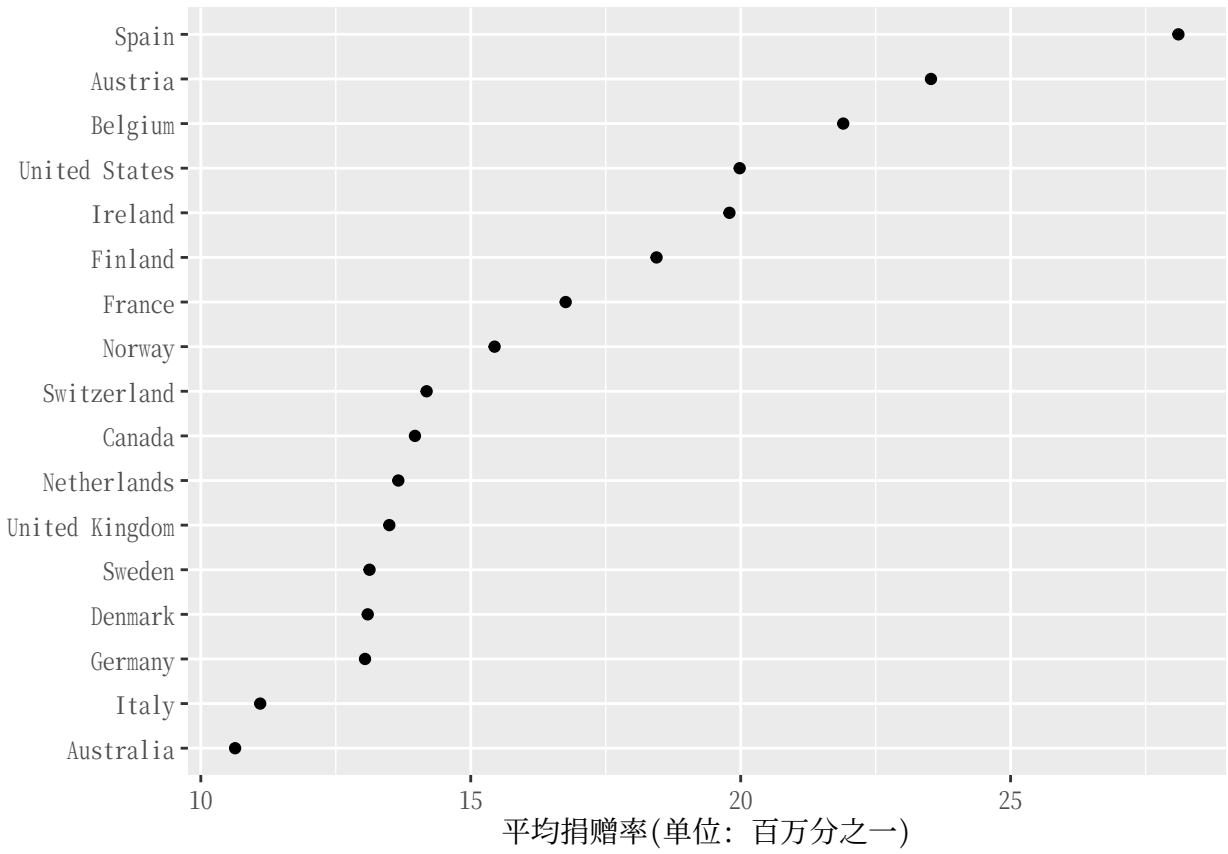
用条形图表现不同国家的平均捐献率：

```
ggplot(organdata2, aes(
      x = reorder(country, donors_mean),
      y = donors_mean)) +
  geom_col() +
  labs(
    x = NULL,
    y = " 平均捐赠率 (单位: 百万分之一)") +
  coord_flip()
```



这样的图形也可以做成点图，称为 Cleveland 点图。不需要再颠倒横纵坐标，直接规定 x 轴为平均捐赠率即可：

```
ggplot(organdata2, aes(
      y = reorder(country, donors_mean),
      x = donors_mean)) +
  geom_point() +
  labs(
    y = NULL,
    x = " 平均捐赠率 (单位: 百万分之一)")
```



注意条形图坐标轴默认从 0 开始，点图则一般从数据最小值开始。

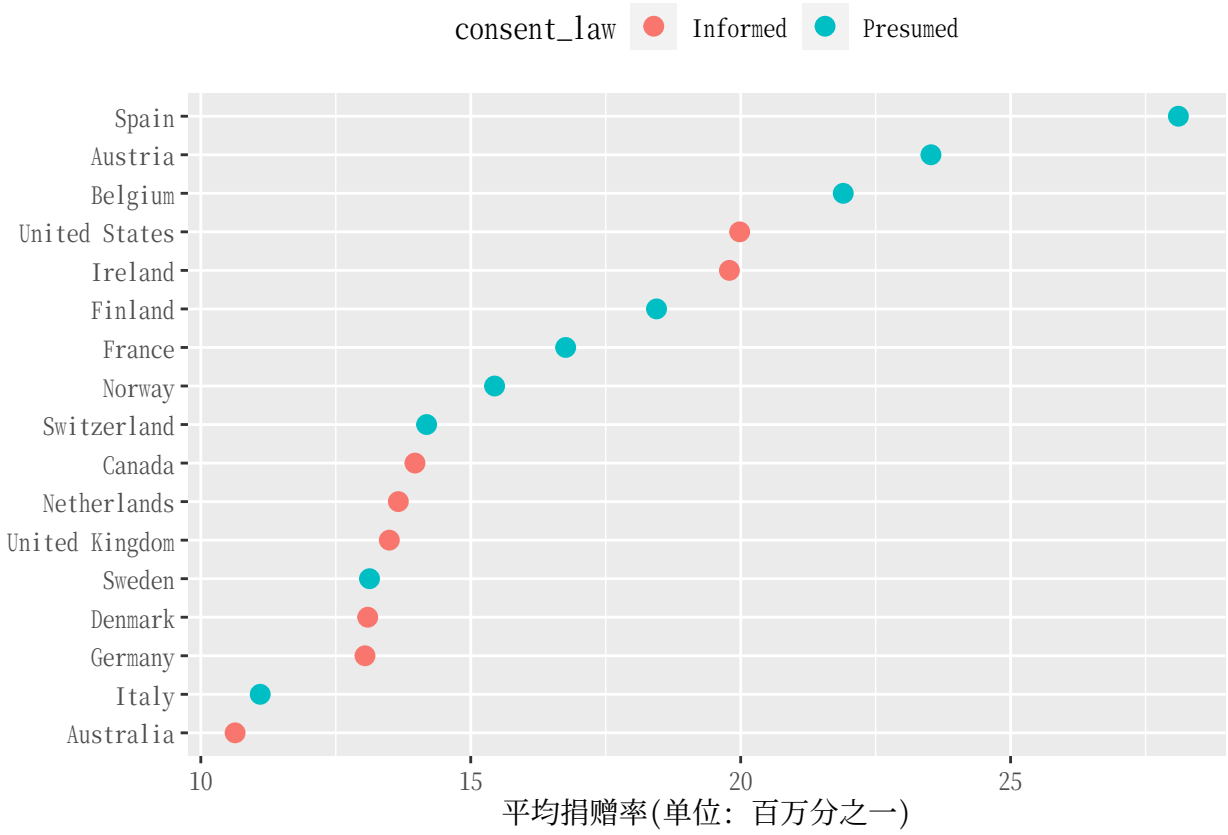
organdata 数据集中变量 `consent_law` 是关于一个国家中器官捐赠是必须告知还是默认捐赠的区别。为了在上面的点图中用不同颜色区分这两种做法，需要在分组汇总阶段就将 `consent_law` 也作为分组变量。这是因为 `summarise()` 函数会自动舍弃分组变量和统计结果之外的原有变量。

```
organdata3 <- organdata |>
  group_by(consent_law, country) |>
  summarise(
    donors_n = sum(!is.na(donors)),
    donors_mean = mean(donors, na.rm=TRUE),
    donors_sd = sd(donors, na.rm=TRUE),
    donors_se = donors_sd / sqrt(donors_n),
    .groups = "drop")
organdata3
```

```
## # A tibble: 17 x 6
##    consent_law country        donors_n donors_mean donors_sd donors_se
##    <chr>       <chr>             <int>       <dbl>     <dbl>     <dbl>
##  1 Informed    Australia            12        10.6      1.14      0.330
```

```
##  2 Informed    Canada              12        14.0     0.751     0.217
##  3 Informed    Denmark             12        13.1     1.47      0.424
##  4 Informed    Germany             12        13.0     0.611     0.176
##  5 Informed    Ireland             12        19.8     2.48      0.715
##  6 Informed    Netherlands         12        13.7     1.55      0.448
##  7 Informed    United Kingdom      12        13.5     0.775     0.224
##  8 Informed    United States       12        20.0     1.33      0.383
##  9 Presumed    Austria             12        23.5     2.42      0.697
## 10 Presumed    Belgium             12        21.9     1.94      0.559
## 11 Presumed    Finland             12        18.4     1.53      0.441
## 12 Presumed    France              12        16.8     1.60      0.461
## 13 Presumed    Italy               12        11.1     4.28      1.23
## 14 Presumed    Norway              12        15.4     1.11      0.320
## 15 Presumed    Spain               12        28.1     4.96      1.43
## 16 Presumed    Sweden              12        13.1     1.75      0.506
## 17 Presumed    Switzerland         12        14.2     1.71      0.493
```

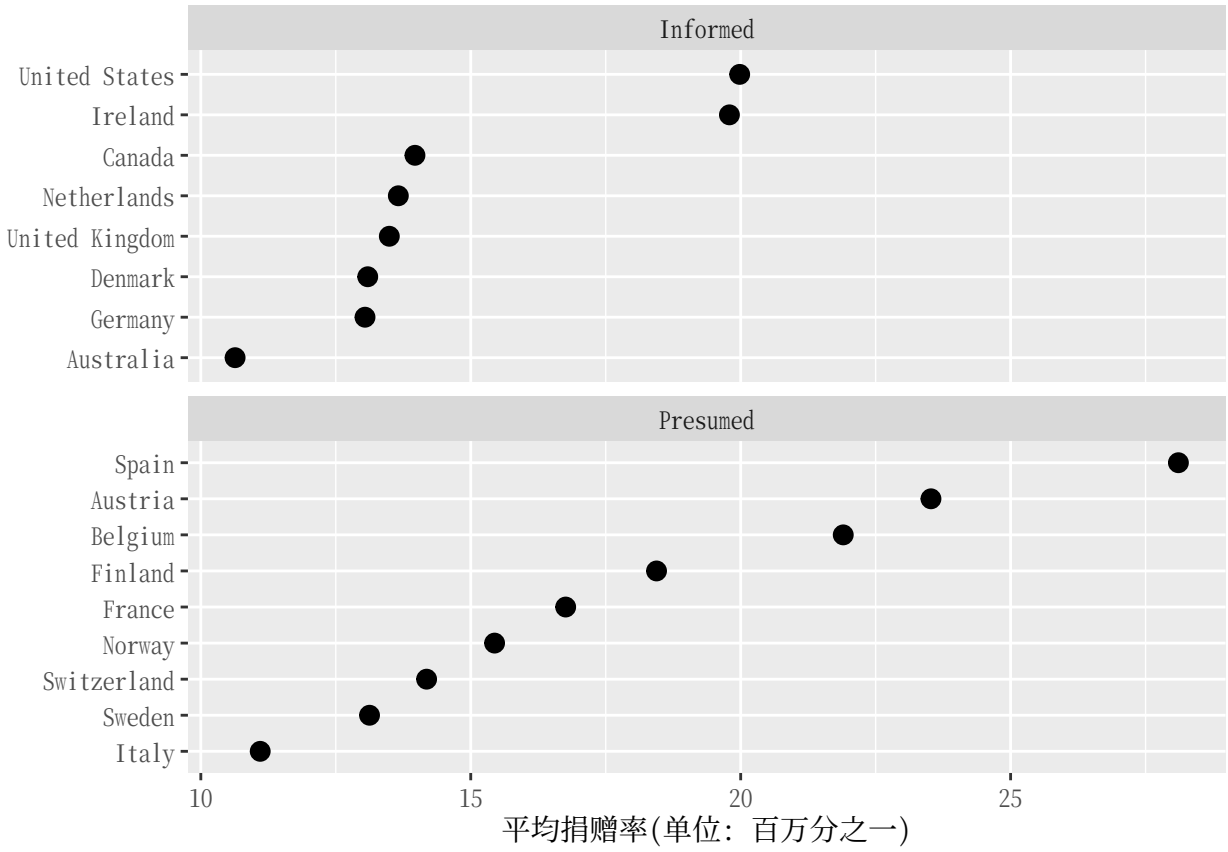
```
ggplot(organdata3, aes(
      y = reorder(country, donors_mean),
      x = donors_mean,
      color = consent_law)) +
  geom_point(size = 3) +
  labs(
    y = NULL,
    x = " 平均捐赠率 (单位: 百万分之一)") +
  theme(legend.position = "top")
```



平均捐赠率最高的三个国家都是不需告知预先假定同意的。

也可以将两种告知规定分成两个切片：

```
ggplot(organdata3, aes(
      y = reorder(country, donors_mean),
      x = donors_mean)) +
  geom_point(size = 3) +
  labs(
    y = NULL,
    x = " 平均捐赠率 (单位: 百万分之一)") +
  facet_wrap(
    ~ consent_law,
    ncol=1, scales = "free_y")
```



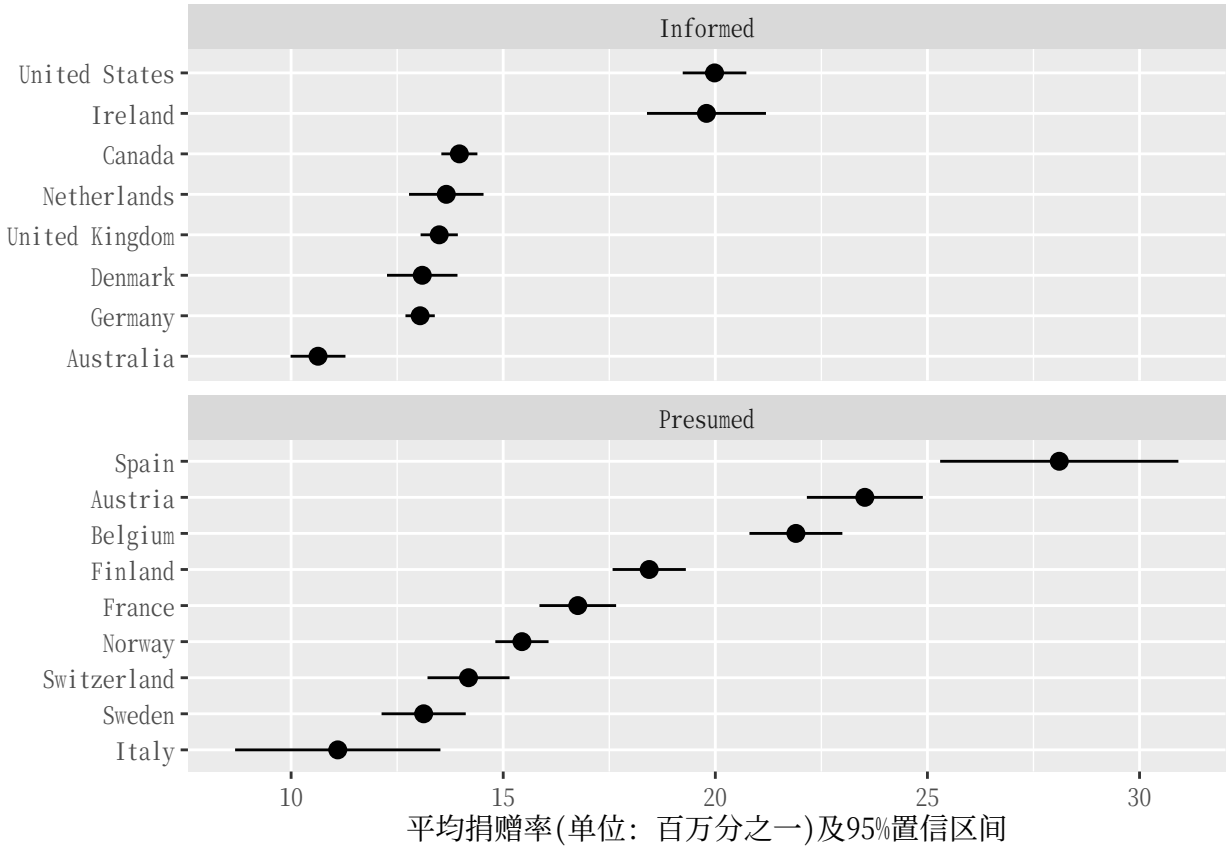
因为纵轴是分类变量，程序中的 `scales = "free_y"` 使得纵轴仅对存在的类留出空间。用了 `ncol = 1` 使得两种告知规定的切片上下排列，便于比较横坐标值。

当每个类别仅有一个数值时，一般推荐使用 Cleveland 点图，而不是条形图或者折线图。Cleveland 点图总是将类别值绘制在 y 轴，将要比较的数量值用 x 坐标表示，并将各类按照数量值大小次序排列。

可以表示平均值的点图上增加一条线，表示误差大小，所用函数为 `geom_pointrange()`。这个函数仅支持对 y 轴加误差线，所以需要用交换坐标轴的办法将分类变量放在 y 轴。比如，画出近似 95% 置信区间范围：

```
ggplot(organdata3, aes(
      x = reorder(country, donors_mean),
      y = donors_mean)) +
  geom_pointrange(mapping = aes(
    ymin = donors_mean - 1.96*donors_se,
    ymax = donors_mean + 1.96*donors_se)) +
  labs(
    x = NULL,
    y = " 平均捐赠率 (单位: 百万分之一) 及 95% 置信区间") +
  coord_flip() +
  facet_wrap(
    ~ consent_law,
```

```
    ncol=1, scales = "free_y")
```



类似的函数还有 `geom_linerange()`、`geom_crossbar()`、`geom_errorbar()`。

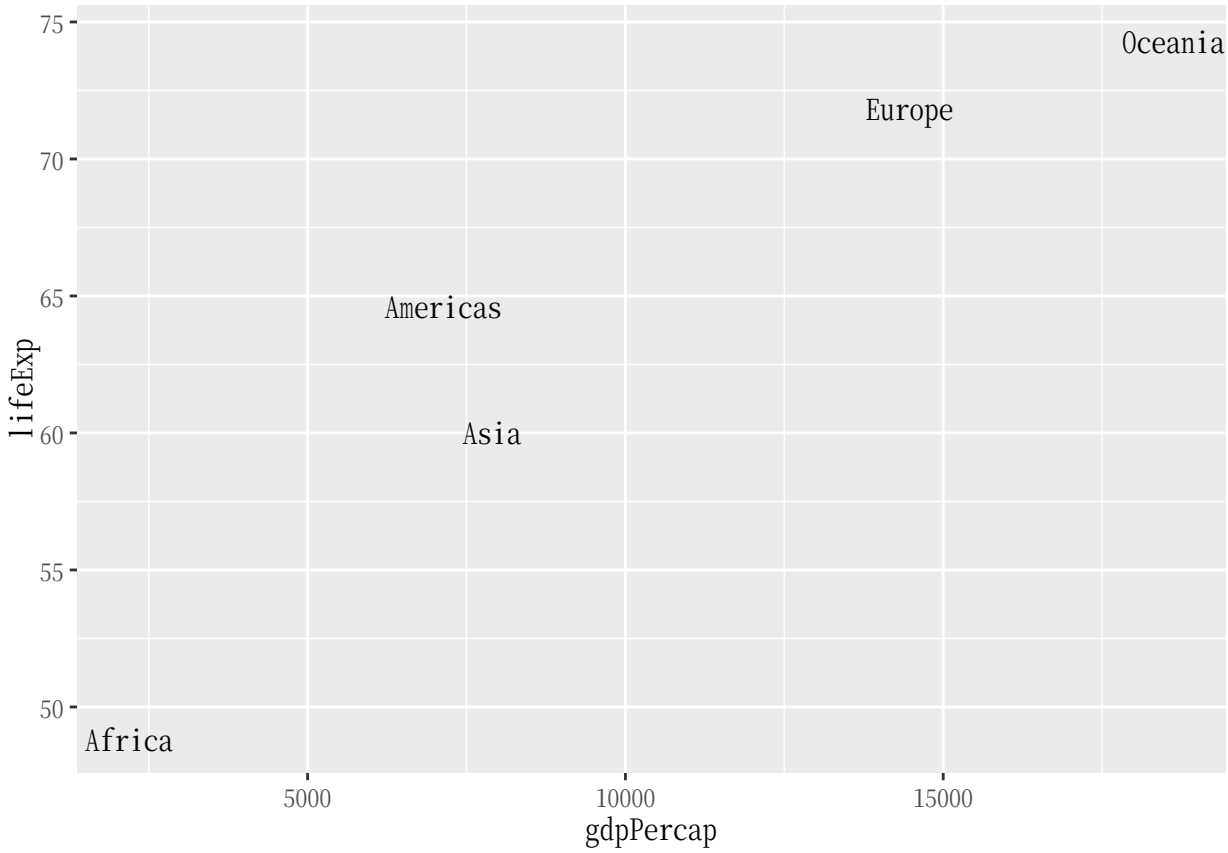
27.8.2 坐标系中的文字

类似于散点图，可以将指定的文字绘制在指定的坐标位置，使用 `geom_text(mapping = aes(label = 字符型变量))`。

例如，gapminder 数据集中各大洲的平均寿命与平均 gdp 的文字散点图：

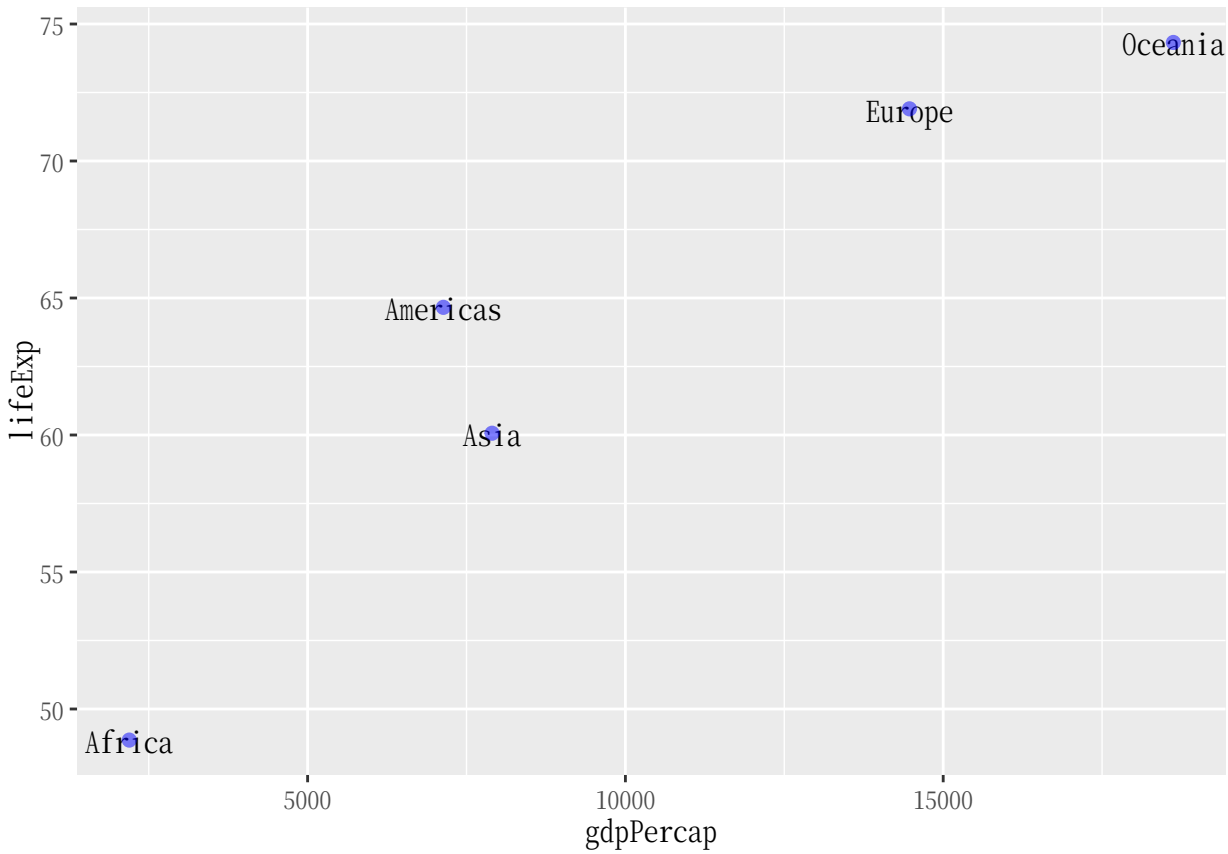
```
gapminder2 <- gapminder |>
  group_by(continent) |>
  summarise(
    lifeExp = mean(lifeExp, na.rm=TRUE),
    gdpPercap = mean(gdpPercap))
p <- ggplot(gapminder2, aes(
  x = gdpPercap,
  y = lifeExp,
```

```
  label = continent))
p + geom_text()
```



可以同时绘制散点：

```
p + geom_text() +
  geom_point(size = 2, col="blue", alpha=0.5)
```



ggrepel 扩展包提供了增强的图形文本功能。`geom_text_repel()` 提供了与 `geom_text()` 类似的功能，但可以避免文字重叠。

考虑 socviz 扩展包的 `elections_historic` 数据集，这是美国历次总统选举情况数据。部分数据显示：

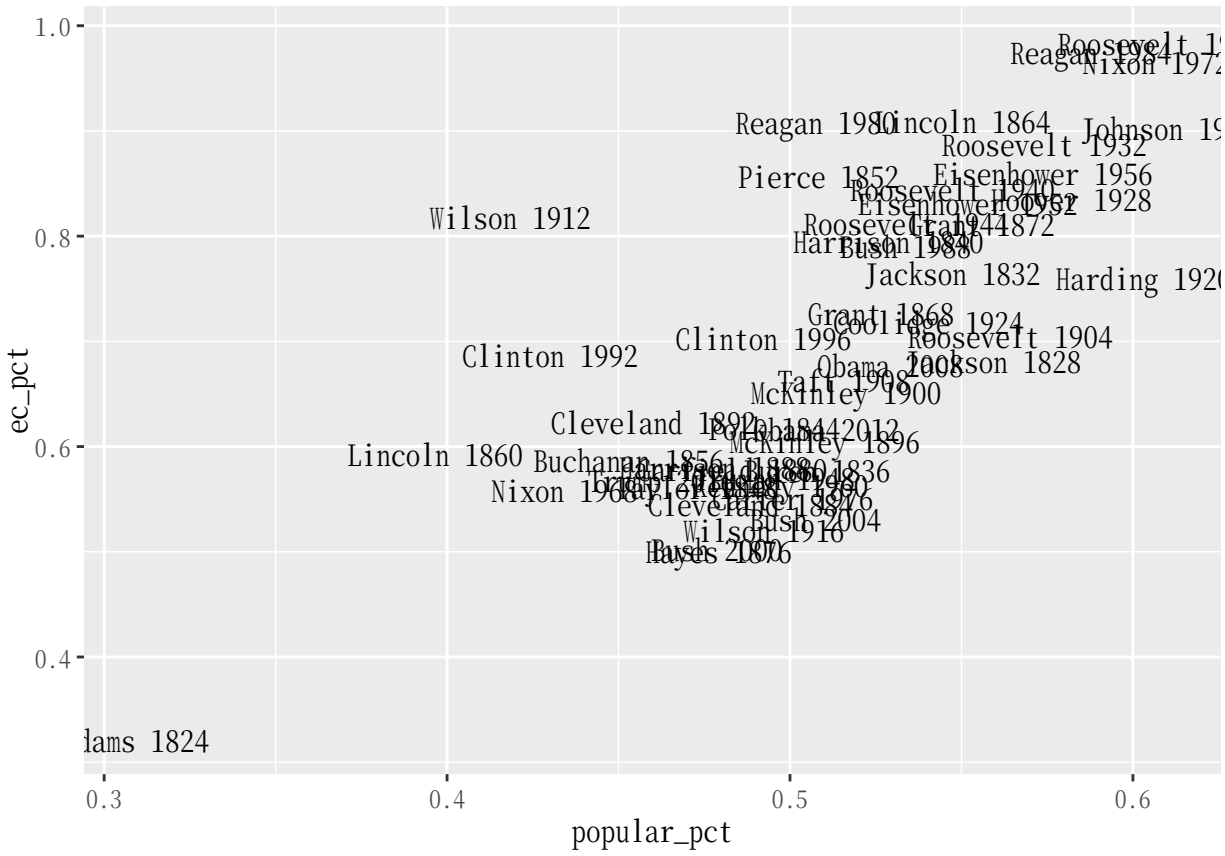
```
elections_historic |>
  select(2:7) |> head(10)
```

```
## # A tibble: 10 x 6
##     year winner                 win_party ec_pct popular_pct popular_margin
##    <int> <chr>                  <chr>      <dbl>       <dbl>          <dbl>
##  1  1824 John Quincy Adams      D.-R.      0.322       0.309        -0.104
##  2  1828 Andrew Jackson         Dem.       0.682       0.559         0.122
##  3  1832 Andrew Jackson         Dem.       0.766       0.547         0.178
##  4  1836 Martin Van Buren       Dem.       0.578       0.508         0.142
##  5  1840 William Henry Harrison Whig       0.796       0.529         0.0605
##  6  1844 James Polk             Dem.       0.618       0.495         0.0145
##  7  1848 Zachary Taylor         Whig       0.562       0.473         0.0479
##  8  1852 Franklin Pierce        Dem.       0.858       0.508         0.0695
##  9  1856 James Buchanan         Dem.       0.588       0.453         0.122
```

```
## 10  1860 Abraham Lincoln        Rep.        0.594        0.396          0.101
```

取 popular_pct(popular 投票支持率) 为横坐标，取 ec_pct(election college 投票支持率) 为纵坐标，将历次结果标在坐标系中：

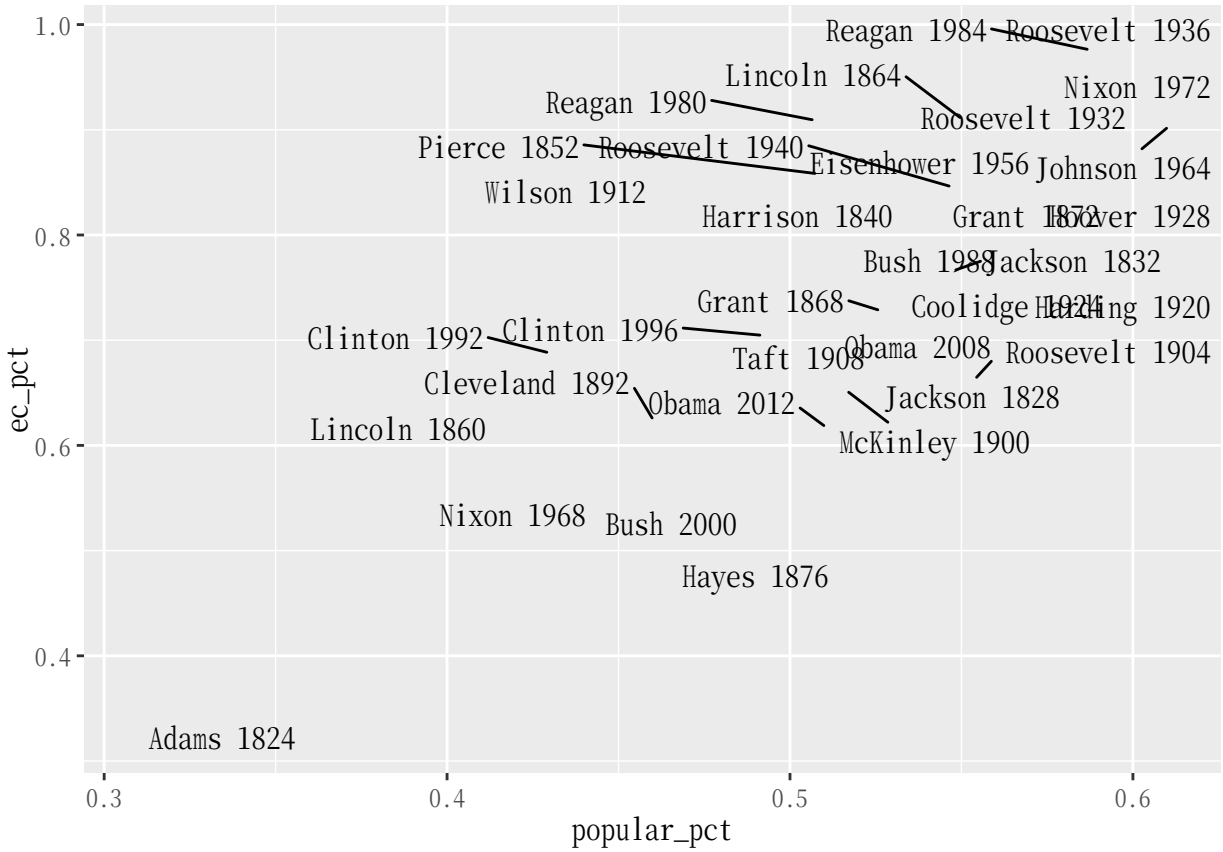
```
p <- ggplot(elections_historic, aes(
      x = popular_pct,
      y = ec_pct,
      label = winner_label))
p + geom_text()
```



因为点比较多，文字也比较长，有很多重叠。ggrepel 包的 geom_text_repel() 则很好地处理了这个问题：

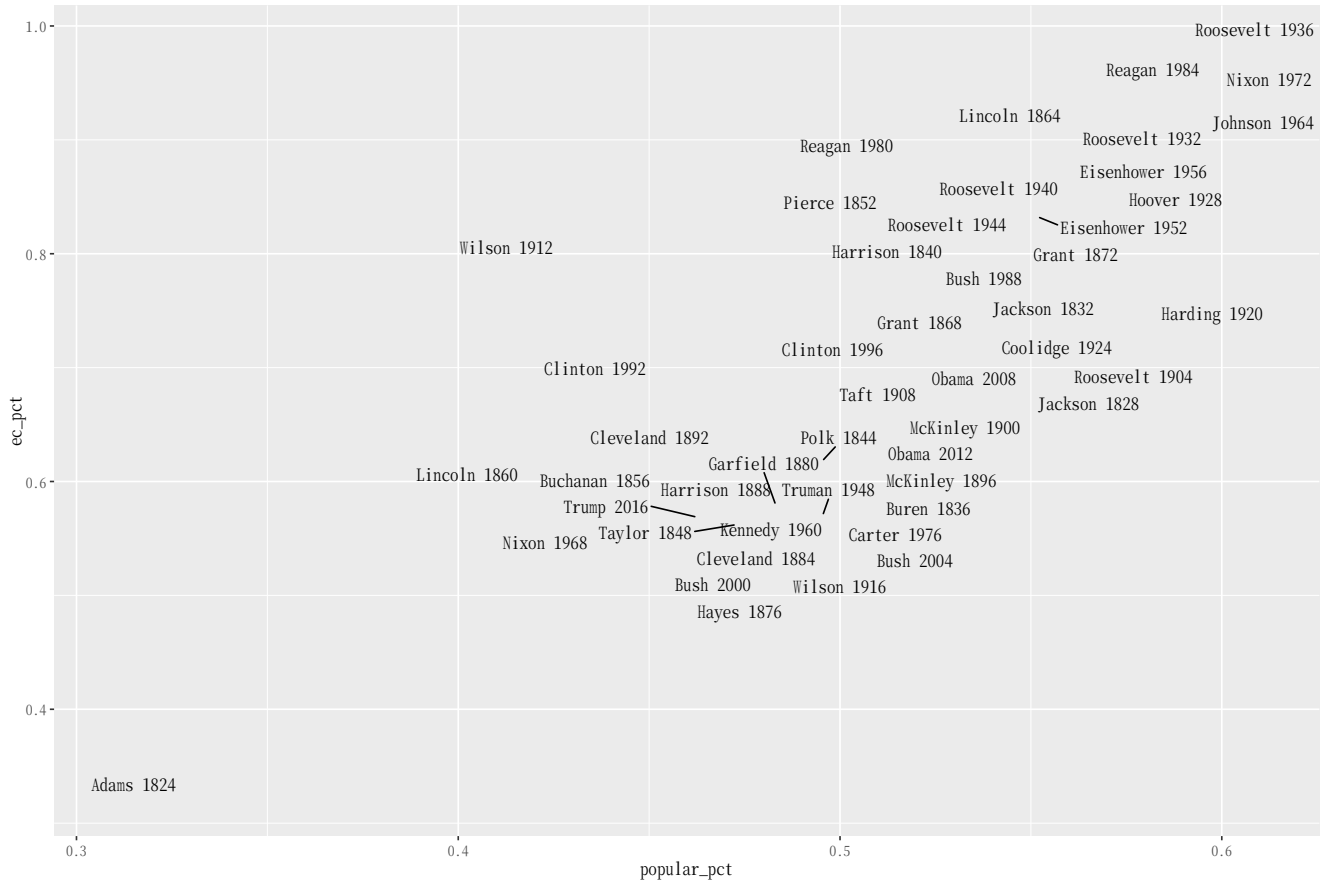
```
library(ggrepel)
p + geom_text_repel()
```

```
## Warning: ggrepel: 16 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



可以看出，其解决重叠问题的方式是用短线指向实际的坐标位置。在 Rmd 文件中，还是有少量的重叠，可以通过在 R 代码段选项中增大 `fig.width` 和 `fig.height` 参数实现，下面的代码段用了选项 `fig.width=12, fig.height=8`：

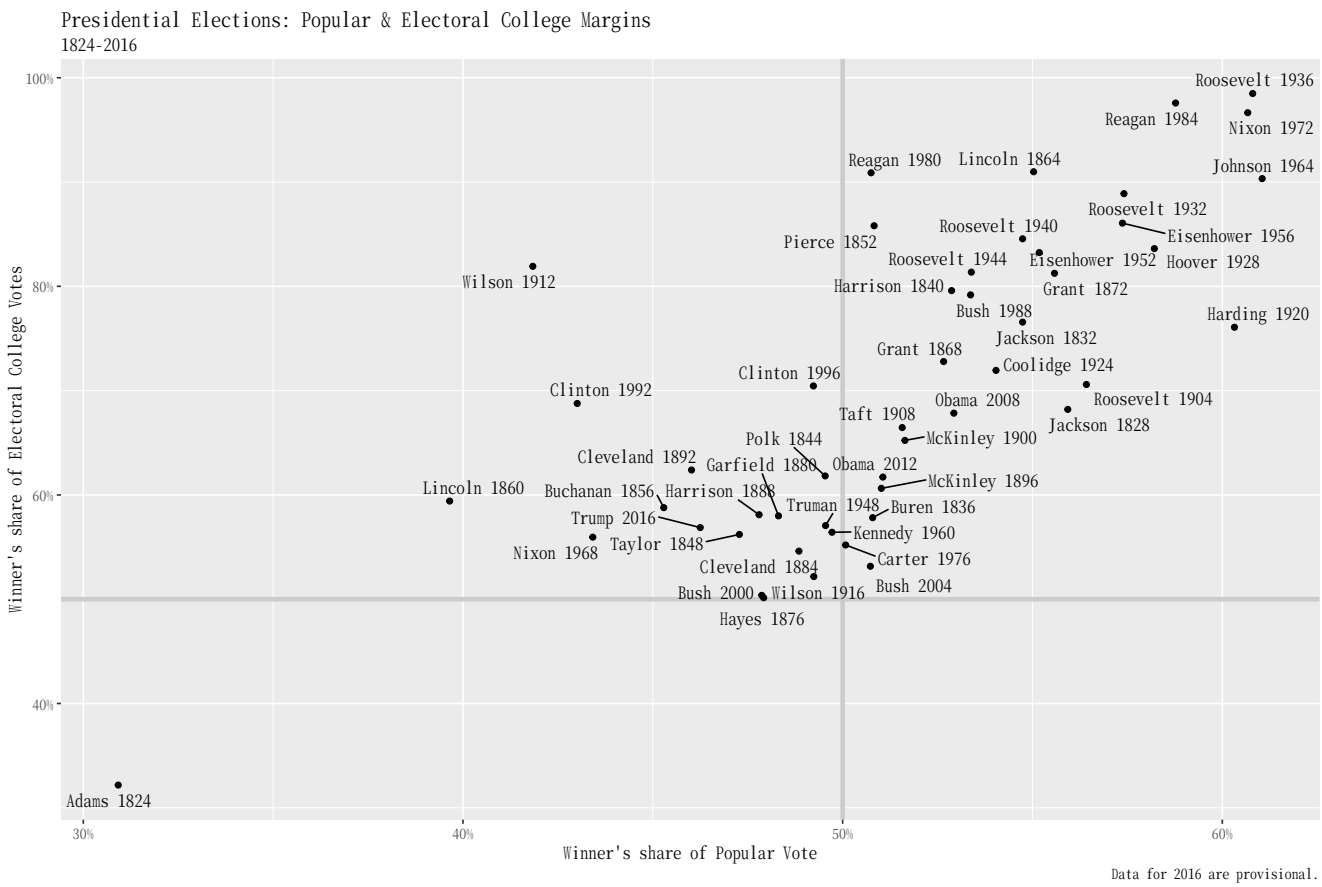
```
p + geom_text_repel()
```



可以用 `scale_x_continuous()` 和 `scale_y_continuous()` 将坐标轴的比例值转换成百分数,用 `geom_hline(yintercept)` 添加横线，用 `geom_vline(xintercept)` 添加竖线，适当地用标注改善图形：

```
p +
  geom_hline(
    yintercept = 0.5,
    size = 1.4,
    col = "gray80") +
  geom_vline(
    xintercept = 0.5,
    size = 1.4,
    col = "gray80") +
  geom_point() +
  geom_text_repel() +
  scale_x_continuous(
    labels = scales::percent) +
  scale_y_continuous(
    labels = scales::percent) +
  labs(
```

```
    x = "Winner's share of Popular Vote",
    y = "Winner's share of Electoral College Votes",
    title =
      "Presidential Elections: Popular & Electoral College Margins",
    subtitle = "1824-2016",
    caption = "Data for 2016 are provisional."
  )
```



为了画斜线，可以用 `geom_abline()` 函数。

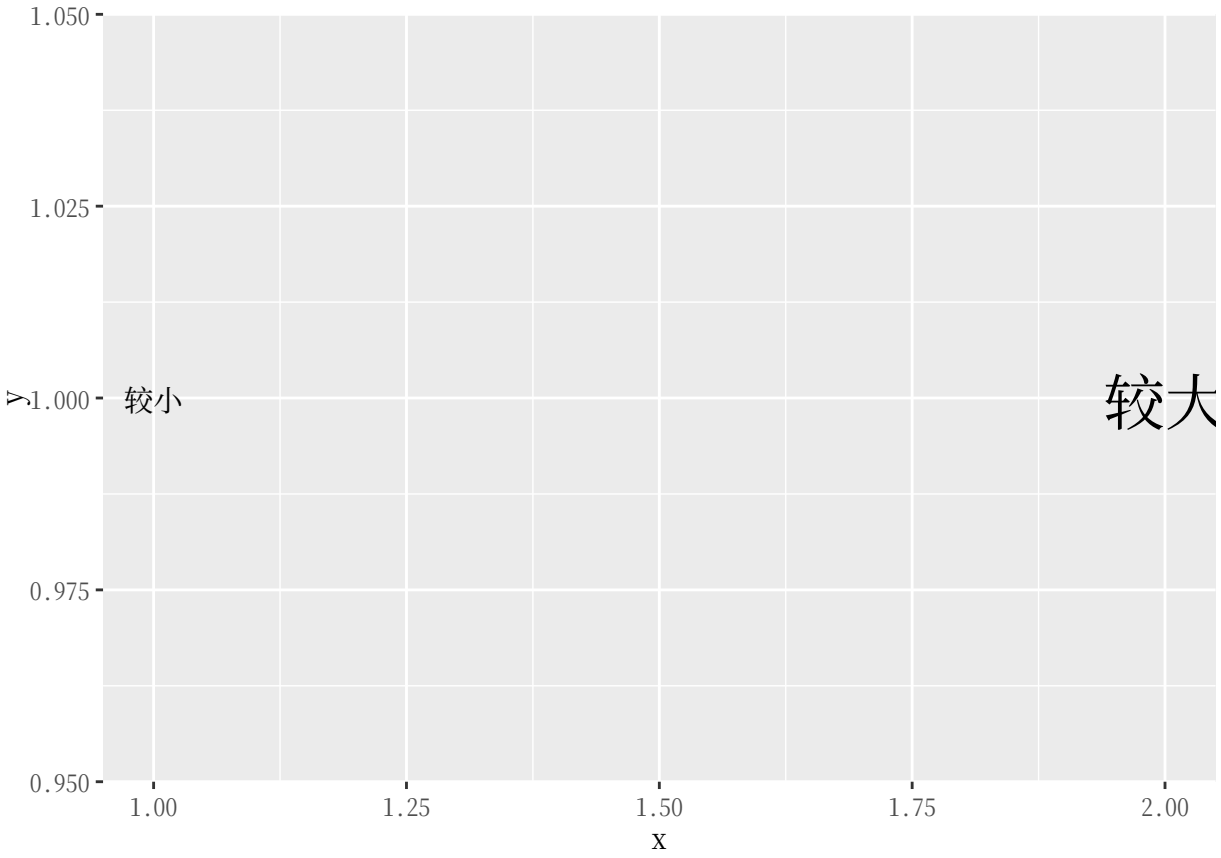
27.8.3 文字选项

用 `geom_text()` 添加文字时，可以使用一些图形选项。

可以用 `size` 选项（不是 aes 映射）指定字符大小，单位为毫米。如：

```
d <- tibble(
  x = 1:2,
  y = c(1,1),
```

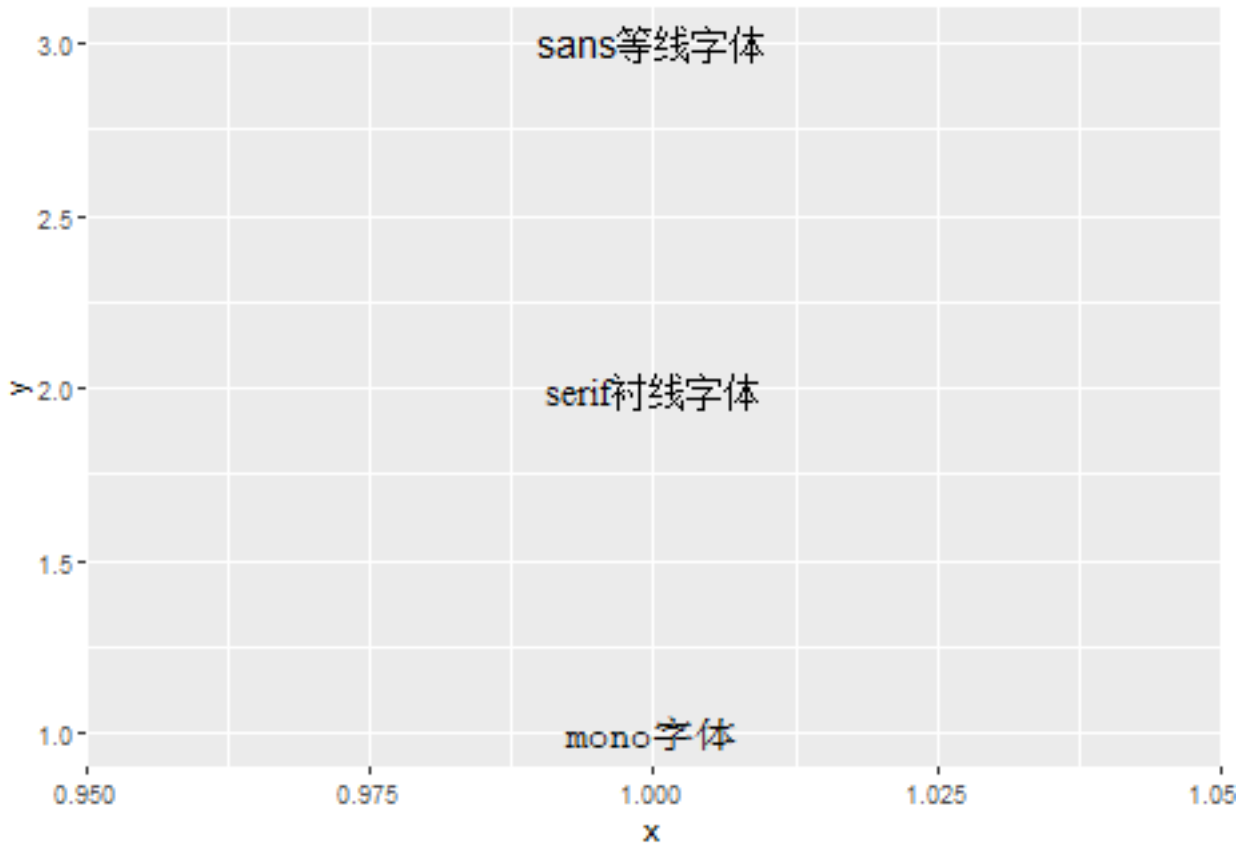
```
  label = c(" 较小", " 较大"))
ggplot() +
  geom_text(
    data = d[1,],
    mapping = aes(
      x = x, y = y, label = label),
    size = 4) +
  geom_text(
    data = d[2,],
    mapping = aes(
      x = x, y = y, label = label),
    size = 8)
```



文字的大小是按毫米计算的，不是按照坐标系中单位计算的，而且放大或者缩小坐标系时文字大小不变。程序不自动判断文字是否超出坐标系边界，所以有时需要人为将坐标范围适当放大。

可以用 `family` 选项选择不同的字体种类，在所有操作系统与绘图输出中都可选的种类包括`"sans"`(这是缺省值，等线字体)、`"serif"`(衬线字体，有笔画粗细的字体)、`"mono"`(类似于打字机字体)。`family` 可以作为选项，也可以作为映射，如：

```
d <- tibble(
  x = 1,
  y = 3:1,
  label = c("sans 等线字体", "serif 衬线字体", "mono 字体"),
  family = c("sans", "serif", "mono"))
ggplot() +
  geom_text(
    data = d,
    mapping = aes(
      x = x, y = y,
      label = label,
      family = family),
    size = 5)
```



要使用其它字体则比较复杂，可参考扩展包 showtext 和 extrafonts。

在 R Markdown 文件中作图使用中文字体要更麻烦一些，很容易变成乱码，需要在文件开始设置：

```
pdf.options(height=10/2.54, width=10/2.54, family="GB1")
```

其中 `height` 和 `width` 选项可以修改为其它高度、宽度值，单位是英寸，上例为宽 10 厘米高 10 厘米。如果还是

有乱码，可以在作图的代码段中指定选项 `dev="png"`，使用栅格图而不是 PDF 默认的矢量图作为输出。

可以用 `fontface` 映射或选项指定`"plain"`(缺省值)、`"bold"`(粗体)、`"italic"`(斜体)。

可以用 `hjust` 映射或选项指定文字的横向对齐方式，包括`"left"`、`"center"`、`"right"`、`"inward"`、`"outward"`。可以用 `vjust` 映射或选项指定文字的纵向对齐方式，可取`"bottom"`、`"middle"`、`"top"`、`"inward"`、`"outward"`。默认为居中对齐。`"inward"` 可以使文字向中心移动，避免边界的文字出界。

可以用 angle 映射或选项指定文字的旋转方向。

可以用 `nudge_x` 和 `nudge_y` 选项指定对每条标签文字的横向和纵向微调，单位是坐标系中的单位。

可以用选项 `check_overlap = TRUE` 要求重叠的文字不全显示出来，使得结果不重叠。

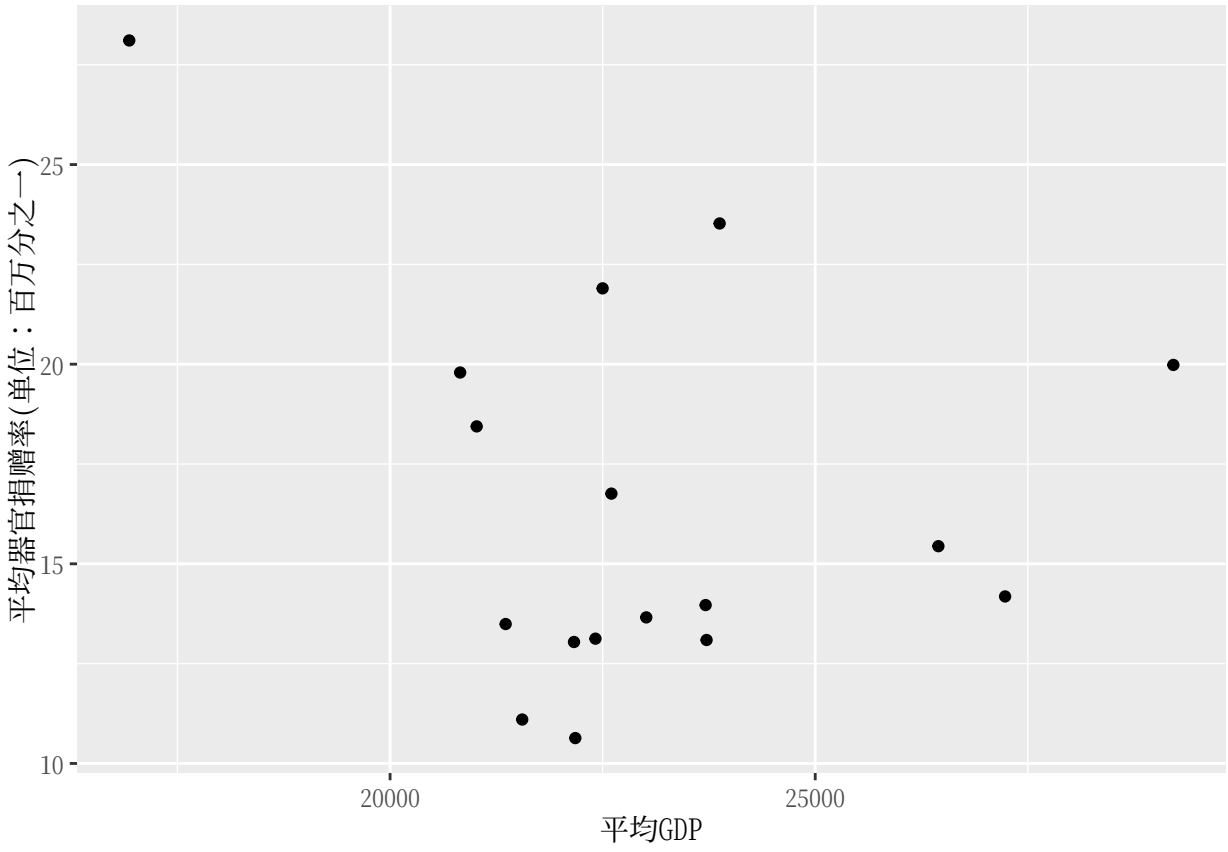
`geom_text()` 函数的一个变种是 `geom_label()`，标签文字会有一个圆角的背景框，可以指定填充颜色。

27.8.4 标出特殊点

在坐标系中标注文字的功能更经常用来标出图形中的特殊点。

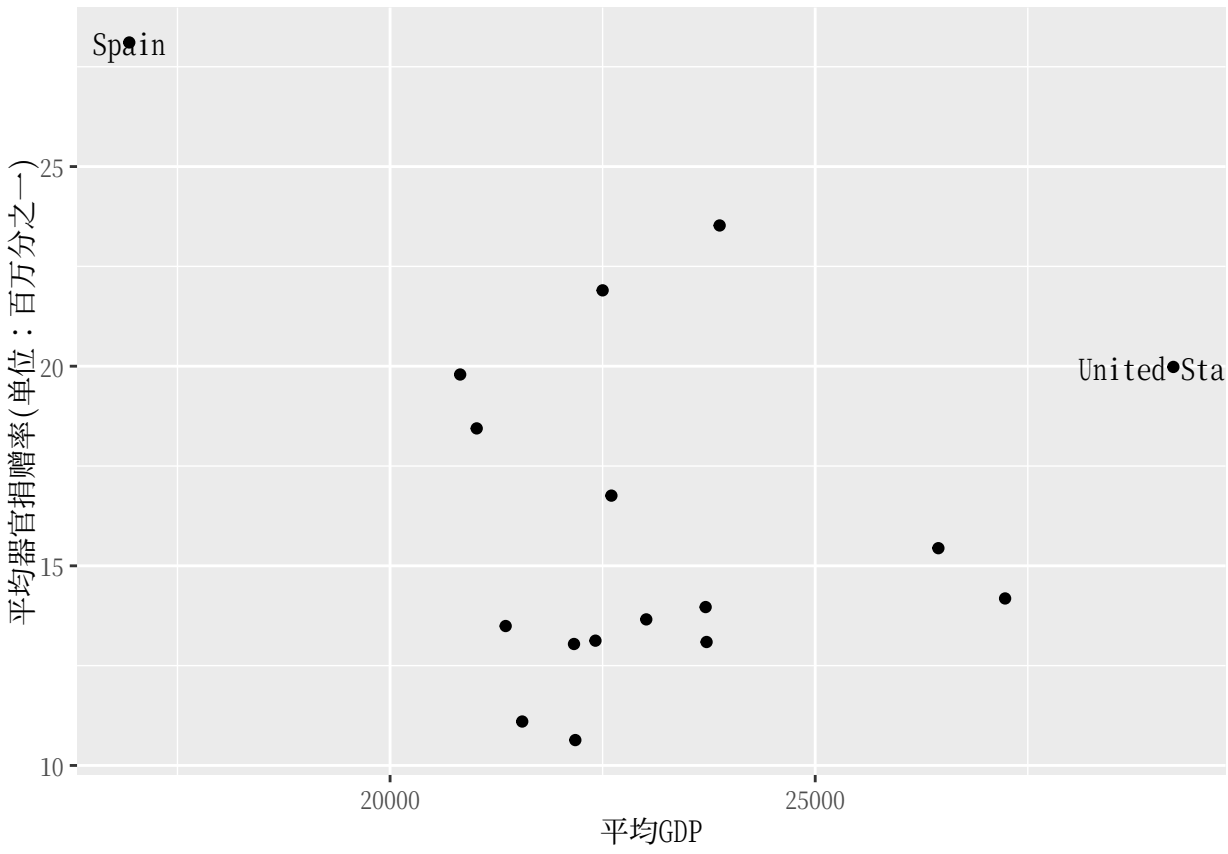
考虑 organdata 中各国的平均捐赠率数据。作平均捐赠率对平均 gdp 的散点图：

```
organdata4 <- organdata |>
  group_by(country) |>
  summarise(
    donors_mean = mean(donors, na.rm=TRUE),
    gdp_mean = mean(gdp, na.rm=TRUE),
    .groups = "drop"  )
p <- ggplot(organdata4, aes(x = gdp_mean, y = donors_mean))
p + geom_point() +
  labs(
    x = " 平均 GDP",
    y = " 平均器官捐赠率 (单位: 百万分之一)")
```



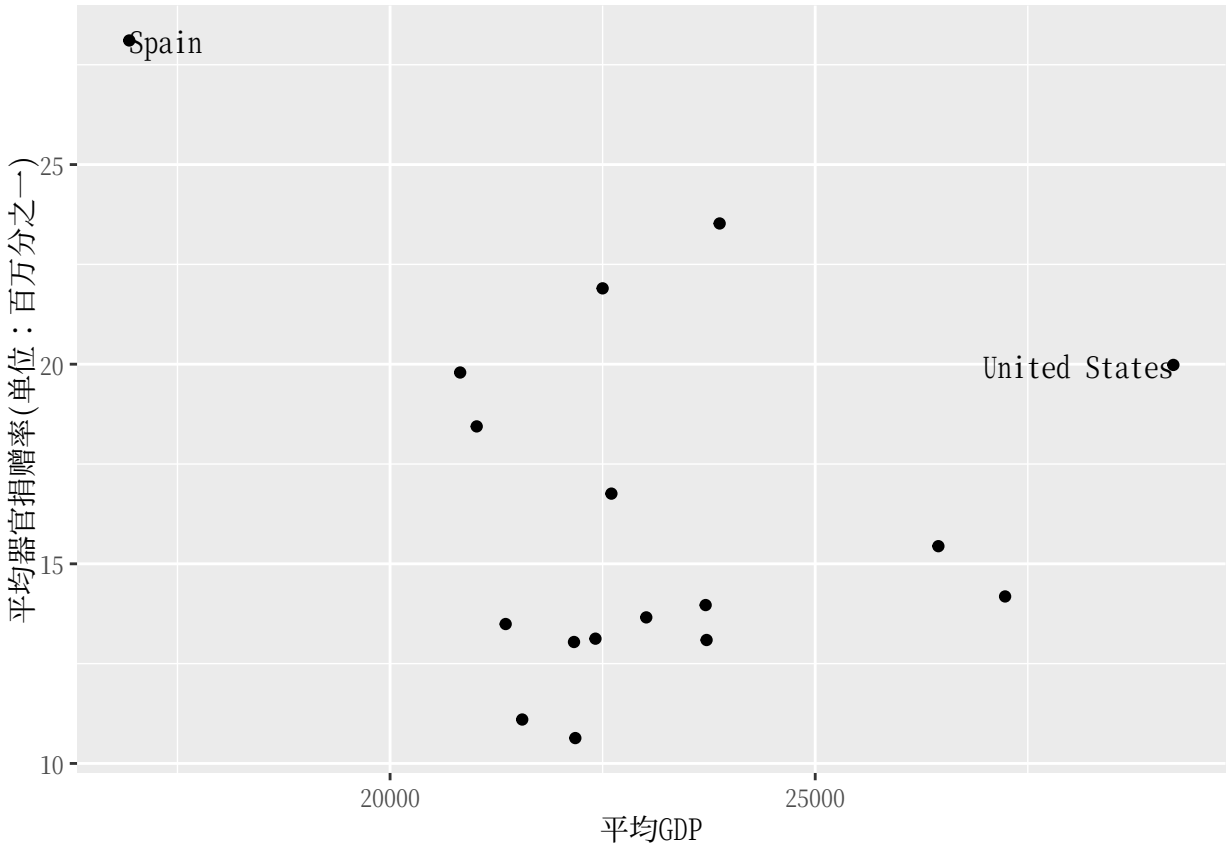
如果需要标出其中的特殊点，就要生成一个数据子集，并在 `geom_text()` 中指定输入数据为此子集：

```
p + geom_point() +
  geom_text(
    data = dplyr::filter(
      organdata4,
      gdp_mean > 27500 | donors_mean > 25),
    mapping = aes(label = country)) +
  labs(
    x = " 平均 GDP",
    y = " 平均器官捐赠率 (单位: 百万分之一)")
```



上面标的文字有超出边界的问题，可以在 `geom_text()` 中加选项 `hjust = "inward"`:

```
library(ggrepel)
p + geom_point() +
  geom_text(
    data = dplyr::filter(
      organdata4, gdp_mean > 27500 | donors_mean > 25),
    mapping = aes(label = country),
    hjust = "inward") +
  labs(
    x = " 平均 GDP",
    y = " 平均器官捐赠率 (单位: 百万分之一)")
```



27.9 刻度 (scale)

在 `ggplot()` 的 `mapping` 参数中指定 `x` 维、`y` 维、`color` 维等，实际上每一维度都有一个对应的默认刻度 (scale)，即将数据值映射到图形中的映射方法。如果需要修改刻度对应的变换或者标度方法，可以调用相应的 `scale_xxx()` 函数。

刻度在绘图过程中依次起到三个作用：

- 某些刻度对原始数据进行变换，比如对数变换；
- 如果有多个图层，比如既有散点图又有曲线拟合，各图层的坐标范围要统一考虑，如果不人为指定范围会自动设定范围使得各个图层的元素都可以容纳在刻度范围内；
- 将数据变量映射到具体的位置、颜色、填充色、大小、符号等。

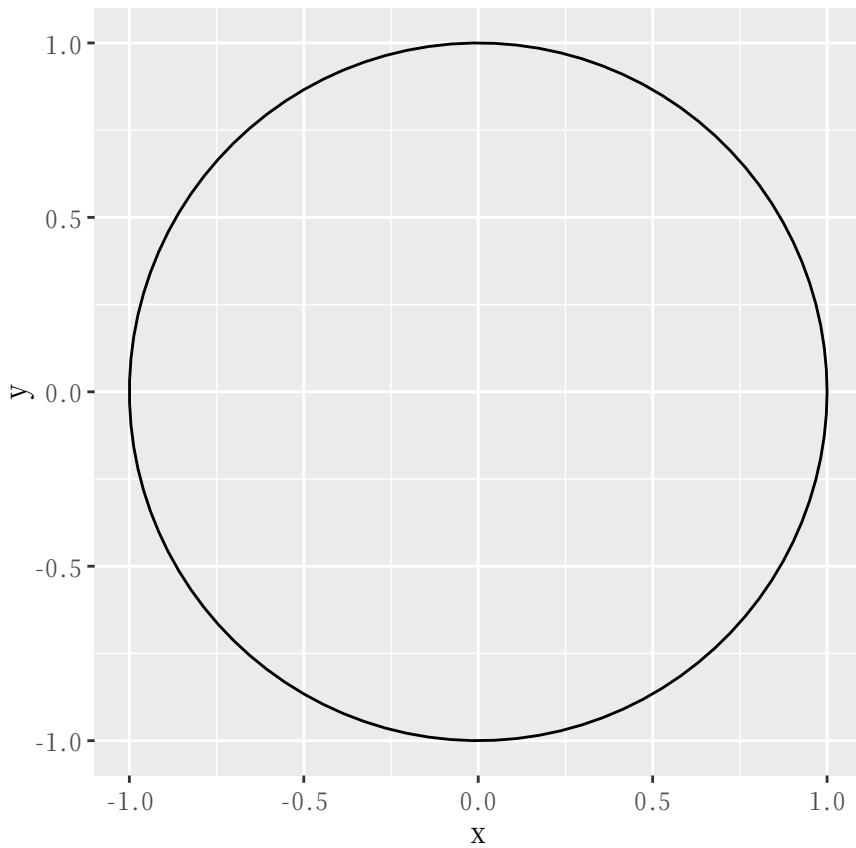
可以映射的维度包括 `x`、`y`、`color`、`fill`、`shape`、`size` 等。其中 `x`、`y` 维度多用于表示连续变量，但是也可以用于分类变量，如 Cleveland 点图就是将 `y` 维度分配给一个分类变量。`color`、`fill` 可以用于无序的分类变量，用于有序变量，也可以用于连续变量。`shape` 只能用于无序的分类变量。

数值或者不同类别可以用平面上的不同位置表示，一般使用直角坐标系，有一对相互垂直的 x 轴和 y 轴，但也可以有其他选择，比如 y 轴可以不与 x 轴垂直，某个轴的刻度可以不是线性的而是经过对数变换的，可以用极坐标系，等等。普通的直角坐标系不需要用 `scale_xxx()` 函数。

直角坐标系中的位置有 x 坐标与 y 坐标，可以分别代表两个变量，这两个变量经常是不同单位的，比如，身高与体重。这时，x 轴与 y 轴的数值之间没有可比性，两个轴的数值范围与轴的实际长度只要不过于极端就没有什么关系。但是，应该尽可能使用比较合适的高宽比。

如果两个轴的变量含义相同，最好使用完全相同的坐标轴范围与坐标轴长度，使得水平与垂直方向上的等长距离在对应的坐标轴上也代表相等的距离。用 `coord_fixed()` 函数指定两个轴的单位为 1:1 长度的坐标系，其中参数 xlim 和 ylim 可以用来指定坐标范围。也可以用 ratio 参数指定一个其他的宽高比。如

```
d <- tibble(
  t = seq(0, 2*pi, length.out=100),
  x = cos(t),
  y = sin(t)
)
ggplot(d, aes(
  x = x, y = y)) +
  geom_path() +
  coord_fixed()
```



上面的程序中，`geom_path()` 按照输入数据集的次序将坐标点连接在一起。如果改用 `geom_line()`，会将数据先按照 `x` 坐标值排序然后再顺序相连。可以看出，结果是一个正圆；如果不使用 `coord_fixed()`，结果不一定是正圆。

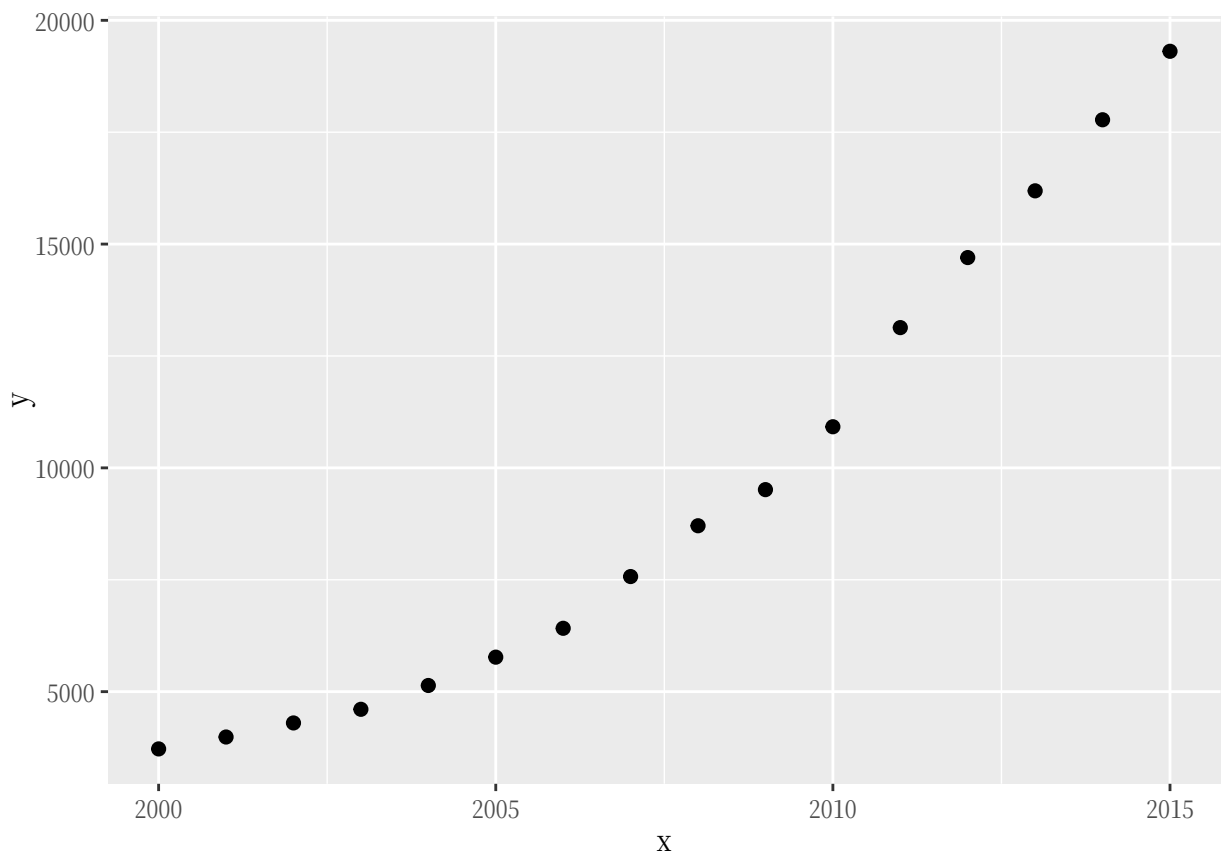
在 `coord_fixed()` 函数中，还可以用 `expand = FALSE` 要求坐标轴范围严格等于数据范围或 xlim 和 ylim 的规定，否则会比数据范围略宽一些。

`scale_x_log10()` 和 `scale_y_log10()` 可以将 x 轴或者 y 轴用对数刻度，这实际上是将数据做常用对数变换，但是相应的坐标轴刻度的数值还标成变换之前的原始值。对数轴方向相同的距离代表相差相同的倍数。经济、金融数据常常需要用对数刻度。

对数轴最好用来代表比例，尤其是使用条形图时，应该从 1 开始而不是从 0 开始。

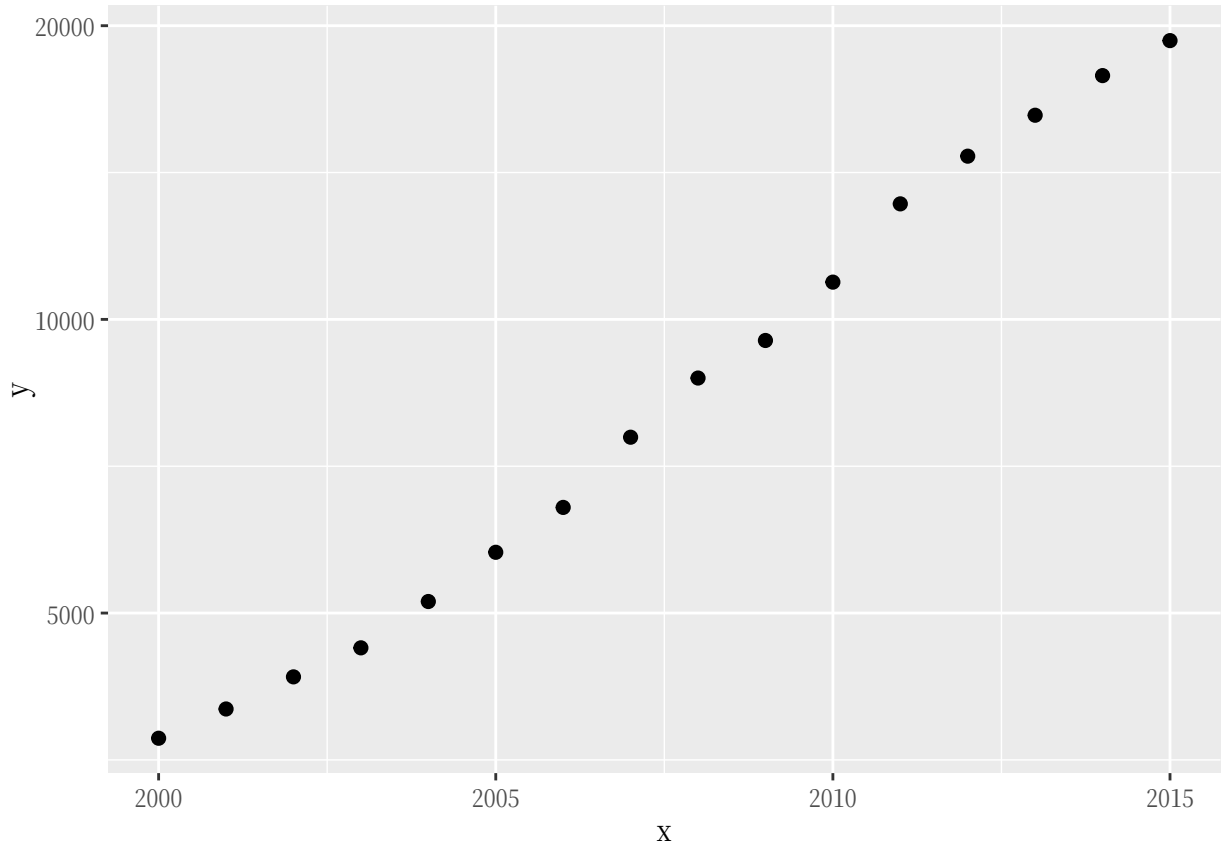
下图是我国 2000-2015 年的居民消费水平（元），可以看出，增长是指数型的：

```
d <- tibble(
  x = 2000:2015,
  y = c(3721L, 3987L, 4301L, 4606L, 5138L, 5771L, 6416L, 7572L, 8707L,
        9514L, 10919L, 13134L, 14699L, 16190L, 17778L, 19308L))
p <- ggplot(d, aes(
  x = x, y = y))
p + geom_point(size = 2)
```



将 y 轴用对数刻度，则散点可以呈现为线性：

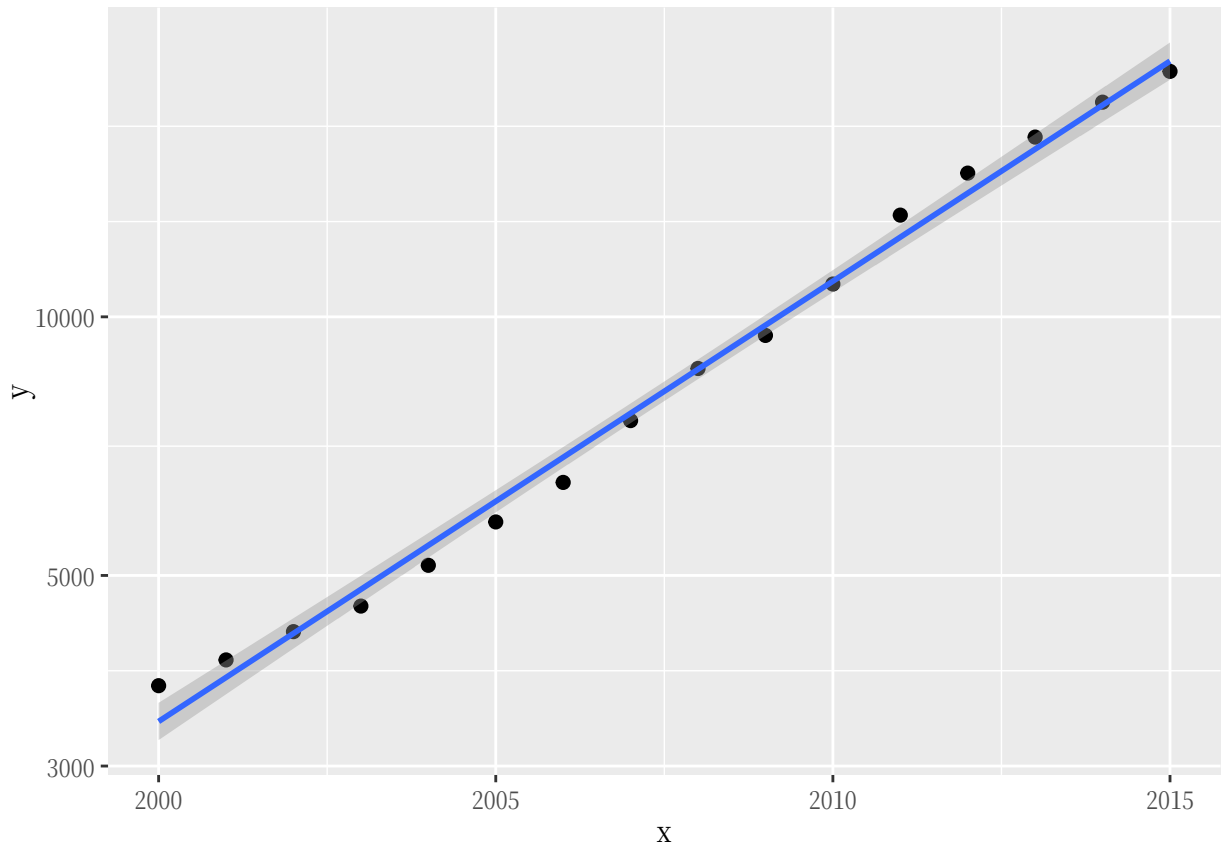
```
p + geom_point(size = 2) +
  scale_y_log10()
```



用 geom_smooth() 添加线性拟合线:

```
p + geom_point(size = 2) +
  geom_smooth(method="lm") +
  scale_y_log10()
```

```
## `geom_smooth()` using formula 'y ~ x'
```

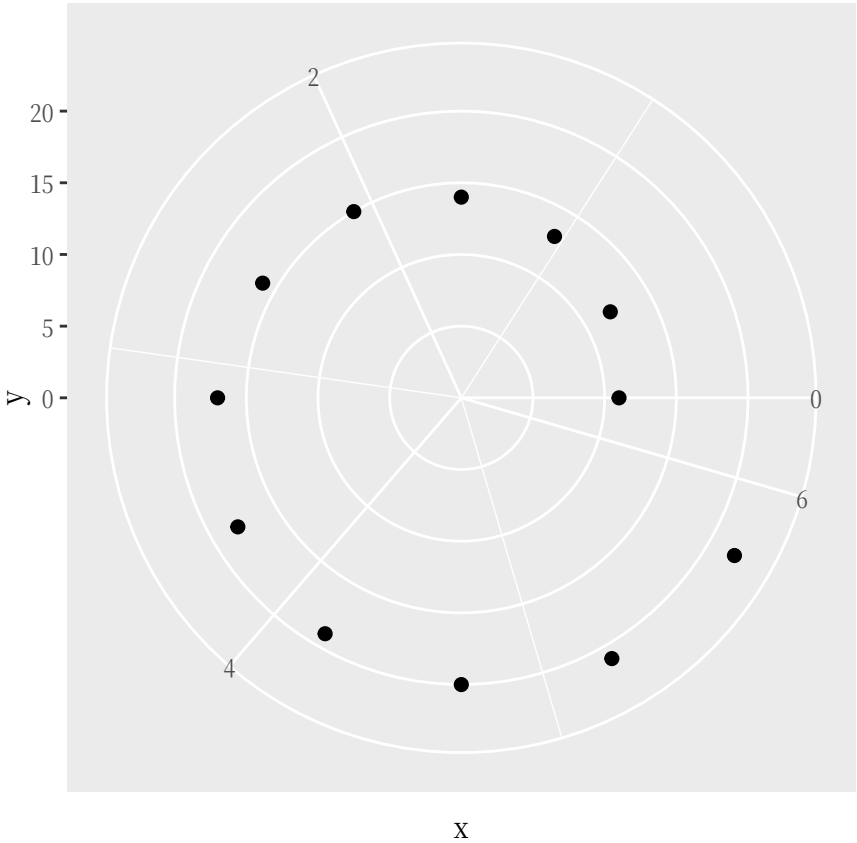


注意 `geom_smooth(method="lm")` 在计算拟合时会自动采用 `log10(y)` 的值。

`scale_x_sqrt()` 和 `scale_y_sqrt()` 与对数轴类似，做的是平方根变换。对于面积值，这种变换有一定意义。

用 `coord_polar()` 指定极坐标系，用 `theta="x"` 或者 `theta="y"` 指定那一维映射到极角。如：

```
d <- tibble(
  x = (0:11)/12*2*pi,
  y = 11:22
)
p <- ggplot(data=d, mapping = aes(
  x = x, y = y))
p + geom_point(size=2) +
  scale_x_continuous(limits=c(0, 2*pi)) +
  scale_y_continuous(limits=c(0, 22)) +
  coord_polar(theta="x", start=-pi/2, direction=-1)
```



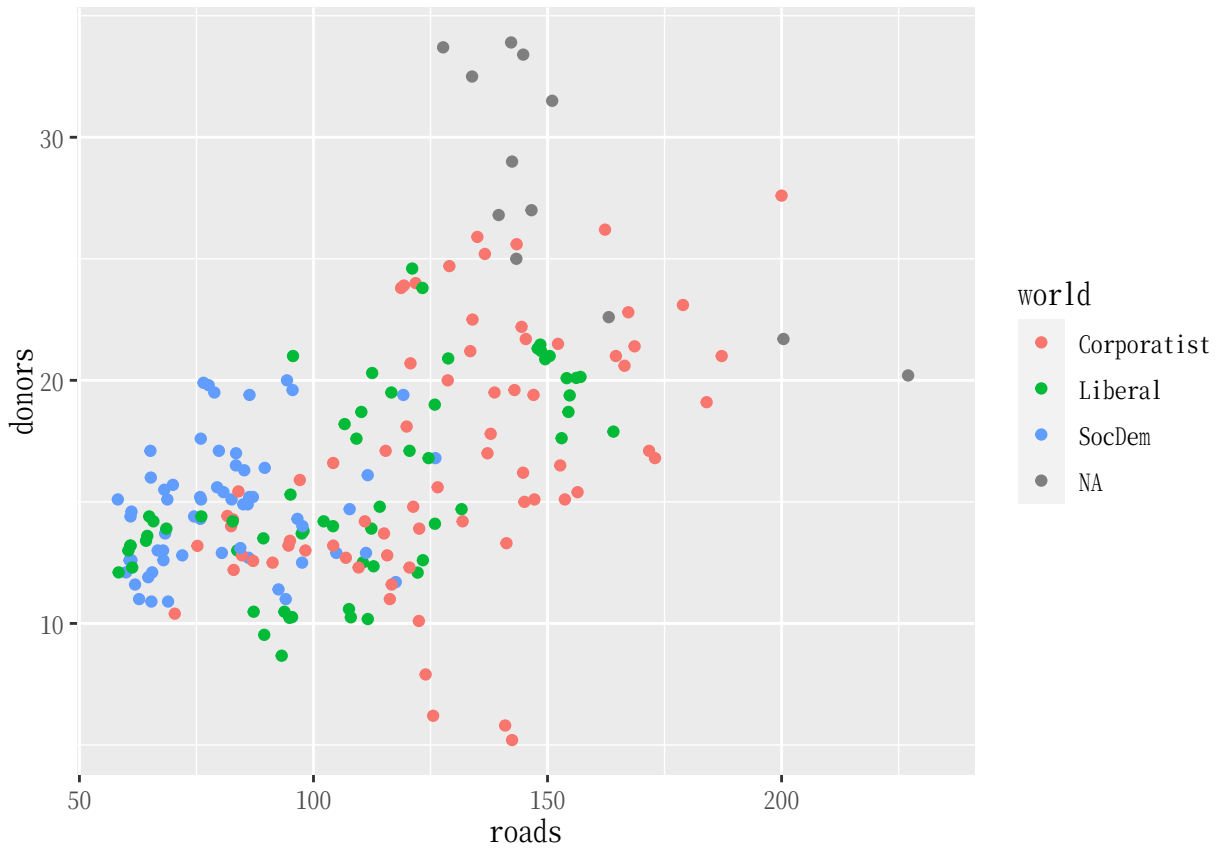
上图中 y 轴为极径的刻度，圆周上的数字表示极角刻度。程序中 `start` 指定极角为 0 的射线与 12 点方向的夹角弧度，`direction=1` 表示顺时针计算角度，`direction=-1` 表示逆时针。`scale_continuous_x()` 和 `scale_continuous_y()` 指定了两个维度的坐标范围。

用 `scale_xxx()` 函数指定非默认的刻度，一般模式为 `scale_` 映射维度 `_` 类型 `()`，“类型” 包括 `continuous`、`discrete`、`log10` 等。可以用来人为指定坐标轴的刻度值，修改 `color` 或者 `fill` 维度所利用的颜色表，等等，用其中的参数指定这些内容。注意坐标轴的标签（标题）用 `labs()` 函数指定，而不是用 `scale_xxx()` 函数指定。

例如，在 organdata 中，作 `donors` 对 `roads`(每十万人的道路交通事故死亡率) 的散点图，并按 `world`(不同福利类型) 对散点染色，x、y、color 这三个维度都用了默认的刻度:

```
p <- ggplot(organdata, aes(
  x = roads,
  y = donors,
  color = world ))
p + geom_point()
```

```
## Warning: Removed 34 rows containing missing values (geom_point).
```

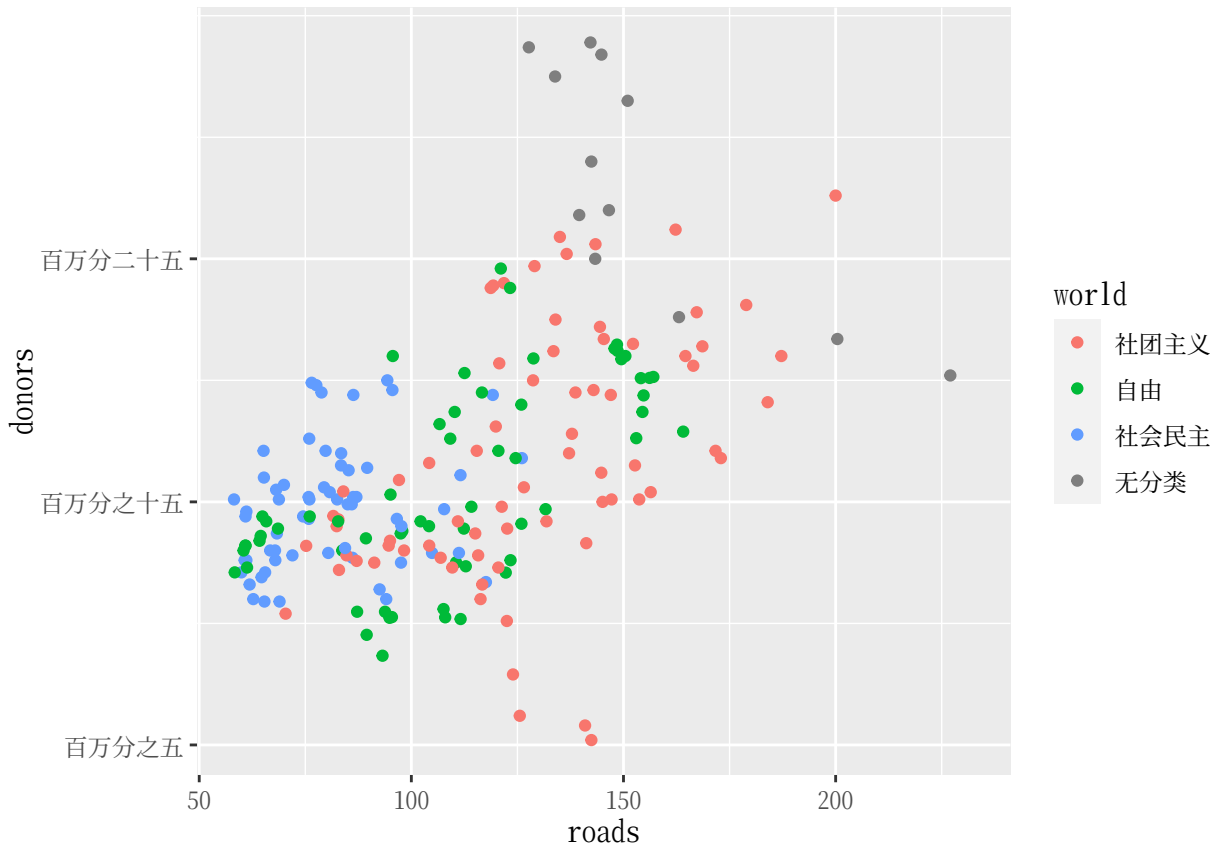


这里有三个维度：连续型的 x、y 维度与无序分类值的 color 维度。color 维度既可以取无序的分类型，也可以取有序或者连续型。x、y 维度有自动的坐标轴，color 维度有自动的图例在图形右侧。如果没有特殊要求，不必调用 scale_xxx() 函数。

下面将 y 轴的刻度值进行人为的规定，将 color 维的标签人为指定：

```
p + geom_point() +
  scale_y_continuous(
    breaks = c(5, 15, 25),
    labels = c(" 百万分之五", " 百万分之十五", " 百万分二十五")  ) +
  scale_color_discrete(labels = c(
    " 社团主义", " 自由", " 社会民主", " 无分类"  ))
```

```
## Warning: Removed 34 rows containing missing values (geom_point).
```



在 scale_ 维度 _contiuous() 函数中，可以用 breaks 指定坐标刻度标线的坐标数值表，用 minor_breaks 指定细刻度的坐标数值表，用 labels 指定在坐标刻度标线处标出的坐标值标签字符串表，用 limits 指定坐标系的范围（两个数的向量），用 expand 指定在将坐标轴范围比数据的范围在两侧分别扩充多少，取 c(0,0) 要求不扩充，默认会左右各扩充 5%。

如果仅要求规定坐标轴范围，也可以用 xlim() 函数和 ylim() 函数，参数为范围的最小值和最大值。

ggplot2 的散点图默认使用灰色背景并带有白色的网格线。theme() 函数的 panel.background 可以指定背景色，panel.grid、panel.grid.xxx 可以指定网格线做法。

27.10 如何使用颜色

可以用颜色来表示分组，比如，不同组的散点用不同颜色，多条曲线用不同颜色；可以用颜色表示数值，用颜色深浅表示绝对值大小；可以用颜色来突出某些要强调的图形元素。

将变量值映射为颜色，可以分为如下四种情况：

- 无序的分类变量；
- 有序的分类变量，仅有大小次序，没有正反；
- 有序的分类变量，有正有反；
- 连续数值变量。

表 27.1: 8 种对色盲可区分的颜色

name	code
black	#000000
orange	#E69F00
sky blue	#56B4E9
bluish green	#009E73
yellow	#F0E442
blue	#0072B2
vermilion	#D55E00
reddish purple	#CC79A7

27.10.1 名义型变量的颜色

将无序的分类值映射到颜色，应该使用完全不相像、很容易区分的颜色，各个颜色应该没有明显次序、没有哪一个与其他明显不同。可以用 RColorBrewer 扩展包提供的调色盘，ggsci 包也提供了一些调色盘。

在 ggplot2 中用 `scale_color_brewer(palette)` 和 `scale_fill_brewer(palette)` 选择 RColorBrewer 中的调色盘。图27.1为无序分类适用的调色盘。称这样的调色盘为名义型 (qualitative)。

色盲的人会分辨不出某些颜色，比如红绿色盲的人无法分辨红色和绿色，蓝绿色盲的人无法分辨蓝色和绿色。男性中有 8% 的人有色盲，所以绘图时应该考虑到这个问题。(Wilke, 2019) 提供了 8 种对色盲也可区分的颜色：

例如，作 gapminder 数据集中各国在 2007 年期望寿命对人均 GDP 的散点图，不同大洲使用不同的颜色，指定 ColorBrewer 的 Set1 调色板：

```
p <- ggplot(data = subset(gapminder, year == 2007),
  mapping = aes(
    x = gdpPercap,
    y = lifeExp))
p + geom_point(mapping = aes(color = continent)) +
  scale_color_brewer(palette = "Set1")
```

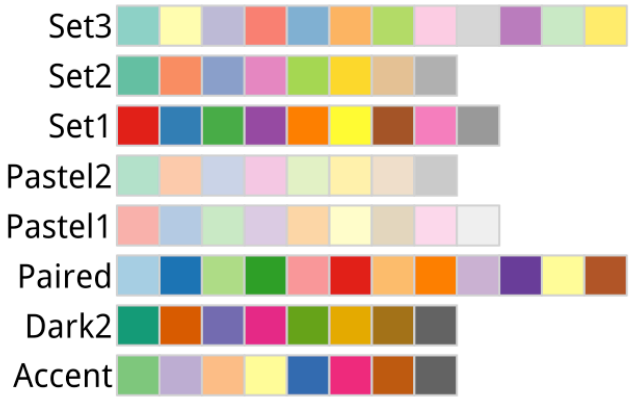
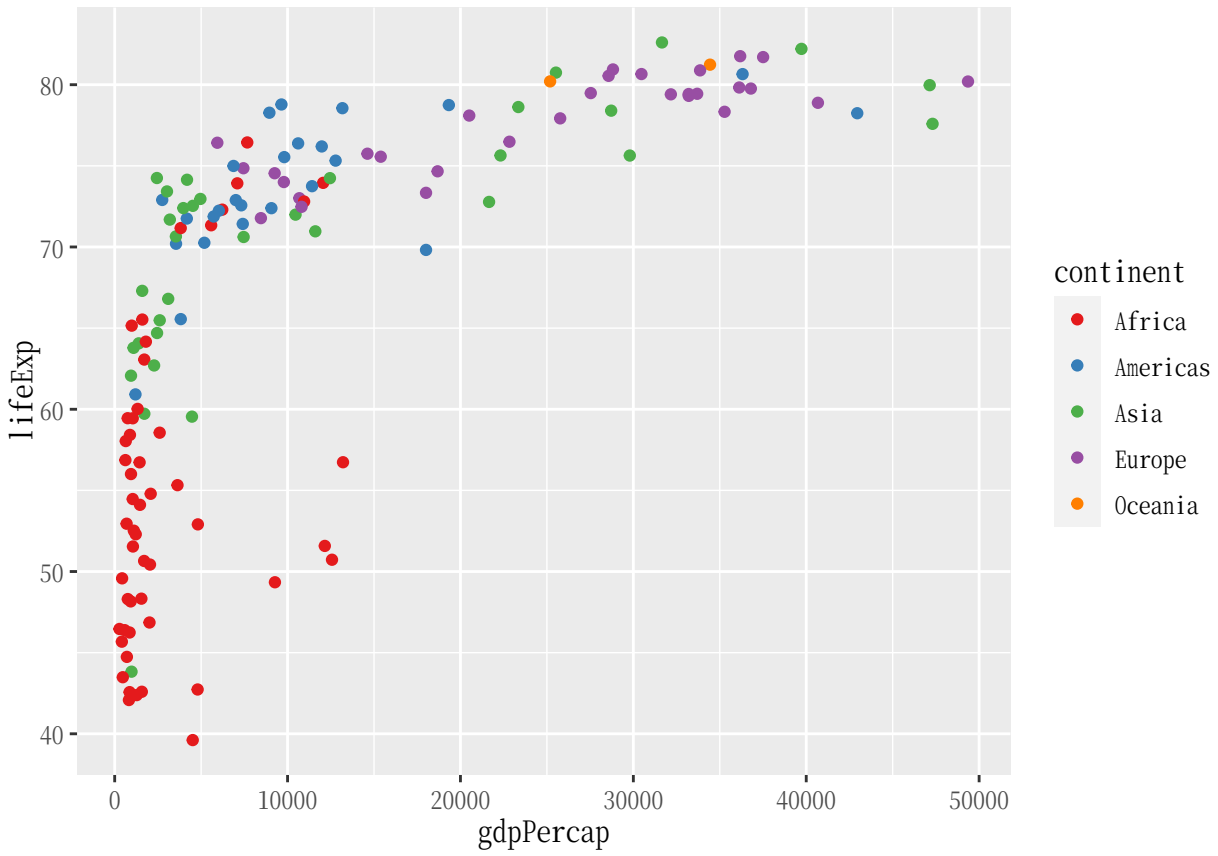


图 27.1: 无序分类适用的调色板



使用颜色进行强调时，可以将要强调的点、线、条形用鲜明的颜色，而非强调的用褪色的颜色。最简单的做法是仅对要强调的内容指定一个鲜明的颜色。

27.10.2 色块图

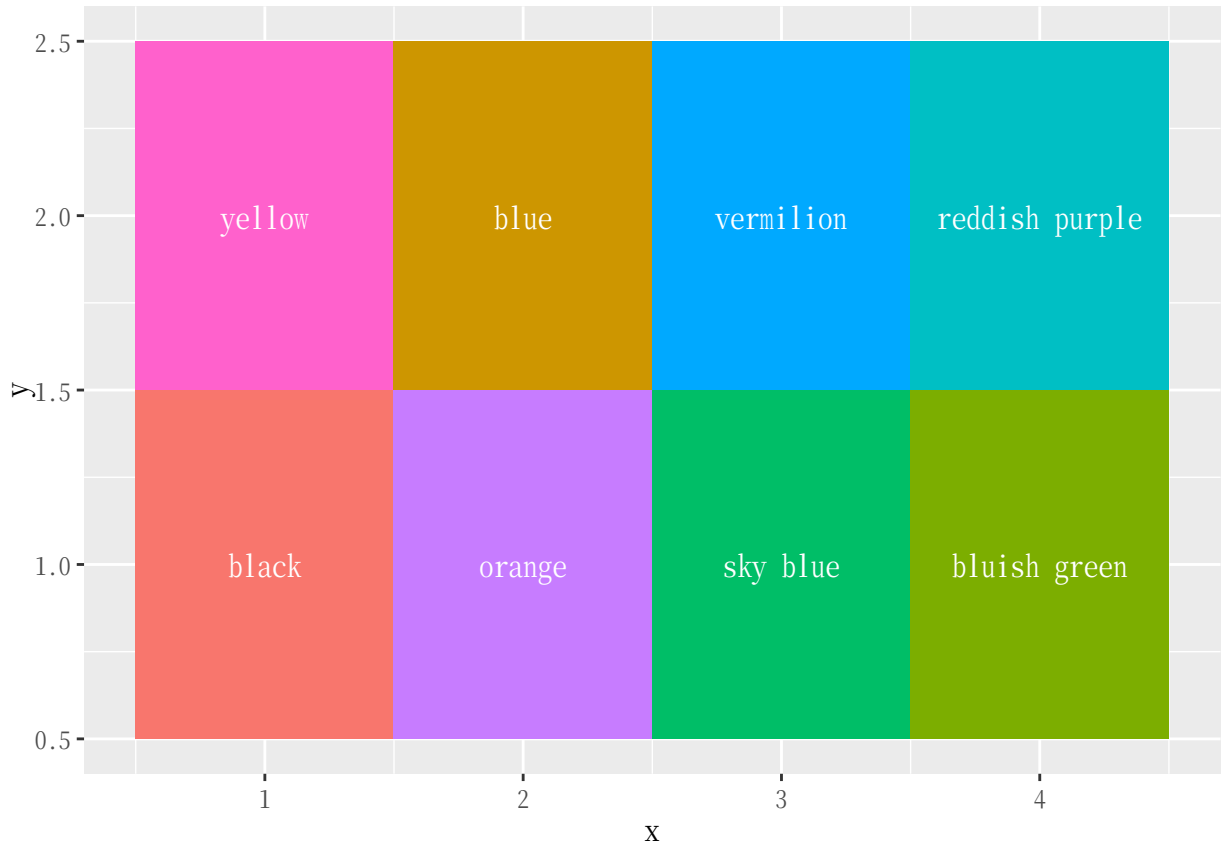
`geom_tile()`、`geom_rect()`、`geom_raster()` 可以用颜色画长方块。`geom_tile()` 指定每个长方块的中心位置 (x 和 y)、宽度 (width) 和高度，`geom_rect()` 指定每个长方块的四个角的左右和上限坐标 (xmin, xmax, ymin, ymax)，`geom_raster()` 是 `geom_tile()` 在长方块大小相同时的快速版本。

例如，表27.1中的颜色图：

```
d <- tibble(
  x = rep(1:4, 2),
  y = rep(1:2, each=4),
  fill = dcolorqua$code,
  label = dcolorqua$name)
p <- ggplot(data = d, mapping = aes(
  x = x, y = y, fill = fill, label = label))
p + geom_raster() +
```

```
  geom_text(color = "white") +
  guides(fill = FALSE)
```

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```



27.10.3 有序型变量的颜色

将连续的或者有序的值映射到颜色，应该使用渐变色，如果都是正值，可以使用从浅到深或者从深到浅的颜色，并且应该使用同一个颜色，只是深浅程度不同。图27.2为单向的有序分类适用的调色板。称这样的调色板为有序型 (sequential)。

27.10.4 相异型变量的颜色

如果颜色代表有正有负的数值（比如反对、中立、支持），则应该以浅色为接近零值，正值与负值分别用两个不同的颜色。另外，如果要代表的数值有明显的中间值，为了强调较低的值与较高的值的对比，也可以使用这样的颜色刻度。图27.3为有正有负的有序分类适用的调色板。称这样的调色板为相异型 (diverging)。相异型的渐变色有可能对色盲的人不可辨识，`Colorbrewer::PiYG` 刻度对色盲人群也可以分辨。

图 27.2: 有序单向分类适用的调色板

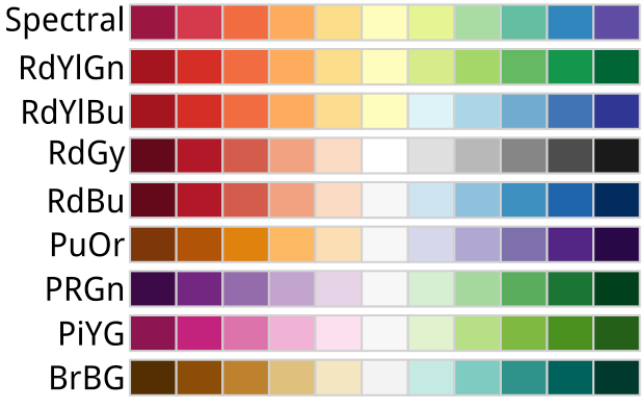


图 27.3: 有序正负分类适用的调色板

27.10.5 连续数值变量的颜色

可以将数值变量映射到颜色 (color 或者 fill 维),ggplot2 可以自动选择映射关系,也可以用 `scale_color_gradient()` 或 `scale_color_gradient()` 函数自己指定一个渐变色，只需要指定渐变色的 `low` 和 `high` 两段颜色，一般需要用颜色代码表示，R 扩展包 colourpicker 可以帮助挑选颜色。

对于只取非负值的变量,对应的渐变色应该是仅从浅到深或者从深到浅的,使用默认参数调用 `scale_color_gradient()` 产生这种效果；对于有正有负的变量，就应该使用两种截然不同的颜色表示正负值，用浅色表示接近 0 的值，使用默认参数调用 `scale_color_gradient2()` 产生这种效果。

27.11 标题、标注、指南

除了 `ggplot()` 指定数据与映射，`geom_xxx()` 作图，还可以用许多辅助函数增强图形。

- `labs()` 可以设置适当的标题和标签。
- `annotate()` 函数可以直接在坐标系内进行文字、符号、线段、箭头、长方形的绘制。
- `guides()` 函数可以控制图例的取舍以及做法。
- `theme()` 函数可以控制一些整体的选项如背景色、字体类型、图例的摆放位置等。

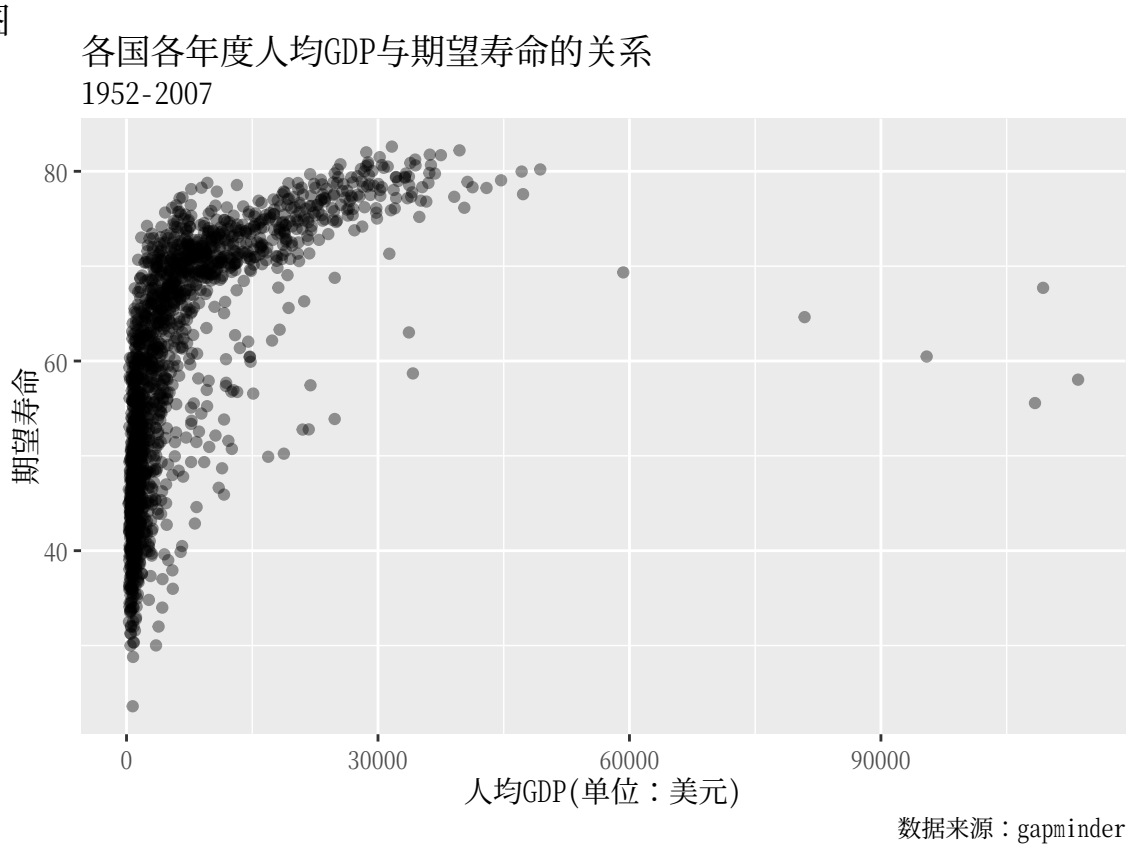
在需要修改图形时，如果修改会影响到相应的 `geom_xxx()` 的主要结果，一般需要在 `ggplot()` 或者该 `geom_xxx()` 函数中将适当的变量映射为某一维度，或者用 `scale_xxx()` 函数进行变换。如果仅仅是一些显示效果的修改，则一般作为 `geom_xxx()` 的选项，或者调用 `labs()`、`theme()`、`guides()` 完成。

27.11.1 标题

函数 `labs()` 可以用来指定图形上方的标题 (title)、副标题 (subtitle)、右下方的标注 (caption)、左上方的标签以及坐标轴标题和其它维的名称。例如：

```
ggplot(gapminder, aes(
    x = gdpPercap,
    y = lifeExp)) +
  geom_point(alpha = 0.4) +
  labs(
    title = " 各国各年度人均 GDP 与期望寿命的关系",
    subtitle = "1952-2007",
    tag = " 散点图",
    caption = " 数据来源: gapminder",
    x = " 人均 GDP(单位: 美元)",
    y = " 期望寿命"  )
```

散点图



在 `labs()` 中用 `x=`、`y=`、`color=` 之类的选项指定坐标轴的标签或者其它维的名称，如果该维用了 `scale_xxx()` 函数，则应该则 `scale_xxx()` 函数中用 `name=` 指定轴标签（名字）。数值型的维度除非是显然的应在标签中包含单位。`labs()` 中或者 `scale_xxx(name=)` 中如果指定文字为 `NULL` 则取消该标题或标签，但是取消坐标轴标签必须是取消标签后含义也显而易见，没有任何疑问的情况。

`labs()` 只是提供了这些标题功能，一般并不会同时使用这些功能。在出版图书内，图形下方一般伴随有图形说明，这时一般就不再使用标题、副标题、标签、标注，而只需写在图的伴随说明文字中，当然，坐标轴标签一般还是需要的。

有一点要注意，默认的标题、坐标轴标签、图例标签中的文字往往偏小，如果单独放大这些图形来看并没有问题，但是作为插图放在书中或者网页中就偏小了。可以用 `theme()` 函数调整字体大小。

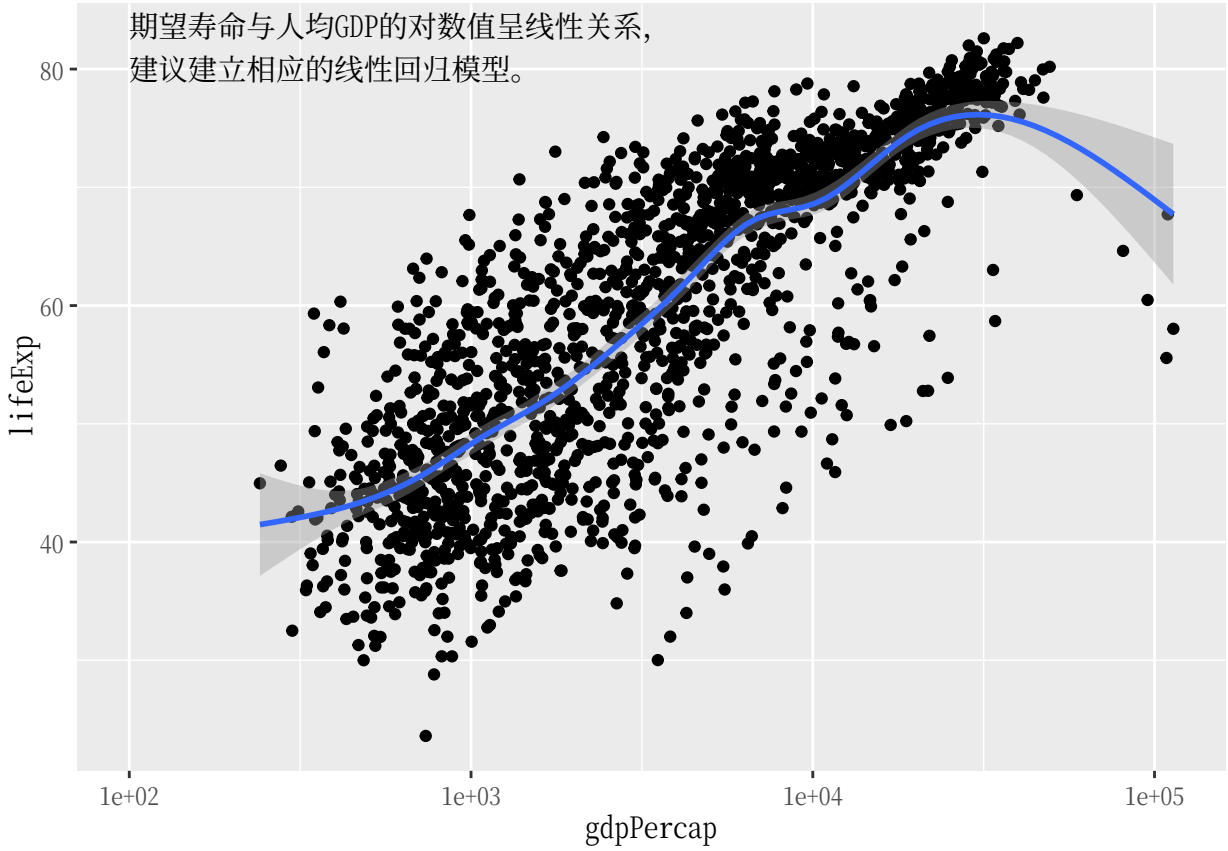
27.11.2 标注功能

通过 `annotate(geom = "text")` 调用 `geom_text()` 的功能，可以在一个散点图中标注多行文字，多行之间用`"\n"` 分开：

```
p <- ggplot(gapminder, aes(
    x = gdpPercap,
    y = lifeExp))
p + geom_point() +
```

```
  geom_smooth(method="gam") +
  scale_x_log10() +
  annotate(
    geom = "text",
    x = 1E2, y = 82, hjust = 0,
    label=paste0(
      " 期望寿命与人均 GDP 的对数值呈线性关系，\n",
      " 建议建立相应的线性回归模型。"))
```

```
## `geom_smooth()` using formula 'y ~ s(x, bs = "cs")'
```

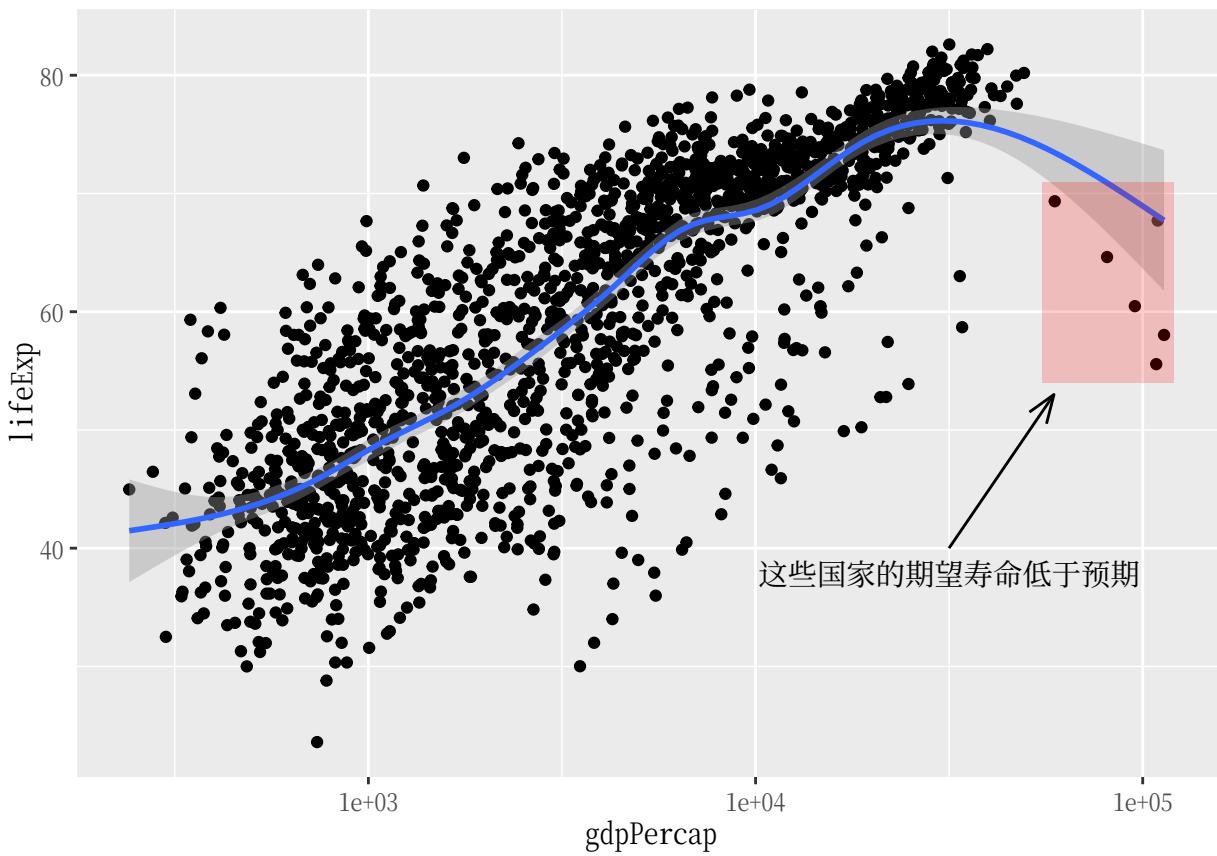


在 annotate() 中选 geom = "rect"，给出长方形的左右和上限界限，可以将上面图形中最右侧偏低的点用长方形填充标出。可以在 annotate() 中选 geom = "line" 画线，需要给出线的起点和终点坐标，可以 arrow 选项要求画箭头，用 arrow() 函数给出箭头的大小、角度等设置，如：

```
p + geom_point() +
  geom_smooth(method="gam") +
  scale_x_log10() +
  annotate(geom = "rect",
```

```
           xmin = 5.5E4, xmax = 1.2E5,
           ymin = 54, ymax = 71,
           fill = "red", alpha = 0.2) +
  annotate(geom = "line",
           x = c(5.9E4, 3.16E4),
           y = c(53,  40),
           arrow = arrow(angle = 20, length = unit(4, "mm"))) +
  annotate(geom = "text",
           x = 3.16E4, y = 38,
           label = " 这些国家的期望寿命低于预期")
```

```
## `geom_smooth()` using formula 'y ~ s(x, bs = "cs")'
```



这样标注的缺点是坐标都需要读图并试错摆放。

上述被标注的国家是：

```
gapminder |>
  filter(gdpPercap > 5.5E4 & gdpPercap < 1.2E5,
```

```
        lifeExp > 54 & lifeExp < 71) |>
  select(country, gdpPercap, lifeExp)
```

```
## # A tibble: 6 x 3
##   country gdpPercap lifeExp
##   <fct>       <dbl>   <dbl>
## 1 Kuwait    108382.    55.6
## 2 Kuwait    113523.    58.0
## 3 Kuwait     95458.    60.5
## 4 Kuwait     80895.    64.6
## 5 Kuwait    109348.    67.7
## 6 Kuwait     59265.    69.3
```

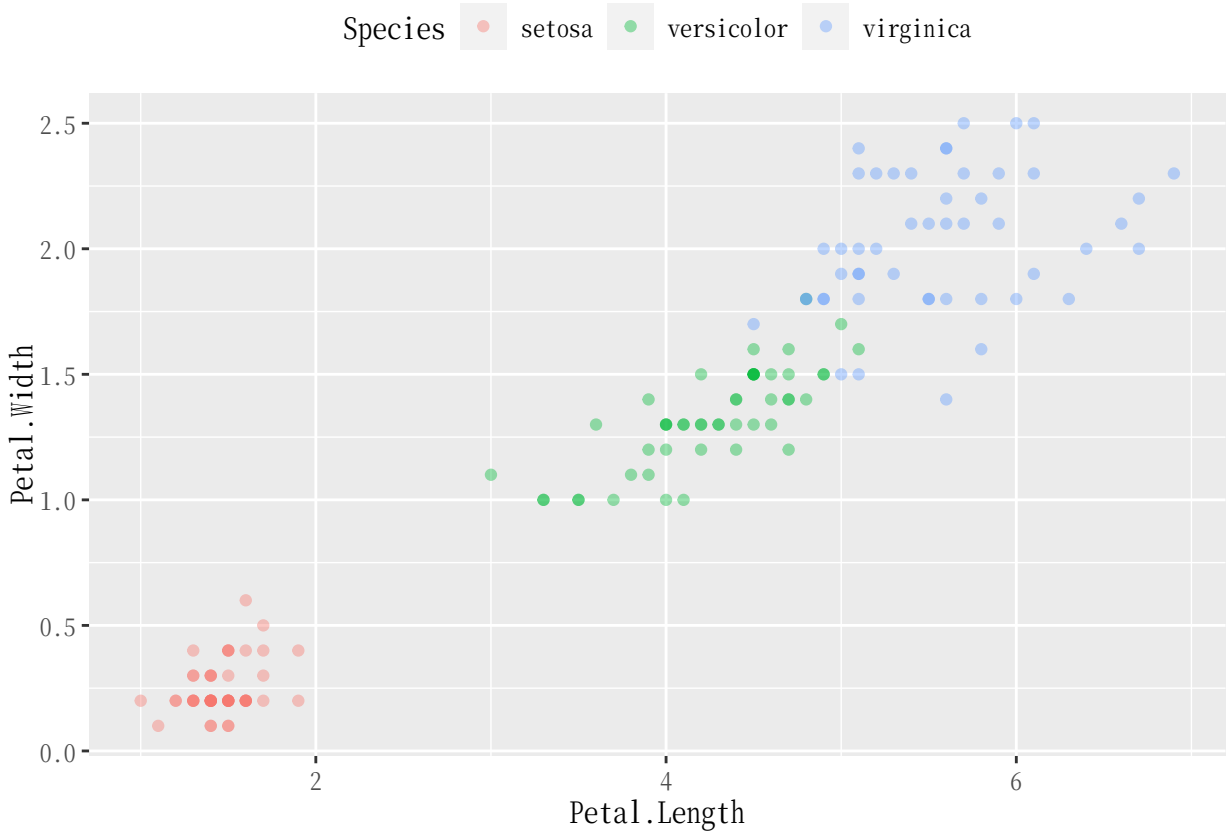
可以用 geom_hline()、geom_vline() 和 geom_abline() 画横线、竖线、斜线。ggplot2 的默认主题会自动画参考线，可以用 theme() 函数指定参考线画法。

27.11.3 指南 (guides)

对于颜色、填充色等维度，会自动生成图例。用 guides(color = "none") 这样的方法可以取消指定维度的图例。

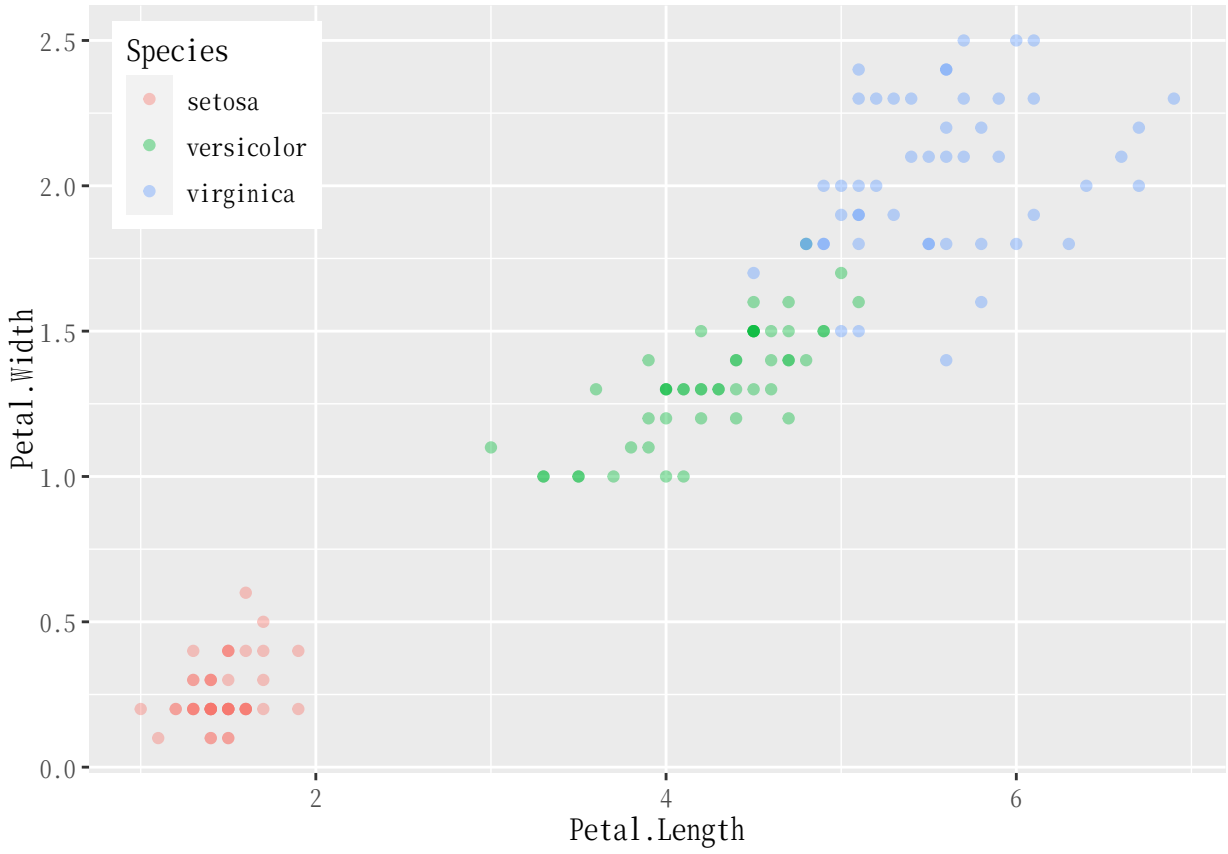
theme() 可以调整一些整体的设置，如背景色、字体、图例的摆放位置。用 theme() 的 legend.position 改变图例的位置，如 theme(legend.position = "top") 可以将图例放置在上方，默认是放置在右侧的。可取值有"none"、"left"、"right"、"bottom"、"top"，如：

```
p <- ggplot(iris, aes(
  x = Petal.Length, y = Petal.Width, color = Species))
p + geom_point(alpha = 0.4) +
  theme(legend.position = "top")
```



图例位置 `legend.position` 还可以指定在作图区域内部用两个百分比数字给定，同时可以用 `legend.just` 指定位置对准图例框的哪一个角，也是用两个百分比数字，比如放在左上角内部：

```
p + geom_point(alpha = 0.4) +
  theme(
    legend.position = c(0.02, 0.98),
    legend.justification = c(0,1)  )
```



某一维的图例做法可以在相应的 `scale_xxx()` 函数中用 `guide=` 指定，或者在 `guides()` 中用维名称 = 指南方法指定，如 `guides(fill = "guides")` 或者 `guides(fill = "colorbar")`。指定的指南还可以是 `guide_legend()` 或者 `guide_colorbar()` 的结果，在这两个函数中可以用多个选项指定指南或颜色条的具体做法。

有一个 directlabels 包，可以巧妙地将各类的图例直接就近标在坐标系内部。

27.12 拼接图形

`facet_wrap()` 和 `facet_grid()` 可以按照某一个或两个分类变量的值将输入数据集分为若干个子集，将每个子集分别在一个切片上绘图。有时还需要将几幅不同图形拼在一起，这些图形可以是同一数据的不同类型图形，也可以是完全无关的图形。

在拼接图形时，各个切片应该具有类似的风格，即背景、配色、字体等应该尽可能一致，上下和左右的坐标轴应尽可能对齐，切片的标签应该尽可能不显眼。

patchwork 和 cowplot 扩展包提供了方便的拼接功能。

27.12.1 利用 patchwork 包拼接图形

patchwork 包提供了拼接图形的功能。如果仅仅将几个图按照等分网格拼接，可以用 | 表示左右拼接，用/表示上下拼接。如：

```
library(patchwork)
```

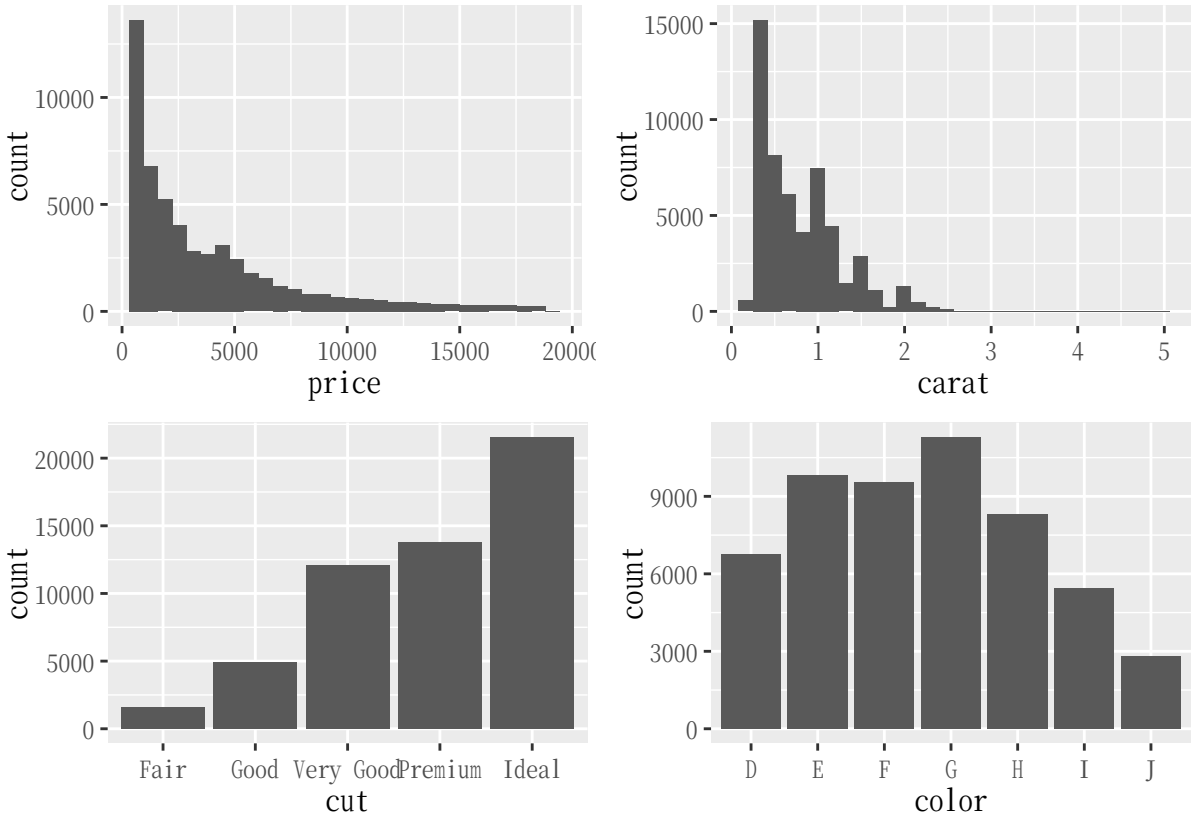
```
##
## Attaching package: 'patchwork'
```

```
## The following object is masked from 'package:cowplot':
##
##     align_plots
```

```
p1 <- ggplot(data=diamonds, mapping=aes(
  x = price)) +
  geom_histogram()
p2 <- ggplot(data=diamonds, mapping=aes(
  x = carat)) +
  geom_histogram()
p3 <- ggplot(data=diamonds, mapping=aes(
  x = cut)) +
  geom_bar()
p4 <- ggplot(data=diamonds, mapping=aes(
  x = color)) +
  geom_bar()
print((p1 | p2) / (p3 | p4))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



如果仅左右并列，可以用 `print(p1 | p2)`。如果仅上下叠放，可以用 `print(p1 / p2)`。也可以用 `+` 连接两个图，这时自动按行填入。

27.12.2 利用 cowplot 包拼接图形

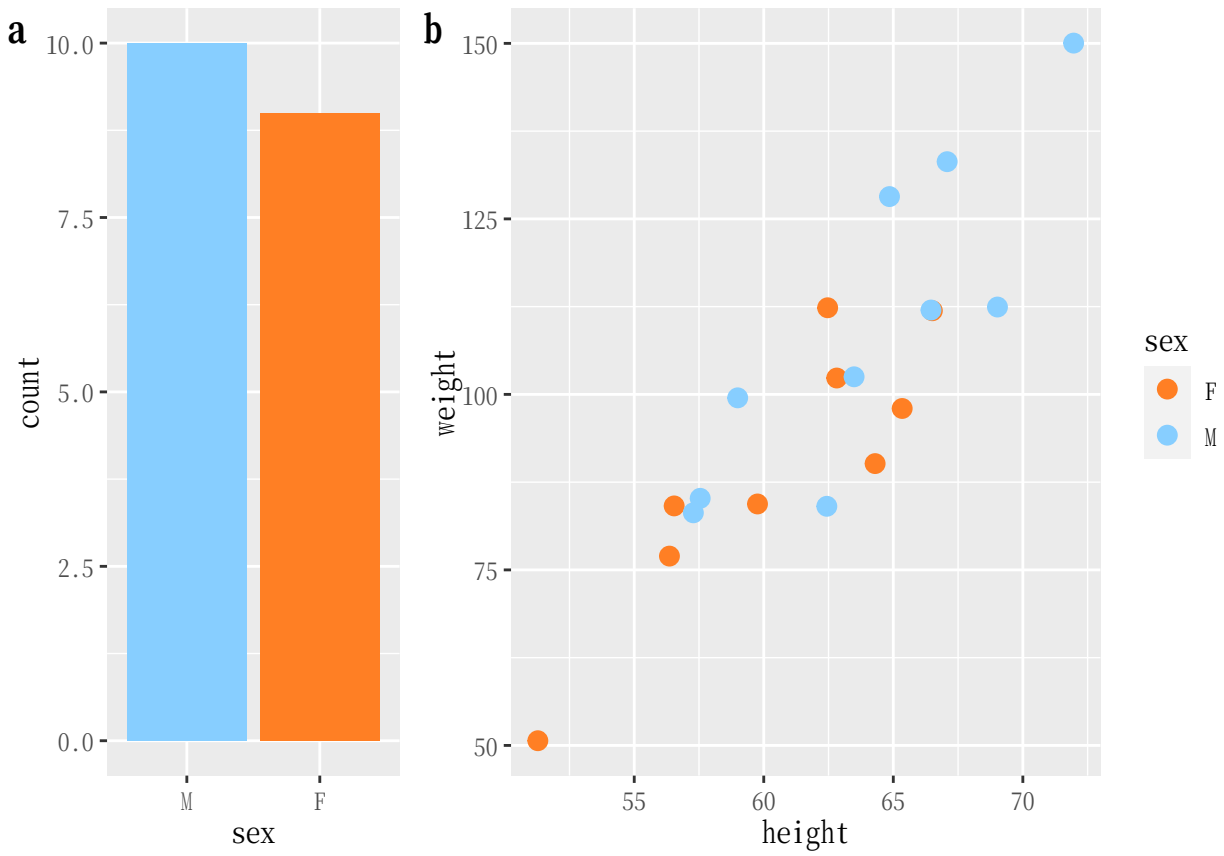
cowplot 包也提供了拼接图形的办法，使用时先将每个图分别赋值给一个 R 变量，然后用 `plot_grid()` 函数摆放在一幅图中。

例如，19 个学生的性别、年龄、身高、体重数据：

```
library(cowplot)
dclass <- read_csv(
  "data/class.csv",
  col_types=cols(
  .default = col_double(),
  name=col_character(),
  sex=col_factor(levels=c("M", "F"))
))
p1 <- ggplot(data = dclass, mapping = aes(x = sex, fill = sex)) +
  geom_bar() +
```

```
  scale_fill_manual(
    guide = "none",
    values = c("F" = "chocolate1",
               "M" = "skyblue1")  )
p2 <- ggplot(data = dclass, mapping = aes(
  x = height, y = weight, color = sex)) +
  geom_jitter(size = 3) +
  scale_color_manual(
    values = c("F" = "chocolate1",
               "M" = "skyblue1")  )

plot_grid(
  p1, p2, labels = "auto",
  rel_widths = c(0.3, 0.6),
  align = "h")
```



plot_grid() 中用 rel_widths 指定了左右图的相对比例，默认是均分的。当图形上下排列时，可以用 rel_height 指定上下排的比例。程序中用了 align = "h" 使得左右图上下对齐，align 默认取"none"，也可以取"h"、"v" 或"hv"。可以用 ncol 指定图的列数，用 nrow 指定图的行数。

还可以通过嵌套调用 `plot_grid()` 方式实现非网格的排列。

27.13 主题

ggplot2 包作图可以实现内容与设计的分离，这里内容就是指数据、映射、统计、图形类型等方面，而设计就是指背景色、颜色表、字体、坐标轴做法、图例位置等的安排。将作图任务分解为内容与设计两个方面，可以让数据科学家不必关心设计有关的元素，而设计可以让专门的艺术设计人才来处理。这种工作分配已经在图书出版、网站、游戏开发等行业发挥了重要作用。

`theme()` 函数用来指定设计元素，称为主题 (theme)，而且可以单独开发 R 扩展包来提供适当的主题。ggthemes 扩展包是一个这样的包。

`theme(legend.position)` 可以用来选择图例位置。`theme_set()` 可以改变后续 ggplot2 作图的主题（配色、字体等）。如 `theme_set(theme_bw())`, `theme_set(theme_dark())` 等。对单次绘图，可以直接用加号连接 `theme_gray()` 等这些主题函数。主题包括 `theme_gray()`（默认主题）、`theme_minimal()`、`theme_classic()` 等。用法如

```
p + theme_classic()
```

ggthemes 扩展包提供了更多的主题选择。用法如：

```
library(ggthemes)
p + theme_economist()
```

hrbrthemes 包提供了一些额外的主题和刻度。ggtech 包提供了技术向的一些主题和刻度。ggdark 包提供了黑色主题。

ggsci 包提供了多种主要期刊的配色方案。

`theme()` 函数还可以直接指定颜色、字体、大小等设置。

27.14 图形定制调整

ggplot2 的默认设置一般能够满足我们的要求，只有在有特殊的图形类型需求或者制作出版用的图形时，才需要对图形进行定制调整。图形调整可以包括：

- 配色、位置摆放等审美方面的调整；
- 面向目标出版物或者目标读者的调整；
- 增加有意义的标注；
- 调整整体的观感。

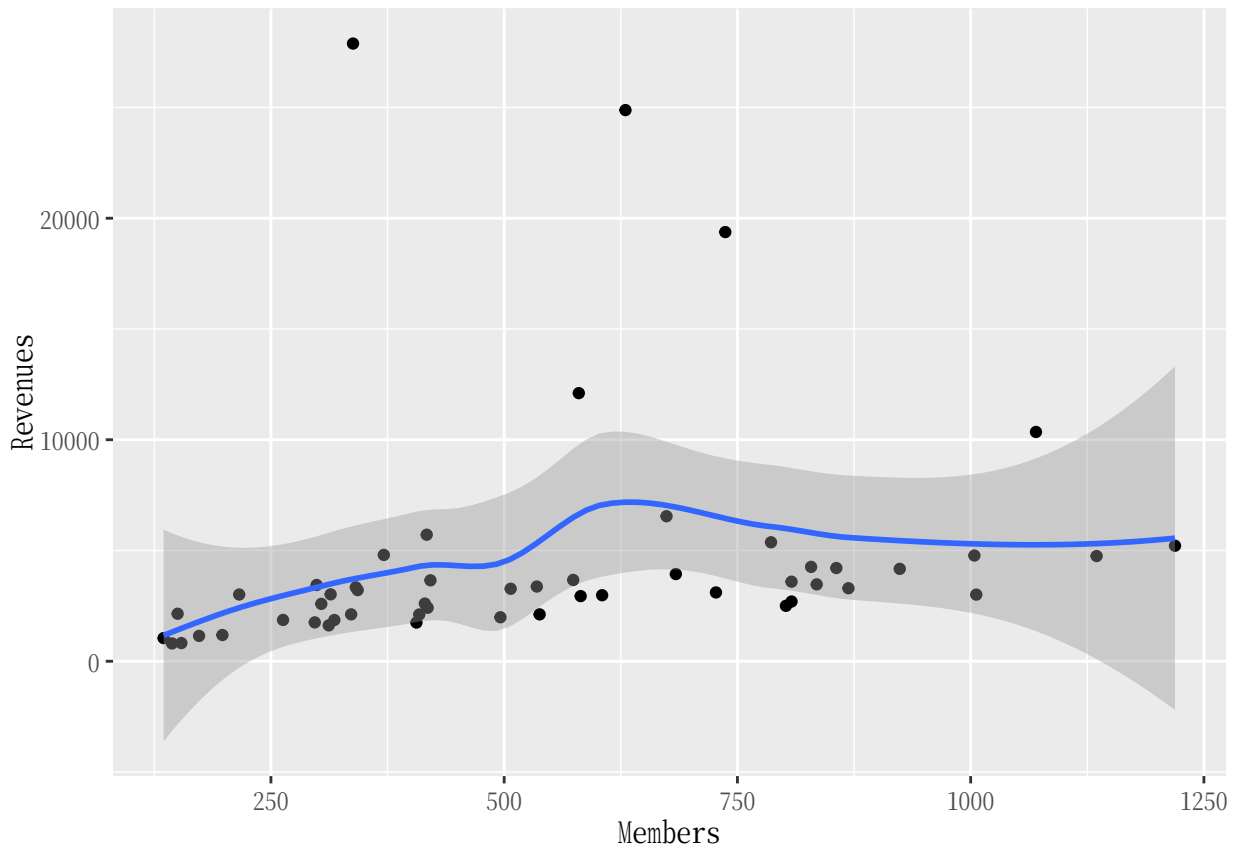
27.14.1 图形逐步调整例子

socviz 扩展包的 asasec 数据集是美国社会学学会 (ASA) 的各分会 2005 年到 2015 年的一些数据，其中的财务数据 (Beginning, Revenues, Expenses, Ending) 虽然各年都有值，但实际是用 2015 年的值填进去的。

作 2014 年收入对会员数的散点图与拟合曲线，每个散点是一个分会：

```
p <- ggplot(
  dplyr::filter(asasec, Year == 2014),
  aes(
    x = Members,
    y = Revenues,
    label = Sname))
p + geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

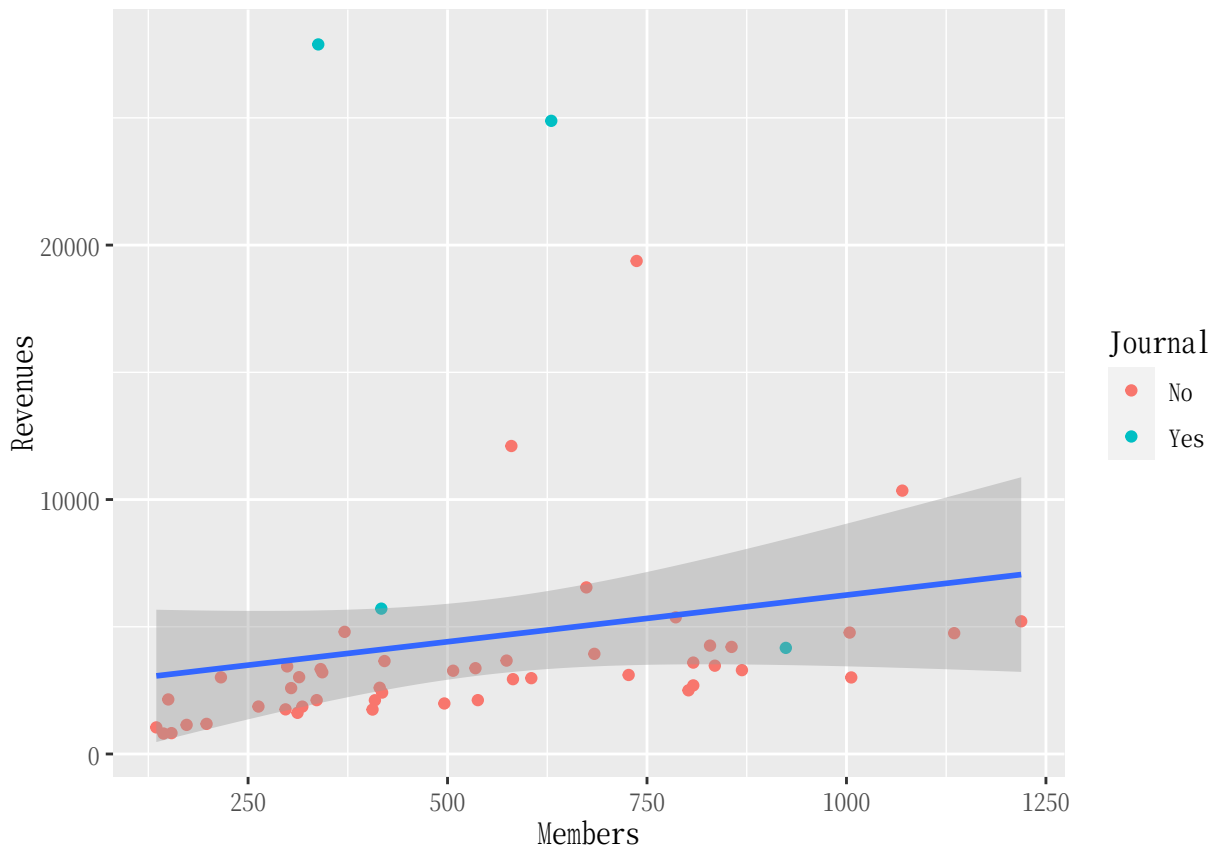


程序中的映射 `label = Sname` 暂时不起作用，在 `geom_text()` 中才需要这一维度。

下面按照有无期刊染色，将平滑方法改为线性回归：

```
p + geom_point(mapping = aes(color = Journal)) +
  geom_smooth(method = "lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```

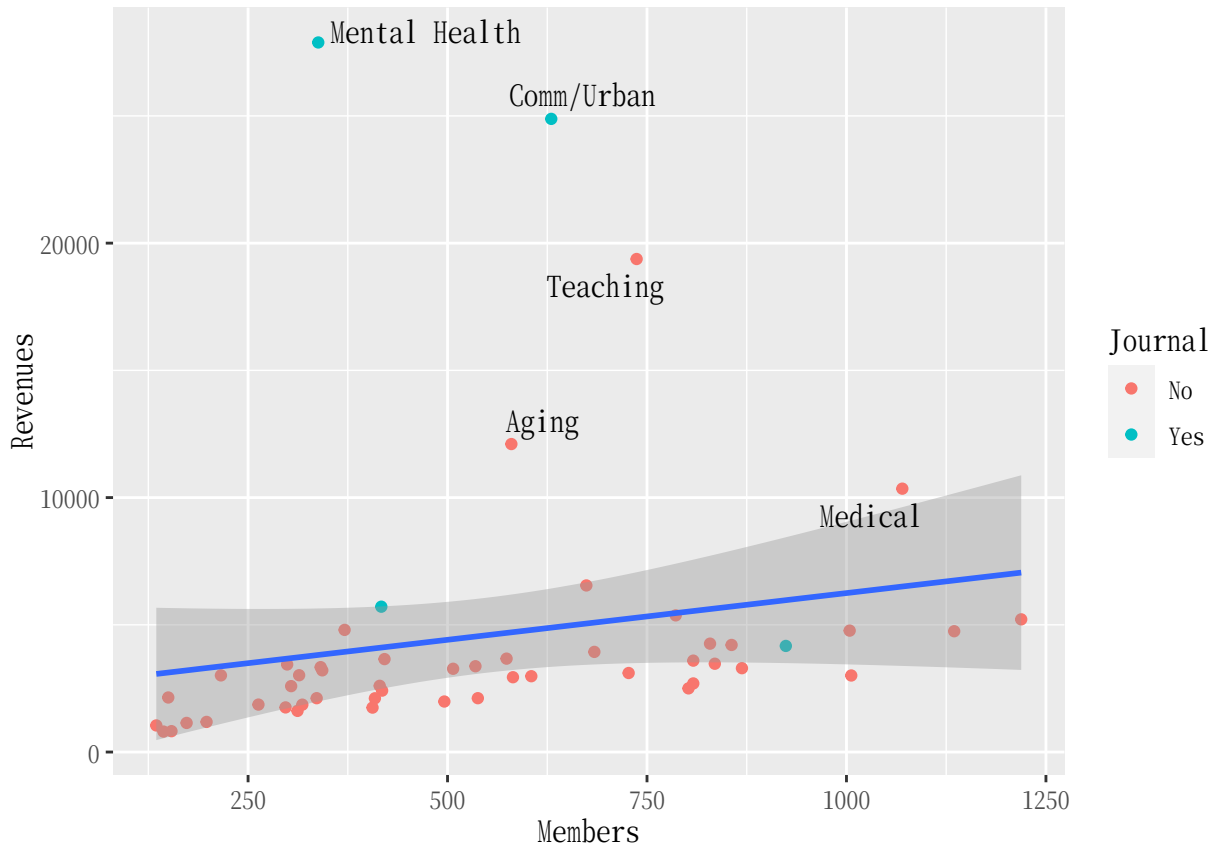


注意 `color = Journal` 的映射仅对 `geom_point()` 有效，如果写在 `ggplot()` 中，就对 `geom_smooth()` 也有效了。

下面标出异常值：

```
p + geom_point(mapping = aes(color = Journal)) +
  geom_smooth(method = "lm") +
  geom_text_repel(
    data = dplyr::filter(
      asasec, Year == 2014, Revenues > 7000))
```

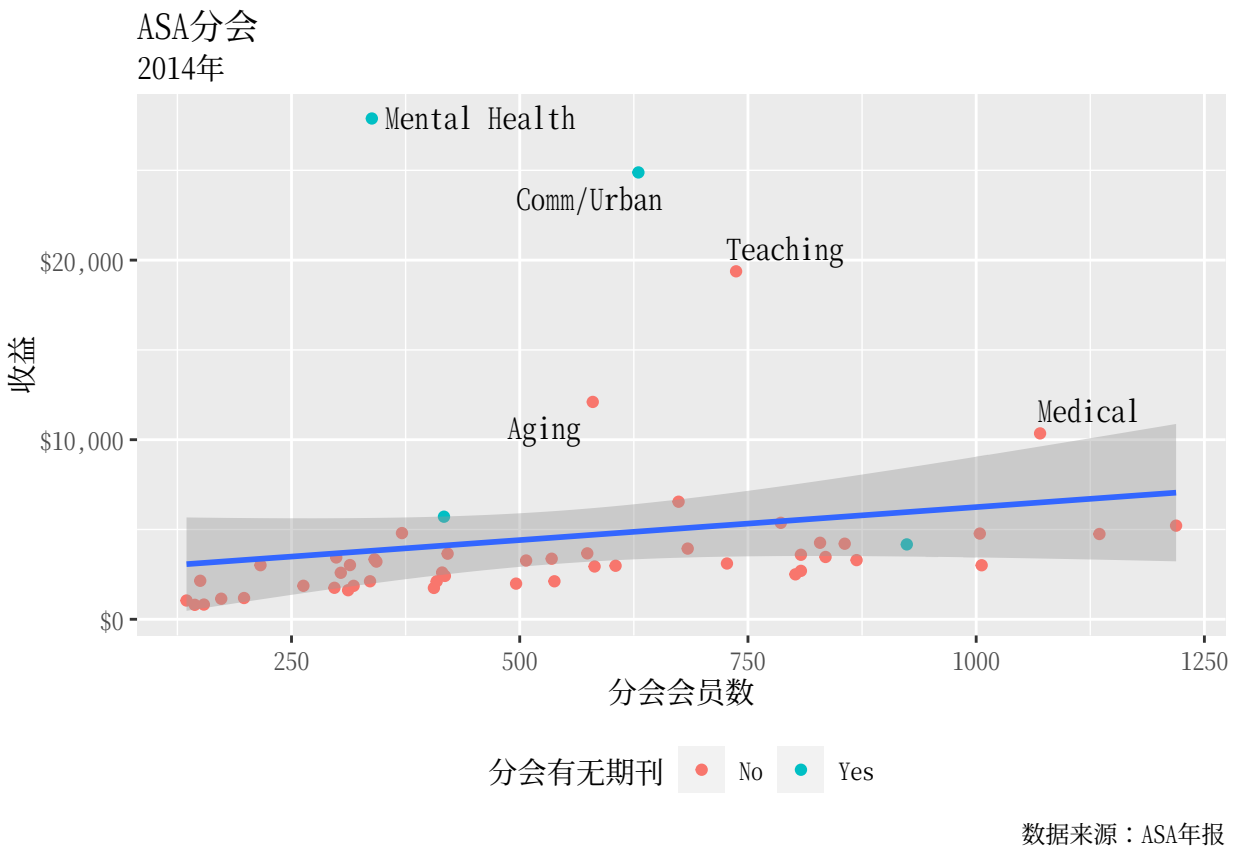
```
## `geom_smooth()` using formula 'y ~ x'
```



下面用 `labs()` 调整标题、用 `scale_y_continuous()` 调整 y 轴刻度标法、用 `theme()` 调整图例位置：

```
p +
  geom_point(
    mapping = aes(color = Journal)) +
  geom_smooth(method = "lm") +
  geom_text_repel(
    data = dplyr::filter(
      asasec,
      Year == 2014, Revenues > 7000)) +
  labs(
    title = "ASA 分会",
    subtitle = "2014 年",
    x = " 分会会员数",
    y = " 收益",
    color = " 分会有无期刊",
    caption = " 数据来源: ASA 年报"  ) +
  scale_y_continuous(labels = scales::dollar) +
  theme(legend.position = "bottom")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



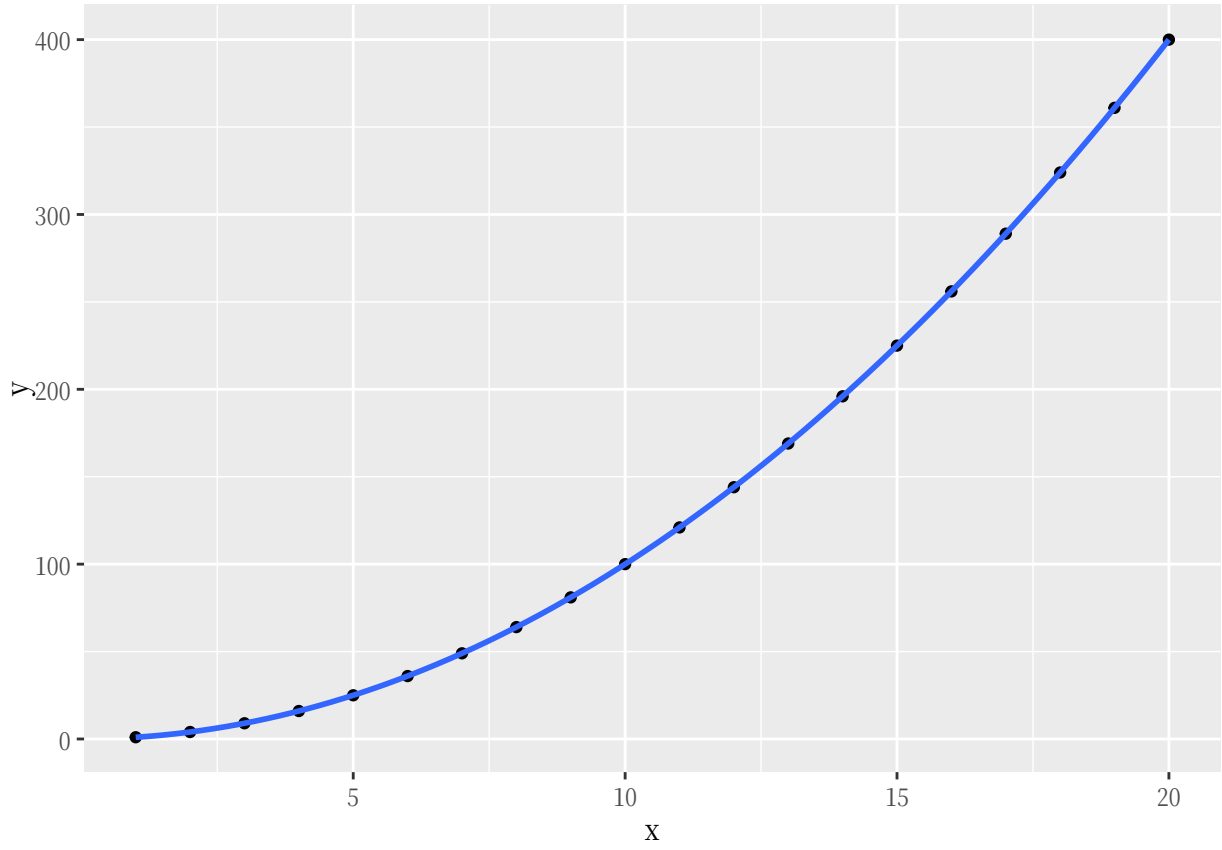
27.15 在自定义函数中调用图形函数

因为 ggplot2 包的函数在指定变量时，用了原样变量名而不是变量名字符串的写法，所以如果在自定义函数中调用这些函数，要作图用的变量名用函数形参传入，就会被理解成形参名本身是要作图的变量。为此，使用 `{{ var_name }}` 这样的语法，`var_name` 是函数形参用来传入要作图的变量。

例如，下面的函数输入数据框和 x 变量名、y 变量名，作散点图并叠加拟合曲线：

```
da <- tibble(x = 1:20, y = x^2)
scat_line <- function(df, x_var, y_var){
  ggplot(df, aes(
    x = {{ x_var }},
    y = {{ y_var }})) +
    geom_point() +
    geom_smooth()
}
scat_line(da, x, y)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



但是，上面的调用不能写成 `scat_line(da, "x", "y")`。

关于在自定义函数中使用 ggplot2，可参考Programming with ggplot2。

27.16 交互作图、动态图形

esquisse 包为 RStudio 提供了用鼠标、快捷图标和菜单进行交互作图的功能，通过安装一个 addin 实现，作图所使用的 ggplot 代码可以显示出来作为样例。

ggedit 也提供了 RStudio addin，可以交互地生成图形，帮助初学者快速掌握 ggplot2 编程。

gganimate 包支持产生动画图形。对某个已有的统计图形，可以使得某个相关的变量值逐步改变，从而使得统计图产生渐变，动态显示此渐变过程。利用了 GIF 动画格式。

27.17 参考文献

- Cleveland, W. S. (1993). The elements of graphing data. Hobart Press.
- Cleveland, W. S. (1994). Visualizing data. Hobart Press.

- Cleveland, W. S., & McGill, R. (1984). Graphical perception: Theory, experimentation, and application to the development of graphical methods. Journal of the American Statistical Association, 79, 531–534.
- Cleveland, W. S., & McGill, R. (1987). Graphical perception: The visual decoding of quantitative information on graphical displays of data. Journal of the Royal Statistical Society Series A, 150, 192–229.
- Edward R. Tufte((1983) The Visual Display of Quantitative Information
- Tufte, E. R. (1983). The visual display of quantitative information. Cheshire, CT: Graphics Press.
- Tufte, E. R. (1990). Envisioning information. Cheshire, CT: Graphics Press.
- Tufte, E. R. (1997). Visual explanations: Images and quantities, evidence and narrative. Cheshire, CT: Graphics Press.
- Wickham, H. (2016). Ggplot2: Elegant graphics for data analysis. New York: Springer.
- Wickham, H., & Chang, W. (2018). Ggplot2: Create elegant data visualisations using the grammar of graphics.
- Wilkinson, L. (2005). The grammar of graphics (Second). New York: Springer.

Chapter 28

ggplot 的各种图形

28.1 介绍

ggplot2 包提供了许多种图形，其作用可以大致地分为：

- 表现数量；
- 表现一维或者二维分布；
- 表现两个变量之间的数量关系，等等。

下面按照其作用分别进行介绍。

主要参考：

- Claus O. Wilke(2019). Fundamentals of Data Visualization. O'Reilly Media. https://serialmentor.com/dataviz/

28.2 表现数量

28.2.1 条形图

设有若干个类，每个类有一个数量属性值。经常用条形图表现数量。

28.2.1.1 简单的条形图

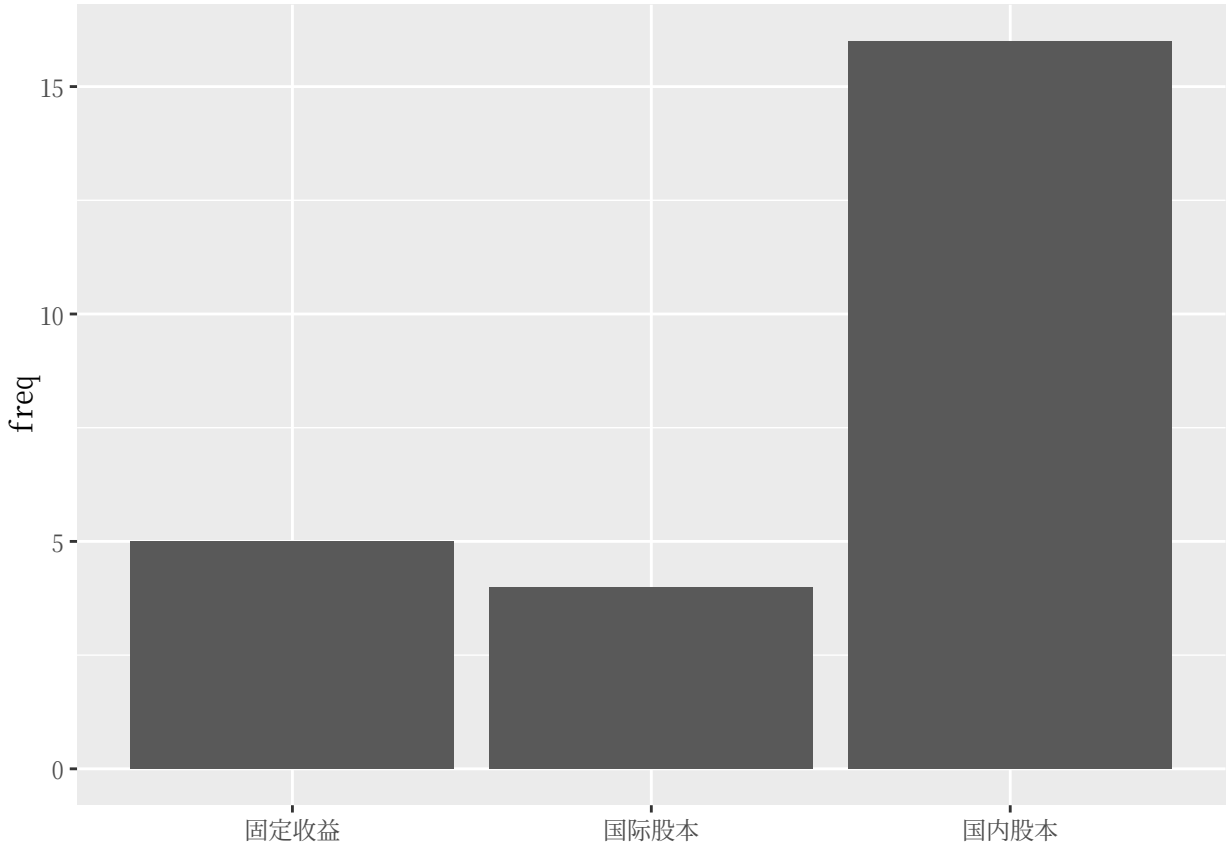
例如，有 25 个共同基金，分为三个类别，各类别的频数为：

```
d_funds_type <- tibble(
  type = c(" 国内股本", " 国际股本", " 固定收益"),
  freq = c(16, 4, 5)
)
knitr::kable(d_funds_type)
```

type	freq
国内股本	16
国际股本	4
固定收益	5

用 `ggplot()` 输入数据并指定映射，用 `geom_col()` 作条形图，x 轴为类别，y 轴为每个类别对应的数值，这里是输入数据中的频数：

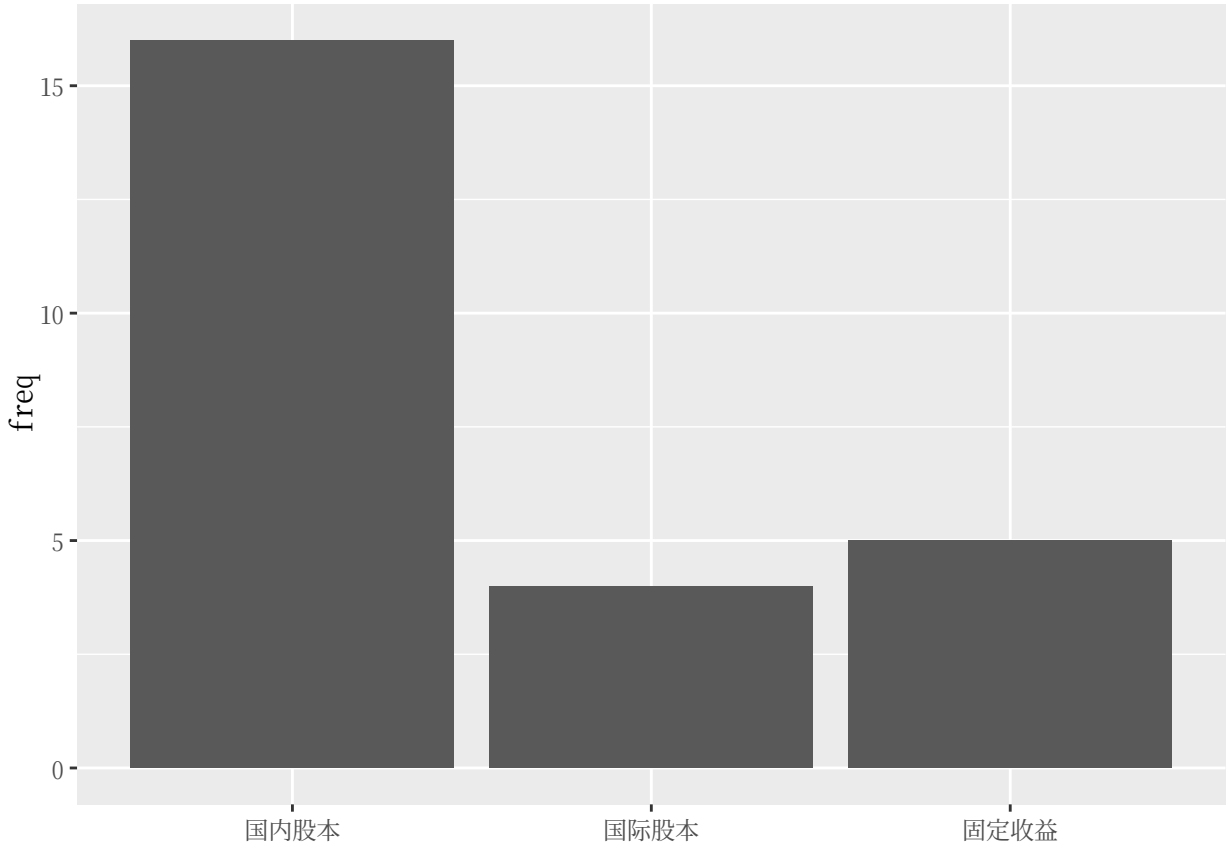
```
ggplot(d_funds_type, aes(
  x = type, y = freq )) +
  geom_col() +
  labs(x = NULL)
```



上面的三个类的次序并不是输入的次序，这是因为字符型变量会自动转换为因子型，因子水平一般按字典序排列。可以在输入时用 `factor(x, levels=<` 指定的因子水平表 `>)` 定义因子并人为指定因子次序，或者用 forcats 包

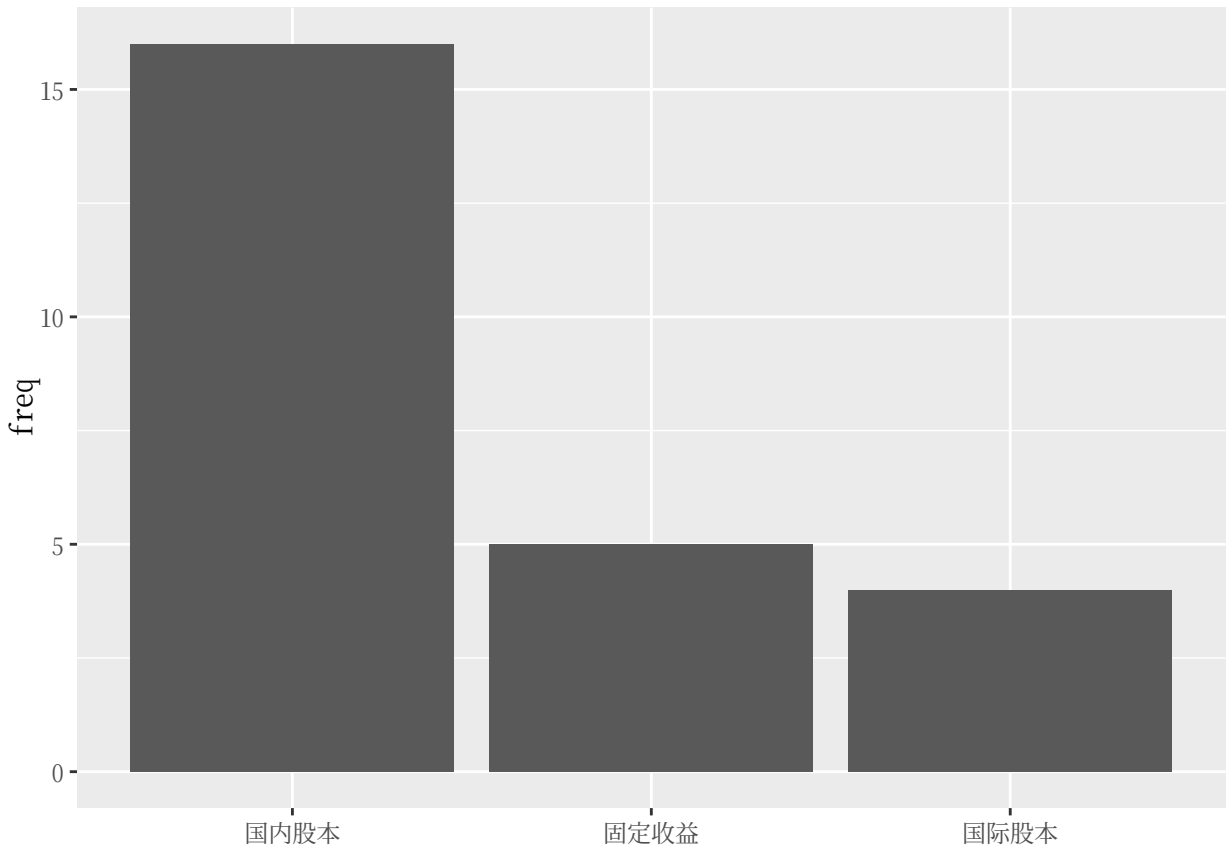
的 fct_relevel() 函数重新指定因子次序。比如，下例用 fct_relevel() 函数指定了因子次序：

```
d_funds_type |>
  mutate(type = fct_relevel(
    type, " 国内股本", " 国际股本", " 固定收益")) |>
  ggplot(aes(
  x = type, y = freq )) +
  geom_col() +
  labs(x = NULL)
```



可以用 forcats 包的 fct_reorder() 函数或者用 stats 包的 reorder() 函数，将因子水平按照某种有意义的次序排列。下例用 fct_reorder() 函数将各个水平按照频数大小排序：

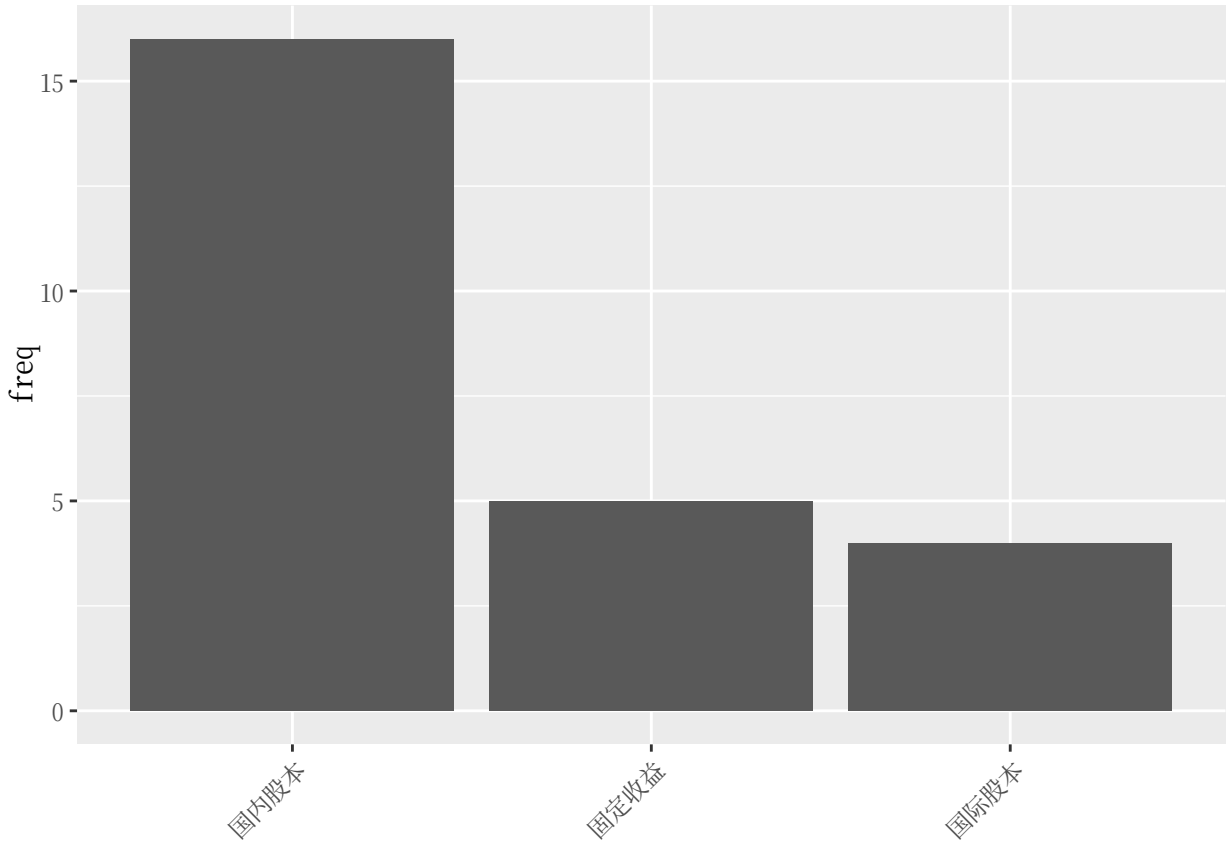
```
p <- ggplot(d_funds_type, mapping = aes(
  x = fct_reorder(type, desc(freq)),
  y = freq ))
p + geom_col() +
  labs(x = NULL)
```



注意，条形图中这种对各条形按照高度重排序的做法仅适用于各类没有自然的次序的情形，如果各类有自然的次序，比如年份、月份、季度、年龄组、收入高低分组等，就应该按照各类的自然次序。

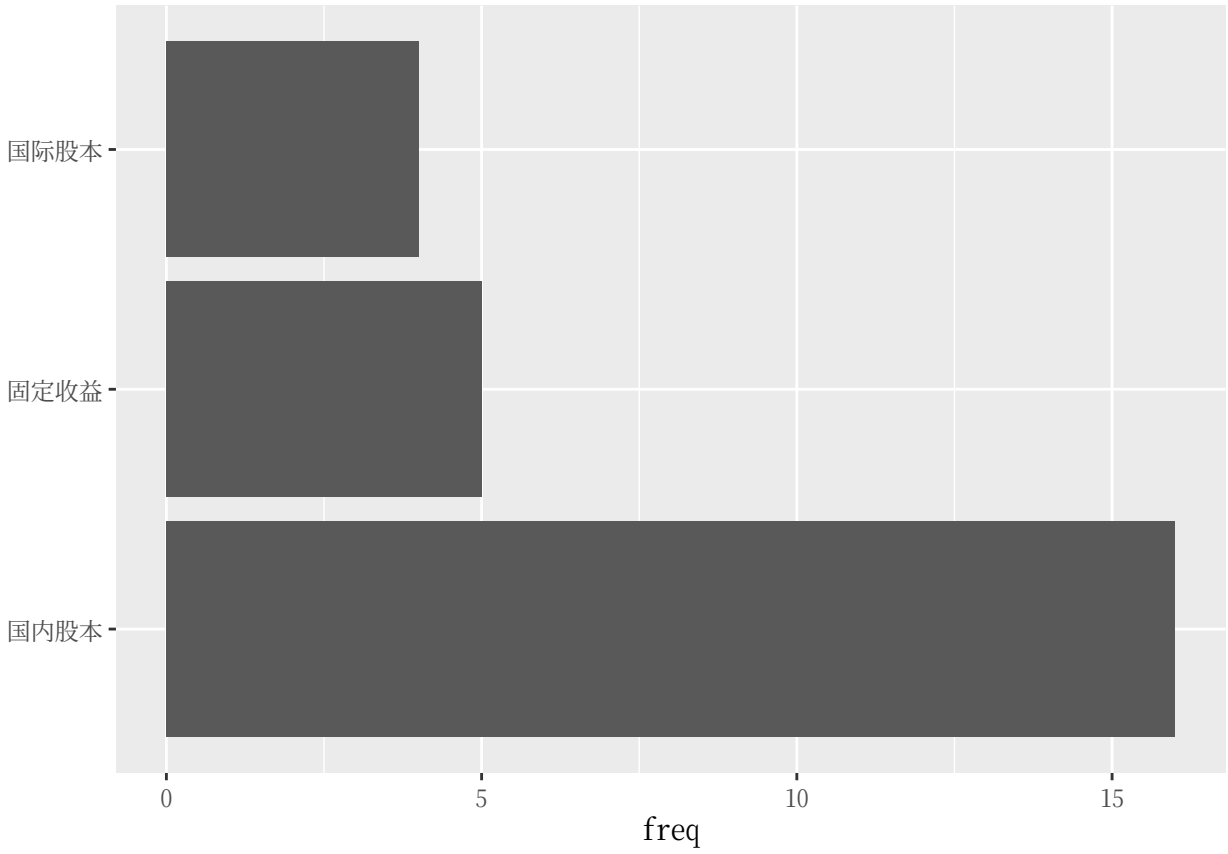
当分类的标签值（如“固定收益”）较长而且类个数较多时，横轴的宽度可能不够用，使得相邻两项的标签重叠，只好省略部分标签。这时，一种办法是将类标签方向变成 45 度角：

```
p + geom_col() +
  theme(
    axis.text.x = element_text(
      angle = 45, vjust = 1, hjust = 1) ) +
  labs(x = NULL)
```



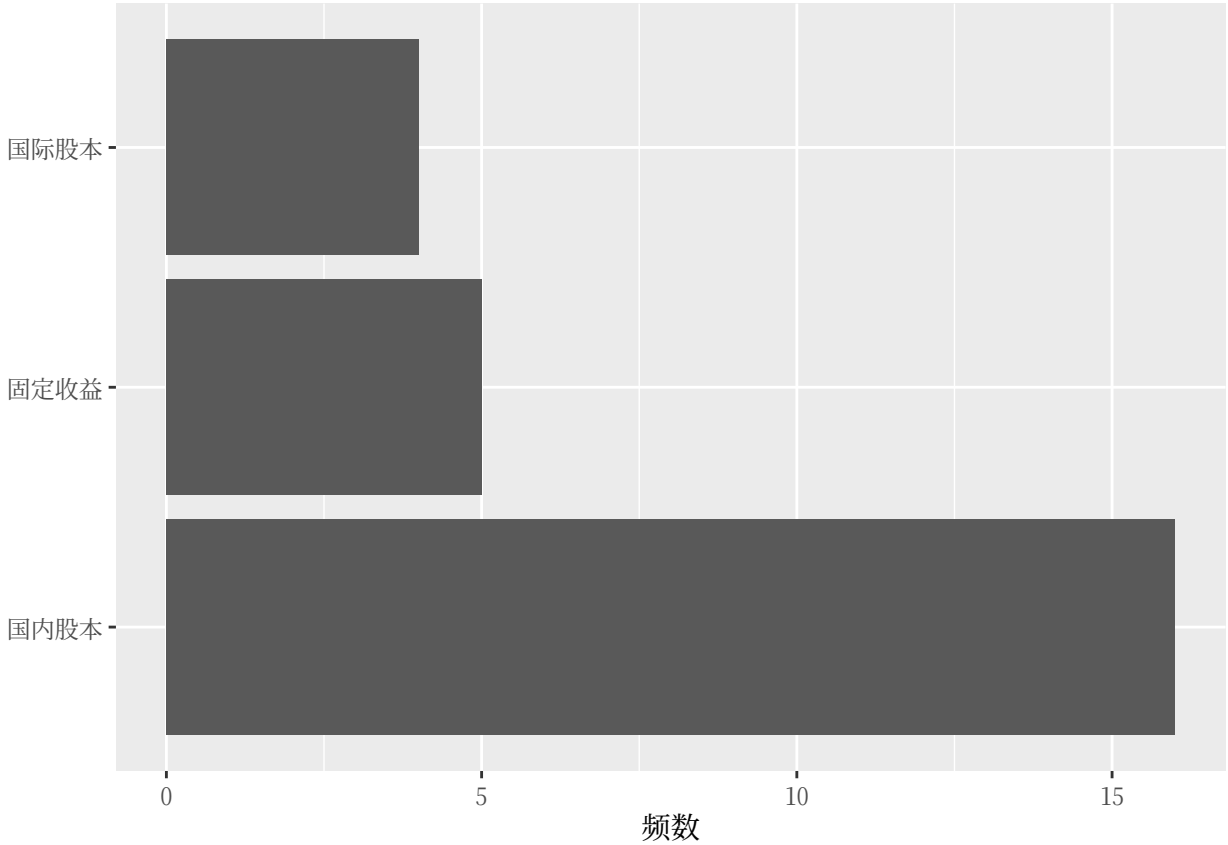
针对条形图的各条形的标签太长的问题，更好的办法是将条形横向放置，这样类标签就到了 y 轴，可以横向显示。这可以在映射中直接修改，但最好还是用 `coord_flip()` 函数颠倒 x 轴与 y 轴的作用，因为有些图形是不允许修改映射的。程序如

```
p + geom_col() +
  coord_flip() +
  labs(x = NULL)
```



下面用 `labs()` 去掉分类标签轴的标题并修改频数轴的标题：

```
p + geom_col() +
  labs(x = NULL,
       y = " 频数") +
  coord_flip()
```



如果需要将频数替换成比例或者百分数，只要修改输入数据增加相应的变量即可。

28.2.1.2 并列和堆叠的条形图

在某次对某物业公司服务的 4 个社区的居民的抽样调查中抽查了 80 位居民，其性别与所属社区如下：

```
mat_comm <- matrix(
  c(9, 10, 13, 4,
  18, 7, 8, 11),
  nrow=4, ncol=2,
  dimnames = list(c(
    "A 社区", "B 社区", "C 社区", "D 社区"),
    c(" 男", " 女")))
knitr::kable(mat_comm)
```

	男	女
A 社区	9	18
B 社区	10	7
C 社区	13	8
D 社区	4	11

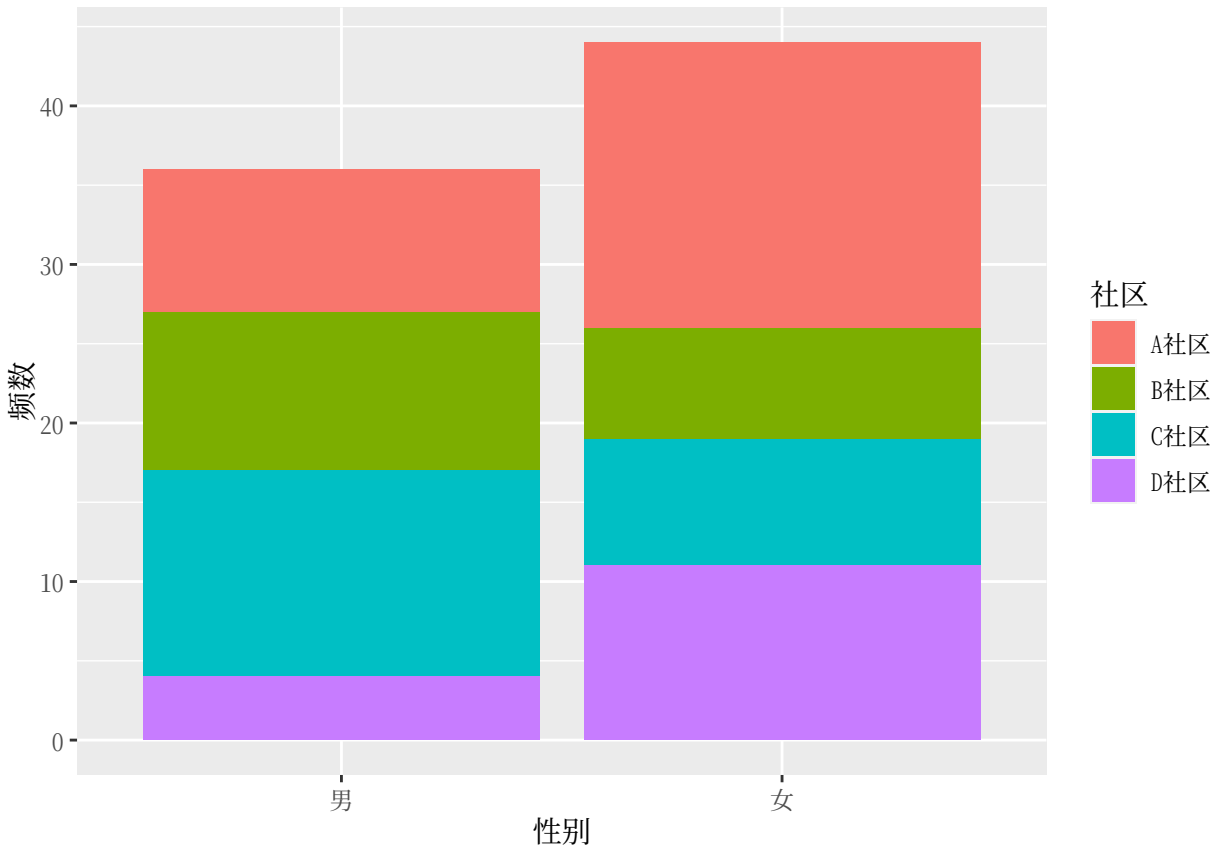
将表格数据转换为数据框：

```
d_comm <- mat_comm |>
  as.table() |>
  as.data.frame() |>
  setNames(c(" 社区", " 性别", " 频数"))
knitr::kable(d_comm)
```

社区	性别	频数
A 社区	男	9
B 社区	男	10
C 社区	男	13
D 社区	男	4
A 社区	女	18
B 社区	女	7
C 社区	女	8
D 社区	女	11

现在数据中有两个分类变量，频数是交叉分类的结果，这样的数据称为“列联表”。这时，将两个分类变量中一个作为大类，映射到 x 维度，另一个作为小类，映射到 fill 维度，频数映射到 y 维度。使用 `geom_col()` 或者 `geom_bar(stat = "identity")`。如：

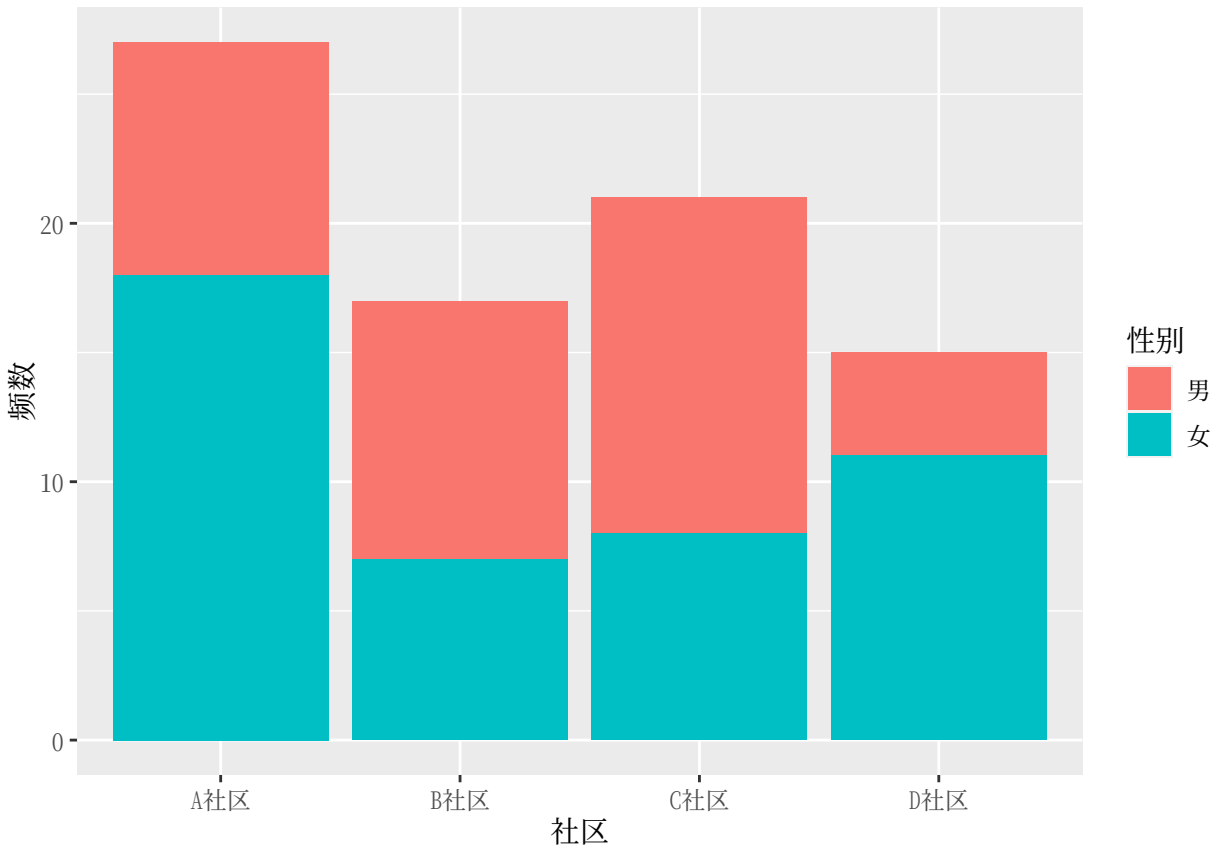
```
ggplot(d_comm, aes(
  x = `性别`, fill = `社区`, y = `频数`)) +
  geom_col()
```



结果为堆叠的条形图，每个大类画一个分段的条形，条形的程度等于各段长度之和。这需要各段长度之和相当于每个大类的某种总计，对于频数，这当然是合理的，每段长度每个交叉小类的频数，而一个条形的总长度是一个大类的频数。当 y 轴映射到其它指标时堆叠条形图不一定合适，比如，如果 y 映射到每个交叉小类的某个指标的标准差，因为标准差的和并不是标准差，所以就不能用堆叠条形图表示。

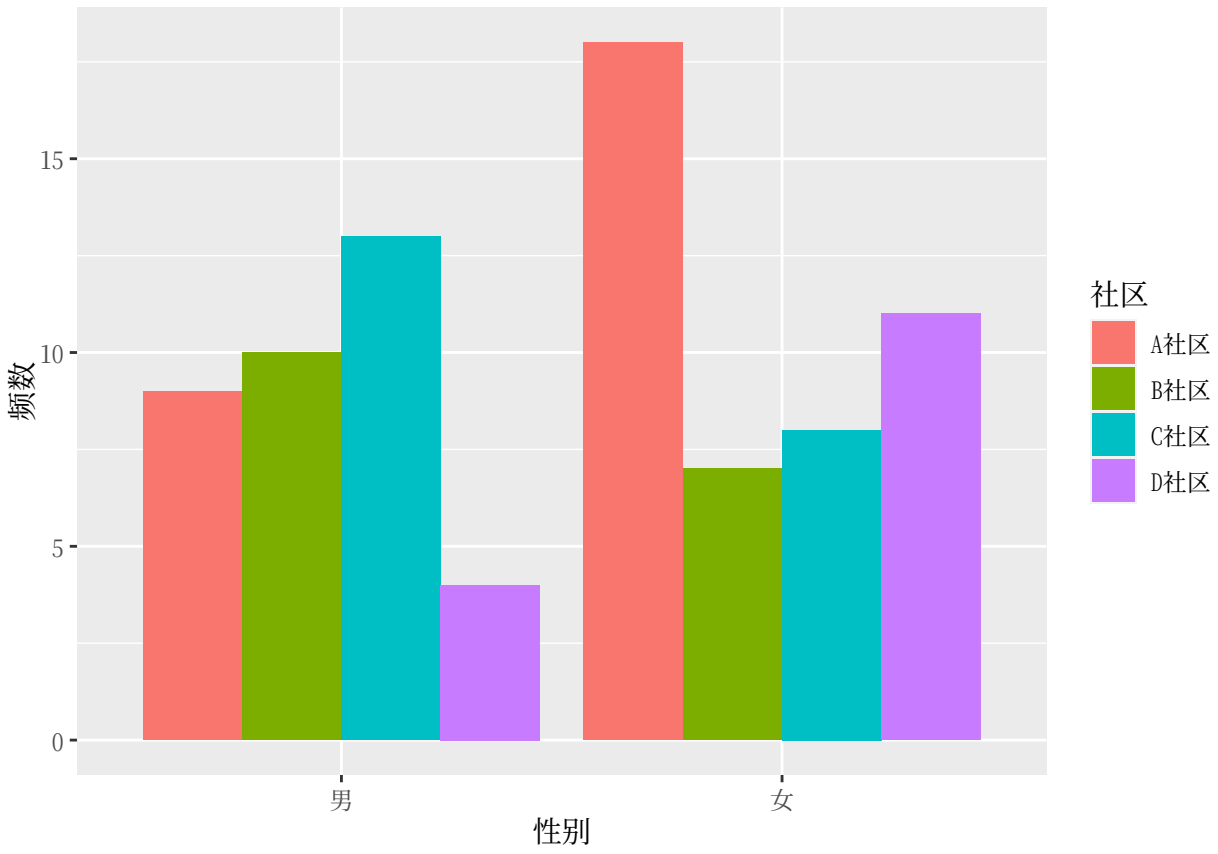
下面将社区作为大类，性别作为小类：

```
ggplot(d_comm, aes(
  x = `社区`, y = `频数`, fill = `性别`)) +
  geom_col()
```



在 geom_col() 和 geom_bar() 中加选项 position = "dodge" 可以将小类并列放置：

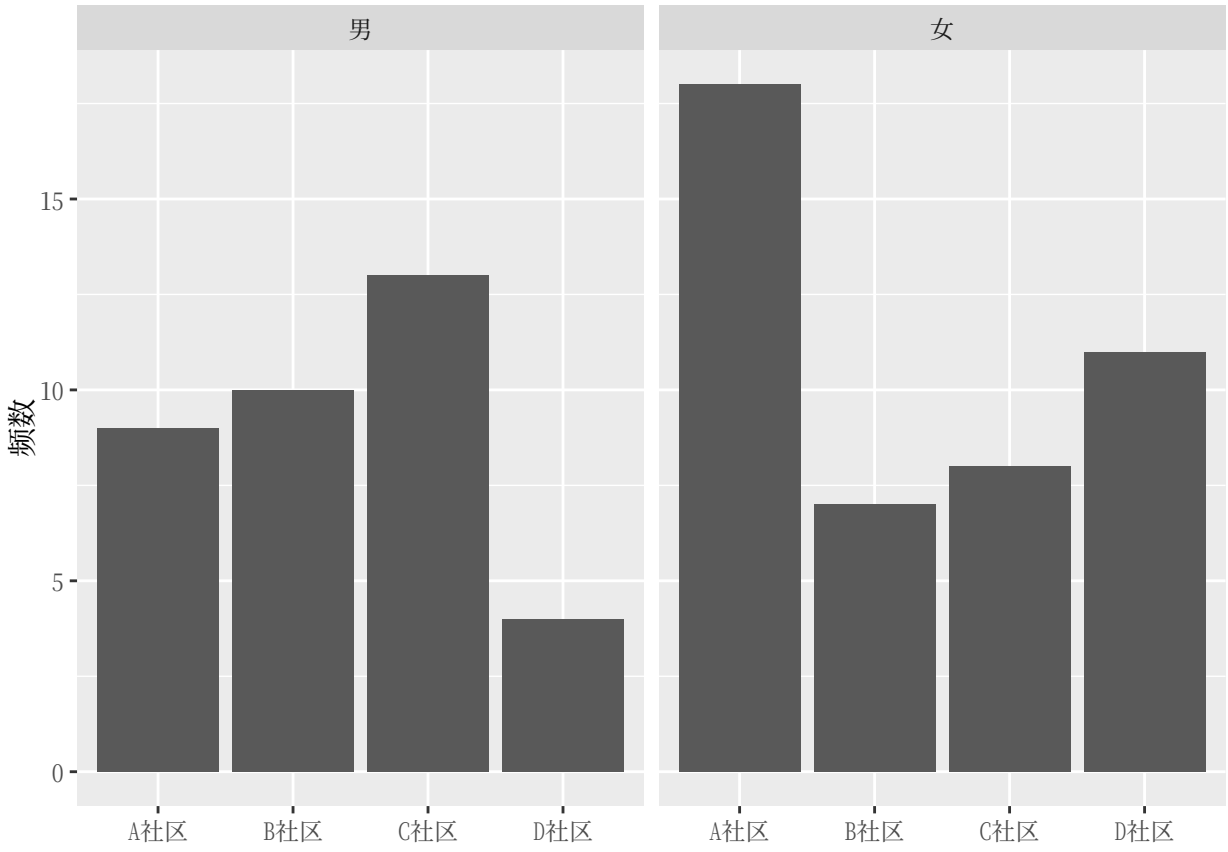
```
ggplot(d_comm, aes(
  x = `性别`, fill = `社区`, y = `频数`)) +
  geom_col(position = "dodge")
```



这样得到并列的条形图。并列条形图就不需要像堆叠条形图那样要求各个小类的 y 轴变量和有意义。堆叠条形图相当于在 `geom_col()` 中加 `position = "stack"`。

也可以将不同的大类放置到不同的切片，用 `facet_wrap()` 或者 `facet_grid()` 函数：

```
ggplot(d_comm, aes(
  x = `社区`, y = `频数`)) +
  geom_col() +
  facet_wrap(~ `性别`) +
  labs(x = NULL)
```

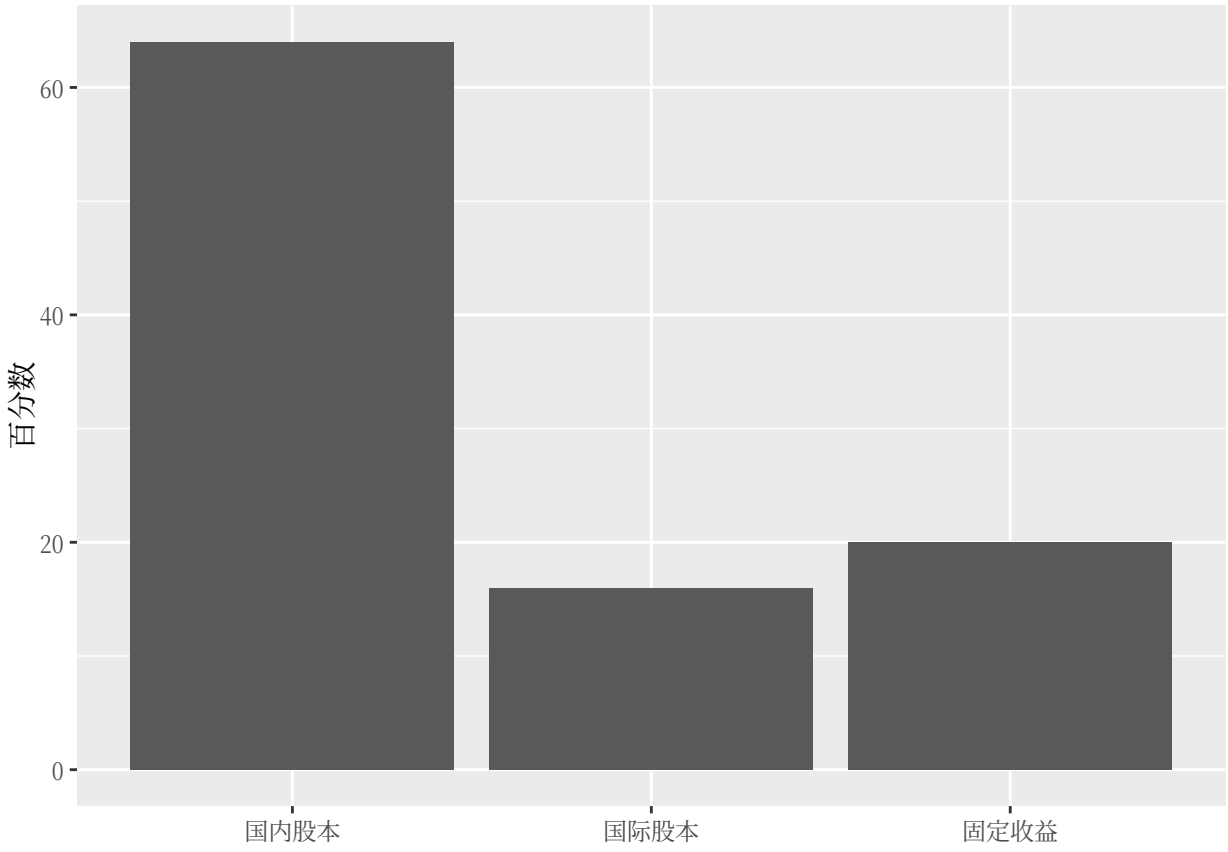


注意各切片默认使用相同的坐标范围。

28.2.1.3 百分数的条形图

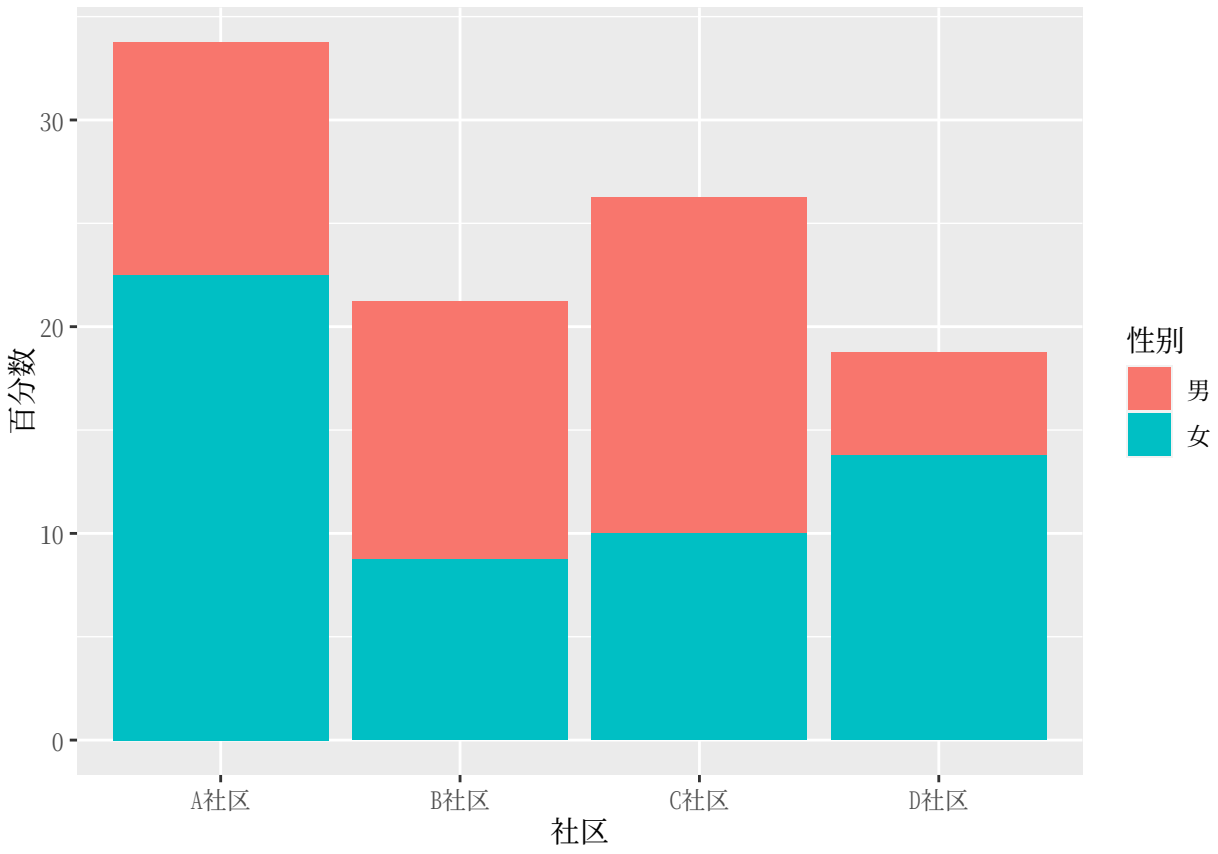
对于只有一个分类变量的条形图，如果 y 轴代表频数，为了转换成比例或者百分比，修改输入数据增加相应的变量，如：

```
d_funds_type |>
  mutate(
    ratio = freq / sum(freq),
    `百分数` = 100*ratio) |>
  ggplot(aes(
    x = fct_relevel(
      type, " 国内股本", " 国际股本", " 固定收益"),
    y = `百分数` )) +
  geom_col() +
  labs(x=NULL)
```



对于有两个分类变量的列联表，如果希望 y 轴表现总百分数，只要修改输入数据集增加分母为总数的百分数，如：

```
d_comm |>
  mutate(`百分数` = `频数` / sum(`频数`) * 100) |>
  ggplot(aes(
    x = `社区`, y = `百分数`, fill=`性别`)) +
  geom_col()
```

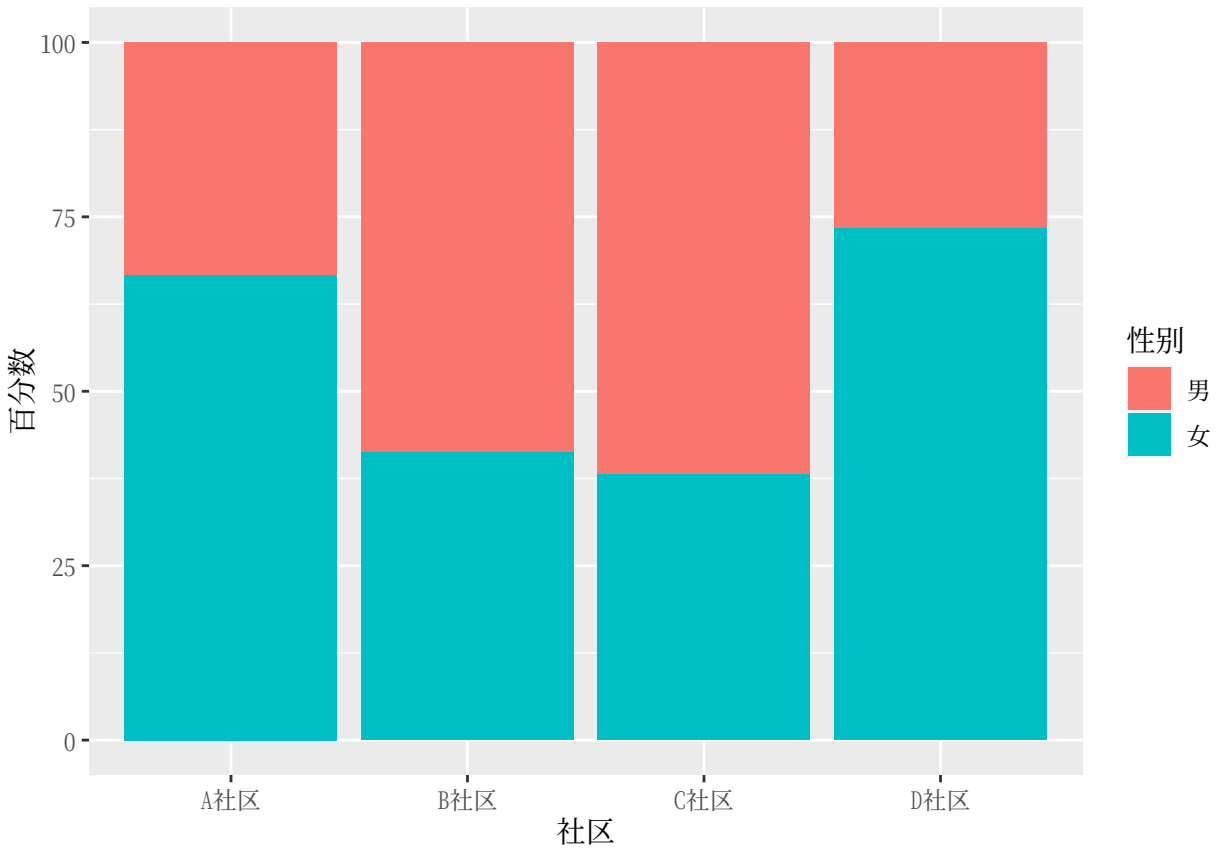


如果 y 轴希望表现每个大类内的百分数，首先增加大类内的百分数，比如下面的程序将社区居民调查数据按社区分为大类，每个社区内计算不同性别的百分数：

```
d_comm |>
  group_by(`社区`) |>
  mutate(`百分数` = `频数` / sum(`频数`) * 100) |>
  ungroup() -> d
```

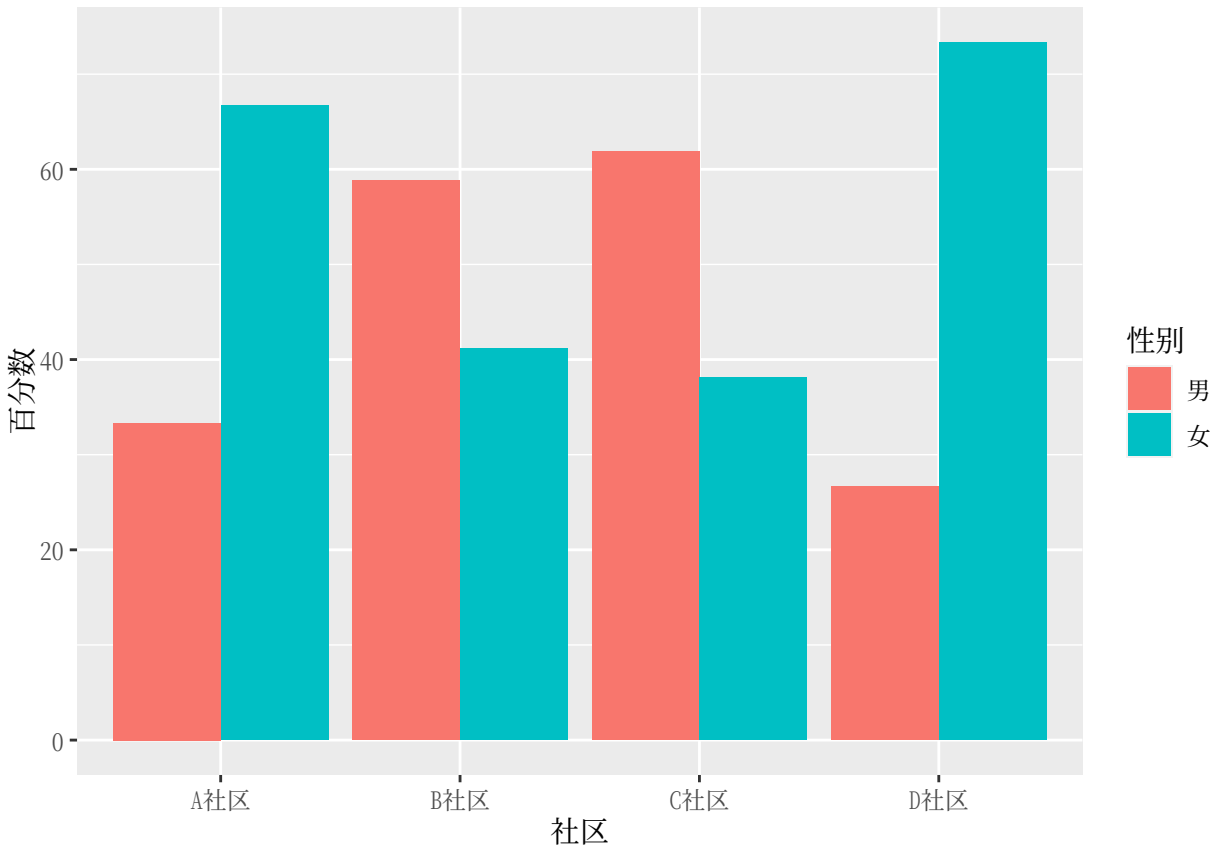
这样，与对频数作图类似就可以做堆叠条形图，每个社区为一个大类，每个大类总计 100%：

```
ggplot(d, aes(
  x = `社区`, fill = `性别`, y = `百分数`)) +
  geom_col()
```



在 `geom_col()` 中加 `position = "dodge"` 可以变成并列条形图，每个社区为一个大类，每个大类中总计 100%：

```
ggplot(d, aes(
  x = `社区`, fill = `性别`, y = `百分数`)) +
  geom_col(position = "dodge")
```



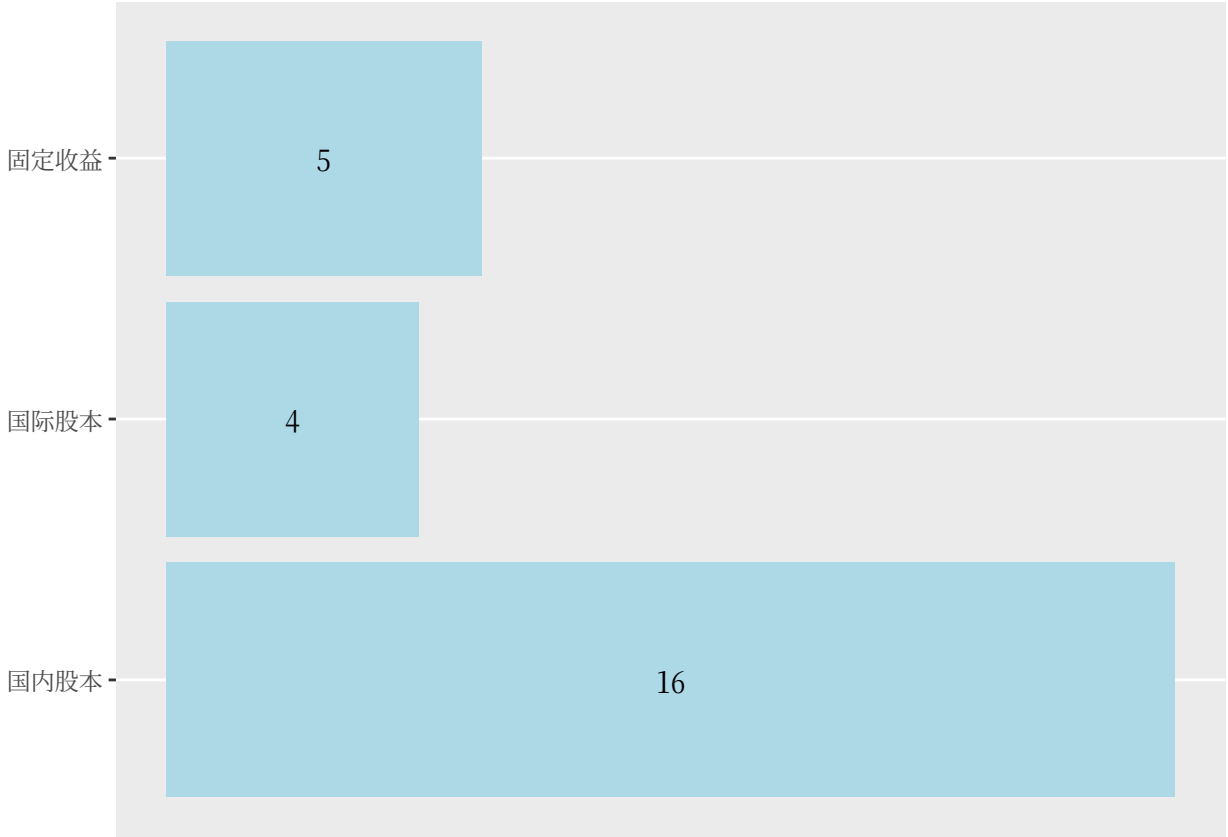
与前一章27.5.1中类似问题相比，对表格数据做比例或者百分数条形图，比基于原始数据直接用 `geom_bar()` 做图要简单而且不易出错。

28.2.1.4 直接标记数字的条形图

比较简单的条形图，可以不使用 y 坐标轴而是对每个条形或者堆叠的每个段标出数字。用 `geom_text()` 函数标数字。如：

```
d_funds_type |>
  mutate(type = fct_relevel(
    type, " 国内股本", " 国际股本", " 固定收益")) |>
  ggplot(aes(
  x = type, y = freq )) +
  geom_col(fill = "lightblue") +
  geom_text(mapping = aes(
    y = freq / 2, label = paste(freq))) +
  scale_y_continuous(breaks = NULL) +
  labs(x = NULL,
       y = NULL) +
```

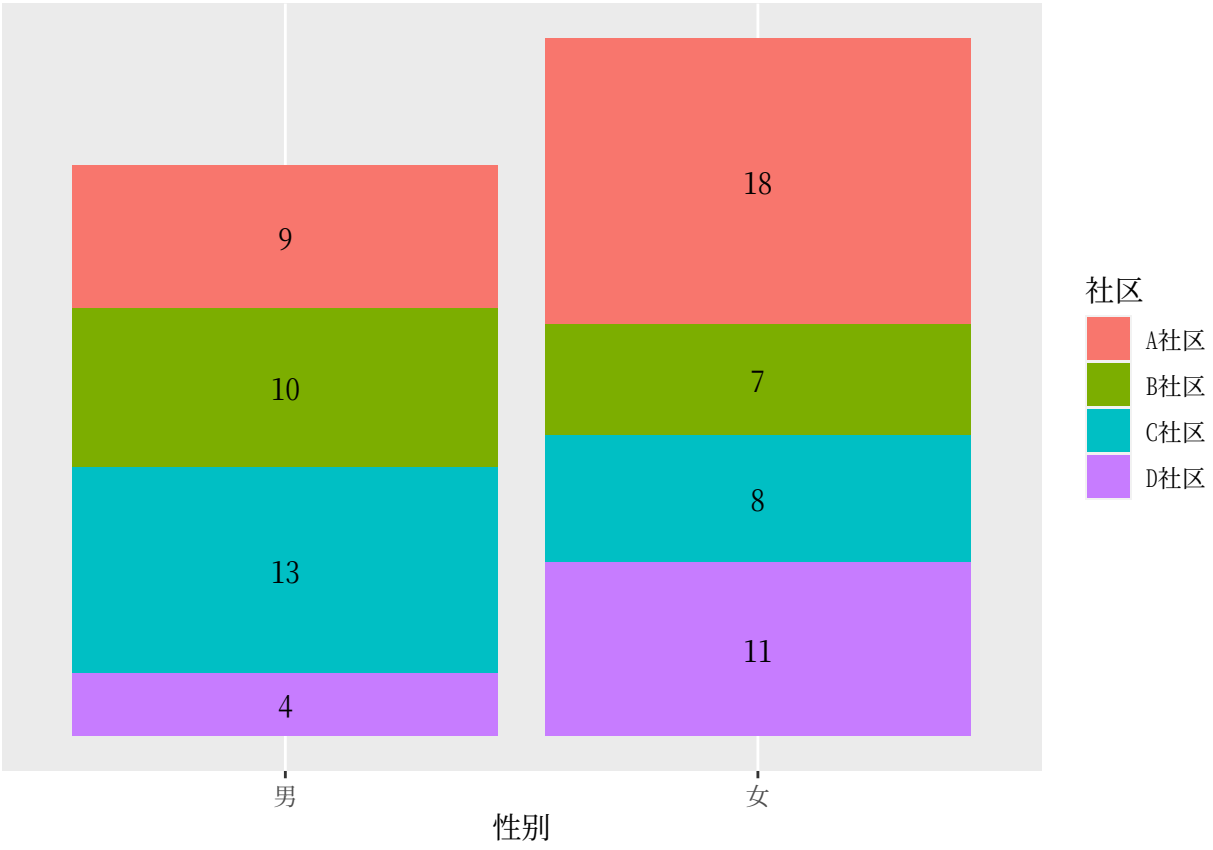
```
  coord_flip()
```



上述程序中，用了 scale_y_continuous() 的 breaks = NULL 选项使得 y 轴（经过对换后画在了 x 轴）不画刻度和刻度值。

堆叠的条形图用数字而非坐标轴的例子如下：

```
d_comm |>
  arrange(`性别`, desc(`社区`)) |>
  group_by(`性别`) |>
  mutate(ylabel = cumsum(`频数`) - 0.5*`频数`) |>
ggplot(aes(
  x = `性别`, fill = `社区`, y = `频数`)) +
  geom_col() +
  geom_text(mapping = aes(
    y = ylabel, label = paste(`频数`)  )) +
  scale_y_continuous(breaks = NULL) +
  labs(y = NULL)
```



上述程序中预先计算了每段的纵坐标值，保存在变量 `ylabel` 中。注意计算过程中的排序与分组操作。

28.2.2 点图

条形图的 y 轴一般都代表数量，最小值从 0 开始，一般不画 y 轴最小值大于 0 的条形图。

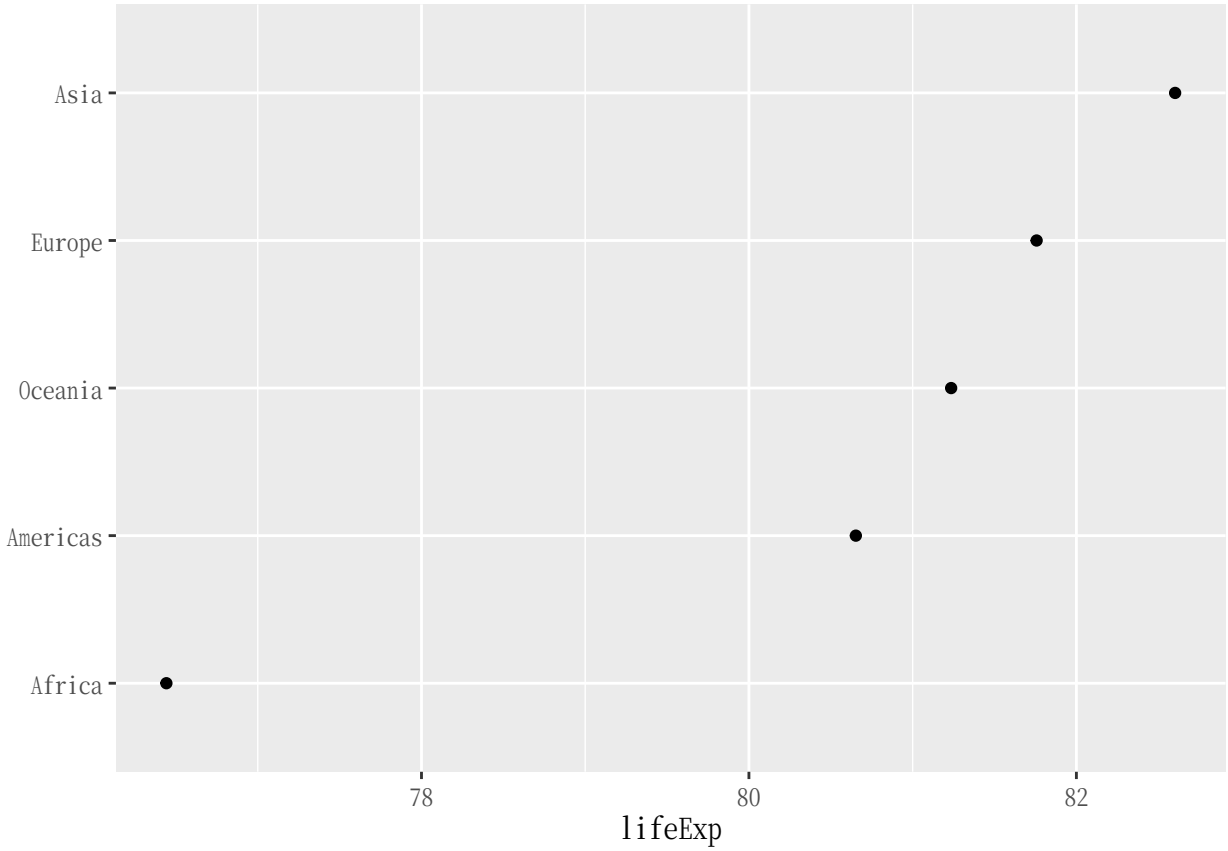
对于不会取 0 的量，可以取 y 轴的范围为某个区间而不必总是从 0 开始。这时，可以用散点位置表示 y 坐标。

考虑 gapminder 包的 gapminder 数据集中各国的平均期望寿命，先求出每个大洲的期望寿命最大值：

```
library(gapminder)
gapminder |>
  select(continent, country, lifeExp) |>
  group_by(continent) |>
  summarise(
    lifeExp = max(lifeExp, na.rm=TRUE),
    .groups = "drop") -> d_gap
```

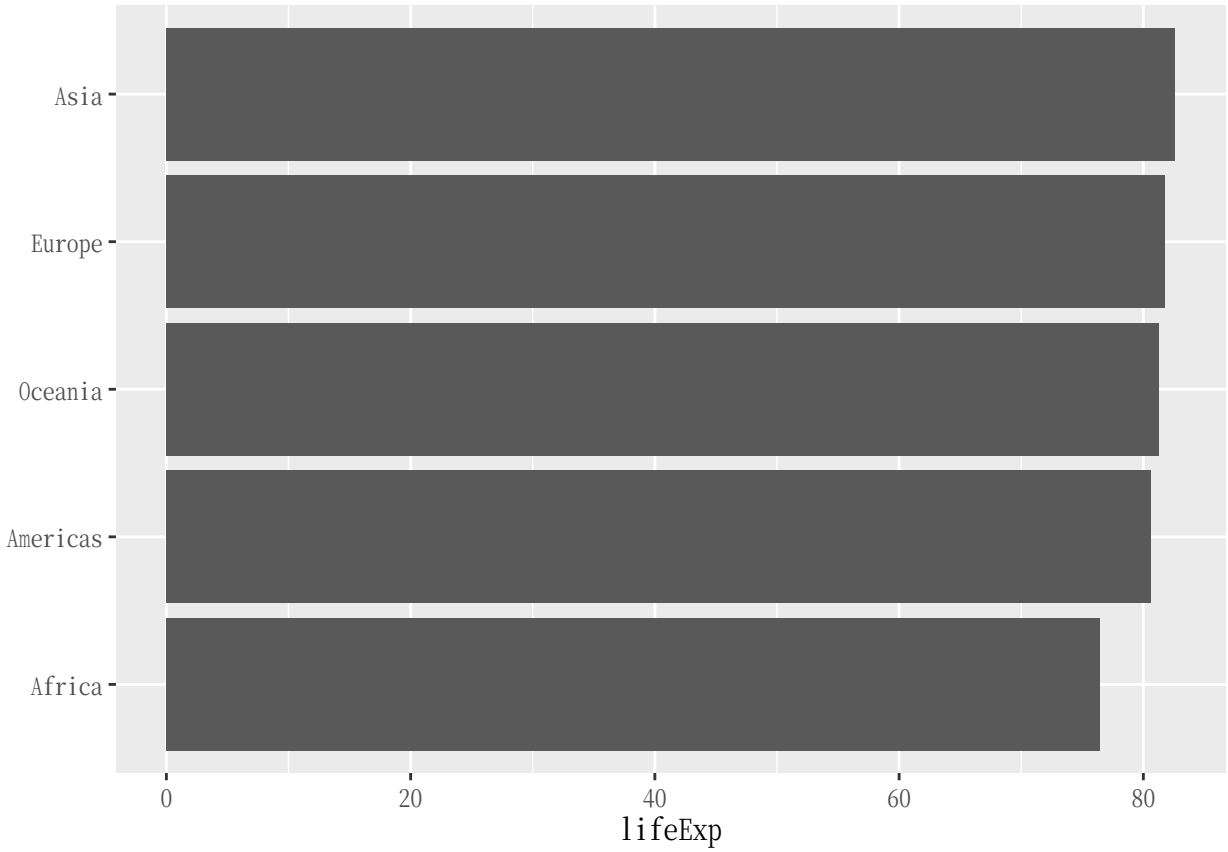
用点图表现各大洲的最大期望寿命：

```
p <- ggplot(d_gap, aes(
  x = fct_reorder(continent, lifeExp),
  y = lifeExp ))
p + geom_point() +
  labs(x=NULL) +
  coord_flip()
```



程序中并没有人为指定 lifeExp 轴的坐标范围，作图程序自动选择了合适的范围。对各大洲按照 lifeExp 的值做了排序。这个图形很好地体现了不同大洲的最大期望寿命的差距。如果使用条形图，因为坐标轴从 0 开始，各个条形的长度会很接近，不利于体现差距：

```
p + geom_col() +
  labs(x=NULL) +
  coord_flip()
```



28.2.3 热力图

如果交叉分类得到许多个交叉组，每组有一个数量需要展示，用堆叠或者并排条形图可能过于复杂，结果很难判读；如果用点图，不同小类之间的区别也不容易认读。

可以用 x 轴和 y 轴分别表示两种分类，在坐标交叉处用色块颜色代表数量，称为热力图。用 `geom_tile()` 作这种图，将数量映射到 fill 维度。

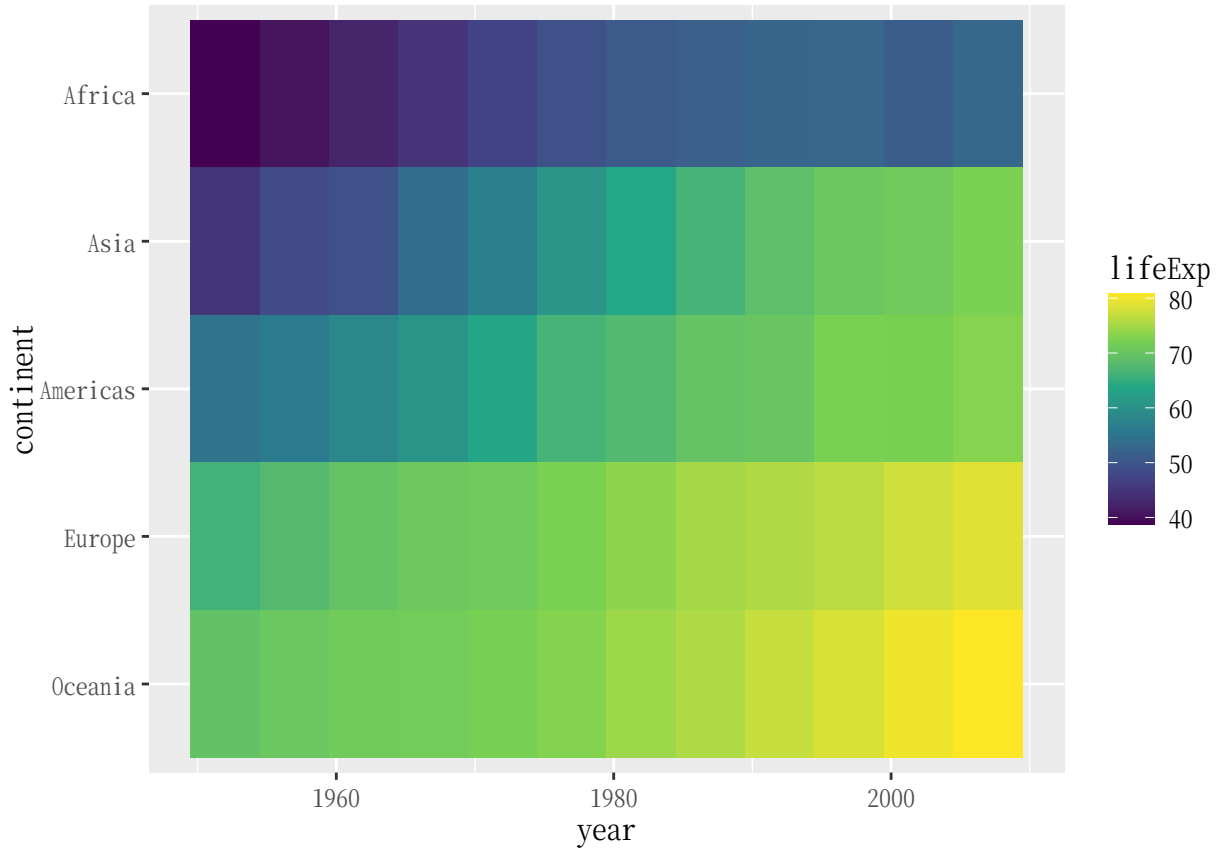
例如，考虑 gapminder 包的 gapminder 数据集中各国的平均期望寿命，先得到每个大洲每年内各国期望寿命的中位数：

```
gapminder |>
  select(continent, year, lifeExp) |>
  group_by(continent, year) |>
  summarise(
    lifeExp = median(lifeExp, na.rm=TRUE),
    .groups = "drop") -> d_gap2a
d_gap2a |>
  filter(year == 2007) |>
  arrange(desc(lifeExp)) -> d_gap2b
```

```
d_gap2a |>
  mutate(
    continent = factor(
      continent,
      levels = d_gap2b$continent)) -> d_gap2
```

上面的程序中将各大洲的次序调整为最后一年的中位期望寿命。作热力图：

```
ggplot(d_gap2, aes(
  x = year, y = continent, fill = lifeExp)) +
  geom_tile() +
  scale_fill_viridis_c()
```



函数 `scale_fill_viridis_c()` 提供了连续变量到颜色的一种映射，在黑白显示时仍适用，且使用色盲的读者。

热力图中不易读出每个交叉组的数量的具体数值，但是比较容易进行组间比较以及展示发展趋势。

28.3 表现分布

对于离散变量，可以用频数、比例、百分数的条形图表现单个离散变量分布，可以用热力图表现两个离散变量的分布。

对于连续型变量，可以用直方图、密度估计图表现单个变量分布，可以对多个变量同时做密度估计图。可以用正态 QQ 图、盒形图、经验分布函数图等表现一元分布。

ggdist 包提供了多种显示一维分布与分布比较的函数。

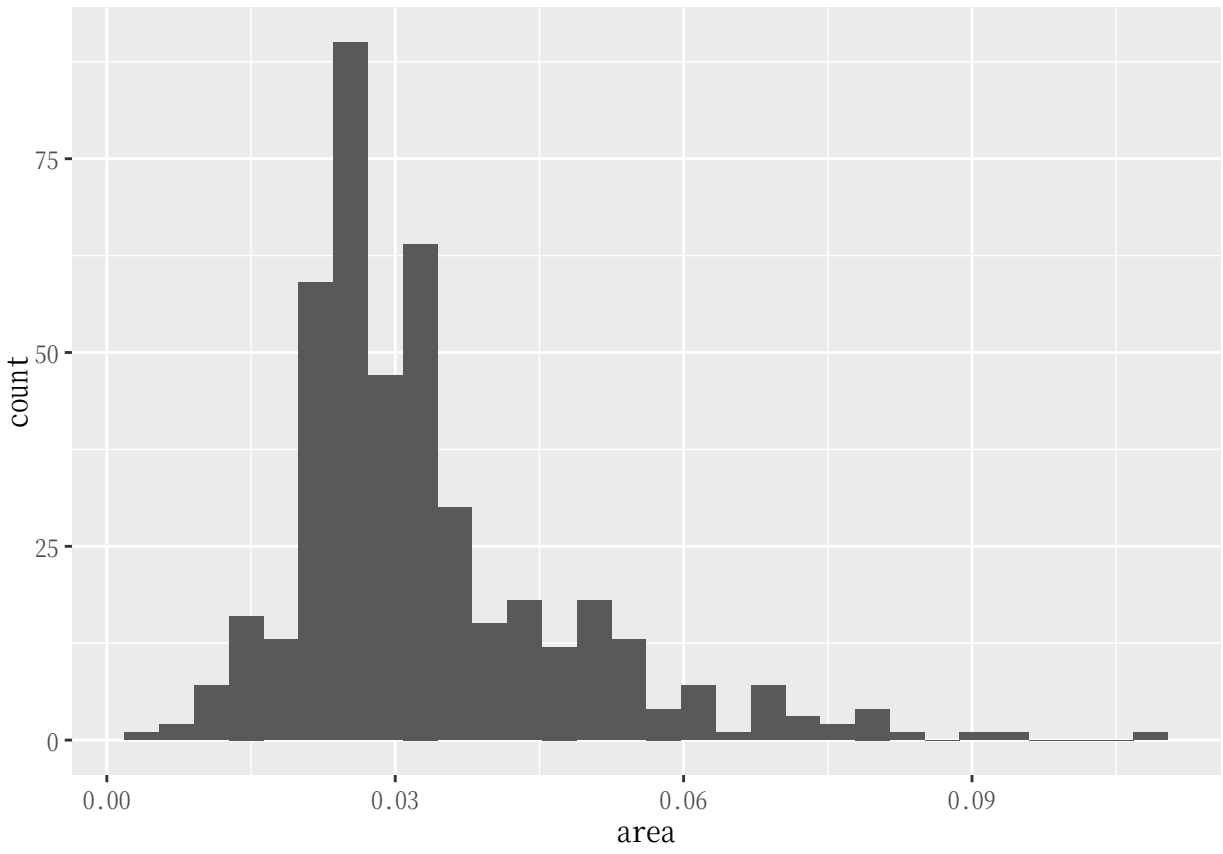
28.3.1 单个一元分布的直方图与密度估计图

直方图是最常用的表现一元连续变量分布的方法。`geom_histogram()` 作直方图，可以自动选取合适的分组个数，也可以人为指定分组个数。直方图的结果受分组个数与分组的开始位置的影响较大，所以使用直方图时应该尝试选用不同的分组数，选择较适当的分组个数。

ggplot2 包中的 `midwest` 数据集包含了美国中西部的一些县的统计数据，如面积（单位：平方英里）。下面的程序对连续取值的数值型变量 area 作频数直方图，自动确定分组个数：

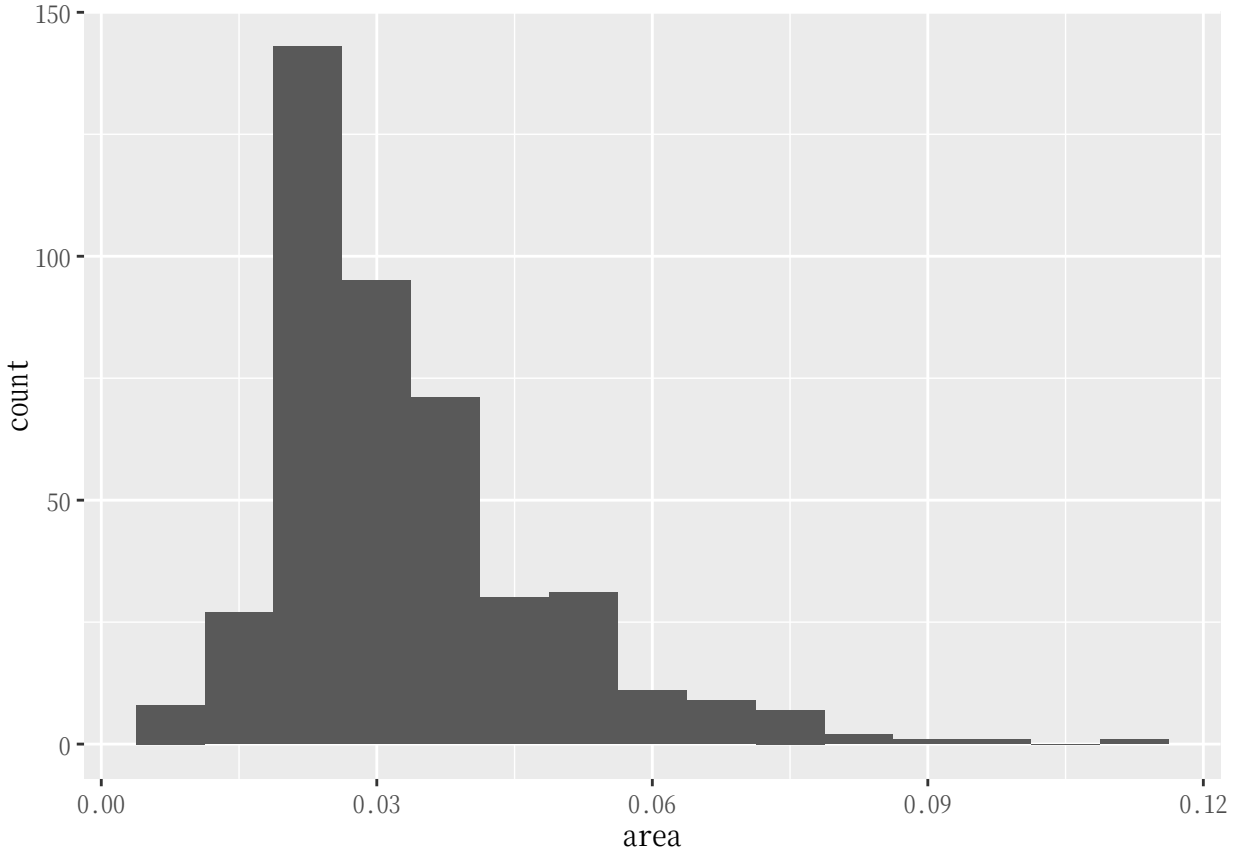
```
p <- ggplot(midwest, aes(
  x = area))
p + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



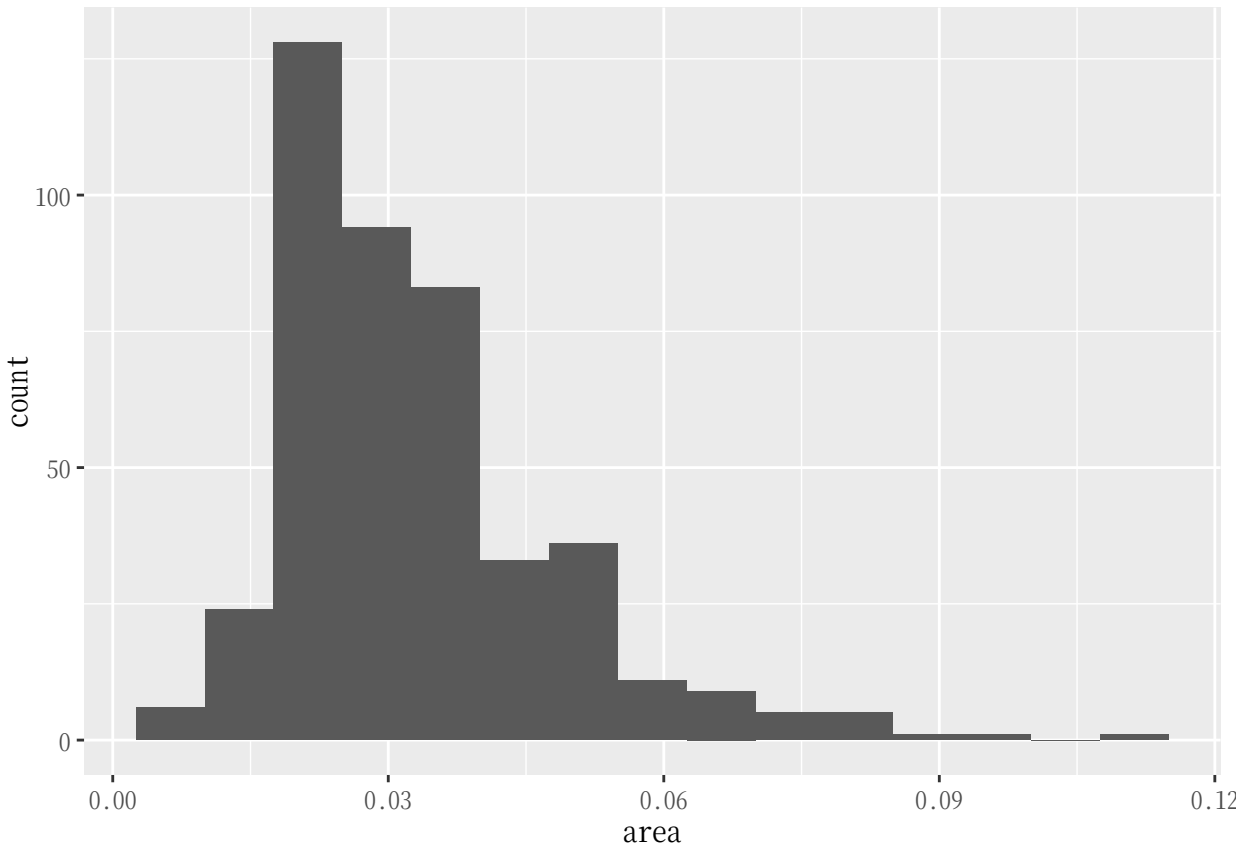
不像 R 基本绘图的 `hist()` 函数，`geom_histogram()` 默认使用 30 个分组，而不是根据样本量大小决定分组，所以一般应该自己指定分组数或者分组宽度。在 `geom_histogram()` 中用 `bins =` 指定分组个数，或者用 `width` 指定分组宽度，如：

```
p + geom_histogram(bins = 15)
```



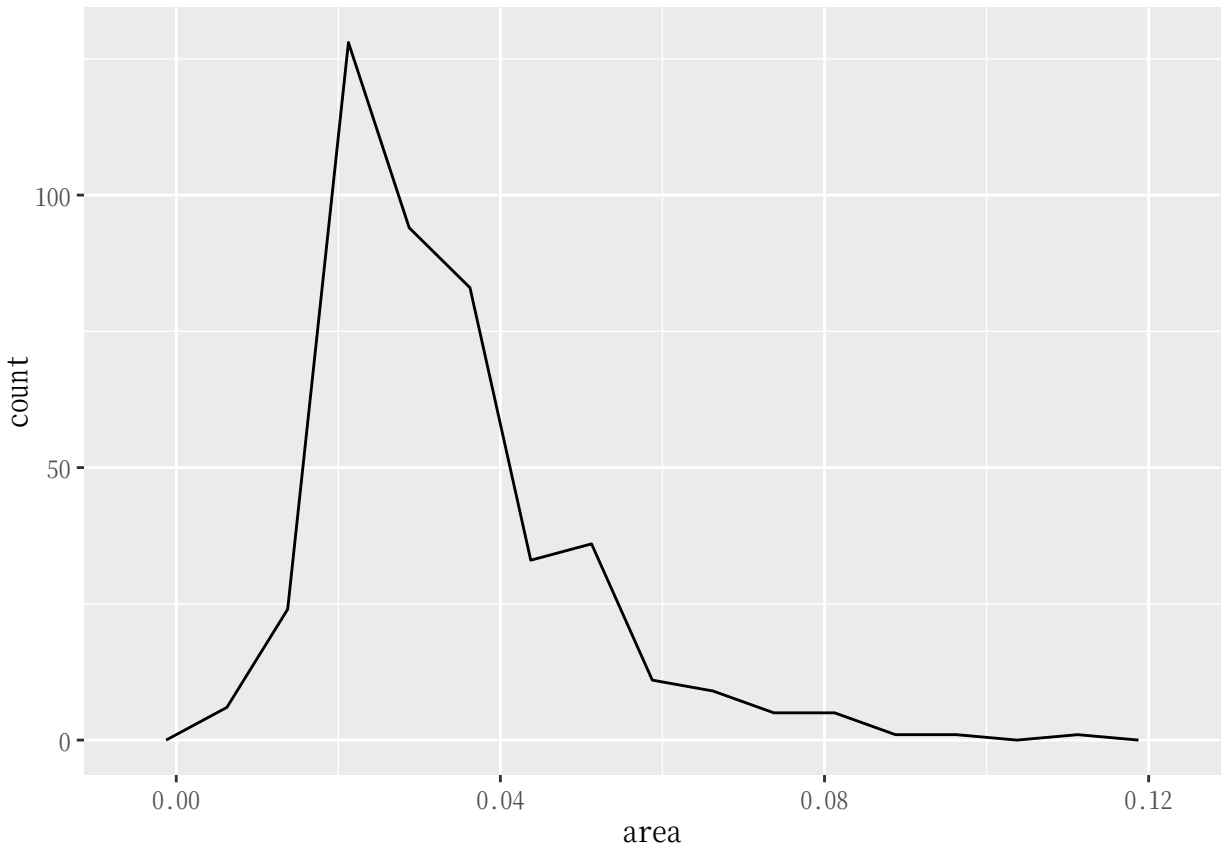
可以用 boundary 或者 center 指定某一个条形的边界或者中心，用 breaks 指定所有分点位置。如

```
p + geom_histogram(bins = 15, boundary = 0.0025)
```



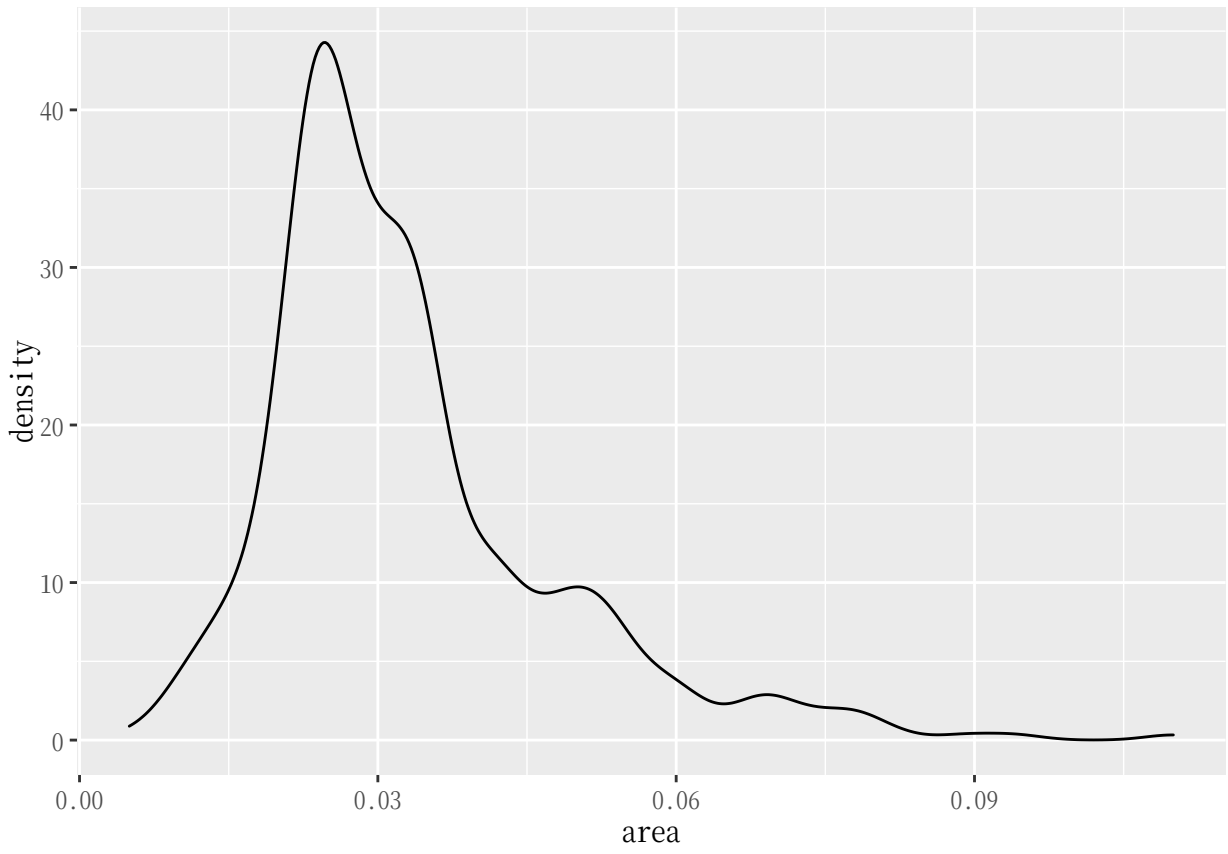
可以将直方图画成连线形式，用 `geom_freqpoly()` 函数，如上图可以改为：

```
p + geom_freqpoly(bins = 15, boundary = 0.0025)
```



上图虽然是连线形式，但本质上还是一个阶梯函数的不同画法。`geom_density()` 可以对连续变量先做核密度估计，然后绘制密度估计曲线，如：

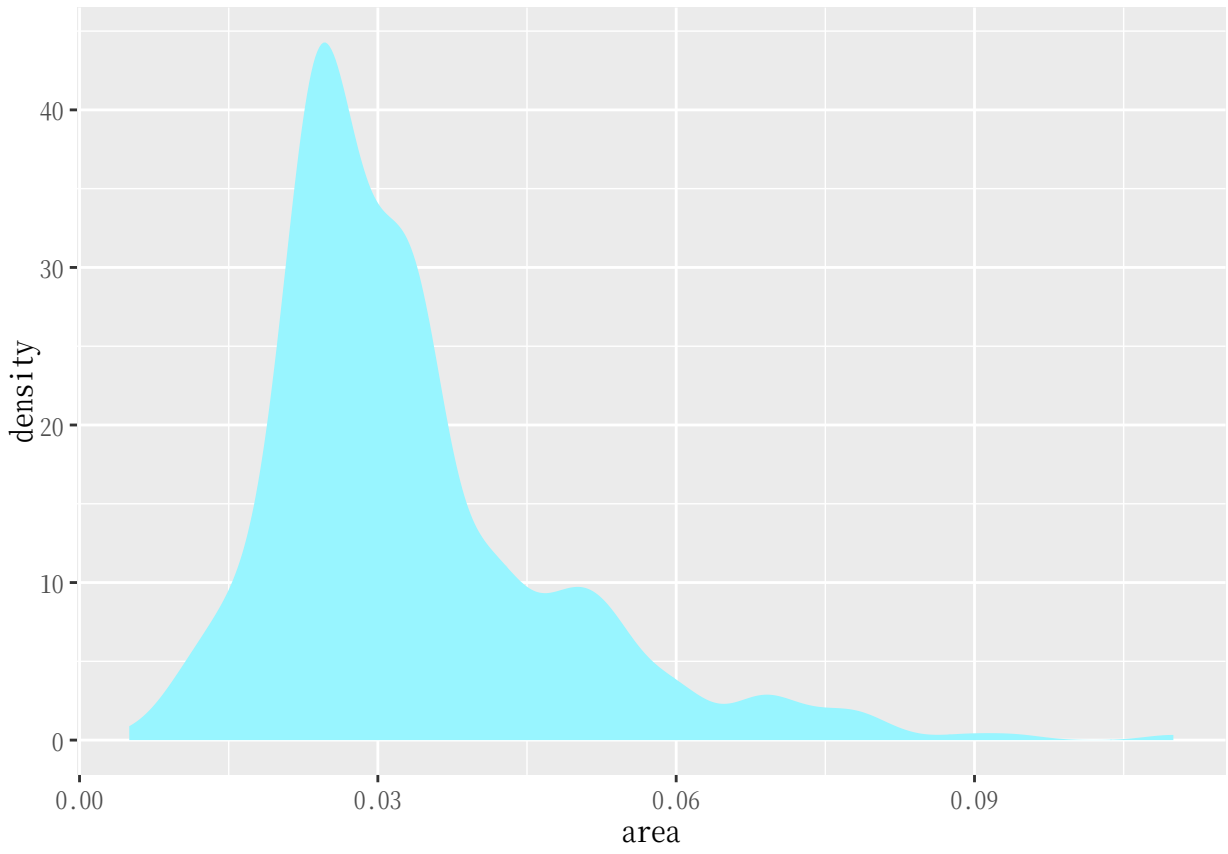
```
p + geom_density()
```



密度曲线估计受到窗宽 `bw` 与核函数 `kernel` 的影响。默认核函数为高斯核（标准正态分布密度作为核函数）。

除了估计曲线，还可以用涂色区域表示密度估计，用 `geom_area(stat = "density")` 函数：

```
p + geom_area(stat = "density", fill = "cadetblue1")
```

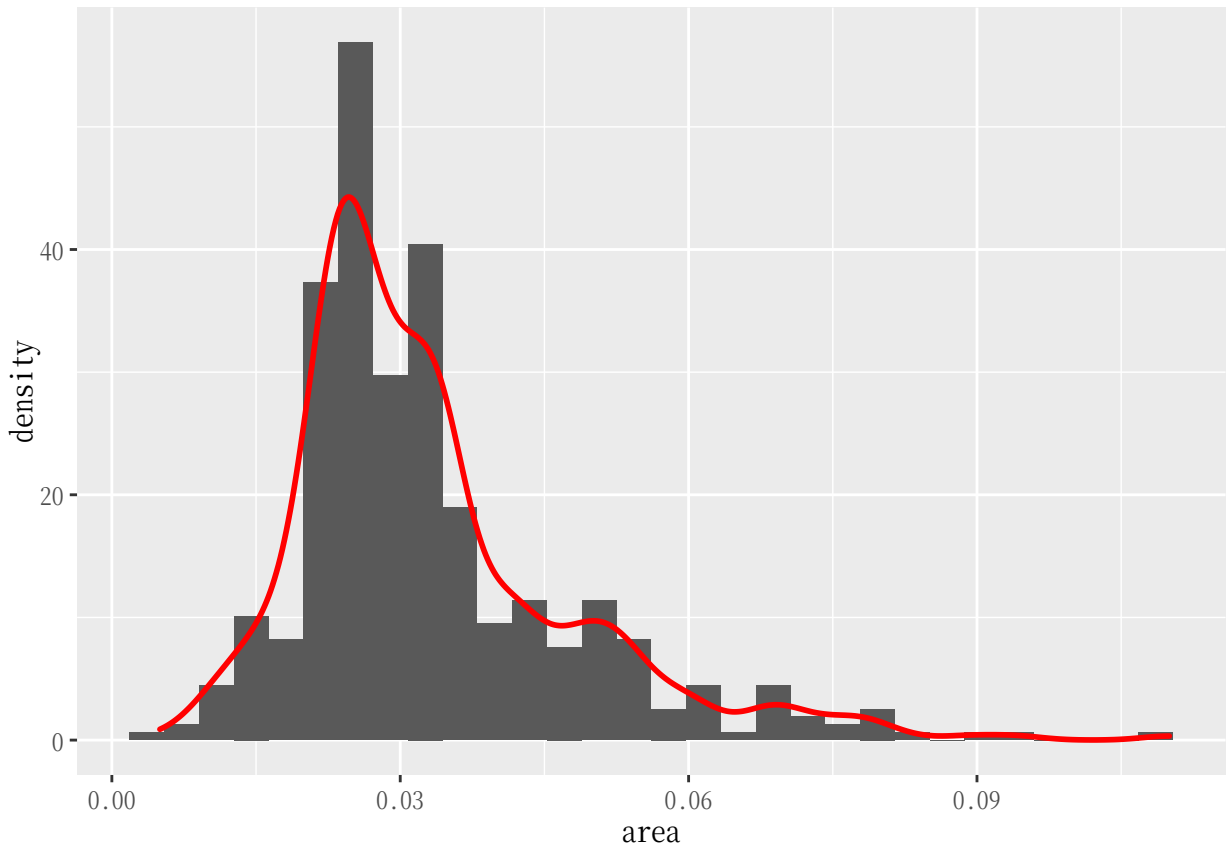


因为核密度估计一般是连续曲线，所以估计的曲线很可能在密度为零的地方估计为正数。ggplot2 在设计时已经考虑了避免这个问题，但是用户自己用 R 的 `density()` 函数作核密度估计时还有这个问题。

可以将直方图与密度估计图合并在一起，这时直方图的纵坐标需指定为密度估计，这样两种图的纵坐标才处于相同的范围：

```
p + geom_histogram(mapping = aes(y = stat(density))) +
  geom_density(color = "red", size = 1)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

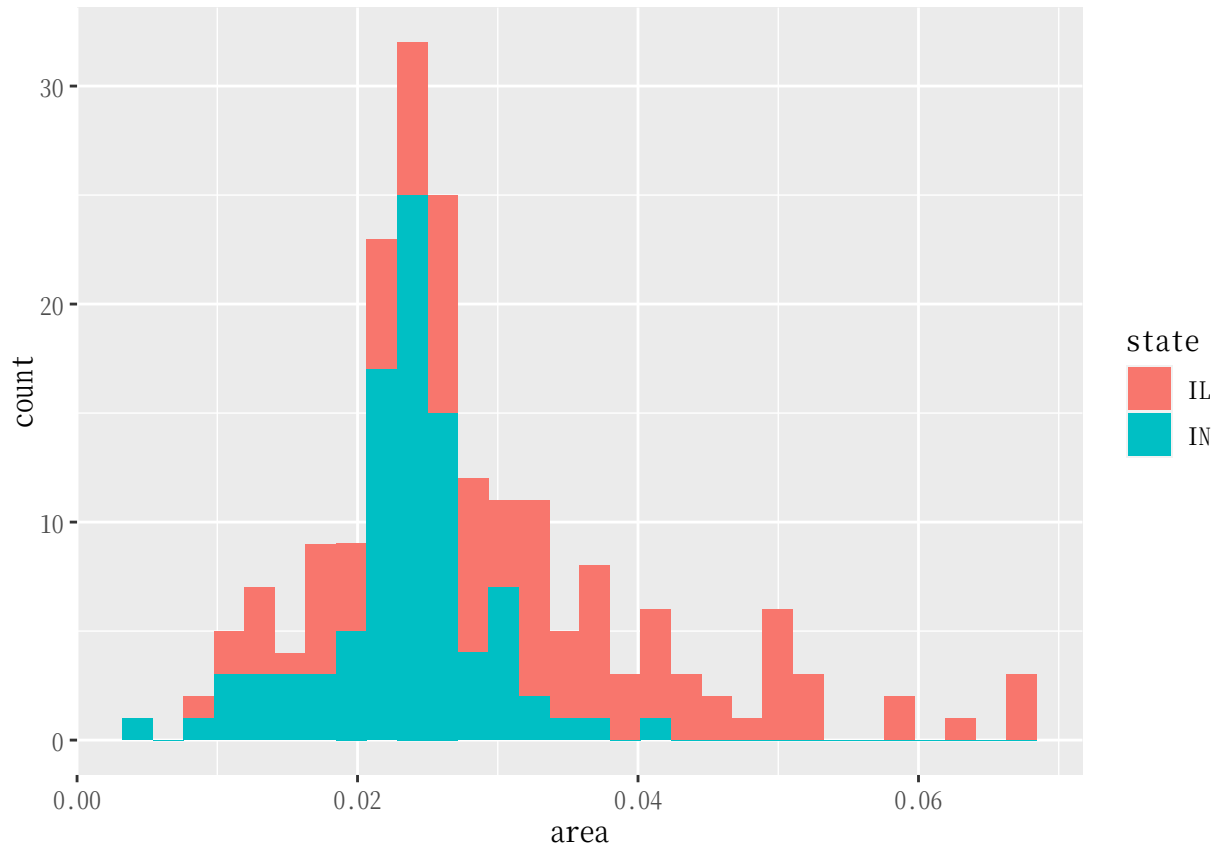


28.3.2 多个一元分布的直方图和密度图形

考虑 ggplot2 包的 midwest 数据集，这是美国中西部若干个县的数据。选取其中的 IL 和 IN 两个州的数据，只要将州名映射为 `fill` 维，就可以在 `geom_histogram()` 或者 `geom_line(stat = "density")` 对各县面积作直方图时按不同州分段显示：

```
midwest |>
  filter(state %in% c("IL", "IN")) |>
  ggplot(aes(
  x = area, fill = state)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

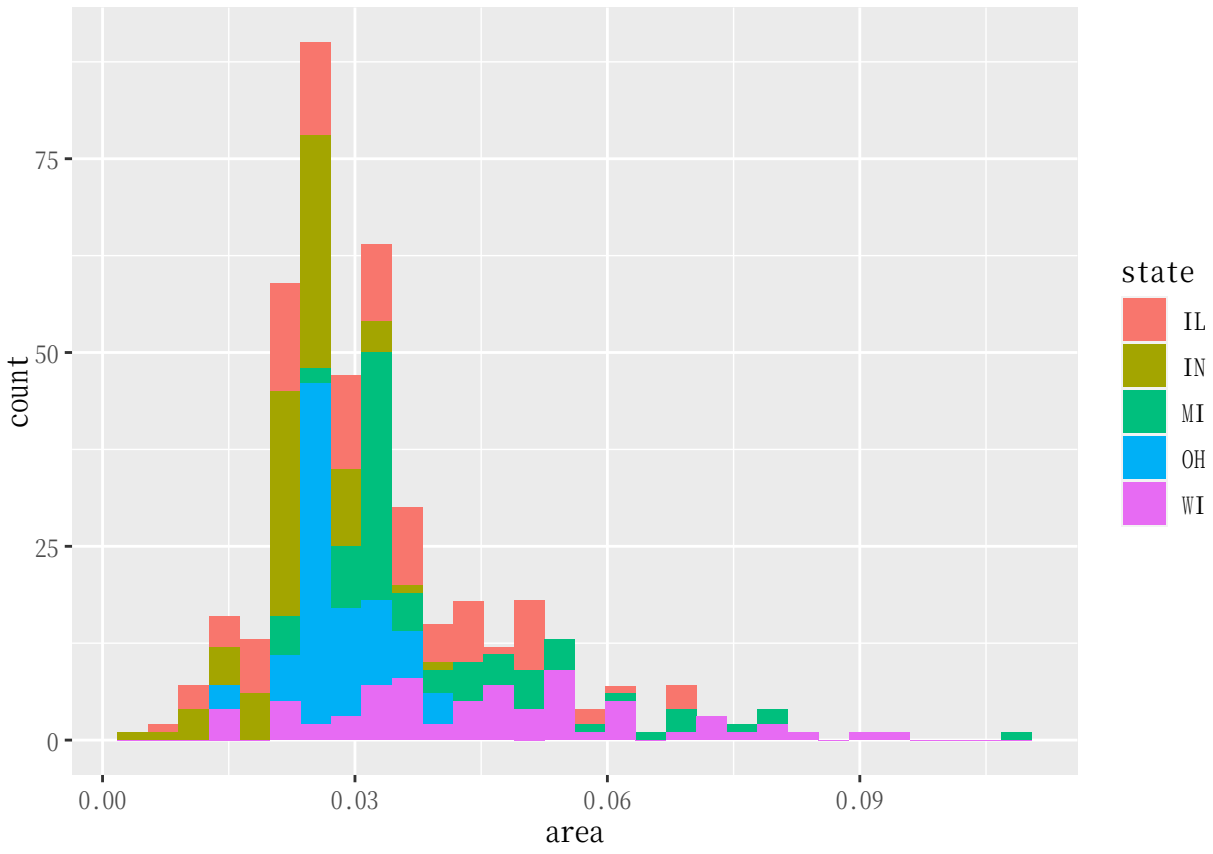


可以看出，IN 州的县面积偏小，IL 州的县面积较大。这样的条形分段的直方图的缺点是，下面的段的直方图形状是清楚的，但是上面一段（上图中为 IL 州）的直方图则看不出形状。

如果将数据中所有的 5 个州都包含并在直方图中分段，图形就变得更难认读：

```
p <- ggplot(midwest, aes(
  x = area))
p + geom_histogram(mapping = aes(fill = state))
```

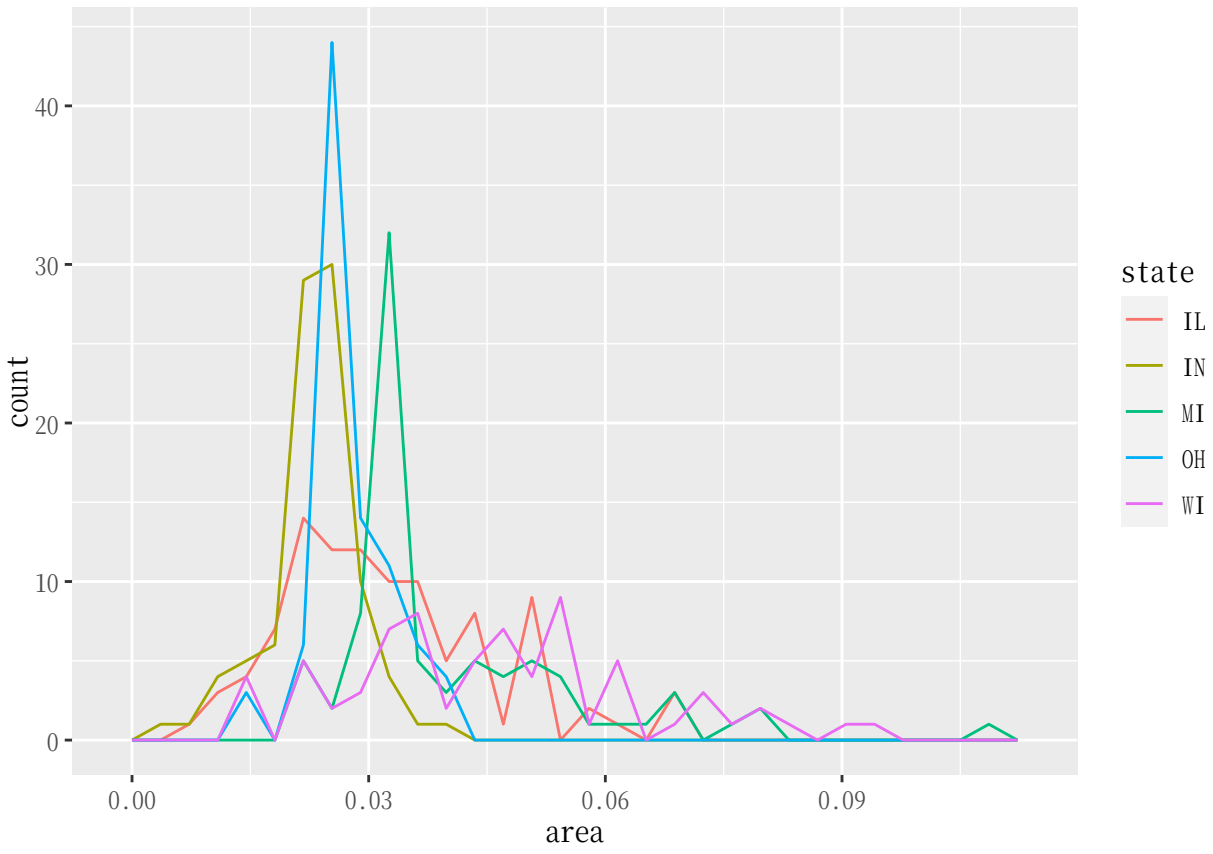
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



可以用折线方式画直方图，不同州用不同颜色：

```
p + geom_freqpoly(mapping = aes(color = state))
```

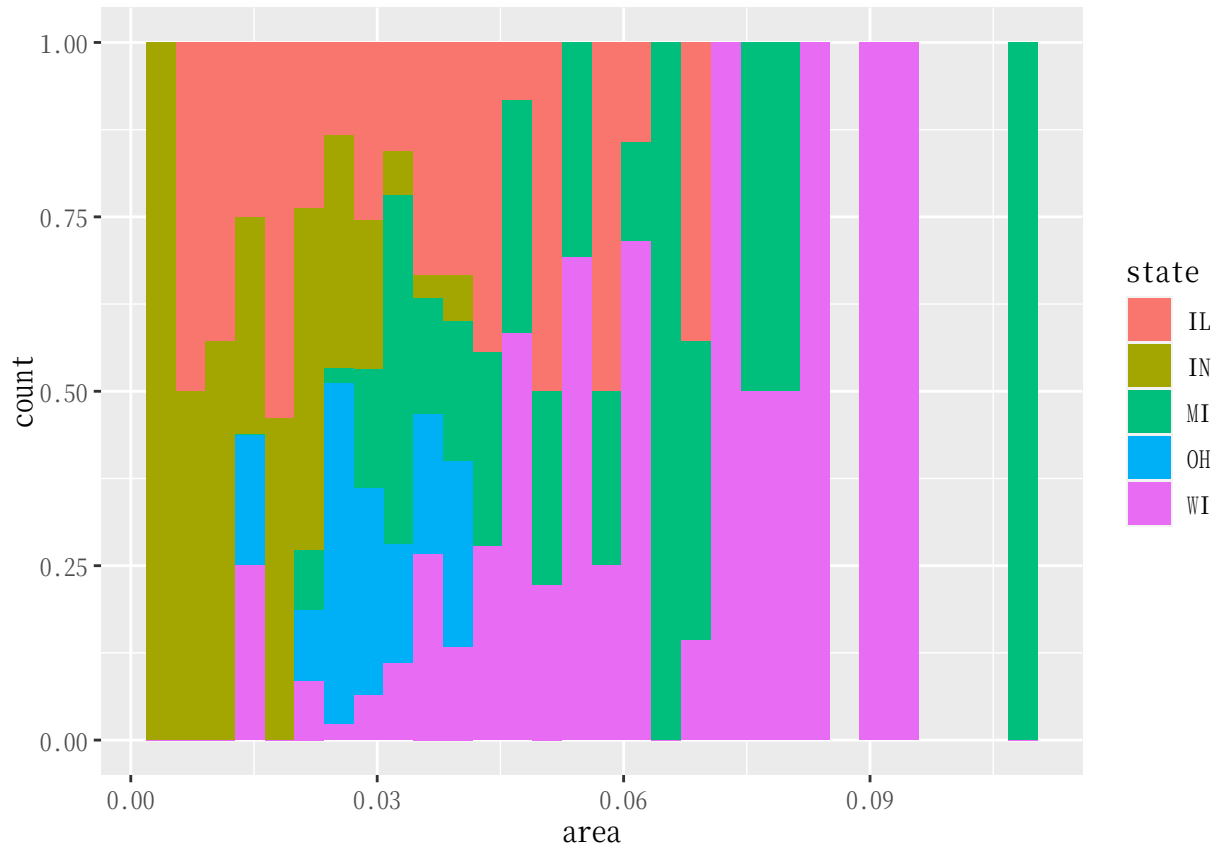
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



可以将州映射到 fill，给 geom_histogram() 加 position = "fill" 选项：

```
p + geom_histogram(
  mapping = aes(fill = state),
  position = "fill",
  na.rm = TRUE)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

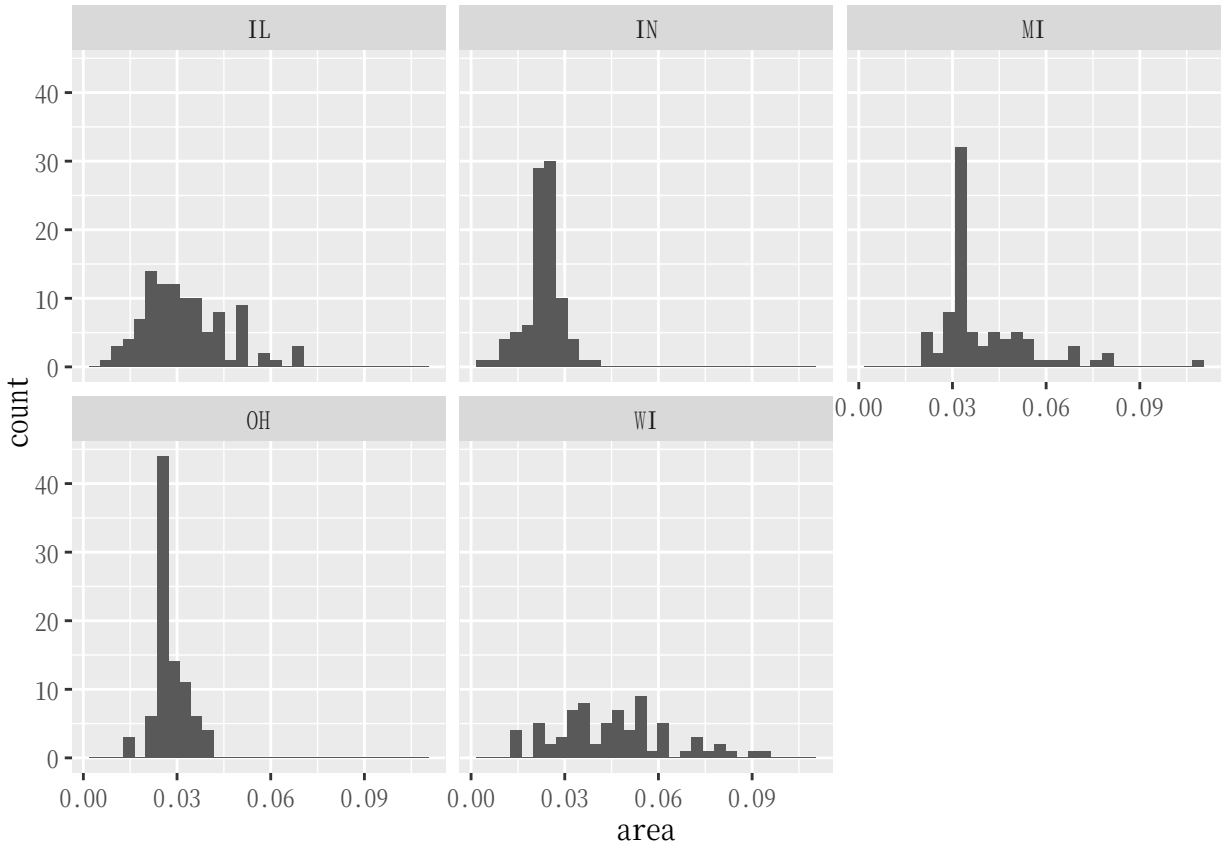


上图的每个条形长度都拉平了，显示的是按 area 分组后每组内不同州的比例大小。

可以将不同的州分配到不同的切片，如：

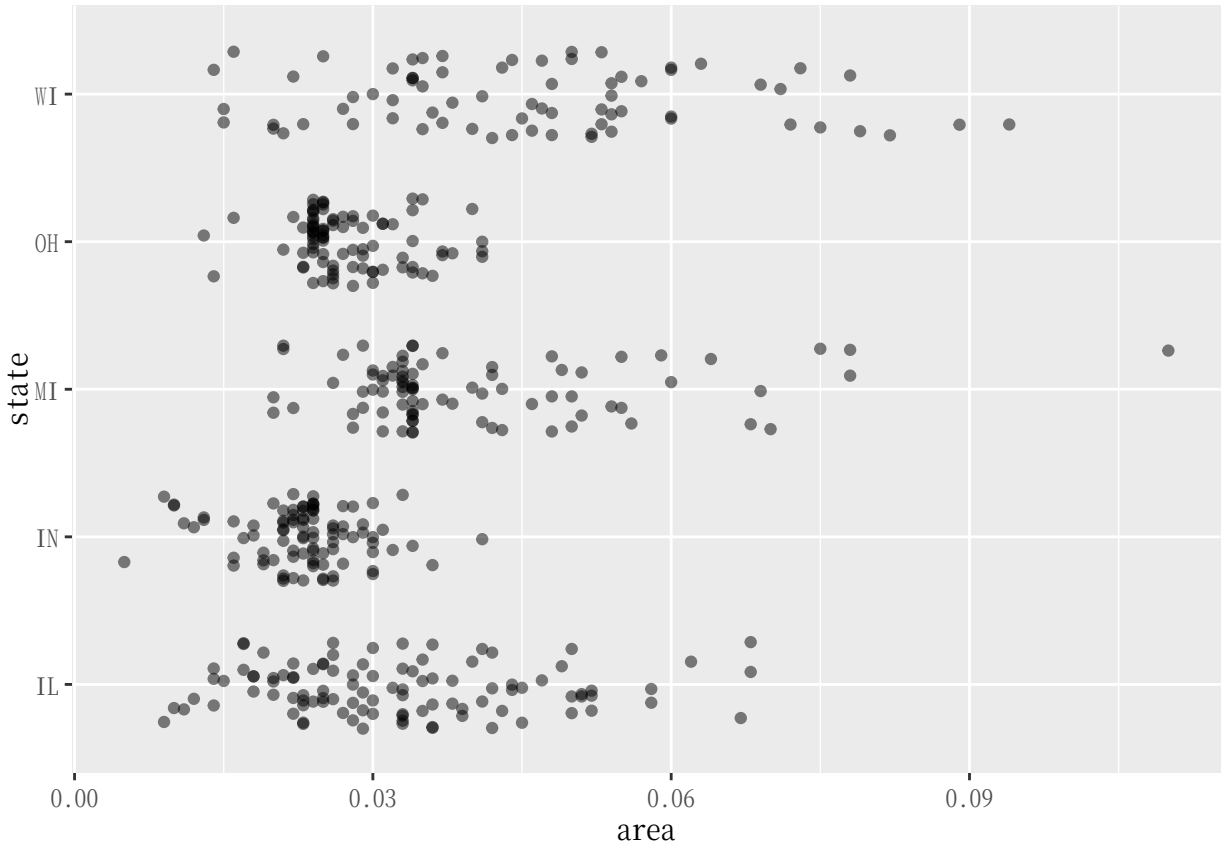
```
ggplot(midwest, aes(
  x = area)) +
  geom_histogram() +
  facet_wrap(~ state)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



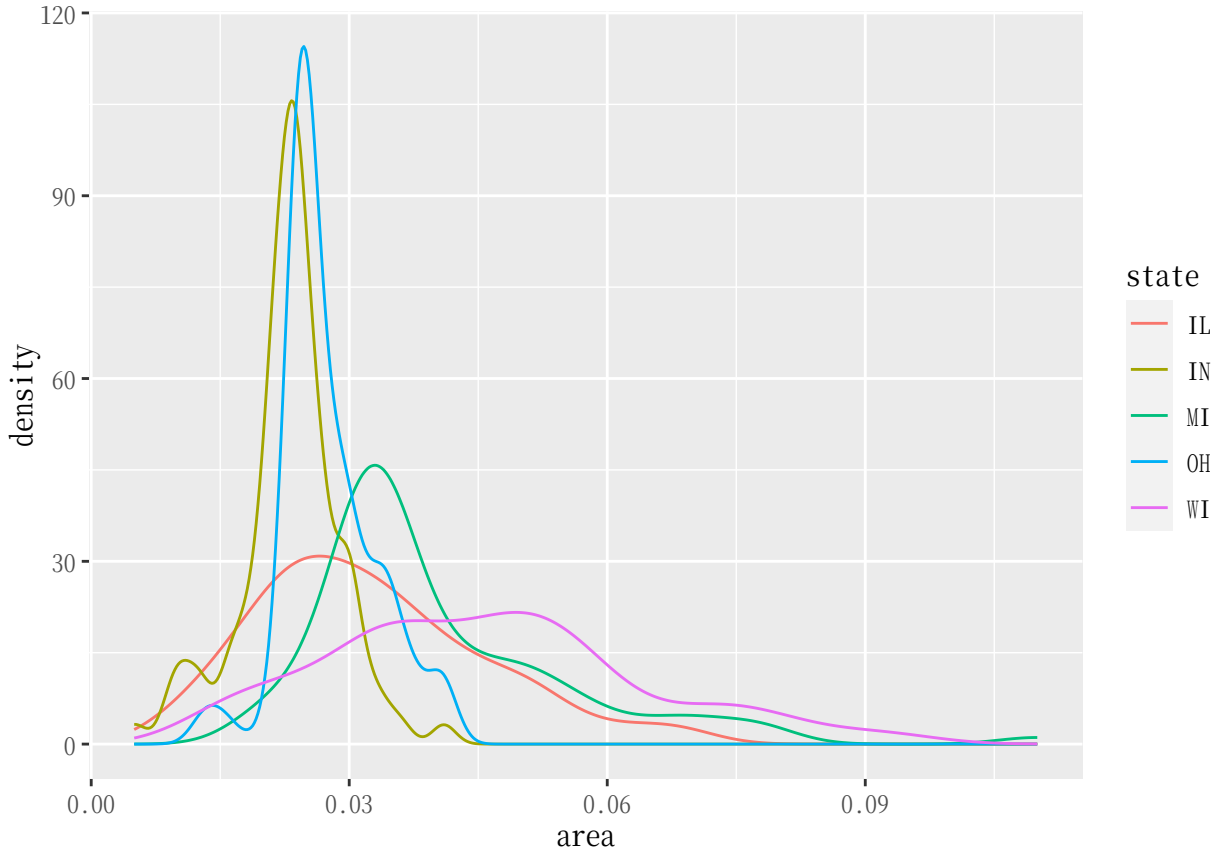
为了表现多个组的分布，也可以将各组的具体数据用散点画出，并添加随机扰动使其不重叠：

```
ggplot(midwest, aes(
  x = area, y = state)) +
  geom_jitter(width = 0, height = 0.3, alpha = 0.5)
```



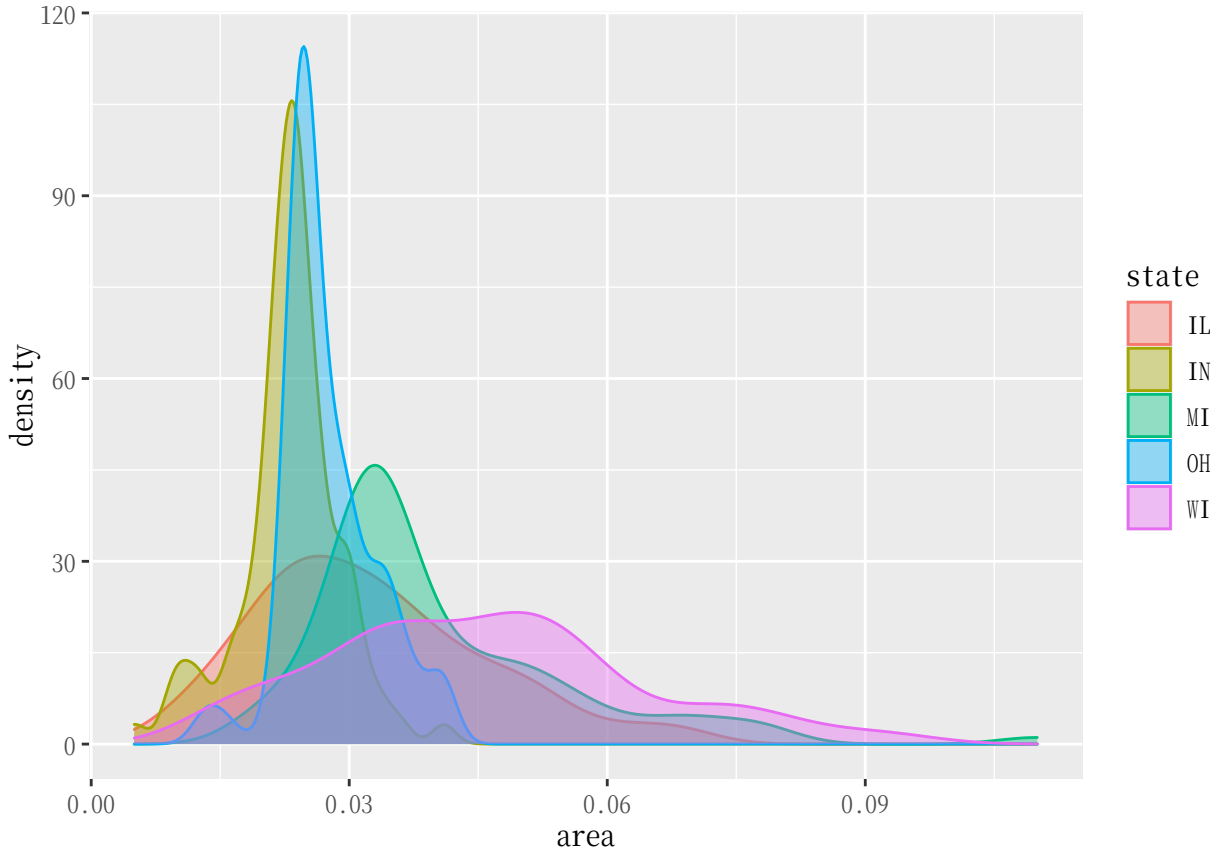
对每个州作密度估计并将多条密度曲线画在同一坐标系可以较好地反映不同州的县面积分布：

```
ggplot(midwest, aes(
  x = area, color = state)) +
  geom_line(stat = "density")
```



增加涂色，用透明度 alpha 使其可以重叠显示：

```
ggplot(midwest, aes(
  x = area, color = state, fill = state)) +
  geom_density(alpha = 0.4)
```



可以利用 `geom_text()` 或者 `geom_text_repl()` 将各曲线的标签标在曲线附近，就可以取消图例显示。

28.3.3 经验分布函数图

直方图和密度估计图能够比较直观地了解数据的分布情况，但是会受到窗宽、分组位置的影响，所以可以看成是对数据的某种建模结果。可以将所有数据观测值画出来，用适当的透明度可以表现点的稠密程度，但是当数据量特别大的时候这种方法就不合适，而且对数据的概括性描述总是必要的。经验分布函数图很好地描述和概括了数据的分布，从中可以很容易得到样本分位数的估计，而且不会受到任何参数选择影响。一组观测值 $X_1, X_2, \ldots, X_n$ 的经验分布函数定义为

$$F_n(x) = \frac{\#\{X_i : X_i \le x\}}{n} = \frac{1}{n}\sum_{i=1}^{n} I_{(-\infty,x]}(x)$$

其中 $\#\{\ldots\}$ 表示集合元素计数，$I_A(x)$ 是集合 A 的示性函数，当 $x \in A$ 时取 1，否则取 0。

设有如下的 10 位学生的成绩：

```
scores <- c(63, 72, 74, 79, 82, 82, 87, 89, 90, 90)
scores
```

```
##  [1] 63 72 74 79 82 82 87 89 90 90
```

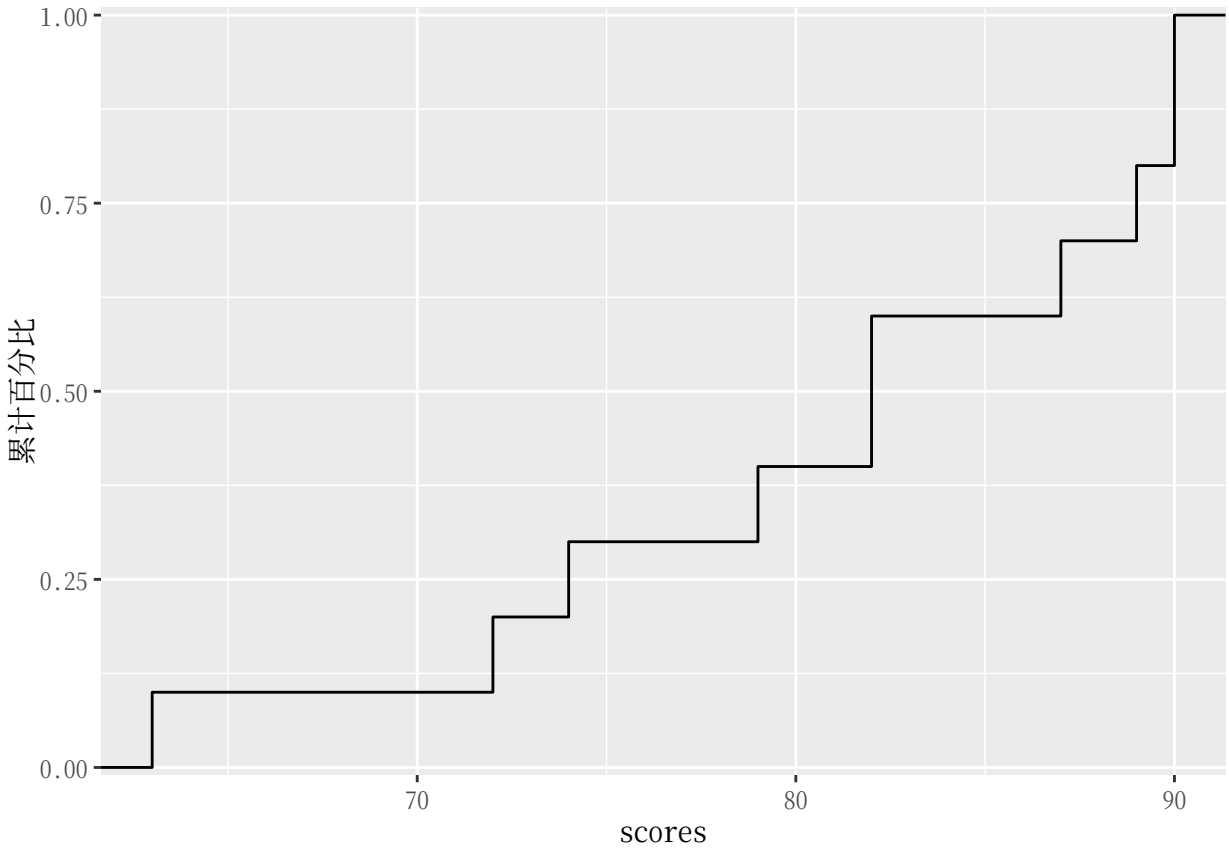
可以制作如下的经验分布函数表格：

```
cdf_table <- function(x){
  x <- sort(x)
  n <- length(x)
  tab <- unname(c(table(x)))
  pct = tab / n
  d <- data.frame(
    x = sort(unique(x)),
    freq = tab,
    pct = pct,
    cumfreq = cumsum(tab),
    cumpct = cumsum(pct))
  d
}
knitr::kable(cdf_table(scores))
```

x	freq	pct	cumfreq	cumpct
63	1	0.1	1	0.1
72	1	0.1	2	0.2
74	1	0.1	3	0.3
79	1	0.1	4	0.4
82	2	0.2	6	0.6
87	1	0.1	7	0.7
89	1	0.1	8	0.8
90	2	0.2	10	1.0

作经验分布函数 (ECDF) 图如下：

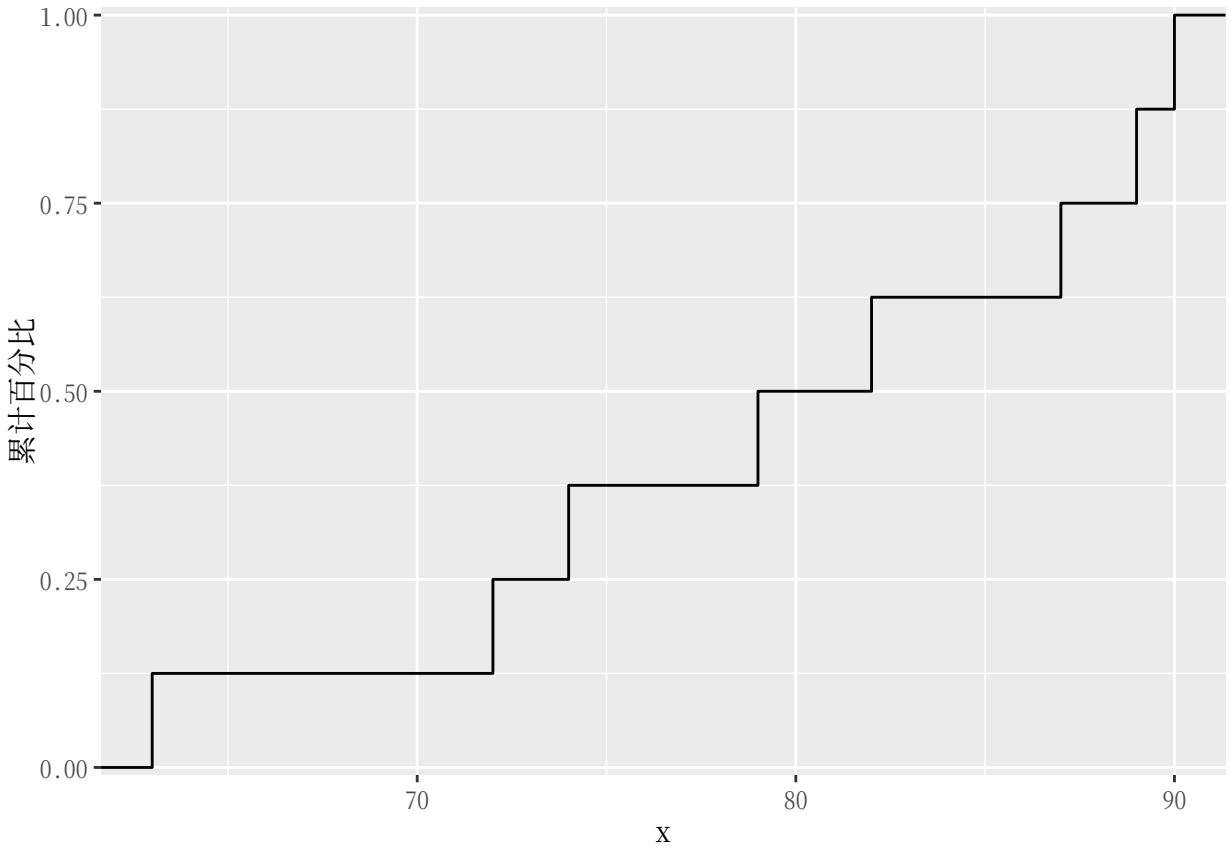
```
d <- data.frame(
  scores = c(63, 72, 74, 79, 82, 82, 87, 89, 90, 90))
ggplot(d, aes(
  x = scores, y = ..y.. )) +
  stat_ecdf(geom = "step") +
  scale_y_continuous(
    limits = c(-.01, 1.01),
    expand = c(0, 0),
    name = " 累计百分比")
```



从上图中，可以看出 80 分及以下的人数 40% 左右，最高分的 25% 学生分数至少为 88 分。当然，准确的数字还需要查看上面的频数分布表。

作经验分布函数图，用了 `stat_ecdf()` 统计函数，其中指定 `geom="step"` 选项，并将 y 轴映射为统计函数的结果`..y..`。如果已经有经验分布函数，则可以直接做阶梯函数图：

```
scores = c(63, 72, 74, 79, 82, 82, 87, 89, 90, 90)
dfreq <- cdf_table(scores)
ggplot(dfreq, aes(
  x = x, y = cumpct )) +
  stat_ecdf(geom = "step") +
  scale_y_continuous(
    limits = c(-.01, 1.01),
    expand = c(0, 0),
    name = " 累计百分比")
```



现实中有许多数据的分布呈现严重的偏态，即有比较多的值偏离数据的中心。比如，全国各省的人口数、居民收入，全世界各国的 GDP，等等。

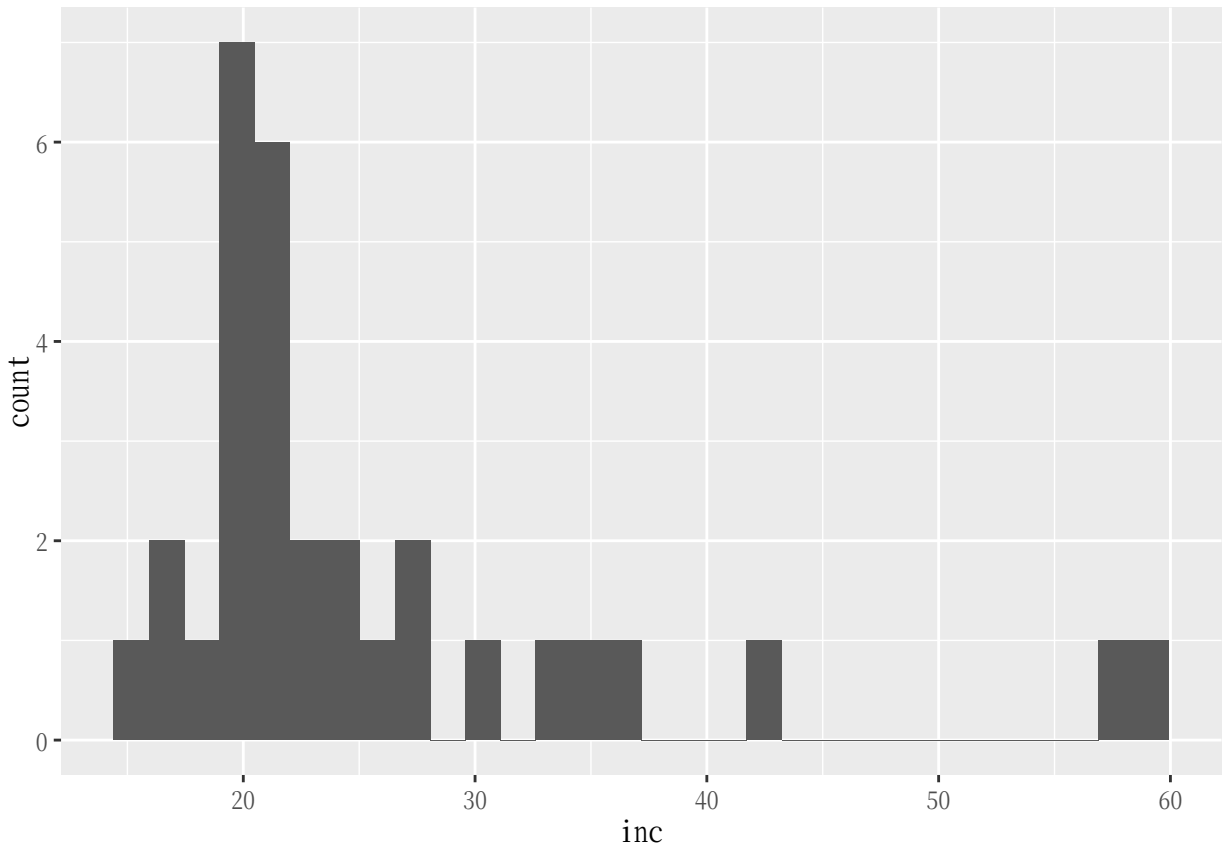
下面是全国 31 个省（不含港澳台）2017 年的人口（单位：万人）、居民年均可支配收入（单位：千元）数据，来自统计年鉴：

province	pop	inc
北京	2171	57
天津	1557	37
河北	7520	21
山西	3702	20
内蒙古	2529	26
辽宁	4369	28
吉林	2717	21
黑龙江	3789	21
上海	2418	59
江苏	8029	35
浙江	5657	42
安徽	6255	22
福建	3911	30
江西	4622	22
山东	10006	27
河南	9559	20
湖北	5902	24
湖南	6860	23
广东	11169	33
广西	4885	20
海南	926	23
重庆	3075	24
四川	8302	20
贵州	3580	17
云南	4801	18
西藏	337	15
陕西	3835	20
甘肃	2626	16
青海	598	19
宁夏	682	21
新疆	2445	20

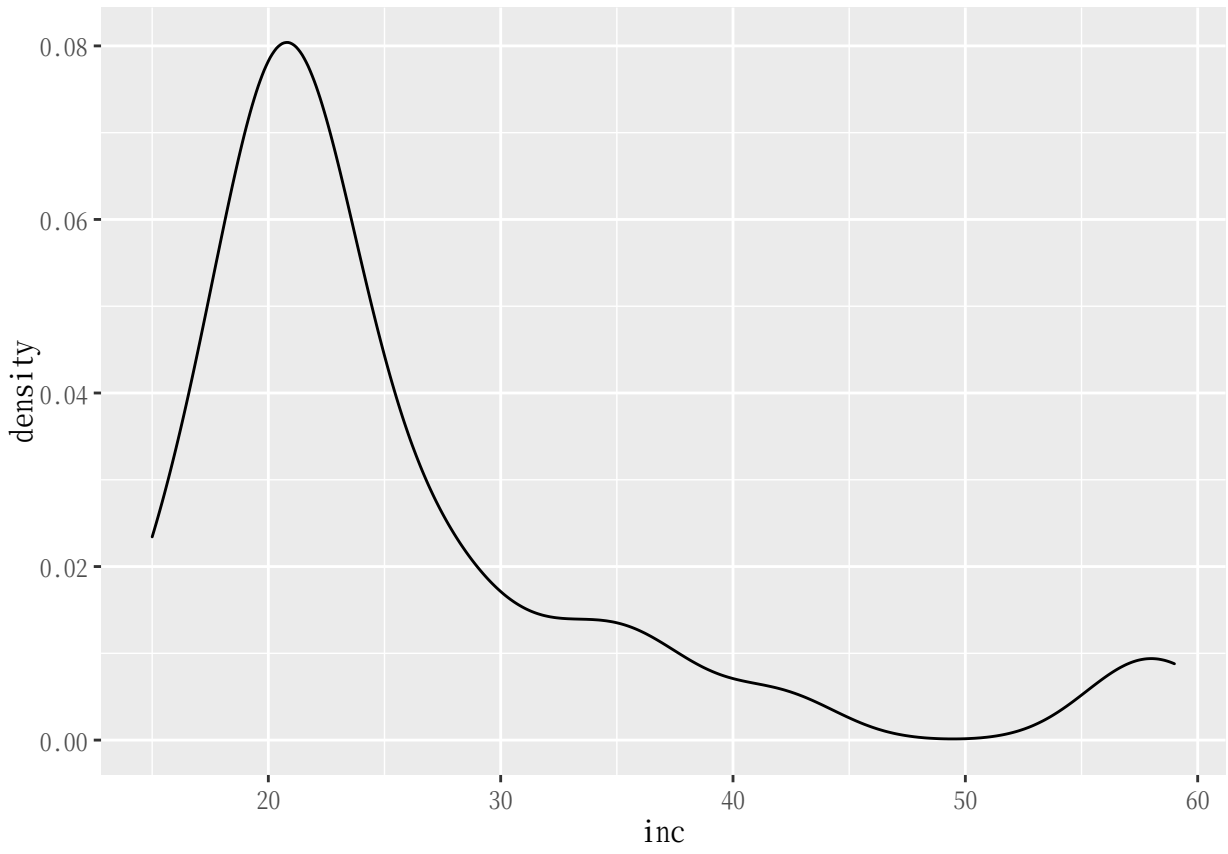
作各省人均可支配收入的直方图以及密度估计图：

```
p <- ggplot(dpop, aes(
  x = inc))
p + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

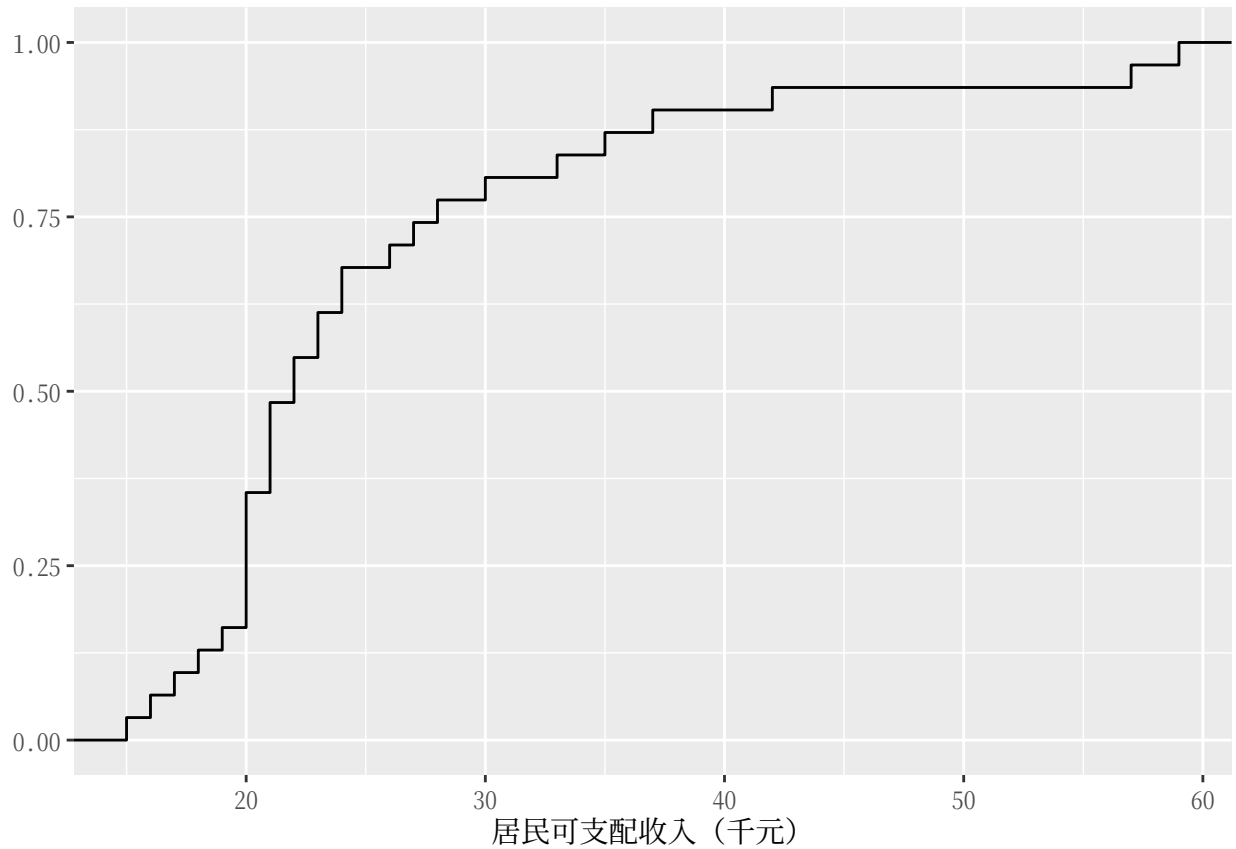



```
p + geom_density()
```



这样的分布称为右偏分布，特点是大部分数据取值集中在左侧，但是有不可忽略的多个数据点出现在远离数据取值中心的右侧。作收入的经验分布函数图：

```
ggplot(dpop, aes(
  x = inc, y = ..y..)) +
  stat_ecdf(geom = "step") +
  labs(x = " 居民可支配收入（千元）", y = NULL)
```



如果将数据做适当的变换，比如对数变换，分布形状可能变得比较正态。

28.3.4 QQ 图

对连续型随机变量 X，设其分布函数为 $F(x)$，其反函数 $F^{-1}(p), p \in (0,1)$ 称为 X 的分位数函数 (quantile function)。为了估计 $F^{-1}(p)$，可以将观测值 $X_1, X_2, \ldots, X_n$ 从小到大排序为 $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$，令 $X_{(i)}$ 作为 $F^{-1}(\frac{i}{n})$ 的估计，或者做连续性修正，作为 $F^{-1}(\frac{i-0.375}{n+0.25})$ 的估计。

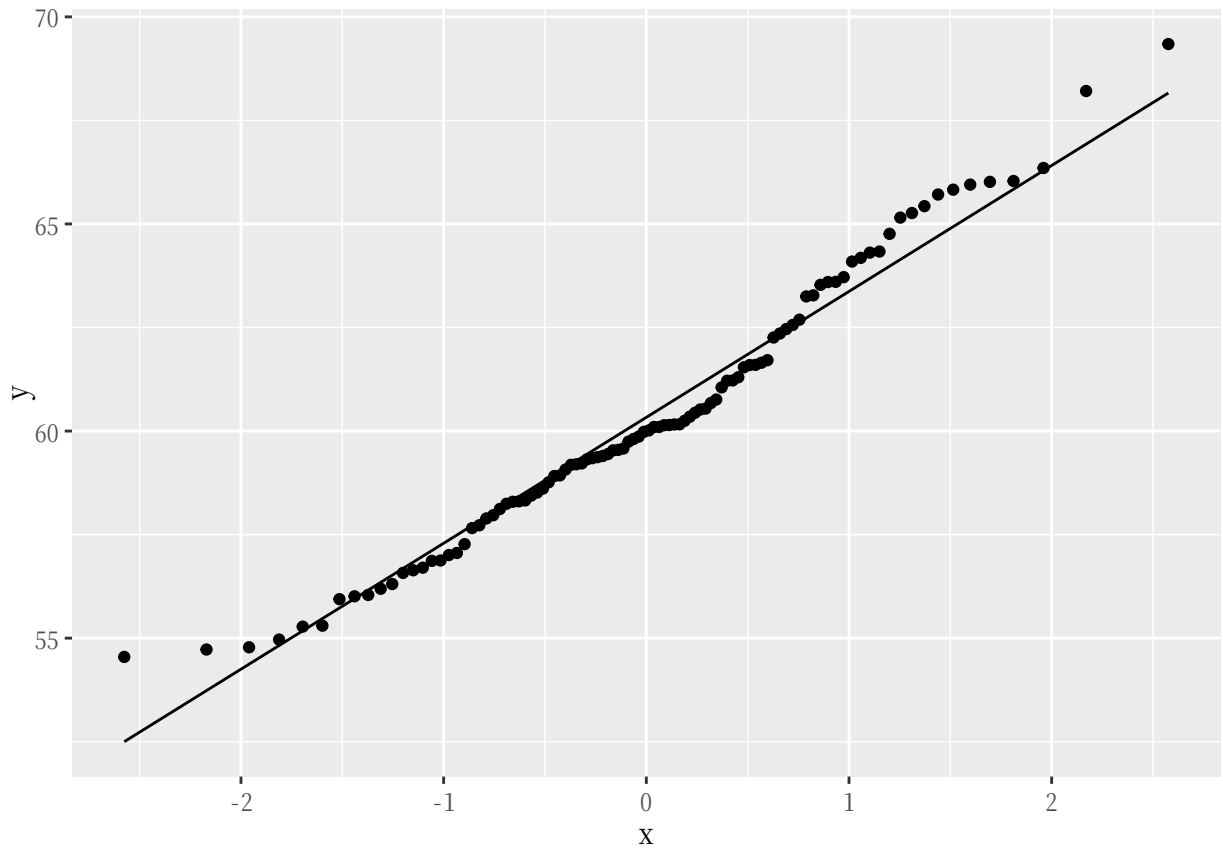
为了对比总体 X 的观测数据与某个理论分布比如正态分布之间的差距，可以用上述的样本分位数为散点纵坐标，以相应的理论分位数为横坐标作散点图，称为 QQ 图。如果总体服从所对比的分布，散点应该在一条直线附近，否则就会有明显的偏离。

最常见的 QQ 图是正态 QQ 图。下面对一些模拟数据做正态 QQ 图。

首先是模拟的正态分布数据:

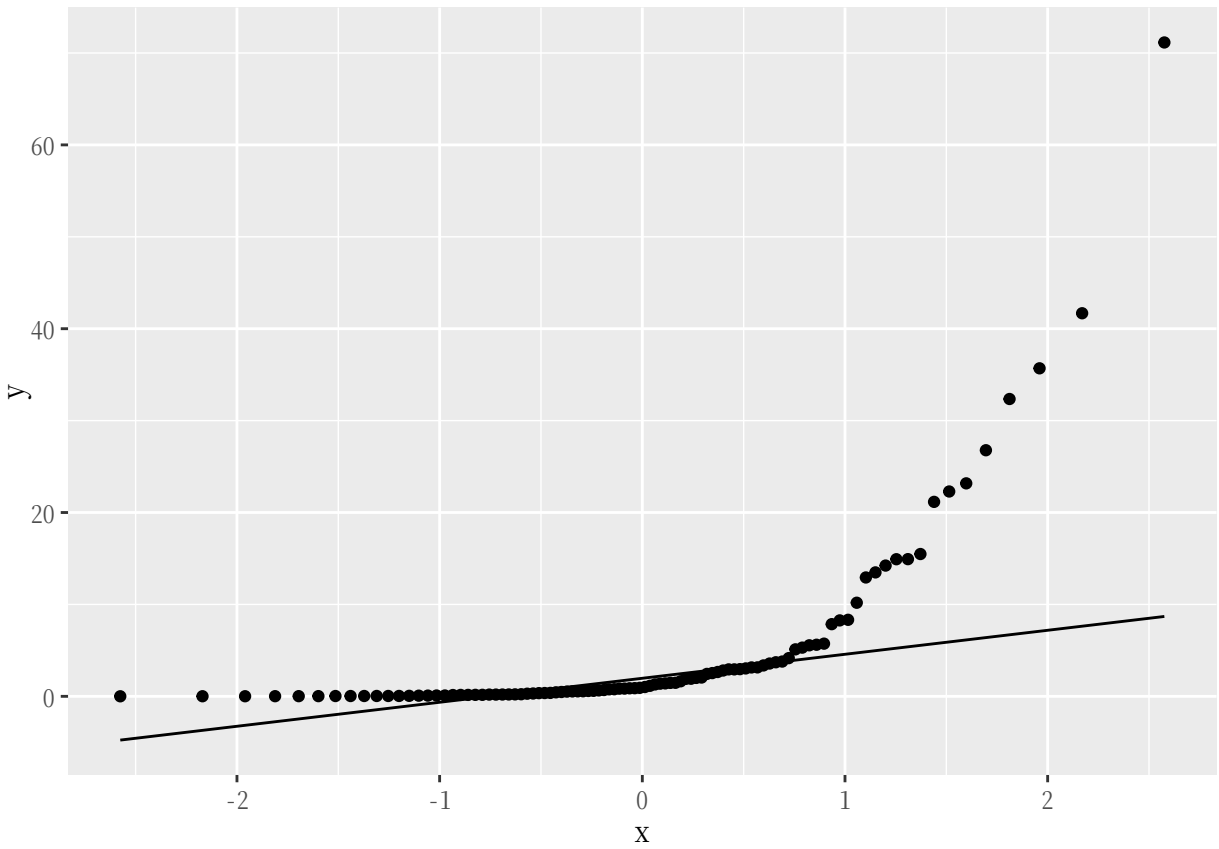
```
set.seed(102)
d <- tibble(x = 60 + rnorm(100, sd = 3))
ggplot(d, aes(
  sample = x )) +
```

```
  geom_qq() +
  geom_qq_line()
```



模拟的对数正态数据：

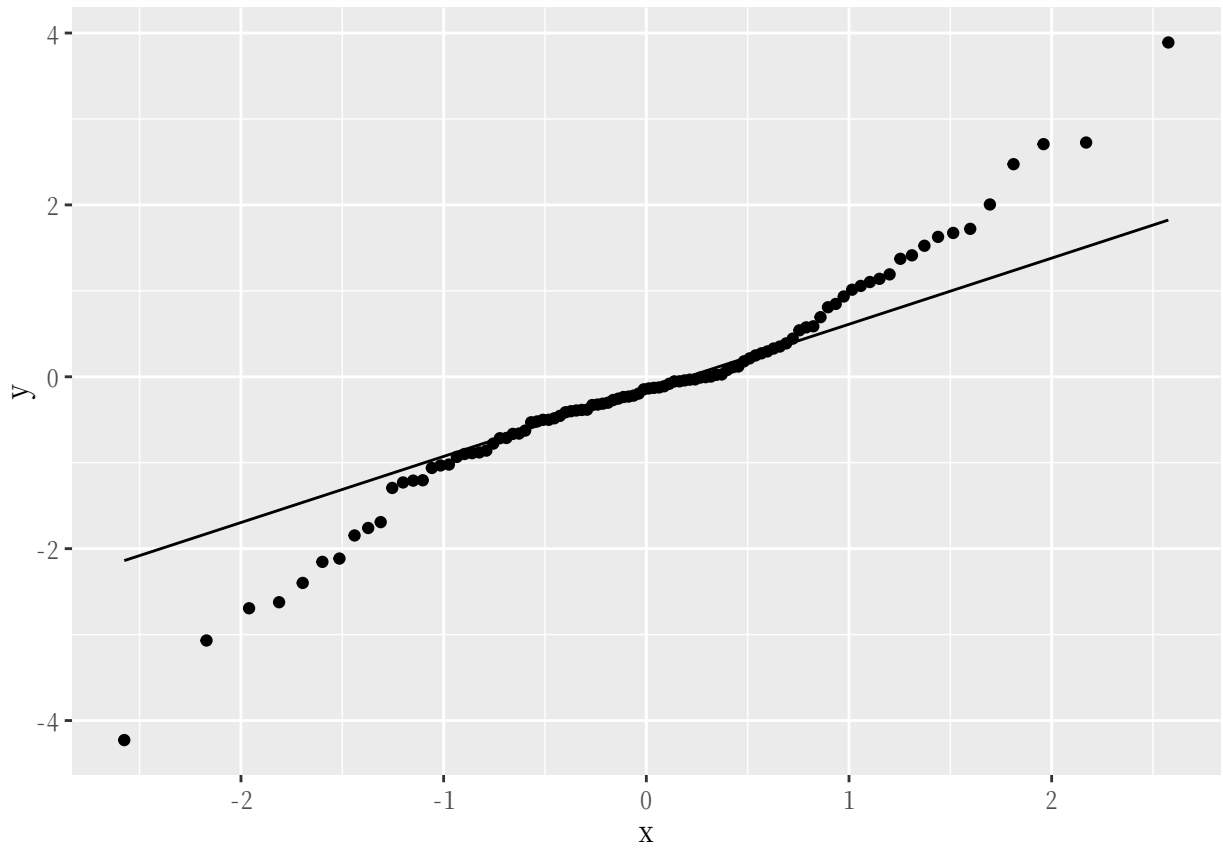
```
set.seed(101)
d <- tibble(x = 10^(rnorm(100, sd = 1)))
ggplot(d, aes(
  sample = x )) +
  geom_qq() +
  geom_qq_line()
```



上图为典型的右偏分布正态 QQ 图形状。

作 t(3) 分布样本的正态 QQ 图：

```
set.seed(104)
d <- tibble(x = rt(100, df=3))
ggplot(d, aes(
  sample = x )) +
  geom_qq() +
  geom_qq_line()
```



上图为典型的两侧都厚尾的分布的正态 QQ 图形状。

28.3.5 盒形图

如果需要同时表现多个分布，上面给出了同时绘制多条密度曲线的方法，但是这主要比较分布形状，而不能比较分布的主要数字特征。盒形图、小提琴图、脊线图 (ridgeline plot) 比较适合同时表现和比较多个分布。

比较多个分布的需求，可能是比较多个类似的变量，比如，同一个学校的不同科目的成绩；也可以是比较不同组的同一属性，比如，比较不同省的城镇居民收入。

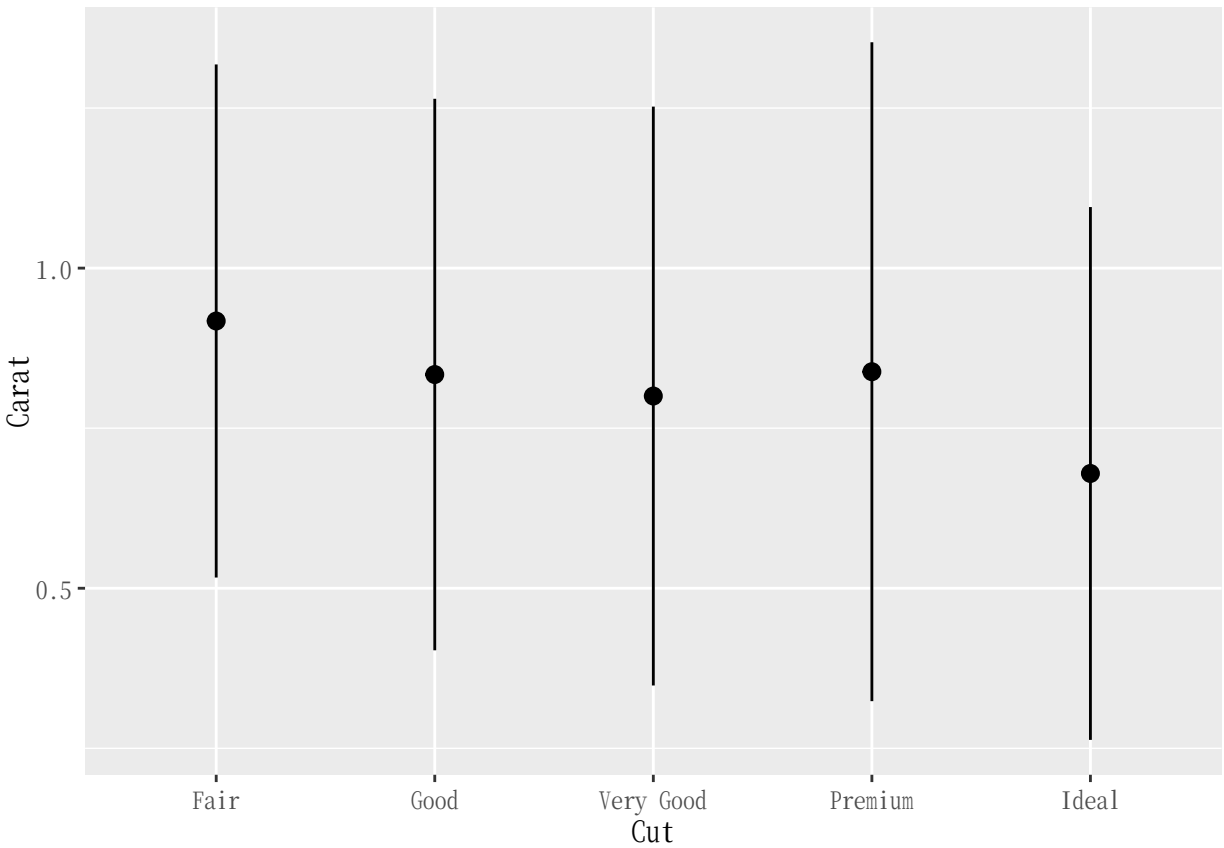
如果想仅表现每个分布的均值和标准差，可以对每个均值加减标准差画一个小的条形。比如，ggplot2 扩展包的 diamonds 数据集包含了 5 万多枚钻石的分级、大小、价格数据，我们从中抽取不含缺失数据的 1000 个观测，按 5 种不同品质（Fair, Good, Very Good, Premium, Ideal）分类，对克拉重量计算每类的均值和标准差：

```
set.seed(101)
diamonds |>
  na.omit() |>
  sample_n(1000) -> ddsmall
ddsmall |>
  group_by(cut) |>
  summarise(
```

```
    mean_carat = mean(carat),
    sd_carat = sd(carat)) -> ddsmall_summ
```

对每个类画以均值为中心、以均值加减标准差为两段的条形：

```
ggplot(ddsmall_summ, aes(
  x = cut,
  y = mean_carat )) +
  geom_point(size = 1.2) +
  geom_pointrange(mapping = aes(
    ymin = mean_carat - sd_carat,
    ymax = mean_carat + sd_carat)) +
  labs(
    x = "Cut",
    y = "Carat")
```

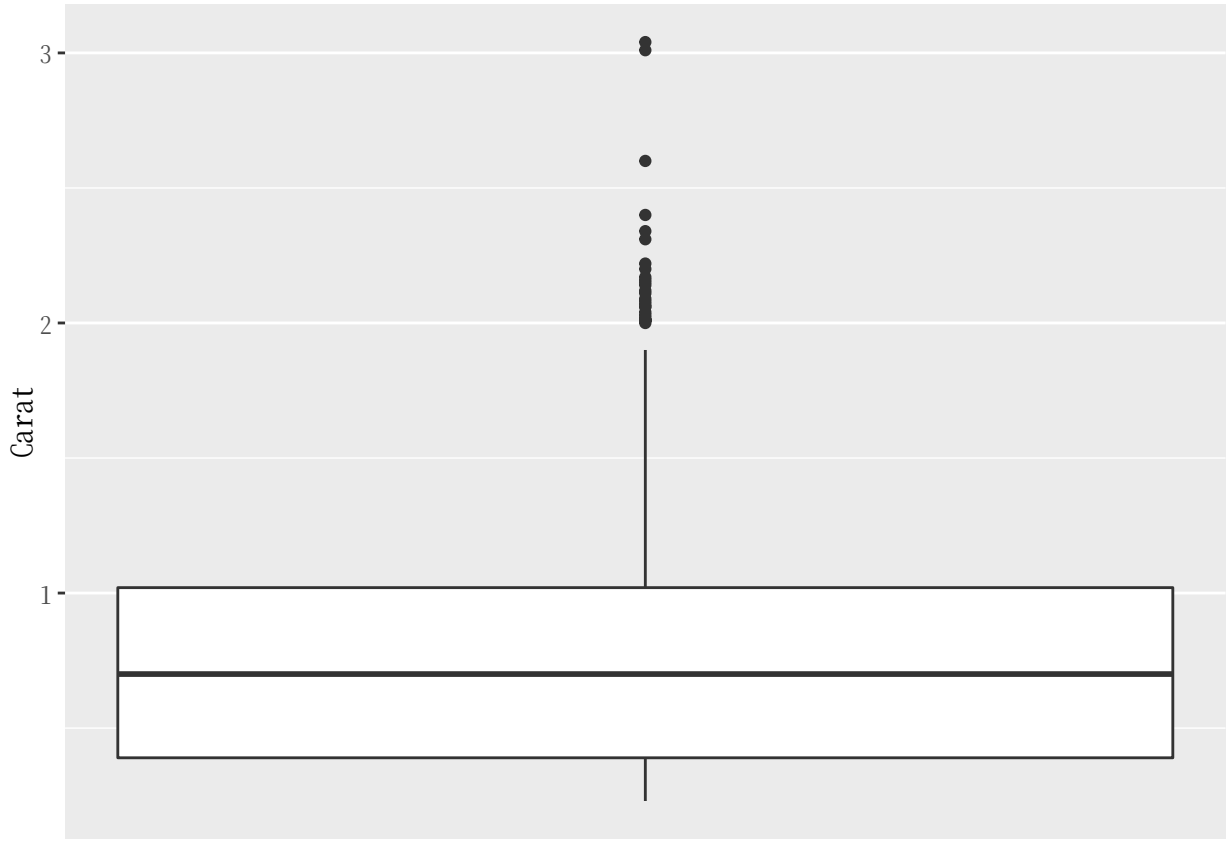


竖线的上下界也可以改为正态情形下均值的近似 95% 置信限，即 $\bar{x} \pm 1.96S/\sqrt{n}$。可以用 `coord_flip()` 颠倒 x 轴和 y 轴。

这种图形对分布信息压缩过甚，对于非正态的分布则不能表现分布的形状。盒形图可以表现分布的中位数、四分之

一和四分之三分位数，最小值、最大值和可能的离群值。`geom_boxplot()` 作盒形图。比如，diamonds 数据集抽样 1000 个观测的克拉重量的盒形图：

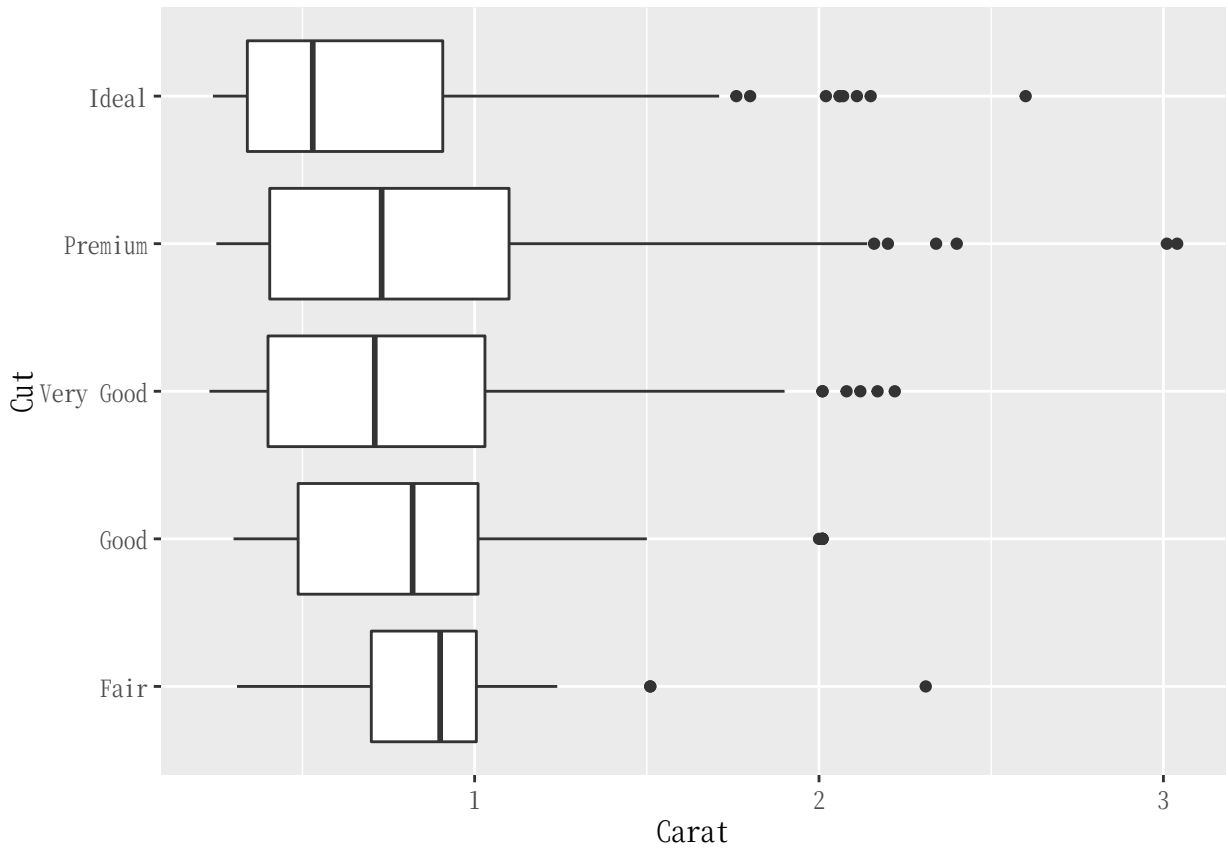
```
ggplot(ddsmall, aes(
  y = carat )) +
  geom_boxplot() +
  scale_x_continuous(breaks = NULL) +
  labs(x = NULL,
       y = "Carat")
```



从这个盒形图可以看出克拉重量呈现右偏分布。

比较不同品质的钻石的克拉重量，并将盒形图横向画出：

```
ggplot(ddsmall, aes(
  x = cut,
  y = carat )) +
  geom_boxplot() +
  labs(x = "Cut",
       y = "Carat") +
  coord_flip()
```



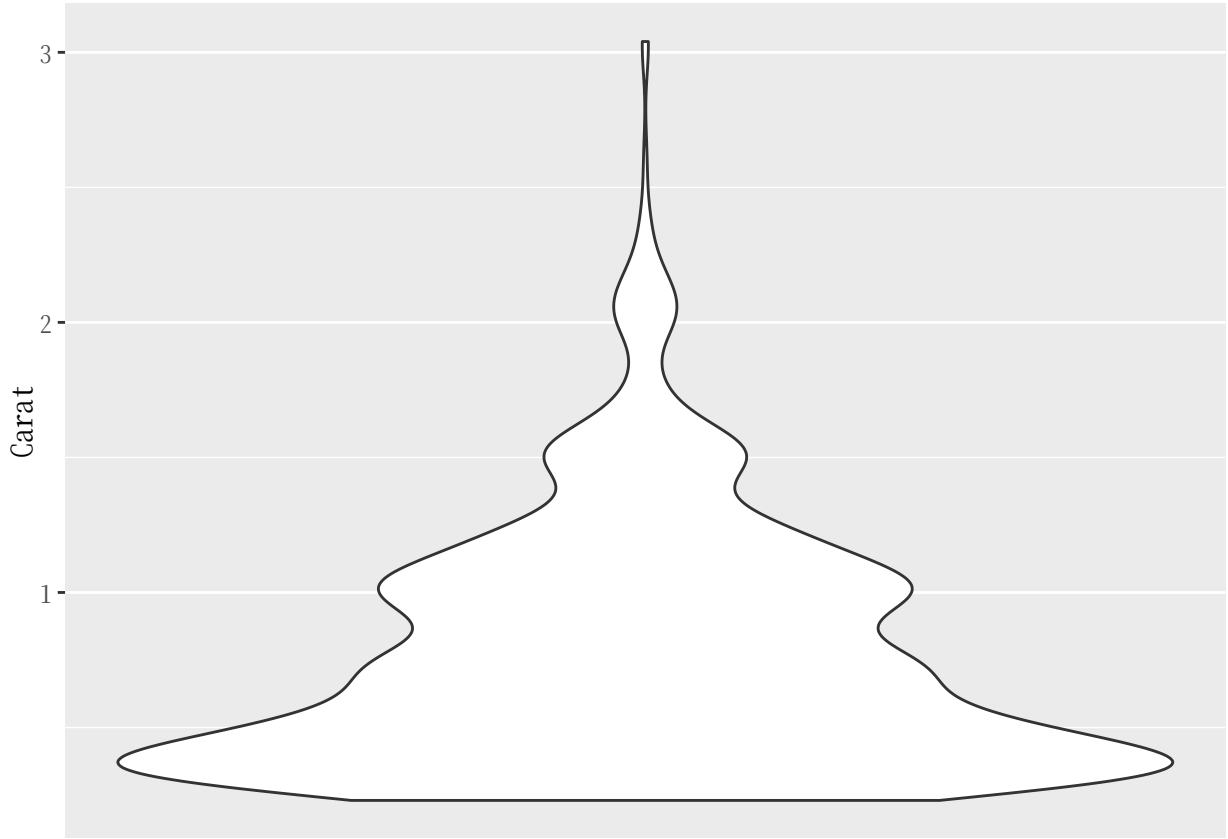
从克拉重量中位数看，品质从 Fair 到 Ideal 基本上是逐步下降的，只有 Very Good 级和 Premium 级的中位重量略有乱序。除了 Fair 和 Good 级，更好品质的三个等级的重量分布都呈右偏，而 Fair 和 Good 级的分布则相对来说不偏态。

盒形图的轮廓线与盒子内部填充色可以用 `color` 和 `fill` 参数指定。

28.3.6 小提琴图

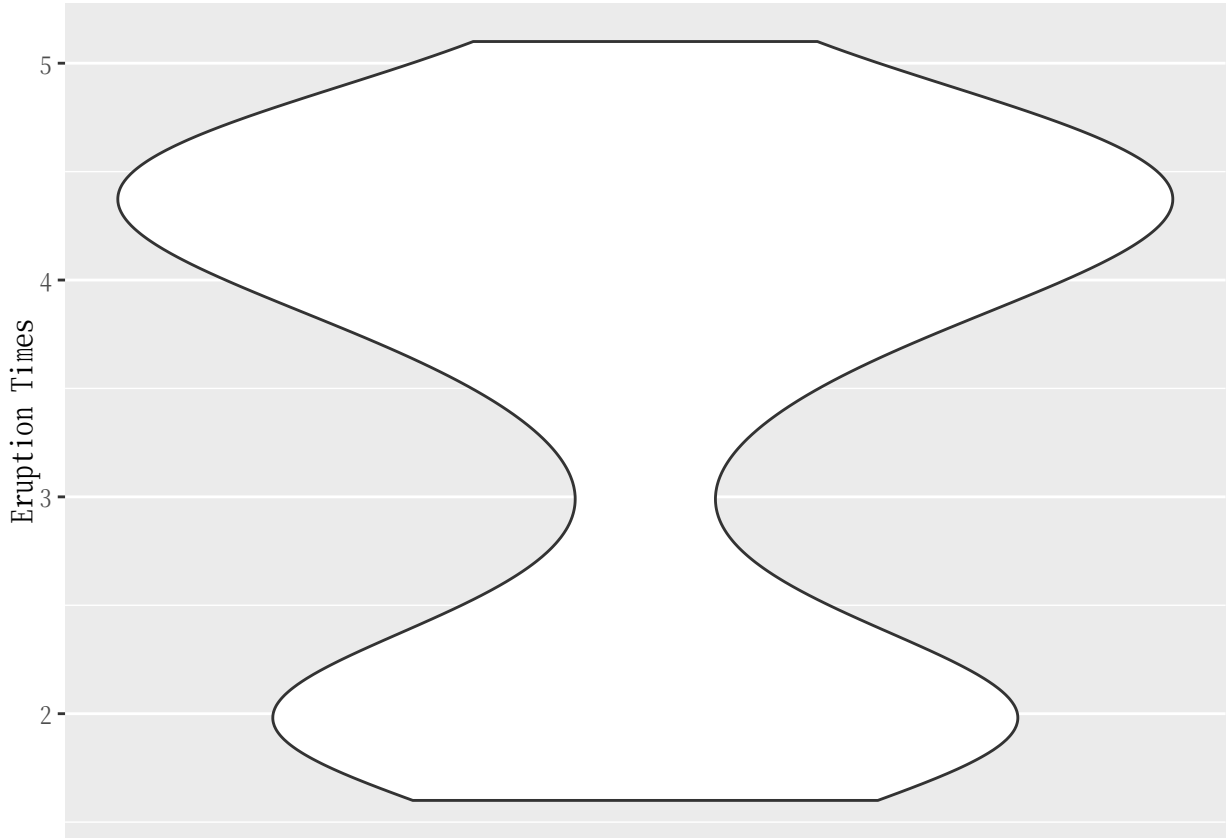
小提琴图是盒形图的一个改进版本，比如，钻石数据抽样数据集的克拉重量的小提琴图：

```
ggplot(ddsmall, aes(
  x = 0, y = carat )) +
  geom_violin() +
  scale_x_continuous(breaks = NULL) +
  labs(
    x = NULL,
    y = "Carat")
```



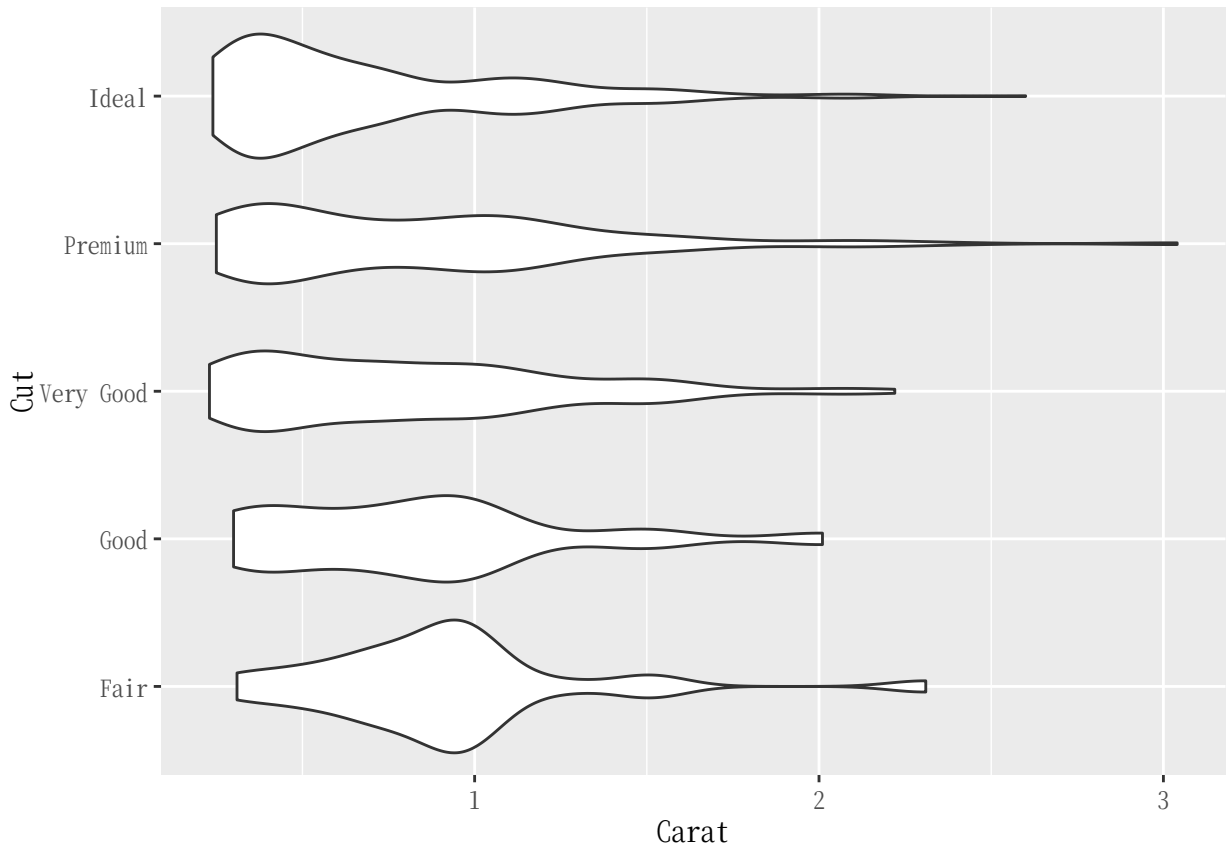
这实际是将密度估计曲线旋转 90 度画出，并左右镜像画两条。它提供了比盒形图更详细的分布信息，比如，可以表现出双峰分布。下面是 R 的 faithful 数据集中变量 eruptions 的小提琴图：

```
data(faithful)
ggplot(faithful, aes(
  x = 0, y = eruptions )) +
  geom_violin() +
  scale_x_continuous(breaks = NULL) +
  labs(
    x = NULL,
    y = "Eruption Times")
```



钻石数据小数据集中各品质级别的钻石克拉重量的小提琴图比较，横向放置：

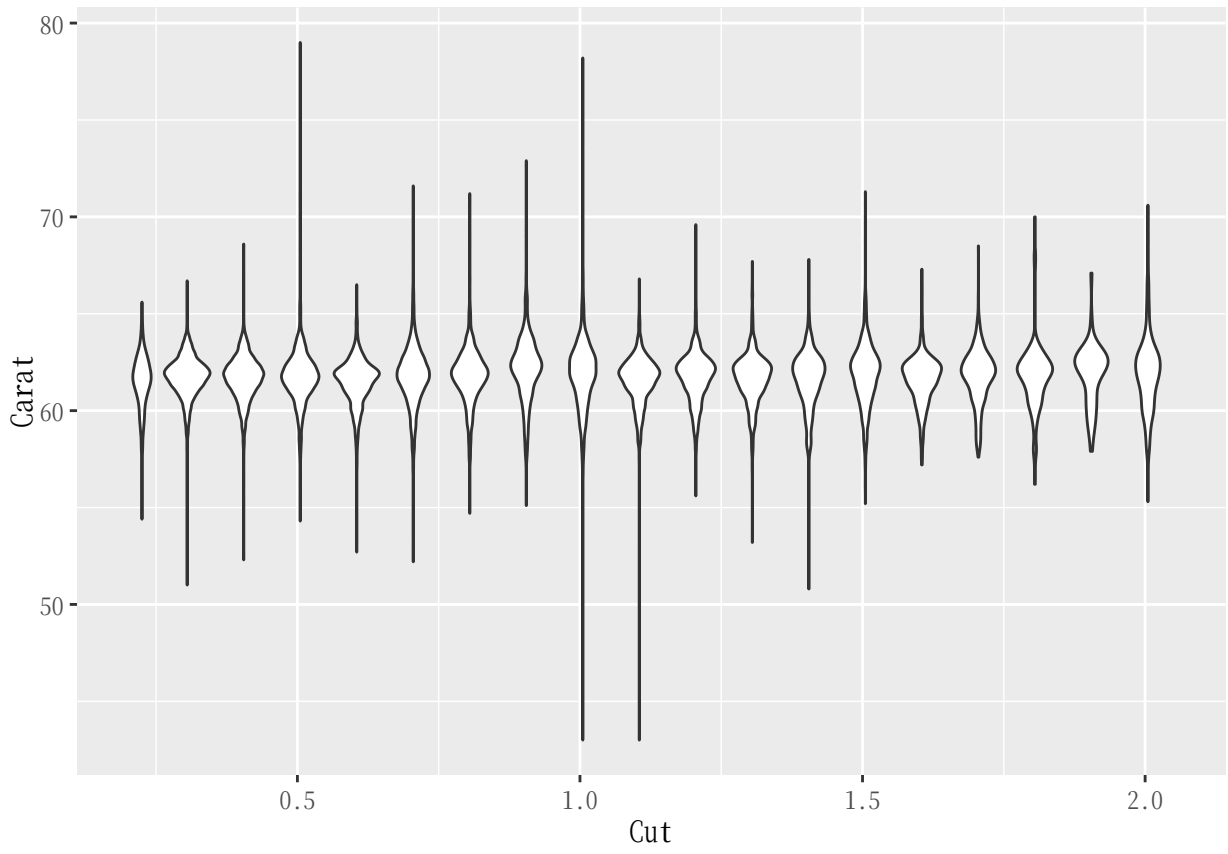
```
ggplot(ddsmall, aes(
  x = cut,
  y = carat )) +
  geom_violin() +
  labs(x = "Cut",
       y = "Carat") +
  coord_flip()
```



对连续变量，可以用 ggplot2 包的 `cut_width()` 函数分组，然后可以分组作小提琴图或者盒形图。例如，按照钻石数据的 `carat` 变量分组，对 `depth` 变量作小提琴图：

```
ggplot(diamonds, aes(
  x = carat,
  y = depth,
  group = cut_width(carat, 0.1) )) +
  geom_violin() +
  xlim(NA, 2.05) +
  labs(
    x = "Cut",
    y = "Carat")
```

```
## Warning: Removed 997 rows containing non-finite values (stat_ydensity).
```

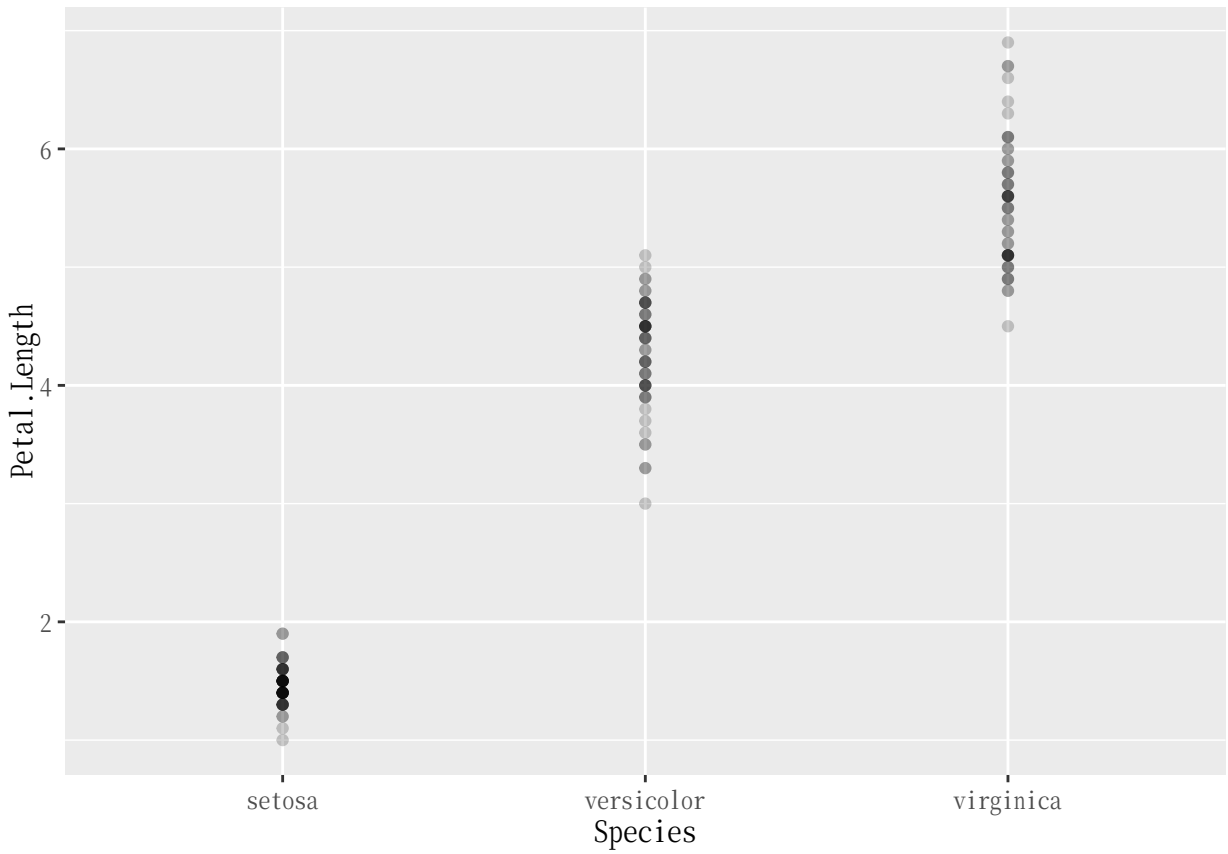


其中 `xlim()` 函数用来指定 x 轴的左右边界，`NA` 表示用数据的值确定。

28.3.7 蜂群图

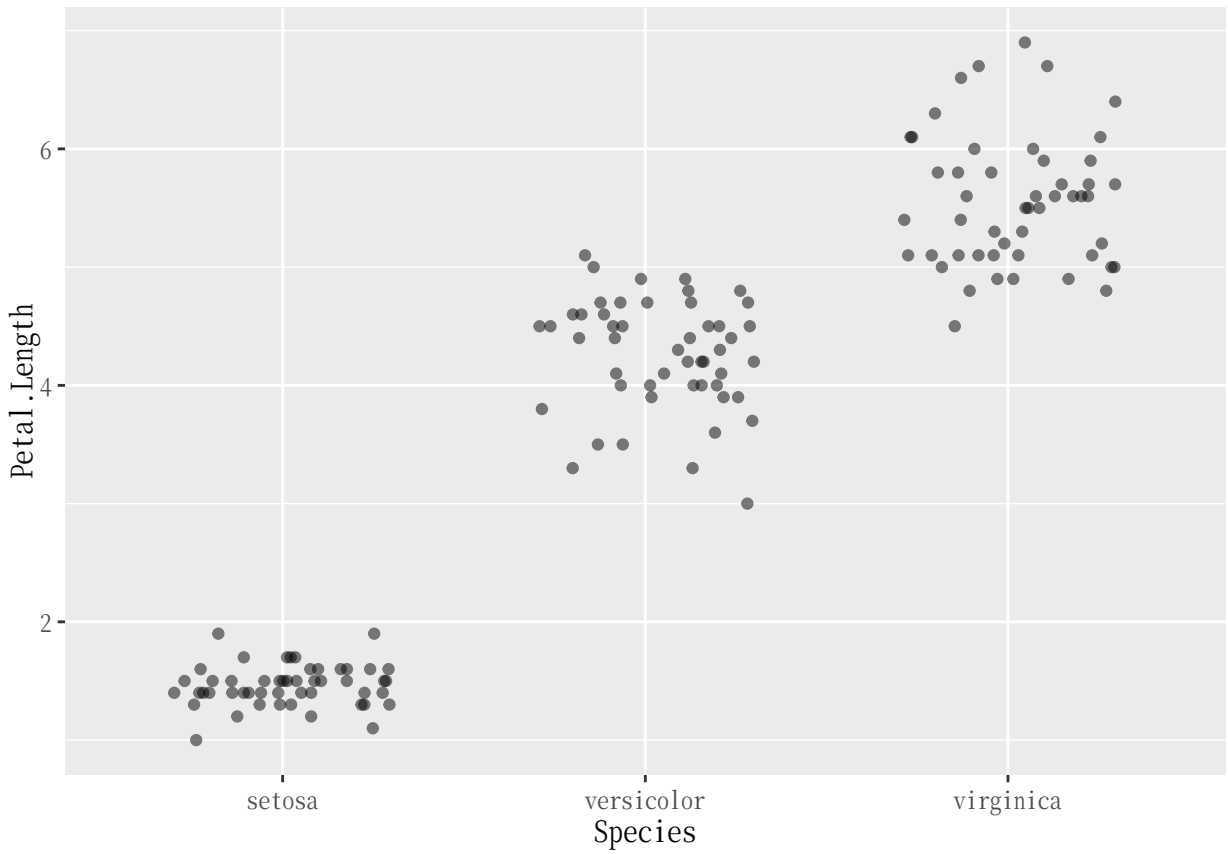
在比较各组的分布时，如果每组观测个数不太多，可以将每个观测值用一个点表示。例如，对 iris 数据，可以将三个品种的花瓣长度变量分别画出散点：

```
data(iris)
p <- ggplot(iris, aes(
  x = Species, y = Petal.Length ))
p + geom_point(alpha = 0.2)
```



这样有许多点重叠，为此使用抖动方法：

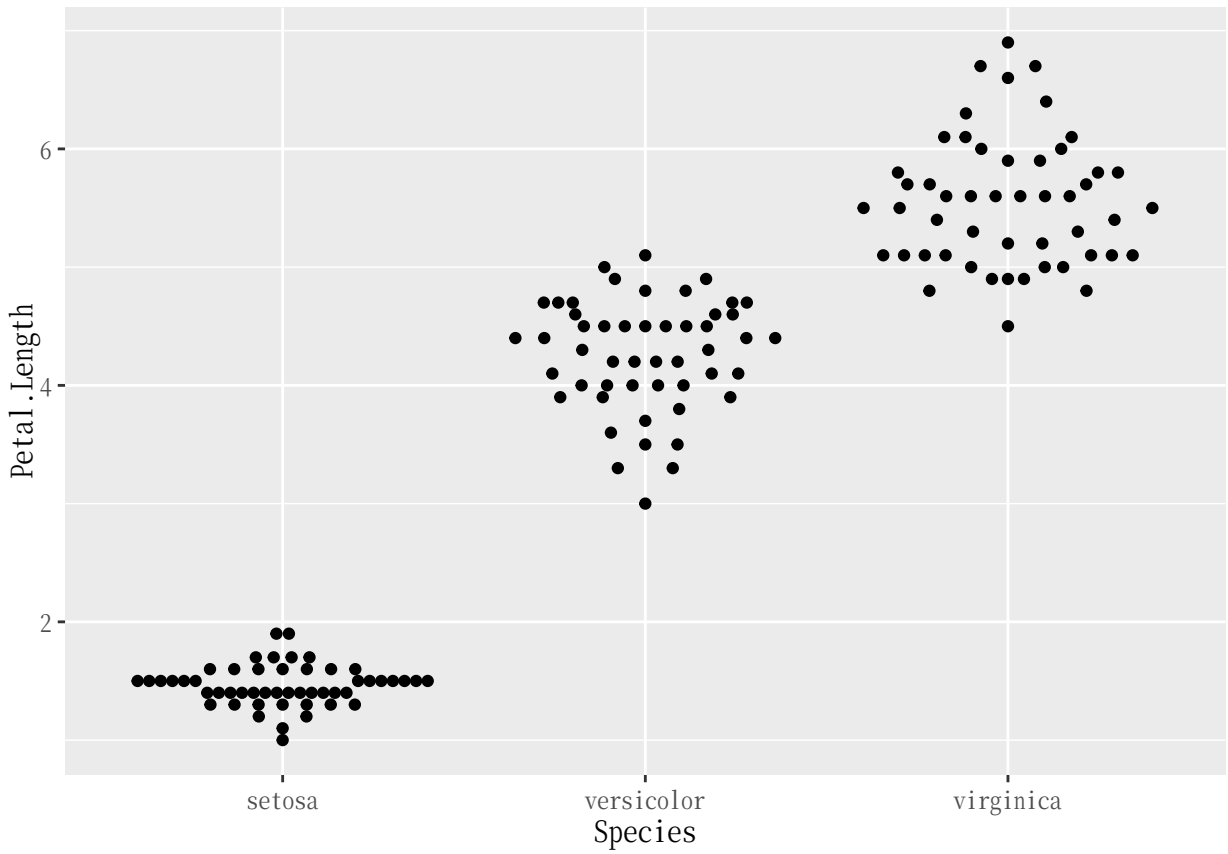
```
p + geom_jitter(width = 0.3, height = 0, alpha = 0.5)
```



这样可以较好地展示每组的具体数值。因为抖动的做法使用了真实随机抖动，所以其效果是可变的而且不保证没有重叠。

ggbeeswarm 包提供了 `geom_quasirandom()`，可以使得抖动结果皆可能不重叠：

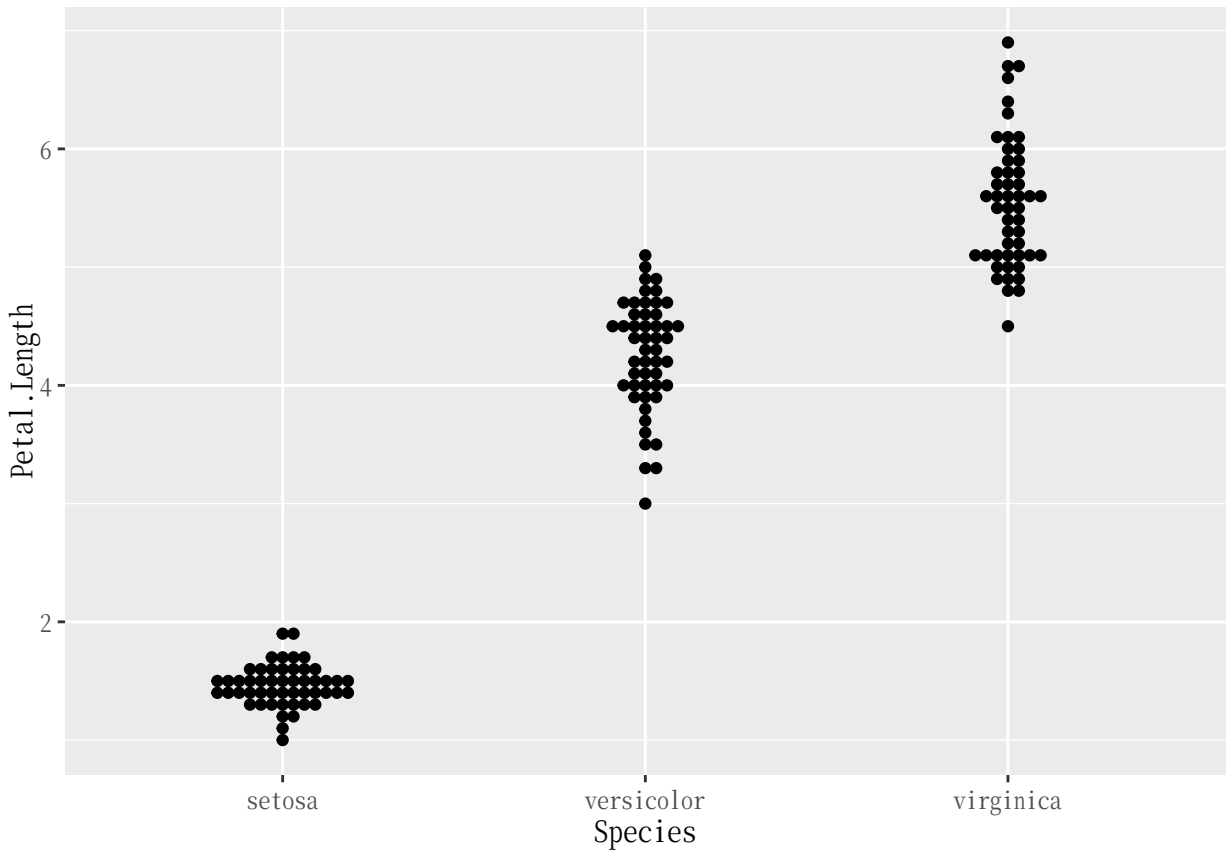
```
library(ggbeeswarm)
p + geom_quasirandom(method="smiley")
```



`geom_quasirandom()` 的 `method` 选项可以取一些画法的变种，如 quasirandom, pseudorandom, smiley, frowney, tukey, tukeyDense，缺省值为如 quasirandom。

beeswarm 图则给出了一种类似的做法：

```
p + geom_beeswarm()
```



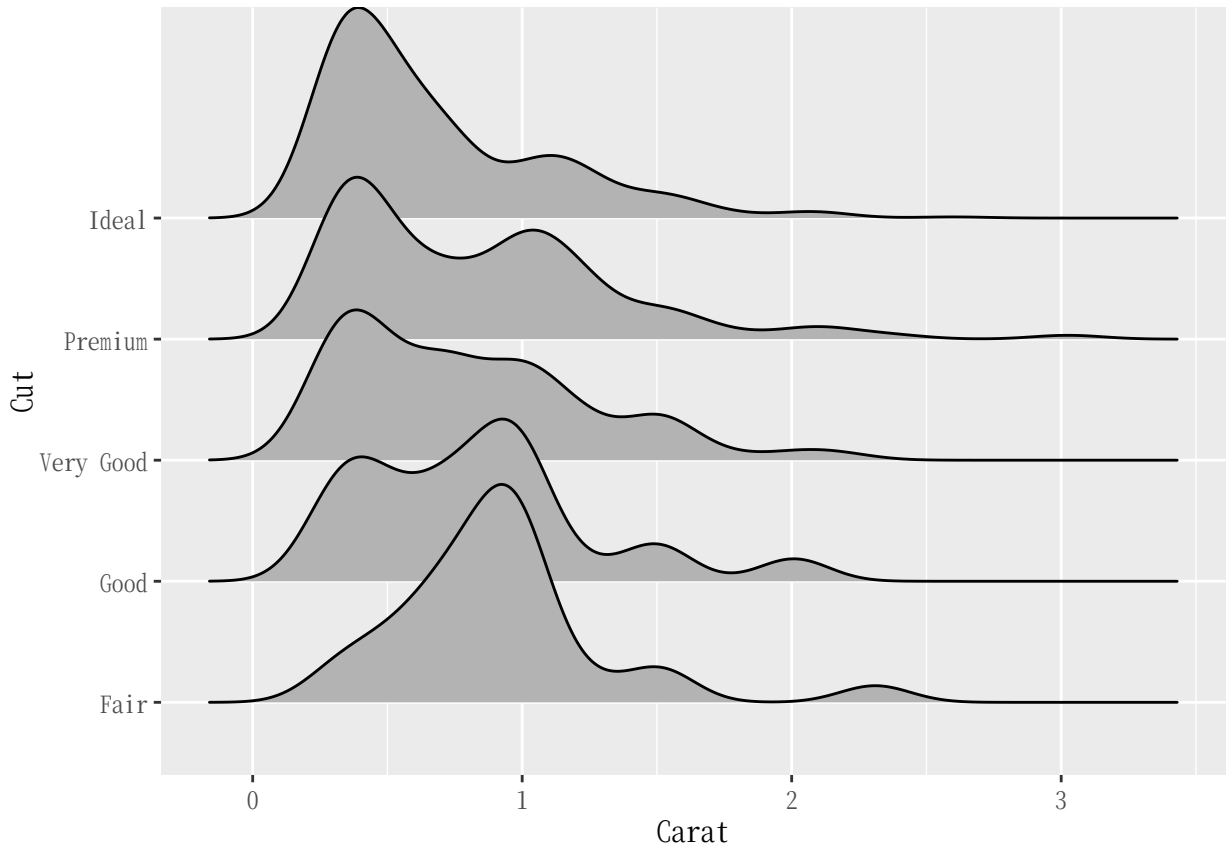
`geom_beeswarm()` 的 `method` 选项可以取 swarm, compactswarm, hex, square, center 等画法，缺省值为 swarm。可以用 `size` 调整点的大小。

28.3.8 多个密度的其它画法

28.3.2给出了将多个密度曲线叠加画出的方法。在对多个组分别画某个连续变量的密度估计时，可以将组别放置在 y 轴上，仍用 x 轴表示该连续变量的取值范围。这种图形尤其适用于考察随着时间或者地点变化的变量分布。ggridges 包提供了 `geom_density_ridges()` 函数作脊线图。对钻石小数据集的各个级别的克拉重量作分布密度脊线图：

```
library(ggridges)
ggplot(ddsmall, aes(
  x = carat,
  y = cut)) +
  geom_density_ridges() +
  labs(
    x = "Carat",
    y = "Cut")
```

```
## Picking joint bandwidth of 0.13
```

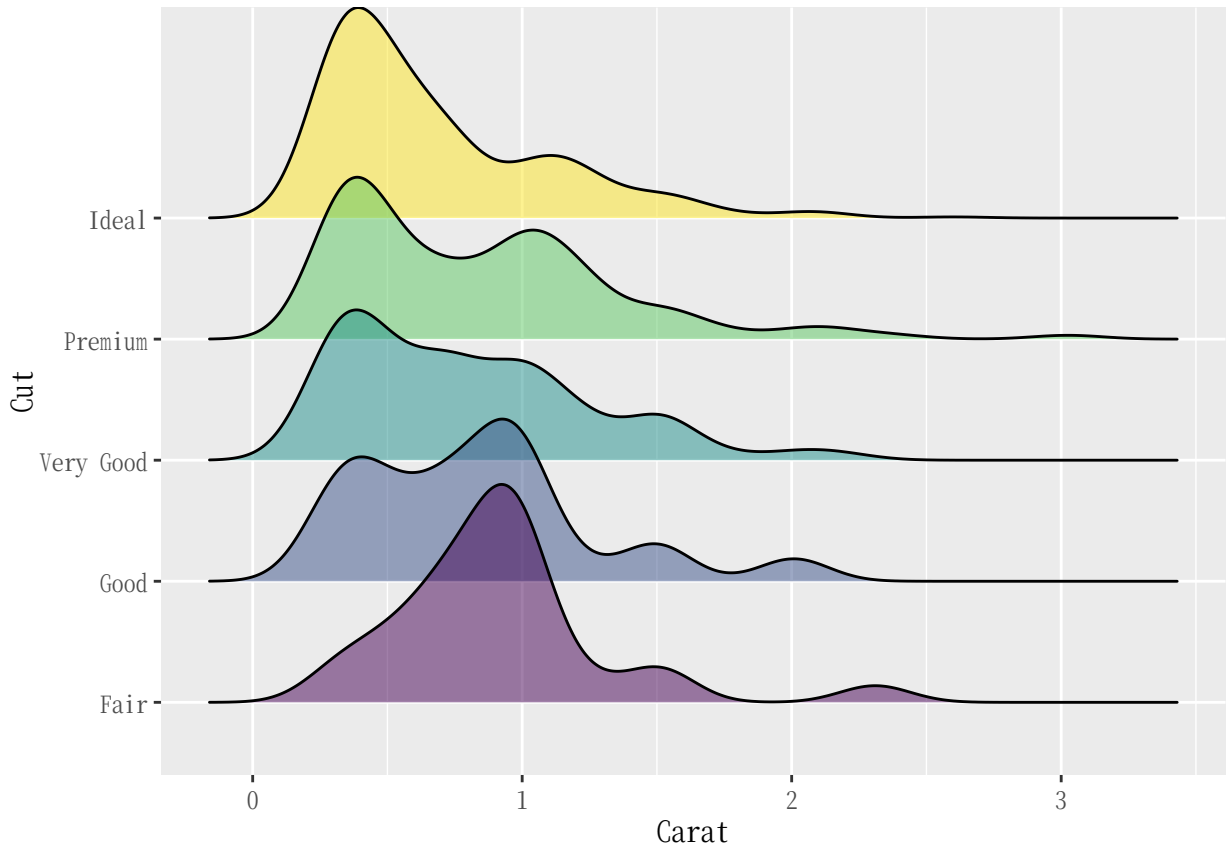


可以添加颜色和透明度：

```
ggplot(ddsmall, aes(
  x = carat,
  y = cut,
  fill = cut)) +
  geom_density_ridges(alpha = 0.5) +
  guides(fill = FALSE) +
  labs(
    x = "Carat",
    y = "Cut")
```

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```

```
## Picking joint bandwidth of 0.13
```



可以看出不同级别的钻石的重量密度峰值的变化，以及一定的多峰分布特征。

28.3.9 二元分布直方图和等值线图

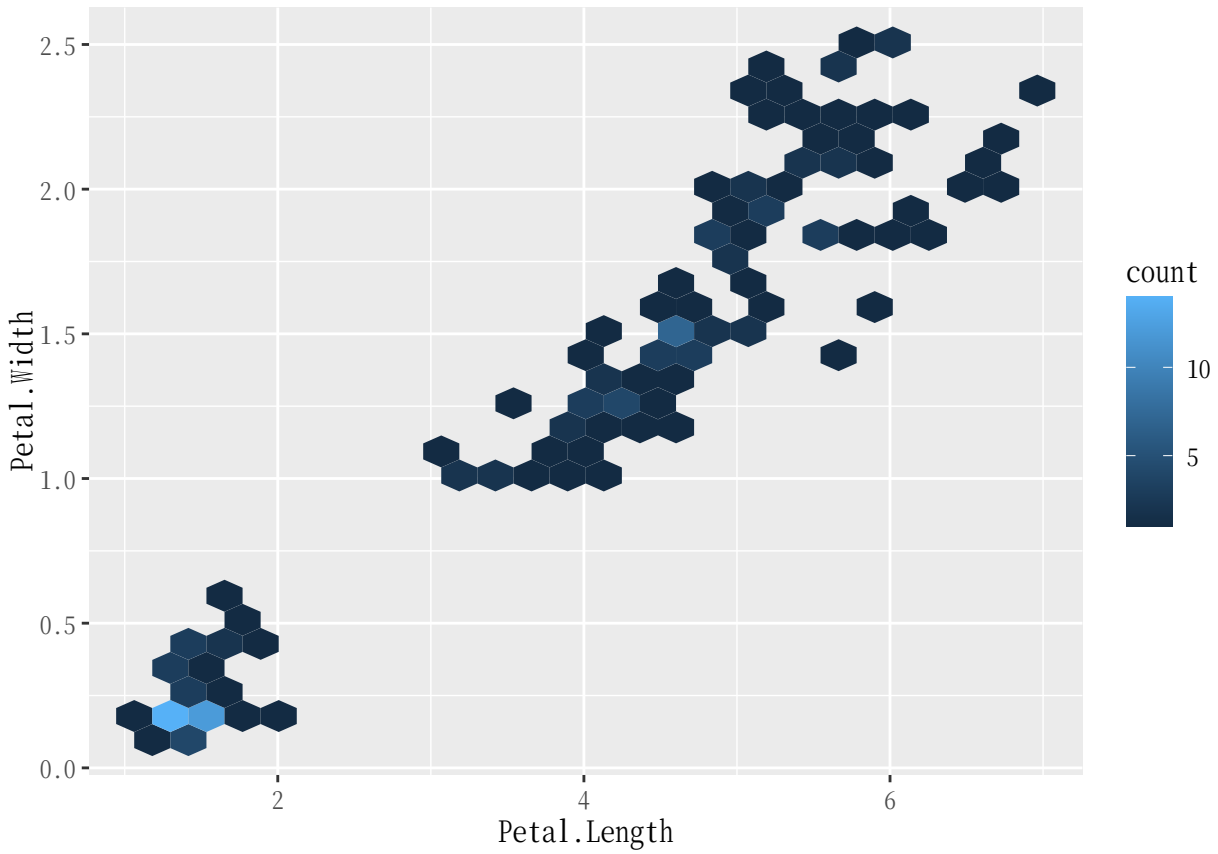
28.3.9.1 蜂窝图

对二元分布样本 (x_i, y_i)，可以用散点图表现其分布情况；当点比较多时，可以使用 `alpha` 参数设定一定的透明度，使得颜色深表示点比较密集，颜色浅表示点比较稀疏。但是，当点的数量很大时，设定透明度就效果不好了，这时稀疏的地方可能基本看不到颜色，稠密的地方就是一团黑，无法分别分布情况。

可以用类似直方图的想法将二元分布的取值区域分块，每块统计频数，然后用不同深浅的颜色为每个小块染色，颜色的深浅代表频数高低。这样的图形称为二维直方图。每个小块可以是正方形或者正六边形。

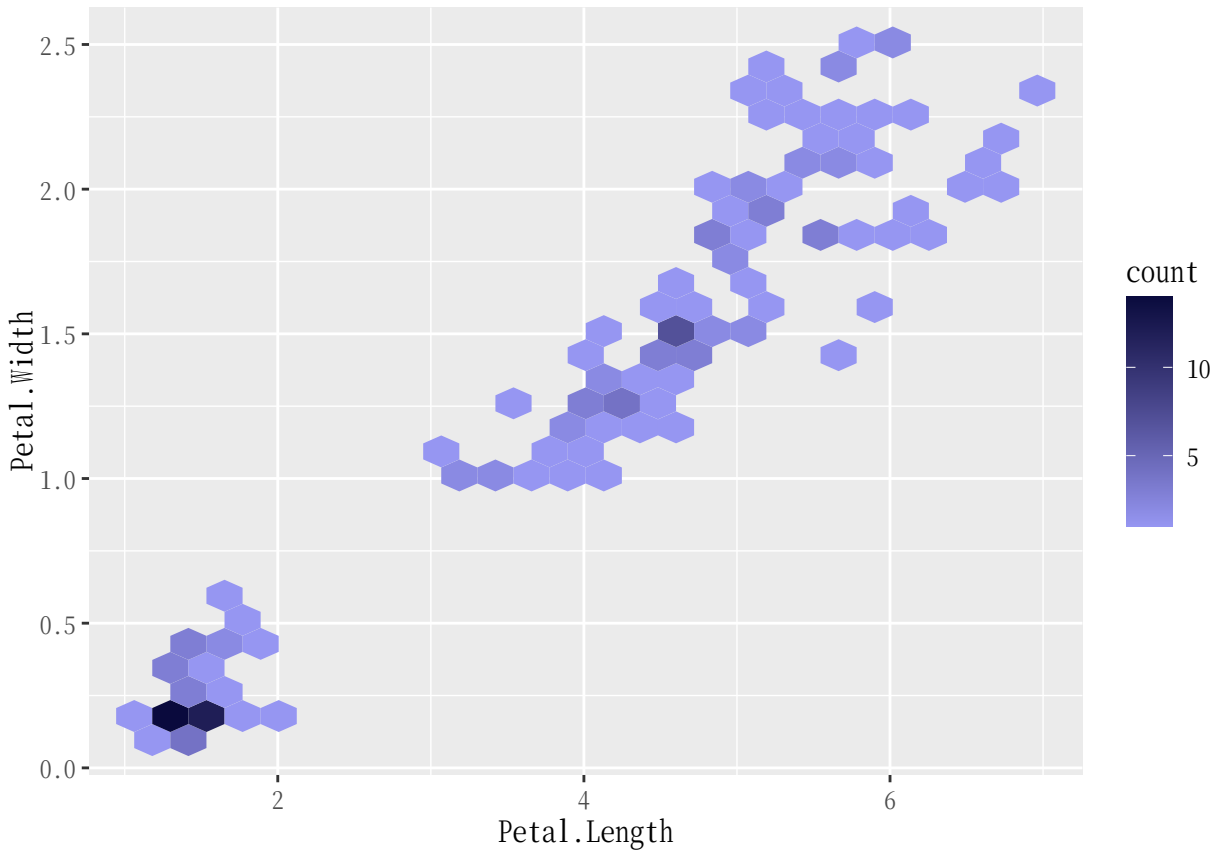
例如，对 iris 数据的花瓣长、宽变量画正六边形的二维直方图：

```
p <- ggplot(data = iris, mapping = aes(
  x = Petal.Length, y = Petal.Width))
p + geom_hex(bins = 25)
```



在 `geom_bins()` 中可以用 `bins=` 指定分块的个数。上图中颜色浅的值高,颜色浅的值低。可以用 `scale_color_gradient` 人为地指定代表频数的渐变色，可以改为用深色代表高值，如：

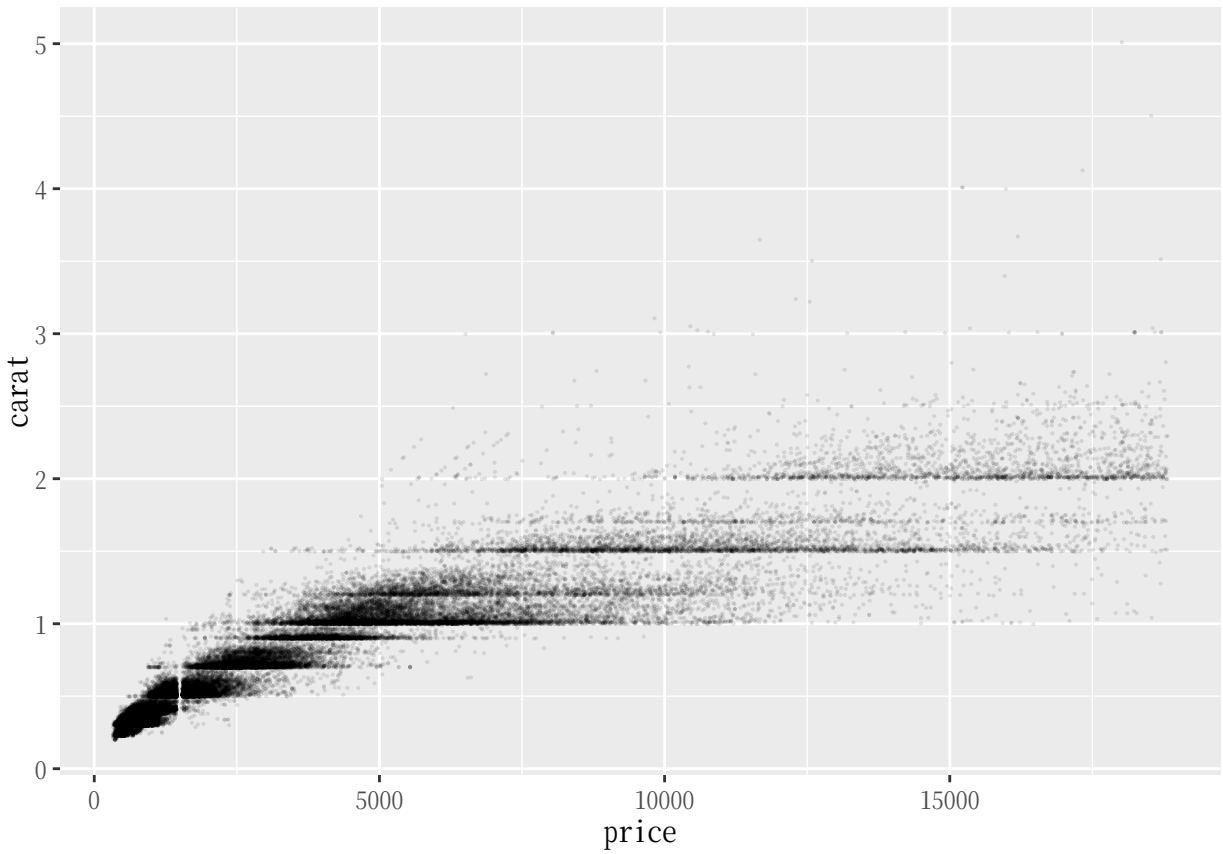
```
p + geom_hex(bins = 25) +
  scale_fill_gradient(
    low = "#9696F2",
    high = "#0A0A3D")
```



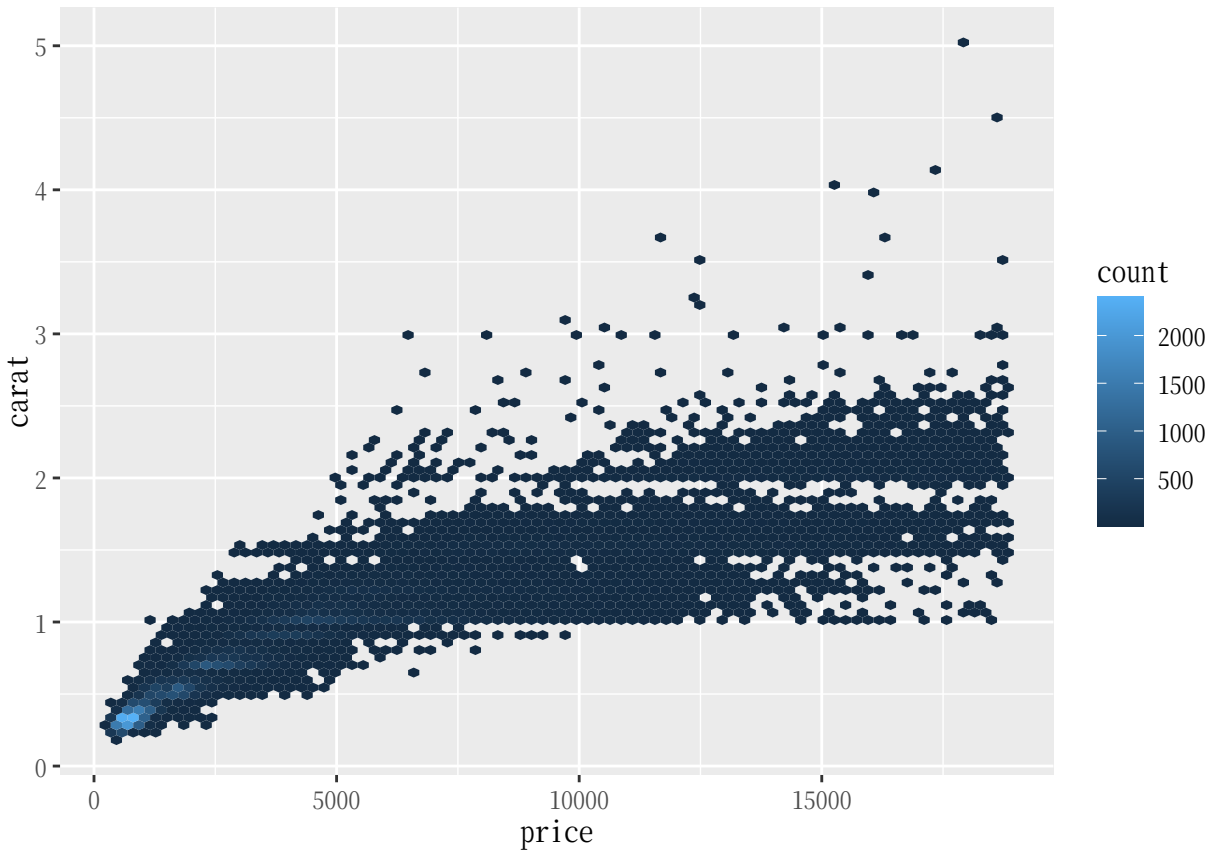
可以利用 colourpicker 包的 colourPicker 函数或者该包的 RStudio Addin – colourPicker 挑选颜色。

下面做 ggplot2 包的 diamonds 数据集中的 price 和 carat 的散点图和二维直方图：

```
p <- ggplot(data = diamonds, mapping = aes(
  x = price, y = carat))
p + geom_jitter(alpha = 0.1, size=0.05)
```



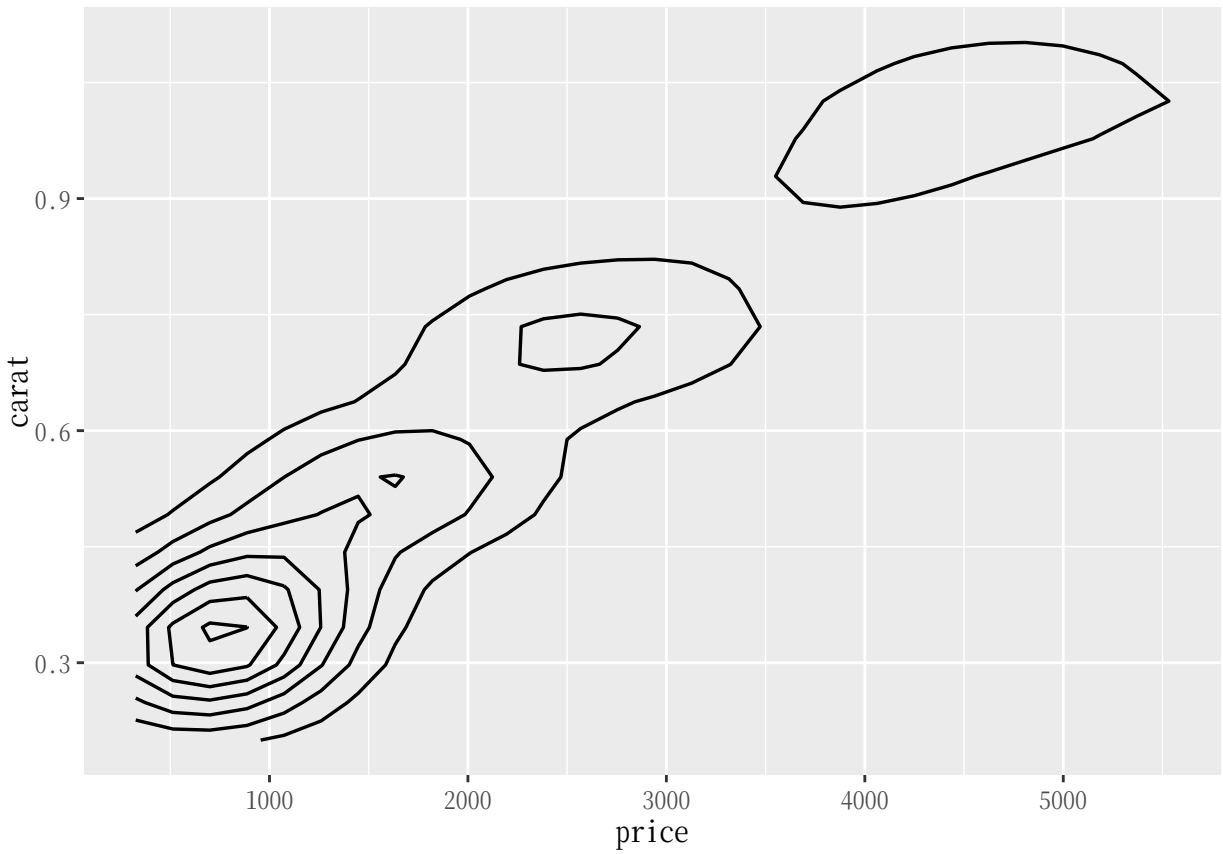

```
p + geom_hex(bins = 80)
```



28.3.9.2 二元密度估计图

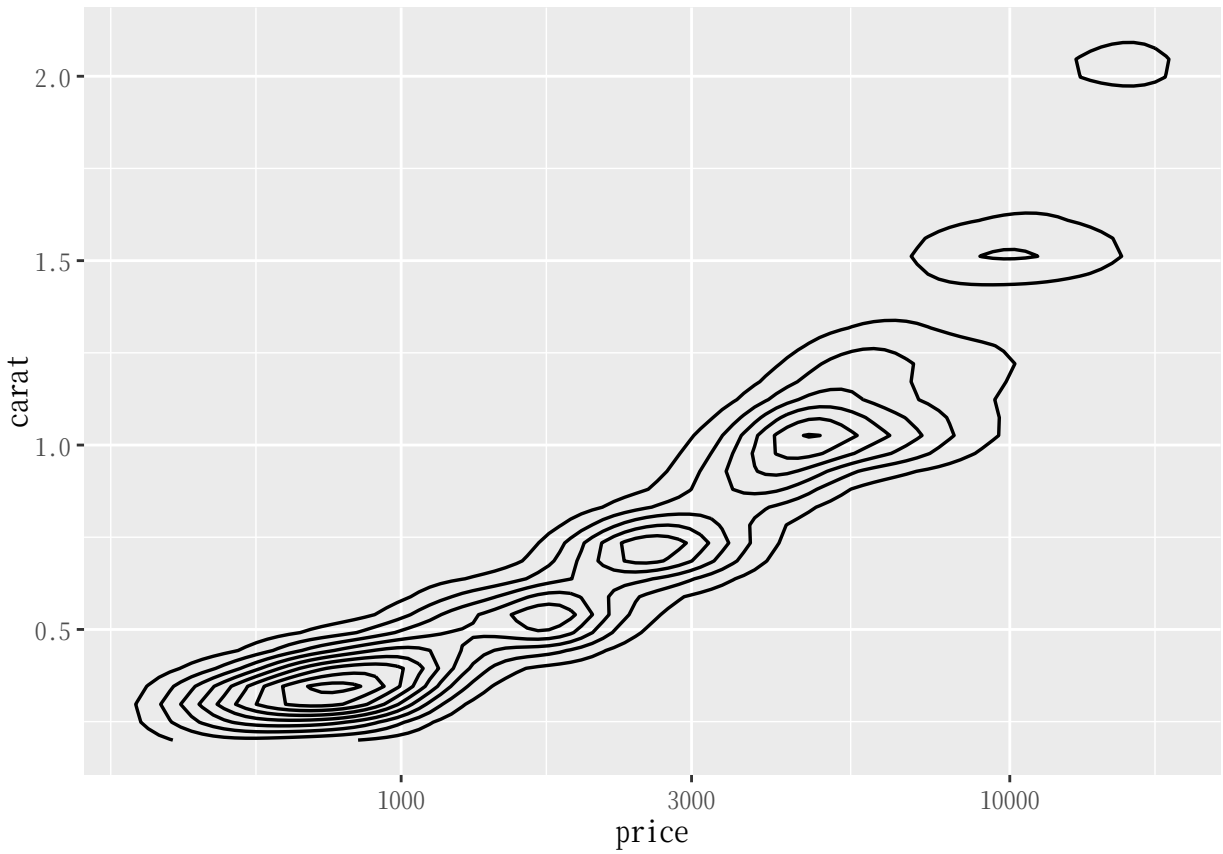
二维直方图估计的是二元阶梯函数，也可以像一元情况估计分布密度那样估计连续的二元分布密度，并用等值线图表示（类似于地图中的等高线）。例如 diamonds 数据集中价格和重量的联合密度估计：

```
p + stat_density_2d(color = "black", size = 0.6)
```



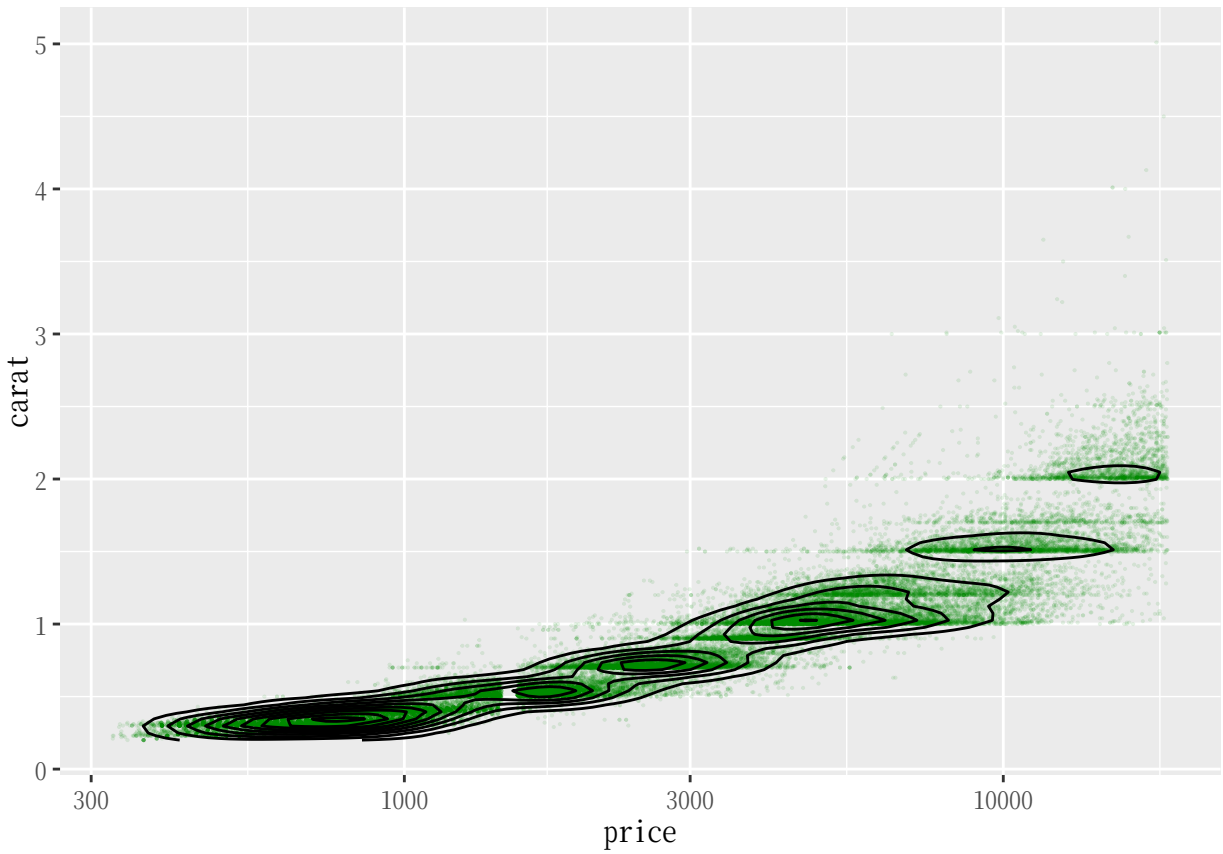
其中 `color` 是等值线颜色，`size` 是等值线粗细，单位是毫米。密度很低的区域被忽略了，为此，将 price 用对数轴：

```
p + stat_density_2d(color = "black", size = 0.6) +
  scale_x_log10()
```



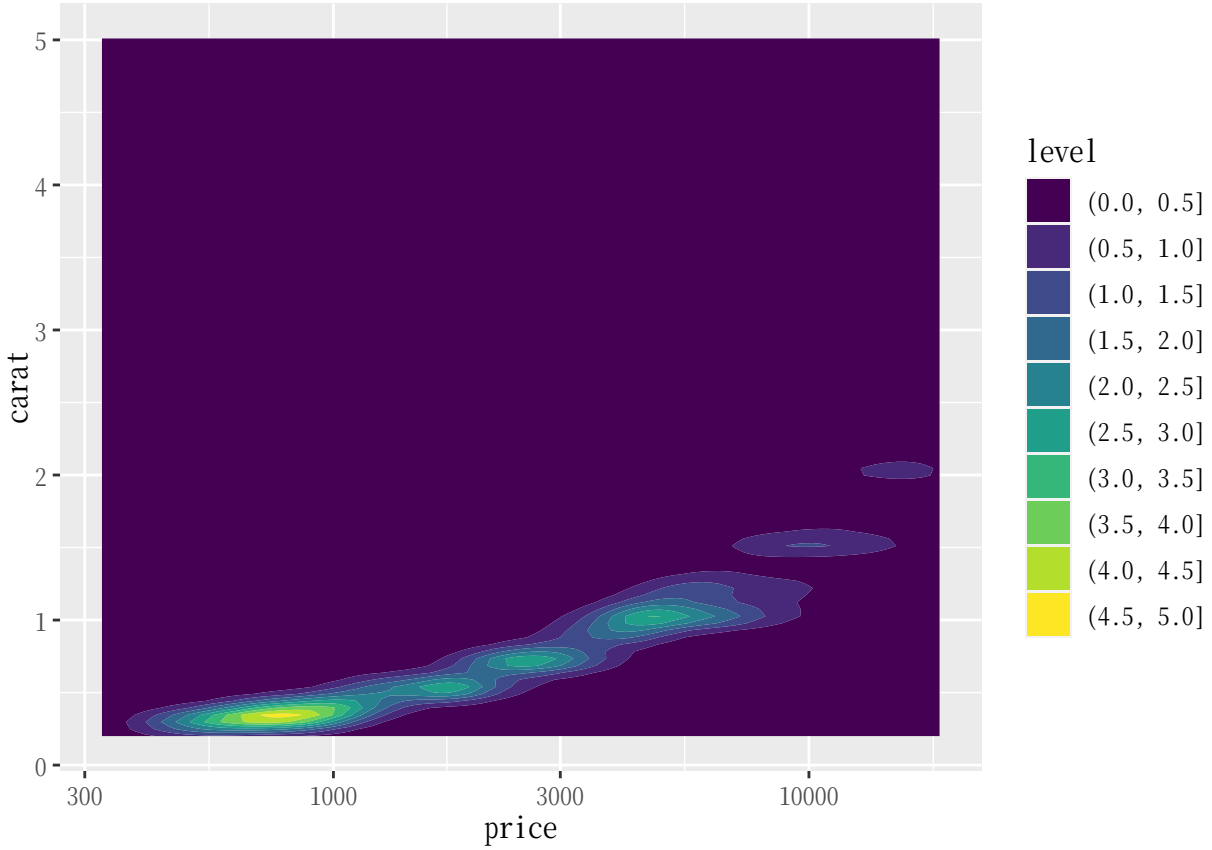
等值线图还可以与散点图叠加在一起：

```
p + geom_point(alpha = 0.1, size=0.05, color = "green4") +
  stat_density_2d(color = "black") +
  scale_x_log10()
```



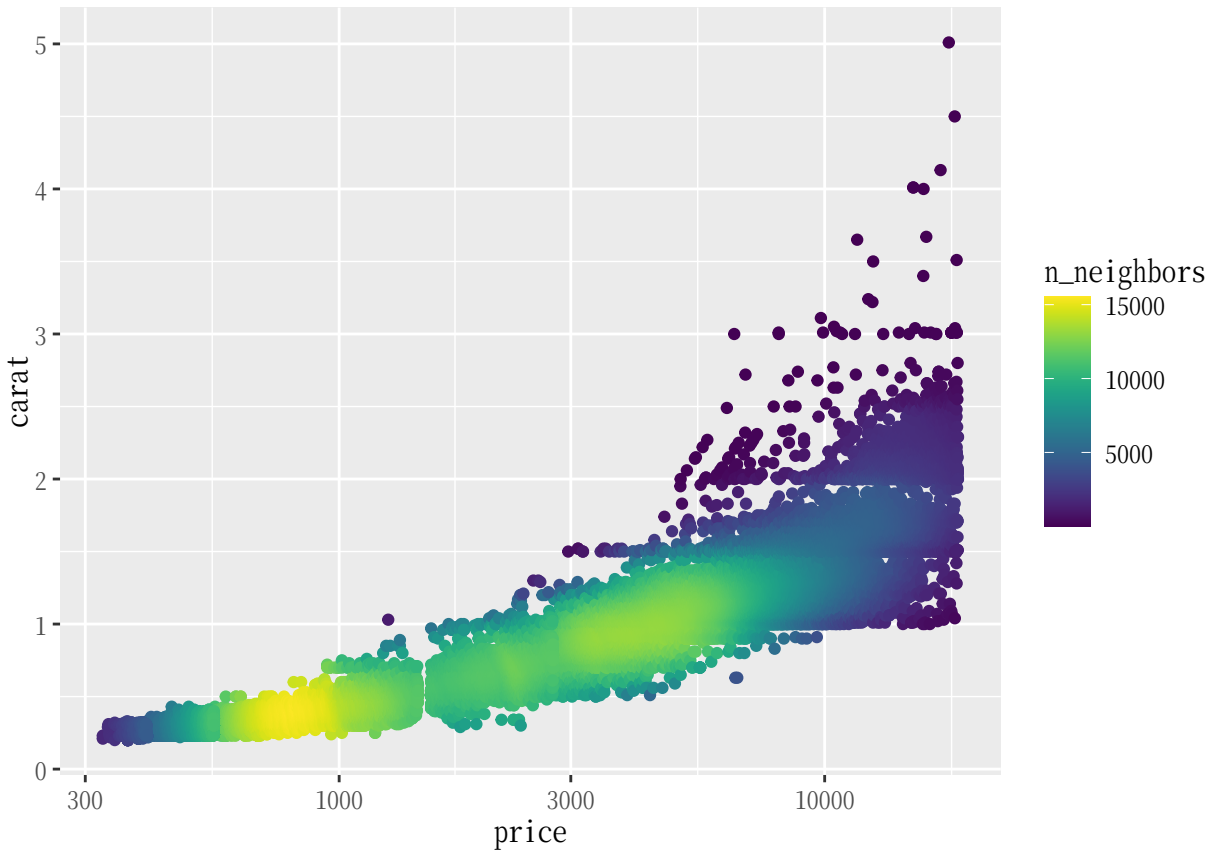
可以用颜色表示密度：

```
p + geom_density_2d_filled(
  mapping = aes(fill = ..level..)) +
  scale_x_log10()
```



ggpointdensity 包可以将散点和颜色表示的二元密度估计叠加在一起：

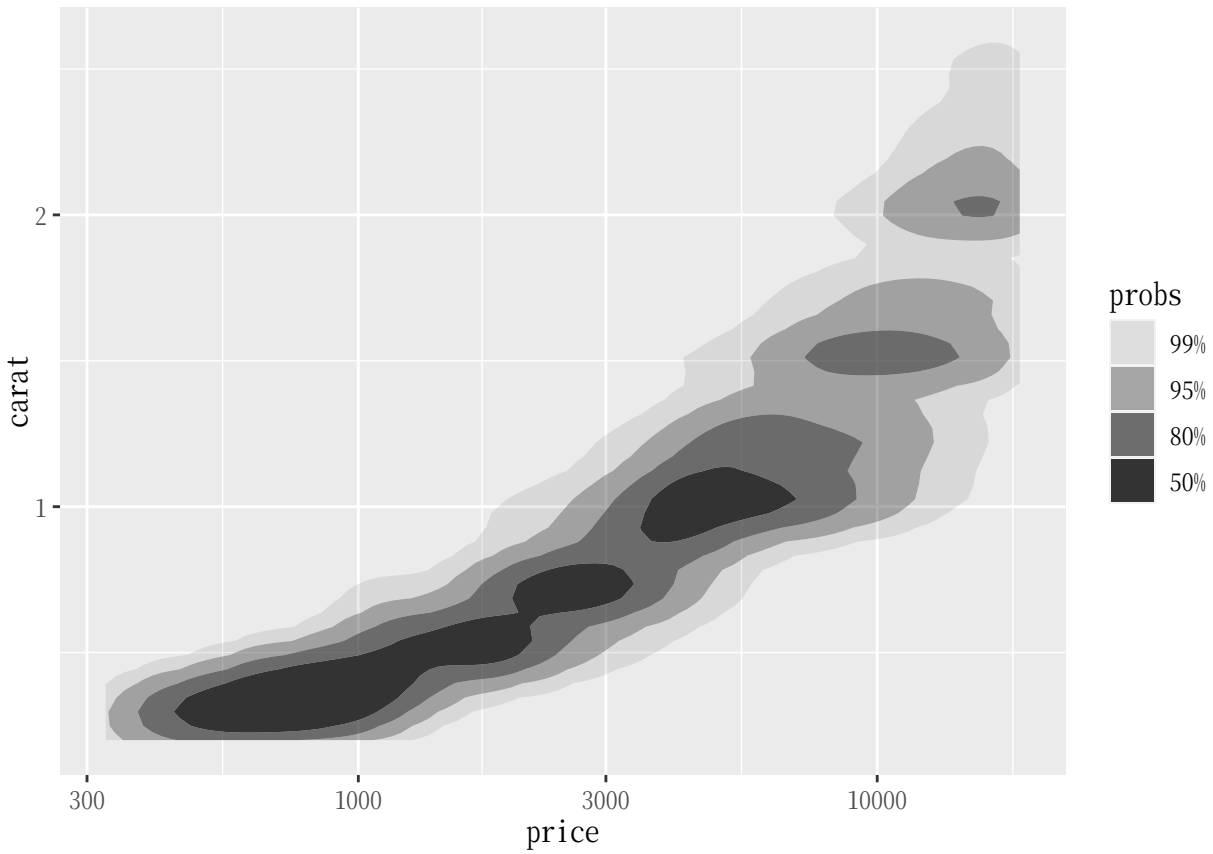
```
library(ggpointdensity)
p + geom_pointdensity() +
  viridis::scale_color_viridis() +
  scale_x_log10()
```



28.3.9.3 对应取值概率的等值线图

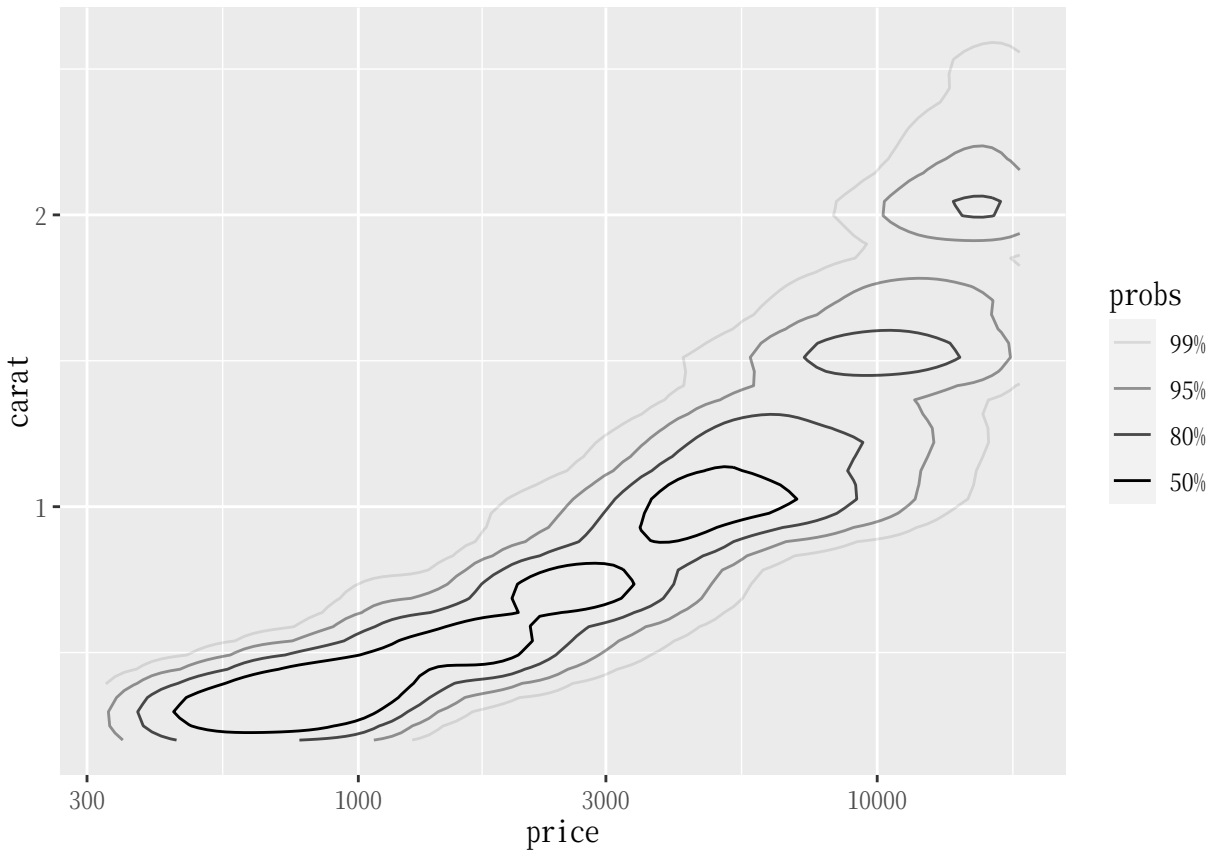
上面的密度图都是取二元密度函数值相等的地方连接起来成为等值线图，没有直接与分布概率挂钩。ggdensity 包则可以直接显示中心区域的取值概率，如：

```
library(ggdensity)
p <- ggplot(data = diamonds, mapping = aes(
  x = price, y = carat))
p + geom_hdr() +
  scale_x_log10()
```



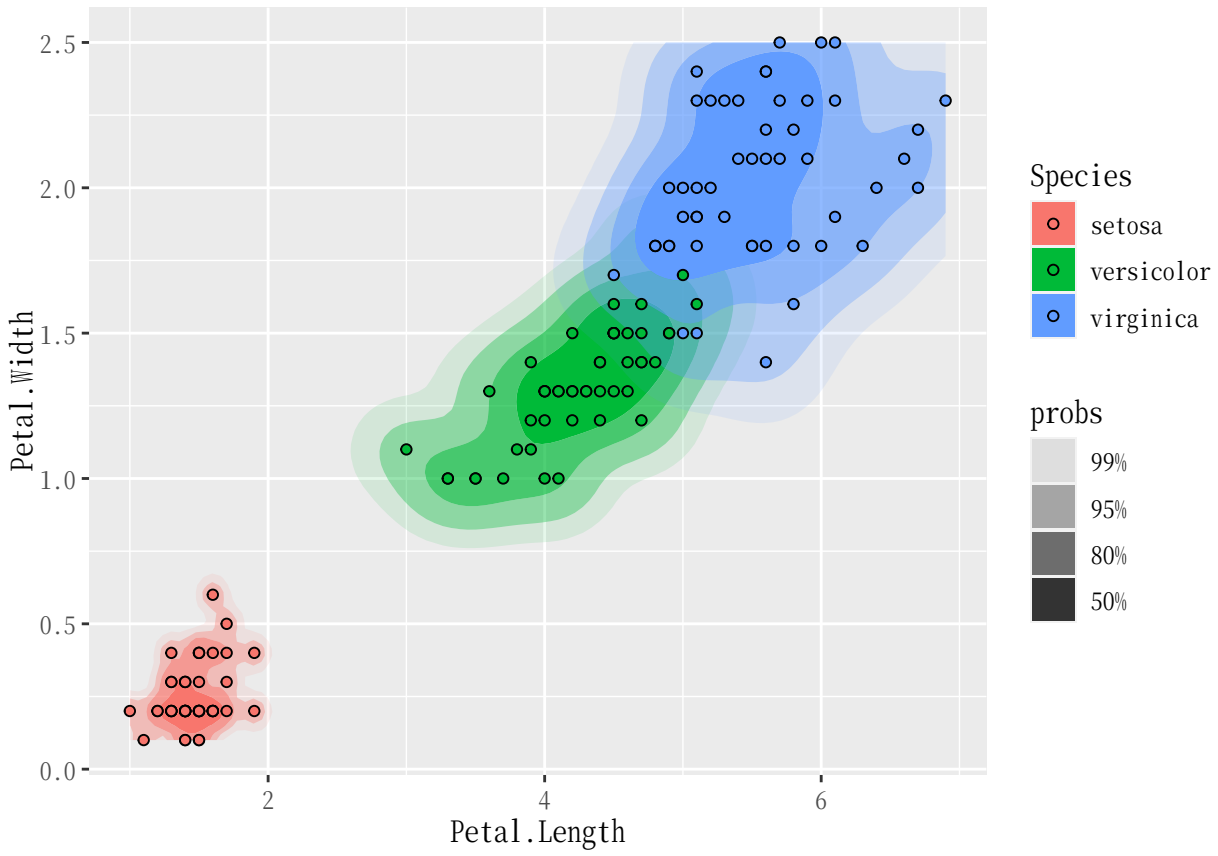
仅画等值线的版本：

```
p <- ggplot(data = diamonds, mapping = aes(
  x = price, y = carat))
p + geom_hdr_lines(size=0.5) +
  scale_x_log10()
```



分组估计的例子：

```
p <- ggplot(data = iris, mapping = aes(
  x = Petal.Length, y = Petal.Width,
  fill = Species))
p + geom_hdr() +
  geom_point(shape=21)
```



上图也可以用切片功能画成三个切片。

28.4 表现比例

条形图、堆叠条形图、并排条形图可以用来表现比例以及分组内的比例。饼图经常用来表现单个分组方式的各组比例。

28.4.1 单个分布

考虑 diamonds 数据集中各级别，首先计算其频数分布：

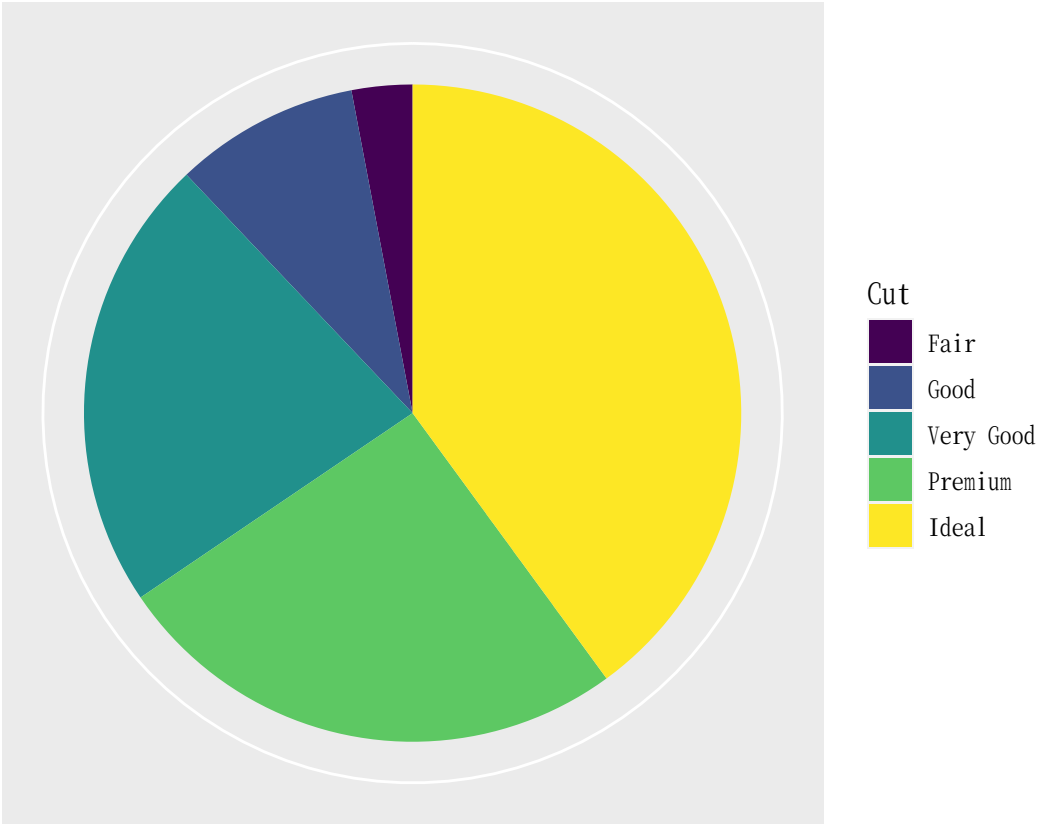
```
ddstab <- cdf_table(diamonds$cut)
knitr::kable(ddstab)
```

x	freq	pct	cumfreq	cumpct
Fair	1610	0.0298480	1610	0.0298480
Good	4906	0.0909529	6516	0.1208009
Very Good	12082	0.2239896	18598	0.3447905
Premium	13791	0.2556730	32389	0.6004635
Ideal	21551	0.3995365	53940	1.0000000

ggplot2 没有专门的饼图函数,可以用 `geom_col()` 与 `coord_polar()` 配合,或者用 ggforce 包的 `geom_arc_bar()`。

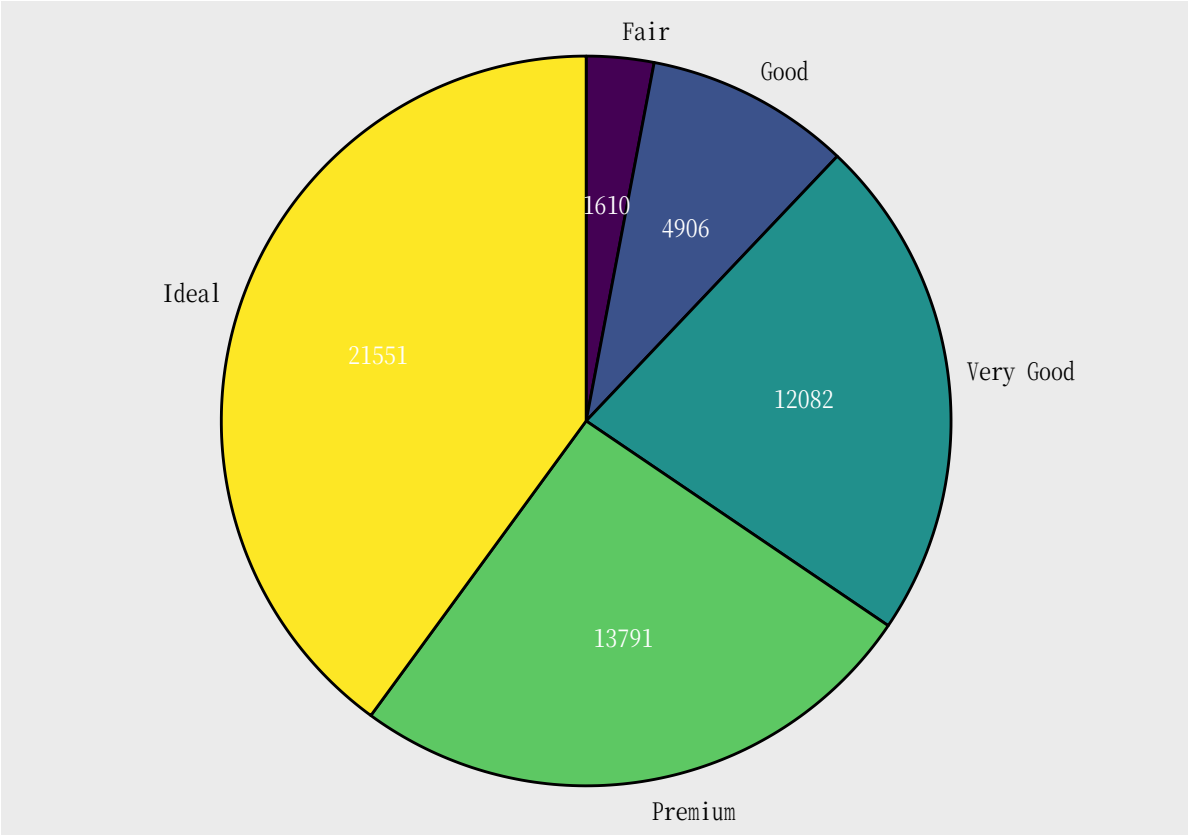
下面作级别分布的饼图，用饼图中色块大小代表不同级别的比例大小：

```
p <- ggplot(data = ddstab, mapping = aes(
  x = 1, y = freq, fill = x ))
p + geom_col() +
  coord_polar(theta = "y") +
  scale_x_continuous(name = NULL, breaks = NULL) +
  scale_y_continuous(name = NULL, breaks = NULL) +
  labs(fill = "Cut")
```



用 `geom_arc_bar()` 函数需要直接指定各个分界线的角度。程序如：

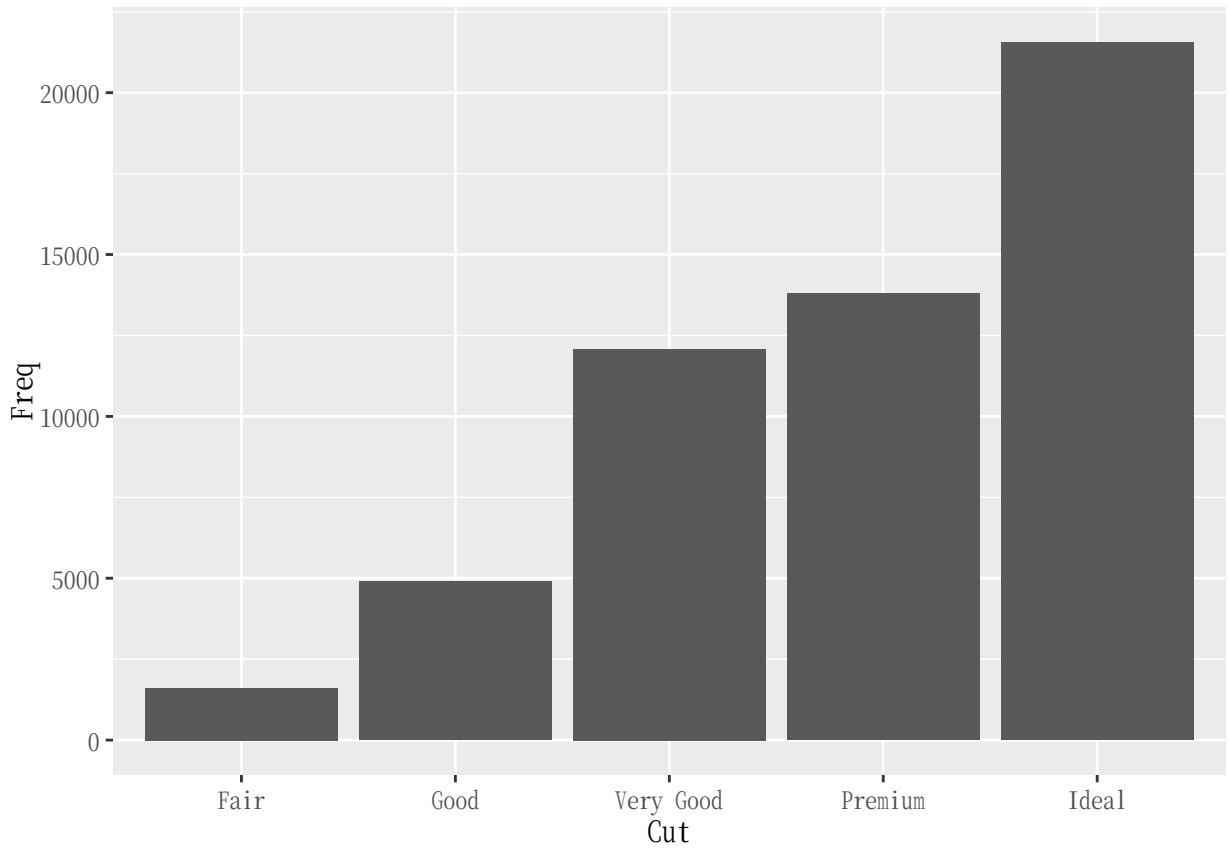
```
library(ggforce)
ddstab |>
  mutate(
    # 每块的结束角度
    end_angle = cumpct * 2*pi,
    # 每块的开始角度
    start_angle = lag(end_angle, default = 0),
    # 每块的中间角度，用于频数数值
    mid_angle = 0.5*(start_angle + end_angle),
    hjust = ifelse(mid_angle>pi, 1, 0),
    vjust = ifelse(mid_angle<pi/2 | mid_angle>3*pi/2, 0, 1)
  ) |>
  ggplot() +
  geom_arc_bar(mapping = aes(
    x0 = 0, y0 = 0, r0 = 0, r = 1.0,
    start = start_angle,
    end = end_angle,
    fill = x )) +
  geom_text(aes(
    x = 1.05*sin(mid_angle),
    y = 1.05*cos(mid_angle),
    label = x,
    hjust = hjust, vjust = vjust),
    size = 3) +
  geom_text(
    aes(
      x = 0.6*sin(mid_angle),
      y = 0.6*cos(mid_angle),
      label = freq ),
    color = "white", size = 3) +
  coord_fixed() +
  scale_x_continuous(
    expand = c(0.1, 0.4), breaks = NULL, name=NULL) +
  scale_y_continuous(breaks = NULL, name=NULL) +
  guides(fill = "none")
```



ggstatsplot 包提供了 `ggpiestats()` 函数，可以自动标注比例，但不能自动标注组标签在饼图边缘。

这样的频数或者比例分布，当然可以用普通的条形图表示，如：

```
ggplot(ddstab, aes(
  x = x, y = freq )) +
  geom_col() +
  labs(x = "Cut", y = "Freq")
```

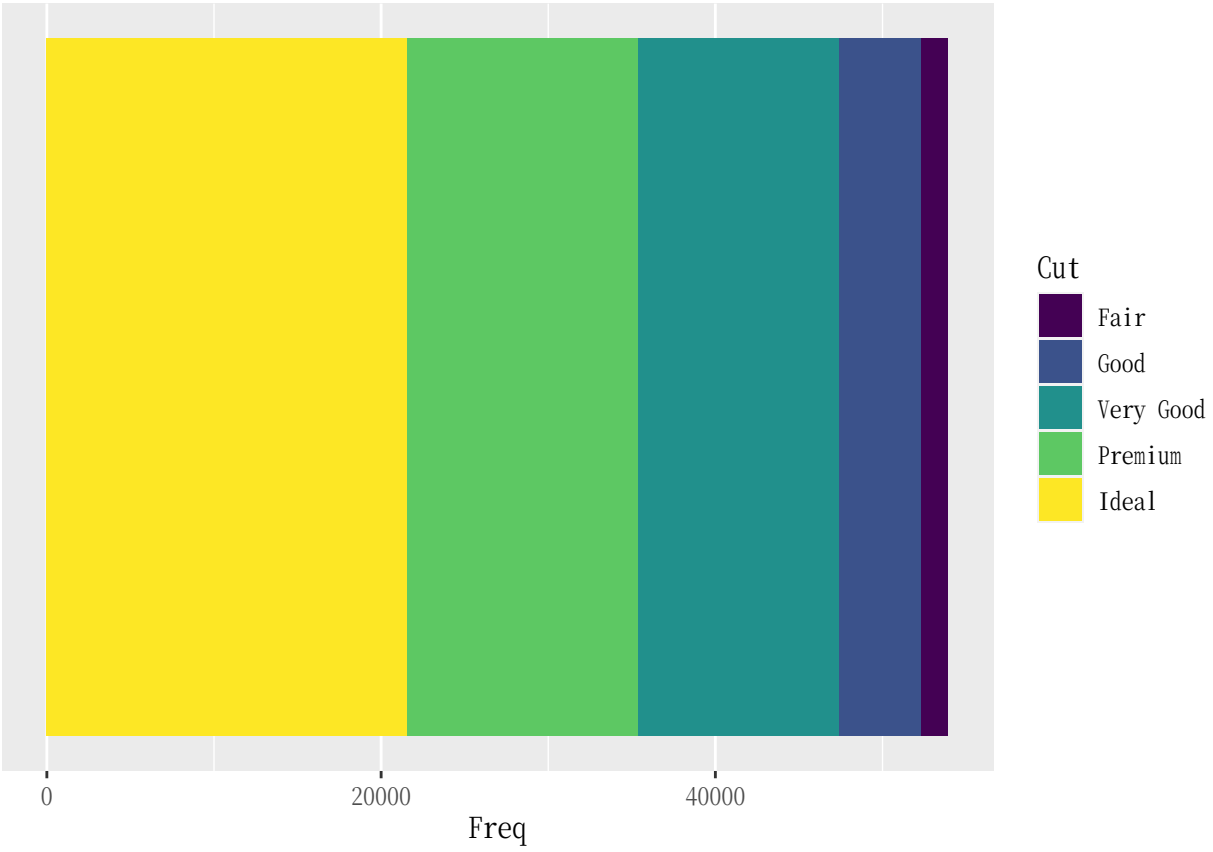


这样的图形比较容易在各类之间比较。也可以做成堆叠的条形图：

```
ggplot(ddstab, aes(
  x = 1, fill = x, y = freq )) +
  geom_col(position = "stack") +
  scale_x_continuous(breaks = NULL, name = NULL) +
  labs(fill = "Cut", y = "Freq")
```

也可以横向放置，用 `cood_flip()`:

```
ggplot(ddstab, aes(
  x = 1, fill = x, y = freq )) +
  geom_col(position = "stack") +
  scale_x_continuous(breaks = NULL, name = NULL) +
  coord_flip() +
  labs(fill = "Cut", y = "Freq")
```

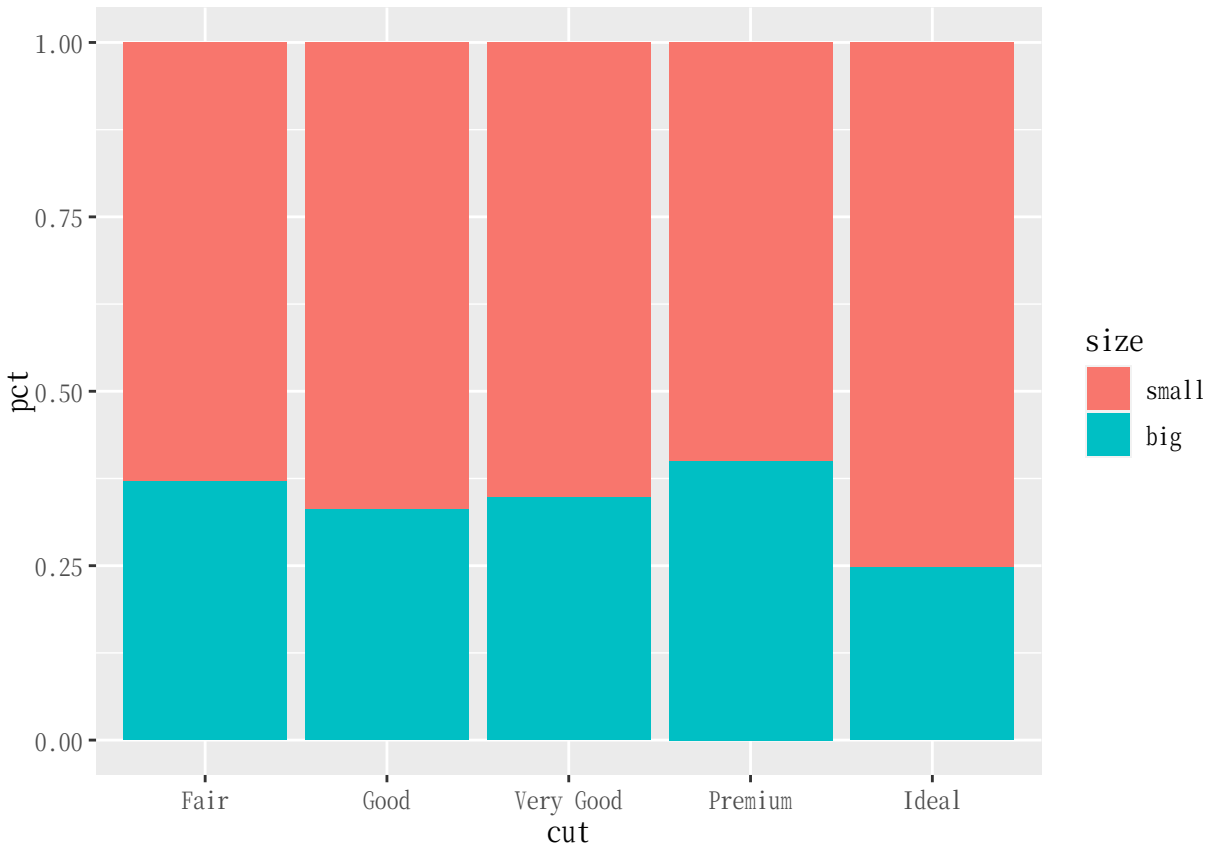


可以用 geom_text() 将各个类别标签和频数的数值直接标在色块上。堆叠的条形图比较容易看到每个类占总数的比例大小。

28.4.2 组间的比例分布的比较

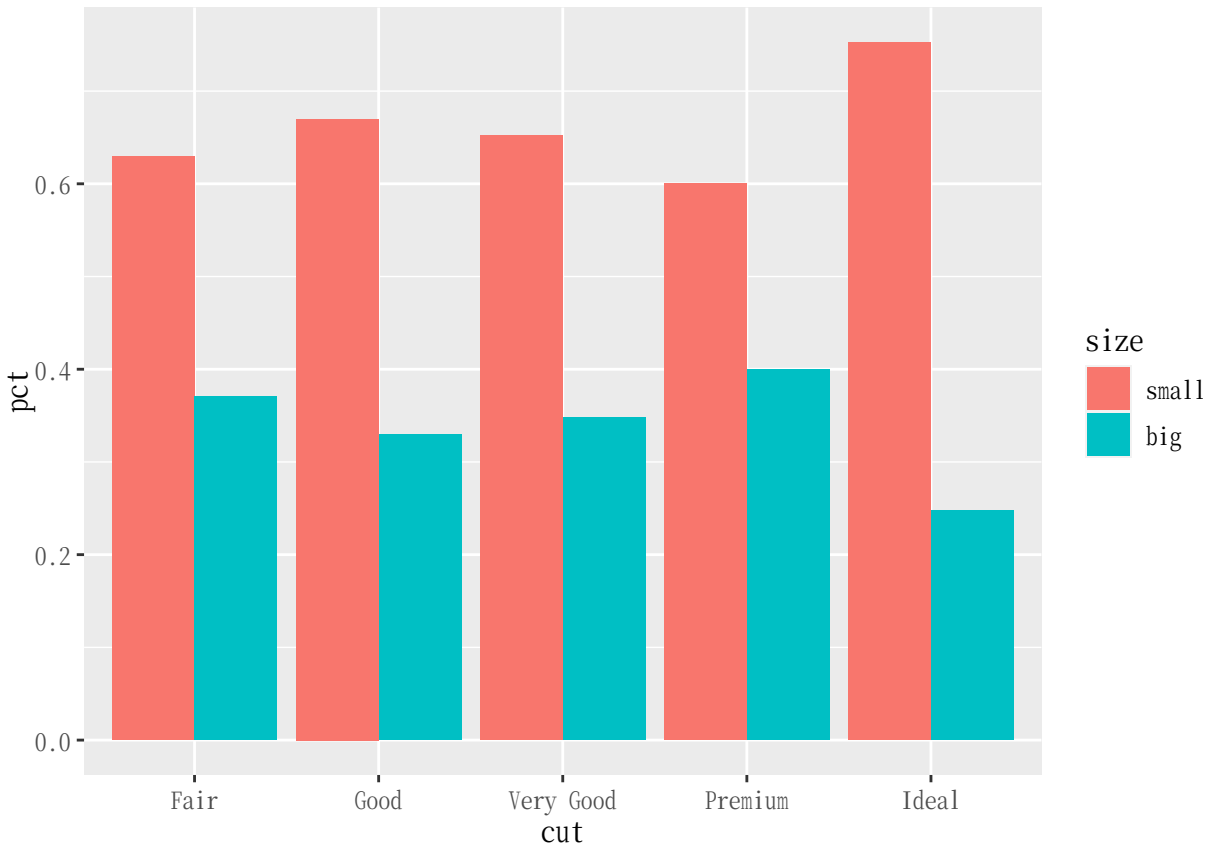
有时需要比较不同组的某个变量的比例分布。比如，我们将钻石按重量分为大和小两种，在 5 种不同品质之间比较大小比例，可以用堆叠的条形图或者并列的条形图。程序：

```
ddsmall |>
  mutate(size = factor(
    ifelse(carat >= 1.0, "big", "small"),
    levels=c("small", "big"))) |>
  count(cut, size) |>
  group_by(cut) |>
  mutate(pct = n / sum(n)) -> ddssize
p <- ggplot(ddssize, aes(
  x = cut, fill = size, y = pct ))
p + geom_col(position = "stack")
```



这种图形可以很容易地在 5 个组直接比较两种大小各自的比例。但是，如果有多于两个的比例要比较，则堆叠条形中位于中间的比例不易比较。可以做并列的条形图，如：

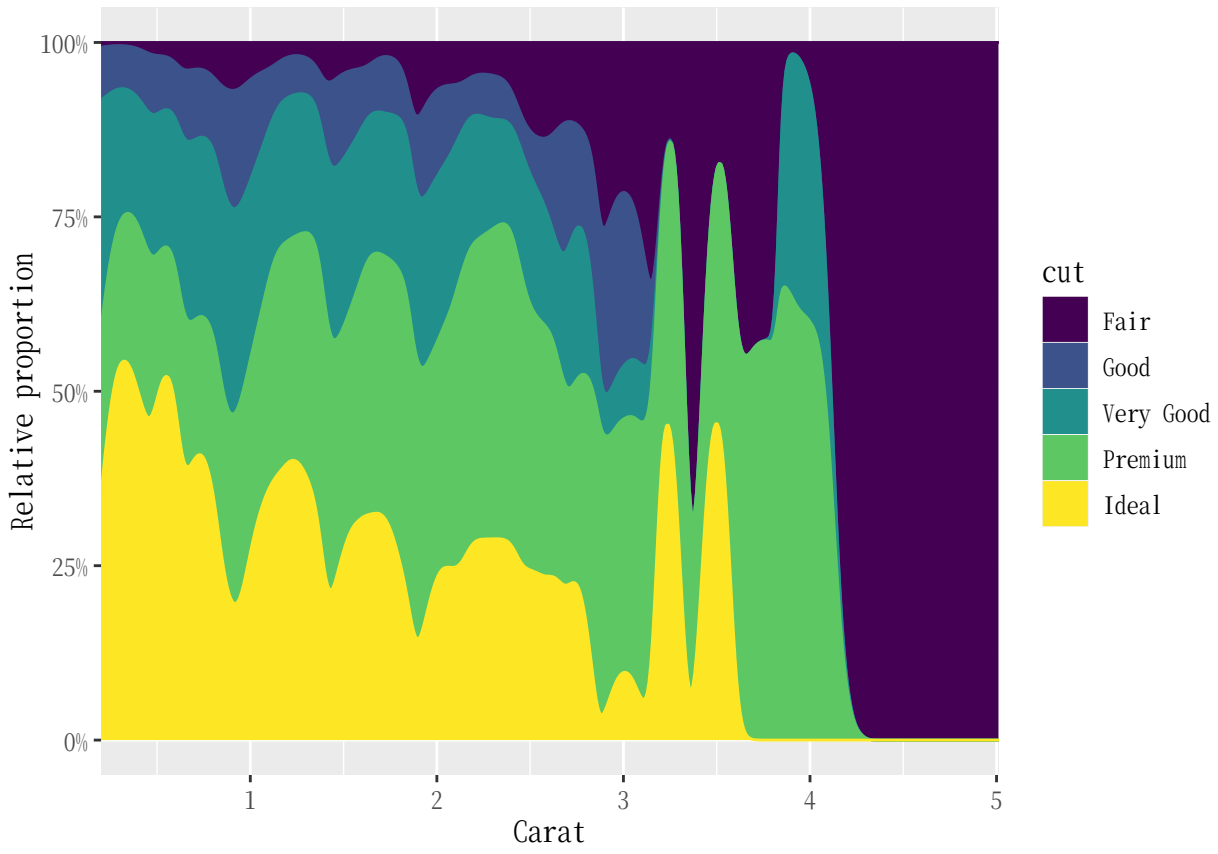
```
p + geom_col(position = "dodge")
```



这样可以比较容易地在不同大类之间比较比例，查看比例的变化趋势。

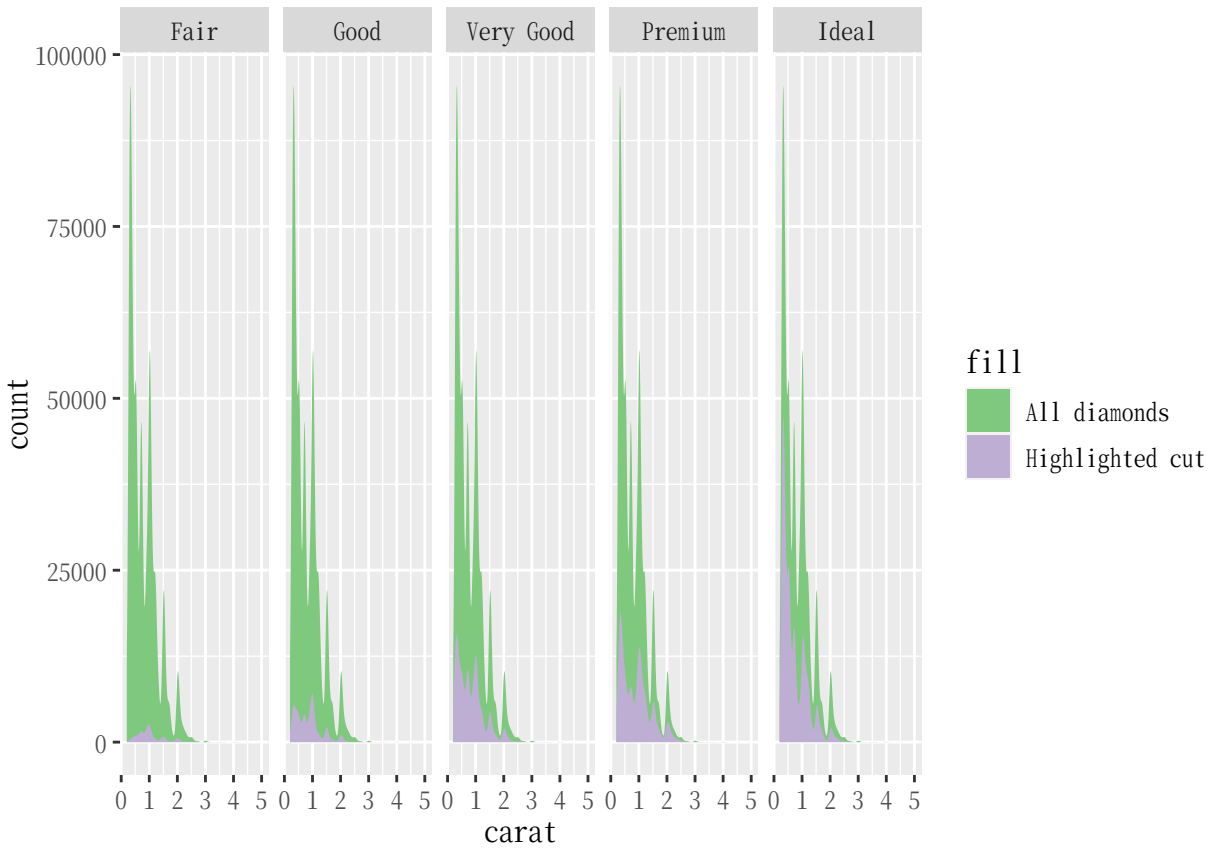
考虑 diamonds 数据集中随重量变化，品质分布的变化。因为重量是连续变化的，所以图形需要变成曲线或者阴影图，每条曲线表示某个品质的累积比例。

```
ggplot(diamonds, aes(
  x = carat,
  y = ..count..,
  fill = cut,
  color = cut)) +
  geom_density(position = "fill") +
  scale_x_continuous(name = "Carat", expand=c(0,0)) +
  scale_y_continuous(
    name = "Relative proportion",
    labels = scales::percent)
```



这样的图形仅表现了每个重量级别的品质比例，无法表现不同重量级别的数量。为此，可以每种品质单独画总的重量分布密度图，但在其中用阴影标出该品质所占比例：

```
ggplot(diamonds, aes(
  x = carat, y = ..count..)) +
  geom_density_line(
    data = select(diamonds, -cut),
    mapping = aes(
      fill = "All diamonds"),
    color = "transparent") +
  geom_density_line(mapping = aes(
    fill = "Highlighted cut"),
    color = "transparent") +
  scale_fill_brewer(type = "qual") +
  facet_wrap(~ cut, nrow = 1)
```



因为程序中用了 `facet_wrap()` 分组，所以密度图（纵轴为个数）本应只有某个品级的数据，但是用了删除 cut 变量的方法，将全集数据的密度图也画出来了。这个数据集中，高品质的钻石更多。

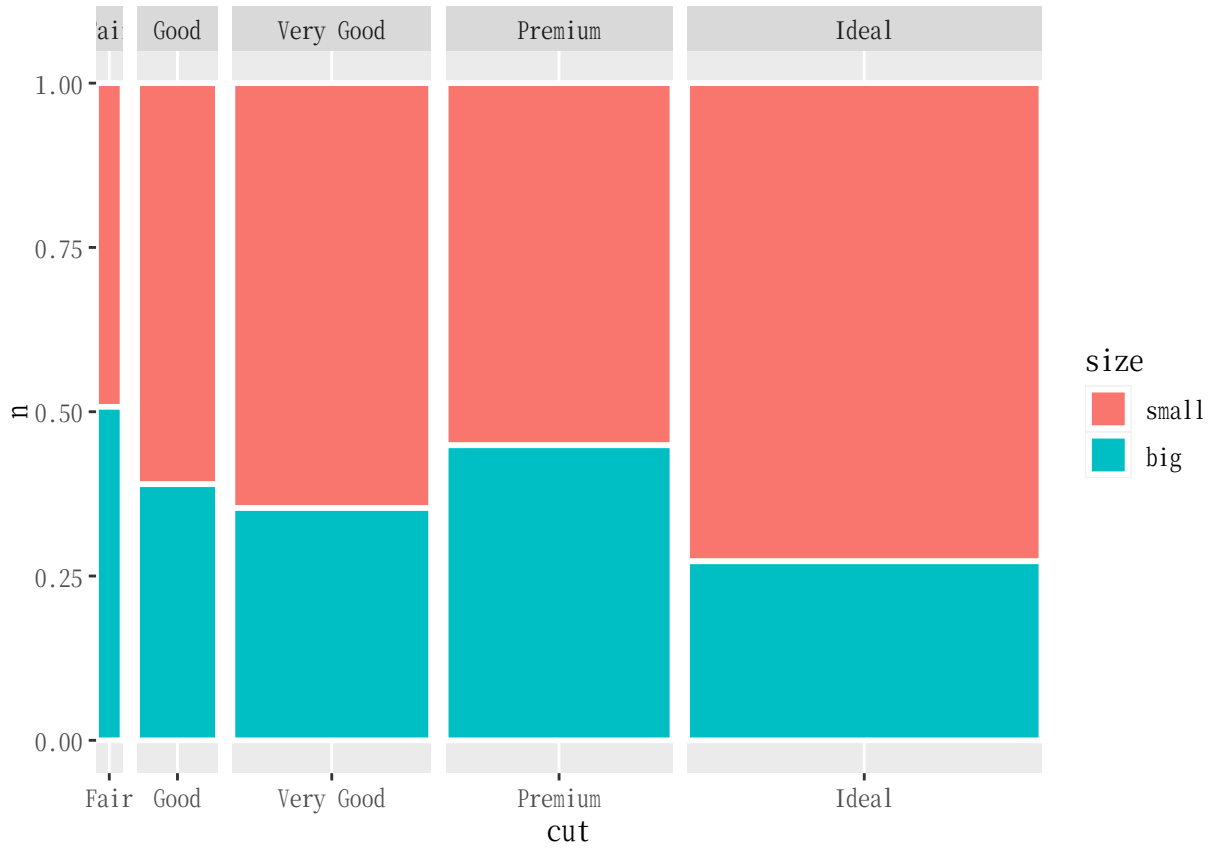
28.4.3 嵌套比例分布的比较

堆叠条形图、并列条形图都可以用来表现两重分类的列联表频数数据。堆叠条形图中每个条形的高度可以用来表现大类频数，每个条形内色块的大小可以用来表现大类内小类的频数和比例。不同大类之间的小类频数和比例相对来说不易比较。并列条形图中每个条形是一个交叉类的频数，所以在大类之间比较小类频数比较容易，但是每个大类的频数则表现得较为模糊。

马赛克图是另外一种表现列联表频数的图形，用色块面积表示每个交叉类的频数。与堆叠条形图相比，这里用了条形宽度而不是条形高度表示大类频数。

```
diamonds |>
  mutate(size = factor(
    ifelse(carat >= 1.0, "big", "small"),
    levels=c("small", "big"))) |>
  count(cut, size) |>
  group_by(cut) |>
  mutate(
```

```
    cutfreq = sum(n),
    .groups = "drop_last") -> ddssize0
ggplot(ddssize0, aes(
  x = cut,
  fill = size,
  y = n,
  width = cutfreq )) +
  geom_col(
    position = "fill",
    color="white",
    size=1) +
  facet_grid(
    ~ cut,
    scales = "free_x",
    space = "free_x")
```



上面的程序用了 `geom_col()`、`facet_grid()` 和 `width` 参数实现马赛克图。

ggmosaic 包可以制作类似图形，但可以用宽度表示大类的频数比例：

```
library(ggmosaic)
ddsmall |>
  mutate(size = factor(
    ifelse(carat >= 1.0, "big", "small"),
    levels=c("small", "big"))) |>
  ggplot() +
  geom_mosaic(mapping = aes(
    x = product(cut),
    fill = size)) +
  theme_mosaic()
```

mosaic 包还支持更多分类变量的频数统计图示。

与马赛克图类似的一种方法是在大类中再细分小类，而且细分小类的办法不是单向划分的。treemapify 包的 `geom_treemap()` 函数可以做树状分类图。如：

```
library(treemapify)
ggplot(ddssize0, aes(
  subgroup = cut,
  fill = interaction(size, cut),
  area = n)) +
  geom_treemap(
    color="white", size=0.5*.pt, alpha=NA) +
  geom_treemap_subgroup_text(
    place = "center", alpha = 0.5, grow = TRUE) +
  geom_treemap_text(mapping = aes(
    label = size),
    color = "white",
    place = "center", grow = FALSE) +
  guides(fill = "none")
```

还可以用平行集（parallel sets）方法表现多个分类之间的关系，以一个分类为主分类染色，可以看出每个主分类与其他子类的关系。ggforce 包的 `geom_paralles_sets()` 函数作这种图。参见https://serialmentor.com/dataviz/nested-proportions.html。

28.5 表现多个变量间的关系

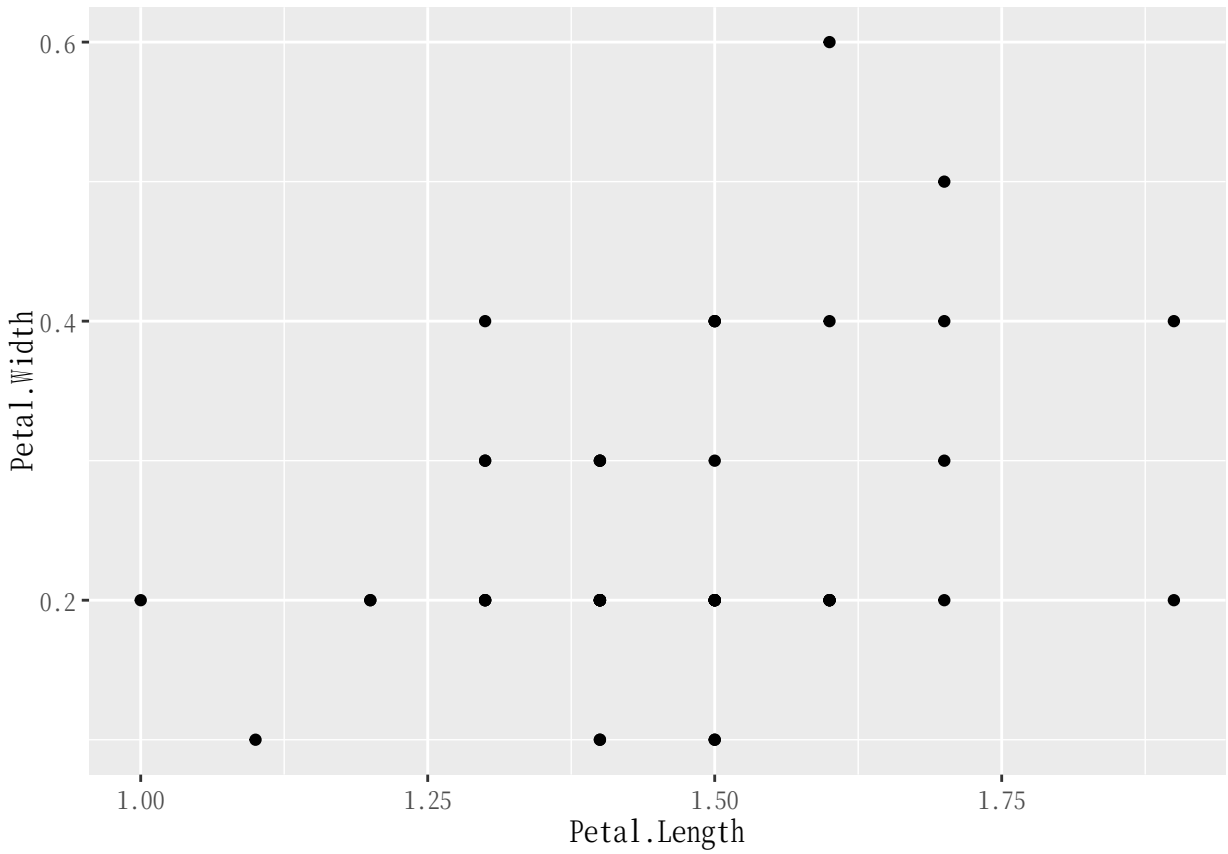
当数据集中有多个变量中，我们除了关心每一个变量的类型、取值集合、分布情况，还关心变量之间的关系，观测的分组情况等。

为表现两个变量之间的关系，最常用的是散点图。多个变量之间可以用散点图矩阵、相关图，可以在散点图中用符号大小、符号颜色、符号形状表示更多维数。对于高维数据，经常需要利用降维方法，如主成分分析 (PCA) 对数据降维，对降维数据作图。

28.5.1 散点图

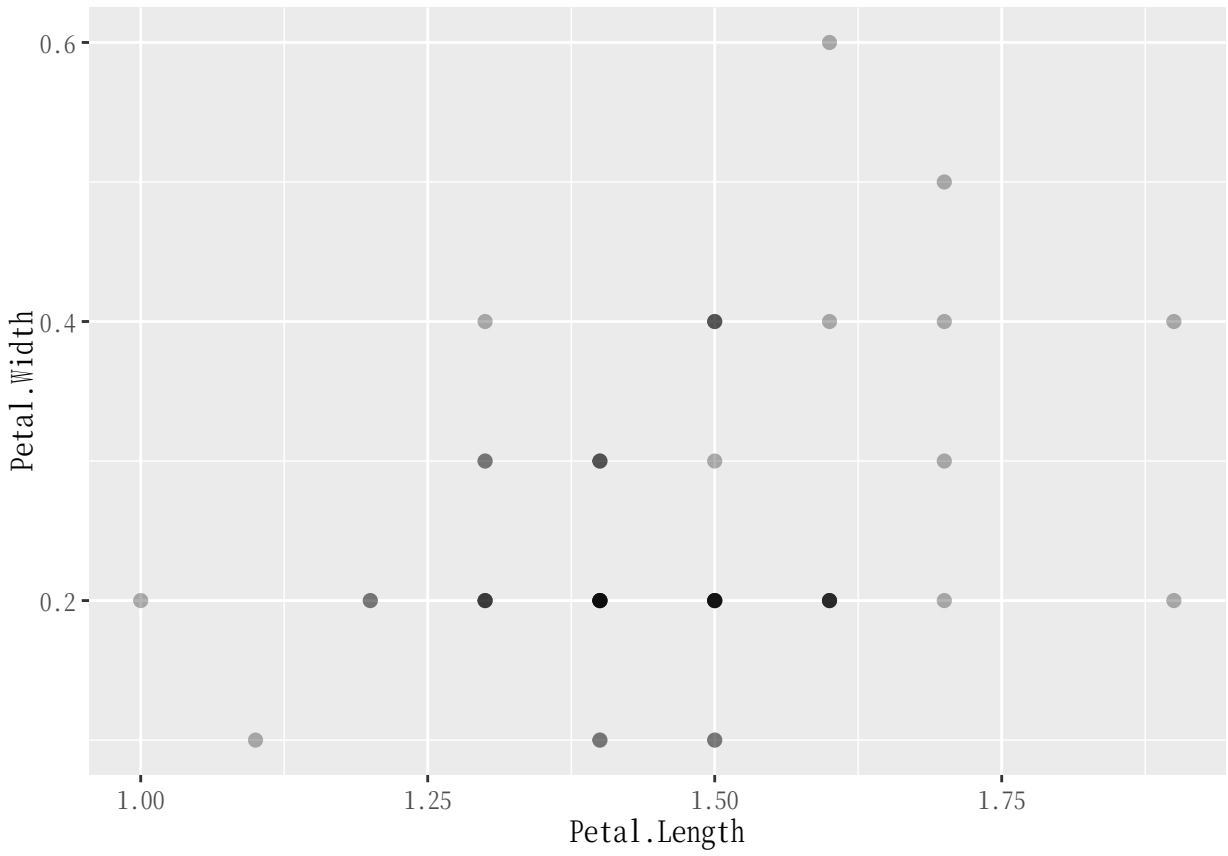
R 软件自带的 iris 数据集中包含了三种鸢尾花的 150 个样品的测量数据，每种各 50 个样品，每个样品测量了花瓣、花萼的长、宽。下面画 50 个 setosa 样品的花瓣长、宽的散点图，可以看出，两种有明显的线性相关关系：

```
p <- ggplot(
  data = filter(iris, Species == "setosa"),
  mapping = aes(x = Petal.Length, y = Petal.Width))
p + geom_point()
```



应该有 50 个点，但实际只看到 23 个点。这是因为许多点重叠在一起了。加上透明度参数可以使得重叠的点颜色更深：

```
p + geom_point(alpha = 0.3, size = 2.0)
```

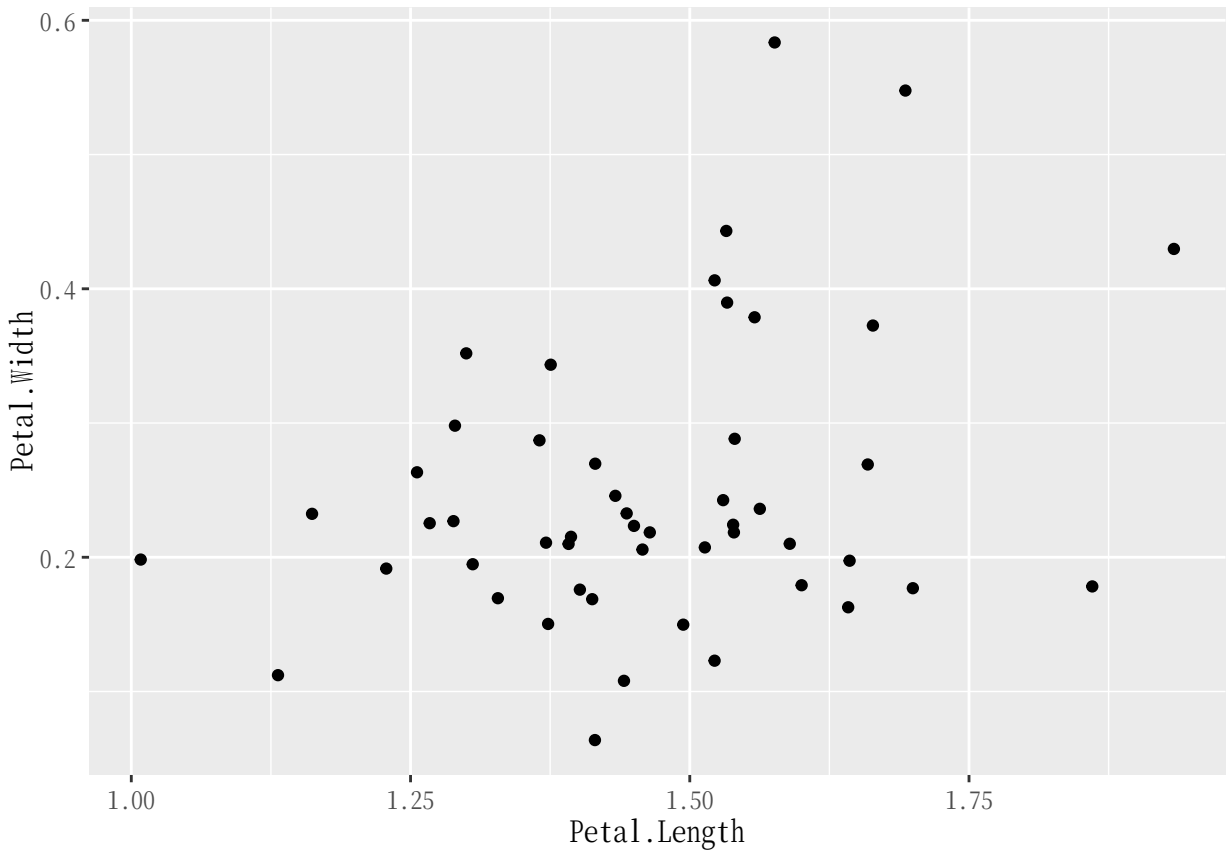


图的效果还是不够明显，不太好判断重叠了多少个点，另外如果不同颜色的点重叠，就无法判断有哪些颜色的点重叠在一起。如果不做说明，读者也不一定了解颜色深浅代表重叠点的多少。

当点数很多，如数百、数千个点时，可以用 `size` 参数画更小的点，这时使用透明度的效果会比较显著。注意，透明度是一个 0 到 1 之间的数值，`1/3` 表示重叠 3 个点会变成不透明，透明度 `1/10` 表示重叠 10 个点会变成不透明。

在点数较少时如果有重叠，可以使用 `geom_jitter()`，使得每个点略有随机偏移，就可以使得各个点基本上不重叠。偏移可以是上下左右同时进行，也可以要求仅上下或者仅左右偏移。例如：

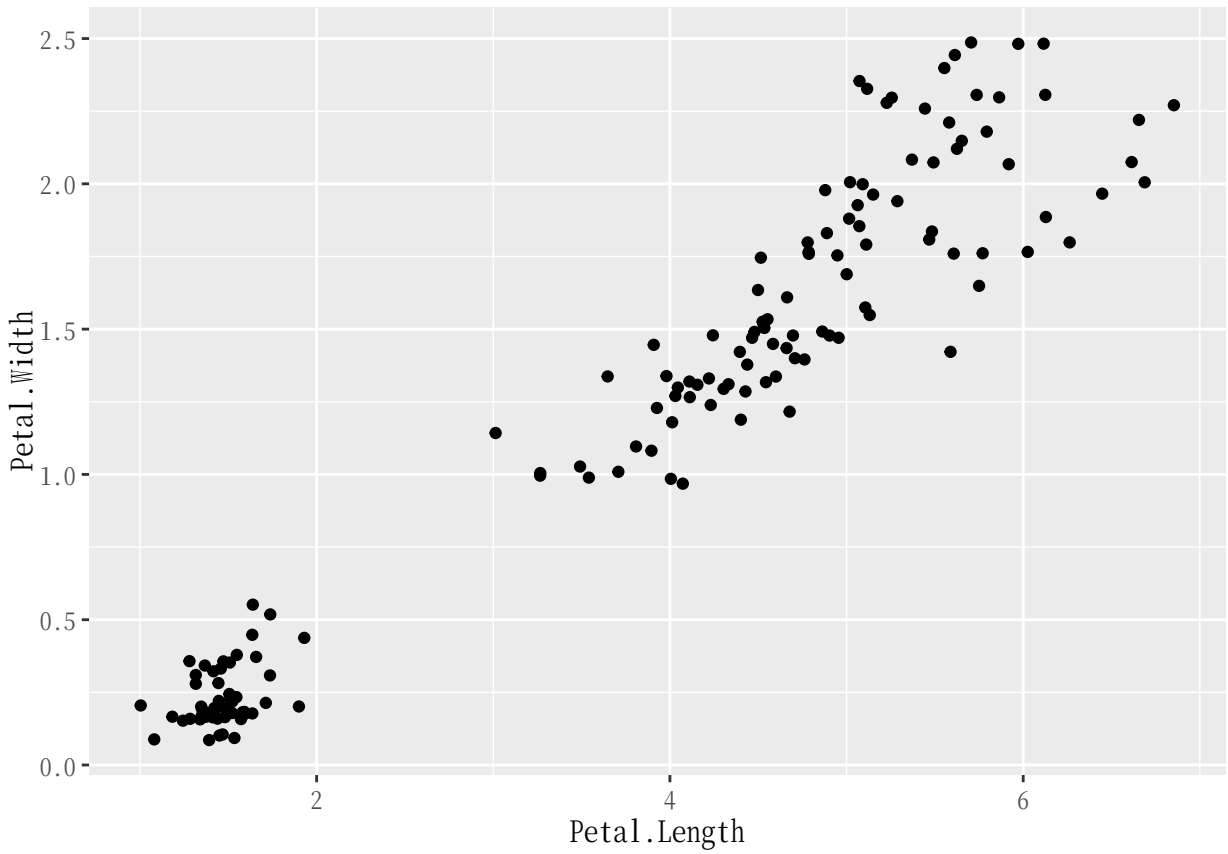
```
p + geom_jitter(width=0.05, height=0.05)
```



geom_jiter() 中的 width 和 height 是左右和上下抖动的幅度百分比，以数据间的最小差距为单位，默认的抖动比例是 40%，一般不超过 50%，否则原来是两个不同坐标值的点可能会看起来是同一坐标值。比如，假设 x 变量的值是精确到小数点后 1 位的，取值如 1.3, 1.4，抖动 40%，坐标可以变成 1.33, 1.46，如果允许抖动 60%，则 1.3 可能变成 1.36，1.4 可能变成 1.34，就无法区分原来不同的值了。

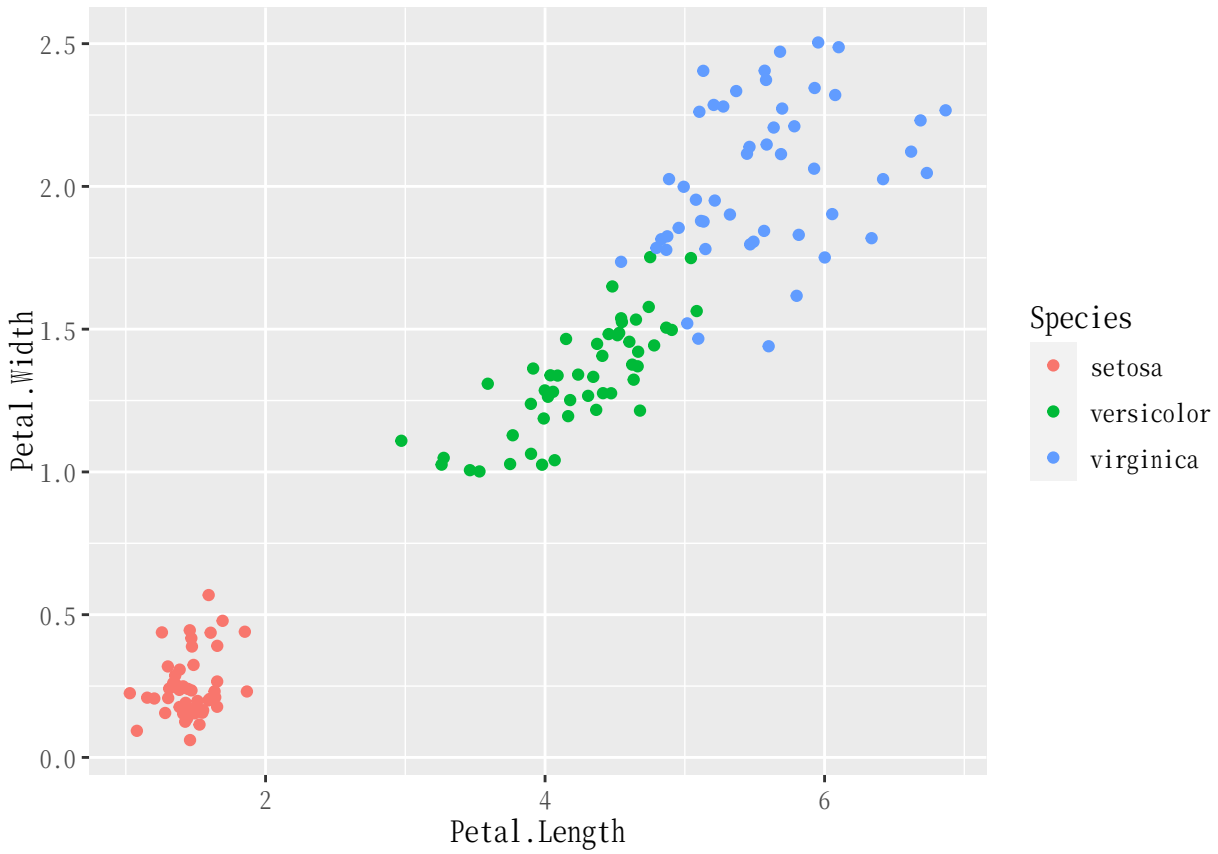
如果做所有的 150 个样品的散点图，则散点图呈现出分组现象：

```
ggplot(iris,  aes(
  x = Petal.Length, y = Petal.Width)) +
  geom_jitter(width=0.05, height=0.05)
```



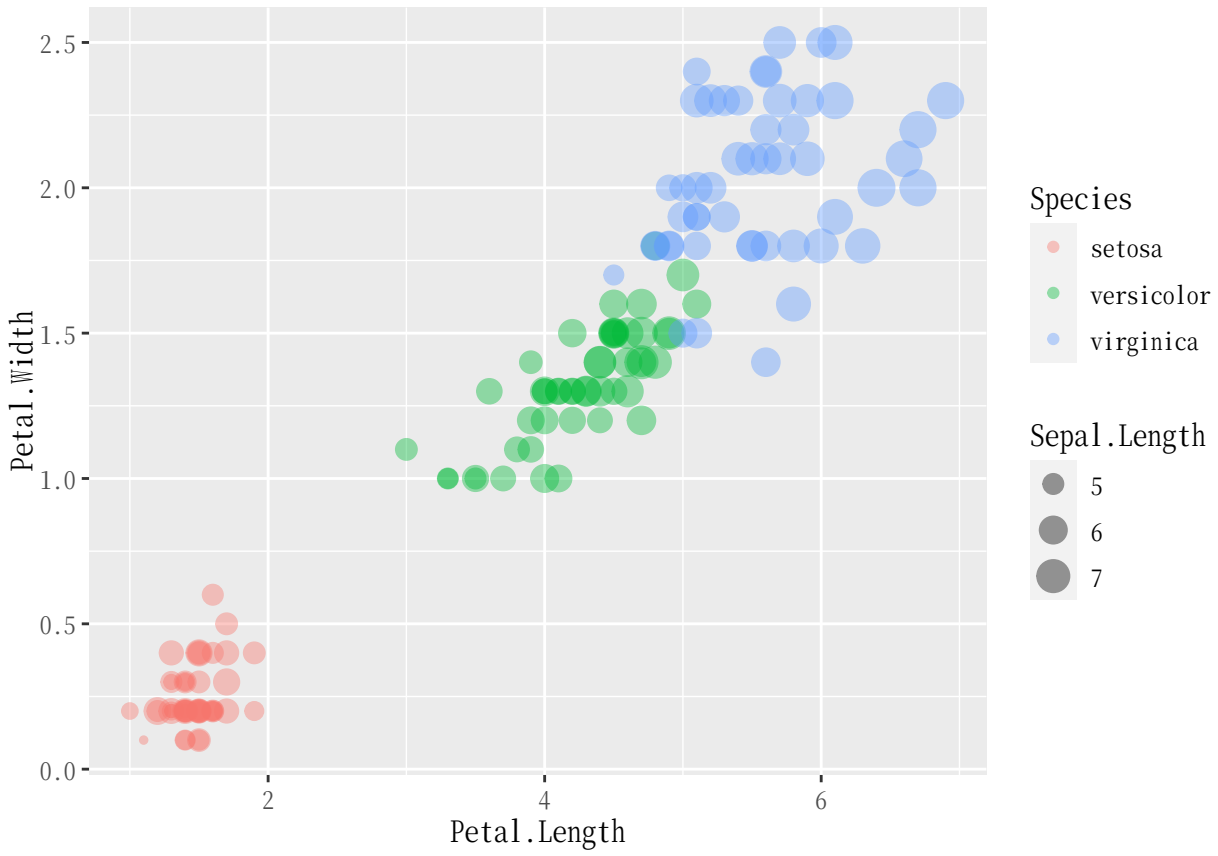
为此，应该设法在散点图中区分三种不同鸢尾花品种。将 `Species` 映射到 color 属性即可区分：

```
ggplot(iris, aes(
  x = Petal.Length,
  y = Petal.Width,
  color = Species)) +
  geom_jitter(width=0.05, height=0.05)
```



如果希望将花萼长度也添加到图形中，因为已经有 x 维、y 维和颜色，只能通过符号大小来添加。将花萼长度映射到 `size` 维：

```
ggplot(iris, aes(
  x = Petal.Length,
  y = Petal.Width,
  size = Sepal.Length,
  color = Species)) +
  geom_point(alpha = 0.4)
```



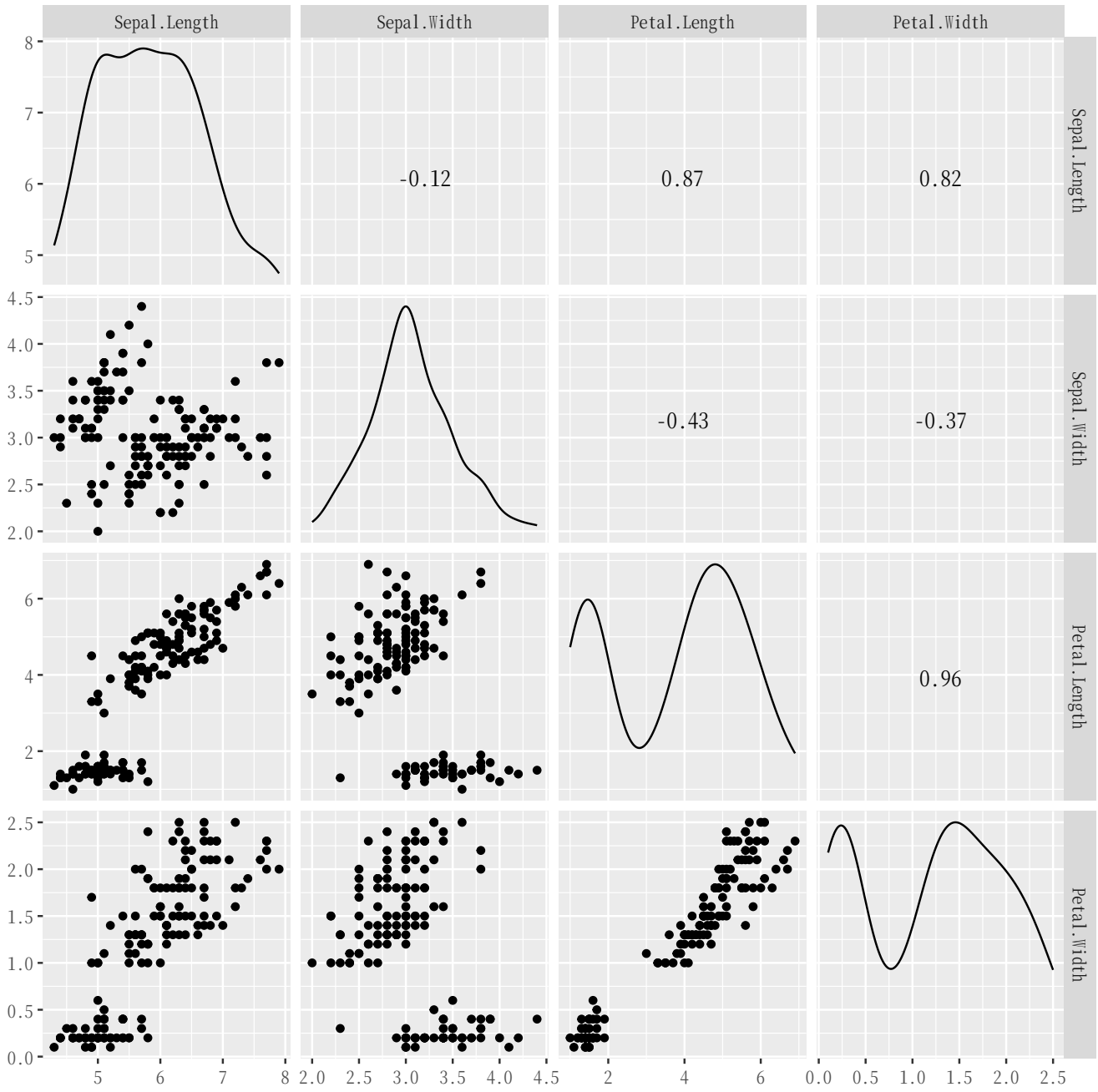
这样的用符号大小代表一个变量数据的图形称为气泡图 (bubble plot)。符号大小是比较难认读的图形刻度。

28.5.2 散点图矩阵

多个变量之间的关系经常用散点图矩阵表示。ggplot2 包没有提供专门的散点图矩阵，基础 R 图形中提供了 pairs 函数作散点图矩阵，GGally 包提供了一个 `ggscatmat()` 函数作散点图矩阵。

例如，对 iris 数据的四个测量值变量作散点图矩阵：

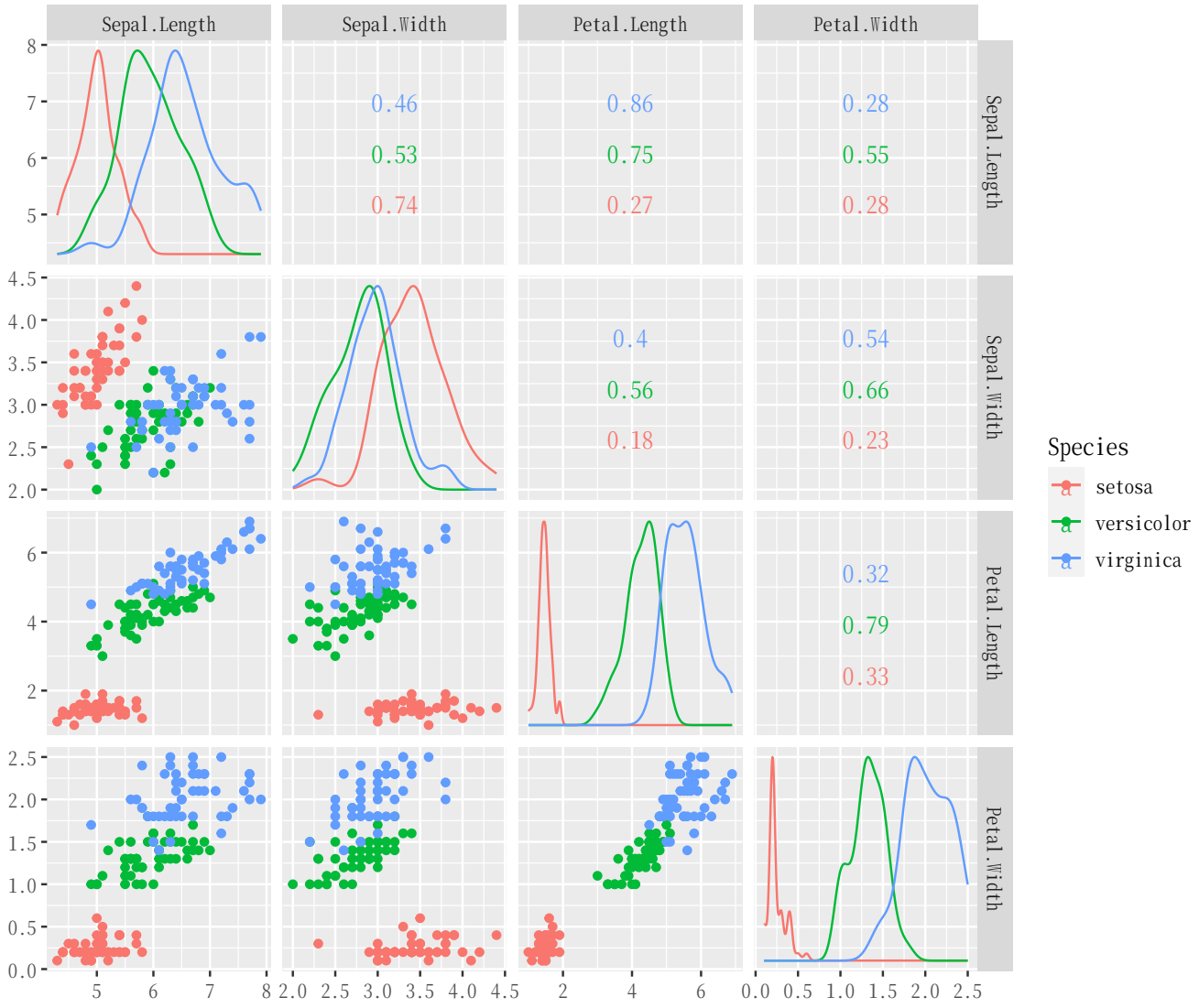
```
library(GGally)
ggscatmat(data = iris, columns = 1:4)
```



排列成矩阵的各个切片的下三角位置是两个变量的散点图，对角线位置是单个变量的核密度估计，上三角位置是两个变量的相关系数。

可以用 color 选项增加 color 映射，可以用 alpha 指定透明度。如：

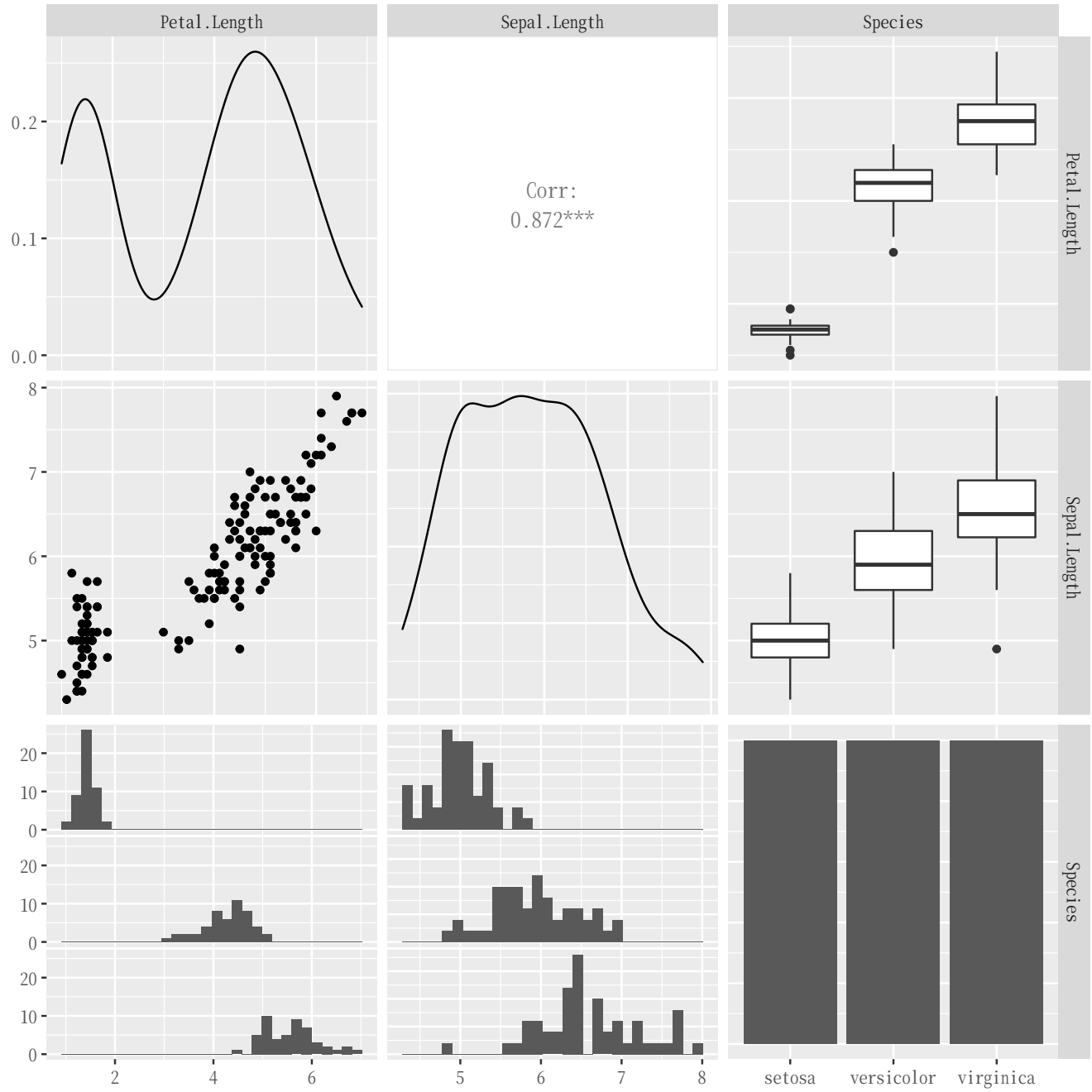
```
ggscatmat(data = iris, columns = 1:4,
  color = "Species")
```



GGally 的 `ggpairs()` 函数提供了另一种矩阵图，可以比较变量两两分布或者关系。例如，取 iris 数据集的花瓣长、花萼长与种类：

```
ggpairs(
  data = iris,
  columns = c("Petal.Length", "Sepal.Length", "Species"))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

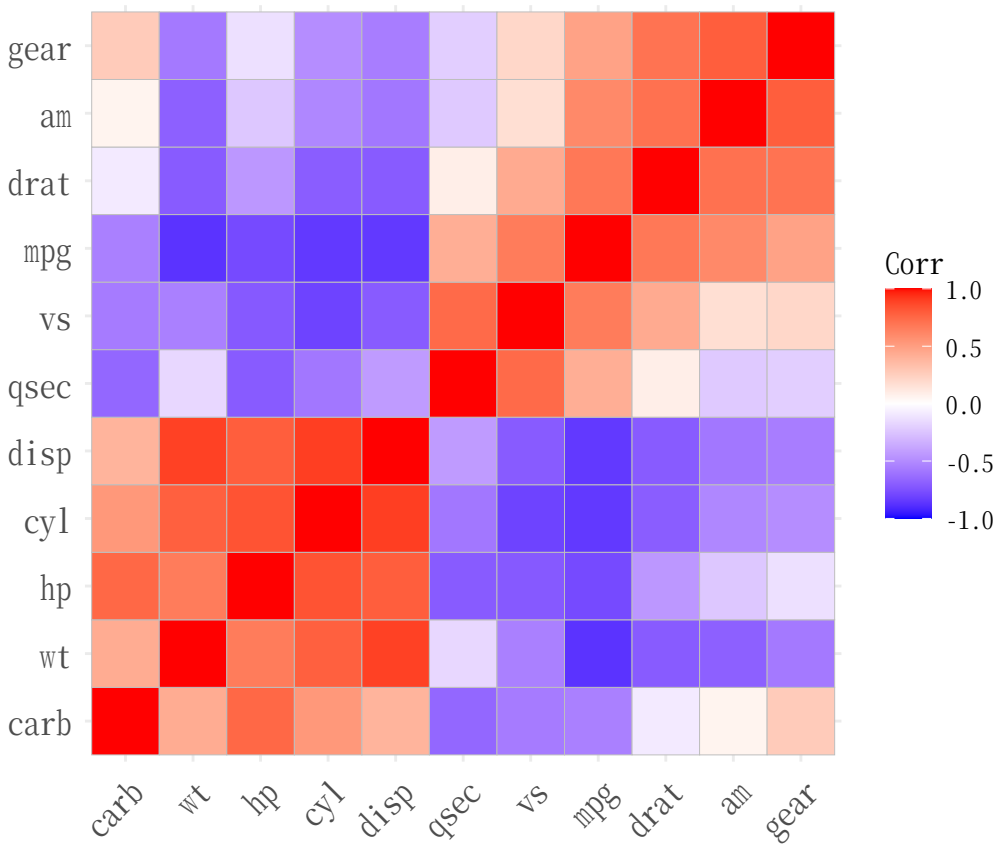


可以看出，对角线位置是单个变量的密度估计或者离散分布条形图，两个连续变量之间有散点图与相关系数，连续变量与分类变量之间有按分类变量分组的直方图和盒形图。

28.5.3 相关系数矩阵图

多个变量之间的相关系数矩阵可以用色块图表示，ggcorrplot 包提供了这样的功能：

```
library(ggcorrplot)
data(mtcars)
ggcorrplot(cor(mtcars),
  hc.order=TRUE)
```



其中 `hc.order=TRUE` 对变量进行聚类，使得正相关的变量排列在相邻位置。可以用 `lab` 选项标出数值，用 `cor_pmat()` 函数计算每个相关系数 p 值，在 `ggcorrplot()` 中用选项 `p.mat` 输入 p 值矩阵后可以标出不显著的相关系数。

GGally 包的 `ggcorr()` 函数、arm 包的 `corrplot()` 函数也能制作相关系数图，功能不如 ggcorrplot 完善。

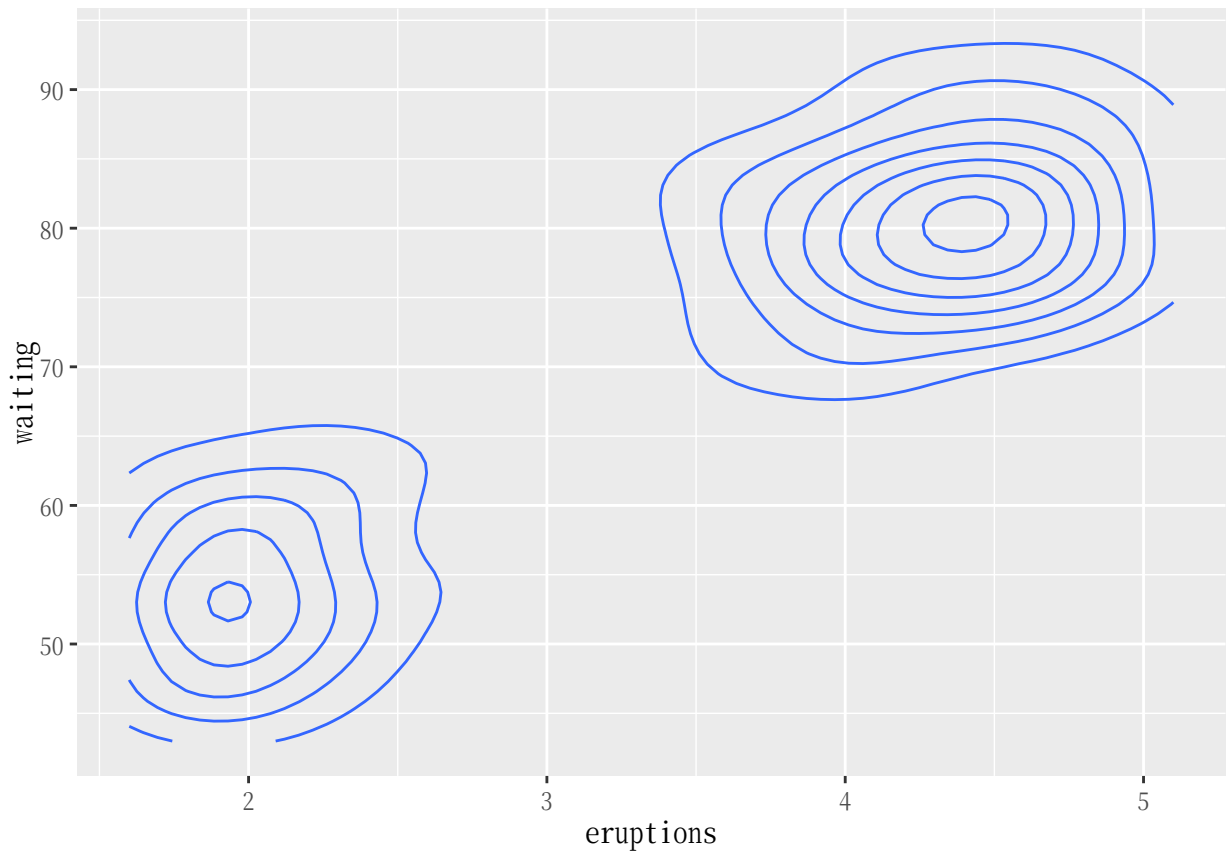
corrgram 包提供了 corrgram 函数作相关系数矩阵图，见26.6.1。

28.5.4 等值线图和热力图

对于已经估计的二元密度曲面，可以用 `geom_contour()` 作等值线图，用 `geom_raster()` 作密度热力图。比如，`faithfuld` 是 Faithful 喷泉的间隔与喷发时间的二维密度估计数据框，作密度的等值线图：

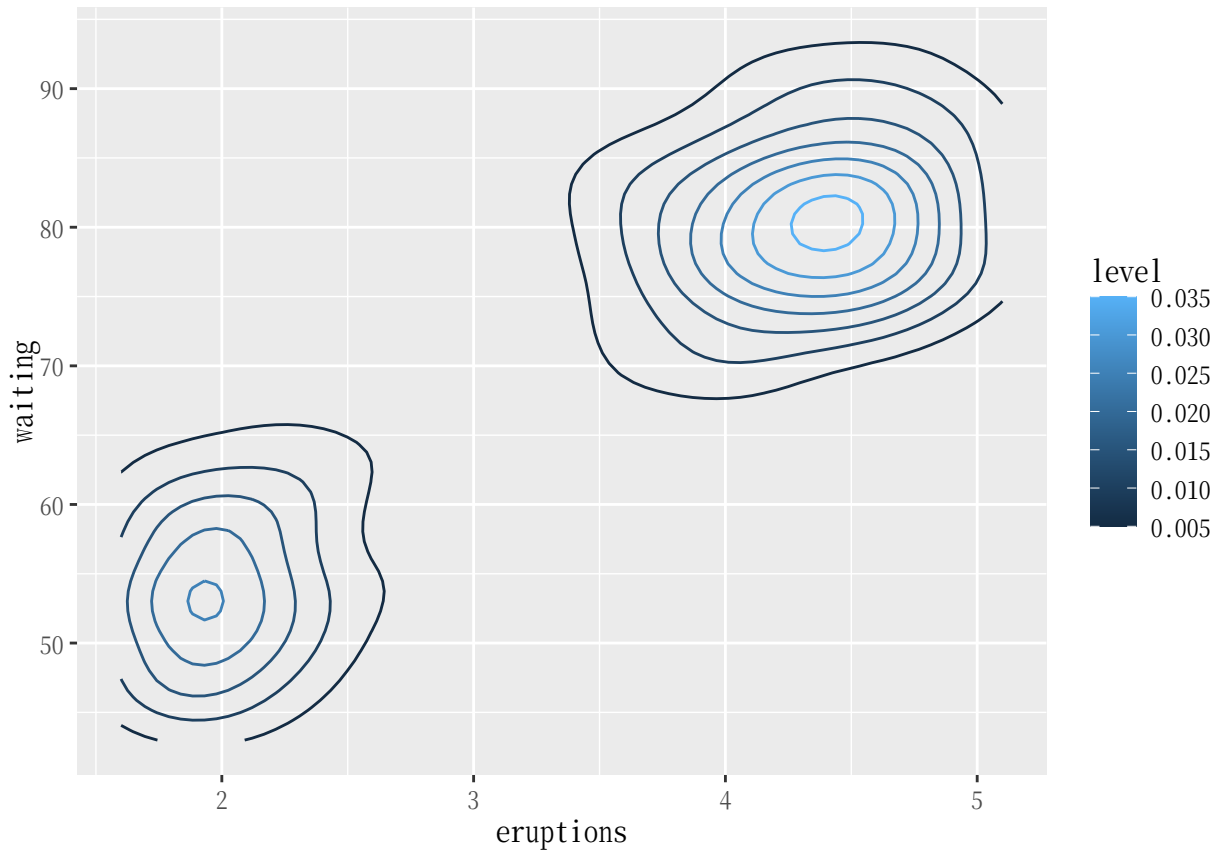
```
p <- ggplot(faithfuld, aes(
  x = eruptions, y = waiting))
```

```
p + geom_contour(mapping = aes(
  z = density))
```



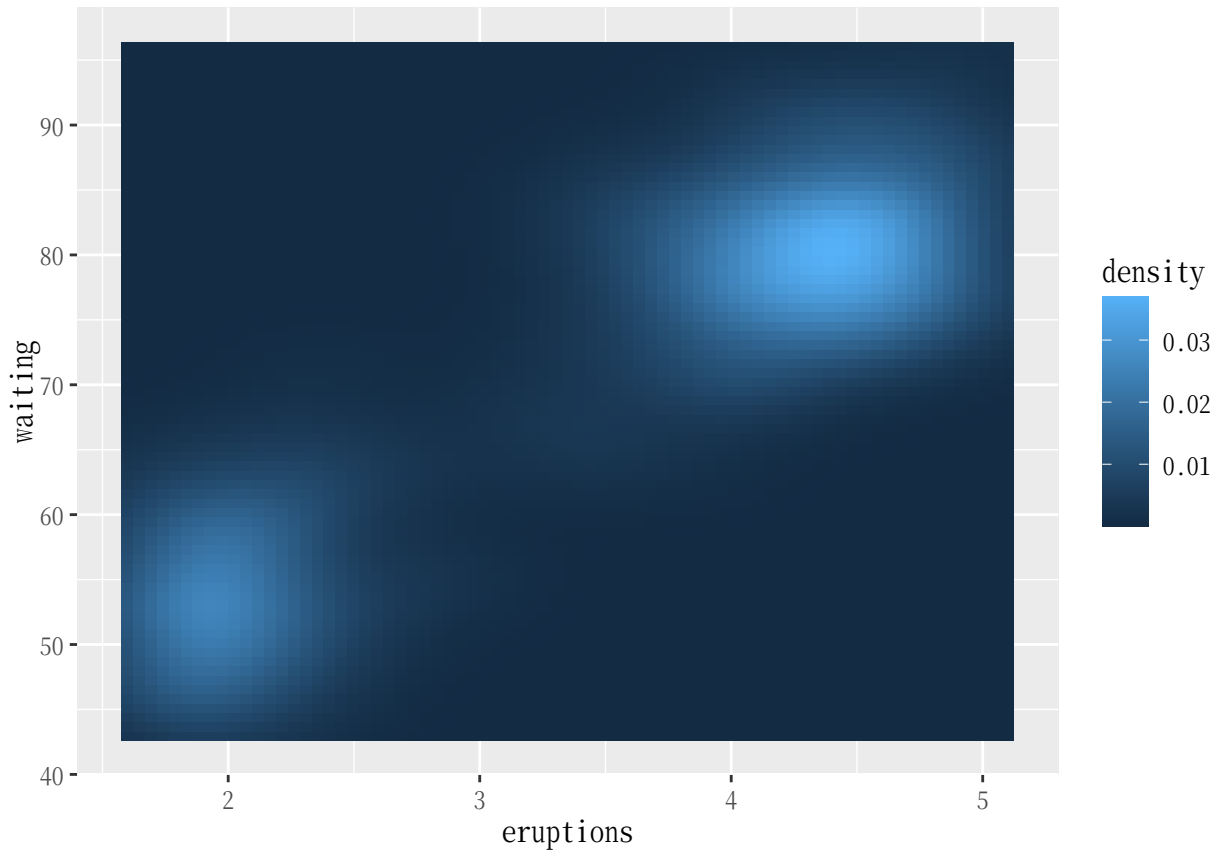
可以用等值线颜色代表密度值大小：

```
p + geom_contour(mapping = aes(
  z = density, color = ..level..))
```



作密度估计的热力图：

```
p + geom_raster(mapping = aes(
  fill = density))
```



28.5.5 数据降维

当数据中有过多的变量时，比如变量达到数百、数千时，即使是散点图矩阵都信息量过大难以认读。这时常用主成分分析等降维方法将数据降维到若干个新变量，对新变量作图。

主成分分析利用变量的协方差阵或者相关阵的特征值分解对原始变量进行线性组合，产生若干个新变量。当原始变量为相同单位且可比时，可以基于协方差阵，否则应该基于相关阵。例如，对 iris 数据集的 4 个测量值作主成分分析：

```
pca1 <- princomp(iris[,1:4], cor=FALSE)
summary(pca1, loadings=TRUE)
```

```
## Importance of components:
##                           Comp.1     Comp.2     Comp.3      Comp.4
## Standard deviation     2.0494032 0.49097143 0.27872586 0.153870700
## Proportion of Variance 0.9246187 0.05306648 0.01710261 0.005212184
## Cumulative Proportion  0.9246187 0.97768521 0.99478782 1.000000000
##
## Loadings:
```

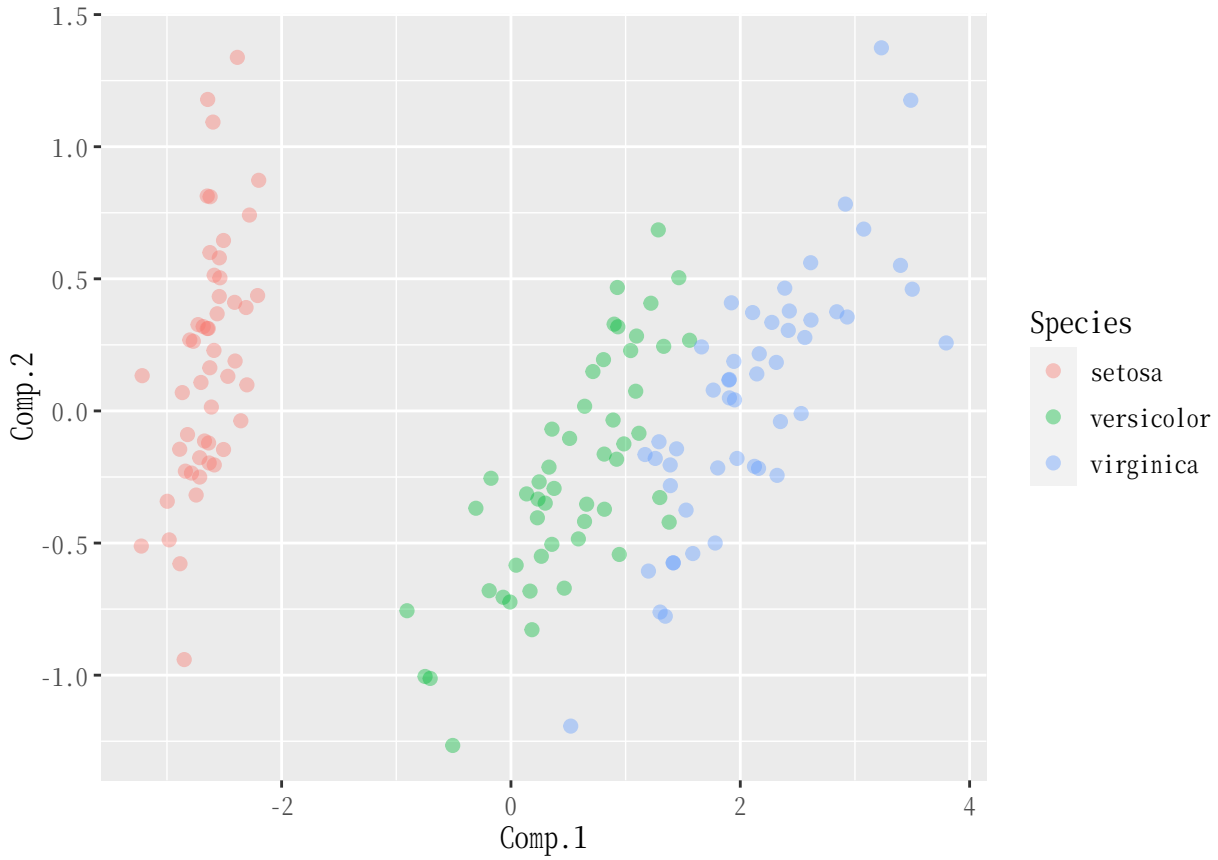
```
##              Comp.1 Comp.2 Comp.3 Comp.4
## Sepal.Length  0.361  0.657  0.582  0.315
## Sepal.Width          0.730 -0.598 -0.320
## Petal.Length  0.857 -0.173        -0.480
## Petal.Width   0.358        -0.546  0.754
```

```
ps1 <- predict(pca1) |> as.data.frame()
ps1 <- cbind(iris, ps1)
```

前两个主成分解释了原始变量中 98% 的方差。

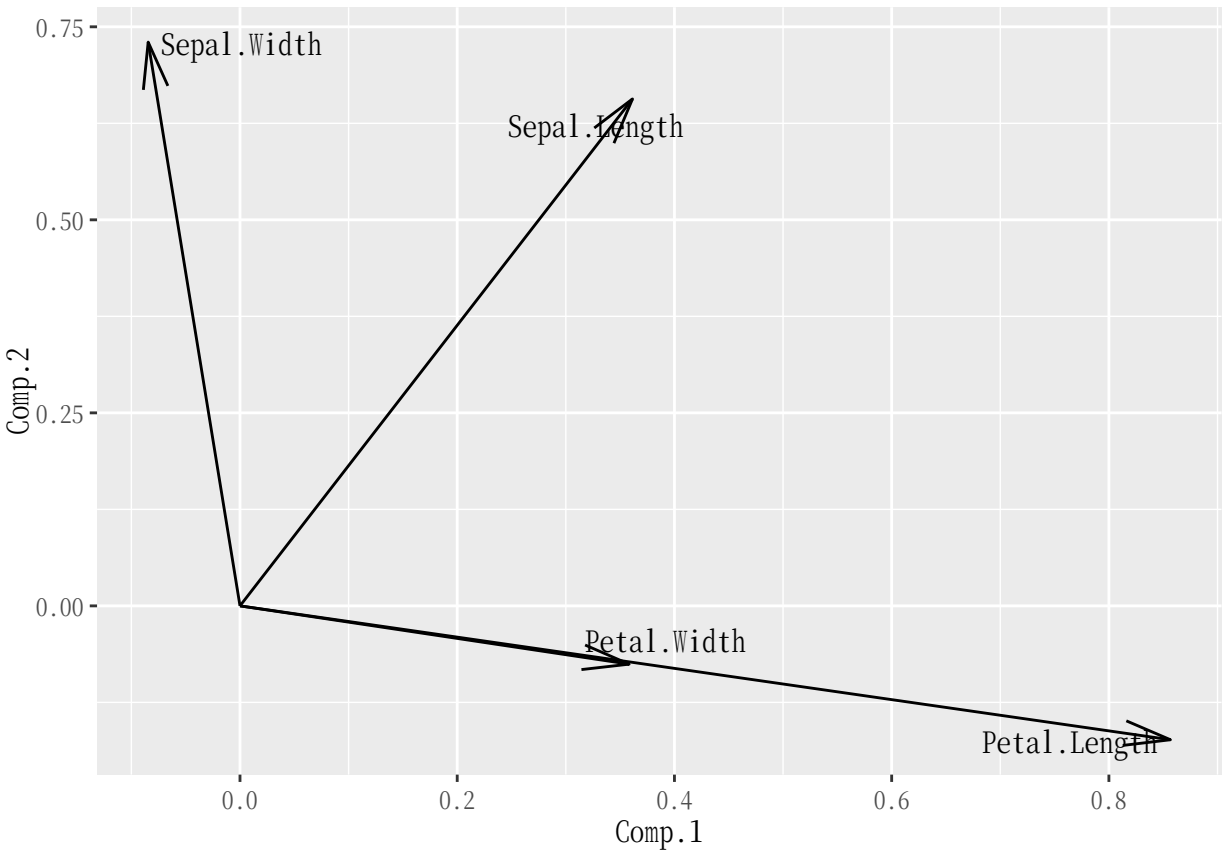
作第一和第二主成分的散点图：

```
ggplot(as.data.frame(ps1), aes(
  x = Comp.1, y = Comp.2, color = Species)) +
  geom_point(size = 2.0, alpha = 0.4)
```



主成分分析取前两个主成分在各原始变量上的载荷可以做矢量图，用来表示各个原始变量与主成分的关系：

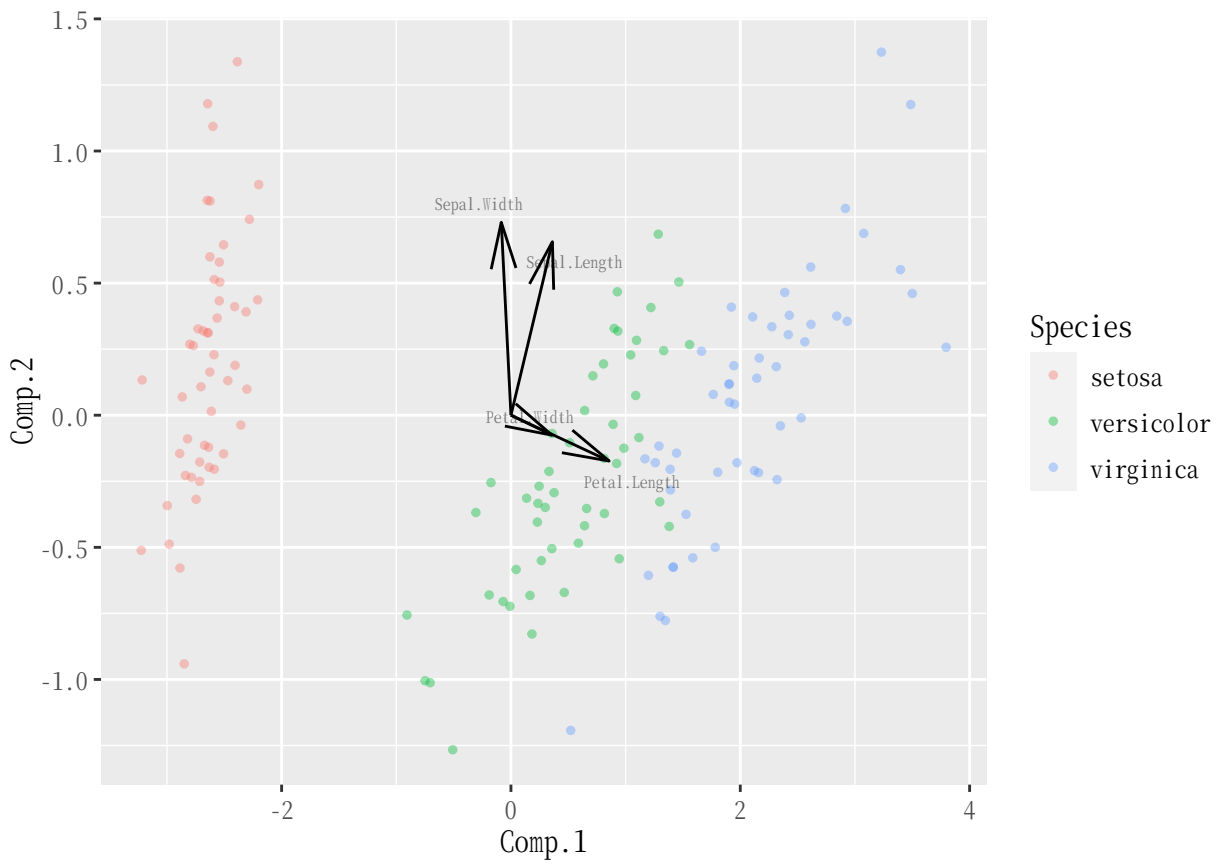
```
d <- as.data.frame(loadings(pca1)[,1:2])
d$vlabel <- rownames(d)
ggplot(d, aes(
  x = Comp.1, y = Comp.2, label = vlabel)) +
  geom_segment(mapping = aes(
    xend = Comp.1, yend = Comp.2),
    x = 0, y = 0,
    arrow = arrow(angle = 15)) +
  geom_text_repel()
```



可以将降维的散点图与变量的载荷图画在同一坐标系内：

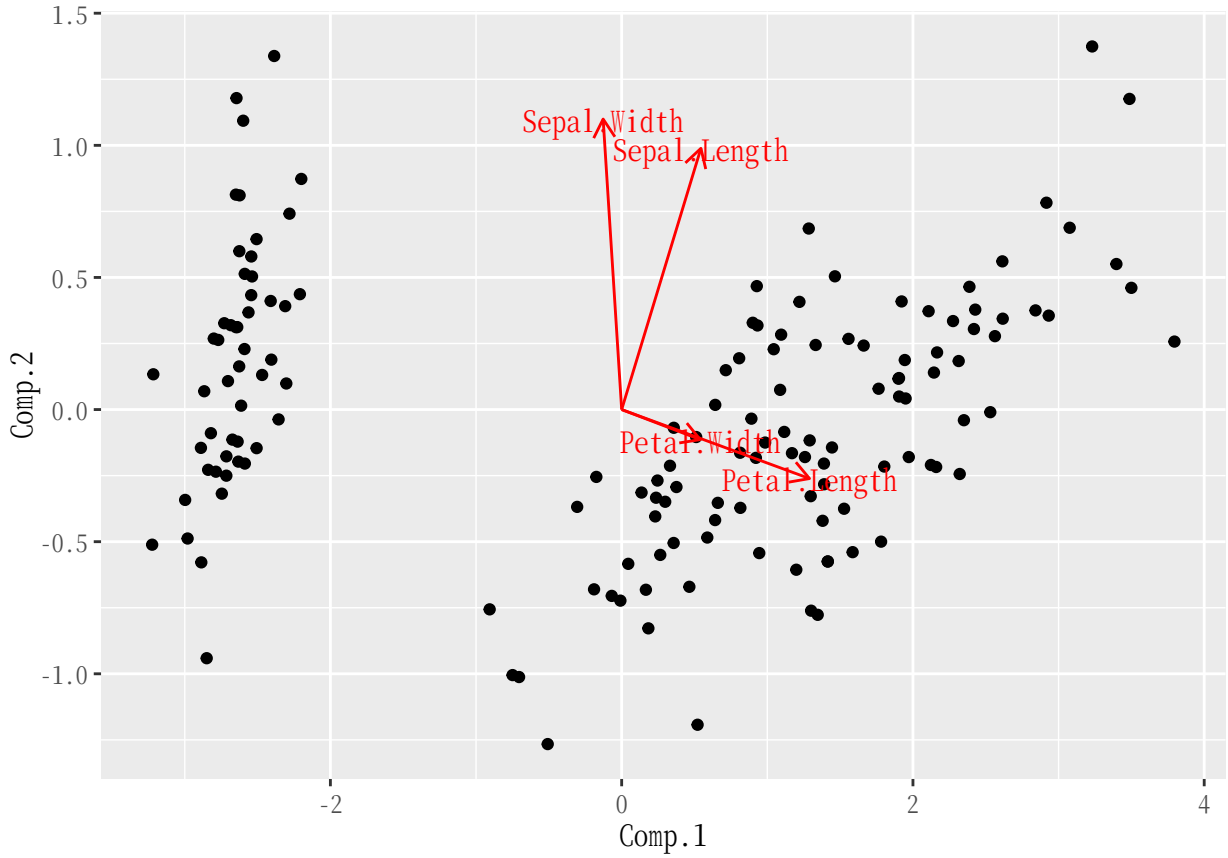
```
ggplot() +
  geom_point(data = ps1, mapping = aes(
    x = Comp.1, y = Comp.2, color = Species),
    size = 1.0, alpha = 0.4) +
  geom_segment(data = d, mapping = aes(
    xend = Comp.1, yend = Comp.2),
    x = 0, y = 0,
```

```
    arrow = arrow(angle = 15)) +
  geom_text_repel(data = d, mapping = aes(
    x = Comp.1, y = Comp.2, label = vlabel),
    alpha = 0.5, size = 2.0)
```



可以直接使用 ggfortify 包作图：

```
library(ggfortify)
dggd <- ps1[,c("Comp.1", "Comp.2")]
dggl <- as.data.frame(loadings(pca1)[,1:2])
dggl[["varnames"]] <- rownames(loadings(pca1))
ggbiplot(
  dggd, dggl,
  loadings=TRUE,
  loadings.label=TRUE,
  loadings.label.label="varnames")
```

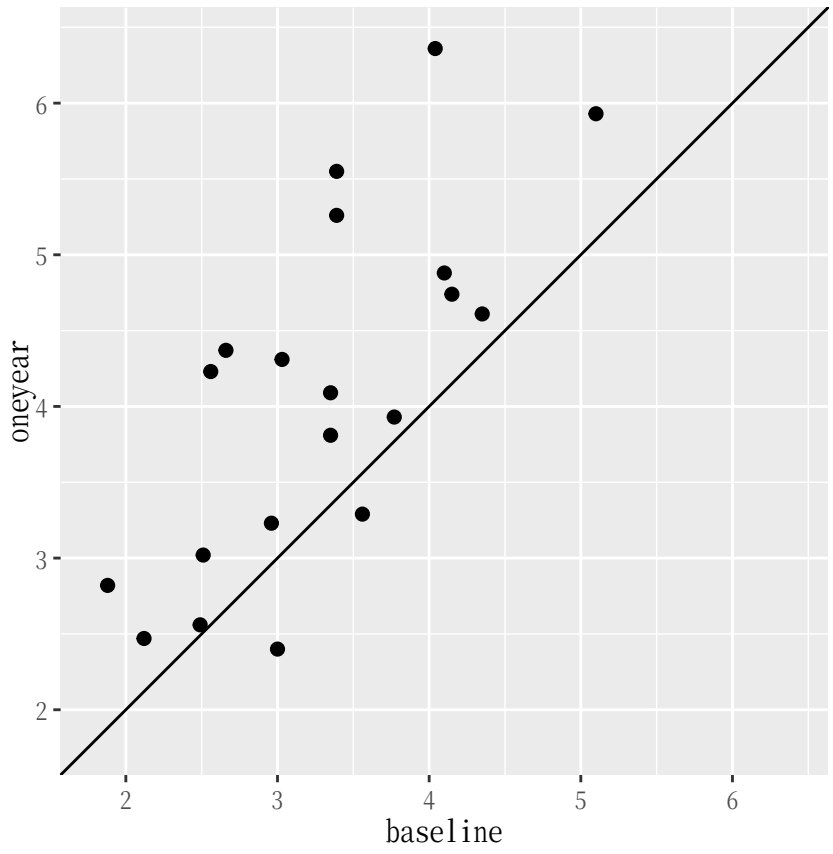


28.5.6 成对数据

成对数据指同一个变量在两个不同时间的测量值，或者同一个体两个取值基本相同的变量。作散点图时，增加 $y = x$ 直线是显然的一个增强显示。另外，还应该强调两个变量之间的差别。

考虑 boot 扩展包的 cd4 数据集，这个数据集包含了 20 名临床试验受试者在开始时和一年以后的 cd4 指标测量值。作这两个指标的散点图：

```
data(cd4, package = "boot")
ggplot(cd4, aes(
  x = baseline, y = oneyear)) +
  geom_point(size = 2.0) +
  geom_abline(slope = 1, intercept = 0) +
  coord_fixed(
    xlim = c(1.80, 6.40), ylim = c(1.80, 6.40))
```

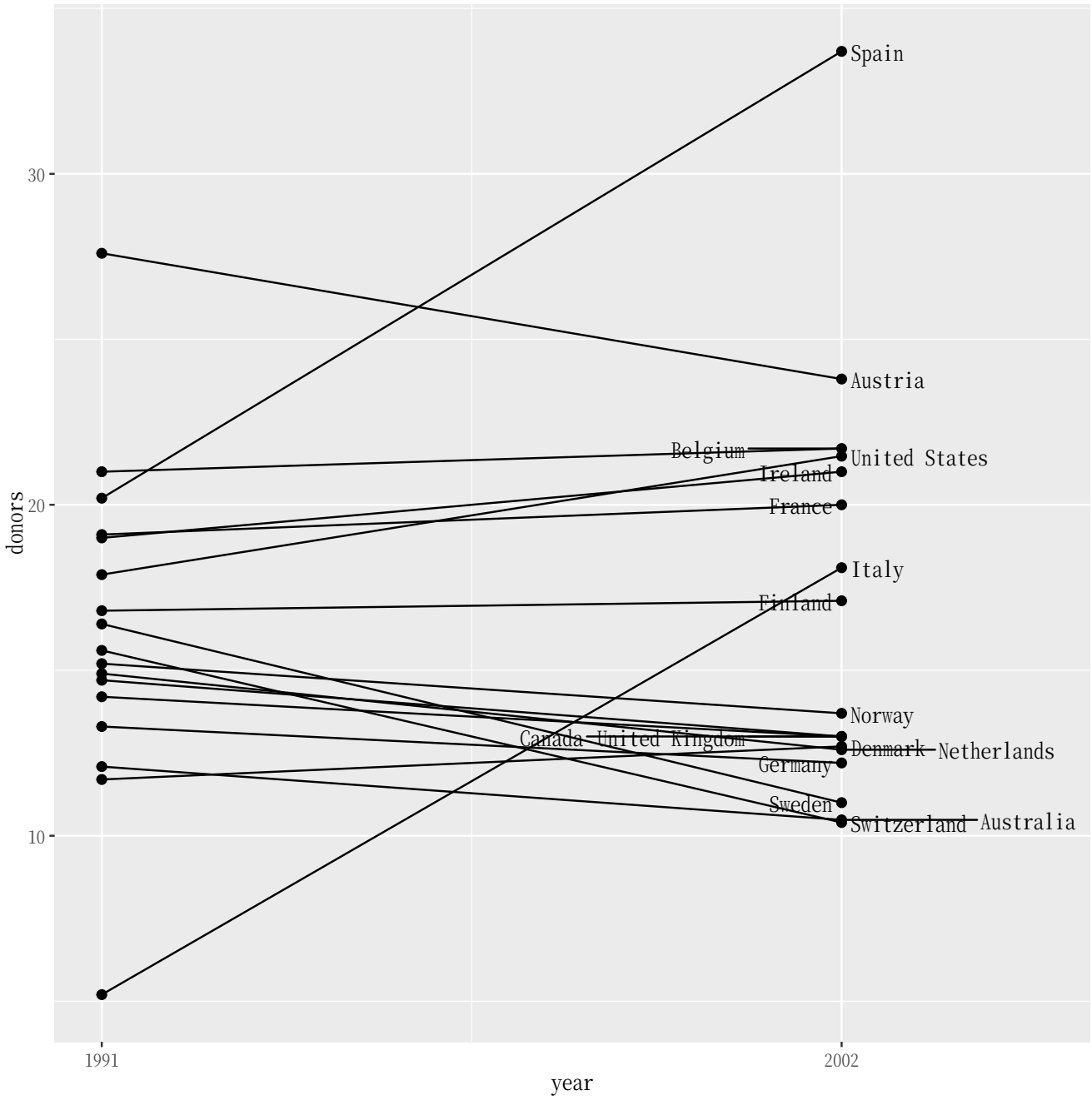


程序中用了 `geom_abline()` 作斜线，用了 `coord_fixed()` 指定 1:1 的宽高比并设定 x 轴和 y 轴相同的坐标范围。可以看出一年后的测量值普遍高于开始时的测量值。

当观测个数较少时，可以用线段连接每个观测的两个变量值，并标出每个观测的标签。

```
data(organdata, package = "socviz")
organdata |>
  select(country, year, donors) |>
  mutate(year = lubridate::year(year)) |>
  filter(year %in% c(1991, 2002)) -> dorgans
ggplot(dorgans, aes(
  x = year, y = donors)) +
  geom_line(aes(group = country)) +
  geom_point(size = 2.0) +
  geom_text_repel(
    data = filter(dorgans, year == 2002),
    mapping = aes(
      x = year, y = donors,
      label = country),
    direction = "x") +
```

```
  scale_x_continuous(
    breaks = c(1991, 2002),
    limits = c(1991, 2005),
    labels = c("1991", "2002"))
```

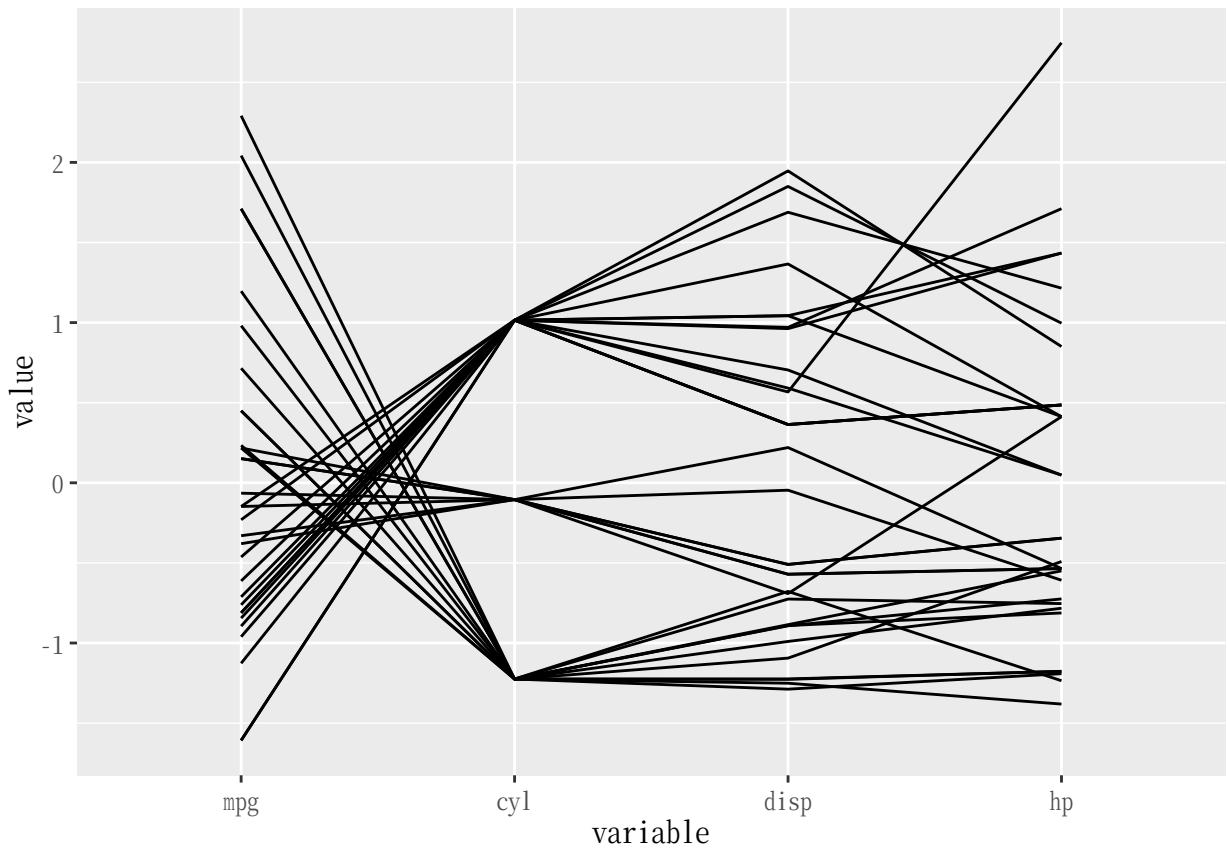


可以看出，这种图可以表现超过两个变量。

28.5.7 平行坐标图

GGally 包的 ggparcoord 函数提供了平行坐标图。

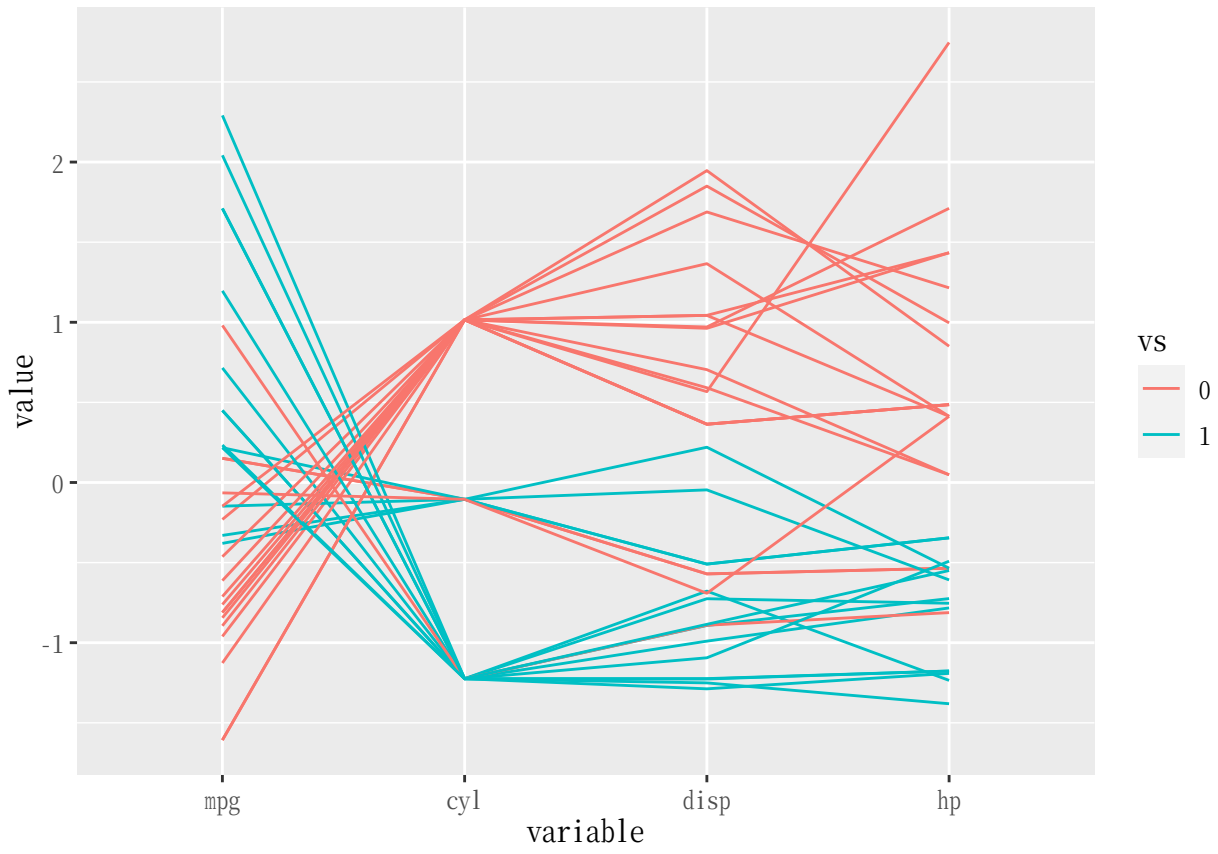
```
library(GGally)
ggparcoord(data = mtcars, columns = c(1:4))
```



相邻的两个变量之间的连线平行提示正相关，交叉提示负相关。因为仅能表现两个相邻变量的相关性，这种图需要适当调整输入的列次序。

以某个变量值分类添加颜色：

```
d <- mtcars |> mutate(vs = factor(vs))
ggparcoord(data = d, columns = c(1:4), groupColumn="vs")
```



28.5.8 雷达图

ggradar 包提供了雷达图功能（需要从 github 安装）。如：

```
library(ggradar)
library(dplyr)
library(scales)
library(tibble)

mtcars_radar <- mtcars |>
  as_tibble(rownames = "group") |>
  mutate_at(vars(-group), rescale) |>
  tail(4) |>
  select(1:10)

ggradar(mtcars_radar)
```

`ggradar` 函数要求仅输入需要作图的变量且预先进行 0-1 标准化，且第一列用作观测的标签。scale 包的 `rescale` 函数作 0-1 标准化。

28.6 时间序列图

某个指标随时间变化的数据称为时间序列，可以以时间为横坐标、该指标为纵坐标作折线图，称为时间序列图。其它有次序的变量也可以用作横轴变量。

`geom_line()` 函数会自动按照 `x` 轴变量的次序连线，即使数据中的 `x` 变量没有从小到大排列都是如此。`geom_path()` 则按照数据集中原有次序连线。

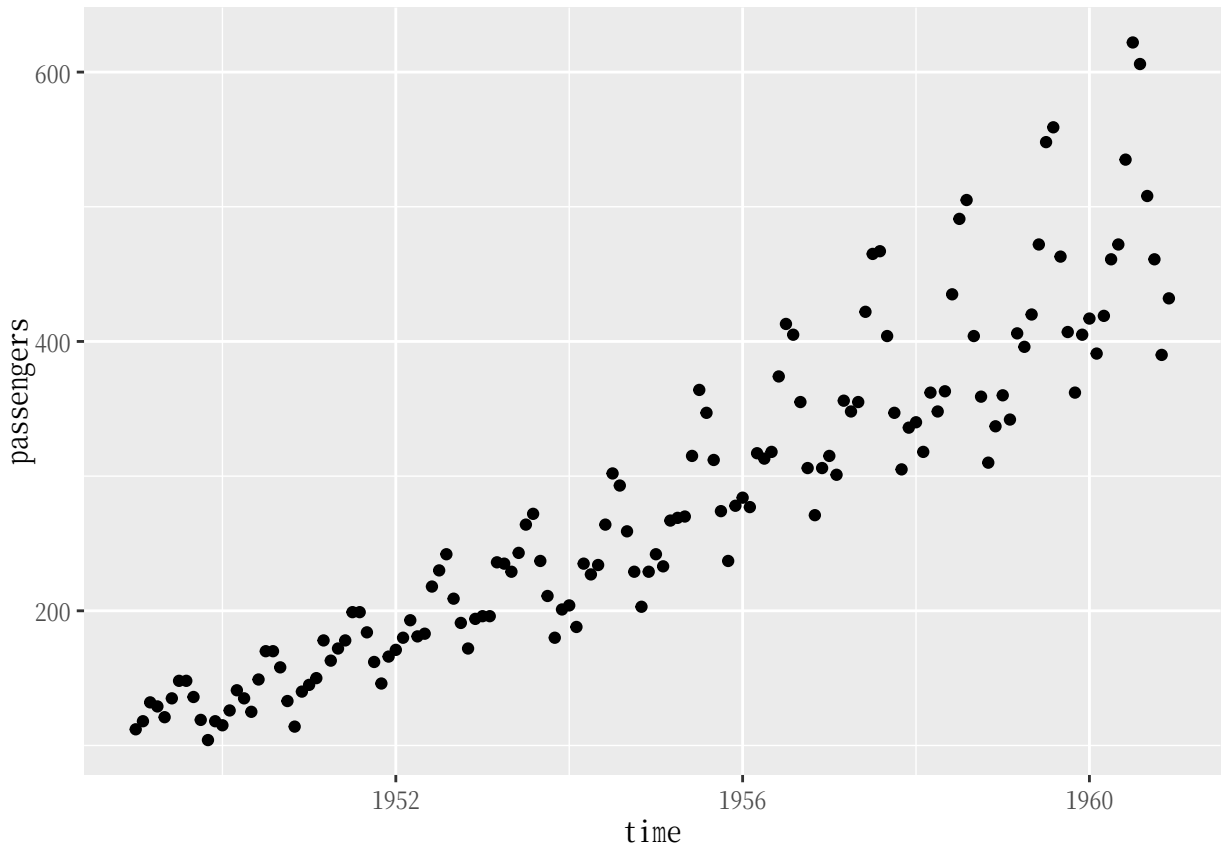
ggfortify 包提供了较好的时间序列图形功能。

28.6.1 一元时间序列

R 软件中提供了一个示例用的时间序列数据 AirPassengers，是美国泛美航空公司 1949-1960 的国际航班订票数的月度数据（单位：千人），12 年 144 个月的数据。

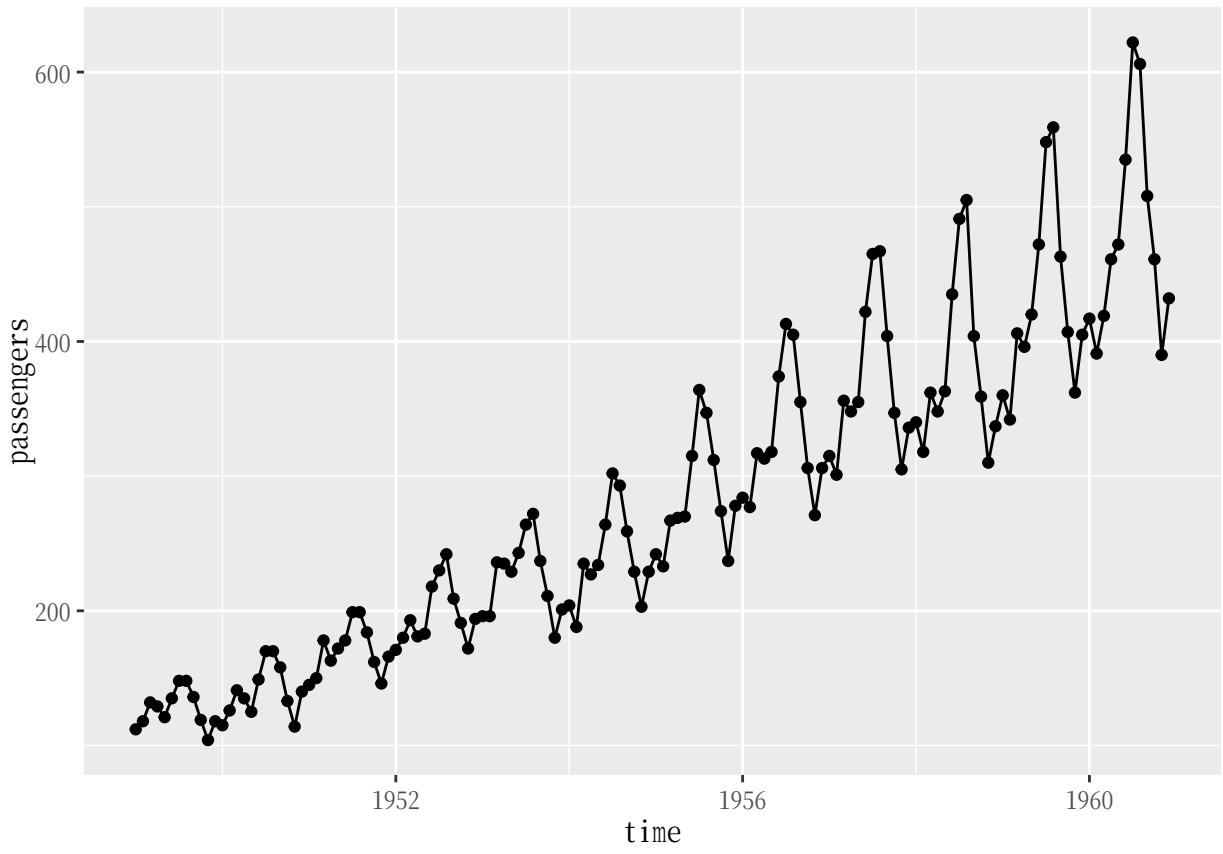
作其散点图：

```
dap <- AirPassengers |>
  (\(y) tibble(
    time = as.vector(time(y)),
    passengers = as.vector(y) ) )()
p <- ggplot(dap, aes(
  x = time, y = passengers))
p + geom_point()
```



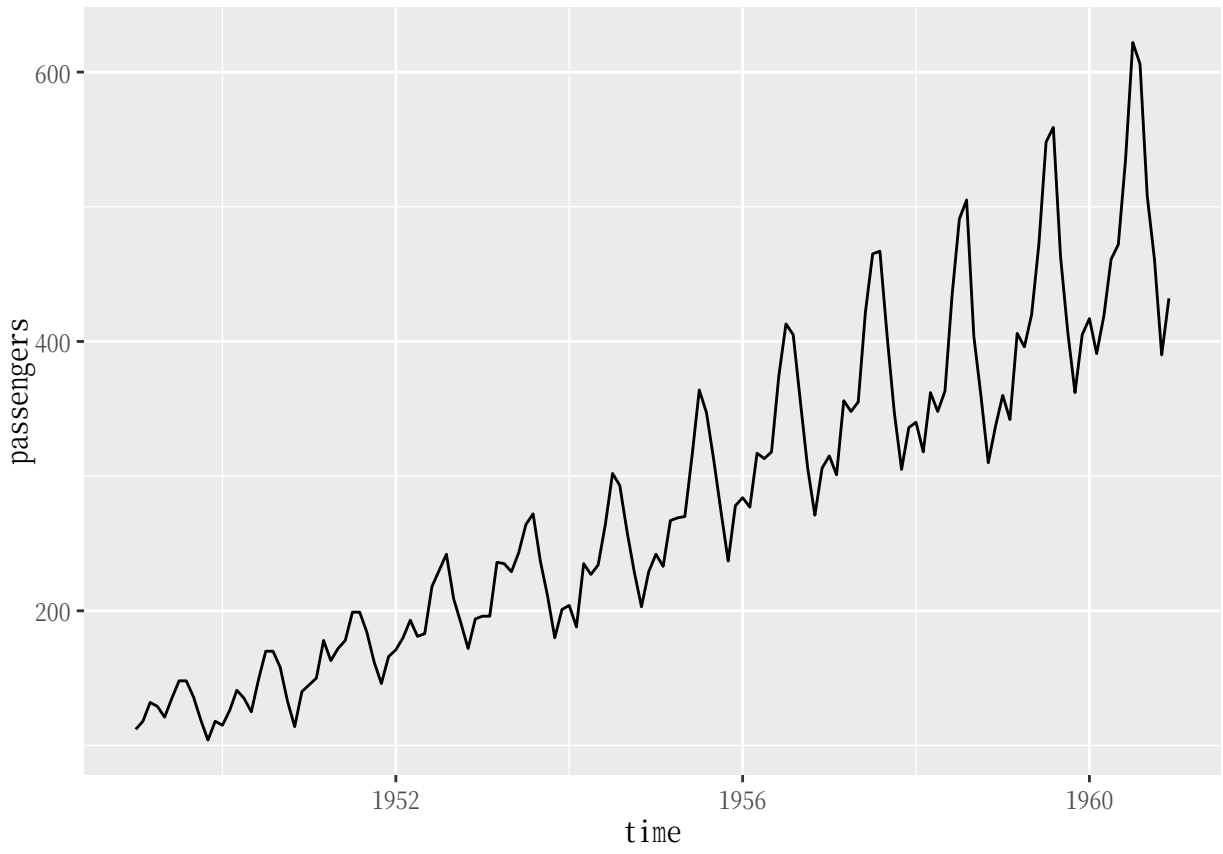
这样的图形在点数较少时还好，点数较多时不易分辨出局部随时间的变化情况。加上连线：

```
p + geom_point() +
  geom_line()
```



也可以仅连线，不画散点：

```
p + geom_line()
```



仅画线的时间序列图更强调变化趋势，不强调个别数据，当观测点很多时一般不画出散点。

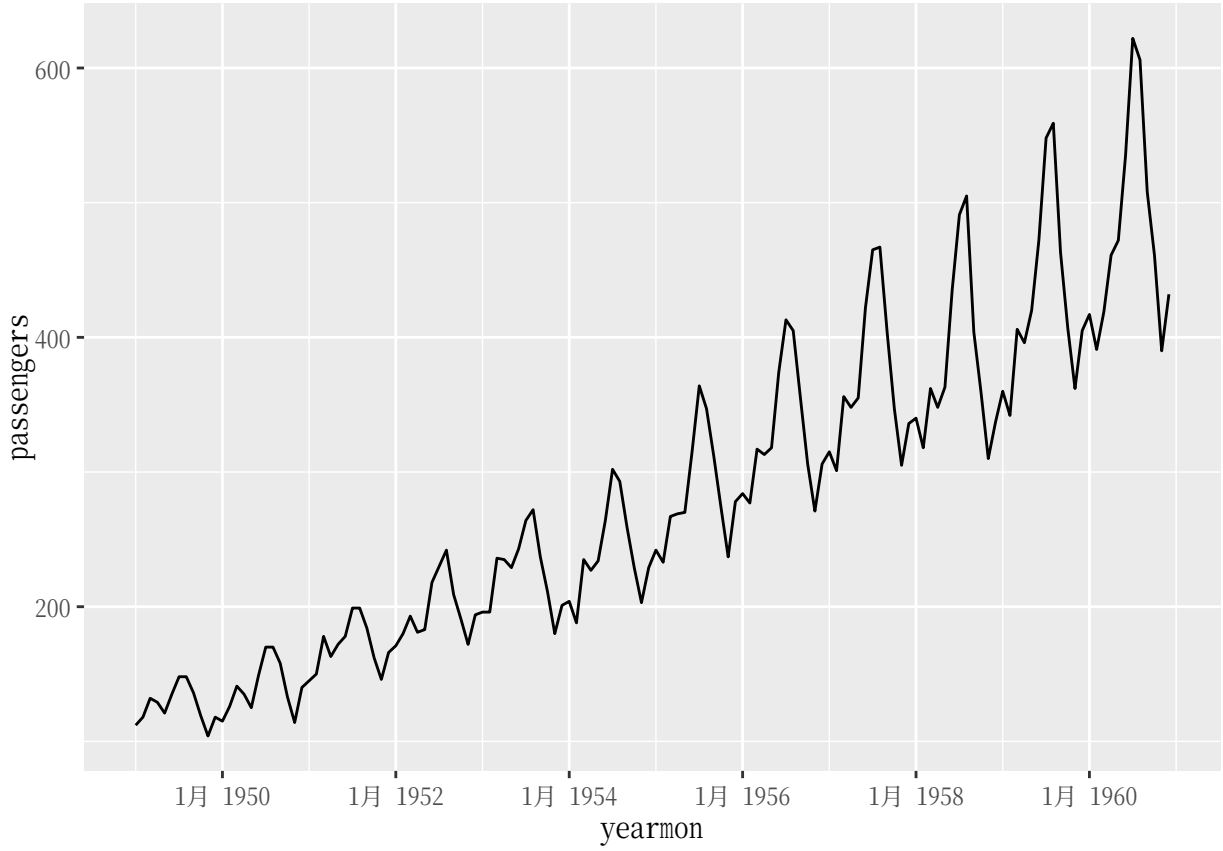
这里的时间是 `time()` 函数返回的以年为单位的浮点数值。时间序列的时间往往是日期值，尤其是日数据，可以用 `scale_x_date()` 函数标日期，如：

```
library(xts)
dap2 <- AirPassengers |>
  as.xts() |>
  (\(y) tibble(
    yearmon = index(y),
    time = as.Date(yearmon),
    passengers = c(coredata(y))))()
knitr::kable(head(dap2, 6))
```

yearmon	time	passengers
1 月 1949	1949-01-01	112
2 月 1949	1949-02-01	118
3 月 1949	1949-03-01	132
4 月 1949	1949-04-01	129
5 月 1949	1949-05-01	121
6 月 1949	1949-06-01	135

数据框 dap2 中的 yearmon 是年月格式的时间，time 是年月日格式的时间。先用年月格式的时间作图：

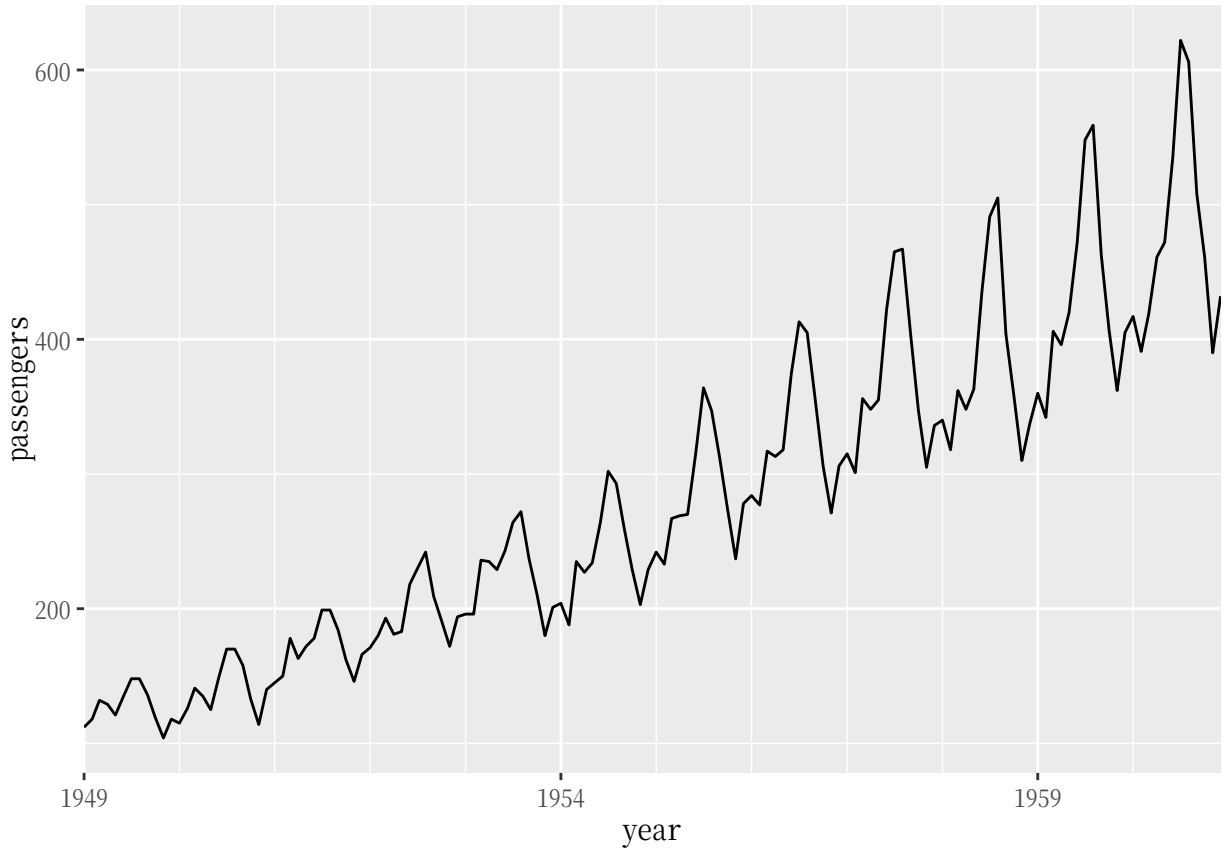
```
ggplot(dap2, aes(
  x = yearmon,
  y = passengers)) +
  geom_line()
```



ggplot2 不支持对年月类型的时间轴进行调整。为了自定义时间轴，需要使用日期格式的坐标轴，使用 dap2 数据框中的 time 列：

```
p <- ggplot(dap2, aes(
  x = time,
  y = passengers))
```

```
p + geom_line() +
  scale_x_date(
    name = "year",
    date_breaks = "5 years",
    date_labels = "%Y",
    date_minor_breaks = "1 year",
    expand = c(0, 0))
```

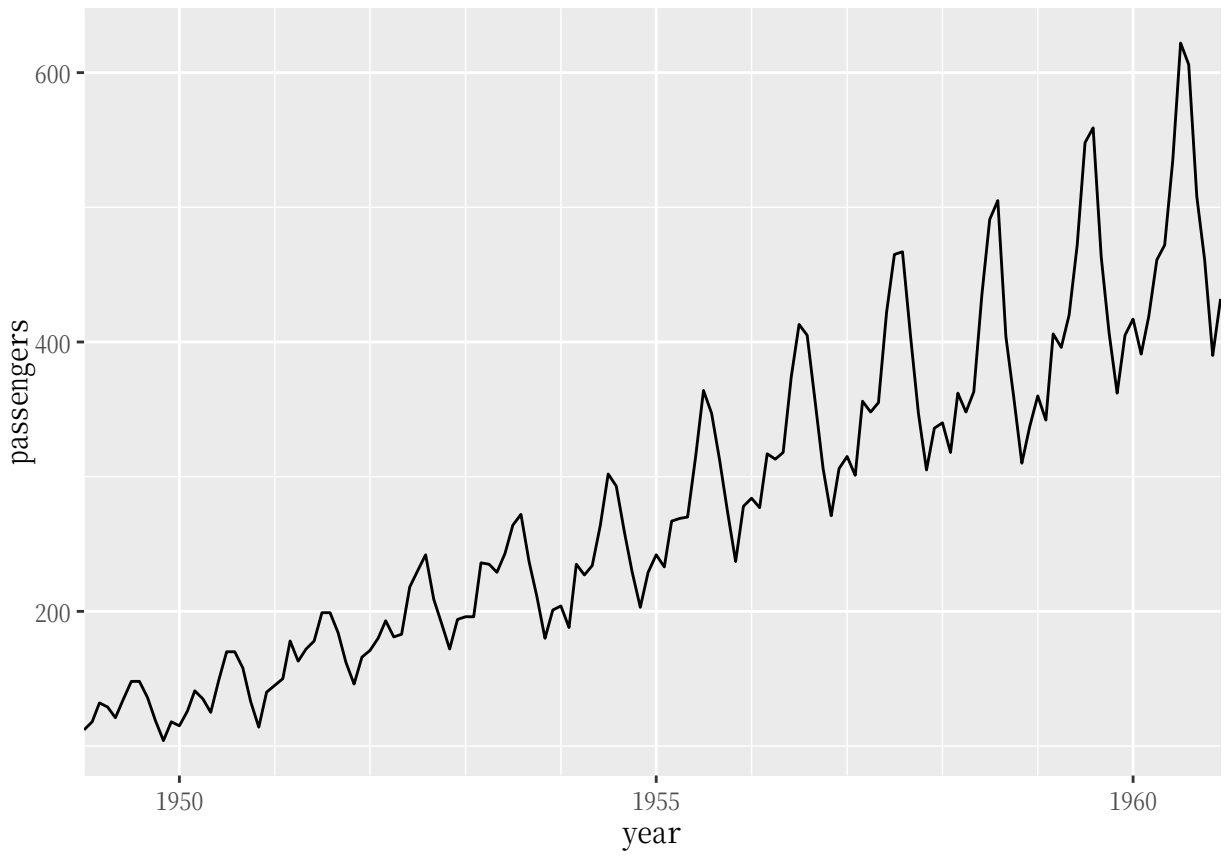


程序中的 time 变量是 Date 类型的日期时间，用 `date_breaks` 选项指定日期轴刻度值的频率，`"5 years"` 就是每 5 年一个刻度值。用 `date_minor_breaks` 选项指定细刻度的频率。`date_labels` 是日期刻度的一种格式规定，比如`"%Y"` 是四位数的年份，`"%Y-%m"` 是年月格式，等等。因为是月度数据，所以刻度线所在的数据点对应该年的 1 月份。

如果需要人为指定刻度位置，可以去掉 `date_breaks` 参数，改用 `breaks` 参数，如：

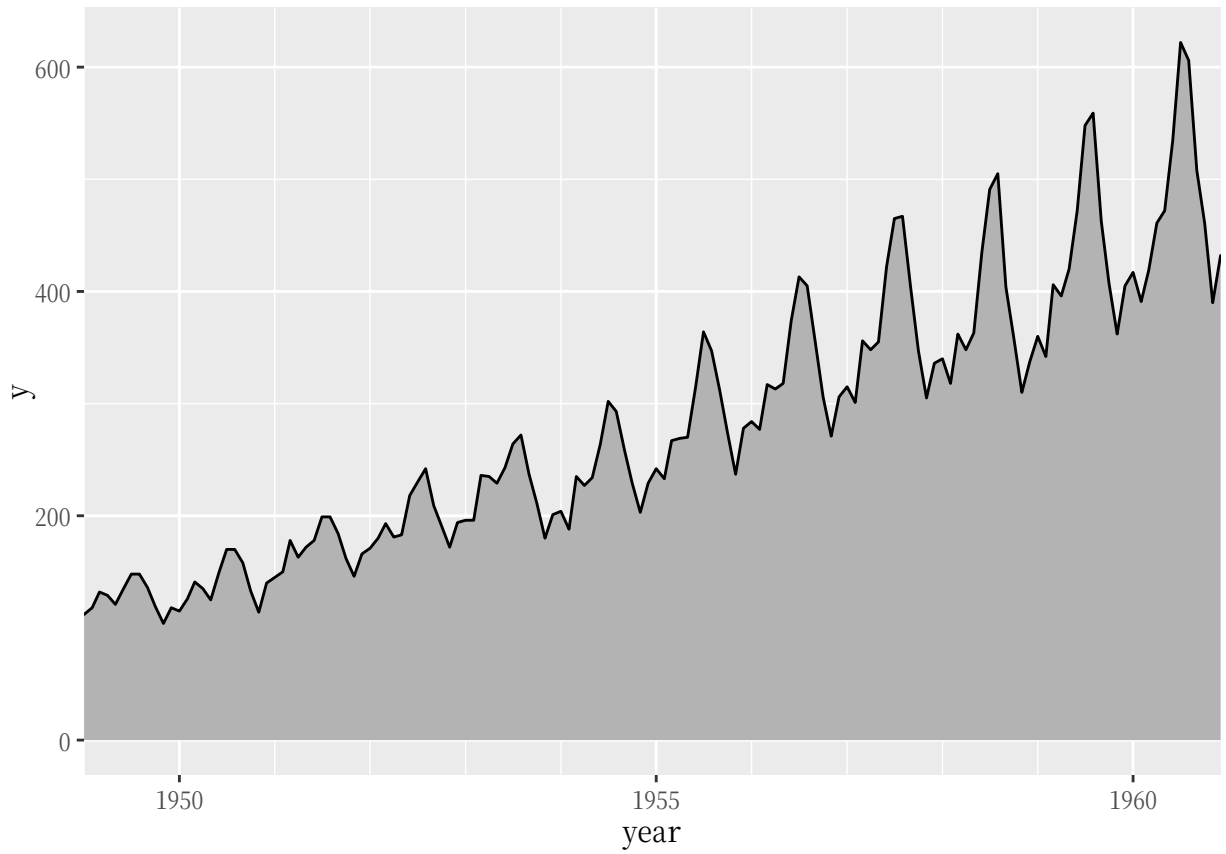
```
p + geom_line() +
  scale_x_date(
    name = "year",
    breaks = lubridate::make_date(c(1950, 1955, 1960)),
    date_labels = "%Y",
```

```
    date_minor_breaks = "1 year",
    expand = c(0, 0))
```



还可以在线下加阴影，但这时 y 轴应为正值变量且坐标范围从 0 开始，如：

```
library(ggridges)
ggplot(dap2, aes(
  x = as.Date(time),
  height = passengers,
  y = 0)) +
  geom_ridgeline() +
  scale_x_date(
    name = "year",
    breaks = lubridate::make_date(c(1950, 1955, 1960)),
    date_labels = "%Y",
    date_minor_breaks = "1 year",
    expand = c(0, 0))
```

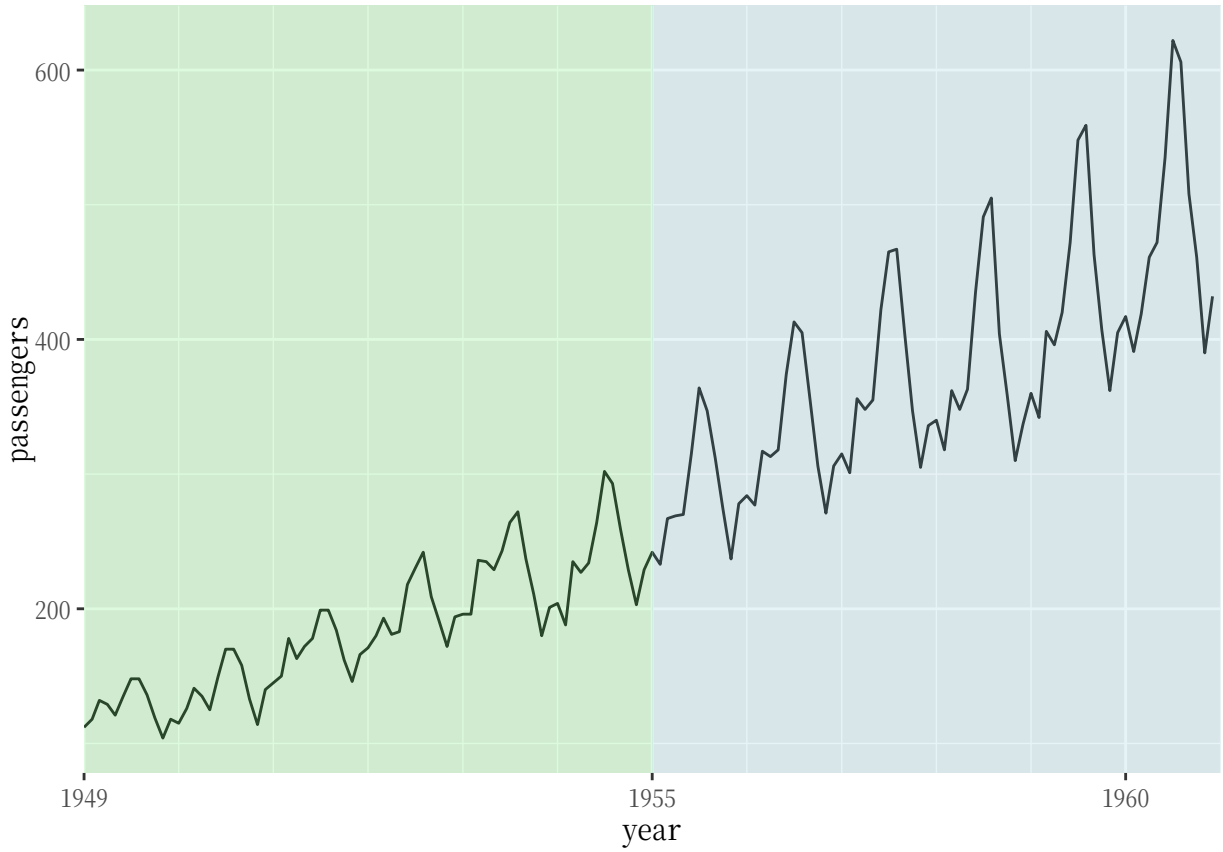


上面的程序用了 ggridges 包的 `geom_ridgeline()` 函数，y 轴对应于 `height` 维。

有时需要用不同阴影色标出不同的时间段，可以使用 `annotate()` 函数，指定 `geom` 为`"rect"`，需要 `xmin`, `xmax`, `ymin`, `ymax` 参数，可用用 alpha 指定一定透明度，用 `fill` 指定填充颜色。例如，将时间分为 1949-1954, 1955-1960 两段，用不同颜色填充：

```
ggplot(dap2, aes(
  x = time,
  y = passengers)) +
  geom_line() +
  annotate(
    geom = "rect",
    ymin = -Inf, ymax = Inf,
    xmin = lubridate::make_date(c(1949, 1955)),
    xmax = lubridate::make_date(c(1955, 1961)),
    fill = c("lightgreen", "lightblue"), alpha = 0.3) +
  scale_x_date(
    name = "year",
    breaks = lubridate::make_date(c(1949, 1955, 1960)),
    date_labels = "%Y",
```

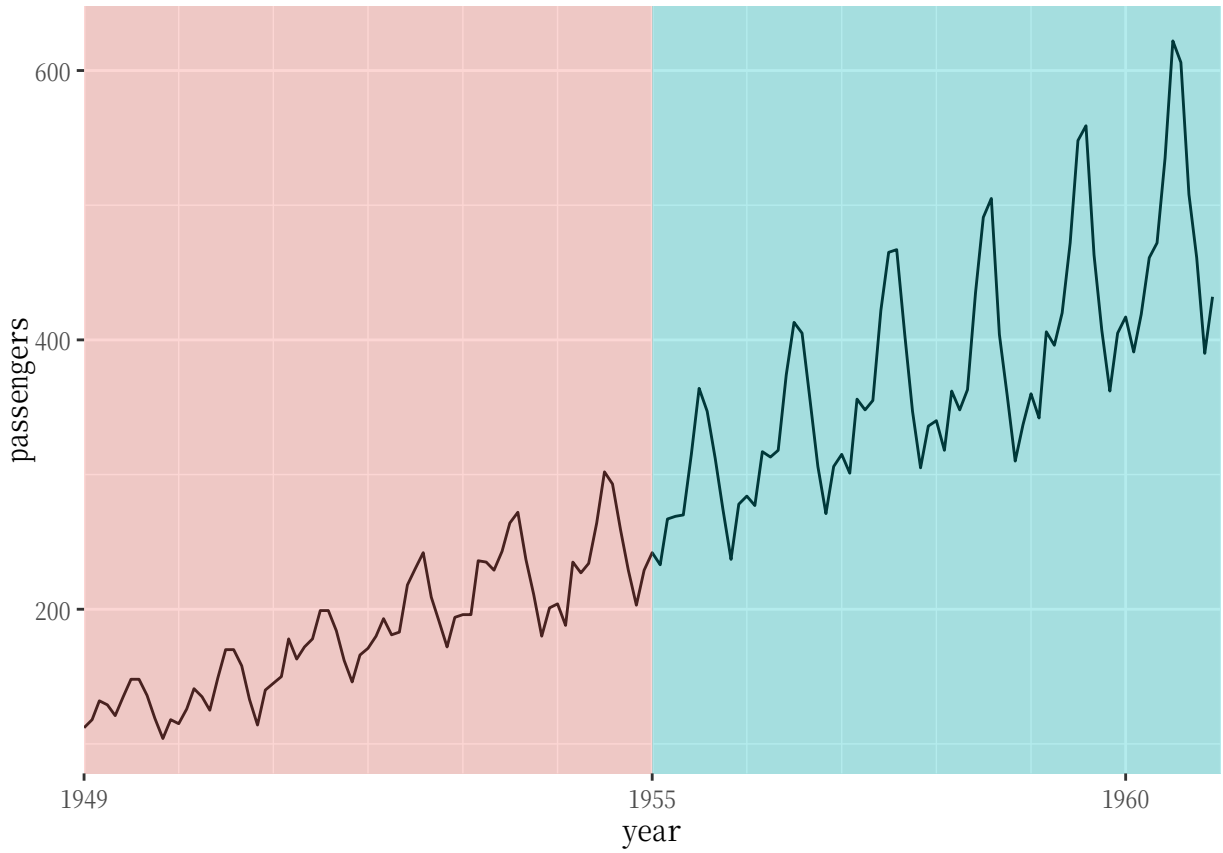
```
    date_minor_breaks = "1 year",
    expand = c(0, 0)) +
  guides(fill = "none")
```



分时间段染色的另一种办法是使用 `geom_rect()` 函数，这时需要有另一个输入数据框，填充色可以自动映射，画曲线与染色的函数要分别输入数据内容：

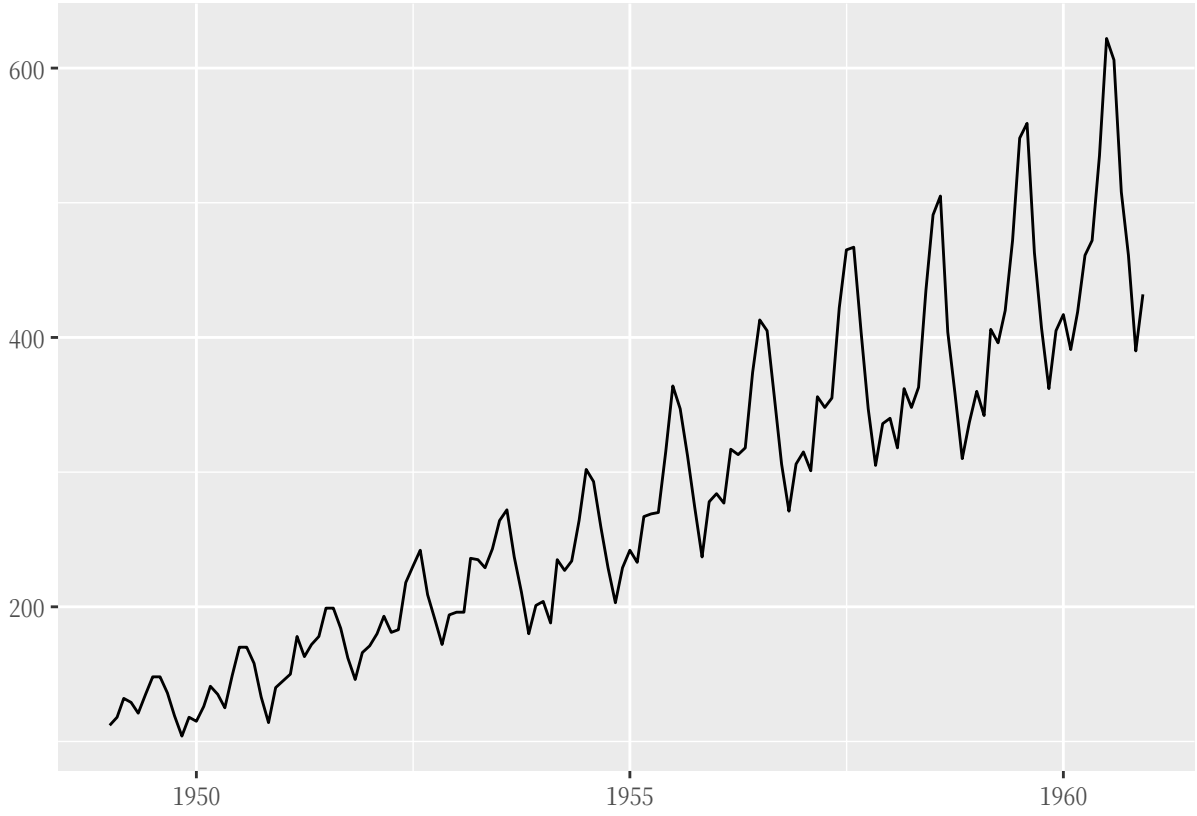
```
dap2b <- data.frame(
  ymin = -Inf, ymax = Inf,
  xmin = lubridate::make_date(c(1949, 1955)),
  xmax = lubridate::make_date(c(1955, 1961)),
  period = c("First", "Second"))
ggplot() +
  geom_line(
    data = dap2,
    mapping = aes(
      x = time, y = passengers)) +
  geom_rect(
    data = dap2b,
    mapping = aes(
```

```
      xmin = xmin, xmax = xmax,
      ymin = ymin, ymax = ymax,
      fill = period ),
    alpha = 0.3) +
  scale_x_date(
    name = "year",
    breaks = lubridate::make_date(c(1949, 1955, 1960)),
    date_labels = "%Y",
    date_minor_breaks = "1 year",
    expand = c(0, 0)) +
  guides(fill = "none")
```



ggfortify 的简单时间序列图：

```
data("AirPassengers")
library(ggfortify)
autoplot(AirPassengers)
```



可以增加选项 `ts.colour` 指定曲线颜色，`ts.linetype` 指定线型。

下面作一个日数据的时间序列图。用 quantmod 包从雅虎金融下载标普 500 指数的日数据：

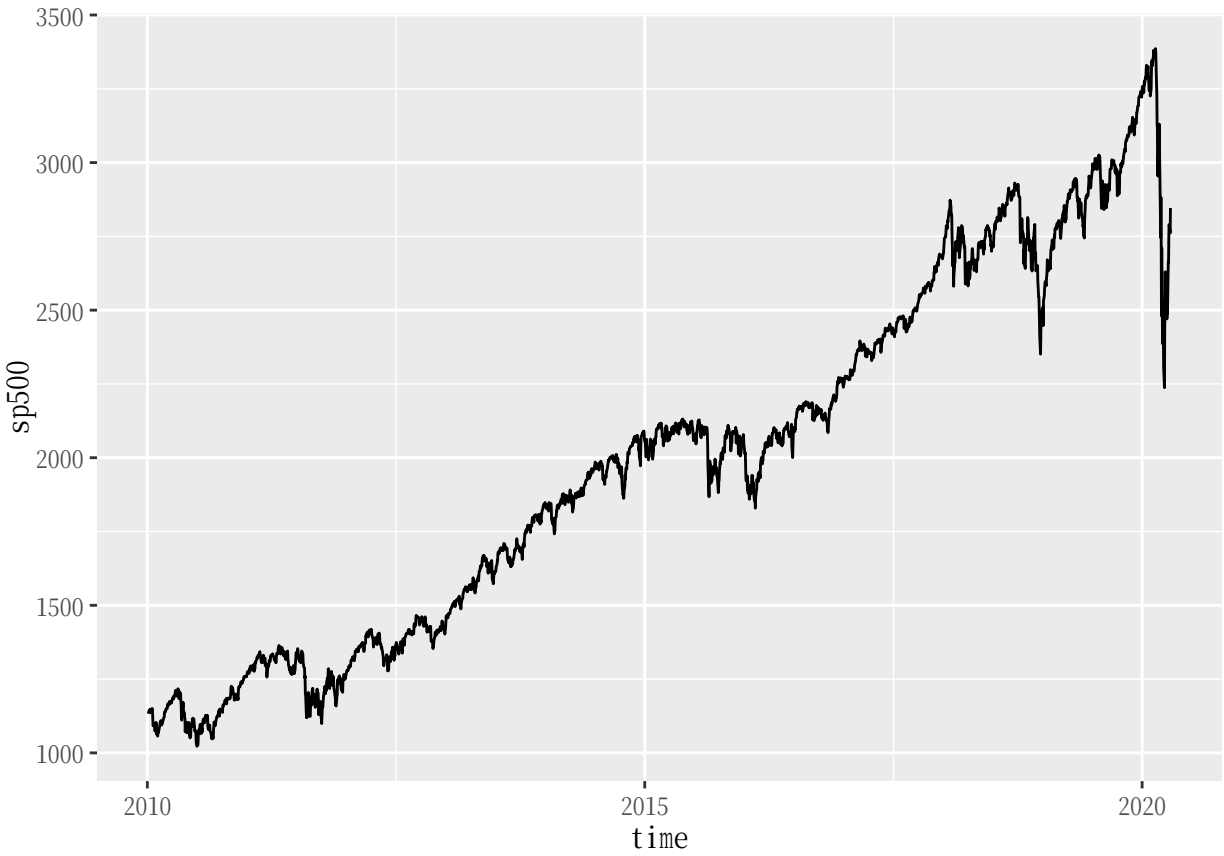
```
library(quantmod)
sp <- getSymbols(
  "^GSPC", src = "yahoo", auto.assign = FALSE
)
save(sp, file="sp500.RData")
```

```
##            GSPC.Open GSPC.High GSPC.Low GSPC.Close GSPC.Volume GSPC.Adjusted
## 2007-01-03   1418.03   1429.42  1407.86    1416.60  3429160000       1416.60
## 2007-01-04   1416.60   1421.84  1408.43    1418.34  3004460000       1418.34
## 2007-01-05   1418.34   1418.34  1405.75    1409.71  2919400000       1409.71
## 2007-01-08   1409.26   1414.98  1403.97    1412.84  2763340000       1412.84
## 2007-01-09   1412.84   1415.61  1405.42    1412.11  3038380000       1412.11
## 2007-01-10   1408.70   1415.99  1405.32    1414.85  2764660000       1414.85
```

quantmod 包也能做时间序列图，还可以做 K 线图、叠加各种技术曲线的图形。

用 ggplot 作标普 500 近 10 年的日收盘价时间序列图，不对坐标轴作调整：

```
dsp <- sp["2010/"] |>
  (\(y) tibble(
    time = index(y),
    sp500 = coredata(y)[,"GSPC.Close"]))()
ggplot(dsp, aes(
  x = time, y = sp500)) +
  geom_line()
```



刻度每年：

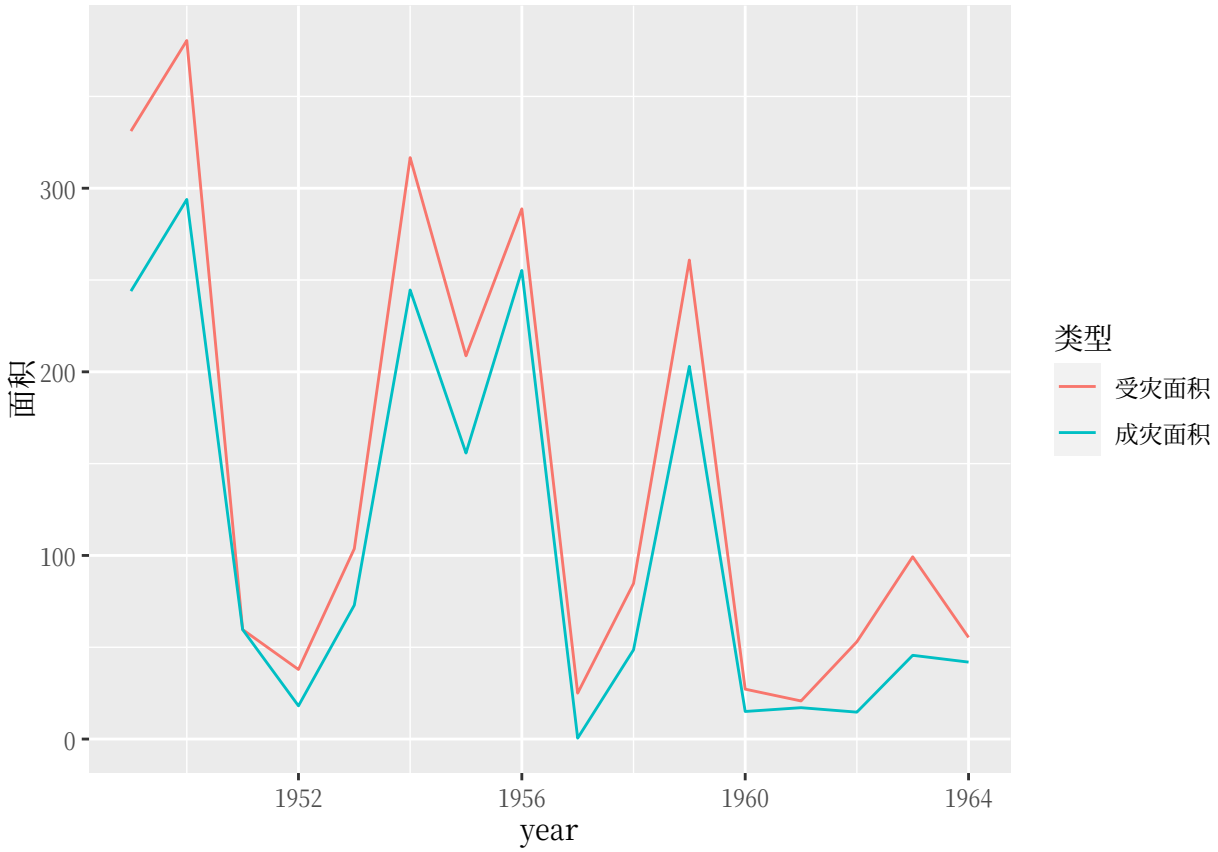
```
ggplot(dsp, aes(
  x = time, y = sp500)) +
  geom_line() +
  scale_x_date(
    name = NULL,
    date_breaks = "1 year",
    date_labels = "%Y",
    date_minor_breaks = "1 year")
```

28.6.2 多元时间序列

随时间记录的指标可以有多个。当这些指标的取值范围相近时，可以共用一个 y 轴，每个指标单独做连线并画在同一坐标系中。要注意的是，ggplot 作多元时间序列图时需要长表数据而非宽表数据，纵坐标变量必须保存在数据集的一个变量中，用另外的变量区分不同的指标。例如，北京地区从 1949 年到 1964 年的受灾面积与成灾面积（单位：万亩）年数据。

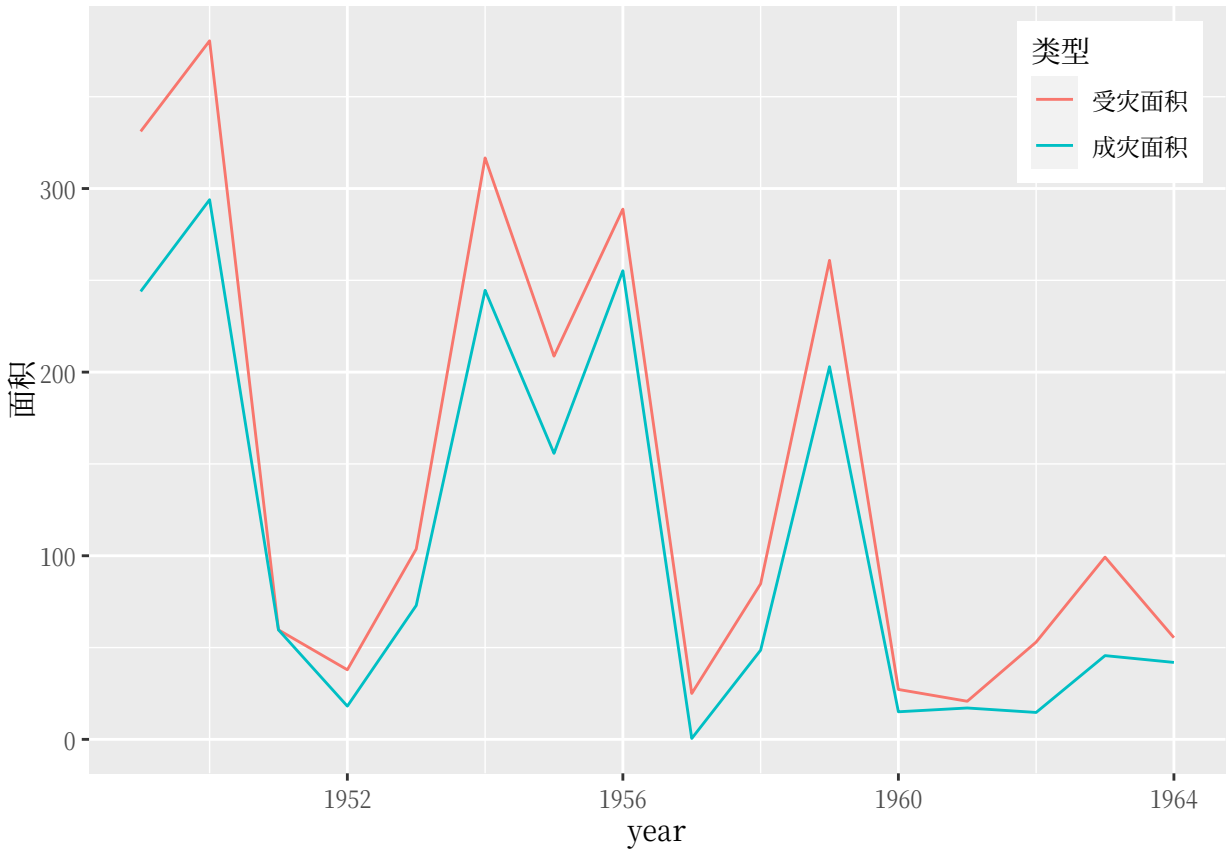
```
dflood <- as.data.frame(matrix(c(
1949 , 1 , 331.12 , 243.96 ,
1950 , 2 , 380.44 , 293.90 ,
1951 , 3 , 59.63 , 59.63,
1952 , 4 , 37.89 , 18.09,
1953 , 5 , 103.66 ,72.92,
1954 , 6 , 316.67 , 244.57,
1955 , 7 , 208.72 , 155.77,
1956 , 8 , 288.79 , 255.22,
1957 , 9 , 25.00 , 0.50,
1958 , 10 , 84.72 , 48.59,
1959 , 11 , 260.89 ,202.96,
```

```
1960, 12 , 27.18 ,15.02,
1961, 13 , 20.74 ,17.09,
1962 , 14 , 52.99 ,14.66,
1963 , 15 , 99.25 , 45.61,
1964 , 16 , 55.36 ,41.90),
byrow=TRUE, ncol=4,
dimnames=list(1949:1964,
c("year", "t", "area1", "area2"))))
dflood2 <- dflood |>
  pivot_longer(
    c(area1, area2),
    names_to = " 类型",
    values_to = " 面积") |>
  mutate(`类型` = factor(
    `类型`, levels = c("area1", "area2"),
    labels = c(" 受灾面积", " 成灾面积")))
p <- ggplot(dflood2, aes(
  x = year, y = `面积`, color = `类型`))
p + geom_line()
```



可以用 guides(label.position) 指定"top", "bottom", "left", "right" 这样的图例位置，还可以用 theme() 函数将图例指定在坐标系内部：

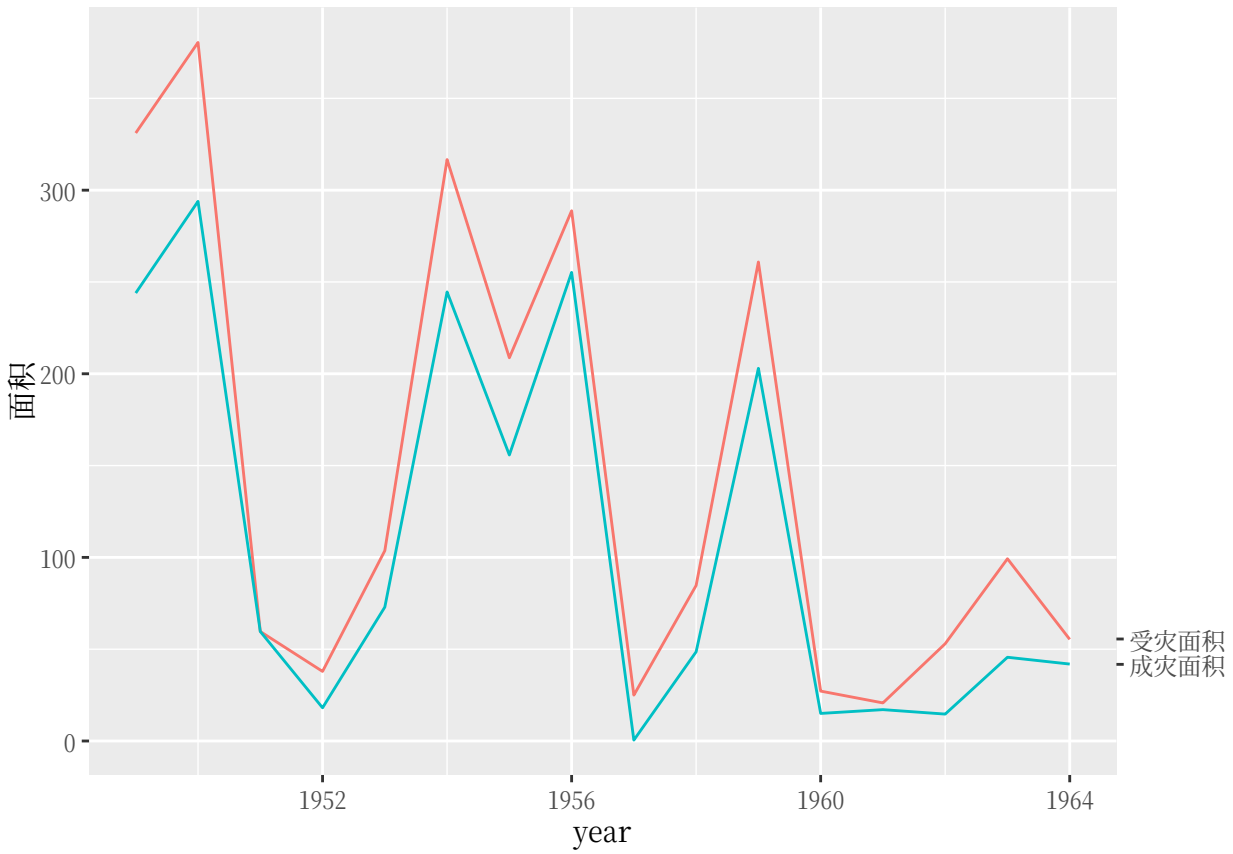
```
p + geom_line() +
  theme(
    legend.position = c(0.98, 0.98),
    legend.justification = c(1,1)  )
```



theme() 函数中 legend.position 给出的是图例框在坐标系内的百分比位置，legend.justification 给出的是图例框坐标的百分比对齐位置，c(1,1) 是将图例框的右上角对齐到给定百分比坐标位置。

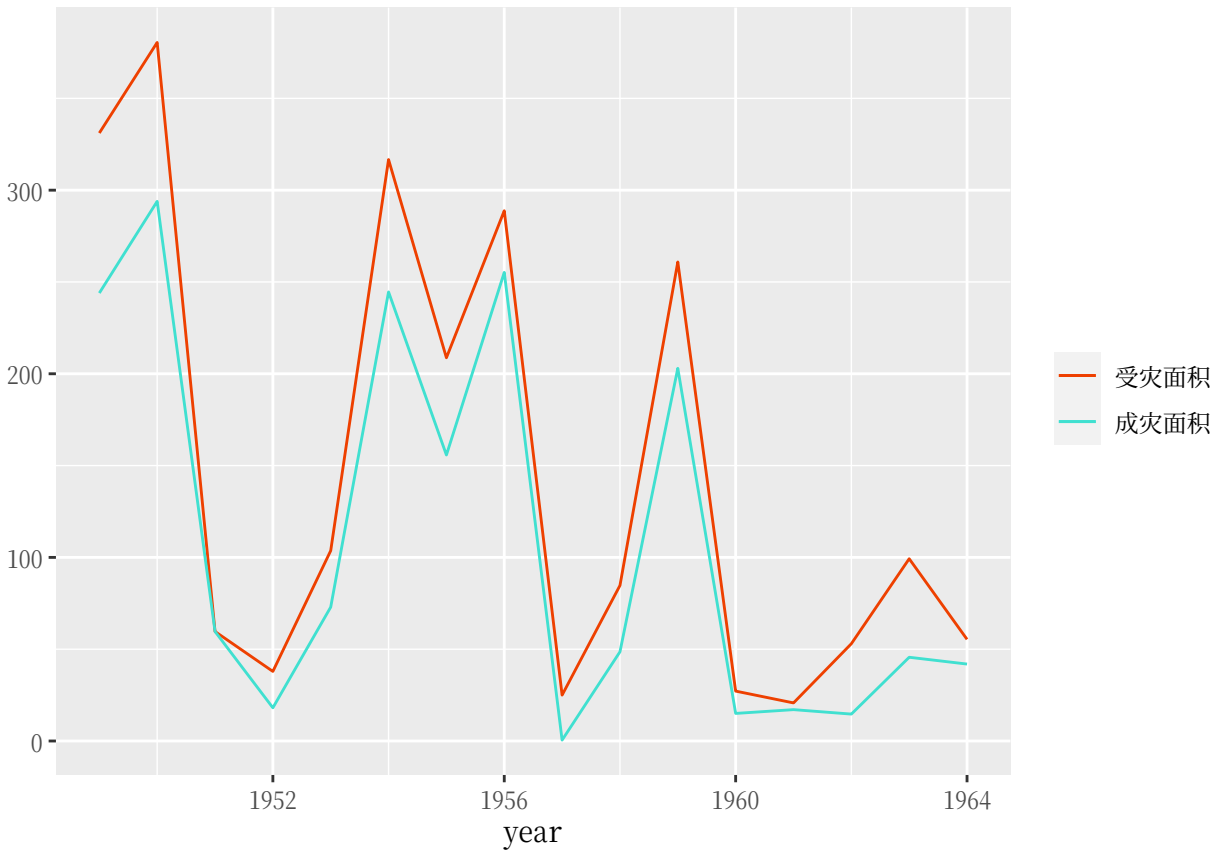
也可以去掉图例，而是用 y 轴的第二坐标轴（画在右侧）的办法直接标出两个序列的标签。如：

```
p + geom_line() +
  guides(color = "none") +
  scale_y_continuous(
    sec.axis = dup_axis(
      name = NULL,
      breaks = subset(dflood2, year == 1964)[[" 面积"]],
      labels = c(" 受灾面积", " 成灾面积")))
```



上面的做法需要将两条曲线的数据合并在一个变量中，并用一个分组变量区分两条曲线。如果数据原来在两列中，可以调用两次 `geom_line()`，但颜色和图例就需要人为指定了，如：

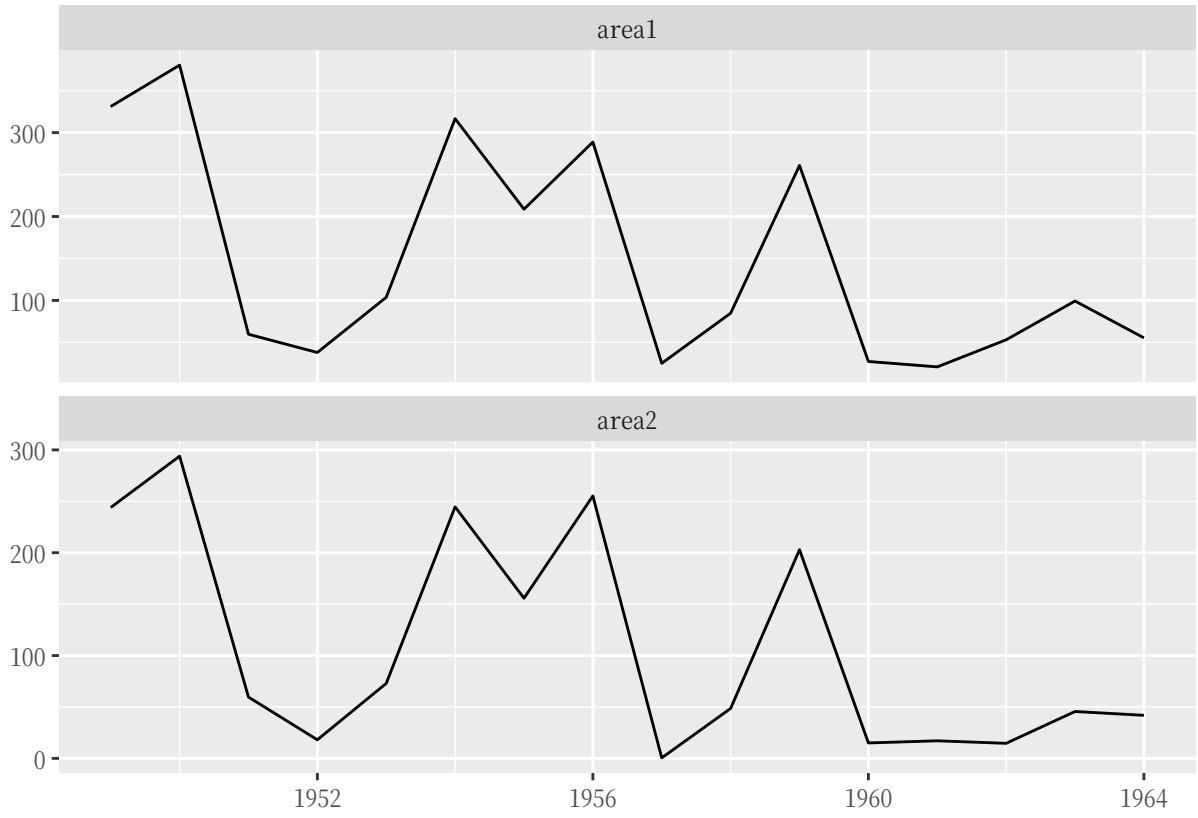
```
ggplot(data = dflood) +
  geom_line(mapping = aes(
    x = year, y = area1, color = " 受灾面积")) +
  geom_line(mapping = aes(
    x = year, y = area2, color = " 成灾面积")) +
  scale_color_manual(
    name = NULL,
    breaks = c(" 受灾面积", " 成灾面积"),
    values = c(
      " 受灾面积" = "orangered2",
      " 成灾面积" = "turquoise")) +
  labs(y = NULL)
```



为了获得适当的图例，上面的程序在两个 `geom_line()` 调用中在 `aes()` 中将 `color` 映射到了两个序列的名称而不是具体颜色，然后使用了 `scale_colormanual()` 函数，用 `breaks` 参数指定了两个序列的次序，使得“成灾面积”在前，“受灾面积”灾后，与图中两条曲线的上下次序一致；用了 `values` 参数指定了从序列名称到具体颜色的对应关系。

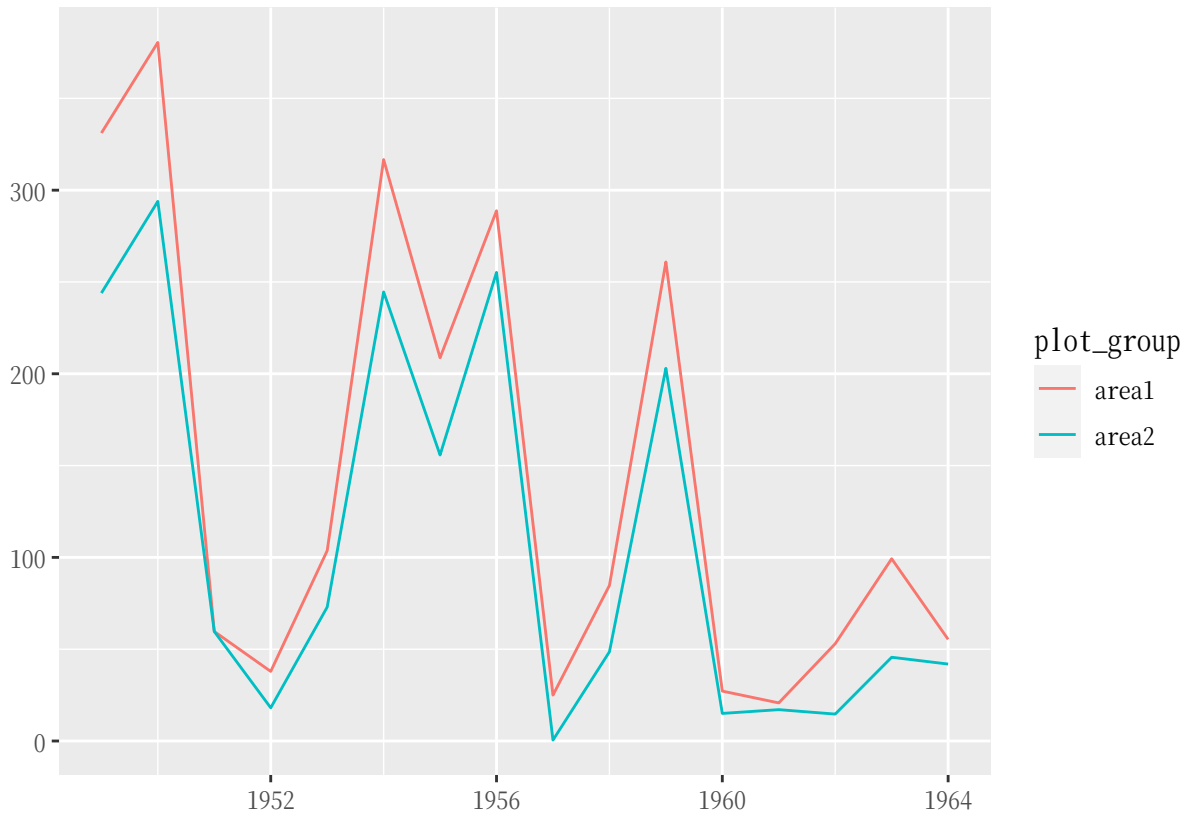
ggfortify 包可以直接对多元时间序列作图。将上述二元时间序列保存为 ts 类型后作图：

```
flood <- ts(as.matrix(dflood[,c("area1", "area2")]),
  start = 1949)
autoplot(flood)
```



也可以要求将两条曲线绘制在同一坐标系：

```
autoplot(flood, facets = FALSE)
```



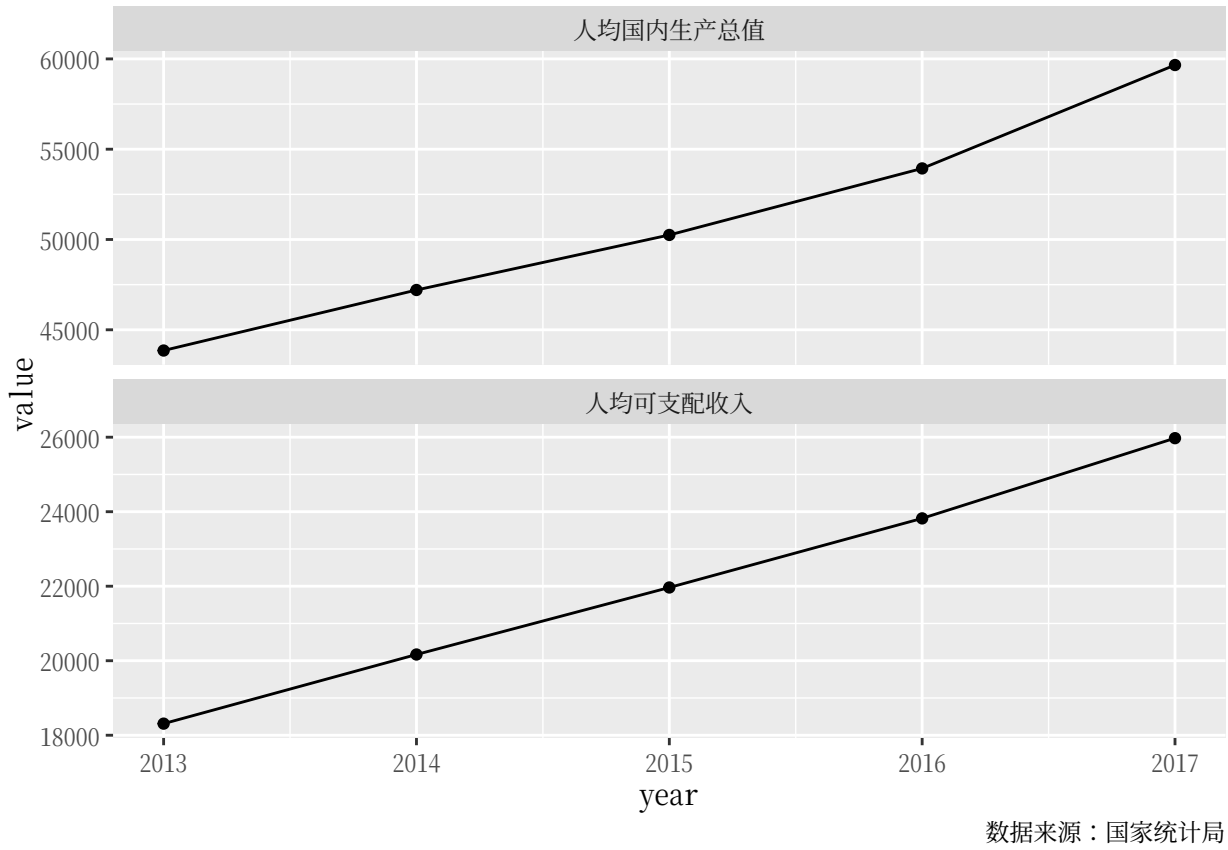
28.6.3 多个不同类型指标的时间序列

有时沿时间变化的多个指标是不可比的，这时就不应将数据画在同一坐标系内，一般可以画多个切片并上下按时间轴对齐，`facet_wrap()` 可以做切片。

例如，考虑我国 2013-2017 年人均国内生产总值和人均可支配收入数据。

```
dinc <- tibble(
  year = 2013:2017,
  `人均可支配收入` = c(18311, 20167, 21966, 23821, 25974),
  `人均国内生产总值` = c(43852, 47203, 50251, 53935, 59660))
dinc2 <- dinc |>
  pivot_longer(
    c(`人均可支配收入`, `人均国内生产总值`),
    names_to = " 指标",
    values_to = "value")
ggplot(dinc2, aes(
  x = year, y = value)) +
  geom_line() +
```

```
  geom_point() +
  facet_wrap(
    ~ `指标`,
    ncol = 1,
    scales = "free_y") +
  labs(caption = " 数据来源: 国家统计局")
```



上述程序在 `facet_wrap()` 中用 `scales = "free_y"` 使得两幅切片的纵轴不需要使用相同坐标范围。横轴默认还使用相同坐标范围并上下对齐。

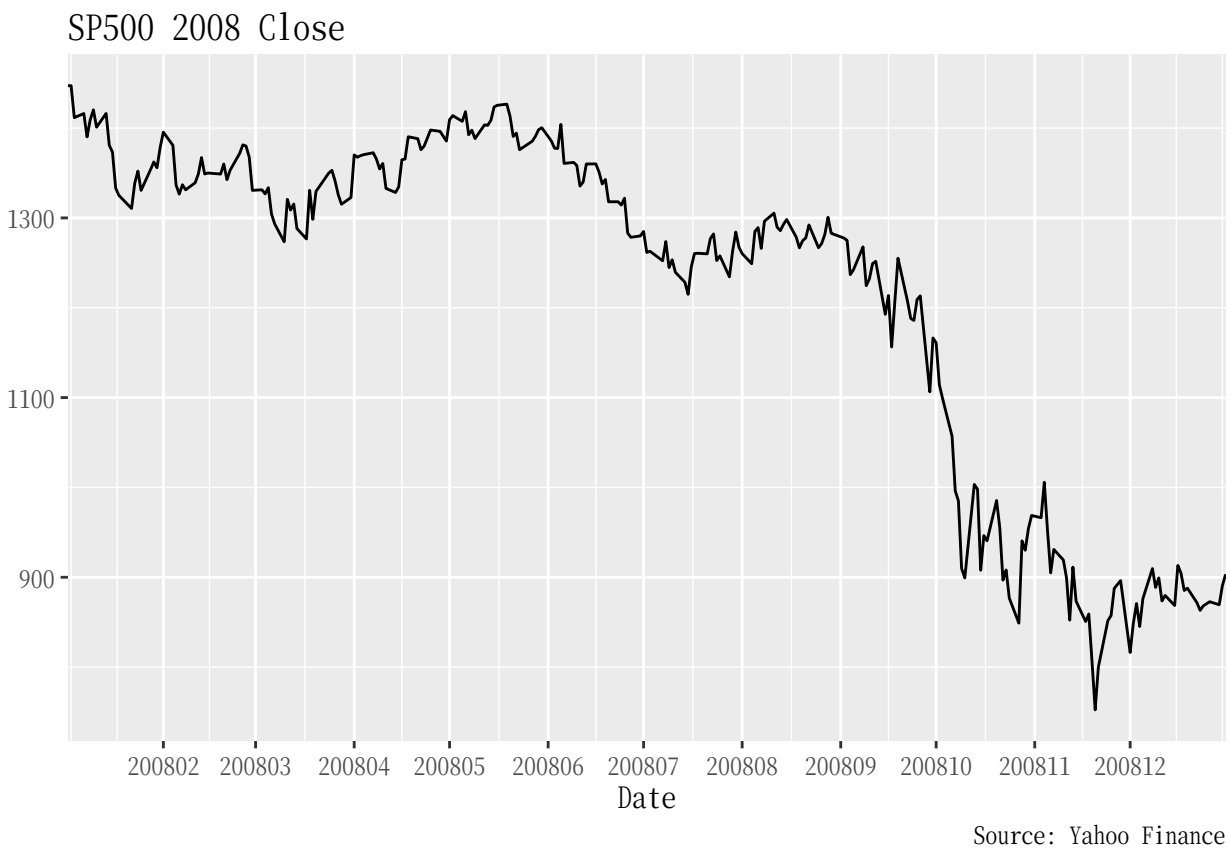
对二元时间序列 (x_t, y_t)，还可以画以 (x_t, y_t) 为节点的轨迹图，但需要标出关键的时间点的时间。用 `geom_path()` 画轨迹，用 ggrepel 包的 `geom_text_repel()` 标时间。

28.7 拟合曲线图

考虑两个变量之间的关系，将其中的一个作为因变量，另一个作为自变量，散点图可以反映其互相影响，可以用拟合曲线进一步概括两者的关系。拟合曲线的方法可以是曲线平滑方法如核回归、局部多项式回归、样条平滑，也可以用一些参数模型如线性回归、二次回归。对于时间序列，还经常分解成趋势项、季节项和不规则项。

28.7.1 平滑方法

考虑标普 500 指数 2008 年的收盘价。



为了平滑上面的曲线，金融界常用滑动平均方法，比如，每 10 天画计算一个平均值并将该平均值的时间对准在 10 天时间窗的最后一天。这样得到的滑动平均线会滞后于原始数据的变化。

先写一个计算滑动平均的函数：

```
moving_average <- function(x, span = 1, center = FALSE){
  co <- rep(1/span, span)
  y <- stats::filter(x, co, method = "convolution",
                     sides = ifelse(center, 2, 1))
  c(y)
}
```

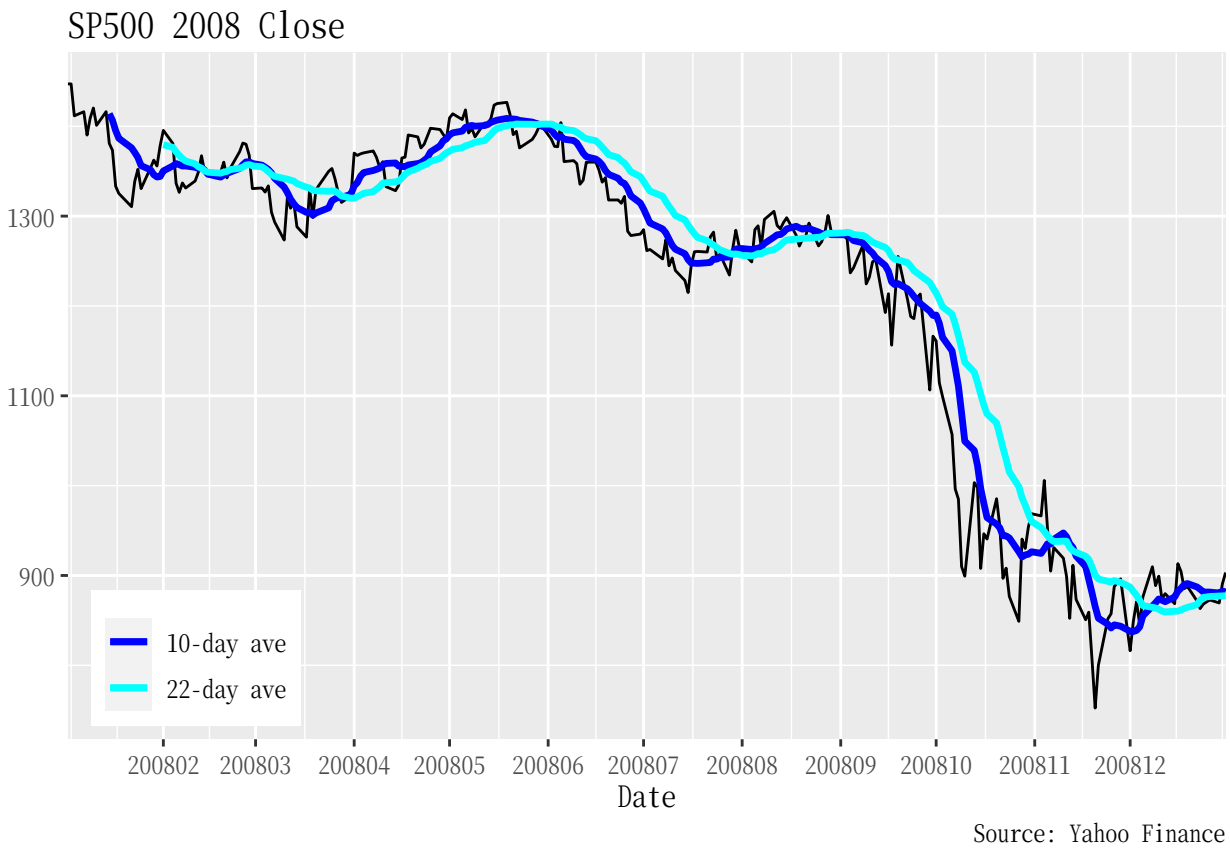
计算 10 日、22 日滑动平均并绘图：

```
dsp2 <- dsp |>
  mutate(
    ma10 = moving_average(close, 10, center = FALSE),
```

```
    ma22 = moving_average(close, 22, center = FALSE),
    ma10c = moving_average(close, 10, center = TRUE),
    ma22c = moving_average(close, 22, center = TRUE)  )
ggplot(dsp2, aes(
  x = time, y = close)) +
  geom_line() +
  geom_line(mapping = aes(
    y = ma10, color = "10d"  ),
    size = 1.2) +
  geom_line(mapping = aes(
    y = ma22, color = "22d"  ),
    size = 1.2) +
  scale_color_manual(
    name = NULL,
    values = c(`10d` = "blue", `22d` = "cyan"),
    breaks = c("10d", "22d"),
    labels = c("10-day ave", "22-day ave")  ) +
  scale_x_date(
    name = "Date",
    date_breaks = "1 month",
    date_labels = "%Y%m",
    expand = c(0,0) ) +
  theme(
    legend.position = c(0.02, 0.02),
    legend.justification = c(0,0)) +
  labs(
    title = "SP500 2008 Close",
    caption = "Source: Yahoo Finance",
    y = NULL)
```

```
## Warning: Removed 9 row(s) containing missing values (geom_path).
```

```
## Warning: Removed 21 row(s) containing missing values (geom_path).
```



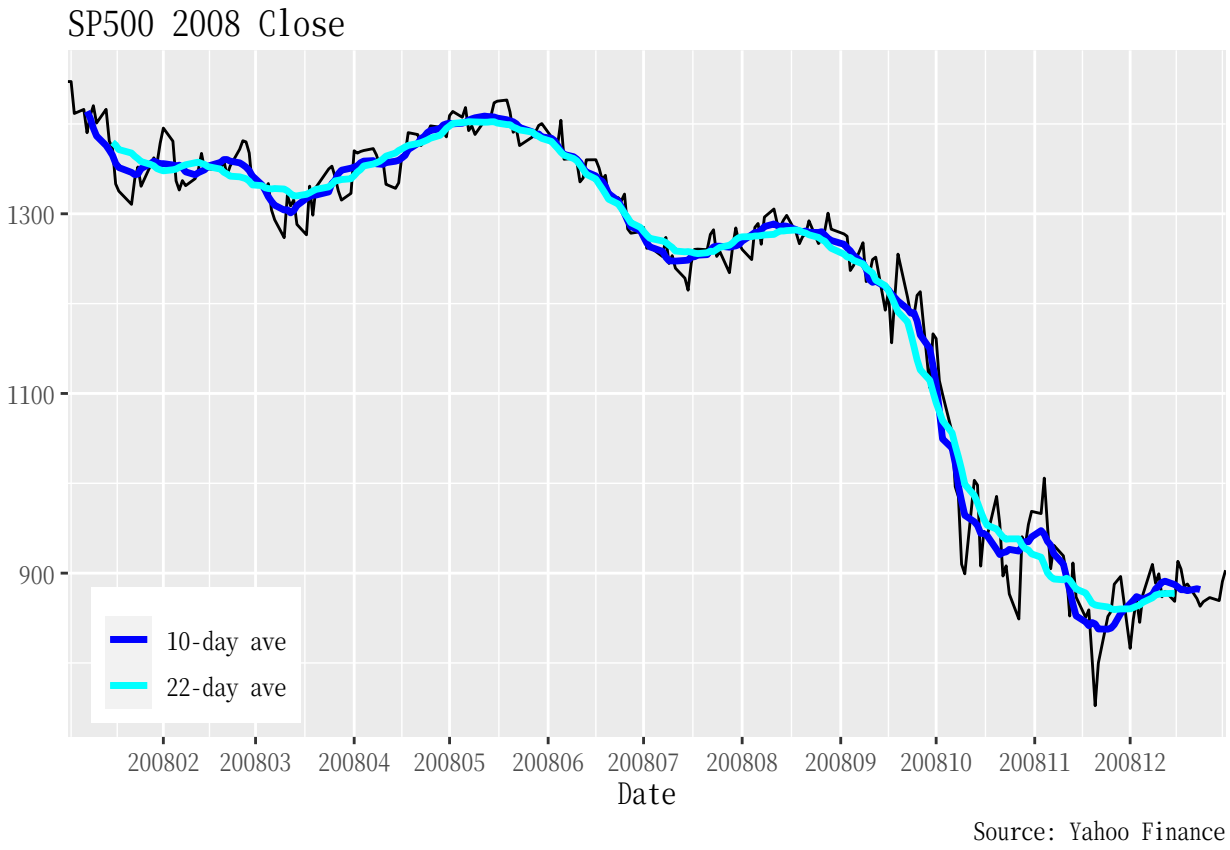
将滑动平均结果对齐在时间窗口末尾的优点是可以实时地更新。对历史数据，可以将滑动平均结果对齐在时间窗口中间，如果时间窗口中有偶数个点，可以选择中间两个点的任何一个，上面的滑动平均计算程序在偶数个点时选择与中间两个点当中第一个点对齐。图形如下：

```
ggplot(dsp2, aes(
  x = time, y = close)) +
  geom_line() +
  geom_line(mapping = aes(
    y = ma10c, color = "10d"  ),
    size = 1.2) +
  geom_line(mapping = aes(
    y = ma22c, color = "22d"  ),
    size = 1.2) +
  scale_color_manual(
    name = NULL,
    values = c(`10d` = "blue", `22d` = "cyan"),
    breaks = c("10d", "22d"),
    labels = c("10-day ave", "22-day ave")  ) +
  scale_x_date(
```

```
    name = "Date",
    date_breaks = "1 month",
    date_labels = "%Y%m",
    expand = c(0,0) ) +
  theme(
    legend.position = c(0.02, 0.02),
    legend.justification = c(0,0)) +
  labs(
    title = "SP500 2008 Close",
    caption = "Source: Yahoo Finance",
    y = NULL)
```

```
## Warning: Removed 9 row(s) containing missing values (geom_path).
```

```
## Warning: Removed 21 row(s) containing missing values (geom_path).
```



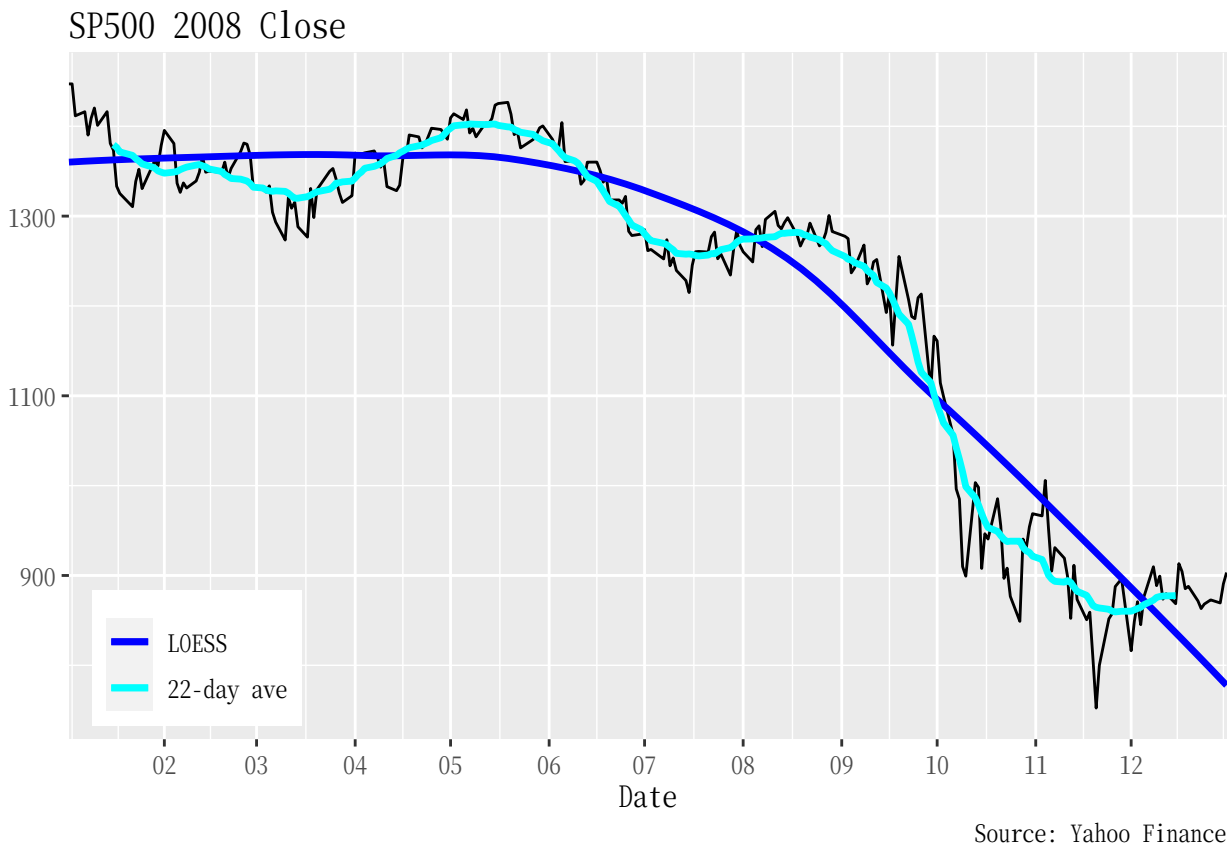
滑动平均平滑是金融数据分析中常用的方法，但是也有明显的缺点。因为需要多个点生成一个平均值，所以在序列的开头和末尾有一部分点是没有平均值的。当时间窗变宽时结果会更光滑，但有时还是不够光滑，这与滑动平均作为加窗平滑是等权而且在边缘处为间断有关。

统计学中提出了各种更复杂但效果更好而平滑方法，比如 LOESS(locally estimated scatterplot smoothing，参见 (Cleveland, 1979))。这是许多局部多项式回归回归方法中的一个典型代表，每个 x 处用附近的点拟合一个局部模型，距离 x 越远的点贡献越小。这样的做法可以得到比滑动平均更平滑的结果。局部多项式方法得到的结果类似于中心对准而且滑动窗口较宽的滑动平均。

```
ggplot(dsp2, aes(
  x = time, y = close)) +
  geom_line() +
  geom_smooth(mapping = aes(color = "LOESS"),
    method = "loess", se = FALSE, size = 1.2) +
  geom_line(mapping = aes(
    y = ma22c, color = "22d"  ),
    size = 1.2) +
  scale_color_manual(
    name = NULL,
    values = c(`LOESS` = "blue", `22d` = "cyan"),
    breaks = c("LOESS", "22d"),
    labels = c("LOESS", "22-day ave")  ) +
  scale_x_date(
    name = "Date",
    date_breaks = "1 month",
    date_labels = "%m",
    expand = c(0,0) ) +
  theme(
    legend.position = c(0.02, 0.02),
    legend.justification = c(0,0)) +
  labs(
    title = "SP500 2008 Close",
    caption = "Source: Yahoo Finance",
    y = NULL)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

```
## Warning: Removed 21 row(s) containing missing values (geom_path).
```



geom_smooth() 可以对输入的 (x, y) 坐标数据进行平滑，选项 se = FALSE 要求不做置信区间。实际上，不仅仅是时间序列数据可以添加如此的平滑曲线，其它的散点图也都可以这样添加平滑曲线。

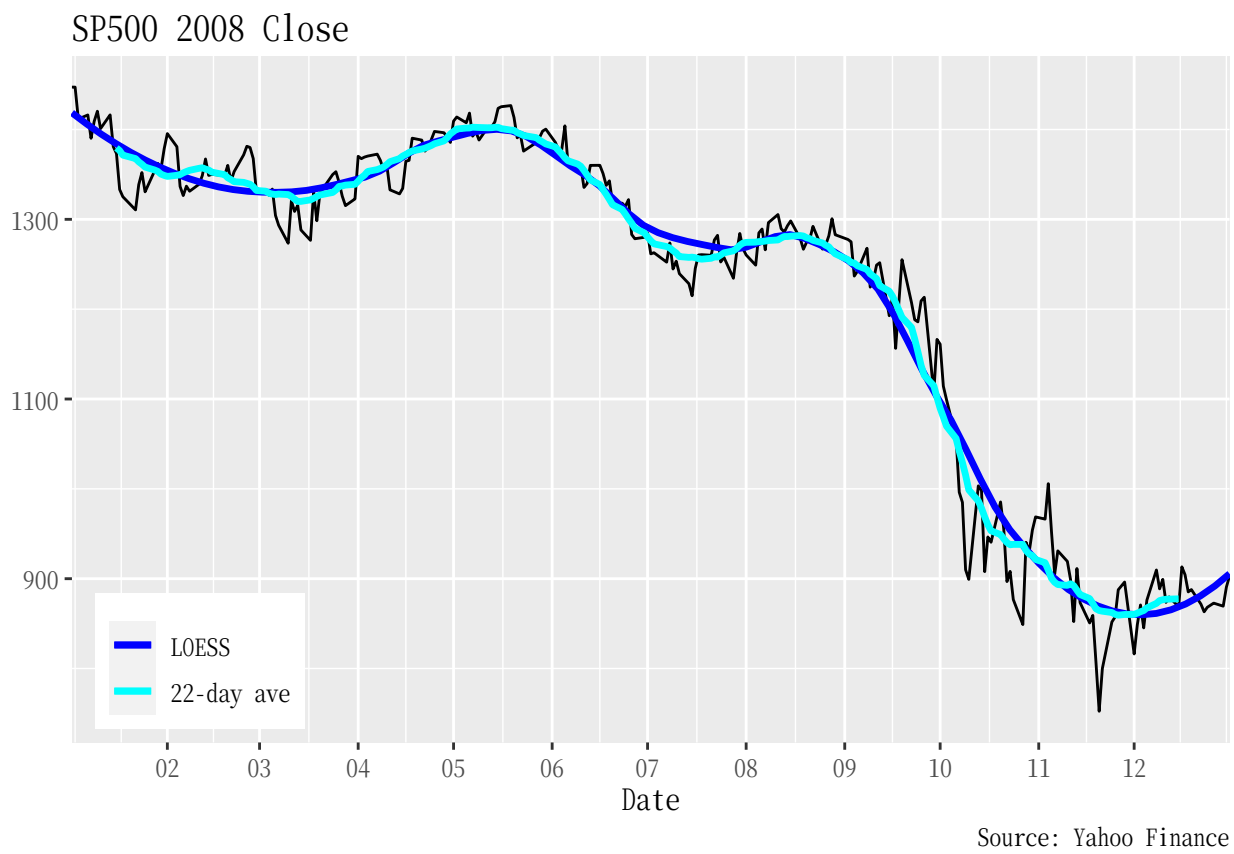
上图的结果是程序自动选择平滑程度的结果，有些过于平滑了。可以人为地用参数控制结果的平滑程度。用参数 span 控制平滑程度，span 取 0 到 1 之间的值，越大的 span 值越光滑：

```
ggplot(dsp2, aes(
  x = time, y = close)) +
  geom_line() +
  geom_smooth(mapping = aes(color = "LOESS"),
    method = "loess", span = 0.40, se = FALSE, size = 1.2) +
  geom_line(mapping = aes(
    y = ma22c, color = "22d"  ),
    size = 1.2) +
  scale_color_manual(
    name = NULL,
    values = c(`LOESS` = "blue", `22d` = "cyan"),
    breaks = c("LOESS", "22d"),
    labels = c("LOESS", "22-day ave")  ) +
  scale_x_date(
```

```
    name = "Date",
    date_breaks = "1 month",
    date_labels = "%m",
    expand = c(0,0) ) +
  theme(
    legend.position = c(0.02, 0.02),
    legend.justification = c(0,0)) +
  labs(
    title = "SP500 2008 Close",
    caption = "Source: Yahoo Finance",
    y = NULL)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

```
## Warning: Removed 21 row(s) containing missing values (geom_path).
```



LOESS 方法计算较慢，所以 ggplot2 在输入点数超过 1000 时默认使用 mgcv 包的 `gam()` 函数进行平滑，对应于 `method = "gam"`。用 gam 时可以指定 `formula = y ~ s(x)`，或者 `formula = y ~ s(x, bs = "cs")`。

要注意的是，平滑结果不仅依赖于平滑参数的选择，也依赖于平滑方法和模型的选择，不同模型可能给出差别很大的结果，所以使用平滑结果做预测时一定要特别谨慎。

28.7.2 参数模型方法

LOESS 等平滑方法属于非参数方法，这些方法本质上没有一个不依赖于数据值的参数模型。另一类建模和平滑方法是假定 y_i 与 x_i 之间服从一定的参数模型，如线性回归模型

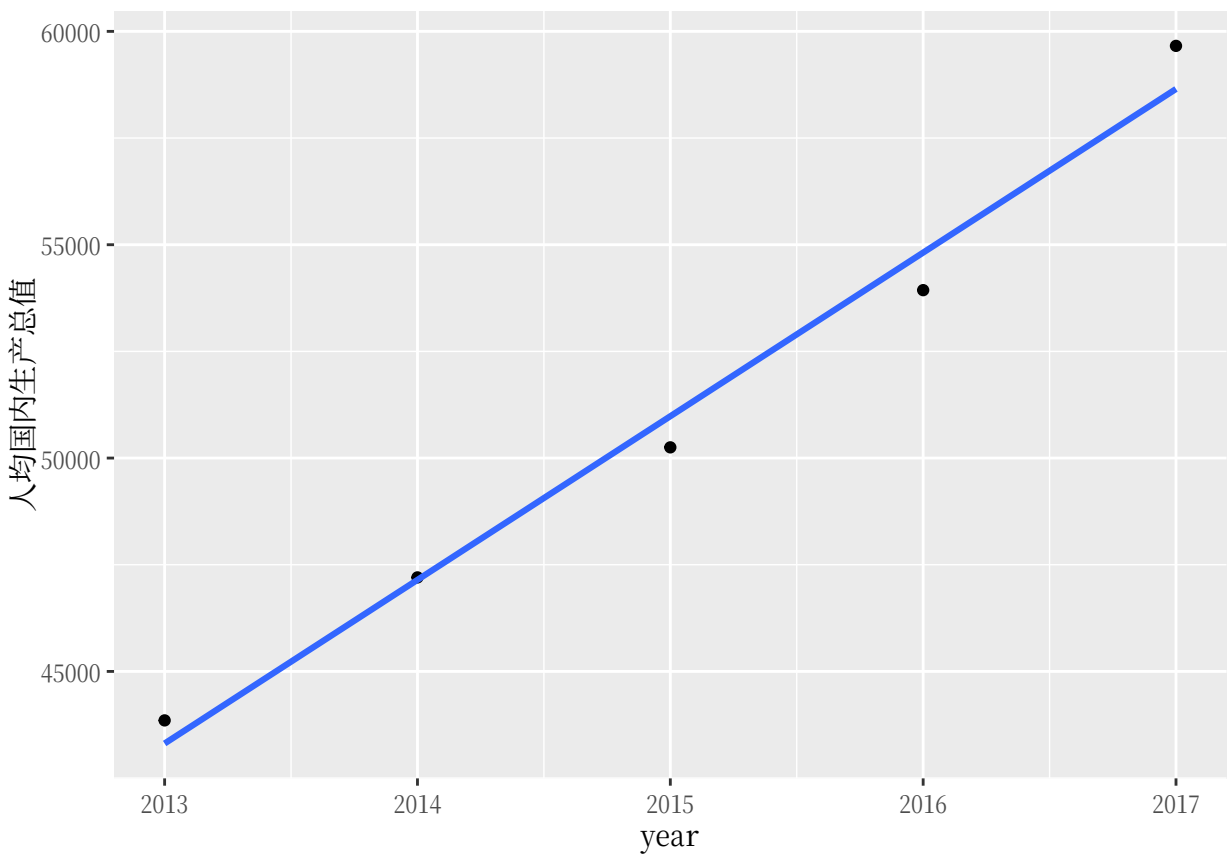
$$y_i = a + bx_i + \varepsilon_i$$

从数据估计模型参数，然后可以得到模型的拟合值（或称预测值）。

`geom_smooth()` 函数已经支持线性回归，比如对 2013-2017 人均国内生产总值数据进行平滑：

```
ggplot(dinc, aes(
  x = year, y = `人均国内生产总值`)) +
  geom_point() +
  geom_smooth(
    method = "lm", size = 1.1, se=FALSE )
```

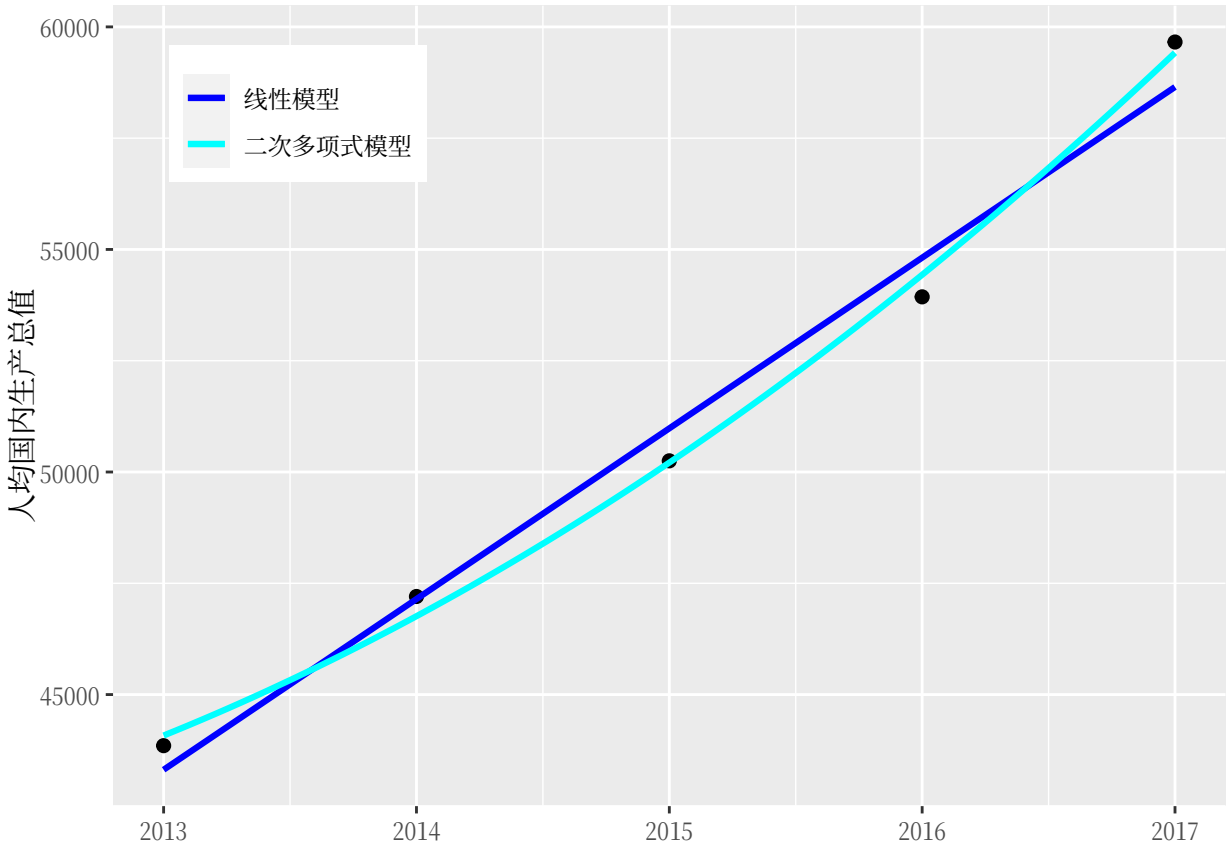
```
## `geom_smooth()` using formula 'y ~ x'
```



可以用 `formula = poly(x, 2)` 将模型变成二次多项式曲线，如：

```
ggplot(dinc, aes(
  x = year, y = `人均国内生产总值`)) +
  geom_point(size = 2) +
  geom_smooth(mapping = aes(col = "lm"),
    method = "lm", size = 1.1, se=FALSE ) +
  geom_smooth(mapping = aes(col = "quadreg"),
    method = "lm", formula = y ~ poly(x, 2),
    size = 1.1, se=FALSE) +
  scale_color_manual(
    name = NULL,
    values = c(`lm` = "blue", `quadreg` = "cyan"),
    breaks = c("lm", "quadreg"),
    labels = c(" 线性模型", " 二次多项式模型")  ) +
  theme(
    legend.position = c(0.05, 0.95),
    legend.justification = c(0, 1)  ) +
  labs(x = NULL)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



与 `method = "lm"` 类似的是 `method = "rlm"`，是稳健线性回归模型，要调用 MASS 包。

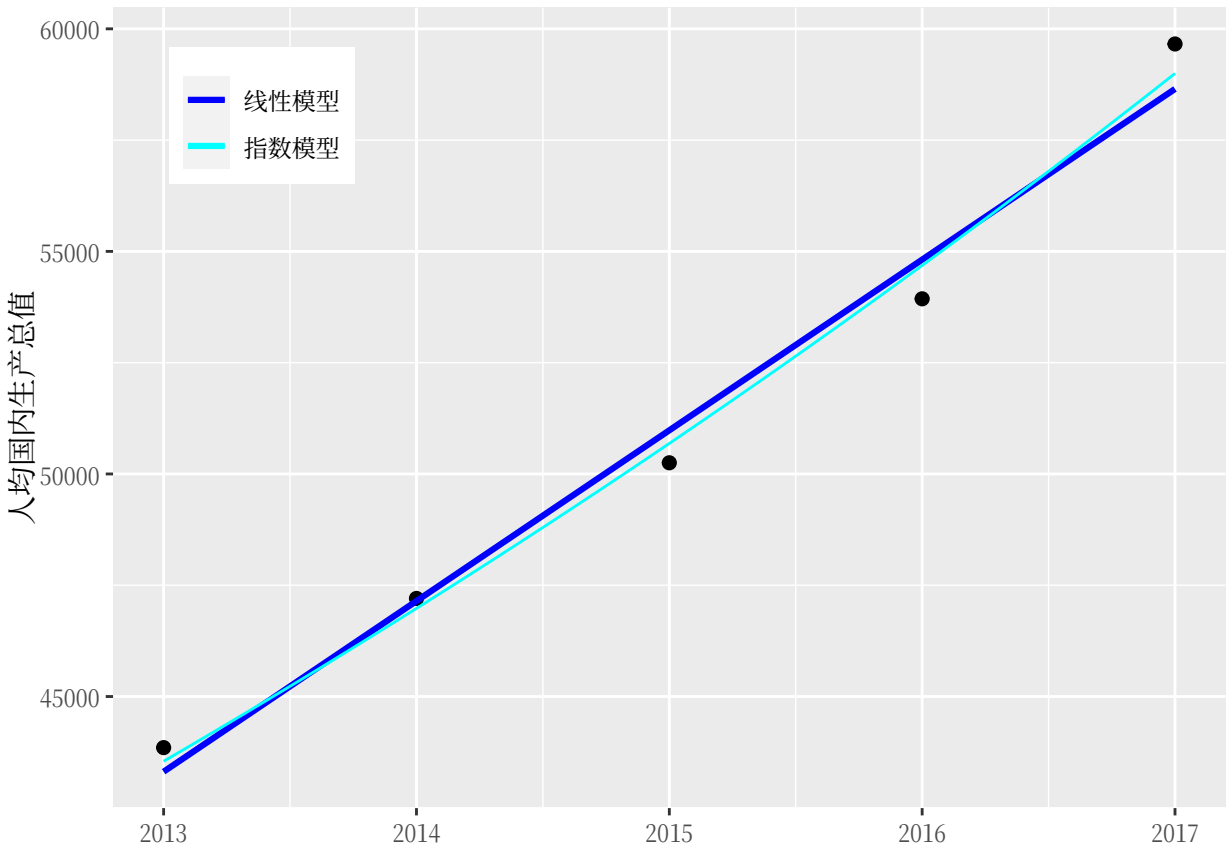
`geom_smooth()` 还支持 `gam()` 函数能支持的模型。更一般的模型可以先用统计建模函数建模，然后在自变量取值范围内取一个网格计算拟合值，最后将原始数据与拟合曲线画在一起。这里拟合曲线实际是将网格点对应的拟合值连接成一条折线。

比如，对上面的数据拟合如下的指数增长模型：

$$y_t = Ae^{bt} + \varepsilon_t$$

```
nls.inc <- nls(
  `人均国内生产总值` ~ A * exp(b * (year-2000)),
  data = dinc,
  start = c(A = 4E4, b = 1E-6))
dinc.pred1 <- data.frame(
  year = seq(2013, 2017, length.out = 100))
dinc.pred1 <- dinc.pred1 |>
  mutate(pred = predict(
      nls.inc, newdata = data.frame(year =  year )))
ggplot(dinc, aes(
  x = year, y = `人均国内生产总值`)) +
  geom_point(size = 2) +
  geom_smooth(mapping = aes(col = "lm"),
    method = "lm", size = 1.1, se=FALSE ) +
  geom_line(data = dinc.pred1, mapping = aes(
    x = year, y = pred, col = "Expmod"  )) +
  scale_color_manual(
    name = NULL,
    values = c(`lm` = "blue", `Expmod` = "cyan"),
    breaks = c("lm", "Expmod"),
    labels = c(" 线性模型", " 指数模型")  ) +
  theme(
    legend.position = c(0.05, 0.95),
    legend.justification = c(0, 1)  ) +
  labs(x = NULL)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



$y = Ae^{bx}$ 这样的模型可以将 y 轴改为对数尺度，这时指数曲线可以表现为直线。

28.8 表现不确定性

统计数据数据分析的建模结果都有一定的不确定性，参数估计和模型预测的不确定性经常用置信区间和置信带表现，绘图时可以画误差条 (error bar) 或置信带。这些方法可以比较精确地描绘不确定性大小，但需要一些概率统计知识才能正确理解。也可以开发一些更直观的图形用来表现不确定性。

28.8.1 表现概率

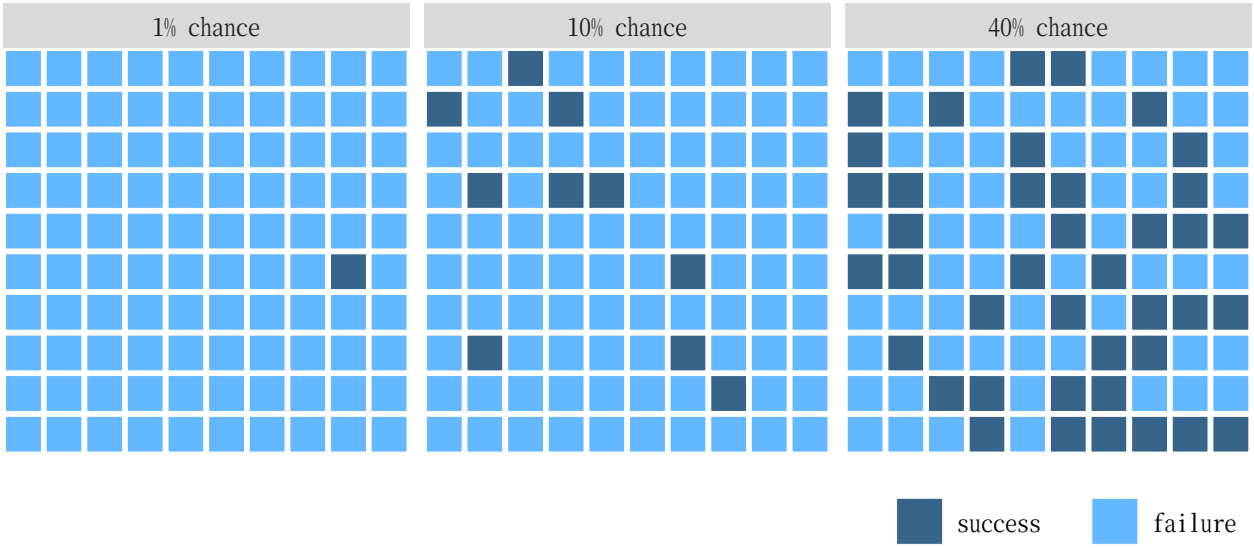
不确定性用概率描述，而概率可以通过多次独立重复试验的频率来理解。可以用一个饼图来表现概率。

一种容易理解的做法是画 100 个色块，用加亮显示部分表示成功部分，加亮显示部分随机摆放以显示结果的随机性。

```
#dg <- expand.grid(x = 1:10, y = 1:10)
set.seed(101)
dg <- expand.grid(ratio = c(0.01, 0.1, 0.4), x = 1:10, y = 1:10)
dg2 <- dg |>
```

```
  group_by(ratio) |>
  mutate(value = {
    n <- n()
    i <- round(n * ratio[1])
    sample(c(rep("S", i), rep("F", n-i)))
  }) |>
  ungroup() |>
  mutate(label = paste0(round(100*ratio), "% chance") )
#set.seed(84524)

ggplot(dg2, aes(
  x = x, y = y, fill = value)) +
  geom_tile(color = "white", size = 1) +
  coord_fixed(expand = FALSE, clip = "off") +
  scale_x_continuous(name = NULL, breaks = NULL) +
  scale_y_continuous(name = NULL, breaks = NULL) +
  scale_fill_manual(
    name = NULL,
    breaks = c("S", "F"),
    labels = c("success   ", "failure"),
    values = c(
      "S" = "steelblue4",
      "F" = "steelblue1"
    ),
    guide = guide_legend(override.aes = list(size = 0))
  ) +
  facet_wrap(~label) +
  theme(
    legend.position = "bottom",
    legend.direction = "horizontal",
    legend.justification = "right",
    # crop plot a little more tightly
    plot.margin = margin(0, 0, 3.5, 0)
  )
```



在只有两种随机结果时上图可以比较直观地表现两种结果的比例和随机性。在有许多可能结果，甚至于随机结果可以在一个区间内随机取值时，上面的方法不再有效，概率分布和概率密度能更准确地描述这种随机取值，但不一定容易理解。分布密度下方的阴影面积比例可以代表某个事件的概率，还可以用密度曲线下面画堆叠的圆的办法更直观地表现不同取值区域的概率大小。

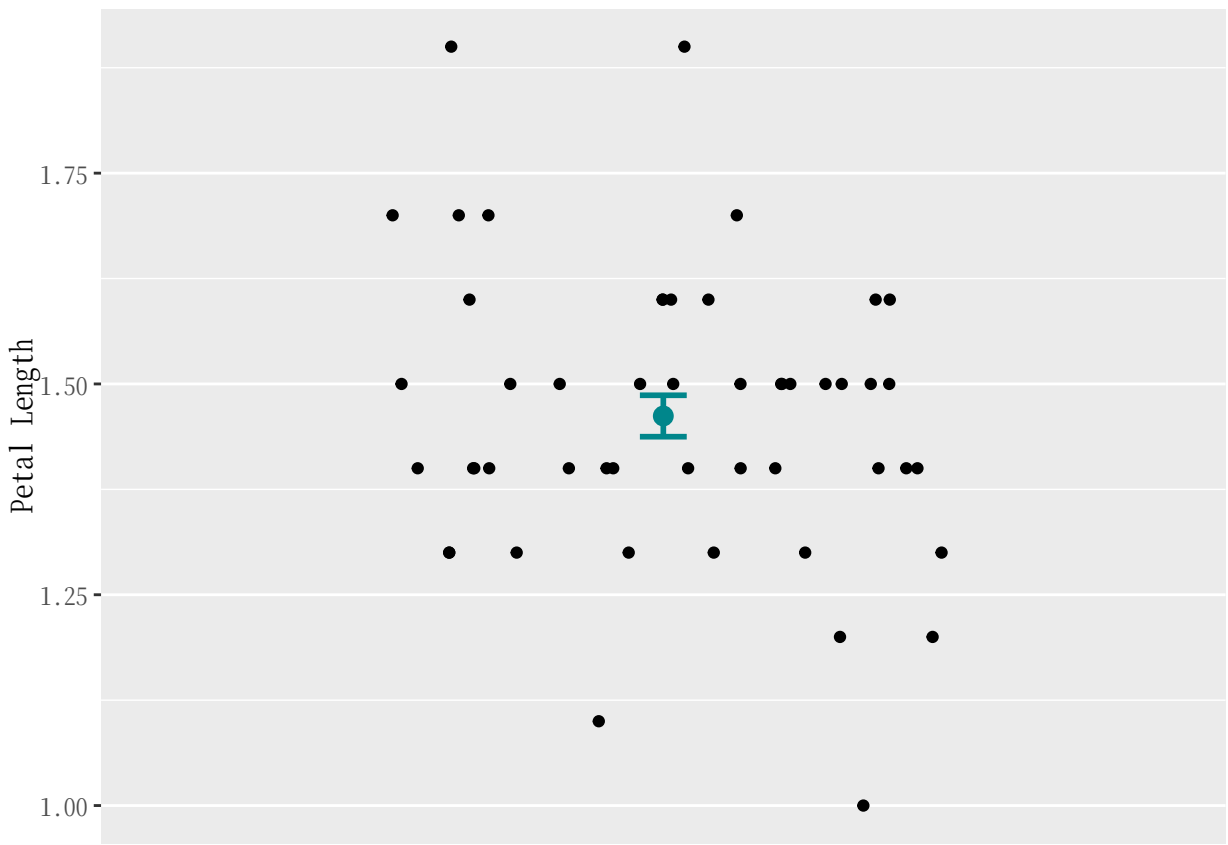
28.8.2 表现点估计精度

统计学中将要了解的分布称为总体，用参数概括总体的分布，从总体中抽取的一部分称为样本，从样本中计算统计量作为总体参数的估计，标准误差是估计量的精度的度量。可以用误差条同时画出参数估计值及对应的标准误差，误差条的两端是点估计加减标准误差。

例如，iris 数据集中 setosa 品种的花瓣长度均值及标准误差，同时画了对应的数据值：

```
d1 <- iris |>
  filter(Species == "setosa")
d1 |>
  summarise(
    mpl = mean(Petal.Length),
    sd = sd(Petal.Length),
    se = sd / sqrt(n())) |>
```

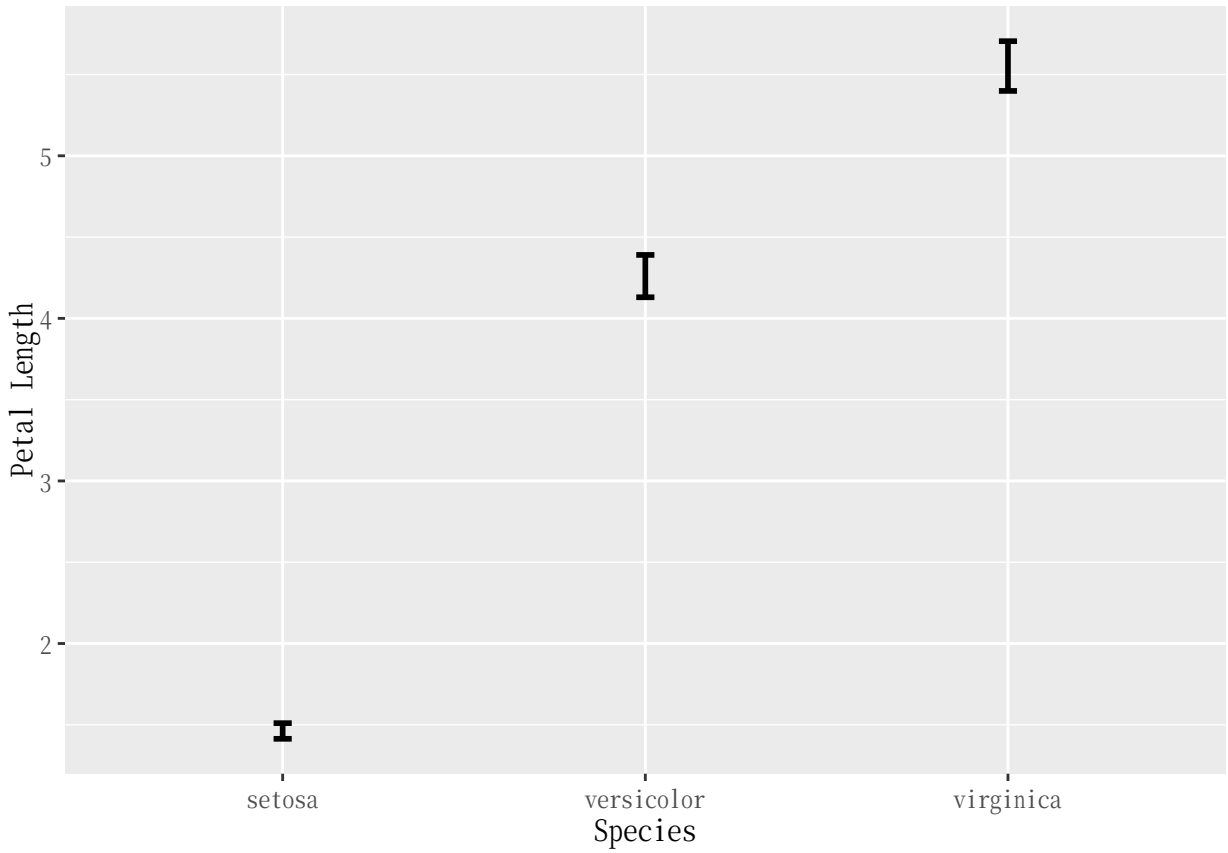
```
ggplot() +
geom_errorbar(
  mapping = aes(
    ymin = mpl - se, ymax = mpl + se,
    x = "setosa"),
  color = "turquoise4", width = 0.05, size = 1) +
geom_point(
  mapping = aes(
    y = mpl,
    x = "setosa"),
  color = "turquoise4", size = 3) +
geom_jitter(data = d1, mapping = aes(
  y = Petal.Length, x = "setosa"),
  width = 0.3, height = 0) +
scale_x_discrete(name = NULL, breaks = NULL) +
labs(y = "Petal Length")
```



在 `geom_errorbar()` 中需要映射 ymin, ymax 和 x 维，分别表示误差条的上下限与 x 位置。

置信区间也可以类似地画出。下面对 iris 数据集中三个品种分别估计花瓣长度均值并画 95% 置信区间。

```
d2summ <- iris |>
  group_by(Species) |>
  summarise(
    mpl = mean(Petal.Length),
    sd = sd(Petal.Length),
    se = sd / sqrt(n()))
p <- ggplot(d2summ, aes(x = Species))
p + geom_errorbar(
    mapping = aes(
      ymin = mpl - 1.96*se, ymax = mpl + 1.96*se),
    width = 0.05, size = 1) +
  labs(y = "Petal Length")
```

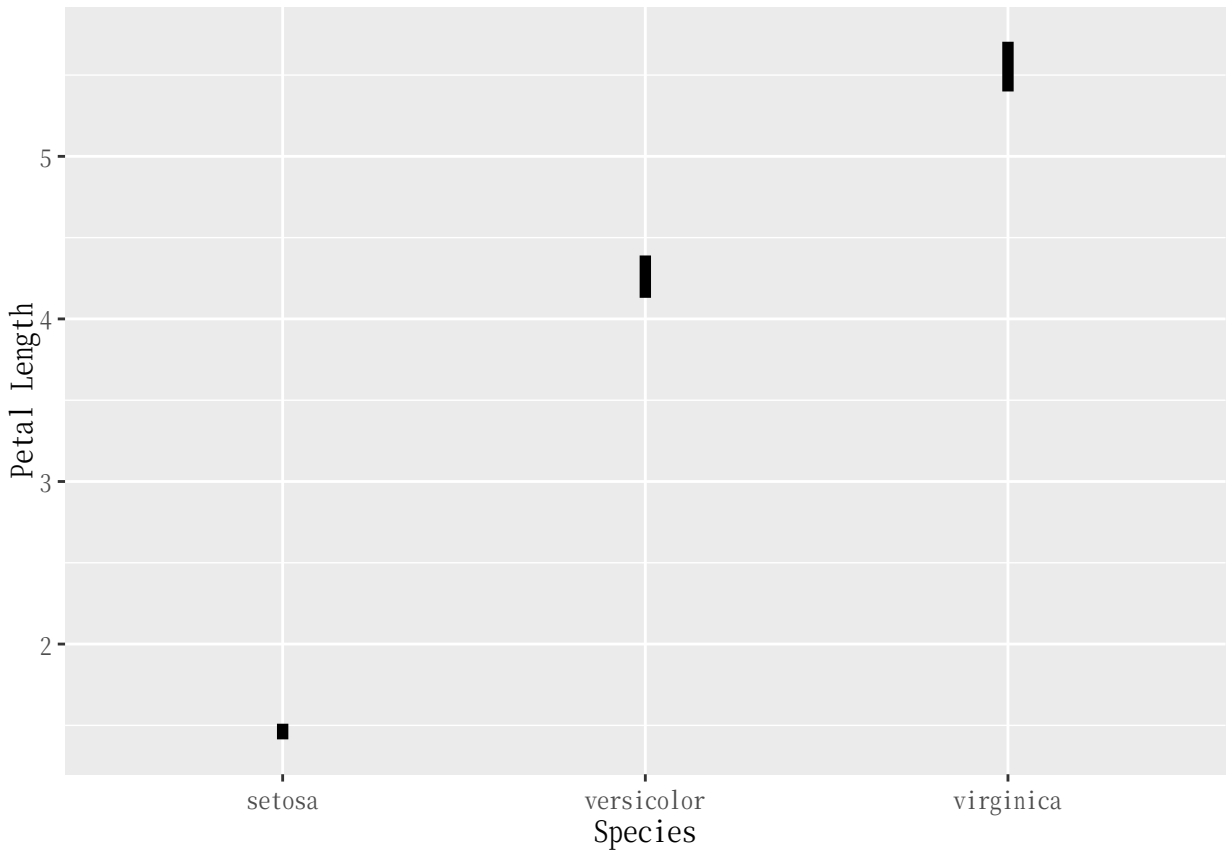


选项 `width` 是端线宽度，用坐标系单位，横轴是因子时两个因子的间距为 1；`size` 是线粗细（毫米）

如果不需要端线，可以用 `geom_linerange()`:

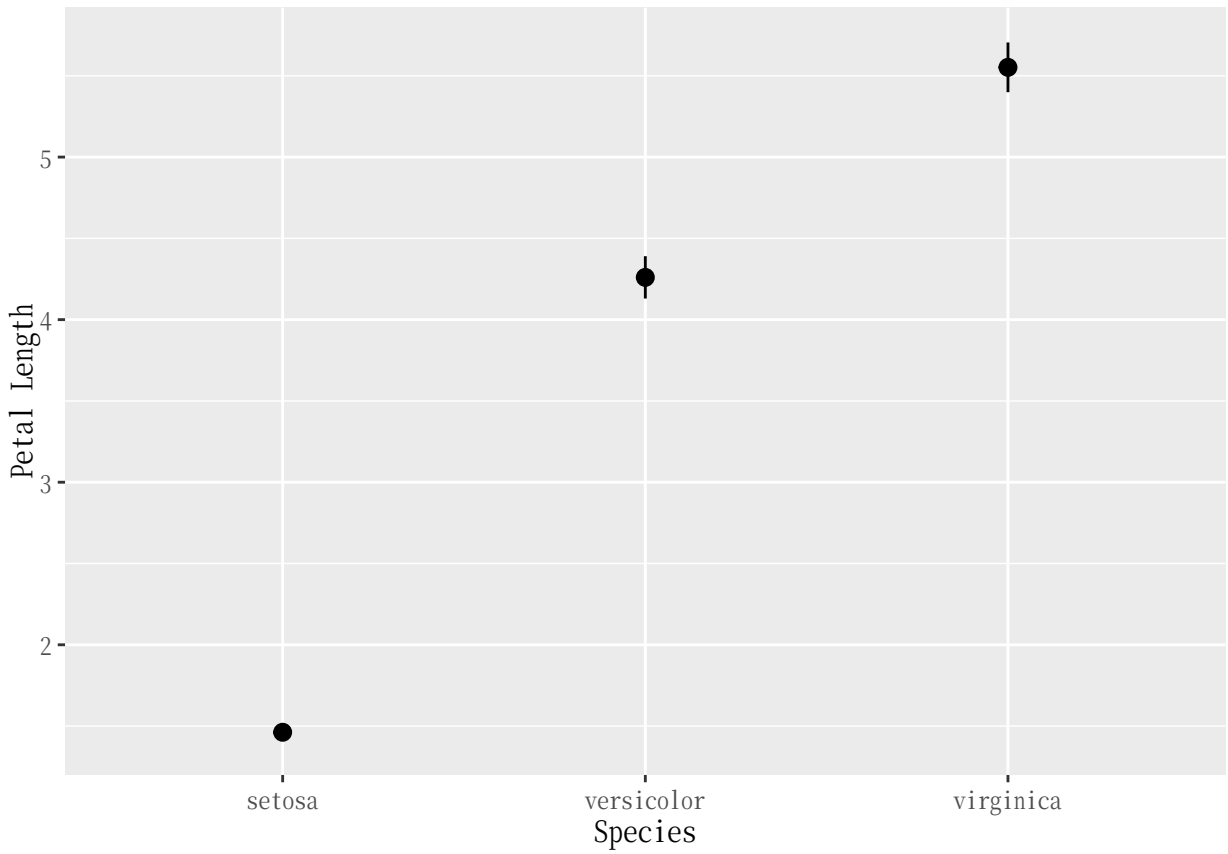
```
p + geom_linerange(
    mapping = aes(
      y = mpl,
```

```
      ymin = mpl - 1.96*se, ymax = mpl + 1.96*se),
    size = 2) +
  labs(y = "Petal Length")
```



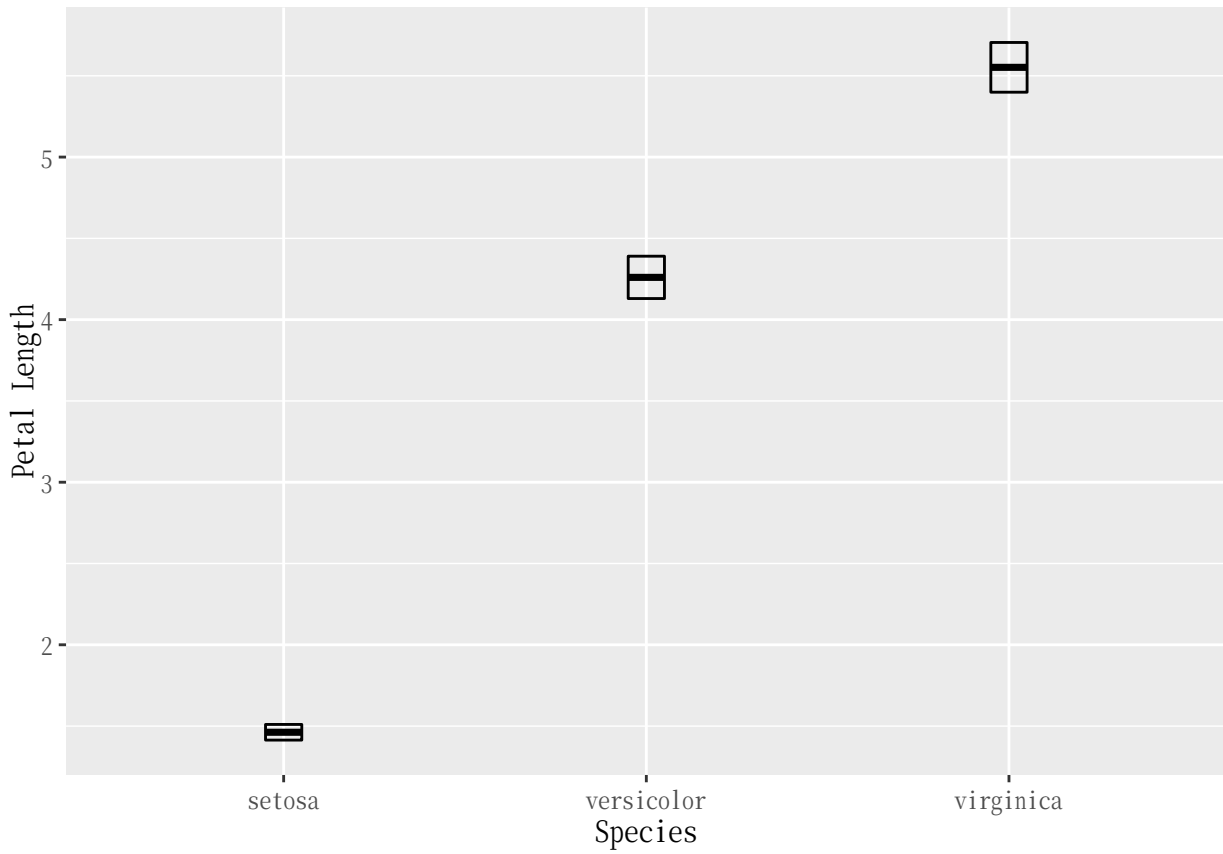
为了画出中心点，可以用 `geom_pointrange()` 函数：

```
p + geom_pointrange(
    mapping = aes(
      y = mpl,
      ymin = mpl - 1.96*se, ymax = mpl + 1.96*se)) +
  labs(y = "Petal Length")
```



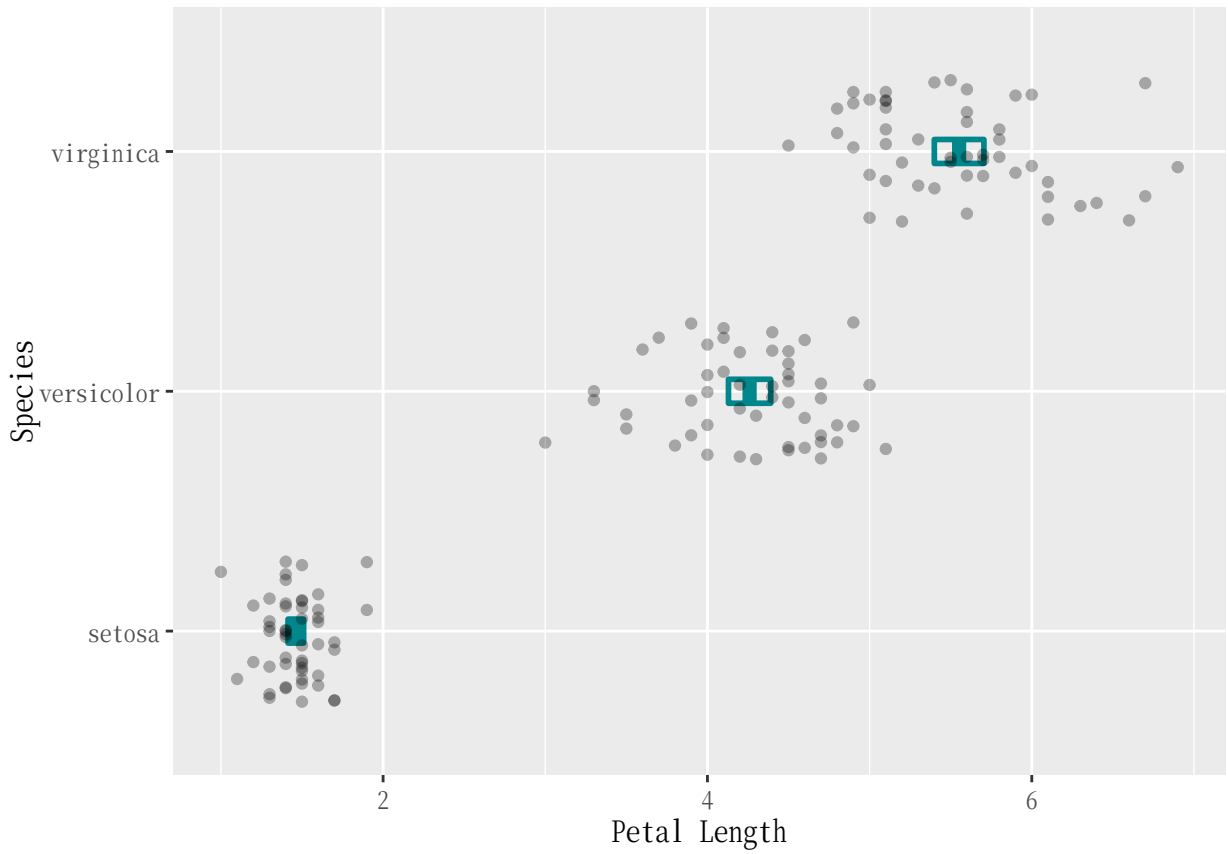
用 geom_crossbar() 可以画成中心有线的盒形：

```
p + geom_crossbar(
    mapping = aes(
      y = mpl,
      ymin = mpl - 1.96*se, ymax = mpl + 1.96*se),
    width = 0.1) +
  labs(y = "Petal Length")
```



下面画三个置信区间以及数据散点，并将 x 轴和 y 轴对调：

```
p + geom_crossbar(
    mapping = aes(
      y = mpl,
      ymin = mpl - 1.96*se, ymax = mpl + 1.96*se),
    color = "turquoise4", width = 0.1, size = 1) +
  geom_jitter(data = iris, mapping = aes(
    y = Petal.Length, x = Species),
    alpha = 0.3, width = 0.3, height = 0) +
  labs(y = "Petal Length") +
  coord_flip()
```



置信区间是频率学派统计学的概念，可以认为在多次独立重复同一置信区间问题时真实参数落入置信区间的比例为给定的置信度，或者零假设下的参数值落入置信区间是对应的假设检验的接受域。而贝叶斯学派统计学也有类似概念，称为 credible interval，这是认为参数作为服从后验分布的随机变量的取值概率为给定置信度最可能取值区间。

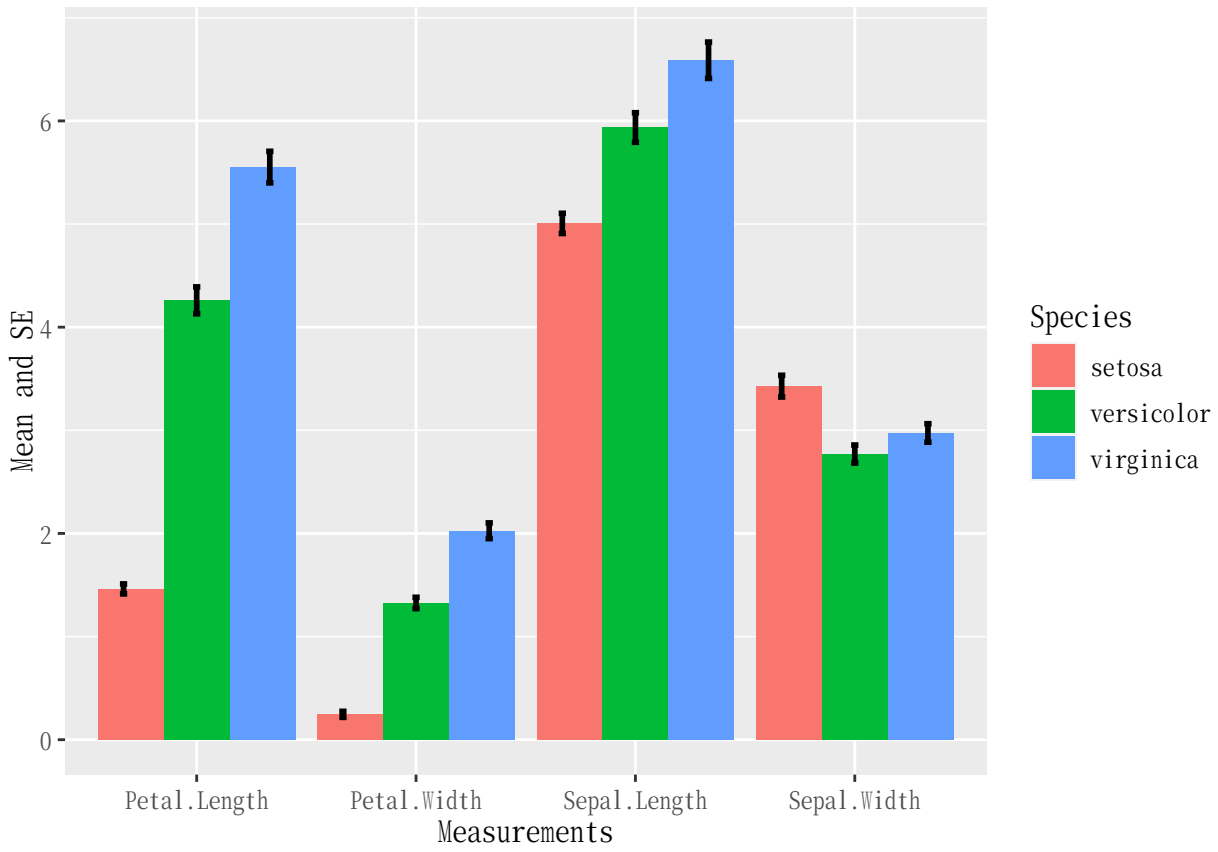
误差条比较容易被不了解相关概念的人误认为数据范围或者参数估计取值范围，所以使用时需要解释其含义。误差条的一个优势是很容易与其他图形一起使用，上面的误差条是与散点图一起使用的，也可以与条形图配合给出条形图表示的比例的误差大小。例如，iris 数据中三个类别的 4 个测量值的均值用条形高度表示，标准误差用误差条表示：

```
d3summ <- iris |>
  gather(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width,
         key = "variable", value = "value") |>
  group_by(Species, variable) |>
  summarise(
    mean = mean(value),
    se = sd(value) / sqrt(n()),
    lbar = mean - 1.96*se,
    hbar = mean + 1.96*se)
```

```
## `summarise()` has grouped output by 'Species'. You can override using the
```

```
## `.groups` argument.
```

```
ggplot(d3summ, aes(
  x = variable, fill = Species, y = mean)) +
  geom_col(position = "dodge") +
  geom_errorbar(mapping = aes(
    ymin = lbar, ymax = hbar),
    position = position_dodge(width = 1),
    width = 0.1, size = 1) +
  labs(x = "Measurements", y = "Mean and SE")
```

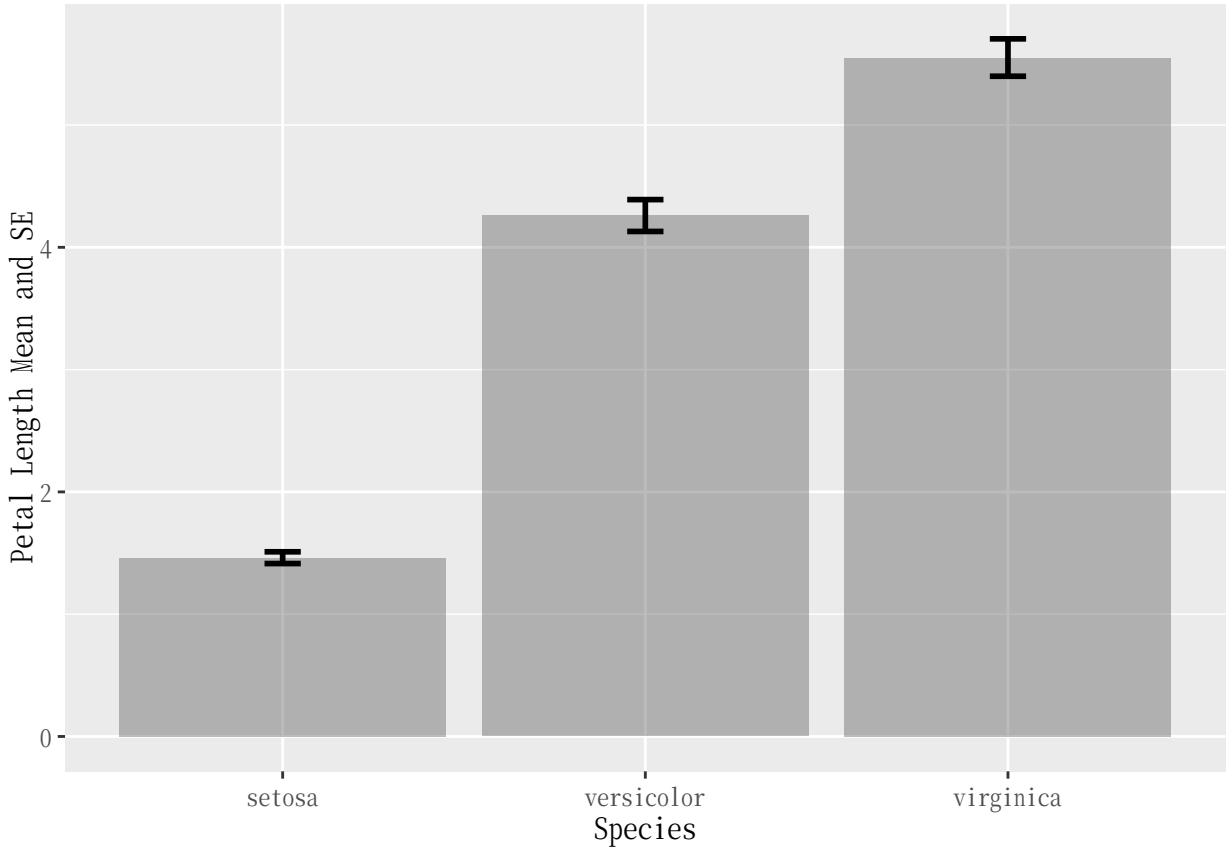


这里 `geom_errorbar()` 的 `position` 参数用了 `position_dodge()` 函数说明两个条形之间的距离。`geom_errorbar()` 的 `width` 参数是条形端线的宽度，如果 `width=0` 就没有了端线。

如果仅有一个变量则会比较简单，比如，花瓣长度的均值和加减标准误差：

```
d3summ |>
  filter(variable == "Petal.Length") |>
  ggplot(aes(
  x = Species, y = mean)) +
  geom_col(alpha = 0.4) +
```

```
  geom_errorbar(mapping = aes(
    ymin = lbar, ymax = hbar),
    width = 0.1, size = 1) +
  labs(x = "Species", y = "Petal Length Mean and SE")
```



为了表现 y 的误差范围随一个连续的 x 变量的变化，可以使用 `geom_ribbon()` 函数；为了表现 y 的误差范围以及中心值随一个连续的 x 变量的变化，可以使用 `geom_smooth(stat = "identity")`。

28.8.3 拟合曲线的置信带

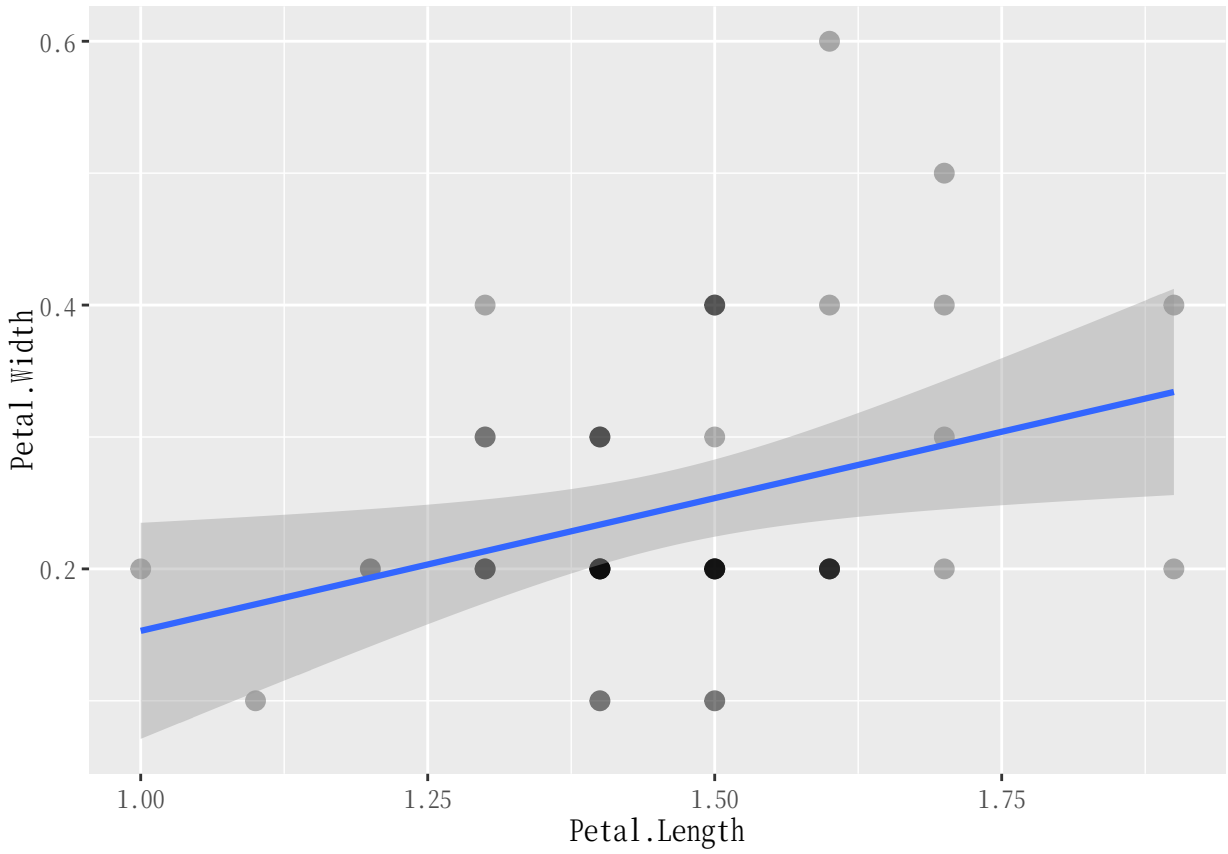
对成对变量数据 (x_i, y_i) 可以拟合平滑曲线，同时往往希望给出曲线类似于置信区间或者 credible interval。作为置信区间解释时，虽然是置信带，但是置信度是针对每个单独的 x 值成立的；如果是贝叶斯置信区间 (credible interval)，则置信带可以理解为在模型参数的后验分布下取值概率为对应置信度的参数所对应的曲线集合。

`geom_smooth()` 函数中加 `se = TRUE` 就可以做置信带。例如，对 iris 数据集中 setosa 品种的花瓣长、宽作线性回归并加置信带：

```
diris1 <- iris |>
  filter(Species == "setosa")
p <- ggplot(diris1, aes(
```

```
  x = Petal.Length, y = Petal.Width))
p + geom_point(size = 3, alpha = 0.3) +
  geom_smooth(
    method = "lm", size = 1.1, se=TRUE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

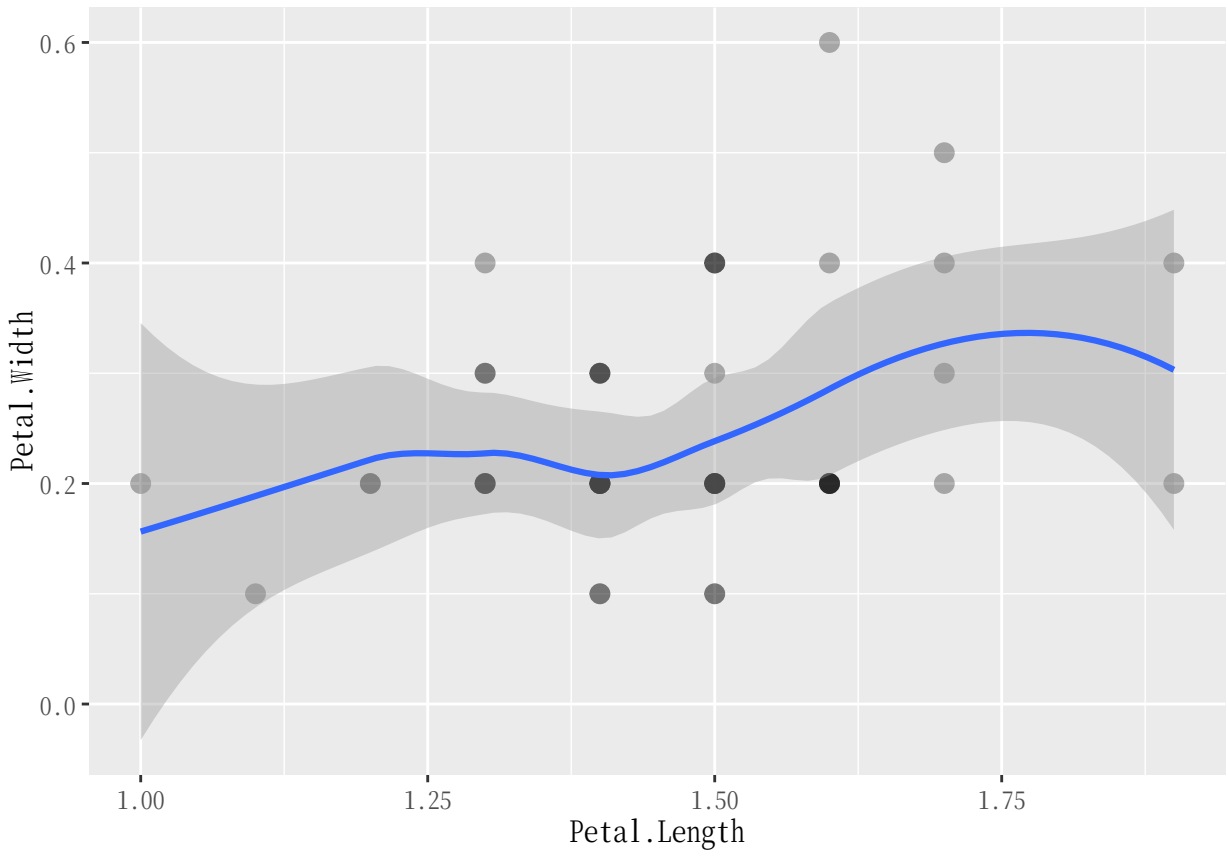


默认的置信度是 0.95，用参数 `level` 控制。`geom_smooth()` 的置信带是逐点意义的。

LOESS 等平滑曲线也可以加置信带，如：

```
p + geom_point(size = 3, alpha = 0.3) +
  geom_smooth(
    method = "loess", size = 1.1, se=TRUE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



28.8.4 随机结果动态演示

随机结果是概率分布确定但每个结果预先未知的。在讲课时，可以用动图（gif 或者 mp4 等格式）动态地演示不同的随机结果。gganimate 扩展包提供了制作动图的功能。

Part VII

统计模型

Chapter 29

探索性数据分析

探索性数据分析 (Exploratory data analysis, EDA) 是在进行推断性统计建模之前，对数据的分布、变量之间的关系、观测之间的聚集等特性用汇总统计、作图等方法进行探索，这是必不可少的步骤。有时我们不知道数据告诉我们什么信息，不知道要提什么问题，这就必须用 EDA 来获得对数据的洞察，提示我们随后要进行的研究。即使一开始研究目标就很明确，我们也需要用 EDA 方法确定数据中是否有错误，变量分布是否满足模型需要的假定，变量值有没有异常，变量之间的关系从图形直观看是否符合我们要建立的模型。

数据中的变量大致可以分为属性变量（分类变量）和区间变量（数值型的变量）。属性变量又分为名义型的，没有大小次序，以及有序的（如学历等级）。区间变量包括取整数值的和取浮点值的变量，其中取整数值的变量当可取值较少时，也呈现出一定的分类变量特征。

以 NHANES 包的 NHANES 数据集为例。这是一个规模较大的示例数据框，可以看作是美国扣除住院病人以外的人群的一个随机样本，有 10000 个观测，有 76 个变量，主题是个人的健康与营养方面的信息。

首先需要了解数据框的行数、列数，有哪些变量，变量的存储类型，可以用 `dplyr::glimpse()`:

```
library(tidyverse) # Wickham 的数据整理的整套工具
library(NHANES); data(NHANES)
```

```
dplyr::glimpse(NHANES)
```

```
## Rows: 10,000
## Columns: 76
## $ ID               <int> 51624, 51624, 51624, 51625, 51630, 51638, 51646, 5164~
## $ SurveyYr         <fct> 2009_10, 2009_10, 2009_10, 2009_10, 2009_10, 2009_10,~
## $ Gender           <fct> male, male, male, male, female, male, male, female, f~
## $ Age              <int> 34, 34, 34, 4, 49, 9, 8, 45, 45, 45, 66, 58, 54, 10, ~
## $ AgeDecade        <fct>  30-39,  30-39,  30-39,  0-9,  40-49,  0-9,  0-9,  40~
## $ AgeMonths        <int> 409, 409, 409, 49, 596, 115, 101, 541, 541, 541, 795,~
```

```
## $ Race1            <fct> White, White, White, Other, White, White, White, Whit~
## $ Race3            <fct> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ Education        <fct> High School, High School, High School, NA, Some Colle~
## $ MaritalStatus    <fct> Married, Married, Married, NA, LivePartner, NA, NA, M~
## $ HHIncome         <fct> 25000-34999, 25000-34999, 25000-34999, 20000-24999, 3~
## $ HHIncomeMid      <int> 30000, 30000, 30000, 22500, 40000, 87500, 60000, 8750~
## $ Poverty          <dbl> 1.36, 1.36, 1.36, 1.07, 1.91, 1.84, 2.33, 5.00, 5.00,~
## $ HomeRooms        <int> 6, 6, 6, 9, 5, 6, 7, 6, 6, 6, 5, 10, 6, 10, 10, 4, 3,~
## $ HomeOwn          <fct> Own, Own, Own, Own, Rent, Rent, Own, Own, Own, Own, O~
## $ Work             <fct> NotWorking, NotWorking, NotWorking, NA, NotWorking, N~
## $ Weight           <dbl> 87.4, 87.4, 87.4, 17.0, 86.7, 29.8, 35.2, 75.7, 75.7,~
## $ Length           <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ HeadCirc         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ Height           <dbl> 164.7, 164.7, 164.7, 105.4, 168.4, 133.1, 130.6, 166.~
## $ BMI              <dbl> 32.22, 32.22, 32.22, 15.30, 30.57, 16.82, 20.64, 27.2~
## $ BMICatUnder20yrs <fct> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ BMI_WHO          <fct> 30.0_plus, 30.0_plus, 30.0_plus, 12.0_18.5, 30.0_plus~
## $ Pulse            <int> 70, 70, 70, NA, 86, 82, 72, 62, 62, 62, 60, 62, 76, 8~
## $ BPSysAve         <int> 113, 113, 113, NA, 112, 86, 107, 118, 118, 118, 111, ~
## $ BPDiaAve         <int> 85, 85, 85, NA, 75, 47, 37, 64, 64, 64, 63, 74, 85, 6~
## $ BPSys1           <int> 114, 114, 114, NA, 118, 84, 114, 106, 106, 106, 124, ~
## $ BPDia1           <int> 88, 88, 88, NA, 82, 50, 46, 62, 62, 62, 64, 76, 86, 6~
## $ BPSys2           <int> 114, 114, 114, NA, 108, 84, 108, 118, 118, 118, 108, ~
## $ BPDia2           <int> 88, 88, 88, NA, 74, 50, 36, 68, 68, 68, 62, 72, 88, 6~
## $ BPSys3           <int> 112, 112, 112, NA, 116, 88, 106, 118, 118, 118, 114, ~
## $ BPDia3           <int> 82, 82, 82, NA, 76, 44, 38, 60, 60, 60, 64, 76, 82, 7~
## $ Testosterone     <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ DirectChol       <dbl> 1.29, 1.29, 1.29, NA, 1.16, 1.34, 1.55, 2.12, 2.12, 2~
## $ TotChol          <dbl> 3.49, 3.49, 3.49, NA, 6.70, 4.86, 4.09, 5.82, 5.82, 5~
## $ UrineVol1        <int> 352, 352, 352, NA, 77, 123, 238, 106, 106, 106, 113, ~
## $ UrineFlow1       <dbl> NA, NA, NA, NA, 0.094, 1.538, 1.322, 1.116, 1.116, 1.~
## $ UrineVol2        <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ UrineFlow2       <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ Diabetes         <fct> No, No, No, No, No, No, No, No, No, No, No, No, No, N~
## $ DiabetesAge      <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ HealthGen        <fct> Good, Good, Good, NA, Good, NA, NA, Vgood, Vgood, Vgo~
## $ DaysPhysHlthBad  <int> 0, 0, 0, NA, 0, NA, NA, 0, 0, 0, 10, 0, 4, NA, NA, 0,~
## $ DaysMentHlthBad  <int> 15, 15, 15, NA, 10, NA, NA, 3, 3, 3, 0, 0, 0, NA, NA,~
## $ LittleInterest   <fct> Most, Most, Most, NA, Several, NA, NA, None, None, No~
## $ Depressed        <fct> Several, Several, Several, NA, Several, NA, NA, None,~
```

```
## $ nPregnancies     <int> NA, NA, NA, NA, 2, NA, NA, 1, 1, 1, NA, NA, NA, NA, N~
## $ nBabies          <int> NA, NA, NA, NA, 2, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ Age1stBaby       <int> NA, NA, NA, NA, 27, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ SleepHrsNight    <int> 4, 4, 4, NA, 8, NA, NA, 8, 8, 8, 7, 5, 4, NA, 5, 7, N~
## $ SleepTrouble     <fct> Yes, Yes, Yes, NA, Yes, NA, NA, No, No, No, No, No, Y~
## $ PhysActive       <fct> No, No, No, NA, No, NA, NA, Yes, Yes, Yes, Yes, Yes, ~
## $ PhysActiveDays   <int> NA, NA, NA, NA, NA, NA, NA, 5, 5, 5, 7, 5, 1, NA, 2, ~
## $ TVHrsDay         <fct> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ CompHrsDay       <fct> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ TVHrsDayChild    <int> NA, NA, NA, 4, NA, 5, 1, NA, NA, NA, NA, NA, NA, 4, N~
## $ CompHrsDayChild  <int> NA, NA, NA, 1, NA, 0, 6, NA, NA, NA, NA, NA, NA, 3, N~
## $ Alcohol12PlusYr  <fct> Yes, Yes, Yes, NA, Yes, NA, NA, Yes, Yes, Yes, Yes, Y~
## $ AlcoholDay       <int> NA, NA, NA, NA, 2, NA, NA, 3, 3, 3, 1, 2, 6, NA, NA, ~
## $ AlcoholYear      <int> 0, 0, 0, NA, 20, NA, NA, 52, 52, 52, 100, 104, 364, N~
## $ SmokeNow         <fct> No, No, No, NA, Yes, NA, NA, NA, NA, NA, No, NA, NA, ~
## $ Smoke100         <fct> Yes, Yes, Yes, NA, Yes, NA, NA, No, No, No, Yes, No, ~
## $ Smoke100n        <fct> Smoker, Smoker, Smoker, NA, Smoker, NA, NA, Non-Smoke~
## $ SmokeAge         <int> 18, 18, 18, NA, 38, NA, NA, NA, NA, NA, 13, NA, NA, N~
## $ Marijuana        <fct> Yes, Yes, Yes, NA, Yes, NA, NA, Yes, Yes, Yes, NA, Ye~
## $ AgeFirstMarij    <int> 17, 17, 17, NA, 18, NA, NA, 13, 13, 13, NA, 19, 15, N~
## $ RegularMarij     <fct> No, No, No, NA, No, NA, NA, No, No, No, NA, Yes, Yes,~
## $ AgeRegMarij      <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 20, 15, N~
## $ HardDrugs        <fct> Yes, Yes, Yes, NA, Yes, NA, NA, No, No, No, No, Yes, ~
## $ SexEver          <fct> Yes, Yes, Yes, NA, Yes, NA, NA, Yes, Yes, Yes, Yes, Y~
## $ SexAge           <int> 16, 16, 16, NA, 12, NA, NA, 13, 13, 13, 17, 22, 12, N~
## $ SexNumPartnLife  <int> 8, 8, 8, NA, 10, NA, NA, 20, 20, 20, 15, 7, 100, NA, ~
## $ SexNumPartYear   <int> 1, 1, 1, NA, 1, NA, NA, 0, 0, 0, NA, 1, 1, NA, NA, 1,~
## $ SameSex          <fct> No, No, No, NA, Yes, NA, NA, Yes, Yes, Yes, No, No, N~
## $ SexOrientation   <fct> Heterosexual, Heterosexual, Heterosexual, NA, Heteros~
## $ PregnantNow      <fct> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
```

结果中 `<int>` 表示取整数值，`<dbl>` 表示取浮点型值，`<fct>` 表示因子，`<chr>` 表示字符串类型。

NHANES 数据中包含了两次考察的结果，所以每个受访者可能有两个观测，少数受访者仅有一个，也有的受访者有多个观测。这样会使得变量的统计分析结果不易解释，因此我们仅筛选其中 `SurveyYr` 为 `2011_12` 的观测，并对每个 ID 仅保留最后一个观测，生成数据框 `d_nhsub`:

```
d_nhsub <- NHANES |>
  dplyr::filter(SurveyYr == "2011_12") |>
  group_by(ID) |>
  slice_tail(n=1) |>
```

```
  ungroup()
dim(d_nhsub)
```

```
## [1] 3211   76
```

29.1 单个分类变量分布

我们筛选出 d_nhsub 数据集中因子型的变量：

```
d_nhsub |>
  map_lgl(is.factor) |>
  keep(identity) |>
  names()
```

```
##  [1] "SurveyYr"         "Gender"           "AgeDecade"        "Race1"
##  [5] "Race3"            "Education"        "MaritalStatus"    "HHIncome"
##  [9] "HomeOwn"          "Work"             "BMICatUnder20yrs" "BMI_WHO"
## [13] "Diabetes"         "HealthGen"        "LittleInterest"   "Depressed"
## [17] "SleepTrouble"     "PhysActive"       "TVHrsDay"         "CompHrsDay"
## [21] "Alcohol12PlusYr"  "SmokeNow"         "Smoke100"         "Smoke100n"
## [25] "Marijuana"        "RegularMarij"     "HardDrugs"        "SexEver"
## [29] "SameSex"          "SexOrientation"   "PregnantNow"
```

以 Education 变量为例，计算其频数和百分比如下：

```
d_nhsub |>
  count(Education) |>
  mutate(
    pct = n / sum(n)
  )
```

```
## # A tibble: 6 x 3
##   Education          n    pct
##   <fct>          <int>  <dbl>
## 1 8th Grade        163 0.0508
## 2 9 - 11th Grade   268 0.0835
## 3 High School      421 0.131
## 4 Some College     695 0.216
## 5 College Grad     624 0.194
## 6 <NA>            1040 0.324
```

在数据清理阶段对分类变量制作频数表，可以发现输入错的类别值。

用基本 R 的 table 等函数统计：

```
table(d_nhsub[["Education"]])
```

```
##
##      8th Grade 9 - 11th Grade    High School   Some College   College Grad
##            163            268            421            695            624
```

base::table() 默认忽略缺失值，这在指标显示时会有误导。加选项 useNA="ifany"：

```
table(d_nhsub[["Education"]],
  useNA = "ifany")
```

```
##
##      8th Grade 9 - 11th Grade    High School   Some College   College Grad
##            163            268            421            695            624
##           <NA>
##           1040
```

百分比表：

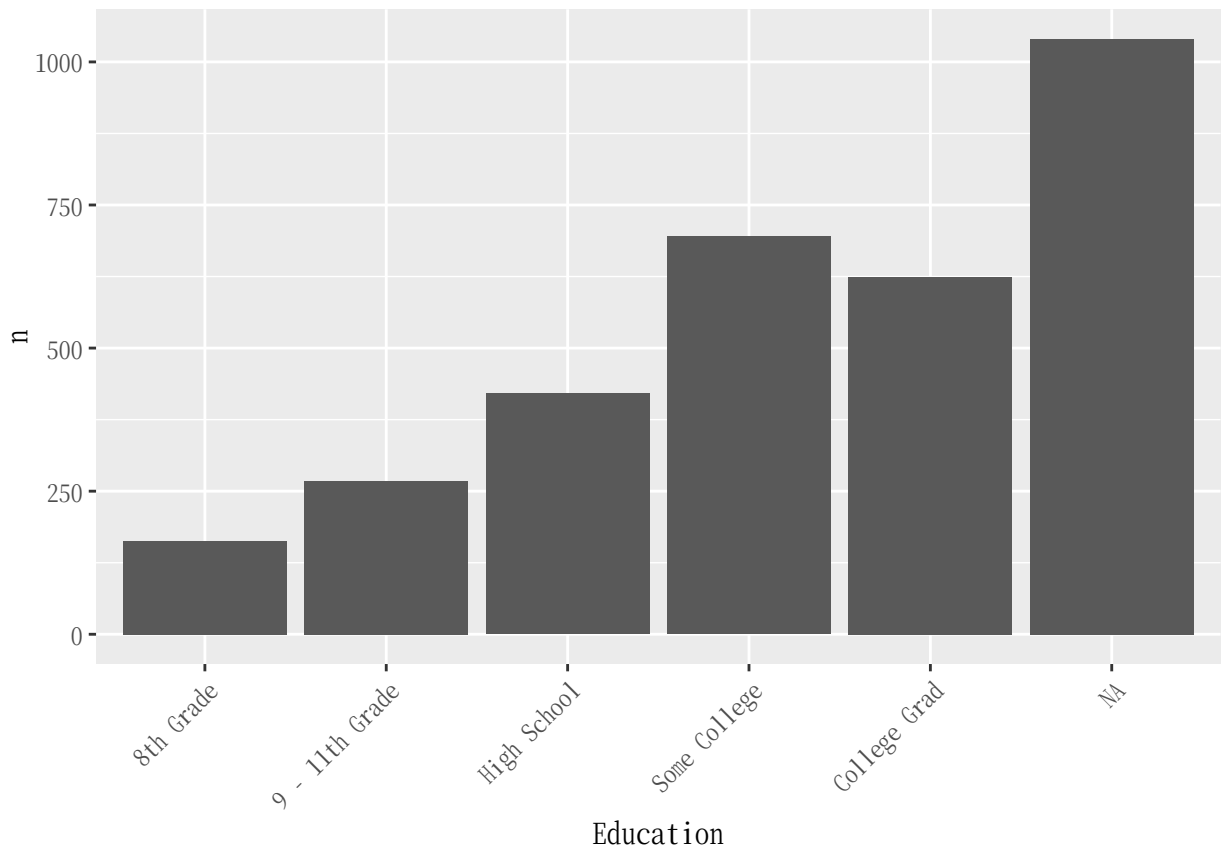
```
table(NHANES[["Education"]],
  useNA = "ifany") |>
  prop.table()
```

```
##
##      8th Grade 9 - 11th Grade    High School   Some College   College Grad
##         0.0451         0.0888         0.1517         0.2267         0.2098
##           <NA>
##         0.2779
```

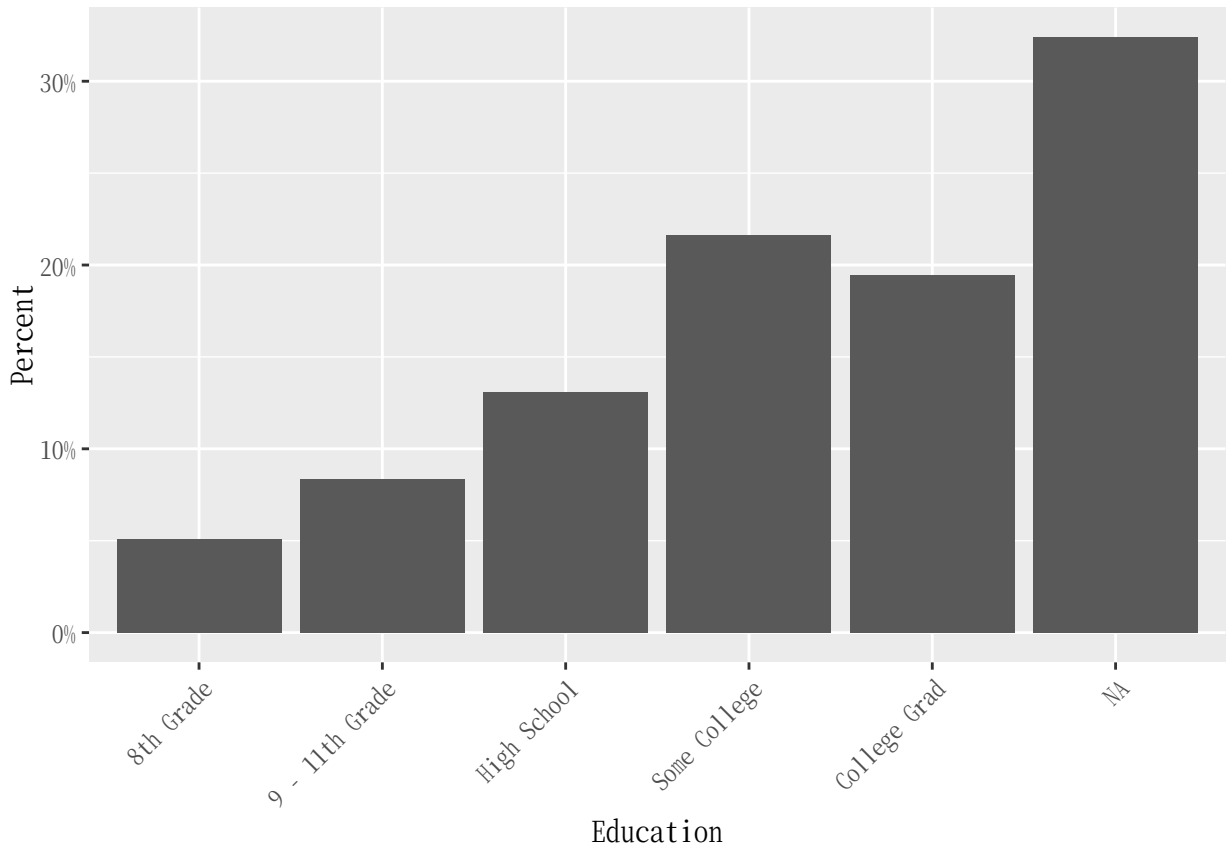
可以直接用 ggplot2::geom_bar() 制作频数条形图，可以用 geom_col() 基于统计的频数表制作频数或者百分比的条形图，如：

```
d_edu <- d_nhsub |>
  count(Education) |>
  mutate(
    pct = n / sum(n)
  )
```

```
ggplot(d_edu, aes(x = Education, y=n)) +
  geom_col() +
  theme(
    axis.text.x = element_text(
      angle = 45, vjust = 1, hjust = 1)  )
```




```
ggplot(d_edu, aes(x = Education, y=pct)) +
  geom_col() +
  scale_y_continuous(
    name = "Percent",
    labels = scales::percent) +
  theme(
    axis.text.x = element_text(
      angle = 45, vjust = 1, hjust = 1)  )
```



29.2 单个区间型变量分布

d_nhsub 中所有浮点型值的变量：

```
d_nhsub |>
  map_lgl(is.double) |>
  keep(identity) |>
  names()
```

```
##  [1] "Poverty"      "Weight"       "Length"       "HeadCirc"     "Height"
##  [6] "BMI"          "Testosterone" "DirectChol"   "TotChol"      "UrineFlow1"
## [11] "UrineFlow2"
```

所有整数值的变量：

```
d_nhsub |>
  map_lgl(is.integer) |>
  keep(identity) |>
  names()
```

```
##  [1] "ID"              "Age"             "AgeMonths"       "HHIncomeMid"
##  [5] "HomeRooms"       "Pulse"           "BPSysAve"        "BPDiaAve"
##  [9] "BPSys1"          "BPDia1"          "BPSys2"          "BPDia2"
## [13] "BPSys3"          "BPDia3"          "UrineVol1"       "UrineVol2"
## [17] "DiabetesAge"     "DaysPhysHlthBad" "DaysMentHlthBad" "nPregnancies"
## [21] "nBabies"         "Age1stBaby"      "SleepHrsNight"   "PhysActiveDays"
## [25] "TVHrsDayChild"   "CompHrsDayChild" "AlcoholDay"      "AlcoholYear"
## [29] "SmokeAge"        "AgeFirstMarij"   "AgeRegMarij"     "SexAge"
## [33] "SexNumPartnLife" "SexNumPartYear"
```

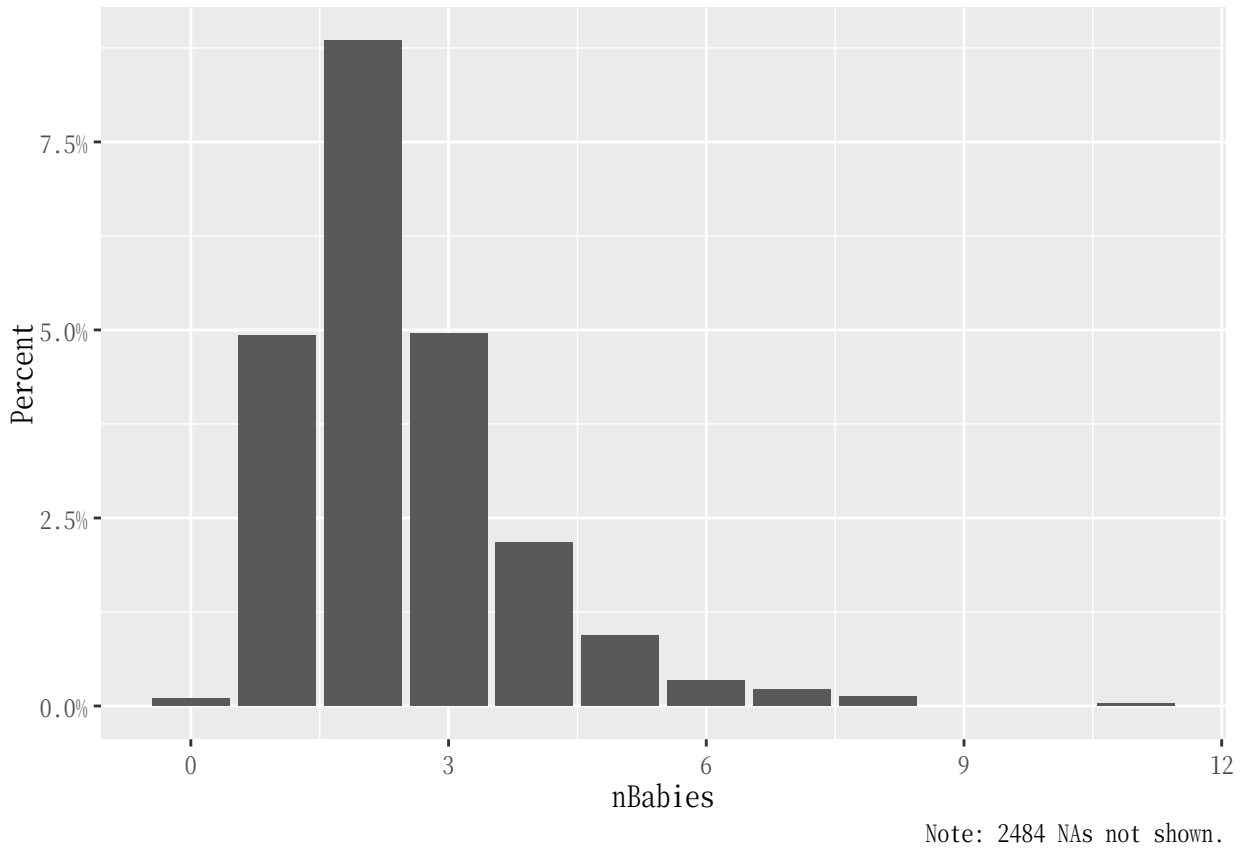
整数值的变量兼有区间型和分类变量的特点。例如，子女数 (nBabies)：

```
d_babies <- d_nhsub |>
  count(nBabies) |>
  mutate(pct = n / sum(n))
knitr::kable(d_babies)
```

nBabies	n	pct
0	3	0.0009343
1	158	0.0492059
2	284	0.0884460
3	159	0.0495173
4	70	0.0218001
5	30	0.0093429
6	11	0.0034257
7	7	0.0021800
8	4	0.0012457
11	1	0.0003114
NA	2484	0.7735908

```
ggplot(d_babies, aes(x = nBabies, y=pct)) +
  geom_col() +
  scale_y_continuous(
    name = "Percent",
    labels = scales::percent) +
  labs(caption="Note: 2484 NAs not shown.")
```

```
## Warning: Removed 1 rows containing missing values (position_stack).
```



变量 Height 是 2 岁以上受访者的身高（单位：厘米）测量值。可以用基本 R 的 `summary()` 函数显示“五数概括”和平均值：

```
summary(d_nhsub[["Height"]])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    83.8   154.8   164.7   159.7   173.2   200.4     133
```

这说明有 133 个缺失值，在非缺失值中，最小为 83.8，最大为 200.4，中位数是 164.7，平均身高为 159.7。有四分之一的人身高低于 154.8，有四分之一的人身高超过 173.2。

可以用 `dplyr::summarize()` 计算指定的统计量，如：

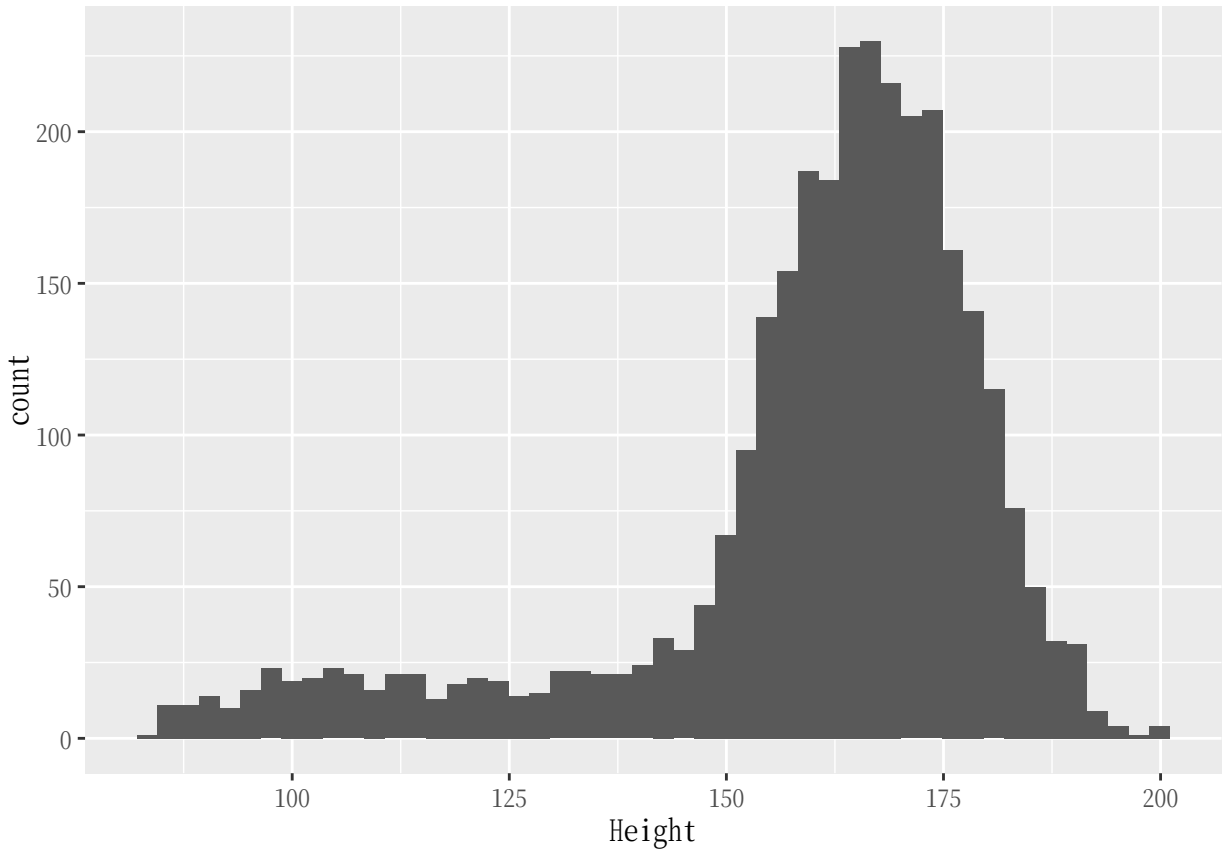
```
d_nhsub |>
  summarize(
    n = n(),
    mean = mean(Height, na.rm=TRUE),
    median = median(Height, na.rm=TRUE),
    std = sd(Height, na.rm=TRUE)
  )
```

```
## # A tibble: 1 x 4
##       n  mean median   std
##   <int> <dbl>  <dbl> <dbl>
## 1  3211  160.   165.  21.5
```

用直方图反映分布情况：

```
ggplot(d_nhsub, aes(x = Height)) +
  geom_histogram(bins=50)
```

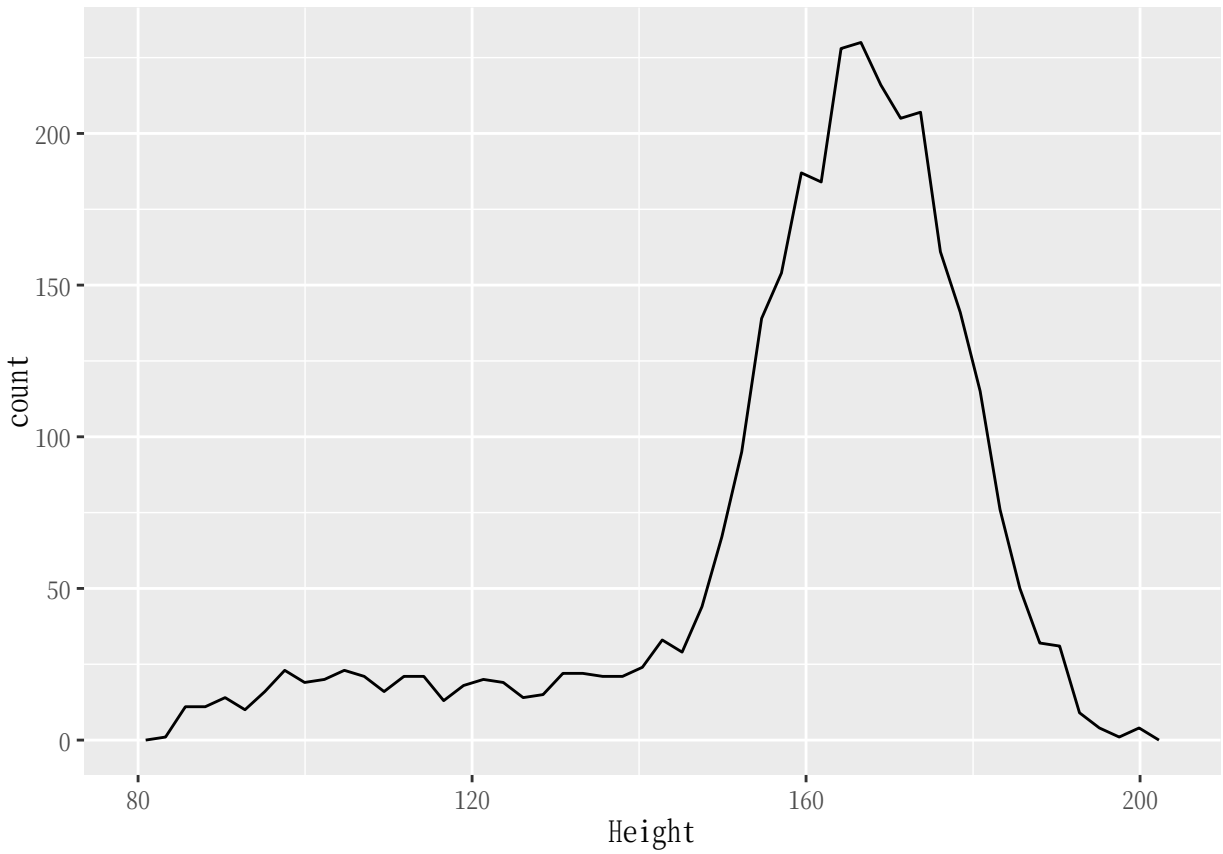
```
## Warning: Removed 133 rows containing non-finite values (stat_bin).
```



可以用折线表示直方图：

```
ggplot(d_nhsub, aes(x = Height)) +
  geom_freqpoly(bins=50)
```

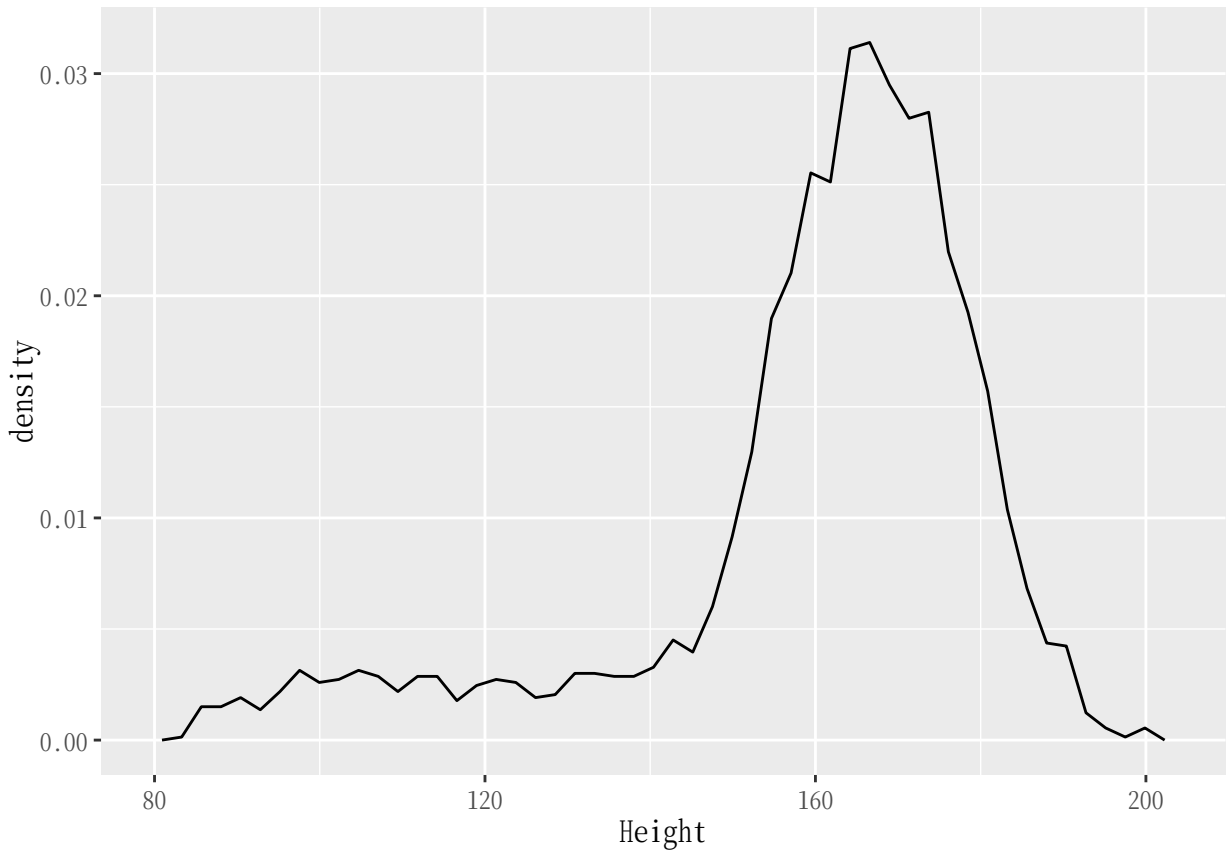
```
## Warning: Removed 133 rows containing non-finite values (stat_bin).
```



以上两个图的纵坐标是频数，将纵坐标改为密度估计：

```
ggplot(d_nhsub, aes(x = Height, y = ..density..)) +
  geom_freqpoly(bins=50)
```

```
## Warning: Removed 133 rows containing non-finite values (stat_bin).
```

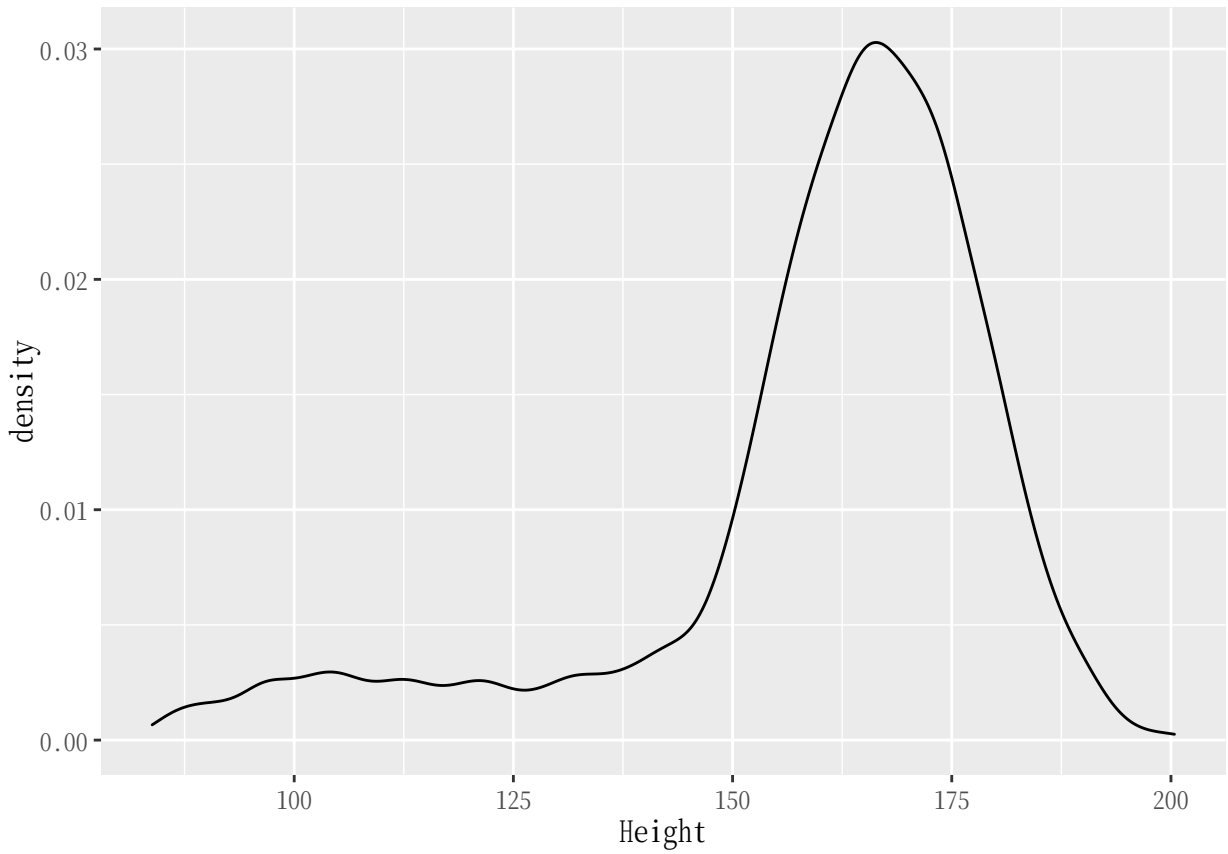


上面的 `y = ..density..` 可以改为 `y = after_stat(density)`。

作身高的核密度估计图：

```
ggplot(d_nhsub, aes(x = Height)) +
  geom_density()
```

```
## Warning: Removed 133 rows containing non-finite values (stat_density).
```

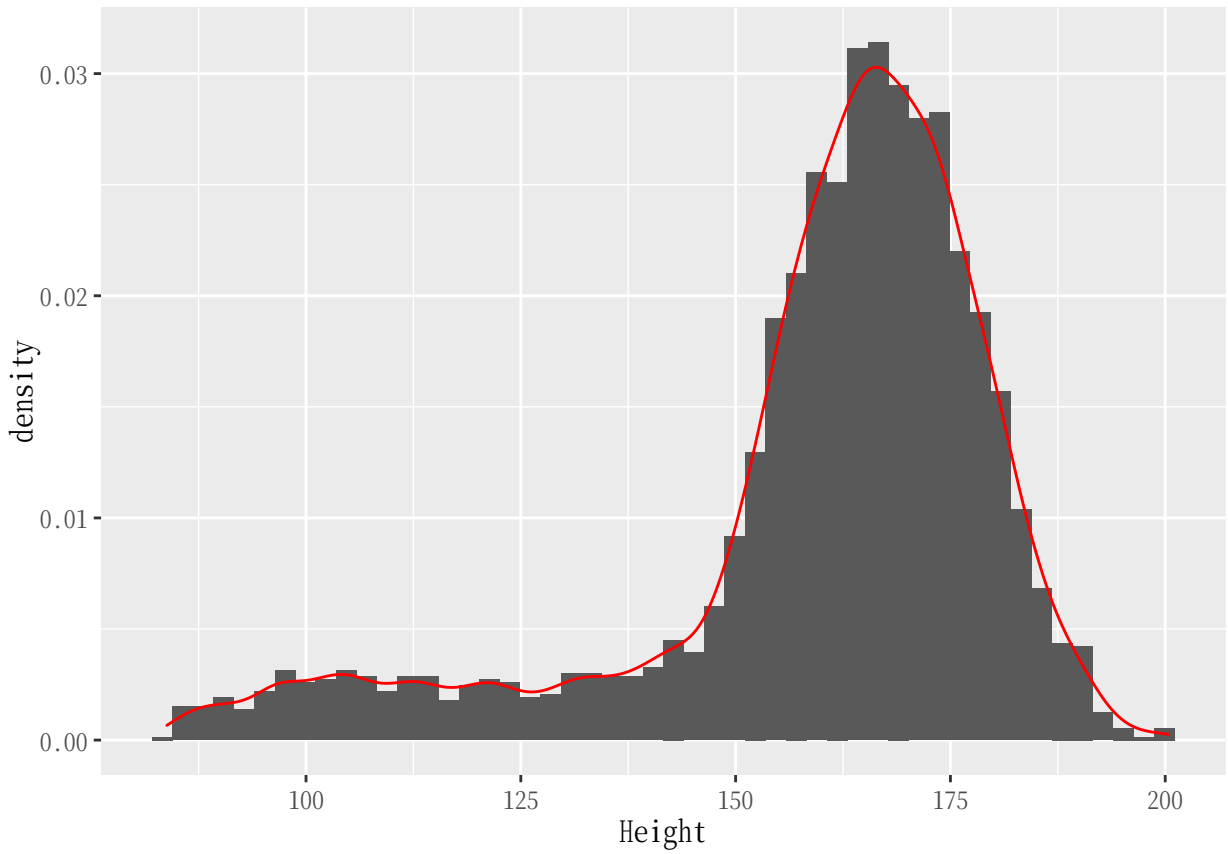


将直方图与核密度估计图叠加：

```
ggplot(d_nhsub, aes(x = Height)) +
  geom_histogram(
    aes(y = ..density..),
    bins=50) +
  geom_density(color="red")
```

```
## Warning: Removed 133 rows containing non-finite values (stat_bin).
```

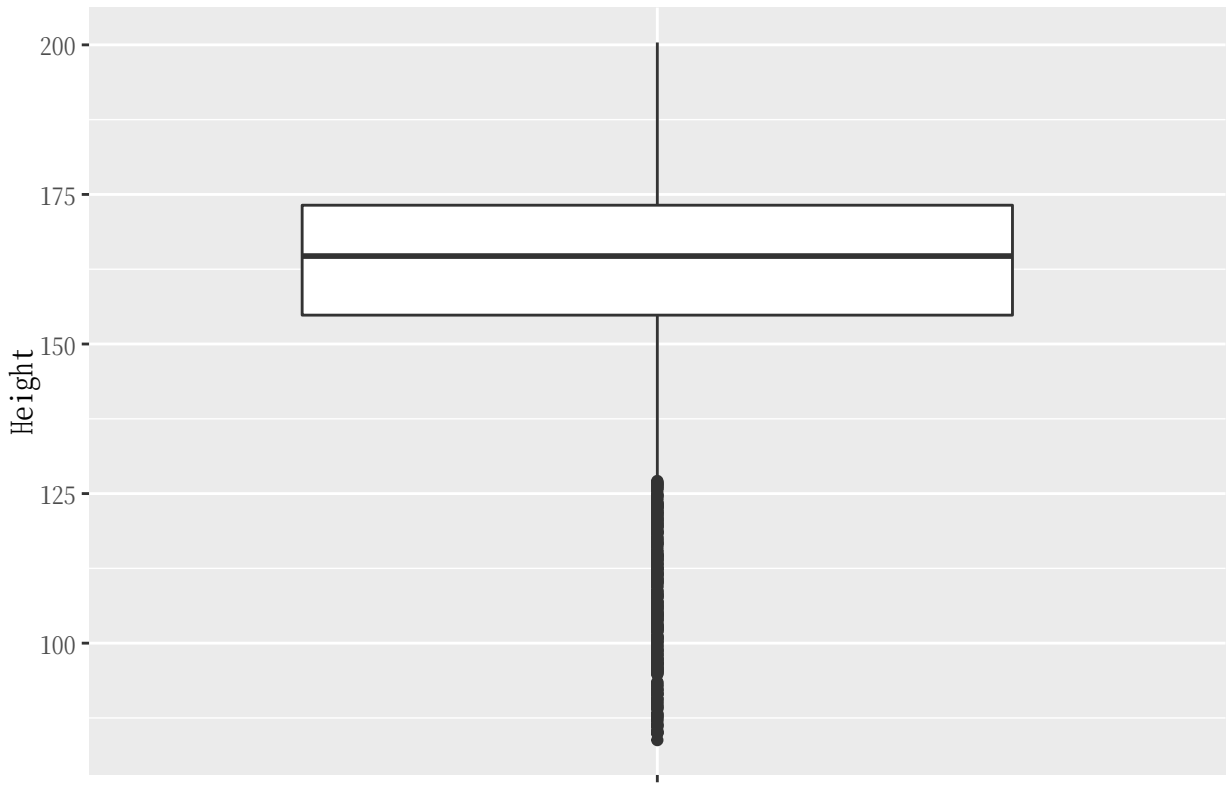
```
## Warning: Removed 133 rows containing non-finite values (stat_density).
```



分布也可以用盒形图、小提琴图表示：

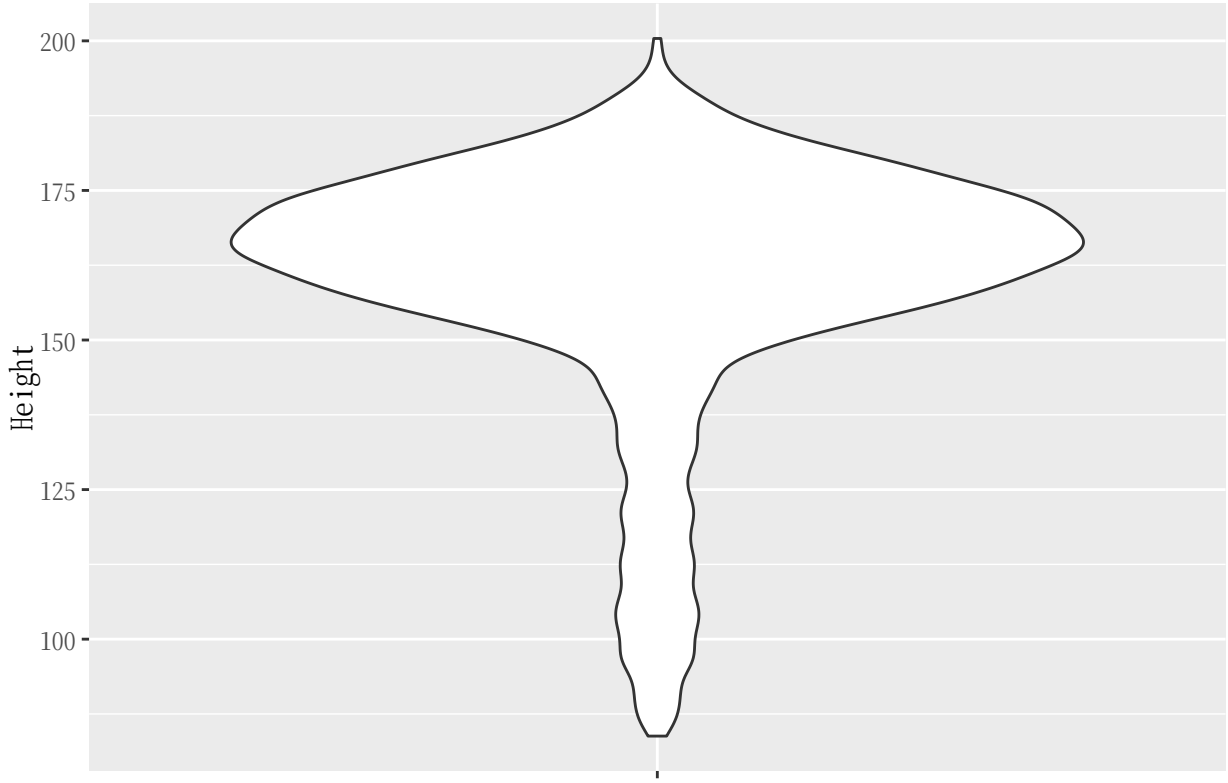
```
ggplot(d_nhsub, aes(x = "", y = Height)) +
  geom_boxplot() +
  labs(x = "")
```

```
## Warning: Removed 133 rows containing non-finite values (stat_boxplot).
```




```
ggplot(d_nhsub, aes(x = "", y = Height)) +
  geom_violin() +
  labs(x = "")
```

```
## Warning: Removed 133 rows containing non-finite values (stat_ydensity).
```

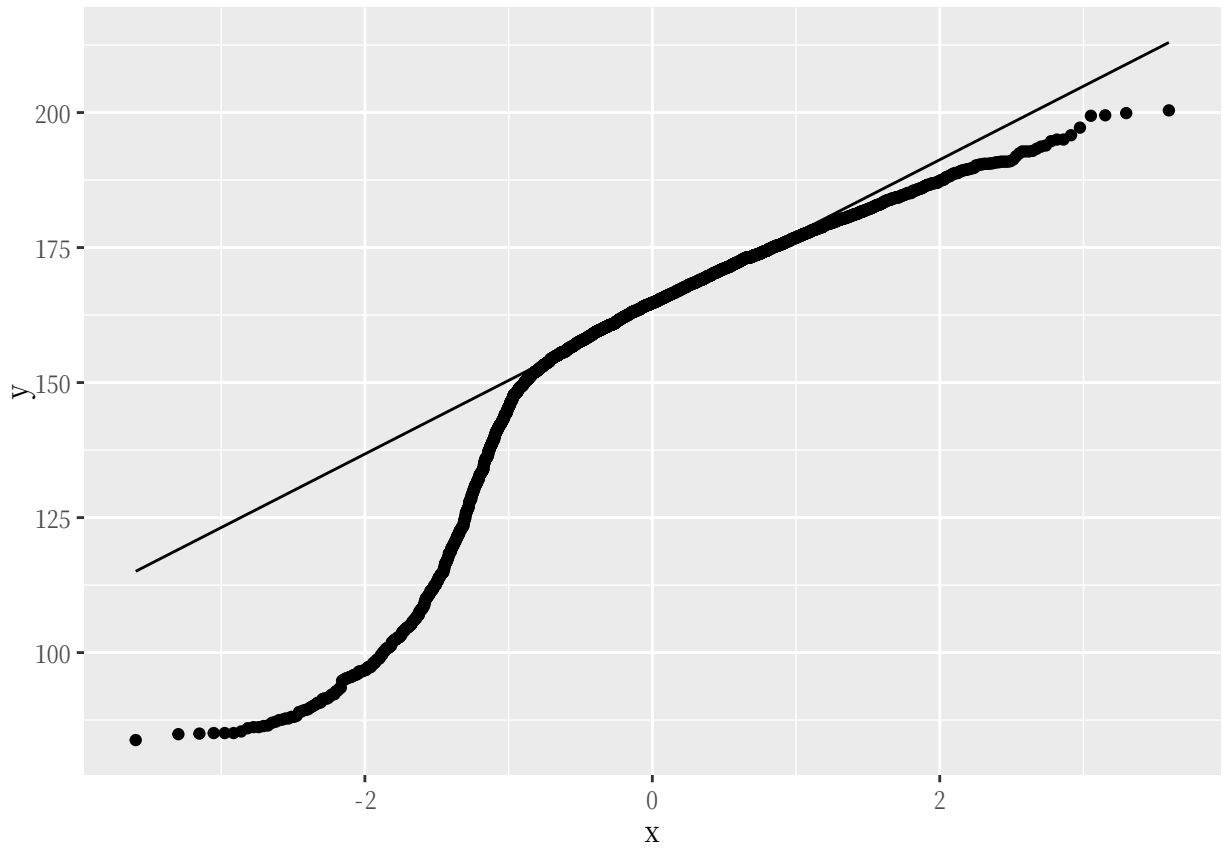


正态 QQ 图：

```
ggplot(d_nhsub, aes(sample = Height)) +
  geom_qq() +
  geom_qq_line()
```

```
## Warning: Removed 133 rows containing non-finite values (stat_qq).
```

```
## Warning: Removed 133 rows containing non-finite values (stat_qq_line).
```

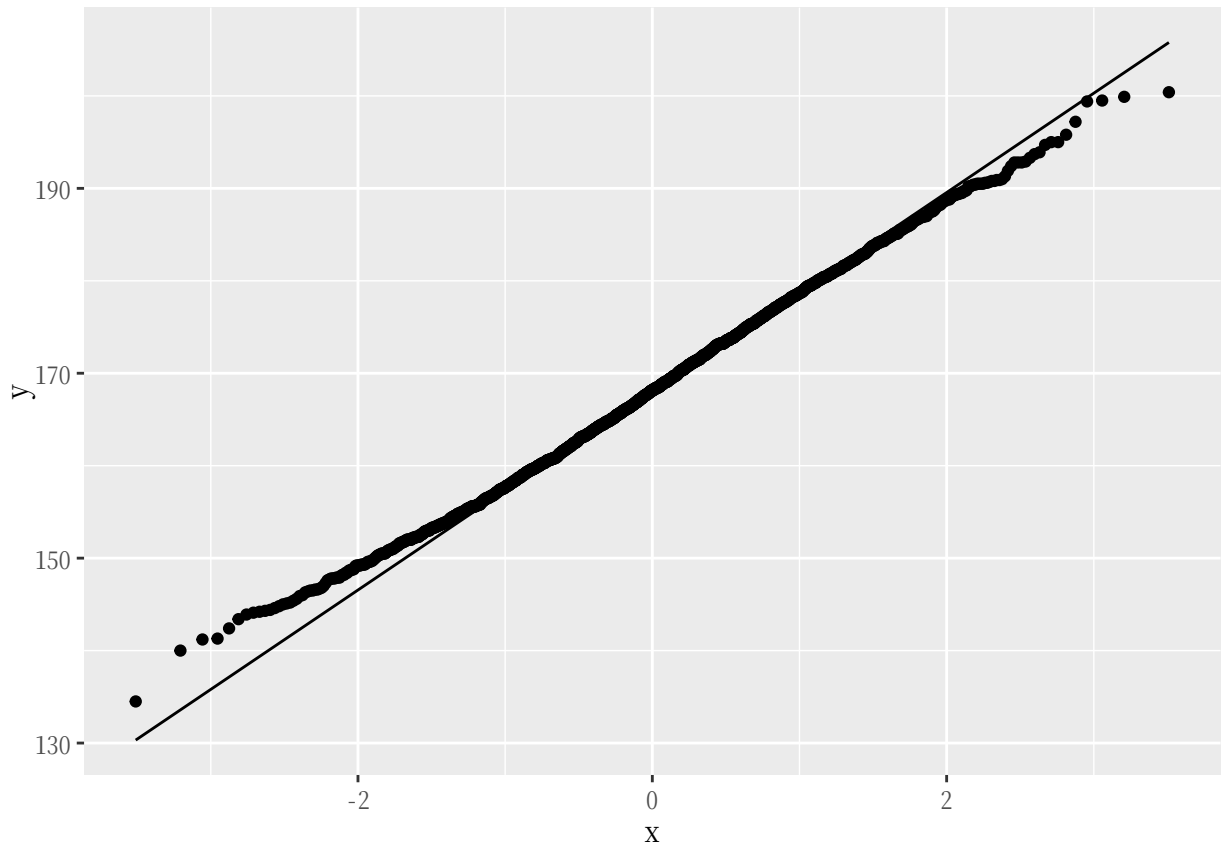


这是明显的左偏分布。这是由未成年与成年的分布差异造成的。选择年龄在 18 岁及以上的部分：

```
d_nhsub |>
  dplyr::filter(Age >= 18) |>
ggplot(aes(sample = Height)) +
  geom_qq() +
  geom_qq_line()
```

```
## Warning: Removed 22 rows containing non-finite values (stat_qq).
```

```
## Warning: Removed 22 rows containing non-finite values (stat_qq_line).
```



29.3 用 skimr 进行简单概括

skimr 扩展包可以对一个数据集的所有变量进行简单概括，分为分类变量和数值型变量两组，如：

```
library(skimr)
```

```
## Warning: package 'skimr' was built under R version 4.2.2
```

```
skim(d_nhsub, Gender, Age, Height) |> print()
```

```
## -- Data Summary ------------------------
##                           Values
## Name                      d_nhsub
## Number of rows            3211
## Number of columns         76
## _______________________
## Column type frequency:
##   factor                  1
```

```
##   numeric                  2
## ________________________
## Group variables            None
##
## -- Variable type: factor ------------------------------------------------------
##   skim_variable n_missing complete_rate ordered n_unique top_counts
## 1 Gender                0             1 FALSE          2 fem: 1621, mal: 1590
##
## -- Variable type: numeric -----------------------------------------------------
##   skim_variable n_missing complete_rate  mean   sd   p0  p25  p50  p75 p100
## 1 Age                   0         1      35.1 23.3  0    14   33   54   80
## 2 Height              133         0.959 160.  21.5 83.8 155. 165. 173. 200.
##   hist
## 1
## 2
```

对于分类变量 Gender，概括内容包括缺失观测数 (missing)，非缺失观测数 (complete)，总观测数 (n)，不同值个数 (n_unique)，取值频数最多的典型值及其频数以及缺失个数 (top_counts)，是否有序因子 (ordered)。

对于连续型（区间型）变量 Age 和 Height，也包括了缺失观测数、非缺失观测数、总观测数，另外给出了样本均值、标准差和 “五数概括”，即最小值 (p0)，最大值 (p100)，中位数 (p50)，四分位数 (p25 和 p75)。

29.4 两个分类变量之间的关系

考察两个分类变量：性别 (Gender) 和教育程度 (Education) 之间的关系。先计算交叉频数分布：

```
d_gened <- d_nhsub |>
  count(Gender, Education) |>
  group_by(Gender) |>
  mutate(in_gend_pct = n / sum(n)) |>
  ungroup()
knitr::kable(d_gened)
```

Gender	Education	n	in_gend_pct
female	8th Grade	68	0.0419494
female	9 - 11th Grade	120	0.0740284
female	High School	215	0.1326342
female	Some College	388	0.2393584
female	College Grad	330	0.2035780
female	NA	500	0.3084516
male	8th Grade	95	0.0597484
male	9 - 11th Grade	148	0.0930818
male	High School	206	0.1295597
male	Some College	307	0.1930818
male	College Grad	294	0.1849057
male	NA	540	0.3396226

其中 `in_gend_pct` 是在性别内部的学历比例。

或：

```
with(d_nhsub, table(
  Education, Gender
))
```

```
##                 Gender
## Education        female male
##   8th Grade          68   95
##   9 - 11th Grade    120  148
##   High School       215  206
##   Some College      388  307
##   College Grad      330  294
```

更详细的列联表：

```
with(d_nhsub, gmodels::CrossTable(
  Education, Gender
))
```

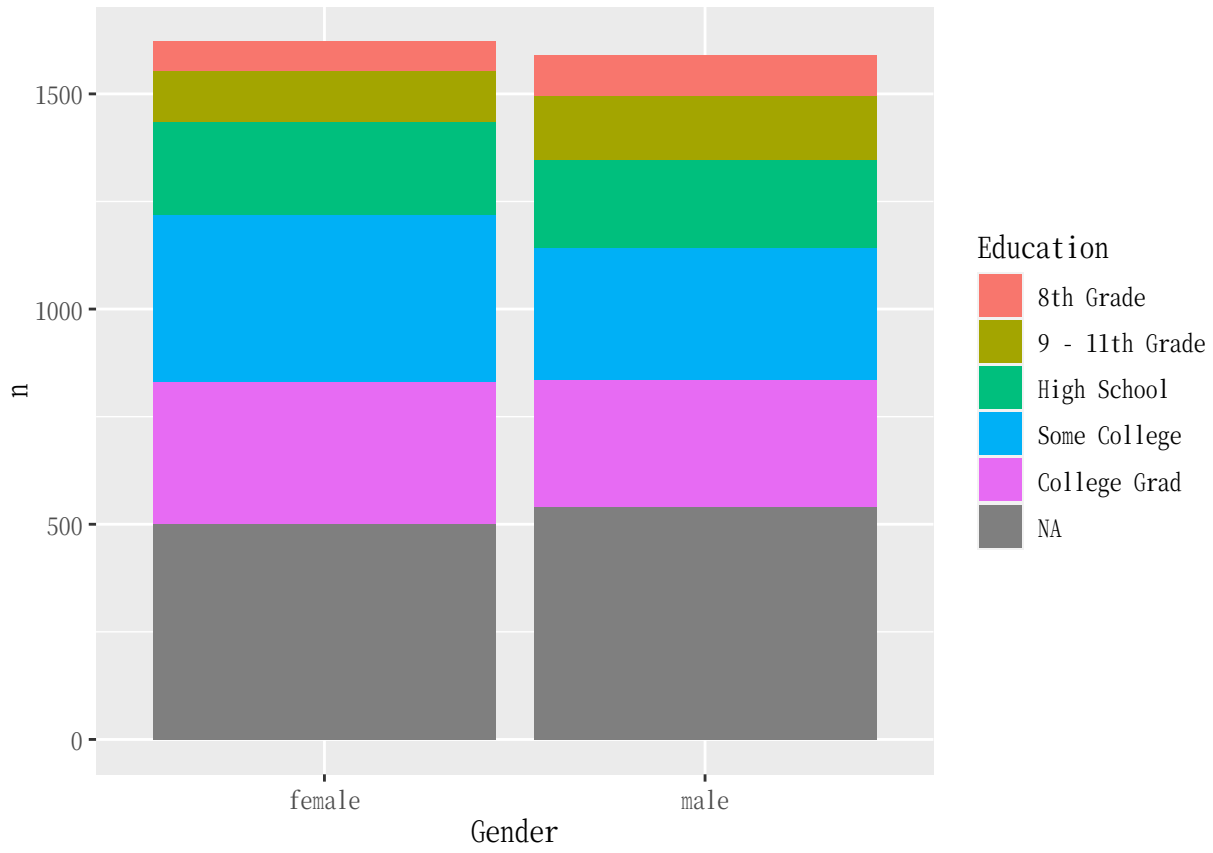
```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
```

```
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  2171
##
##
##                 | Gender
##      Education |    female |      male | Row Total |
## ---------------|-----------|-----------|-----------|
##      8th Grade |        68 |        95 |       163 |
##                |     3.105 |     3.315 |           |
##                |     0.417 |     0.583 |     0.075 |
##                |     0.061 |     0.090 |           |
##                |     0.031 |     0.044 |           |
## ---------------|-----------|-----------|-----------|
## 9 - 11th Grade |       120 |       148 |       268 |
##                |     2.442 |     2.607 |           |
##                |     0.448 |     0.552 |     0.123 |
##                |     0.107 |     0.141 |           |
##                |     0.055 |     0.068 |           |
## ---------------|-----------|-----------|-----------|
##    High School |       215 |       206 |       421 |
##                |     0.026 |     0.028 |           |
##                |     0.511 |     0.489 |     0.194 |
##                |     0.192 |     0.196 |           |
##                |     0.099 |     0.095 |           |
## ---------------|-----------|-----------|-----------|
##   Some College |       388 |       307 |       695 |
##                |     2.365 |     2.525 |           |
##                |     0.558 |     0.442 |     0.320 |
##                |     0.346 |     0.292 |           |
##                |     0.179 |     0.141 |           |
## ---------------|-----------|-----------|-----------|
##   College Grad |       330 |       294 |       624 |
##                |     0.189 |     0.201 |           |
##                |     0.529 |     0.471 |     0.287 |
```

```
##                 |     0.294 |     0.280 |           |
##                 |     0.152 |     0.135 |           |
## ----------------|-----------|-----------|-----------|
##   Column Total  |      1121 |      1050 |      2171 |
##                 |     0.516 |     0.484 |           |
## ----------------|-----------|-----------|-----------|
##
##
```

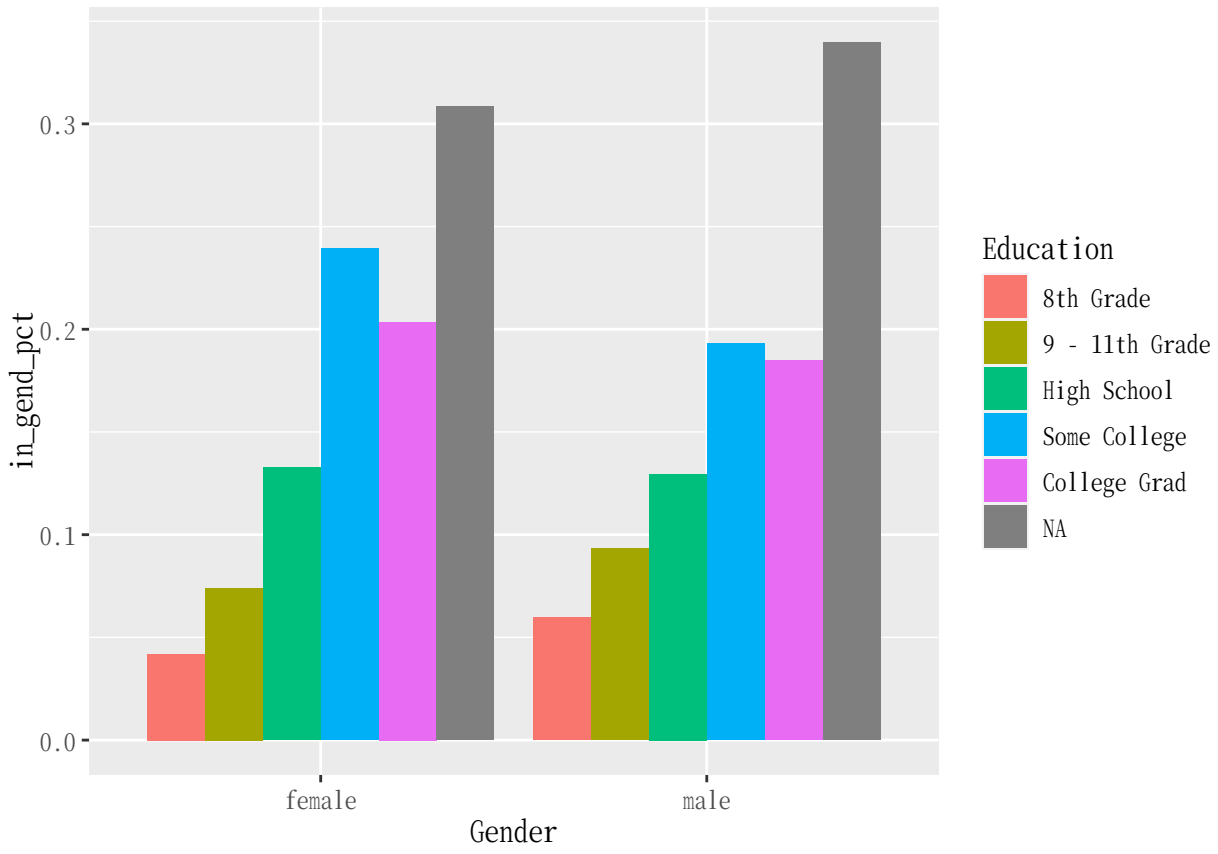
比较男女之间教育程度的堆叠条形图：

```
ggplot(d_gened, aes(
  x = Gender, fill = Education, y = n)) +
  geom_col(position = "stack")
```



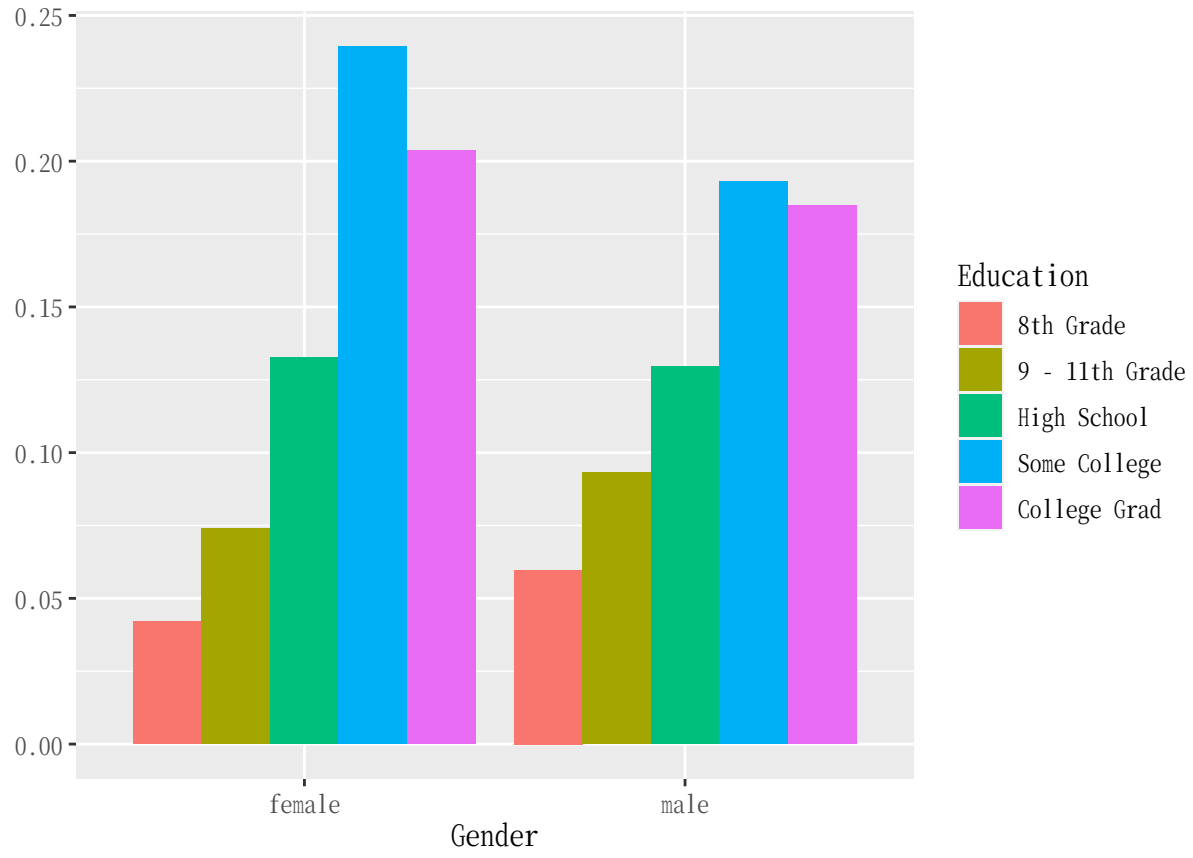
并列条形图，使用百分比比较：

```
ggplot(d_gened, aes(
  x = Gender, fill = Education, y = in_gend_pct)) +
  geom_col(position = "dodge")
```



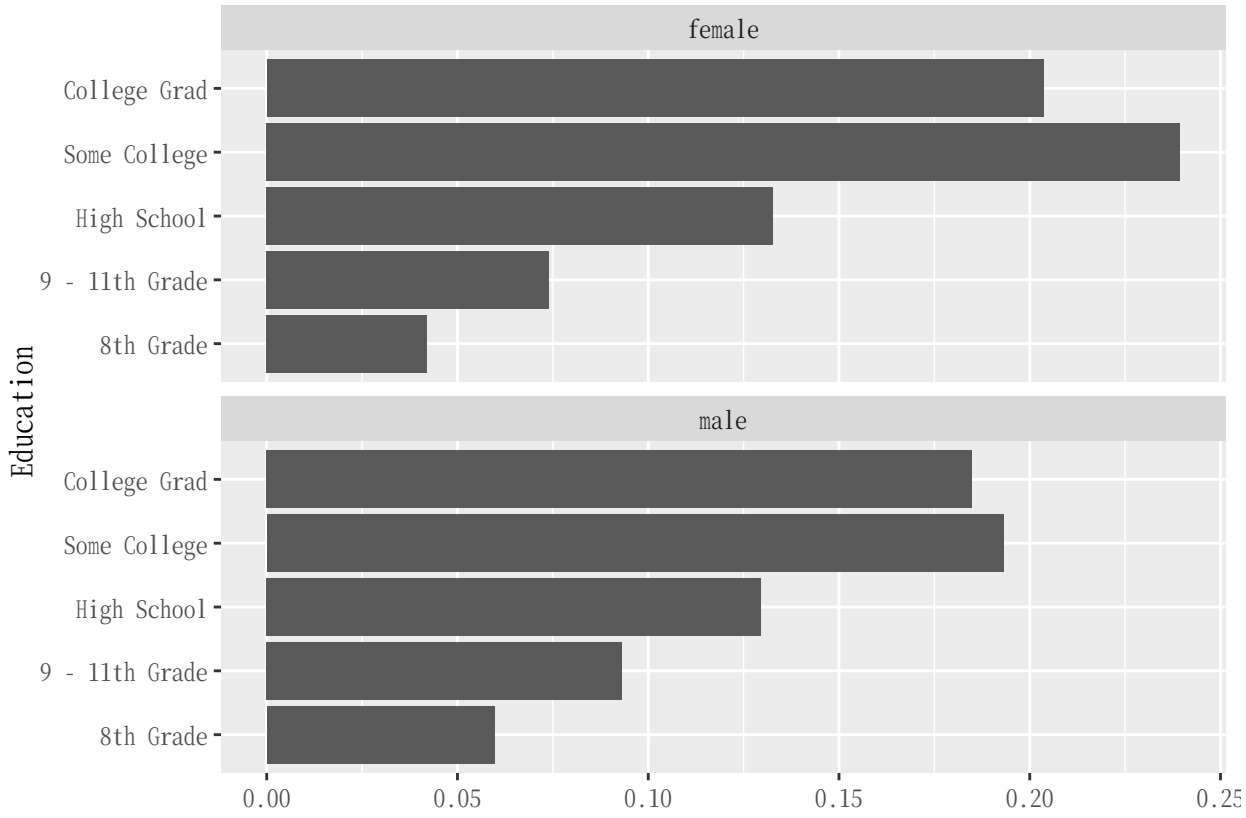
不显示缺失值：

```
ggplot(drop_na(d_gened), aes(
  x = Gender, fill = Education, y = in_gend_pct)) +
  geom_col(position = "dodge") +
  labs(y="")
```



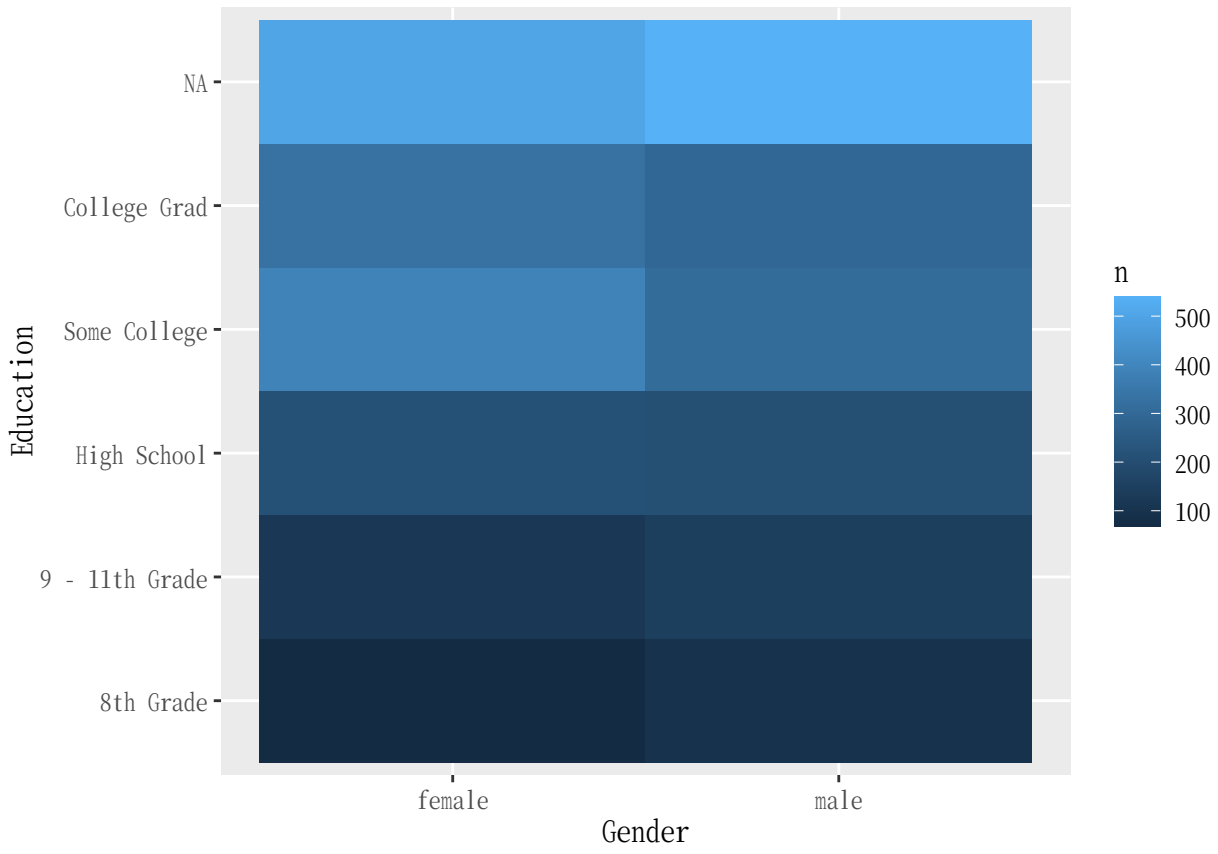
分两组比较分布：

```
ggplot(drop_na(d_gened), aes(
  x = Education, y = in_gend_pct)) +
  facet_wrap(~ Gender, ncol=1) +
  geom_col() +
  coord_flip() +
  labs(y="")
```



较复杂的频数表，也可以看成一个矩阵，用色块图表示频数，如：

```
ggplot(d_gened, aes(
  x = Gender, y = Education, fill = n)) +
  geom_tile()
```



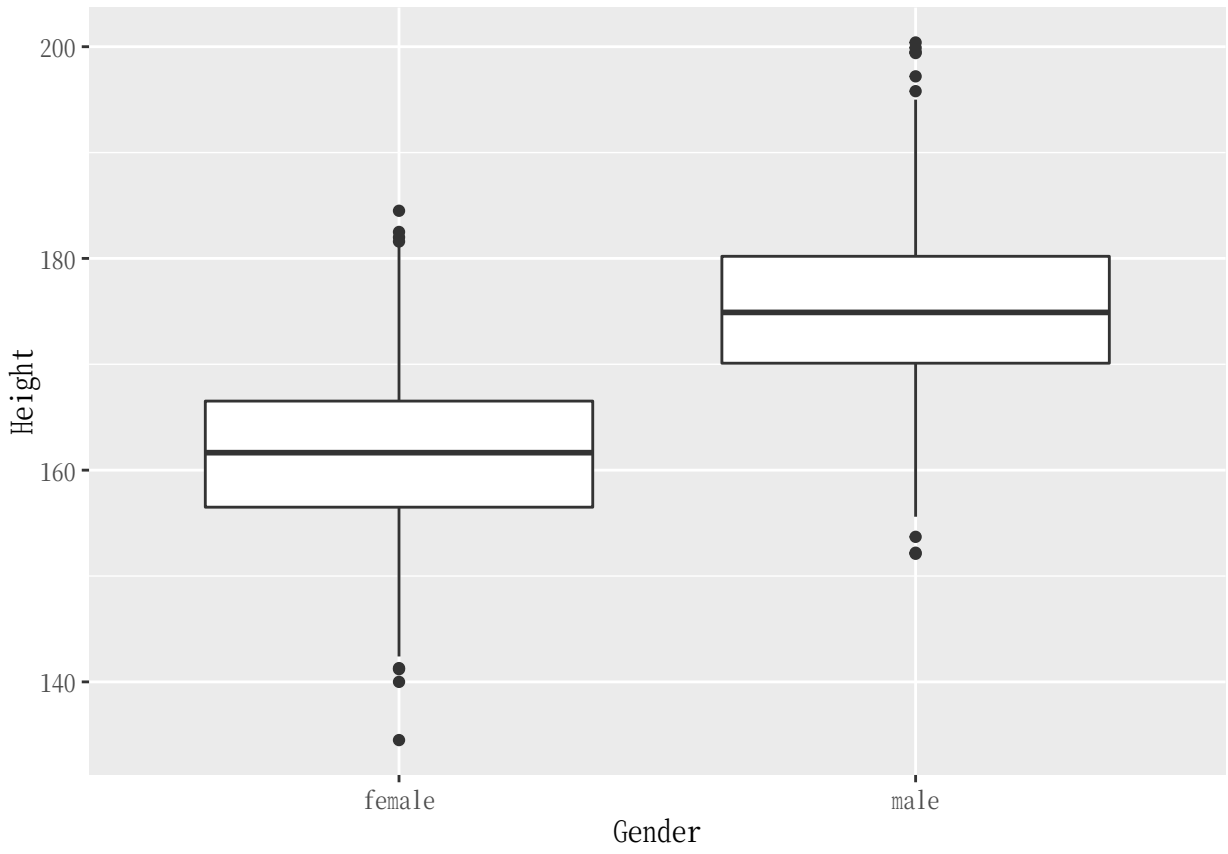
29.5 一个分类变量与一个区间型变量之间的关系

考虑不同性别的身高的比较。我们选择 18 岁以上的样本。

男、女身高的并列盒形图：

```
d_genht <- d_nhsub |>
  dplyr::filter(Age >= 18) |>
  select(Gender, Height, Age, Weight, BMI)
ggplot(d_genht, aes(
  x = Gender, y = Height)) +
  geom_boxplot()
```

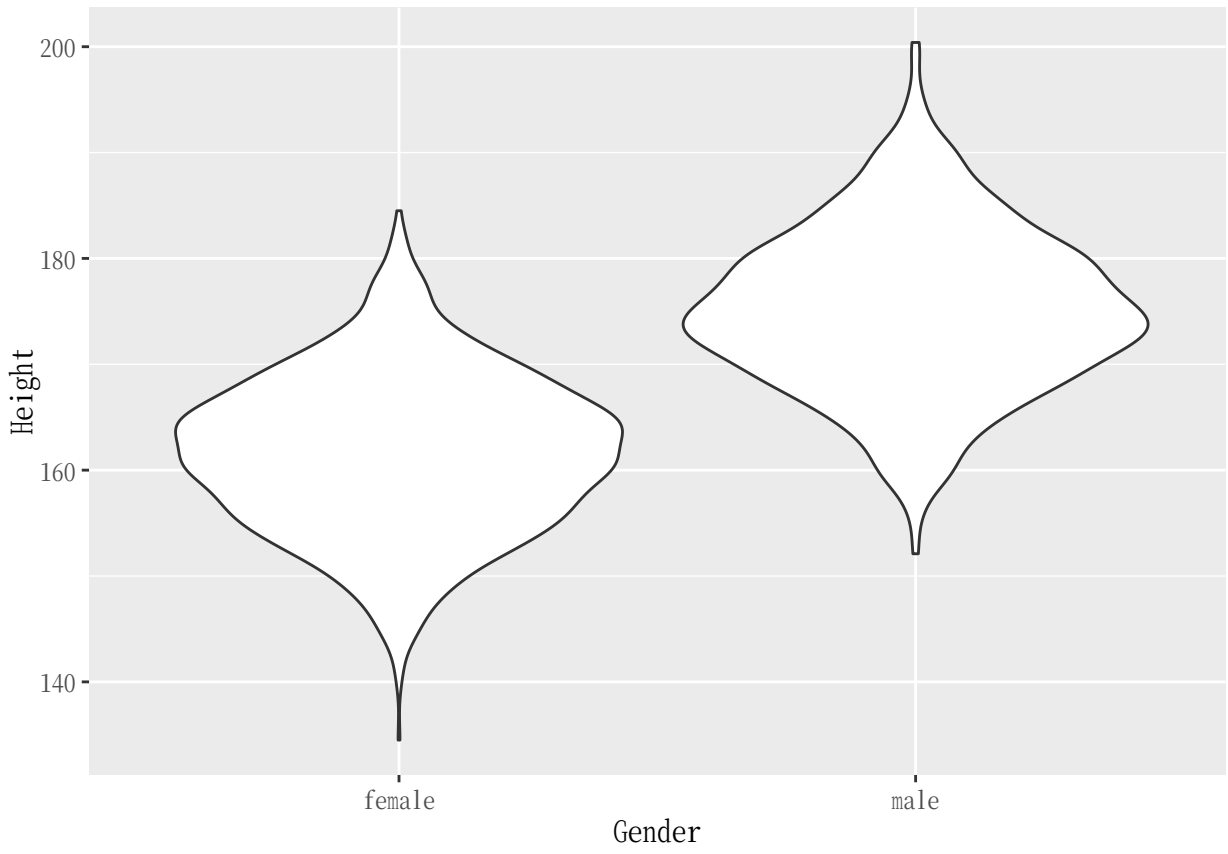
```
## Warning: Removed 22 rows containing non-finite values (stat_boxplot).
```



并列小提琴图：

```
ggplot(d_genht, aes(
  x = Gender, y = Height)) +
  geom_violin()
```

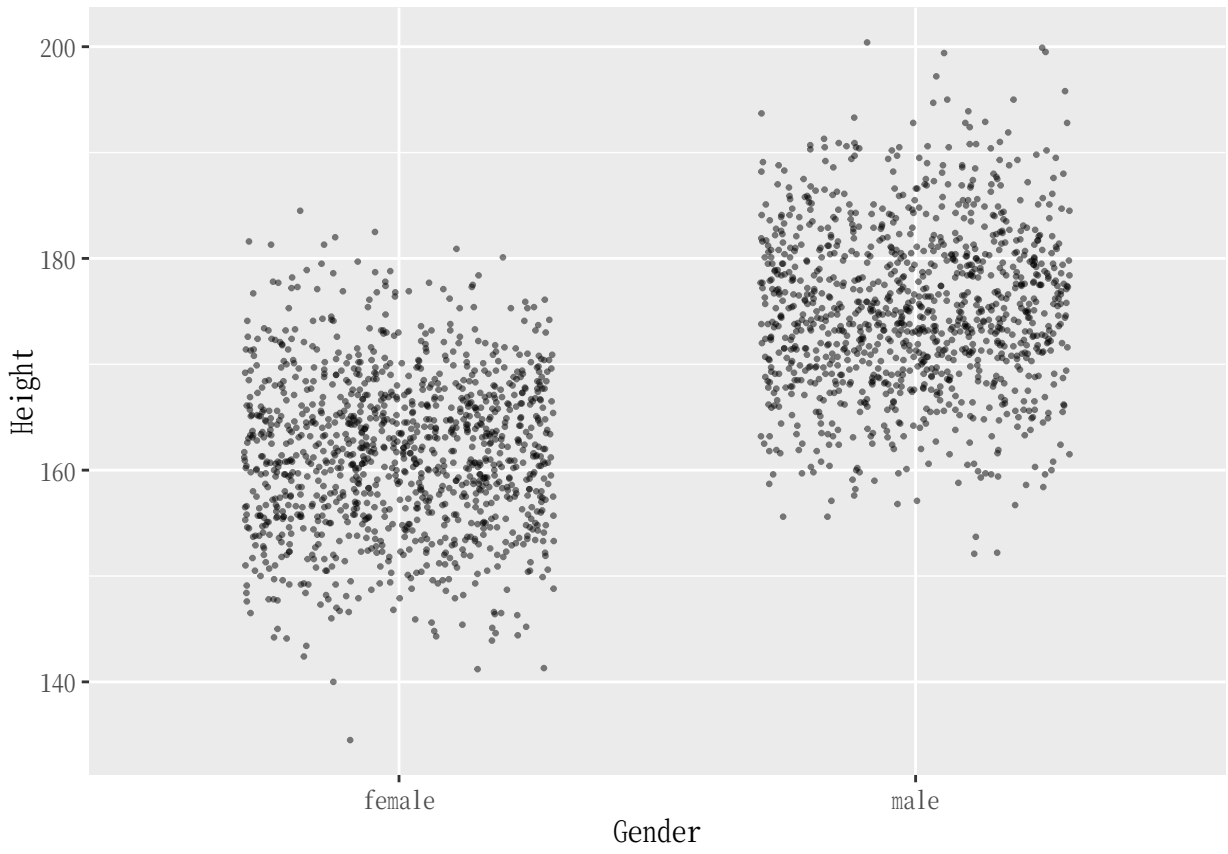
```
## Warning: Removed 22 rows containing non-finite values (stat_ydensity).
```



点图：

```
ggplot(d_genht, aes(
  x = Gender, y = Height)) +
  geom_jitter(
    width = 0.3, height = 0,
    size = 0.5, alpha = 0.5)
```

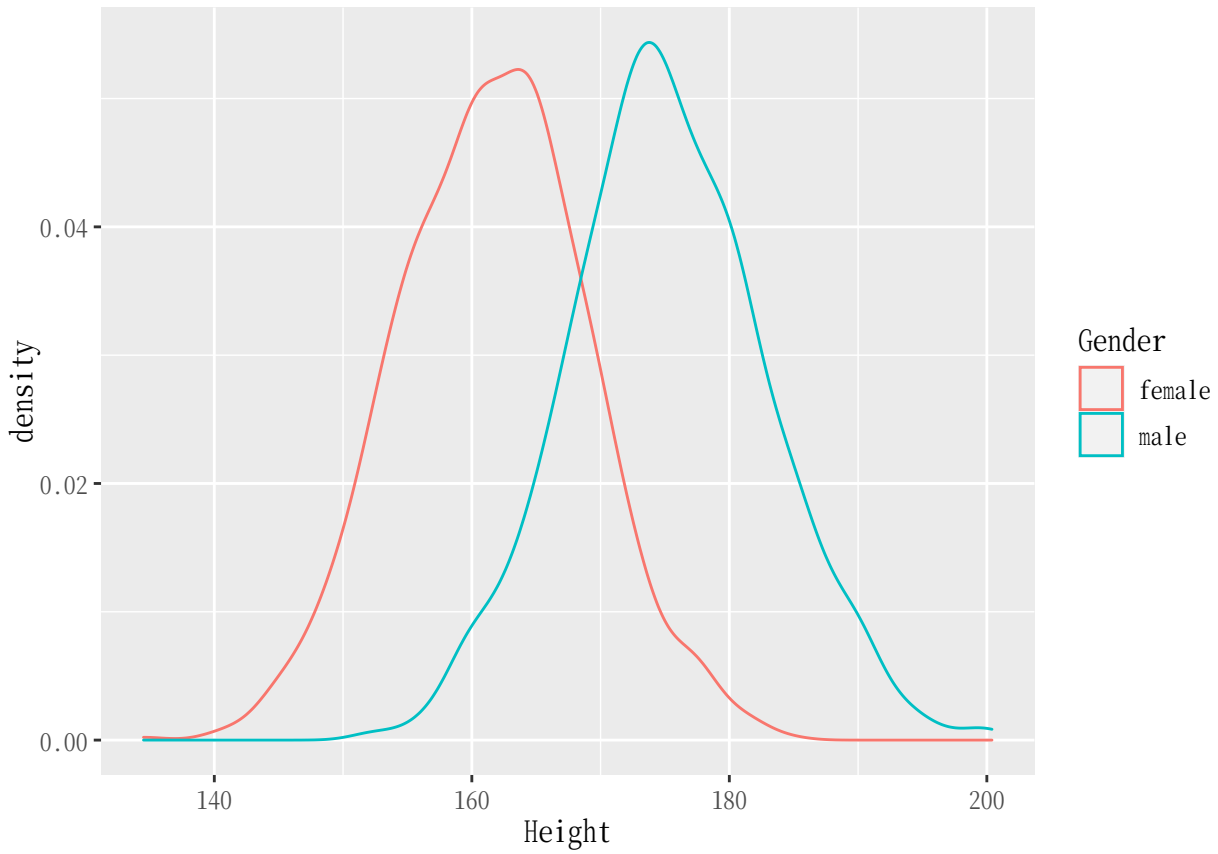
```
## Warning: Removed 22 rows containing missing values (geom_point).
```



比较核密度估计：

```
ggplot(d_genht, aes(
  x = Height,
  color = Gender)) +
  geom_density()
```

```
## Warning: Removed 22 rows containing non-finite values (stat_density).
```



29.6 两个区间型变量之间的关系

考虑身高 (Height) 和体重 (Weight) 的关系。可以计算两者的相关系数：

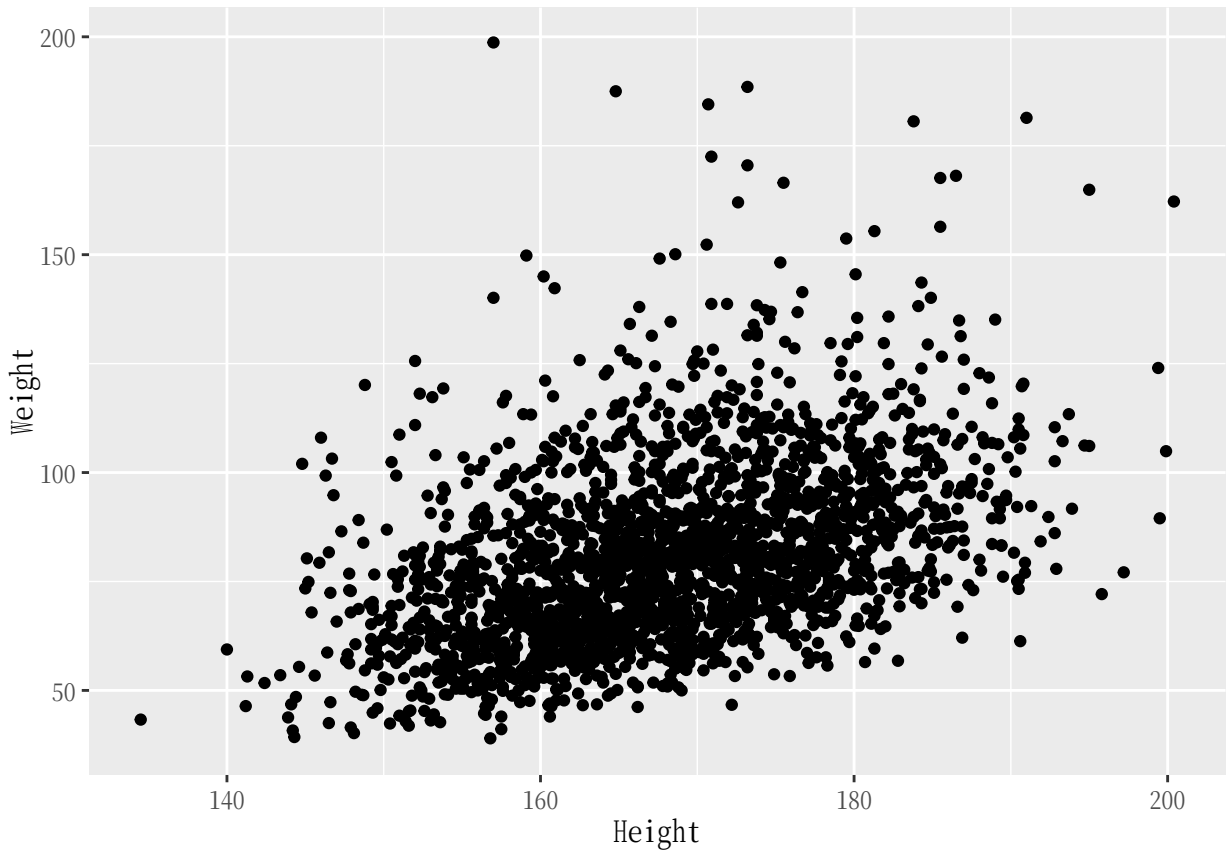
```
d_genht |>
  summarize(cor = cor(Height, Weight, use="complete.obs"))
```

```
## # A tibble: 1 x 1
##     cor
##   <dbl>
## 1 0.434
```

用散点图表示两者关系：

```
ggplot(d_genht, aes(x = Height, y = Weight)) +
  geom_point()
```

```
## Warning: Removed 24 rows containing missing values (geom_point).
```



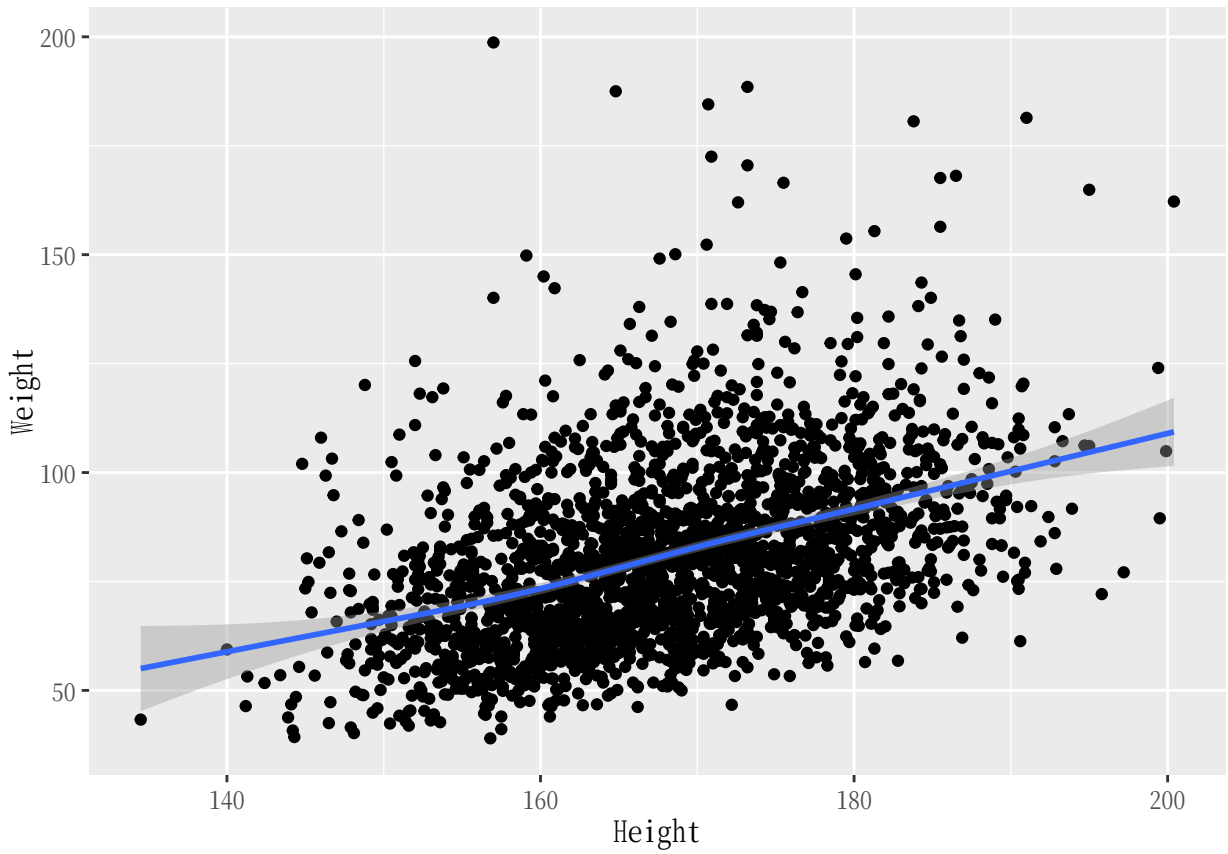
叠加拟合曲线：

```
ggplot(d_genht, aes(x = Height, y = Weight)) +
  geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

```
## Warning: Removed 24 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 24 rows containing missing values (geom_point).
```



多个区间变量之间可以计算相关系数矩阵：

```
d_genht |>
  select(Height, Weight, BMI, Age) |>
  cor(use = "pairwise.complete.obs")
```

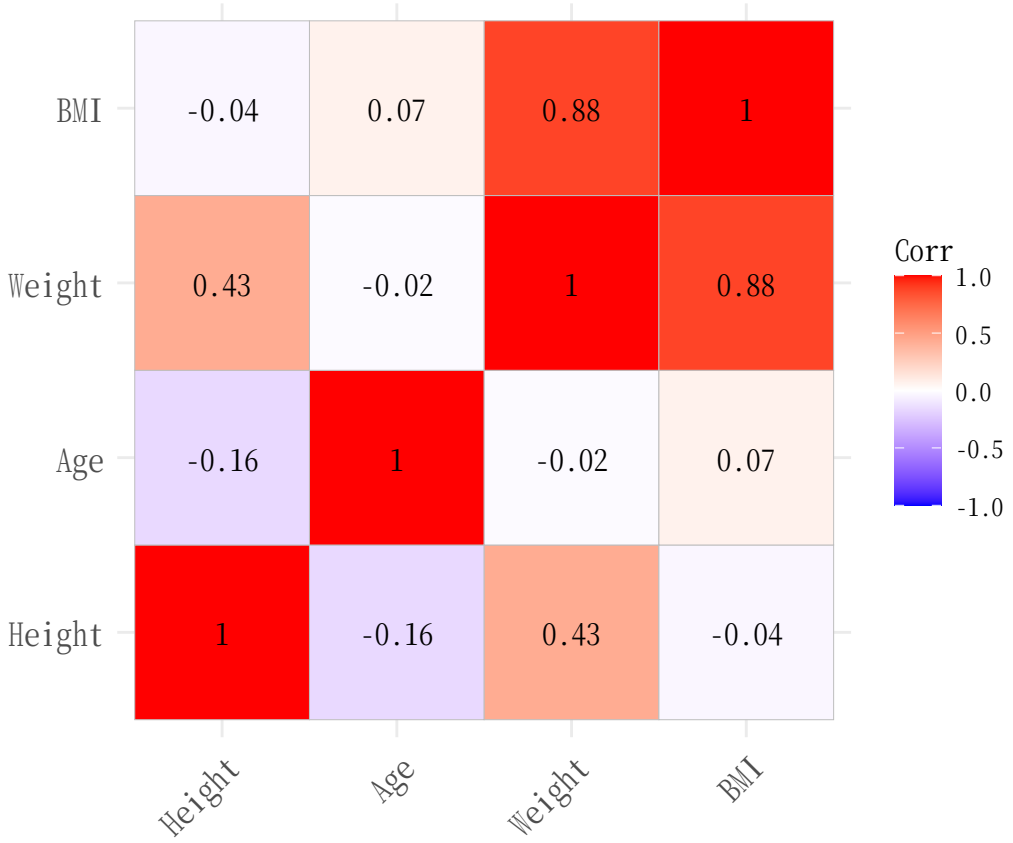
```
##             Height      Weight         BMI         Age
## Height  1.00000000  0.43360892 -0.03996646 -0.15849296
## Weight  0.43360892  1.00000000  0.87627484 -0.01645548
## BMI    -0.03996646  0.87627484  1.00000000  0.06683342
## Age    -0.15849296 -0.01645548  0.06683342  1.00000000
```

可以用图形方式表示相关系数矩阵：

```
d_cor <- d_genht |>
  select(Height, Weight, BMI, Age) |>
  cor(use = "pairwise.complete.obs")
library(ggcorrplot)
```

```
## Warning: package 'ggcorrplot' was built under R version 4.2.2
```

```
ggcorrplot(d_cor,
  hc.order=TRUE,
  lab=TRUE)
```



变量不太多时，可以作散点图矩阵：

```
library(GGally)
```

```
## Warning: package 'GGally' was built under R version 4.2.2

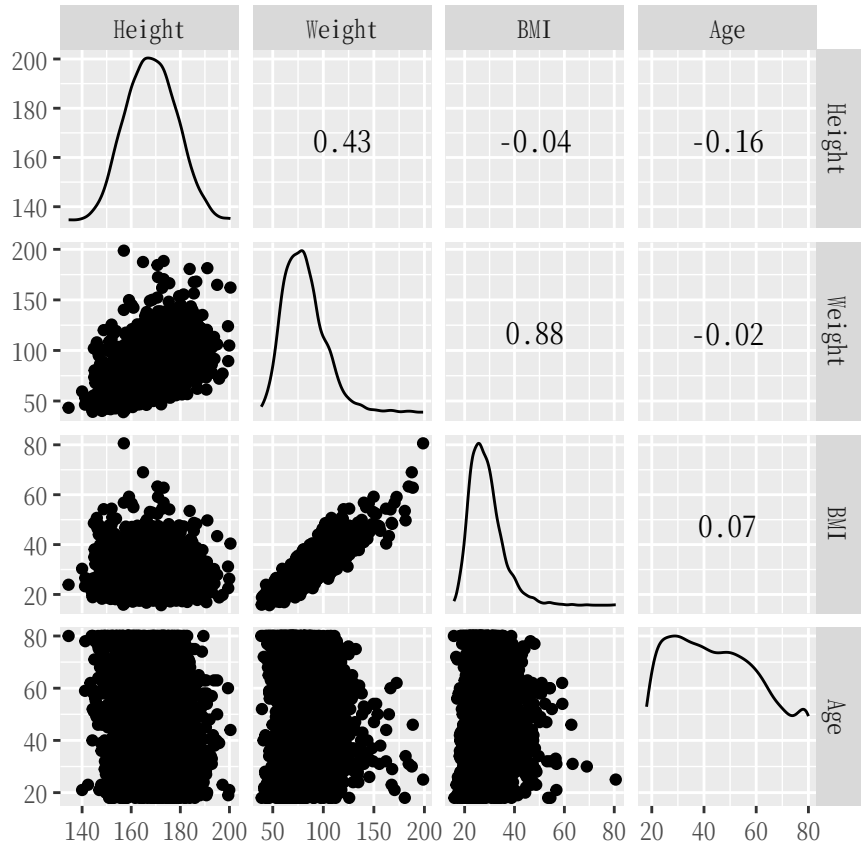
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
d_genht |>
  select(Height, Weight, BMI, Age) |>
  ggscatmat()
```

```
## Warning: Removed 22 rows containing non-finite values (stat_density).
```

```
## Warning: Removed 20 rows containing non-finite values (stat_density).

## Warning: Removed 24 rows containing non-finite values (stat_density).
```



29.7 一个分类变量对两个区间型变量的影响

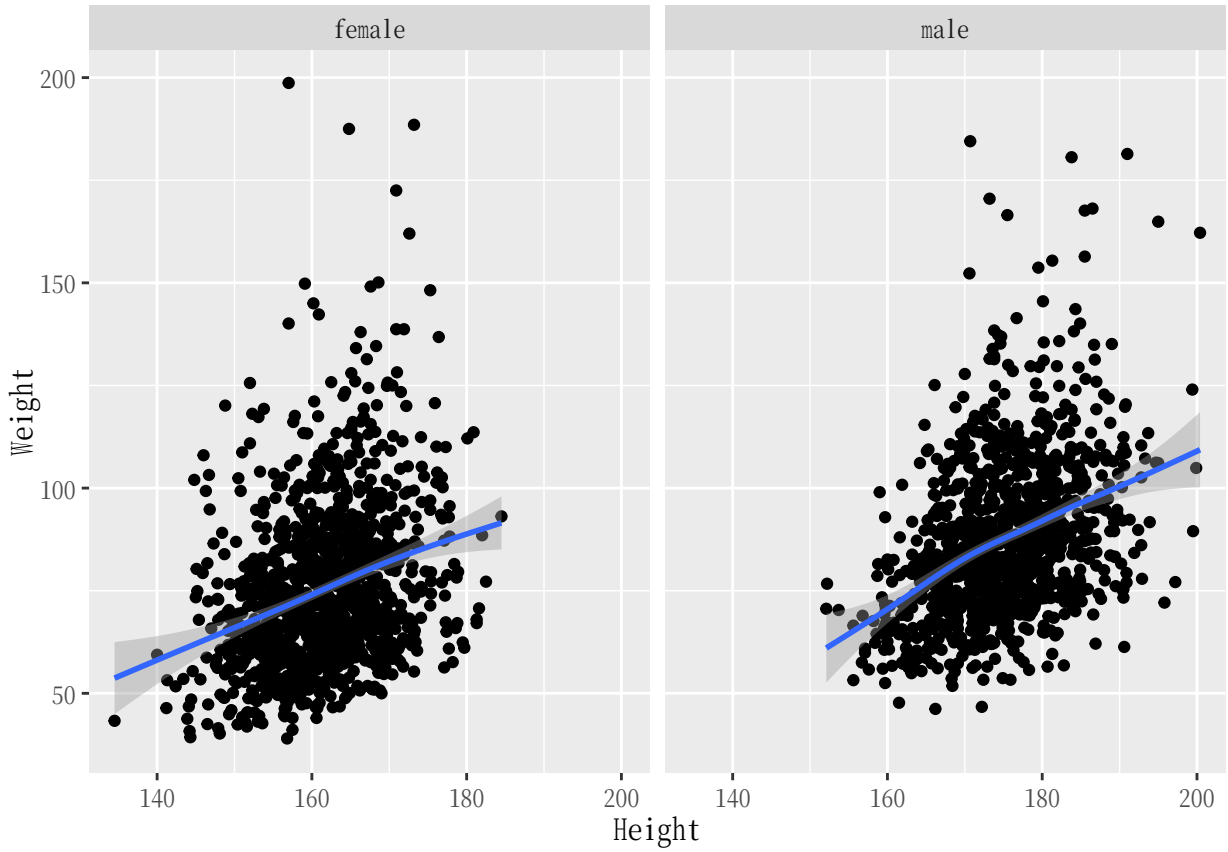
为了研究一个变量 z 对另外两个变量 x、y 的影响，可以按 z 的值切片作其它两个变量的图形，如比较不同性别的体重、身高关系：

```
ggplot(d_genht, aes(x = Height, y = Weight)) +
  geom_point() +
  geom_smooth() +
  facet_wrap(~ Gender)
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'

## Warning: Removed 24 rows containing non-finite values (stat_smooth).

## Warning: Removed 24 rows containing missing values (geom_point).
```



29.8 时间序列数据

如果数据含有时间序列，将时间序列画成曲线图可以发现趋势、季节性等特征，参见28.6。

Chapter 30

R 初等统计分析

这一部分讲授如何用 R 进行统计分析，包括基本概括统计和探索性数据分析，置信区间和假设检验，回归分析与各种回归方法，广义线性模型，非线性回归与平滑，判别树和回归树，等等。

主要参考书：

- (Venables & Ripley, 2002)
- (Kabacoff, 2012)

30.1 概率分布

R 中与 `xxx` 分布有关的函数包括：

- `dxxx(x)`，即 `xxx` 分布的分布密度函数 (PDF) 或概率函数 (PMF)$p(x)$。
- `pxxx(q)`，即 `xxx` 分布的分布函数 (CDF)$F(q) = P(X \le q)$。
- `qxxx(p)`，即 `xxx` 分布的分位数函数 $q(p)$, $p \in (0, 1)$，对连续型分布，$q(p) = F^{-1}(p)$，即 $F(x) = p$ 的解 x。
- `rxxx(n)`，即 `xxx` 的随机数函数，可以生成 n 个 `xxx` 的随机数。

`dxxx(x)` 函数可以加选项 `log=TRUE`，用来计算`\ln p(x)`，这在计算对数似然函数时有用，比 `log(dxxx(x))` 更精确。

`pxxx(q)` 可以加选项 `lower.tail=FALSE`，变成计算 $P(X > q) = 1 - F(q)$。

`qxxx(p)` 可以加选项 `lower.tail=TRUE`，表示求 $P(X > x) = p$ 的解 x；可以加选项 `log.p=TRUE`，表示输入的 p 实际是 $\ln p$。

这些函数都可以带有自己的分布参数，有些分布参数有缺省值，比如正态分布的缺省参数值为零均值单位标准差。

具体的分布类型可以在 R 命令行用`?Distributions` 查看列表。常用的分布密度有：

- 离散分布有 dbinom 二项分布, dpois 泊松分布，dgeom 几何分布，dnbinom 负二项分布，dmultinom 多项分布，dhyper 超几何分布。
- 连续分布有 dunif 均匀分布，dnorm 正态分布，dchisq 卡方分布，dt t 分布 (包括非中心 t)，df F 分布，dexp 指数分布，dweibull 威布尔分布，dgamma 伽马分布，dbeta 贝塔分布，dlnorm 对数正态分布，dcauchy 柯西分布, dlogis 逻辑斯谛分布。

R 的扩展包提供了更多的分布，参见 R 网站的如下链接：

- https://cran.r-project.org/web/views/Distributions.html

30.2 最大似然估计

R 函数 optim()、nlm()、optimize() 可以用来求函数极值，因此可以用来计算最大似然估计。optimize() 只能求一元函数极值。

30.2.1 一元正态分布参数最大似然估计

正态分布最大似然估计有解析表达式。作为示例，用 R 函数进行数值优化求解。

对数似然函数为：

$$\ln L(\mu, \sigma^2) = -\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln\sigma^2 - \frac{1}{2\sigma^2}\sum(X_i - \mu)^2$$

定义 R 的优化目标函数为上述对数似然函数去掉常数项以后乘以 -2，求其最小值点。目标函数为：

```
objf.norm1 <- function(theta, x){
  mu <- theta[1]
  s2 <- exp(theta[2])
  n <- length(x)
  res <- n*log(s2) + 1/s2*sum((x - mu)^2)
  res
}
```

其中 θ_1 为均值参数 μ，θ_2 为方差参数 σ^2 的对数值。x 是样本数值组成的 R 向量。

可以用 optim 函数来求极小值点。下面是一个模拟演示:

```
norm1d.mledemo1 <- function(n=30){
  mu0 <- 20
  sigma0 <- 2
  set.seed(1)
```

```
  x <- rnorm(n, mu0, sigma0)

  theta0 <- c(0,0)
  ores <- optim(theta0, objf.norm1, x=x)
  print(ores)
  theta <- ores$par
  mu <- theta[1]
  sigma <- exp(0.5*theta[2])
  cat('真实 mu=', mu0, ' 公式估计 mu=', mean(x),
      ' 数值优化估计 mu=', mu, '\n')
  cat('真实 sigma=', sigma0,
      '公式估计 sigma=', sqrt(var(x)*(n-1)/n),
      ' 数值优化估计 sigma=', sigma, '\n')
}
norm1d.mledemo1()
```

```
## $par
## [1] 20.166892  1.193593
##
## $value
## [1] 65.83709
##
## $counts
## function gradient
##      115       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## 真实mu= 20  公式估计mu= 20.16492  数值优化估计mu= 20.16689
## 真实sigma= 2 公式估计sigma= 1.817177  数值优化估计sigma= 1.816291
```

也可以用 nlm() 函数求最小值点，下面是正态分布最大似然估计的另一个演示：

```
norm1d.mledemo2 <- function(){
  set.seed(1)
  n <- 30
```

```
  mu <- 20
  sig <- 2
  z <- rnorm(n, mean=mu, sd=sig)
  neglogL <- function(parm) {
      -sum( dnorm(z, mean=parm[1],
                  sd=exp(parm[2]), log=TRUE) )
  }

  res <- nlm(neglogL, c(10, log(10)))
  print(res)
  sig2 <- exp(res$estimate[2])
  cat('真实 mu=', mu, ' 公式估计 mu=', mean(z),
      ' 数值优化估计 mu=', res$estimate[1], '\n')
  cat('真实 sigma=', sig,
      '公式估计 sigma=', sqrt(var(z)*(n-1)/n),
      ' 数值优化估计 sigma=', sig2, '\n')
  invisible()
}
norm1d.mledemo2()
```

```
## $minimum
## [1] 60.48667
##
## $estimate
## [1] 20.1649179  0.5972841
##
## $gradient
## [1] 1.444576e-05 9.094805e-06
##
## $code
## [1] 2
##
## $iterations
## [1] 28
##
## 真实mu= 20  公式估计mu= 20.16492  数值优化估计mu= 20.16492
## 真实sigma= 2 公式估计sigma= 1.817177  数值优化估计sigma= 1.817177
```

函数 `optim()` 缺省使用 Nelder-Mead 单纯型搜索算法，此算法不要求计算梯度和海色阵，算法稳定性好，但是收敛速度比较慢。可以用选项 `method="BFGS"` 指定 BFGS 拟牛顿法。这时可以用 `gr=` 选项输入梯度函数，缺省使

用数值微分计算梯度。如：

```
norm1d.mledemo1b <- function(n=30){
  mu0 <- 20
  sigma0 <- 2
  set.seed(1)
  x <- rnorm(n, mu0, sigma0)

  theta0 <- c(1,1)
  ores <- optim(theta0, objf.norm1, x=x, method="BFGS")
  print(ores)
  theta <- ores$par
  mu <- theta[1]
  sigma <- exp(0.5*theta[2])
  cat('真实 mu=', mu0, ' 公式估计 mu=', mean(x),
      ' 数值优化估计 mu=', mu, '\n')
  cat('真实 sigma=', sigma0,
      '公式估计 sigma=', sqrt(var(x)*(n-1)/n),
      ' 数值优化估计 sigma=', sigma, '\n')
}
norm1d.mledemo1b()
```

```
## $par
## [1] 20.164916  1.194568
##
## $value
## [1] 65.83704
##
## $counts
## function gradient
##       41       17
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## 真实mu= 20  公式估计mu= 20.16492  数值优化估计mu= 20.16492
## 真实sigma= 2 公式估计sigma= 1.817177  数值优化估计sigma= 1.817177
```

注意使用数值微分时，参数的初值的数量级应该与最终结果相差不大，否则可能不收敛，比如上面程序取初值 $(0,0)$ 就会不收敛。

`optim()` 函数支持的方法还有`"CG"` 是一种共轭梯度法，此方法不如 BFGS 稳健，但是可以减少存储量使用，对大规模问题可能更适用。`"L-BFGS-B"` 是一种区间约束的 BFGS 优化方法。`"SANN"` 是模拟退火算法，有利于求得全局最小值点，速度比其它方法慢得多。

函数 `optimize()` 进行一元函数数值优化计算。例如，在一元正态分布最大似然估计中，在对数似然函数中代入 $\mu = \bar{x}$，目标函数（负 2 倍对数似然函数）为

$$h(\sigma^2) = \ln(\sigma^2) + \frac{1}{\sigma^2}\sum(X_i - \bar{X})^2$$

下面的例子模拟计算 σ^2 的最大似然估计：

```
norm1d.mledemo3 <- function(n=30){
  mu0 <- 20
  sigma0 <- 2
  set.seed(1)
  x <- rnorm(n, mu0, sigma0)

  mu <- mean(x)
  ss <- sum((x - mu)^2)/length(x)
  objf <- function(delta, ss) log(delta) + 1/delta*ss
  ores <- optimize(objf, lower=0.0001,
                   upper=1000, ss=ss)
  delta <- ores$minimum
  sigma <- sqrt(delta)

  print(ores)
  cat('真实 sigma=', sigma0,
      '公式估计 sigma=', sqrt(var(x)*(n-1)/n),
      ' 数值优化估计 sigma=', sigma, '\n')
}
norm1d.mledemo3()
```

```
## $minimum
## [1] 3.30214
##
## $objective
## [1] 2.194568
##
## 真实sigma= 2 公式估计sigma= 1.817177  数值优化估计sigma= 1.817179
```

30.3 单样本均值的检验和置信区间

R 的 ggstatsplot 扩展包提供了将假设检验结果图示化的功能。

30.3.1 大样本情形

设总体 X 期望为 μ，方差为 σ^2。$X_1, X_2, \ldots, X_n$ 是一组样本。

在大样本时，令

$$Z = \frac{\bar{x} - \mu}{S/\sqrt{n}} \overset{\bullet}{\sim} \mathrm{N}(0,1)$$

其中 $\overset{\bullet}{\sim}$ 表示近似服从某分布。$\bar{x}$ 为样本均值，S 为样本标准差。σ 未知且大样本时，μ 近似 $1-\alpha$ 置信区间为

$$\bar{x} \pm \mathrm{qnorm}(1-\alpha/2)S/\sqrt{n}$$

用 `BSDA::z.test(x, sigma.x=sd(x), conf.level=1-alpha)` 可以计算如上的近似置信区间。

对于假设检验问题

$$H_0 : \mu = \mu_0 \longleftrightarrow H_a : \mu \neq \mu_0$$
$$H_0 : \mu \leq \mu_0 \longleftrightarrow H_a : \mu > \mu_0$$
$$H_0 : \mu \geq \mu_0 \longleftrightarrow H_a : \mu < \mu_0$$

定义检验统计量

$$Z = \frac{\bar{x} - \mu_0}{S/\sqrt{n}}$$

当 $\mu = \mu_0$ 时 Z 近似服从 N(0,1) 分布。可以用 `BSDA::z.test(x, mu=mu0, sigma.x=sd(x))` 进行双侧 Z 检验。加选项 `alternative="greater"` 作右侧检验，加选项 `alternative="less"` 作左侧检验。

如果上述问题中标准差 σ 为已知，应该在 Z 的公式用 σ 替换 S，即

$$Z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$$

计算 μ 的置信区间程序为 `BSDA::z.test(x, sigma.x=sigma0)`，计算 Z 检验的程序为 `BSDA::z.test(x, mu=mu0, sigma.x=sigma0)`，仍可用选项 `alternative=` 指定双侧、右侧、左侧。

30.3.2 小样本正态情形

如果样本量较小（小于 30），则需要假定总体服从正态分布 $\mathrm{N}(\mu, \sigma^2)$。

当 σ 已知时，有

$$Z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}} \sim \mathrm{N}(0,1)$$

计算 μ 的置信区间程序为 `BSDA::z.test(x, sigma.x=sigma0)`，计算 Z 检验的程序为 `BSDA::z.test(x, mu=mu0, sigma.x=sigma0)`。

可以自己写一个计算 Z 检验的较通用的 R 函数：

```
z.test.1s <- function(
  x, n=length(x), mu=0, sigma=sd(x), alternative="two.sided"){
  z <- (mean(x) - mu) / (sigma / sqrt(n))
  if(alternative=="two.sided"){   # 双侧检验
    pvalue <- 2*(1 - pnorm(abs(z)))
  } else if(alternative=="less"){ # 左侧检验
    pvalue <- pnorm(z)
  } else if(alternative=="greater"){ # 右侧检验
    pvalue <- 1 - pnorm(z)
  } else {
    stop("alternative unknown!")
  }

  c(stat=z, pvalue=pvalue)
}
```

这个函数输入样本 x 和 $\sigma = \sigma_0$，或者输入 $\bar{x}$ 为 x 和 $\sigma = \sigma_0$，输出检验统计量值和 p 值。

如果 σ 未知，应使用 t 统计量检验

$$T = \frac{\bar{x} - \mu}{S/\sqrt{n}}$$

当 $\mu = \mu_0$ 时 $T \sim t(n-1)$。如果 x 保存了样本，用 `t.test(x, conf.level=1-alpha)` 计算 μ 的置信区间；用 `t.test(x, mu=mu0, alternative="two.side")` 计算 $H_0 : \mu = \mu_0$ 的双侧检验，称为单样本 t 检验。

如果已知的 $\bar{x}$ 和 S 而非具体样本数据，可以自己写一个 R 函数计算 t 检验：

```
t.test.1s <- function(
  x, n=length(x), sigma=sd(x), mu=0,
  alternative="two.sided"){
  tstat <- (mean(x) - mu) / (sigma / sqrt(n))
  if(alternative=="two.sided"){   # 双侧检验
    pvalue <- 2*(1 - pt(abs(tstat), n-1))
  } else if(alternative=="less"){ # 左侧检验
    pvalue <- pt(tstat, n-1)
  } else if(alternative=="greater"){ # 右侧检验
    pvalue <- 1 - pt(tstat, n-1)
  } else {
    stop("Alternative unknown!")
  }

  c(stat=tstat, pvalue=pvalue)
}
```

这个函数允许输入 $\bar{x}$ 到 `x` 中，输入 n，输入 S 到 `sigma` 中，然后计算 t 检验。

30.3.3 统计假设检验显著性的解释

统计假设检验如果拒绝了 H_0，也称 “结果显著”、“有显著差异” 等。这只代表样本与零假设的差距足够大，可以以比较小的代价（第一类错误概率）排除这样的差距是由于随机样本的随机性引起的，但是并不表明这样的差距是有科学意义或者有现实意义的。

尤其是超大样本时，完全没有实际意义的差距也会检验出来是统计显著的。

这时，可以设定一个 “有实际意义的差距”δ，当差距超过 δ 时才认为有实际差距，如：

$$H_0 : |\mu - \mu_0| \leq \delta \longleftrightarrow |\mu - \mu_0| > \delta$$

统计假设检验如果不拒绝 H_0，也称为 “结果不显著”、“没有显著差异”，通常不认为 H_0 就是对的，看法是没有充分理由推翻 H_0。

进一步收集更多的数据，有可能就会发现显著差异。

例如，H_0 是嫌疑人无罪，最后不拒绝 H_0，只能是没有充分证据证明有罪，如果后来收集到了关键的罪证，就还可以再判决嫌疑人有罪。

30.3.4 置信区间与双侧检验的关系

设总体为正态分布 $\mathrm{N}(\mu, \sigma^2)$, σ 已知。随机抽取了 n 个样品，计算得 $\bar{x}$。μ 的 $1-\alpha$ 置信区间为

$$\bar{x} \pm z_{\alpha/2}\sigma/\sqrt{n}$$

置信区间理论依据：

$$P\left(\frac{|\bar{x} - \mu|}{\sigma/\sqrt{n}} \leq z_{\alpha/2}\right) = 1 - \alpha \tag{*}$$

对双侧检验

$$H_0 : \mu = \mu_0 \longleftrightarrow H_a : \mu \neq \mu_0$$

当

$$\frac{|\bar{x} - \mu_0|}{\sigma/\sqrt{n}} \leq z_{\alpha/2}$$

时接受 H_0。

接受 H_0 的条件，就是 μ_0 满足 (*) 式的大括号中的条件，即当且仅当 μ_0 落入 μ 的 $1-\alpha$ 置信区间时接受 H_0。

其它检验也与置信区间有类似关系，但是单侧检验对应于单侧置信区间。

30.3.5 单样本均值比较例子

载入假设检验部分示例程序所用数据：

```
load("data/hyptest-data.RData")
```

30.3.5.1 咖啡标重的单侧检验

美国联邦贸易委员会定期检查商品标签是否与实际相符。Hiltop Coffee 大罐咖啡标签注明含量为 3 磅。每罐具体多一些或少一些并不要紧，只要总体平均值不少于 3 磅，消费者利益就未受侵害。

只有在有充分证据时才应该做出分量不够的判决进而对厂家进行处罚。所以零假设是 $H_0 : \mu \geq 3$，对立假设是 $H_a : \mu < 3$。

收集数据进行统计判决后，如果不拒绝 H_0，就不需要采取行动；如果拒绝了 H_0，就说明有足够证据认为厂商灌装分量不足，应进行处罚。

随机抽取了 36 件样品，计算平均净重 $\bar{x}$ 作为总体均值 μ 的估计。

如果 $\bar{x} < 3$，则 H_0 有理由被怀疑。

$\bar{x}$ 比 $\mu_0 = 3$ 少多少，才让我们愿意拒绝 H_0？做出拒绝 H_0 的决定可能会犯第一类错误，如果差距很小，这样的差距很可能是随机抽样的随机性引起的，拒绝 H_0 有很大的机会犯错。

必须先确定一个能够容忍的第一类错误概率。调查员认为，可以容忍 1% 的错误拒绝 H_0 的机会（不该处罚时给出了处罚）以换得能够在厂商分量不足时正确地做出处罚决定的机会。即选检验水平 $\alpha = 0.01$。

当 H_0 实际成立时，边界 $\mu = \mu_0 = 3$ 处犯第一类错误的概率最大（这里最不容易区分），所以计算第一类错误概率只要在边界处计算。

统计假设检验第一步是确定零假设与对立假设，第二步是确定检验水平 α。第三步是抽取随机样本，计算检验统计量。

以往经验证明净重的标准差是 $\sigma = 0.18$，且净重的分布为正态分布。

则 $\bar{x}$ 的抽样分布为 $\mathrm{N}(\mu, \sigma^2/n)$，$\bar{x}$ 的标准误差为 $\mathrm{SE}(\bar{x}) = 0.18/\sqrt{36} = 0.03$。

令

$$z = \frac{\bar{x} - \mu_0}{\mathrm{SE}(\bar{x})}$$

则 H_0 成立且 μ 取边界处的 μ_0 时 $z \sim \mathrm{N}(0, 1)$。

当 $\bar{x}$ 比 $\mu_0 = 3$ 小很多时应拒绝 H_0。即检验统计量 z 很小时拒绝 H_0。

如果 $z = -1$，标准正态分布取-1 或比-1 更小的值的概率是 0.1586 (用 R 程序 `pnorm(-1)` 计算)，这个概率不小，不小于 $\alpha = 0.01$，这时不应该拒绝 H_0。

如果 $z = -2$，标准正态分布取-2 或比-2 更小的值的概率是 0.0228，(用 R 程序 `pnorm(-2)` 计算)，这个概率已经比较小了，但还不小于预先确定的第一类错误概率 $\alpha = 0.01$，这时不应该拒绝 H_0。

如果 $z=-3$，标准正态分布取-3 或比-3 更小的值的概率是 0.0013，(用 R 程序 `pnorm(-3)` 计算)，这个概率已经很小了，也小于预先确定的第一类错误概率 $\alpha=0.01$，这时应该拒绝 H_0。

对左侧检验问题 $H_0:\mu\geq\mu_0\longleftrightarrow H_a:\mu<\mu_0$, 若 Z 表示标准正态分布随机变量，z 是检验统计量的值，在 $\mu=\mu_0$ 时 z 的抽样分布是标准正态分布，可以计算抽样分布中取到等于 z 以及比 z 更小的值的概率 $P(Z\leq z)$，称为检验的 p 值。

p 值越小，说明 $\bar{x}$ 比 μ_0 小得越多，拒绝 H_0 也越有充分证据。当且仅当 p 值小于检验水平 α 时拒绝 H_0。

p 值小于 α，说明如果 H_0 真的成立，出现 $\bar{x}$ 那么小的检验统计量值的概率也很小，小于 α，所以第一类错误概率小于等于 α。

一般地，p 值是从样本计算的一个在假定零假设成立情况下，检验统计量取值更倾向于对立假设成立的概率。p 值越小，拒绝 H_0 越有依据。

设咖啡标称重量问题中抽取了 36 件样品，测得 $\bar{x}=2.92$ 磅，计算得

$$z=\frac{\bar{x}-\mu_0}{\sigma/\sqrt{n}}=\frac{2.92-3}{0.18/\sqrt{36}}=-2.67$$

p 值为 $P(Z\leq-2.67)$(Z 是标准正态分布随机变量)，用 R 计算得 `pnorm(-2.67)`$=0.0038\leq\alpha=0.01$，拒绝 H_0，应对该厂商进行处罚。

已经预先确定了检验水平 $\alpha=0.01$ 就不能在看到检验结果后更改检验水平。但是，报告出 p 值后，其他的人可以利用这样的信息，不同的人或利益方可能有不同的检验水平的选择，这通过同一个 p 值都可以给出需要的结果。

p 值也称为 “**观测显著性水平**”，因为它是能够拒绝 H_0 的最小的可选检验水平，选取比 p 值更小的检验水平就不能拒绝 H_0 了。

用自定义的 `z.test.1s()` 计算：

```
z.test.1s(x=2.92, mu=3, n=36, sigma=0.18, alternative="less")
```

```
##          stat       pvalue
## -2.666666667  0.003830381
```

p 值为 0.004，在 0.01 水平下拒绝原假设，认为咖啡平均重量显著地低于标称的 3 磅。

设 `Coffee` 数据框的 `Weight` 是这 36 个样本值，也可以将程序写成：

```
z.test.1s(Coffee[["Weight"]], mu=3, sigma=0.18, alternative="less")
```

```
##          stat       pvalue
## -2.666666667  0.003830381
```

或

```
BSDA::z.test(Coffee[["Weight"]], mu=3, sigma.x=0.18, alternative="less")
```

```
##
##  One-sample z-Test
##
## data:  Coffee[["Weight"]]
## z = -2.6667, p-value = 0.00383
## alternative hypothesis: true mean is less than 3
## 95 percent confidence interval:
##        NA 2.969346
## sample estimates:
## mean of x
##      2.92
```

30.3.5.2 高尔夫球性能的双侧检验

美国高尔夫协会对高尔夫运动器械有一系列的规定。MaxFlight 生产高品质的高尔夫球，平均的飞行距离是 295 码。

生产线有时会有波动，当生产的球平均飞行距离不足 295 码时，用户会感觉不好用；当生产的球平均飞行距离超过 295 码时，美国高尔夫协会禁止使用这样的球。

该厂商定期抽检生产线上下来的产品，每次抽取 50 只。

当平均飞行距离与 295 码没有显著差距时，不需要采取措施；如果有显著差距，就需要调整生产线。

所以取 $H_0 : \mu = 295$, $H_a : \mu \neq 295$。选检验水平 $\alpha = 0.05$。

随机抽取 n 个样品，在可控条件下测试出每个的飞行距离，计算出平均值 $\bar{x}$。

当 $\bar{x}$ 比 μ_0(这里是 295) 小很多或者 $\bar{x}$ 比 μ_0 大很多时拒绝 H_0，否则不拒绝 H_0。

设已知总体标准差 $\sigma = 12$, 且飞行距离服从正态分布。

取统计量

$$z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$$

在 H_0 成立时，z 的抽样分布为标准正态分布。

- 设测得 $\bar{x} = 297.6$，超过了 $\mu_0 = 295$。

$$z == \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}} = \frac{297.6 - 295}{12/\sqrt{50}} = 1.53$$

H_0 成立时 z 抽样分布为标准正态分布，设 Z 是标准正态分布随机变量，Z 取到 1.53 或者比 1.53 更倾向于反对 H_0 的值概率？这包括 $P(Z \geq 1.53)$，还应包括 $P(Z \leq -1.53)$。

所以 p 值是 $P(|Z| \geq |z|)$，R 中计算为 `2*(1 - pnorm(abs(z)))`。p 值等于 0.1260，超过检验水平，不拒绝 H_0，不需要调整生产线。

用自定义的 `z.test.1s()` 检验：

```
z.test.1s(GolfTest[['Yards']], mu=295, sigma=12, alternative="two.sided")
```

```
##      stat    pvalue
## 1.5320647 0.1255065
```

或用 `BSDA::z.test()`:

```
BSDA::z.test(GolfTest[['Yards']], mu=295, sigma.x=12, alternative="two.sided")
```

```
##
##  One-sample z-Test
##
## data:  GolfTest[["Yards"]]
## z = 1.5321, p-value = 0.1255
## alternative hypothesis: true mean is not equal to 295
## 95 percent confidence interval:
##  294.2738 300.9262
## sample estimates:
## mean of x
##     297.6
```

p 值为 0.13，在 0.05 水平下不拒绝 H_0，生产的球的平均飞行距离与标准的 295 码之间没有显著差异。

30.3.5.3 Heathrow 机场打分的检验

某旅行杂志希望根据乘机进行商务旅行的意见给跨大西洋门户机场评分。乘客评分取 0 到 10 分，超过 7 分算作是服务一流的机场。在每个机场随机选取 60 位商务旅行乘客打分评价。英国 Heathrow 机场的打分样本 $\bar{x} = 7.25$, $S = 1.052$。英国 Heathrow 机场可以算是服务一流吗？

由于随机样本的随机性，要判断的是 $\mu > 7$ 是否成立，这里 $\bar{x} = 7.25$ 并不能很确定第说 $\mu > 7$ 成立。

因为将机场评为一流需要有充分证据，所以将对立假设 H_a 设为 $\mu > 7$，然后取 $H_0 : \mu \leq 7$。

取检验水平 $\alpha = 0.05$。

计算检验统计量

$$t = \frac{\bar{x} - \mu_0}{S/\sqrt{n}} = \frac{7.25 - 7}{1.052/\sqrt{60}} = 1.84$$

设 T 是服从 $t(n-1)$ 分布的随机变量，p 值为 $P(T \geq 1.84)$。用 R 软件计算为 `1 - pt(1.84, 60-1)`$= 0.0354$。p 值小于 α，拒绝 H_0，认为平均评分显著高于 7，可以认为 Heathrow 机场是服务一流。

从原始样本数据用 `t.test()` 检验：

```
t.test(AirRating[["Rating"]], mu=7, alternative="greater")
```

```
##
##  One Sample t-test
##
## data:  AirRating[["Rating"]]
## t = 1.8414, df = 59, p-value = 0.03529
## alternative hypothesis: true mean is greater than 7
## 95 percent confidence interval:
##  7.023124      Inf
## sample estimates:
## mean of x
##      7.25
```

用自定义的 `t.test.1s()` 和样本统计量计算：

```
t.test.1s(x=7.25, sigma=1.052, n=60, mu=7, alternative="greater")
```

```
##       stat     pvalue
## 1.84077155 0.03534255
```

30.3.5.4 玩具厂商订货量的假设检验

玩具厂商 Holiday Toys 通过超过 1000 家玩具商店售货。为了准备即将到来的冬季销售季节，销售主管需要确定今年某新款玩具的生产量。凭经验判断平均每家商店需要 40 件。

为了确定，随机抽取了 25 家商店，提供了新款玩具的特点、进货价和建议销售价，让他们给出订货量估计。目的是判断按每家 40 件来生产是否合适。

因为多生产或少生产都不合适，所以取零假设为 $H_0 : \mu = 40$。对立假设为 $H_a : \mu \neq 40$。只有拒绝 H_0 时才需要更改生产计划。

取检验水平 $\alpha = 0.05$。样本计算得 $n = 25$，$\bar{x} = 37.4$, $S = 11.79$。

检验统计量

$$t = \frac{\bar{x} - \mu_0}{S/\sqrt{n}} = \frac{37.4 - 40}{11.79/\sqrt{25}} = -1.10$$

设 T 为服从 $\mathrm{t}(n-1)$ 分布的随机变量，p 值为 $2P(|T| > |-1.10|)$，用 R 计算为 `2*(1 - pt(abs(-1.10), 25-1))`$= 0.2822$。

不拒绝 H_0，所以不需要更改生产计划。

从原始样本数据用 t.test() 检验：

```
t.test(Orders[["Units"]], mu=40, alternative="two.sided")
```

```
##
##  One Sample t-test
##
## data:  Orders[["Units"]]
## t = -1.1026, df = 24, p-value = 0.2811
## alternative hypothesis: true mean is not equal to 40
## 95 percent confidence interval:
##  32.5334 42.2666
## sample estimates:
## mean of x
##      37.4
```

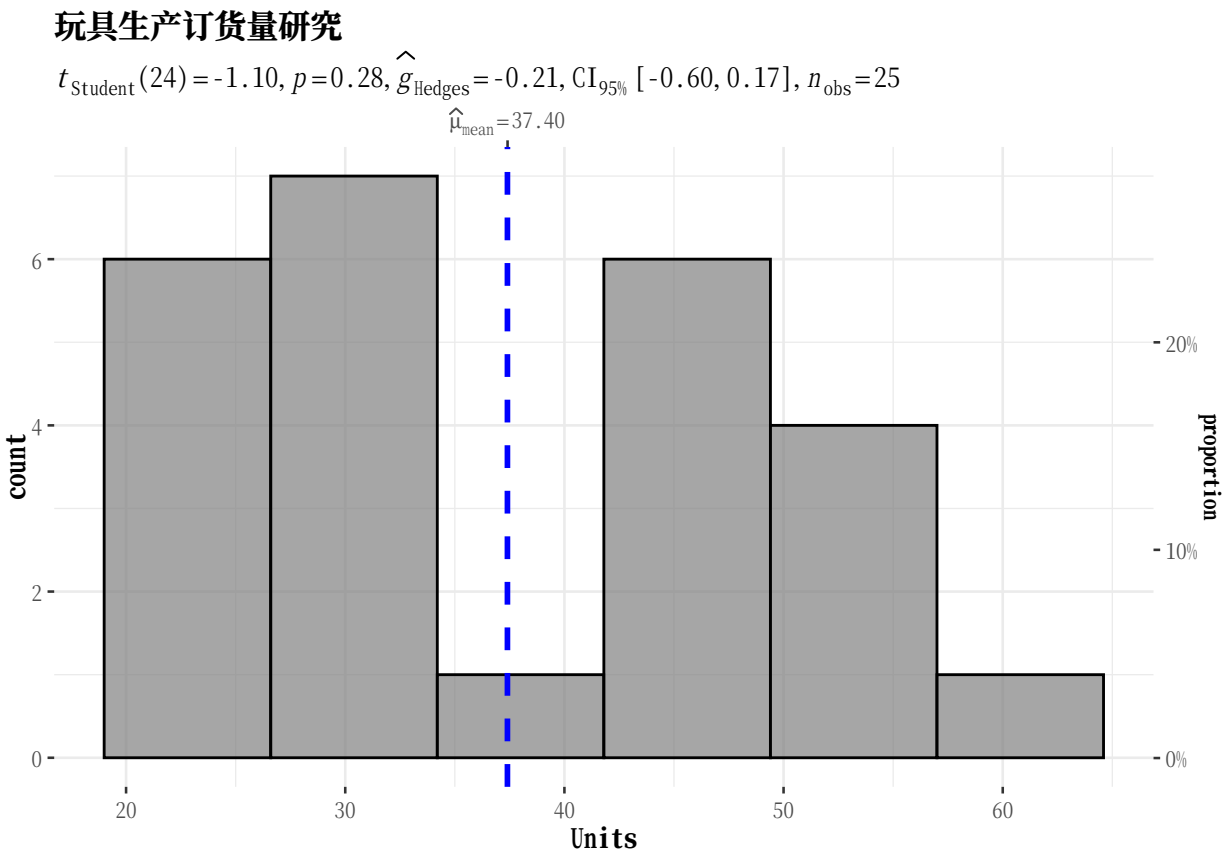
用自定义的 t.test.1s() 和样本统计量计算：

```
t.test.1s(x=37.4, sigma=11.79, n=25, mu=40, alternative="two.sided")
```

```
##       stat     pvalue
## -1.1026293  0.2811226
```

利用 ggstatsplot 图示：

```
library(ggstatsplot)
gghistostats(
  data              = Orders,
  x                 = Units,
  title      = " 玩具生产订货量研究",
  test.value = 40
)
```



30.3.6 单样本均值比较样本量和功效计算

检验的功效，是指对立假设成立时检验拒绝 H_0 的概率 $1-\beta$，其中 β 是第二类错误，即当对立假设成立时错误地接受 H_0 的概率。需要足够大的样本量才能使得检验能够发现实际存在的显著差异。因为要计算功效需要一个确定的对立假设下的分布，而不是一族分布，所以需要指定要检验的具体零假设和对立假设分布，或者指定一个效应大小值。

设 $H_0 : \mu = \mu_0$，对立假设是双侧或者单侧，为了能够在 $\mu = \mu_1 \neq \mu_0$ 时拒绝 H_0 的概率达到指定的功效 $1-\beta = 0.80$，设检验水平为 α，$|\mu_1 - \mu_0| = \delta$，需要的样本量为可以用 R 的 `power.t.test()` 计算，如：

```
power.t.test(delta = 2.5, sd = 1.5,
  type = "one.sample",
  alternative = "two.sided",
  sig.level=0.05,
  power=0.80)
```

```
##
##      One-sample t test power calculation
##
```

```
##                n = 5.04919
##            delta = 2.5
##               sd = 1.5
##        sig.level = 0.05
##            power = 0.8
##      alternative = two.sided
```

其中 delta 是要发现的均值差距下限，sd 是样本标准差，如果未知需要先用一个小样本进行估计，sig.level 是检验水平，power 是要求达到的功效。可以用 alternative 参数选择双侧（two.sided）或单侧（one.sided）。结果表明所求样本量为 6。

样本量和功效是互相决定的。前面的例子是给定要达到的功效，求样本量。也可以输入样本量，求可达到的功效，如：

```r
power.t.test(delta = 2.5, sd = 1.5,
  type = "one.sample",
  alternative = "two.sided",
  sig.level=0.05,
  n=10)
```

```
##
##      One-sample t test power calculation
##
##                n = 10
##            delta = 2.5
##               sd = 1.5
##        sig.level = 0.05
##            power = 0.9964471
##      alternative = two.sided
```

所以在单样本 t 检验中，若标准差为 $\sigma = 1.5$，零假设和对立假设下的均值差为 $|\mu_1 - \mu_0| = 2.5$，检验水平 $\alpha = 0.05$，作双侧 t 检验，则样本量 $n = 10$ 时功效可达到 99.64%。

实际上，要发现的均值之间的差别依赖于标准差，而比值 $\frac{|\mu_1-\mu_0|}{\sigma}$ 则完全决定了要发现的均值差别，样本量与功效的关系在比值 $\frac{|\mu_1-\mu_0|}{\sigma}$ 给定时也完全确定。所以，称比值

$$d = \frac{|\mu_1 - \mu_0|}{\sigma}$$

为要检验的 “效应大小”(effect size)。上面例子中 $d = 2.5/1.5 = 1.67$，属于较大的效应，所以很容易检出，只需要较小的样本量就可以达到足够的功效。

(Cohen, 1988) 给出了单样本 t 检验时常用的效应大小：

- 小 (small): 0.2;
- 中 (medium): 0.5;
- 大 (large): 0.8。

要发现的效应越小，就需要越大的样本量。要发现的效应很大时，较小的样本量就可以获得较高的功效。

R 扩展包 pwr 提供了功效与样本量计算问题。比如，单样本 t 检验，要检验中等效应大小，取检验水平 0.05，为了达到 0.80 的功效，可计算如：

```
library(pwr)
```

```
## Warning: package 'pwr' was built under R version 4.2.2
```

```
cohen.ES(test = "t", size = "medium")
```

```
##
##      Conventional effect size from Cohen (1982)
##
##            test = t
##            size = medium
##     effect.size = 0.5
```

```
pwr.t.test(
  type = "one.sample",
  alternative="two.sided",
  d = 0.5,
  sig.level = 0.05,
  power = 0.80 )
```

```
##
##      One-sample t test power calculation
##
##               n = 33.36713
##               d = 0.5
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
```

结果表明，样本量要达到 $n = 34$。

也可以输入样本量，计算可达到的功效：

```
pwr.t.test(
  type = "one.sample",
  alternative="two.sided",
  d = 0.5,
  sig.level = 0.05,
  n = 30 )
```

```
##
##      One-sample t test power calculation
##
##               n = 30
##               d = 0.5
##       sig.level = 0.05
##           power = 0.7539647
##     alternative = two.sided
```

单样本双侧 t 检验，为了检验中等效应大小，样本量 $n = 30$ 对应的功效是 75%。

例 30.1. 以希思罗机场打分数据为例。

H_0 下 $\mu = 7.0$, 实际数据的 $\bar{x} = 7.25$，标准差用样本标准差 $S = 1.052$ 代替，样本量 $n = 60$，计算 $\bar{x} - \mu_0 = 7.25 - 0.25$ 这样的均值差，其效应大小为

$$d = \frac{\bar{x} - \mu_0}{S} = \frac{7.25 - 7.0}{1.052} = 0.2376$$

这个效应比较小，对应于较大的样本量或者较小的功效。

计算这样的效应大小的功效，取检验水平 0.05:

```
pwr.t.test(
  type = "one.sample",
  alternative="greater",
  d = (7.25 - 7.0)/1.052,
  sig.level = 0.05,
  n = 60 )
```

```
##
##      One-sample t test power calculation
##
##               n = 60
##               d = 0.2376426
##       sig.level = 0.05
```

```
##           power = 0.5693621
##     alternative = greater
```

功效只有 57%。计算需要多大样本量可达到 80% 功效：

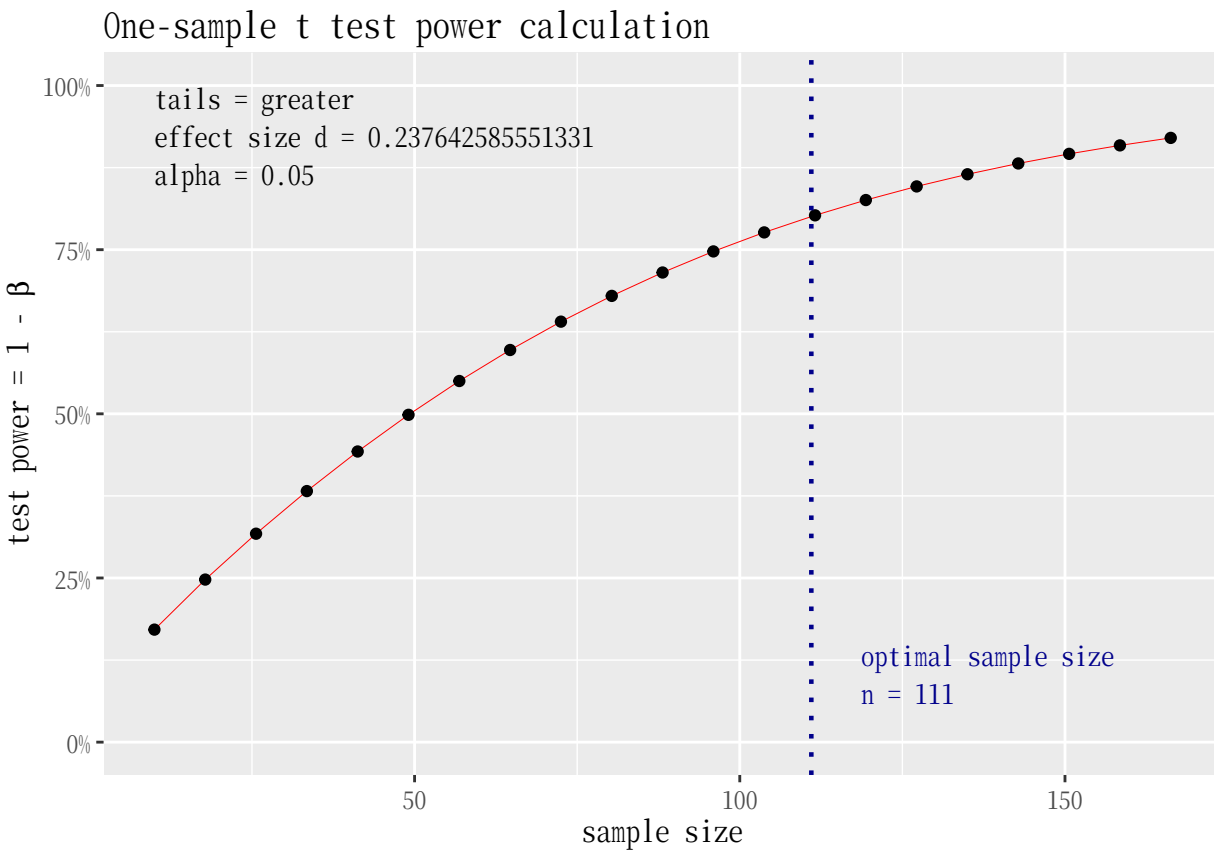
```
pwr.t.test(
  type = "one.sample",
  alternative="greater",
  d = (7.25 - 7.0)/1.052,
  sig.level = 0.05,
  power = 0.80 )
```

```
##
##      One-sample t test power calculation
##
##               n = 110.8416
##               d = 0.2376426
##       sig.level = 0.05
##           power = 0.8
##     alternative = greater
```

样本量要达到 $n = 111$。

功效随样本量变化曲线图示：

```
pwr.t.test(
  type = "one.sample",
  alternative="greater",
  d = (7.25 - 7.0)/1.052,
  sig.level = 0.05,
  power = 0.80 ) |>
  plot()
```



30.4 均值比较

30.4.1 独立两样本 Z 检验

假设有两个独立的总体 X, Y，例如，男性与女性的寿命。要比较两个总体的期望 $\mu_1 = EX$ 和 $\mu_2 = EY$。设两个总体的方差分别为 σ_1^2 和 σ_2^2。可以对均值作双侧、左侧、右侧检验。比如，双侧检验为

$$H_0 : \mu_1 = \mu_2 \longleftrightarrow H_a : \mu_1 \neq \mu_2$$

如果两个总体都服从正态分布，而且分别的方差 σ_1^2 和 σ_2^2 已知，设 X 的样本量为 n_1 的样本得到样本均值 $\bar{x}$，设 Y 的样本量为 n_2 的样本得到样本均值 $\bar{y}$，取统计量

$$Z = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$$

其中分母是分子的标准误差。当 $\mu_1 = \mu_2$ 时 $Z \sim \mathrm{N}(0, 1)$。

如果分布不一定是正态分布但是样本量足够大，可以令

$$Z = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{S_{\bar{x}}^2}{n_1} + \frac{S_{\bar{y}}^2}{n_2}}}$$

当 $\mu_1 = \mu_2$ 时 Z 近似服从 $\mathrm{N}(0,1)$。

检验问题还可以变成 $\mu_1 - \mu_2$ 与某个 δ 的比较，如

$$H_0 : \mu_1 \le \mu_2 - \delta \longleftrightarrow H_a : \mu_1 > \mu_2 - \delta$$

这个对立假设是“不次于”的结论。

可以写一个大样本方差未知或已知，小样本独立两正态总体方差已知情况做 Z 检验的 R 函数：

```
z.test.2s <- function(
  x, y, n1=length(x), n2=length(y), delta=0,
  sigma1=sd(x), sigma2=sd(y), alternative="two.sided"){
  z <- (mean(x) - mean(y) - delta) / sqrt(sigma1^2 / n1 + sigma2^2 / n2)
  if(alternative=="two.sided"){   # 双侧检验
    pvalue <- 2*(1 - pnorm(abs(z)))
  } else if(alternative=="less"){ # 左侧检验
    pvalue <- pnorm(z)
  } else if(alternative=="greater"){ # 右侧检验
    pvalue <- 1 - pnorm(z)
  } else {
    stop("alternative unknown!")
  }

  c(stat=z, pvalue=pvalue)
}
```

函数允许输入两组样本到 `x` 和 `y` 中，输入 σ_1 和 σ_2 到 `sigma1` 和 `sigma2` 中，这时不输入 `sigma1` 和 `sigma2` 则从样本中计算样本统计量代替。也允许输入 $\bar{x}$ 和 $\bar{y}$ 到 `x` 和 `y` 中，输入 σ_1 和 σ_2 到 `sigma1` 和 `sigma2` 中，或者输入 S_x 和 S_y 到 `sigma1` 和 `sigma2` 中，输入 n_1 到 `n1` 中，输入 n_2 到 `n2` 中，计算独立两样本均值比较的 Z 检验。

30.4.2 独立两样本 t 检验

如果样本是小样本，但两个总体分别服从正态分布，两个总体的方差未知，但已知 $\sigma_1^2 = \sigma_2^2$。这时 $\sigma_1^2 = \sigma_2^2$ 可以统一估计为

$$S_p^2 = \frac{1}{n_1 + n_2 - 2}\left((n_1 - 1)S_x^2 + (n_2 - 1)S_y^2\right)$$

取检验统计量

$$T = \frac{\bar{x} - \bar{y}}{S_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

当 $\mu_1 = \mu_2$ 时 $T \sim \mathrm{t}(n_1 + n_2 - 2)$。

当 x 和 y 存放了两组独立样本时，可以用 t.test(x, y, var.equal=TRUE) 计算两样本 t 检验，用 alternative= 选项指定双侧、右侧、左侧检验。

可以自己写一个这样的 R 函数作两样本 t 检验，允许只输入统计量而非具体样本值：

```
t.test.2s <- function(
  x, y, n1=length(x), n2=length(y),
  sigma1=sd(x), sigma2=sd(y), delta=0,
  alternative="two.sided"){
  sp <- sqrt(1/(n1+n2-2) * ((n1-1)*sigma1^2 + (n2-1)*sigma2^2))
  t <- (mean(x) - mean(y) - delta) / (sp * sqrt(1 / n1 + 1 / n2))
  if(alternative=="two.sided"){   # 双侧检验
    pvalue <- 2*(1 - pt(abs(t), n1+n2-2))
  } else if(alternative=="less"){ # 左侧检验
    pvalue <- pt(t, n1+n2-2)
  } else if(alternative=="greater"){ # 右侧检验
    pvalue <- 1 - pt(t, n1+n2-2)
  } else {
    stop("alternative unknown!")
  }

  c(stat=t, pvalue=pvalue)
}
```

如果两个独立正态总体的方差不相等，样本量不够大，可以用如下的在 H_0 下近似服从 t 分布的检验统计量：

$$T = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{S_x^2}{n_1} + \frac{S_y^2}{n_2}}}$$

自由度为

$$m = \frac{\left(S_x^2/n_1 + S_y^2/n_2\right)^2}{\frac{\left(S_x^2/n_1\right)^2}{n_1-1} + \frac{\left(S_y^2/n_2\right)^2}{n_2-1}}$$

这个检验称为 Welch 两样本 t 检验，或 Satterthwaite 两样本 t 检验当 x 和 y 存放了两组独立样本时，可以用 t.test(x, y, var.equal=FALSE) 计算 Welch 两样本 t 检验。

如果方差是否相等不易判断，可以直接选用 Welch 检验。

因为大样本时 t 分布与标准正态分布基本相同，所以非正态总体大样本 Z 检验也可以用 t.test() 计算。

30.4.3 成对均值比较问题

设总体包含两个 X, Y 分量，比如，一组病人的服药前血压与服药后血压。这两个变量是相关的，不能用独立两总体的均值比较方法比较 $\mu_1 = EX$ 与 $\mu_2 = EY$。

若 $\xi = X - Y$ 服从正态分布，则可以对 ξ 进行均值为零的单样本 t 检验。这样的检验称为成对均值的 t 检验。

若 `x` 和 `y` 分别为 X 和 Y 的样本值，可以用 `t.test(x, y, paired=TRUE)` 计算计算成对均值 t 检验。

30.4.4 均值比较的例子

30.4.4.1 顾客平均年龄差别比较

HomeStyle 是一个连锁家具店，两家分店一个在城区，一个在郊区。经理发现同一种家具有可能在一个分店卖的很好，另一个分店则不好。怀疑是顾客人群的人口学差异（年龄、性别、种族等）。

独立地抽取两个分店的顾客样本，比较平均年龄。设城内的分店顾客年龄 $\sigma_1 = 9$ 已知，郊区的分店顾客年龄 $\sigma_2 = 10$ 已知。

城内调查了 $n_1 = 36$ 位顾客，年龄的样本平均值 $\bar{x} = 40$; 郊区调查了 $n_2 = 49$ 位顾客，年龄的样本平均值 $\bar{y} = 35$。$\bar{x} - \bar{y} = 5$。

作双侧检验，水平 $\alpha = 0.05$。

计算 Z 统计量：

$$Z = \frac{40 - 35}{\sqrt{\frac{9^2}{36} + \frac{10^2}{49}}} = 2.4138$$

p 值为 $P(|V| \geq |2.4138|)$，其中 V 为标准正态分布随机变量。用 R 计算为 `2*(1 - pnorm(abs(2.4138)))`=0.016，两个分店的顾客年龄有显著差异，城里的顾客平均年龄显著地高。

基于样本统计量用自定义的 `z.test.2s()` 计算：

```
z.test.2s(n1=36, x=40, sigma1=9,
          n2=49, y=35, sigma2=10,
          alternative="two.sided")
```

```
##       stat     pvalue
## 2.41379310 0.01578742
```

30.4.4.2 两个银行营业所顾客平均存款比较

某银行要比较两个营业所的支票账户的平均存款额。作双侧检验，水平 0.05。没有方差信息时可以直接选用 Welch 检验。

设第一个营业所随机抽查了 $n_1 = 28$ 个支票账户，均值为 $\bar{x} = 1025$(美元)，样本标准差为 $S_1 = 150$，第二个营业所随机抽查了 $n_2 = 22$ 个支票账户，均值为 $\bar{y} = 910$, 样本标准差为 $S_2 = 125$。

从原始数据出发计算检验：

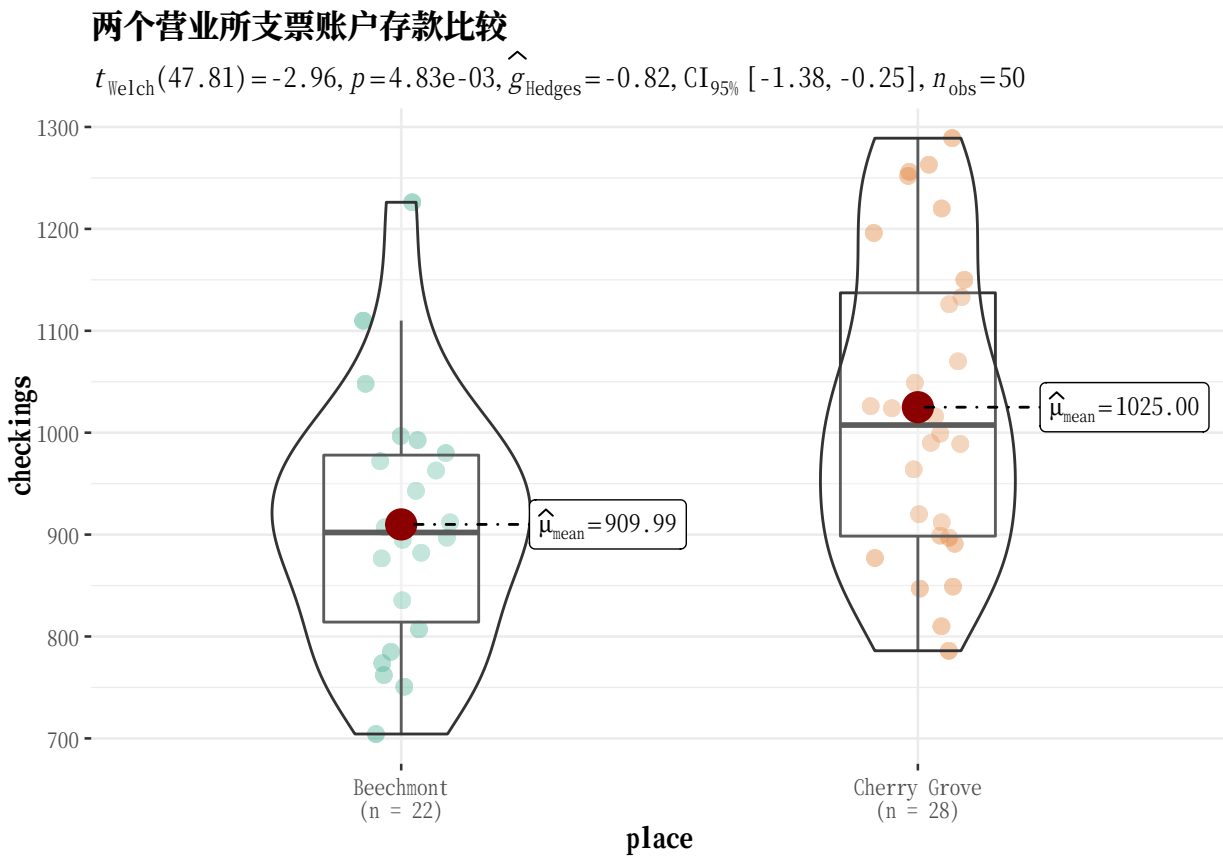
```
t.test(CheckAcct[[1]], CheckAcct[[2]],
  var.equal = FALSE, alternative = "two.sided")
```

```
##
##  Welch Two Sample t-test
##
## data:  CheckAcct[[1]] and CheckAcct[[2]]
## t = 2.956, df = 47.805, p-value = 0.004828
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   36.77503 193.25224
## sample estimates:
## mean of x mean of y
## 1025.0000  909.9864
```

p 值为 0.005，在 0.05 水平下显著，两个营业所的账户平均余额有显著差异。

用 ggstatsplot 包图示：

```
d <- rbind(
  data.frame(place = names(CheckAcct)[1],
    checkings = CheckAcct[[1]]),
  data.frame(place = names(CheckAcct)[2],
    checkings = CheckAcct[[2]]))
library(ggstatsplot)
ggbetweenstats(
  data  = d,
  x     = place,
  y     = checkings,
  title = " 两个营业所支票账户存款比较"
)
```



30.4.4.3 两种工具软件的比较

考虑开发信息系统的两种工具软件的比较。新的软件声称比旧的软件能加快进度。设使用旧软件时平均项目完成时间为 μ_1，使用新软件时为 μ_2。

检验：

$$H_0 : \mu_1 \leq \mu_2 \longleftrightarrow H_a : \mu_1 > \mu_2$$

(对立假设是新软件的工期更短) 取 0.05 水平。

```
t.test(SoftwareTest[['Current']], SoftwareTest[['New']],
       var.equal = FALSE, alternative = 'greater')
```

```
##
##  Welch Two Sample t-test
##
## data:  SoftwareTest[["Current"]] and SoftwareTest[["New"]]
## t = 2.2721, df = 21.803, p-value = 0.01665
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
```

```
##  9.514309      Inf
## sample estimates:
## mean of x mean of y
##       325       286
```

p 值为 0.017, 使用新的工具软件的平均项目完成时间显著地少于使用旧工具软件。

30.4.4.4 两种工艺所需时间的比较

工厂希望比较同一生产问题两种工艺各自需要的时间，将选用时间较短的工艺。可以一组工人用工艺 1，一组工人用工艺 2，随机分组。两组之间工人的个体差异会使得差异的比较误差较大。

所以，让同一个工人分别采用两种工艺，次序先后随机。这样不会引入个体差异的影响，精度较高。使用成对检验。用双侧检验，水平 0.05。

```
t.test(Matched[["Method 1"]], Matched[["Method 2"]],
       paired=TRUE, alternative = "two.sided")
```

```
##
##  Paired t-test
##
## data:  Matched[["Method 1"]] and Matched[["Method 2"]]
## t = 2.1958, df = 5, p-value = 0.07952
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  -0.05120834  0.65120834
## sample estimates:
## mean difference
##             0.3
```

30.4.5 均值比较的样本量和功效计算

针对假定两组方差相等的独立两样本 t 检验，设已知合并的标准差估计 σ，要检验的两组均值差的下界为 δ，可以用 `power.t.test()` 估计需要的样本量，如：

```
power.t.test(
  delta = 2.5, sd = 1.5,
  type = "two.sample",
  alternative = "two.sided",
  sig.level=0.05,
  power=0.90)
```

```
##
##      Two-sample t test power calculation
##
##               n = 8.649245
##           delta = 2.5
##              sd = 1.5
##       sig.level = 0.05
##           power = 0.9
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

这里输出的样本量是两组各自的样本量，假定两组样本量相等。结果表明两组都至少要有 $n = 9$ 个观测。

对于成对 t 检验，只要将 `type` 输入为`"paired"`，因为成对 t 检验实际上是对两个变量的差值进行检验，所以 `sigma` 参数输入的是差值的标准差。

pwr 包可以用来计算功效和样本量。对于独立两样本比较问题，定义要检验的效应大小：

$$d = \frac{\mu_2 - \mu_1}{\sigma},$$

其中 μ_1, μ_2 是两组的均值，σ 是两组共同的标准差，d 为效应大小。给定检验水平、单双侧、效应大小，功效与样本量可以互相唯一决定。

常用的 d 值有：

- 小：$d = 0.2$;
- 中: $d = 0.5$;
- 大: $d = 0.8$。

例 30.2. 考虑两个银行营业所顾客平均存款比较问题，用独立两样本 t 检验。

以估计的均值为两组均值，估计的合并标准差为 σ，计算效应大小：

```
x <- na.omit(CheckAcct[[1]])
y <- na.omit(CheckAcct[[2]])
d <- (mean(y) - mean(x)) / sd(c(x, y)); d
```

```
## [1] -0.7681087
```

计算两组各 $n = (28 + 22)/2 = 25$ 个观测时的功效：

```r
pwr.t.test(
  type="two.sample",
  alternative="two.sided",
  d = -0.7681,
  sig.level = 0.05,
  n = 25 )
```

```
##
##      Two-sample t test power calculation
##
##               n = 25
##               d = 0.7681
##       sig.level = 0.05
##           power = 0.7583484
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

功效为 76%。

计算功效达到 80% 所需样本量：

```r
pwr.t.test(
  type="two.sample",
  alternative="two.sided",
  d = -0.7681,
  sig.level = 0.05,
  power = 0.80 )
```

```
##
##      Two-sample t test power calculation
##
##               n = 27.60138
##               d = 0.7681
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

功效 80% 需要每组有 28 个观测。

但是，在这里我们用了观测到的两组均值的差作为我们要发现的差别。实际上，我们可能需要发现更小一些的效应，这时需要更大的样本量。比如，指定 Cohen 规定的“中等”效应：

```
cohen.ES(test = "t", size = "medium")
```

```
##
##      Conventional effect size from Cohen (1982)
##
##            test = t
##            size = medium
##     effect.size = 0.5
```

```
pwr.t.test(
  type="two.sample",
  alternative="two.sided",
  d = 0.5,
  sig.level = 0.05,
  power = 0.80 )
```

```
##
##      Two-sample t test power calculation
##
##               n = 63.76561
##               d = 0.5
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

要发现中等的效应，需要将样本量增大到每组各 64 个观测。

当要比较的两组的样本量不同时可以用 `pwr.t2n.test()` 计算功效，输入 `n1` 和 `n2` 两个样本量。比如，$n_1 = 28$, $n_2 = 22$，比较中等的均值差异效应：

```
pwr.t2n.test(
  alternative="two.sided",
  d = 0.5,
  sig.level = 0.05,
  n1 = 28, n2 = 22 )
```

```
##
##      t test power calculation
##
##              n1 = 28
##              n2 = 22
##               d = 0.5
##       sig.level = 0.05
##           power = 0.4052476
##     alternative = two.sided
```

功效仅有 41%。

30.4.6 多元均值置信域

待补充。

30.5 单个比例的假设检验和置信区间

30.5.1 方法

设要比较的是总体的某个比例，如选民中对候选人甲的支持率。这时总体是两点分布总体，支持为 1，不支持为 0，参数为 $p = P(X = 1)$。其它的比例问题类似，p 是 “成功概率”。

关于单总体的比例，也有双侧、右侧、左侧检验问题：

$$
\begin{aligned}
H_0 : p = p_0 &\longleftrightarrow H_a : p \neq p_0 \\
H_0 : p \leq p_0 &\longleftrightarrow H_a : p > p_0 \\
H_0 : p \geq p_0 &\longleftrightarrow H_a : p < p_0
\end{aligned}
$$

在 R 中，大样本情况下可以用 `prop.test(x, n, p=p0)` 作单个比例的检验，`n` 是样本量，`x` 是样本中符合条件（如支持）的个数，检验使用近似卡方统计量，用 `alternative` 选项选择双侧、右侧、左侧检验。这个函数也给出比例 p 的近似 $1-\alpha$ 置信区间，用 `conf.level` 选项指定置信度。

大样本时还有

$$
\frac{\hat{p} - p}{\sqrt{p(1-p)/n}}
$$

近似服从标准正态分布，可以据此计算 Z 检验。自定义的 R 函数如下：

```
prop.test.1s <- function(x, n, p=0.5, alternative="two.sided"){
  phat <- x/n
  zstat <- (phat - p)/sqrt(p*(1-p)/n)
```

```
  if(alternative=="two.sided"){   # 双侧检验
    pvalue <- 2*(1 - pnorm(abs(zstat)))
  } else if(alternative=="less"){ # 左侧检验
    pvalue <- pnorm(zstat)
  } else if(alternative=="greater"){ # 右侧检验
    pvalue <- 1 - pnorm(zstat)
  } else {
    stop("alternative unknown!")
  }

  c(stat=zstat, pvalue=pvalue)
}
```

因为比例问题本质上是一个二项分布推断问题，所以 R 还提供了 `binom.test(x, n, p=p0)` 函数进行精确检验并计算精确置信区间，但其结果略保守。用法与 `prop.test()` 类似。

30.5.2 高尔夫培训女生比例检验例子

橡树溪高尔夫培训机构的女性学员比例较少，只有 20%。为了增加女性学员，设计了促销措施。一个月后调查，希望证明女性学员比例增加了。

设女性学员比例为 p，

$$H_0 : p \leq 0.20 \longleftrightarrow H_a : p > 0.20$$

取检验水平 0.05。抽查了 400 名学员，其中 100 名是女性学员，$\hat{p} = 0.25$。

用 `prop.test()` 检验：

```
prop.test(100, 400, p=0.20, alternative = "greater")
```

```
##
##  1-sample proportions test with continuity correction
##
## data:  100 out of 400, null probability 0.2
## X-squared = 5.9414, df = 1, p-value = 0.007395
## alternative hypothesis: true p is greater than 0.2
## 95 percent confidence interval:
##  0.2149649 1.0000000
## sample estimates:
##    p
## 0.25
```

检验 p 值为 0.0074，小于检验水平 0.05，所以女性学员比例已经显著地高于过去的 20% 了。

用自定义的大样本 Z 检验函数 `prop.test.1s()`:

```
prop.test.1s(100, 400, p=0.20, alternative = "greater")
```

```
##        stat      pvalue
## 2.500000000 0.006209665
```

检验 p 值为 0.0062，与基于卡方检验的 `prop.test()` 结果略有差别。

用基于二项分布的 `binom.test()` 检验:

```
binom.test(100, 400, p=0.20, alternative = "greater")
```

```
##
##  Exact binomial test
##
## data:  100 and 400
## number of successes = 100, number of trials = 400, p-value = 0.008595
## alternative hypothesis: true probability of success is greater than 0.2
## 95 percent confidence interval:
##  0.2146019 1.0000000
## sample estimates:
## probability of success
##                   0.25
```

p 值为 0.0086。

30.5.3 单个比例检验的样本量和功效

pwr 包的 `pwr.p.test()` 函数可以进行单个比例假设检验的样本量和功效计算。该函数用参数 `h` 指定要检验的比例差异大小，这个比例差异的效应大小用 `ES.h()` 函数进行计算。比如，$H_0: p = 0.20$, $H_a: p = 0.25$，则比例差异的效应大小为：

```
ES.h(p1 = 0.25, p2 = 0.20)
```

```
## [1] 0.1199023
```

效应大小的计算公式为 ((Cohen, 1988))：

$$h = 2\arcsin\sqrt{p_1} - 2\arcsin\sqrt{p_2}. \tag{30.1}$$

比例差异的常用值为：

- 小 (small): $h = 0.2$;
- 中 (medium): $h = 0.5$;
- 大 (large): $h = 0.8$。

公式(30.1)定义的比例差异效应大小与 p_1 和 p_2 的具体值有关系。比如，同样是 $p_1 - p_2 = 0.05$：

```
ES.h(p1 = 0.10, p2 = 0.05)
```

```
## [1] 0.1924743
```

这个效应比 0.20 和 0.25 的差异效应大了许多。公式(30.1)考虑到了两个比例的相对差距，而不仅是绝对差距。

给定效应大小、检验水平后，可以给定样本量计算功效，如：

```
pwr.p.test(
  h = ES.h(p1 = 0.25, p2 = 0.20),
  sig.level = 0.05,
  alternative = "greater",
  n = 400)
```

```
##
##      proportion power calculation for binomial distribution (arcsine transformation)
##
##               h = 0.1199023
##               n = 400
##       sig.level = 0.05
##           power = 0.774333
##     alternative = greater
```

功效为 77%。计算达到 80% 功效所需的样本量：

```
pwr.p.test(
  h = ES.h(p1 = 0.25, p2 = 0.20),
  sig.level = 0.05,
  alternative = "greater",
  power = 0.80)
```

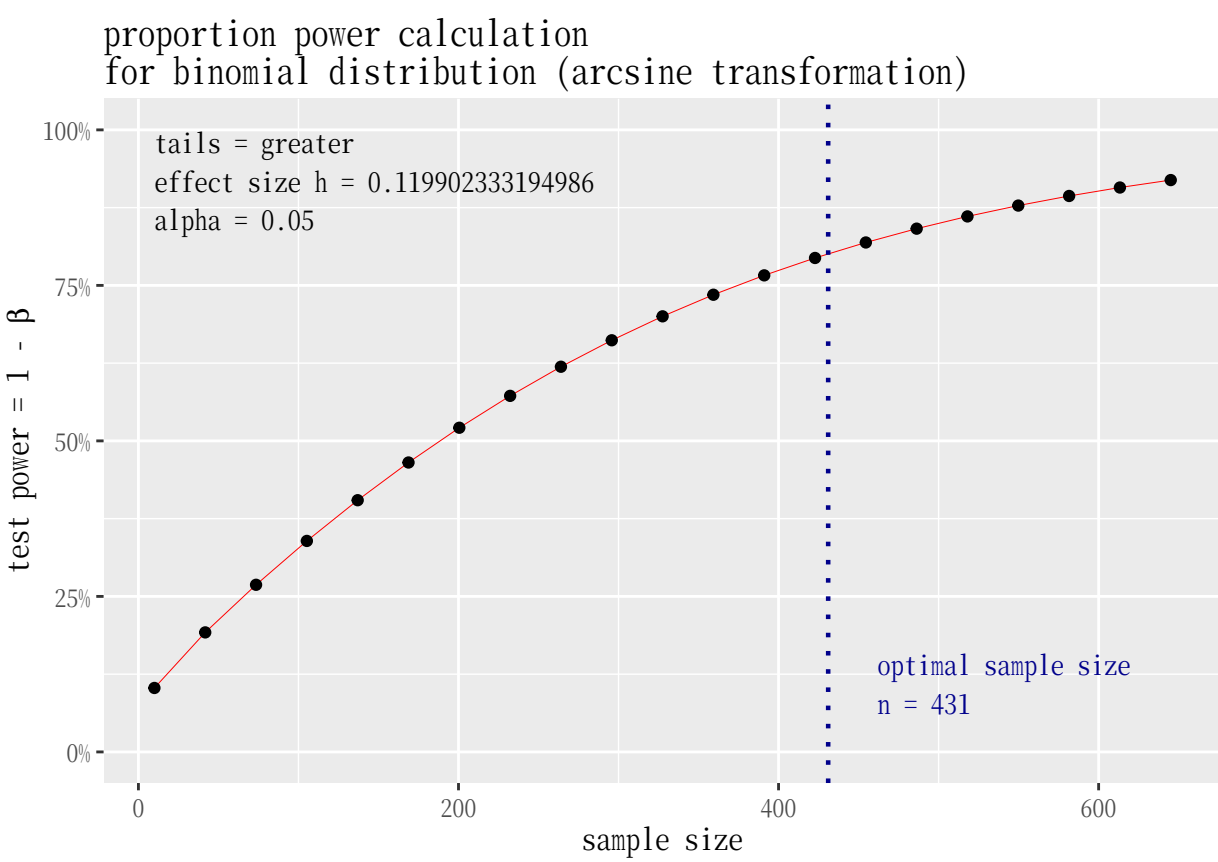
```
##
##      proportion power calculation for binomial distribution (arcsine transformation)
##
##               h = 0.1199023
```

```
##                n = 430.044
##        sig.level = 0.05
##            power = 0.8
##      alternative = greater
```

所需样本量为 431。

还可以图示功效与样本量的关系曲线：

```
pwr.p.test(
  h = ES.h(p1 = 0.25, p2 = 0.20),
  sig.level = 0.05,
  alternative = "greater",
  power = 0.80) |>
  plot()
```



30.6 两个比例的比较

30.6.1 方法

设有 X 和 Y 两个独立的总体，如男性和女性，比较 $p_1 = P(X=1)$ 和 $p_2 = P(Y=1)$。

检验问题包括：

$$H_0 : p_1 = p_2 \longleftrightarrow H_\mathrm{a} : p_1 \neq p_2$$
$$H_0 : p_1 \leq p_2 \longleftrightarrow H_\mathrm{a} : p_1 > p_2$$
$$H_0 : p_1 \geq p_2 \longleftrightarrow H_\mathrm{a} : p_1 < p_2$$

在大样本情况下，可以用 R 函数 `prop.test(c(nsucc1, nsucc2), c(n1,n2), alternative=...)` 进行独立两样本比例的比较检验，其中 `n1` 和 `n2` 是两个样本量，`nsucc1` 和 `nsucc2` 是两个样本中符合特征的个数（个数除以样本量即样本中的比例），`alternative` 的选择也是`"two.sided"`、`"less"`、`"greater"`。

还可以比较 $p_1 - p_2$ 与某个 δ。设两个总体中分别抽取了 n_1 和 n_2 个样本点，样本比例分别为 $\hat{p}_1$ 和 $\hat{p}_2$。估计共同的样本比例

$$\hat{p} = \frac{n_1\hat{p}_1 + n_2\hat{p}_2}{n_1 + n_2}$$

当 $\delta = 0$ 时取统计量

$$Z = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\hat{p}(1-\hat{p})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

当 $p_1 = p_2$ 时 Z 在大样本情况下近似服从标准正态分布。当 $\delta \neq 0$ 时取统计量

$$Z = \frac{\hat{p}_1 - \hat{p}_2 - \delta}{\sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{n_1} + \frac{\hat{p}_2(1-\hat{p}_2)}{n_2}}}$$

当 $p_1 - p_2 = \delta$ 时 Z 在大样本情况下近似服从 N(0,1) 分布。

可以定义 Z 检验 R 函数为：

```
prop.test.2s <- function(x, n, delta=0.0, alternative="two.sided"){
  phat <- sum(x)/sum(n)
  p <- x / n
  if(delta==0.0){
    zstat <- (p[1] - p[2])/sqrt(phat*(1-phat)*(1/n[1] + 1/n[2]))
  } else {
    zstat <- (p[1] - p[2] - delta)/sqrt(p[1]*(1-p[1])/n[1] + p[2]*(1-p[2])/n[2])
  }
  if(alternative=="two.sided"){   # 双侧检验
    pvalue <- 2*(1 - pnorm(abs(zstat)))
  } else if(alternative=="less"){ # 左侧检验
    pvalue <- pnorm(zstat)
  } else if(alternative=="greater"){ # 右侧检验
```

```
    pvalue <- 1 - pnorm(zstat)
  } else {
    stop("alternative unknown!")
  }

  c(stat=zstat, pvalue=pvalue)
}
```

独立两总体比例比较的小样本情形意义不大，可考虑使用 Fisher 精确检验，R 函数为 `fisher.test()`。

30.6.2 报税代理分理处的错误率比较

某个代理报税的机构希望比较下属的两个分理处申报纳税返还的出错率。各自随机选取取了 $n_1 = 250$ 和 $n_2 = 300$ 件申报，第一个分理处错了 35 件，错误率 $\hat{p}_1 = 0.14$; 第二个分理处错了 27 件，错误率 $\hat{p}_2 = 0.09$。

作双侧检验，水平 0.10。

用 `prop.test()`:

```
prop.test(c(35,27), c(250,300), alternative = "two.sided")
```

```
##
##  2-sample test for equality of proportions with continuity correction
##
## data:  c(35, 27) out of c(250, 300)
## X-squared = 2.9268, df = 1, p-value = 0.08712
## alternative hypothesis: two.sided
## 95 percent confidence interval:
##  -0.007506845  0.107506845
## sample estimates:
## prop 1 prop 2
##   0.14   0.09
```

p 值 0.087，有显著差异，第一处的错误率高。因为比例需要较大样本量，所以样本量不太大时，有人用 0.10 这样的检验水平。

用自定义的 `prop.test.2s()`:

```
prop.test.2s(c(35,27), c(250,300), alternative = "two.sided")
```

```
##       stat     pvalue
## 1.84618928 0.06486473
```

使用 Fisher 精确检验：

```
fisher.test(
  rbind(c(35, 250-35),
        c(27, 300-27)),
  alternative = "two.sided")
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  rbind(c(35, 250 - 35), c(27, 300 - 27))
## p-value = 0.07809
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##  0.9345221 2.9208940
## sample estimates:
## odds ratio
##   1.644457
```

`fisher.test()` 要求输入 2×2 矩阵，两行分别为两组的成功数和失败数。上面的检验结果 p 值为 0.078，在 0.10 水平下两个比例有显著差异。

30.6.3 比例比较的样本量

如果预先近似估计了两个总体的比例分别约为 p_1 和 p_2，则检验两个比例有无显著差距所需的样本量可以利用 `power.prop.test()` 计算，如：

```
power.prop.test(
  power = 0.80,
  sig.level = 0.05,
  p1 = 0.15, p2 = 0.30 )
```

```
##
##      Two-sample comparison of proportions power calculation
##
##               n = 120.4719
##              p1 = 0.15
##              p2 = 0.3
##       sig.level = 0.05
##           power = 0.8
```

```
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

输出的样本量是每组需要的样本量。结果表明每组需要 $n = 121$ 个观测。

也可以给定样本量计算功效，如：

```
power.prop.test(
  n = 100,
  sig.level = 0.05,
  p1 = 0.15, p2 = 0.30 )
```

```
##
##      Two-sample comparison of proportions power calculation
##
##               n = 100
##              p1 = 0.15
##              p2 = 0.3
##       sig.level = 0.05
##           power = 0.7222795
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

功效为 72%。

pwr 包的 `pwr.2p.test()` 函数可以进行比例比较的样本量和功效计算。如：

```
pwr.2p.test(
  h = ES.h(p1 = 0.30, p2 = 0.15),
  power = 0.80,
  sig.level = 0.05)
```

```
##
##      Difference of proportion power calculation for binomial distribution (arcsine transformation)
##
##               h = 0.3638807
##               n = 118.5547
##       sig.level = 0.05
##           power = 0.8
```

```
##     alternative = two.sided
##
## NOTE: same sample sizes
```

需要各自 119 个观测。

计算中等比例差异效应的样本量：

```
pwr.2p.test(
  h = 0.5,
  power = 0.80,
  sig.level = 0.05)
```

```
##
##      Difference of proportion power calculation for binomial distribution (arcsine transformation)
##
##               h = 0.5
##               n = 62.79088
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: same sample sizes
```

需要各自 63 个观测值。注意需要检测的比例差异效应越大，需要的样本量就越小。

实际上，当要比较的两个样本样本量不同时，可以用 `pwr.2p2n.test()` 函数来计算功效，将 `n=` 选项替换成 `n1=` 和 `n2=` 选项。比如，$n_1 = 250, n_2 = 300$，检验小的比例差异效应：

```
pwr.2p2n.test(
  h = 0.2,
  n1 = 250, n2 = 300,
  sig.level = 0.05)
```

```
##
##      difference of proportion power calculation for binomial distribution (arcsine transformation)
##
##               h = 0.2
##              n1 = 250
##              n2 = 300
##       sig.level = 0.05
```

```
##           power = 0.6463766
##     alternative = two.sided
##
## NOTE: different sample sizes
```

也可以给定功效以及 n1，求 n2:

```
pwr.2p2n.test(
  h = 0.2,
  n1 = 250,
  power = 0.8,
  sig.level = 0.05)
```

```
##
##      difference of proportion power calculation for binomial distribution (arcsine transformation)
##
##               h = 0.2
##              n1 = 250
##              n2 = 912.1748
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: different sample sizes
```

n_2 为 913。

30.7 方差的假设检验和置信区间

30.7.1 单总体方差的假设检验和置信区间

方差和标准差是总体的重要数字特征，在金融应用中标准差代表波动率，在工业生产中标准差也是重要的质量指标，如机床加工精度可以用标准差表示。

对总体方差 σ^2 可以做双侧、左侧、右侧检验，与某个 σ_0^2 比较。

设总体为正态总体，检验问题：

$$
\begin{aligned}
H_0 :&\sigma^2 = \sigma_0^2 \longleftrightarrow H_\mathrm{a} : \sigma^2 \neq \sigma_0^2 \\
H_0 :&\sigma^2 \geq \sigma_0^2 \longleftrightarrow H_\mathrm{a} : \sigma^2 < \sigma_0^2 \\
H_0 :&\sigma^2 \leq \sigma_0^2 \longleftrightarrow H_\mathrm{a} : \sigma^2 > \sigma_0^2
\end{aligned}
$$

可以用如下的自定义函数计算单正态总体方差的检验：

```
var.test.1s <- function(x, n=length(x), var0, alternative="two.sided"){
  if(length(x)==1){ # 输入的是方差
    varx <- x
  } else {
    varx <- var(x)
  }
  xi <- (n-1)*varx/var0

  if(alternative=="less"){
    pvalue <- pchisq(xi, n-1)
  } else if (alternative=="right"){
    pvalue <- pchisq(xi, n-1, lower.tail=FALSE)
  } else if(alternative=="two.sided"){
    pvalue <- 2*min(pchisq(xi, n-1), pchisq(xi, n-1, lower.tail=FALSE))
  }

  c(statistic=xi, pvalue=pvalue)
}
```

可以输入样本数据到 `x` 中，输入 σ_0^2 到 `var0` 中，进行检验；也可以输入 S^2 到 `x` 中，输入样本量 n 到 `n` 中，输入 σ_0^2 到 `var0` 中，进行检验。

30.7.2 独立两总体方差的比较

设两个独立的正态总体的方差分别为 σ_1^2, σ_2^2，有双侧、左侧、右侧检验：

$$
\begin{aligned}
H_0 :&\sigma_1^2 = \sigma_2^2 \longleftrightarrow H_\mathrm{a} : \sigma_1^2 \neq \sigma_2^2 \\
H_0 :&\sigma_1^2 \geq \sigma_2^2 \longleftrightarrow H_\mathrm{a} : \sigma_1^2 < \sigma_2^2 \\
H_0 :&\sigma_1^2 \leq \sigma_2^2 \longleftrightarrow H_\mathrm{a} : \sigma_1^2 > \sigma_2^2
\end{aligned}
$$

设两个总体分别抽取 n_1 和 n_2 个样本点，样本方差分别为 S_x^2 和 S_y^2。

检验统计量

$$F = \frac{S_x^2}{S_y^2}$$

在 $\sigma_1^2 = \sigma_2^2$ 时 F 服从 $F(n_1 - 1, n_2 - 1)$ 分布。

如果输入 `x, y` 为两个样本的具体值，可以用 `var.test(x, y, alternative=...)` 检验，其中 `alternative` 的选择是`"two.sided"`、`"less"`、`"greater"`。

`var.test()` 假定总体都服从正态分布。R 中 `mood.test()` 是一种不要求正态分布假定的检验方法。`bartlett.test()` 检验多个独立正态总体方差是否全相等。

30.7.3 方差检验例子

例：某车间生产铜丝，生产一向比较稳定。为了检查生产工艺是否受控，从产品中随机抽取 10 根铜丝检查其折断力，得数据如下：

$$578, 572, 570, 568, 572, 570, 570, 572, 596, 584.$$

问：从样本看，能否认为折断力指标的标准差保持在以往的 8.0 水平？

用前面的自定义函数进行方差的卡方检验，作双侧检验：

```
var.test.1s(x = c(
  578, 572, 570, 568, 572,
  570, 570, 572, 596, 584),
  var0 = 64,
  alternative = "two.sided")
```

```
##  statistic     pvalue
## 10.6500000  0.6009287
```

检验 p 值为 0.60，在 0.05 水平下不拒绝零假设，可认为生产工艺水平受控。

例：某车间要比较在两种不同温度下生产的针织品断裂强度。设两种温度下分别抽取的产品样品断裂强度如下：

$$\begin{array}{ll} 70℃ & 20.5, 18.8, 19.8, 20.9, 21.5, 19.5, 21.0, 21.2 \\ 80℃ & 17.7, 20.3, 20.0, 18.8, 19.0, 20.1, 20.2, 19.1 \end{array}$$

问：两种温度下断裂强度的方差有误显著差异？

用 `var.test()`：

```
var.test(c(
  20.5, 18.8, 19.8, 20.9, 21.5, 19.5, 21.0, 21.2), c(
  17.7, 20.3, 20.0, 18.8, 19.0, 20.1, 20.2, 19.1),
  alternative = "two.sided")
```

```
##
##  F test to compare two variances
##
## data:  c(20.5, 18.8, 19.8, 20.9, 21.5, 19.5, 21, 21.2) and c(17.7, 20.3, 20, 18.8, 19, 20.1, 20.2,
## F = 1.069, num df = 7, denom df = 7, p-value = 0.9322
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.214011 5.339386
## sample estimates:
## ratio of variances
##           1.068966
```

检验 p 值为 0.93，在 0.05 水平下不拒绝零假设，认为两种温度下的断裂强度的方差没有显著差异。

30.8 拟合优度检验

30.8.1 各类比例相等的检验

设类别变量 X 有 m 个不同类别，抽取 n 个样本点后，各类的频数分别为 $f_1, f_2, \dots, f_m$。

零假设为所有类的比例相同；对立假设为各类比例不完全相同。

在零假设下，每个类的期望频数为 n/m。

检验统计量为

$$\chi^2 = \sum_{i=1}^{m} \frac{(f_i - \frac{n}{m})^2}{\frac{n}{m}}$$

当 χ^2 超过某一临界值时拒绝零假设。在 H_0 成立且大样本情况下 χ^2 统计量近似服从 $\chi^2(m-1)$ 分布（自由度为组数减 1）。

设 `x` 中包含各类的频数，R 函数 `chisq.test(x)` 作各类的总体比例相等的拟合优度卡方检验。

例：投掷一枚骰子 1000 次，一点到六点的出现次数分别为：168, 159, 168, 180, 167, 158，问这枚骰子是否均匀？

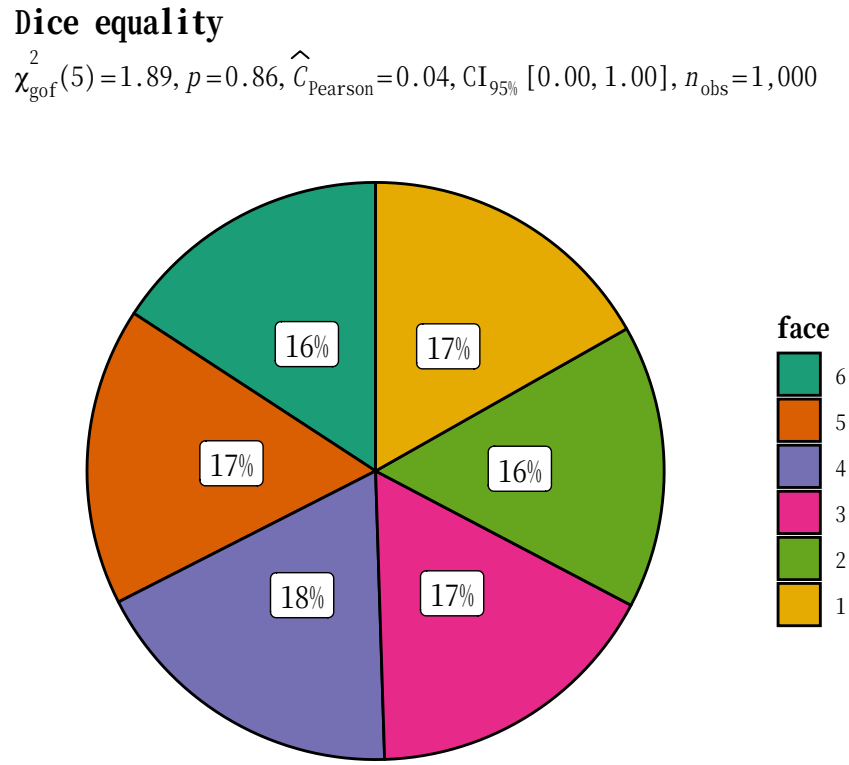
解答：利用 `chisq.test()` 函数，得

```
chisq.test(c(168, 159, 168, 180, 167, 158))
```

```
##
##  Chi-squared test for given probabilities
##
## data:  c(168, 159, 168, 180, 167, 158)
## X-squared = 1.892, df = 5, p-value = 0.8639
```

卡方统计量等于 1.892, 零假设下服从 $\chi^2(5)$ 分布，p 值为 0.8639，结果不显著，不拒绝零假设，认为六个面的概率相等，骰子是均匀的。

```
library(ggstatsplot)
ggpiestats(
  data          = data.frame(face=1:6,
                  counts=c(168, 159, 168, 180, 167, 158)),
  x             = face,
  counts        = counts,
  title         = "Dice equality"
)
```



30.8.2 各类比例为指定值的检验

设分类变量 X 的各类每类有一个总体比例的假设，希望作为零假设检验。

设共有 m 个类，在零假设中，各类的比例分别为 $p_1, p_2, \dots, p_m$。

取了 n 个样本点组成的样本，各类的频数分别为 $f_1, f_2, \dots, f_m$，期望频数分别为 $np_1, np_2, \dots, np_m$。

检验统计量为

$$\chi^2 = \sum_{i=1}^{m} \frac{(f_i - np_i)^2}{np_i}$$

在 H_0 下成立且大样本情况下 χ^2 近似服从 $\chi^2(m-1)$，自由度为组数减 1。因为是大样本检验法，每个组的期望频数不能低于 5，否则可以合并较小的类。

设 `x` 中包含各类的频数，`p` 中包含 H_0 假定的各类的概率，R 函数 `chisq.test(x, p)` 作各类的总体比例为指定概率的拟合优度卡方检验。

例: 市场占有率的检验例子

某类产品的当前市场占比为公司 A 占 30%, 公司 B 占 50%，公司 C 占 20%。近期 C 公司推出了新产品代替本公司的老产品，公司 C 委托一个咨询公司调查分析新产品是否导致了市场占有率的变动。抽查了 200 位消费者，购买三个公司的产品的人数分配为：

$$(48, 98, 54).$$

设 p_1, p_2, p_3 分别为三个公司现在的市场占有率，零假设为:

$$H_0 : p_1 = 0.30,\ p_2 = 0.50,\ p_3 = 0.20$$

取检验水平 0.05。

卡方统计量

$$\chi^2 = \frac{(48 - 200 \times 0.3)^2}{200 \times 0.3} + \frac{(98 - 200 \times 0.5)^2}{200 \times 0.5} + \frac{(54 - 200 \times 0.2)^2}{200 \times 0.2} = 7.34$$

p 值 $= P(\chi^2(3-1) > 7.34) = 0.02548$，拒绝零假设，认为市场占有率有变化。

用 `chisq.test()` 计算：

```
chisq.test(c(48, 98, 54), p=c(0.3, 0.5, 0.2))
```

```
##
##  Chi-squared test for given probabilities
##
## data:  c(48, 98, 54)
## X-squared = 7.34, df = 2, p-value = 0.02548
```

p 值为 0.025，在 0.05 水平下显著，可以认为市场占有率有显著变化。

注意，因为卡方检验本身也是利用了大样本近似分布，所以 p 值结果本身就是近似地，一般仅使用一到两位有效数字，不要报告过多有效数字。

30.8.3 带有未知参数的单分类变量的拟合优度假设检验

如果 H_0 下的概率有未知参数，先用最大似然估计法估计未知参数，然后将未知参数代入计算各类的概率，从而计算期望频数，计算卡方统计量。求 p 值时自由度要改为 “组数减 1 减去未知参数个数”。

例：设某次选举有 5 位候选人，其中呼声最高的是甲和乙。问：甲、乙的支持率相等吗？

这不能用独立的两个比例的比较来解决，因为这两个比例是互斥的，不属于独立情况。

将类别简化为：甲，乙，其他。零假设为 “甲、乙的支持率都等于 p，其他三位候选人的支持率为 $1-2p$，p 未知”；对立假设是甲、乙的支持率不相等。

假设调查了 1000 位选民，其中 300 名支持甲，200 名支持乙，500 名支持其他三位候选人。

在假定零假设成立的情况下先用最大似然估计法估计未知参数 p：$\hat{p} = (300 + 200)/1000/2 = 0.25$。这样，三个类的概率估计为 $(0.25, 0.25, 0.50)$。

然后，计算检验统计量：

```
chisq.test(c(300, 200, 500), c(0.25, 0.25, 0.50))
```

```
## Warning in chisq.test(c(300, 200, 500), c(0.25, 0.25, 0.5)): Chi-squared
## approximation may be incorrect

##
##  Pearson's Chi-squared test
##
## data:  c(300, 200, 500) and c(0.25, 0.25, 0.5)
## X-squared = 3, df = 2, p-value = 0.2231
```

其中的 p 值所用的自由度错误，我们需要自己计算 p 值：

```
res <- chisq.test(c(300, 200, 500), c(0.25, 0.25, 0.50))
```

```
## Warning in chisq.test(c(300, 200, 500), c(0.25, 0.25, 0.5)): Chi-squared
## approximation may be incorrect
```

```
c(statistic=res$statistic,
  pvalue=pchisq(res$statistic, res$parameter - 1,
                lower.tail=FALSE))
```

```
## statistic.X-squared    pvalue.X-squared
##          3.00000000          0.08326452
```

检验 p 值为 0.08 而非原来的 0.22。在 0.05 水平下不显著，结论为两个候选人的支持率没有显著差异。

30.8.4 拟合优度检验的功效和样本量

pwr 包可以计算拟合优度检验的功效和样本量。

30.8.4.1 市场占有率例子

考虑前面的市场占有率的例子。零假设为：

$$H_0 : p_1 = 0.30,\ p_2 = 0.50,\ p_3 = 0.20.$$

设要计算功效的对立假设的具体情况是:

$$H_1 : p_1 = 0.24,\ p_2 = 0.49,\ p_3 = 0.27.$$

据此先计算卡方检验的效应，使用 `ES.w1()` 函数：

```
null <- c(0.30, 0.50, 0.20)
alt <- c(0.24, 0.49, 0.27)
ES.w1(null, alt)
```

```
## [1] 0.1915724
```

卡方检验效应大小的常用取值为：

- 小：0.1;
- 中: 0.3;
- 大: 0.5。

所以 0.19 属于中小规模的效应，需要较大样本量。计算功效大小：

```
pwr.chisq.test(
  w = ES.w1(null, alt),
  N = 200,
  df = 3-1,
  sig.level = 0.05)
```

```
##
##      Chi squared power calculation
##
##               w = 0.1915724
##               N = 200
##              df = 2
##       sig.level = 0.05
##           power = 0.677598
##
## NOTE: N is the number of observations
```

功效为 68%。计算达到 80% 功效所需样本量：

```
pwr.chisq.test(
  w = ES.w1(null, alt),
  power = 0.80,
  df = 4-1,
  sig.level = 0.05)
```

```
##
##      Chi squared power calculation
##
##               w = 0.1915724
##               N = 297.0726
##              df = 3
##       sig.level = 0.05
##           power = 0.8
##
## NOTE: N is the number of observations
```

样本量为 297。

计算中等大小的效应以及 80% 功效所需样本量：

```
cohen.ES(test = "chisq", size = "medium")
```

```
##
##      Conventional effect size from Cohen (1982)
##
##            test = chisq
##            size = medium
##     effect.size = 0.3
```

```
pwr.chisq.test(
  w = 0.3,
  power = 0.80,
  df = 4-1,
  sig.level = 0.05)
```

```
##
##      Chi squared power calculation
##
##               w = 0.3
##               N = 121.1396
##              df = 3
##       sig.level = 0.05
##           power = 0.8
##
## NOTE: N is the number of observations
```

中等的效应大小只需要 122 个观测。

30.8.4.2 候选人例子

考虑前面的 5 个候选人的例子。将第 3、4、5 位候选人合并为一类。零假设为:

$$H_0 : p_1 = p_2 = p, p_3 = 1 - 2p.$$

其中 p 是未知参数。为了计算功效，需要给出对立假设下的一个具体分布。这里仅指定要检验中等的效应。于是功效为：

```
pwr.chisq.test(
  w = 0.3,
  N = 1000,
  df = 3-1-1,
  sig.level = 0.05)
```

```
##
##      Chi squared power calculation
##
##               w = 0.3
##               N = 1000
##              df = 1
##       sig.level = 0.05
##           power = 1
##
## NOTE: N is the number of observations
```

功效为 100%，所以中等大小的效应用 1000 个样本点过于充裕了。

如果按实际数据作为对立假设下的实际分布，对应的效应大小为：

```
null <- c(0.25, 0.25, 0.50)
alt <- c(0.30, 0.30, 0.50)
ES.w1(null, alt)
```

```
## [1] 0.1414214
```

这也对应于较小的效应，意味着较低的功效或者较大的样本量。比如，功效为：

```
pwr.chisq.test(
  w = ES.w1(null, alt),
  N = 1000,
  df = 3-1-1,
  sig.level = 0.05)
```

```
##
##      Chi squared power calculation
##
##               w = 0.1414214
##               N = 1000
##              df = 1
##       sig.level = 0.05
##           power = 0.9940005
##
## NOTE: N is the number of observations
```

因为样本量很大，所以功效达到了 99%。计算达到 80% 功效所需样本量：

```
pwr.chisq.test(
  w = ES.w1(null, alt),
  power = 0.80,
  df = 3-1-1,
  sig.level = 0.05)
```

```
##
##      Chi squared power calculation
##
##               w = 0.1414214
##               N = 392.443
##              df = 1
##       sig.level = 0.05
##           power = 0.8
##
## NOTE: N is the number of observations
```

需要的样本量为 393。

30.9 检验分布类型

vcd 包提供了一个 goodfit 函数，可以用来拟合指定的某种理论分布 (包括泊松、二项、负二项分布)，并检验服从该理论分布的零假设。

例如，我们生成一组速率参数为 2 的泊松随机数，检验其分布是否泊松分布：

```
library(vcd)
```

```
## Loading required package: grid
```

```
set.seed(101)
datax <- rpois(100, 2)
summary(goodfit(datax, "poisson"))
```

```
##
##   Goodness-of-fit test for poisson distribution
##
##                        X^2 df  P(> X^2)
## Likelihood Ratio 4.289456  5 0.5085374
```

检验其是否服从二项分布，取二项分布试验数为 10：

```
summary(goodfit(datax, "binomial", par = list(size = 10)))
```

```
##
##   Goodness-of-fit test for binomial distribution
##
##                        X^2 df  P(> X^2)
## Likelihood Ratio 5.052451  5 0.4095126
```

与二项分布也相容。在检验分布类型时，这种两种分布都不能拒绝的情况是常见的。

30.10 列联表独立性卡方检验

30.10.1 检验方法

设分类变量 X 有 m 个类，分类变量 Y 有 k 个类，**随机**抽取了样本量为 n 的样本，设 X 的第 i 类与 Y 的第 j 类交叉的频数为 f_{ij}。

零假设为 X 与 Y 相互独立，即行变量与列变量相互独立。

交叉频数列成列联表形式，第 i 行第 j 列为 f_{ij}。共有 mk 个单元格，将这 mk 个单元格的频数值看成是来自有 mk 个类别的多项分布的样本的频数结果。注意，如果样本不是随机抽取，而是按照 X 或者安装 Y 分类观察或者试验得到的，结果就不能看作是来自 mk 个类别的多项分布样本，这里的方法就不适用。

第 i 行的行和为 $f_{i\cdot}$, X 的第 i 类的百分比为 $r_i = f_{i\cdot}/n$;

第 j 列的列和为 $f_{\cdot j}$, Y 的第 j 类的百分比为 $c_j = f_{\cdot j}/n$。

当 X, Y 独立时，(i, j) 格子的期望频数为 $E_{ij} = n \times r_i \times c_j$。

列联表独立性检验统计量

$$\chi^2 = \sum_{i=1}^{m} \sum_{j=1}^{k} \frac{(f_{ij} - E_{ij})^2}{E_{ij}}$$

其中 $E_{ij} = nr_i c_j$，r_i 为 X 第 i 类的百分比，c_j 为 Y 第 j 类的百分比。

在 H_0 下 χ^2 近似服从 $(m-1)(k-1)$ 自由度的卡方分布。设统计量值为 c，p 值为 $P(\chi^2((m-1)(k-1)) > c)$。

如果 `x`, `y` 分别是两个变量的原始观测值，`chisq.test(x, y)` 可以做列联表独立性检验。如果 `x` 保存了矩阵格式的列联表，矩阵行名是 X 各个类的名称，矩阵列名是 Y 各个类的名称，则 `chisq.test(x)` 可以做列联表独立性检验。

列联表卡方检验法的检验统计量在零假设下的卡方分布是大样本情况的近似分布。如果每个变量仅有两个类，每个类的期望频数不能少于 5; 如果有多个单元格，期望频数少于 5 的单元格的个数不能超过 20%，否则应该合并较小的类。

如果列联表的数据不是来自于总数 n 固定的随机抽样，而是按照 X 变量分组抽取的，这时零假设就变成了 m 个独立的总体的多项分布是否相同的问题，这个问题恰好也可以使用上面的卡方统计量，近似分布的自由度不变。如果数据按 Y 变量分组抽取也一样。

30.10.2 列联表独立性卡方检验例子

30.10.2.1 性别与啤酒种类的独立性检验

Alber's Brewery of Tucson, Arizona 是啤酒制造与销售商。有三类啤酒产品：淡啤酒，普通啤酒，黑啤酒。

了解不同顾客喜好有利于制定更精准的销售策略。希望了解男女顾客对不同类型的偏好有没有显著差异，实际就是检验性别与啤酒类型偏好的独立性。

随机抽取了 150 位顾客，得到如下的列联表：

```
ctab.beer <- rbind(c(
  20, 40, 20),
  c(30,30,10))
colnames(ctab.beer) <- c("Light", "Regular", "Dark")
rownames(ctab.beer) <- c("Male", "Female")
addmargins(ctab.beer)
```

```
##        Light Regular Dark Sum
## Male      20      40   20  80
```

```
## Female    30      30   10  70
## Sum       50      70   30 150
```

列联表独立性检验：

```
chisq.test(ctab.beer)
```

```
##
##  Pearson's Chi-squared test
##
## data:  ctab.beer
## X-squared = 6.1224, df = 2, p-value = 0.04683
```

在 0.05 水平下认为啤酒类型偏好与性别有关。男性组的偏好分布、女性组的偏好分布、所有人的偏好分布：

```
tab2 <- round(prop.table(addmargins(ctab.beer, 1), 1), 3)
rownames(tab2)[3] <- "All"
tab2
```

```
##        Light Regular  Dark
## Male   0.250   0.500 0.250
## Female 0.429   0.429 0.143
## All    0.333   0.467 0.200
```

可以看出男性中对淡啤酒偏好偏少，女性中对淡啤酒偏好偏多。

30.10.3 功效与样本量计算

在上述啤酒种类偏好的例子中，假设要在 0.01 水平下检验出中等的效应大小，样本量 $N = 150$，计算其功效如下：

```
cohen.ES(test = "chisq", size = "medium")
```

```
##
##      Conventional effect size from Cohen (1982)
##
##            test = chisq
##            size = medium
##     effect.size = 0.3
```

```
pwr.chisq.test(
  w = 0.3,
  N = 150,
  df = (2-1)*(3-1),
  sig.level = 0.05)
```

```
##
##      Chi squared power calculation
##
##               w = 0.3
##               N = 150
##              df = 2
##       sig.level = 0.05
##           power = 0.9185674
##
## NOTE: N is the number of observations
```

功效为 92%。

如果明确指定了对立假设，则可以用 `ES.w2()` 计算所对应的效应大小。比如，零假设是性别与啤酒种类独立，而对立假设是性别与偏好的如下概率分布：

Gender \ Beer	Light	Regular	Dark
Male	2/15	4/15	2/15
Female	3/15	3/15	1/15

就需要按此计算要检验的效应大小：

```
dist_tab <- matrix(
  c(2, 4, 2,
    3, 3, 1)/15,
  byrow=TRUE, ncol=3,
  dimnames = list(
    c("Male", "Female"),
    c("Light", "Regular", "Dark")))
dist_tab
```

```
##            Light   Regular       Dark
## Male   0.1333333 0.2666667 0.13333333
## Female 0.2000000 0.2000000 0.06666667
```

```
ES.w2(dist_tab)
```

```
## [1] 0.2020305
```

计算此效应大小对应的功效：

```
pwr.chisq.test(
  w = ES.w2(dist_tab),
  N = 150,
  df = (2-1)*(3-1),
  sig.level = 0.05)
```

```
##
##      Chi squared power calculation
##
##               w = 0.2020305
##               N = 150
##              df = 2
##       sig.level = 0.05
##           power = 0.5932776
##
## NOTE: N is the number of observations
```

因为要发现的效应更小，所以达到的功效 (59%) 较低。这说明，如果啤酒偏好数据中反映的比例是真实情况的话，实际上有 41% 的概率会不拒绝零假设。

计算达到 80% 功效所需的样本量：

```
pwr.chisq.test(
  w = ES.w2(dist_tab),
  power = 0.80,
  df = (2-1)*(3-1),
  sig.level = 0.05)
```

```
##
##      Chi squared power calculation
##
##               w = 0.2020305
##               N = 236.0499
##              df = 2
```

```
##       sig.level = 0.05
##           power = 0.8
##
## NOTE: N is the number of observations
```

需要的样本量为 236。

30.11 非参数检验

常用的有独立两样本比较的 Wilcoxon 秩和检验，单样本的符号秩检验和符号检验等，

30.11.1 单样本的符号秩检验

`wilcox.test(x, mu)` 进行符号秩检验，适用于非正态总体，检验 $H_0 : \mu = \mu_0$。

例如，Heathrow 机场打分是否在 7 分以上的检验：

```
wilcox.test(AirRating[["Rating"]], mu=7, alternative="greater")
```

```
## Warning in wilcox.test.default(AirRating[["Rating"]], mu = 7, alternative =
## "greater"): cannot compute exact p-value with ties

## Warning in wilcox.test.default(AirRating[["Rating"]], mu = 7, alternative =
## "greater"): cannot compute exact p-value with zeroes

##
##  Wilcoxon signed rank test with continuity correction
##
## data:  AirRating[["Rating"]]
## V = 457, p-value = 0.0469
## alternative hypothesis: true location is greater than 7
```

p 值为 0.047，在 0.05 水平下拒绝原假设，认为打分显著超过 7 分。

非参数检验一般功效较低，这里的 p 值 0.047 就比用 t 检验得到的 0.035 的 p 值大一些。p 值越大，越不容易得到显著的检验结果。

30.11.2 独立两样本比较的 Wilcoxon 秩和检验

`wilcox.test(x, y)` 可以对独立两组进行位置参数的比较，这是一个非参数检验。

例如，对两个银行营业所顾客平均存款的比较：

```
wilcox.test(CheckAcct[[1]], CheckAcct[[2]],
  alternative = "two.sided")
```

```
## Warning in wilcox.test.default(CheckAcct[[1]], CheckAcct[[2]], alternative =
## "two.sided"): cannot compute exact p-value with ties

##
##  Wilcoxon rank sum test with continuity correction
##
## data:  CheckAcct[[1]] and CheckAcct[[2]]
## W = 447, p-value = 0.00679
## alternative hypothesis: true location shift is not equal to 0
```

p 值为 0.007，小于 0.05，两个营业所的顾客平均存款有显著差异。

30.11.3 成对比较的 Wilcoxon 配对检验

对成对比较问题，可以计算两个变量的差进行符号秩检验，称为 Wilcoxon 配对检验。

例如，同一组工人比较两种工艺的时间的问题：

```
wilcox.test(Matched[["Method 1"]], Matched[["Method 2"]],
       paired=TRUE, alternative = "two.sided")
```

```
## Warning in wilcox.test.default(Matched[["Method 1"]], Matched[["Method 2"]], :
## cannot compute exact p-value with zeroes

##
##  Wilcoxon signed rank test with continuity correction
##
## data:  Matched[["Method 1"]] and Matched[["Method 2"]]
## V = 14, p-value = 0.1056
## alternative hypothesis: true location shift is not equal to 0
```

p 值为 0.11，超过 0.05，认为两种工艺时间的差异不显著。

30.12 使用 infer 包计算假设检验和置信区间

R 的 infer 扩展包提供了与 tidyverse 系统习惯做法一致的进行假设检验的方法。在进行理论推断时，主要使用随机模拟方法进行计算，也支持基于理论分布的方法。这个包的当前版本（1.2.9001）还有一些错误，不能用于较正式的研究问题。

以数据框（tibble）为输入，用动词 `specify` 指定针对的变量，用 `hypothesis` 指定假设检验（包括置信区间所针对的参数），用 `generate` 生成零假设下的多组模拟样本（可以是 bootstrap 样本，也可以是理论分布而不需要模拟），用 `calculate` 计算统计量或者对多组模拟样本计算产生满足检验统计量（或枢轴量）抽样分布的随机样本，或者用 `assume` 指定理论抽样分布。从抽样分布的样本或理论分布，可以用 `visualize` 查看抽样分布直方图，用 `shaded_p_value` 在抽样分布直方图中标出统计量值位置并加亮显示 p 值对应面积，用 `get_p_value` 计算 p 值，用 `get_confidence_interval` 计算置信区间，用 `shaded_confidence_interval` 在抽样分布直方图中加亮显示置信区间。

30.12.1 单样本均值的比较

考虑咖啡标重的检验问题。

30.12.1.1 随机模拟方法

infer 包主要使用随机模拟方法生成多组随机样本，产生服从检验统计量抽样分布的样本，然后计算推断结果。

计算样本均值：

```
xbar <- Coffee |>
  specify(response = Weight) |>
  calculate(stat = "mean")
xbar
```

```
## Response: Weight (numeric)
## # A tibble: 1 x 1
##    stat
##   <dbl>
## 1  2.92
```

用 `specify` 在数据框中指定要分析的变量。在 `calculate` 中指定为 `stat = "mean"`，则计算均值统计量。

用 bootstrap 方法生成多组样本，然后对每个样本计算样本均值，得到服从检验统计量抽样分布的样本（近似）：

```
null_dist <- Coffee |>
  specify(response = Weight) |>
  hypothesize(null = "point", mu = 3) |>
  generate(reps = 1000) |>
  calculate(stat = "mean")
```

```
## Setting `type = "bootstrap"` in `generate()`.
```

`hypothesize()` 中的 `null = "point"`，就是指单样本的某个参数的检验问题，用 `mu = 3` 指定了零假设是 H_0：$\mu = 3$。用 `generate()` 按照 bootstrap 方法生成多组再抽样随机样本。用 `calculate()` 对每一组再抽样随机样本计算均值统计量，作为 bootstrap 方法得到的样本均值的抽样分布近似。

计算左侧 p 值：

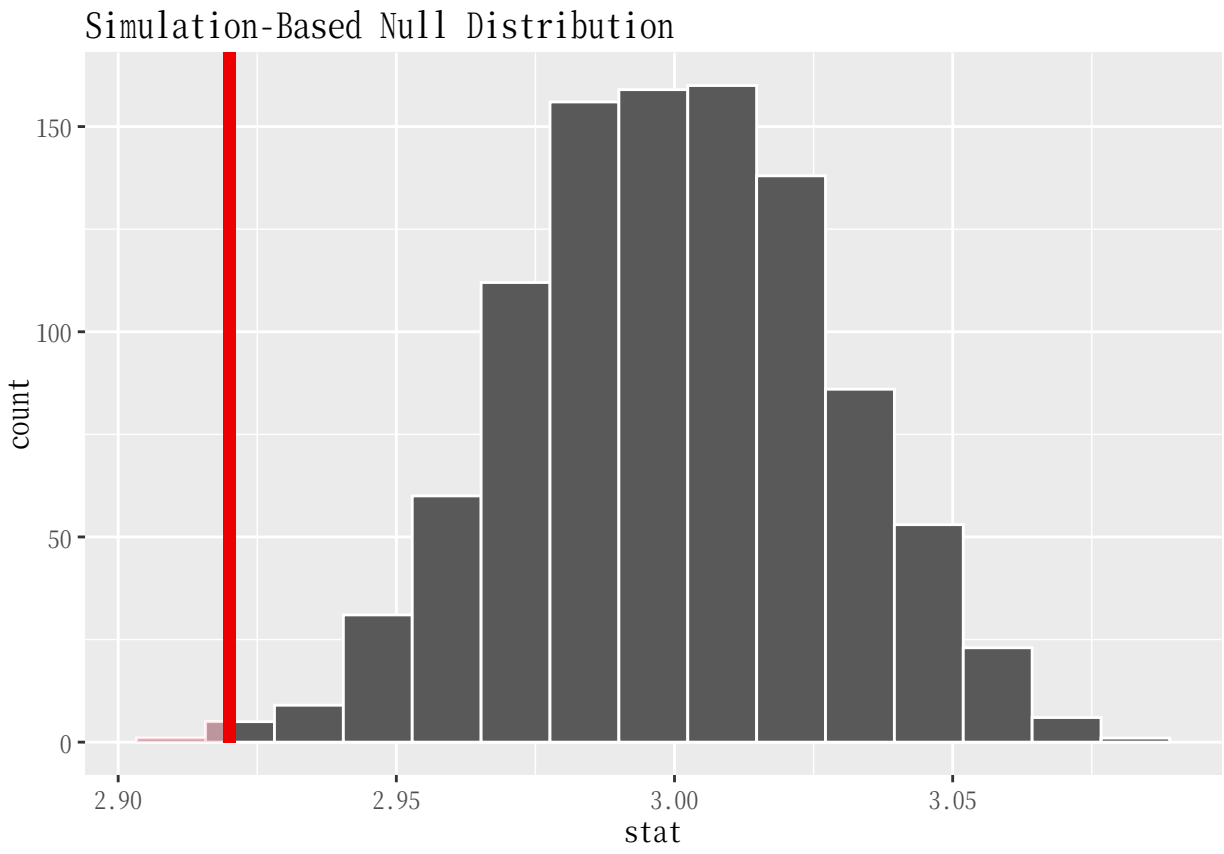
```
null_dist |>
  get_p_value(obs_stat = xbar, direction = "less")
```

```
## # A tibble: 1 x 1
##   p_value
##     <dbl>
## 1   0.002
```

程序中输入了均值统计量的 bootstrap 分布（用一个很大的 bootstrap 均值统计量的随机样本代表）和实际数据的均值统计量，计算了实际数据的均值统计量在 bootstrap 分布中的左侧概率作为 p 值。`direction` 可取`"less"`(左侧)、`"greater"`(右侧) 和`"two-sided"`(双侧，或`"both"`)。

对 $\bar{x}$ 抽样分布（bootstrap 近似）抽样分布作直方图，并标出 $\bar{x}$ 值，高亮显示 p 值对应的面积：

```
null_dist |>
  visualise() +
  shade_p_value(obs_stat = xbar, direction = "less")
```



计算置信区间，用 bootstrap 样本分位数：

```
null_dist |>
  get_confidence_interval(
    level = 0.95, type = "percentile")
```

```
## # A tibble: 1 x 2
##   lower_ci upper_ci
##      <dbl>    <dbl>
## 1     2.95     3.05
```

计算置信区间，用 bootstrap 样本计算的标准误差：

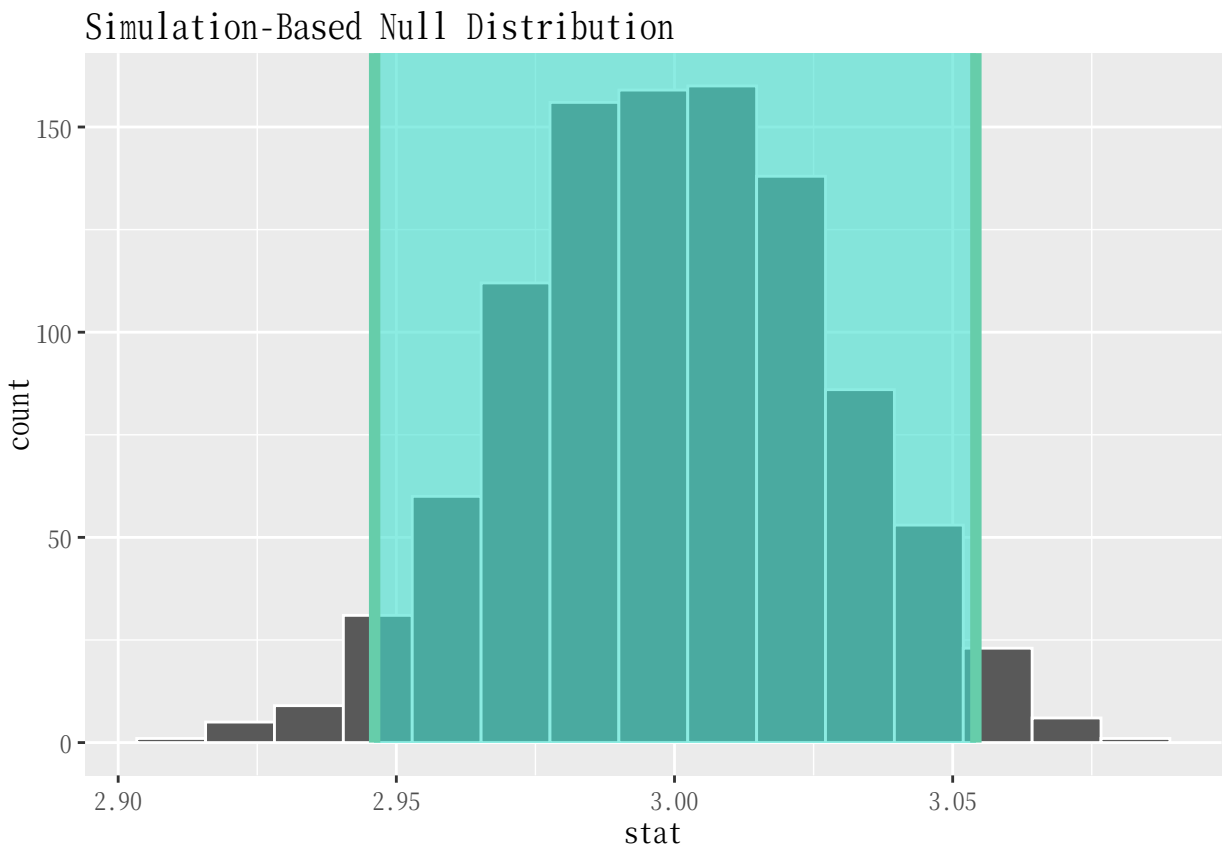
```
null_dist |>
  get_confidence_interval(
    level = 0.95, type = "se",
    point_estimate = xbar)
```

```
## # A tibble: 1 x 2
##   lower_ci upper_ci
```

```
##       <dbl>     <dbl>
## 1      2.87      2.97
```

置信区间作图：

```
ci <- null_dist |>
  get_confidence_interval(
    level = 0.95, type = "percentile")
null_dist |>
  visualise() +
  shade_confidence_interval(ci)
```



30.12.1.2 理论方法

用 t 检验的理论方法，先计算统计量：

```
t_stat <- Coffee |>
  specify(response = Weight) |>
  hypothesize(null = "point", mu = 3) |>
```

```
  calculate(stat = "t")
t_stat
```

```
## Response: Weight (numeric)
## Null Hypothesis: point
## # A tibble: 1 x 1
##    stat
##   <dbl>
## 1 -2.82
```

指定检验统计量理论分布：

```
t_dist <- Coffee |>
  specify(response = Weight) |>
  assume(distribution = "t")
```

计算左侧检验的 p 值：

```
t_dist |>
  get_p_value(obs_stat = t_stat, direction = "less")
```

```
## # A tibble: 1 x 1
##   p_value
##     <dbl>
## 1 0.00389
```

p 值为 0.004，在 0.01 水平下拒绝零假设，认为该品牌咖啡实际重量显著低于标称重量。

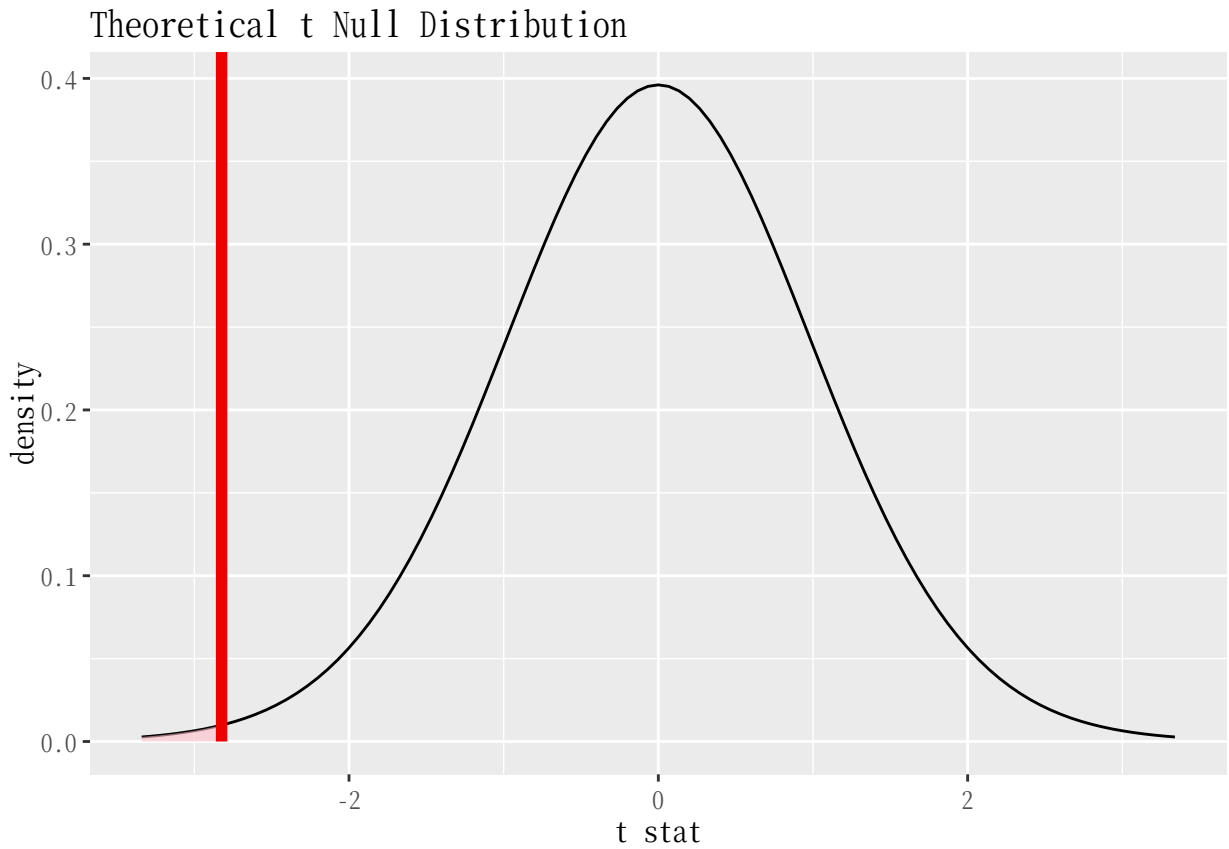
计算 μ 的置信区间：

```
xbar <- Coffee |>
  specify(response = Weight) |>
  calculate(stat = "mean")
t_dist |>
  get_confidence_interval(level = 0.95,
    point_estimate = xbar)
```

```
## # A tibble: 1 x 2
##   lower_ci upper_ci
##      <dbl>    <dbl>
## 1     2.86     2.98
```

作统计量抽样分布图，并标出统计量值，加亮显示 p 值对应面积：

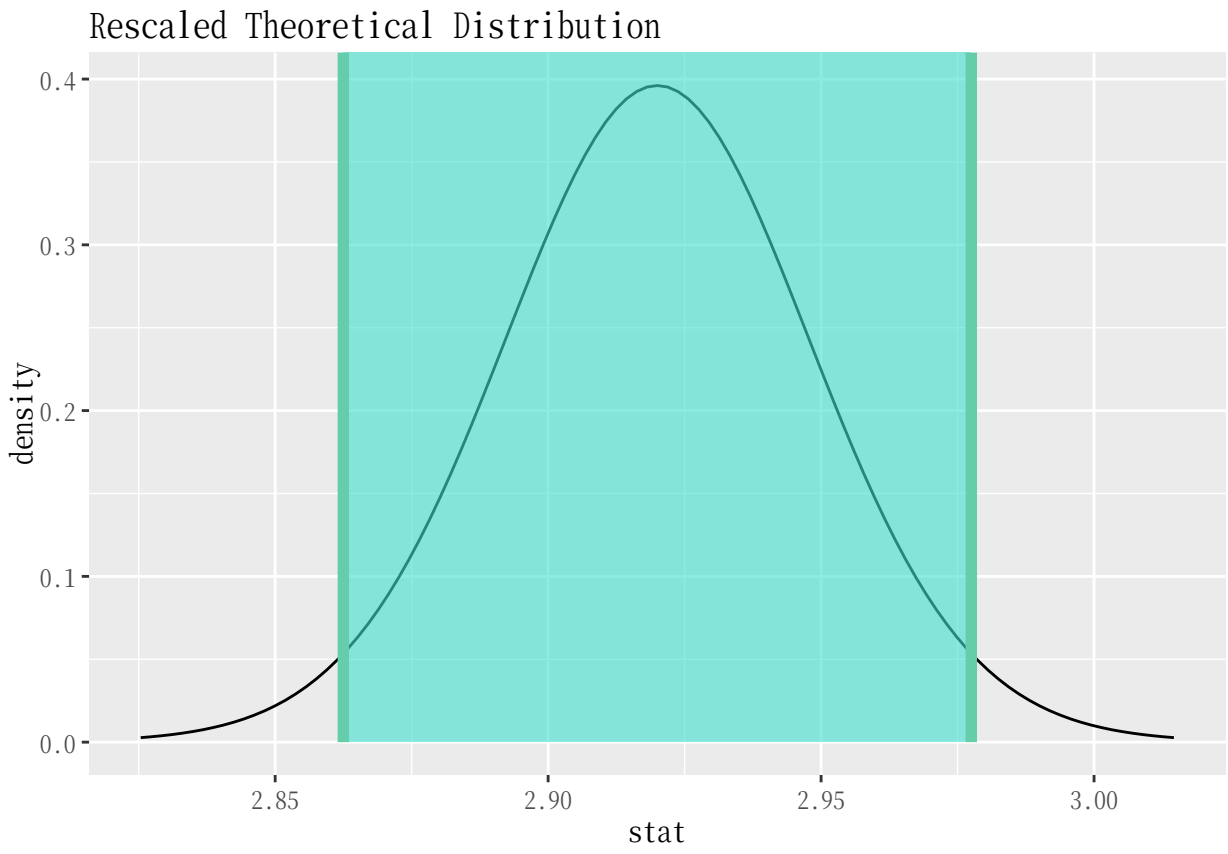
```
t_dist |>
  visualise() +
  shade_p_value(obs_stat = t_stat, direction = "less")
```



注意 `visualize` 和 `shade_p_value` 之间用 `+` 号而不是 `|>` 连接。

作置信区间图并与 $\bar{x}$ 抽样分布叠加：

```
xbar <- Coffee |>
  specify(response = Weight) |>
  calculate(stat = "mean")
ci <- t_dist |>
  get_confidence_interval(level = 0.95,
    point_estimate = xbar)
t_dist |>
  visualise() +
  shade_confidence_interval(ci)
```



函数 `t_test()` 提供了 t 检验和置信区间计算功能，如：

```
Coffee |>
  t_test(
    Weight ~ NULL,
    alternative = "less",
    mu = 3)
```

```
## # A tibble: 1 x 7
##   statistic  t_df p_value alternative estimate lower_ci upper_ci
##       <dbl> <dbl>   <dbl> <chr>          <dbl>    <dbl>    <dbl>
## 1     -2.82    35 0.00389 less            2.92     -Inf     2.97
```

30.12.2 独立两样本均值比较

考虑前面的两个银行营业所账户平均存款比较问题。需要将数据转换为长表格式，即分组变量占一列，存款占一列：

```
da_bank <- CheckAcct |>
  pivot_longer(all_of(c("Cherry Grove", "Beechmont")),
```

```
    names_to = "Shop",
    values_to = "Balance",
    values_drop_na = TRUE)
```

计算均值的差：

```
obs_stat <- da_bank |>
  specify(Balance ~ Shop) |>
  calculate(stat = "diff in means",
    order = c("Cherry Grove", "Beechmont"))
obs_stat
```

```
## Response: Balance (numeric)
## Explanatory: Shop (factor)
## # A tibble: 1 x 1
##    stat
##   <dbl>
## 1  115.
```

在 calculate 中指定了要计算的统计量是均值的差，并且次序是"Cherry Grove" 营业所均值减去"Beechmont" 营业所均值。

用两组随机置换方法生成多组随机样本，作为零假设下的样本，计算多个均值差统计量，代表零假设下均值差统计量的分布：

```
null_dist <- da_bank |>
  specify(Balance ~ Shop) |>
  hypothesize(null = "independence") |>
  generate(reps = 1000) |>
  calculate(stat = "diff in means",
    order = c("Cherry Grove", "Beechmont"))
```

```
## Setting `type = "permute"` in `generate()`.
```

其中 hypothesize(null = "independence") 表示零假设是独立两组比较问题，这里配合 calculate 中 stat = "diff in means"，可知零假设是独立两组均值的比较问题。generate() 生成零假设下的多个样本，使用了置换检验方法，即保持样本中 Balance(账户存款额) 不变，但是将所有 n 个（n 为样本量）营业所标签 (Shop) 随机打乱重排，这样就抹去了两个营业所的区别，得到的新样本代表了零假设成立情况下的分布。

计算双侧检验 p 值：

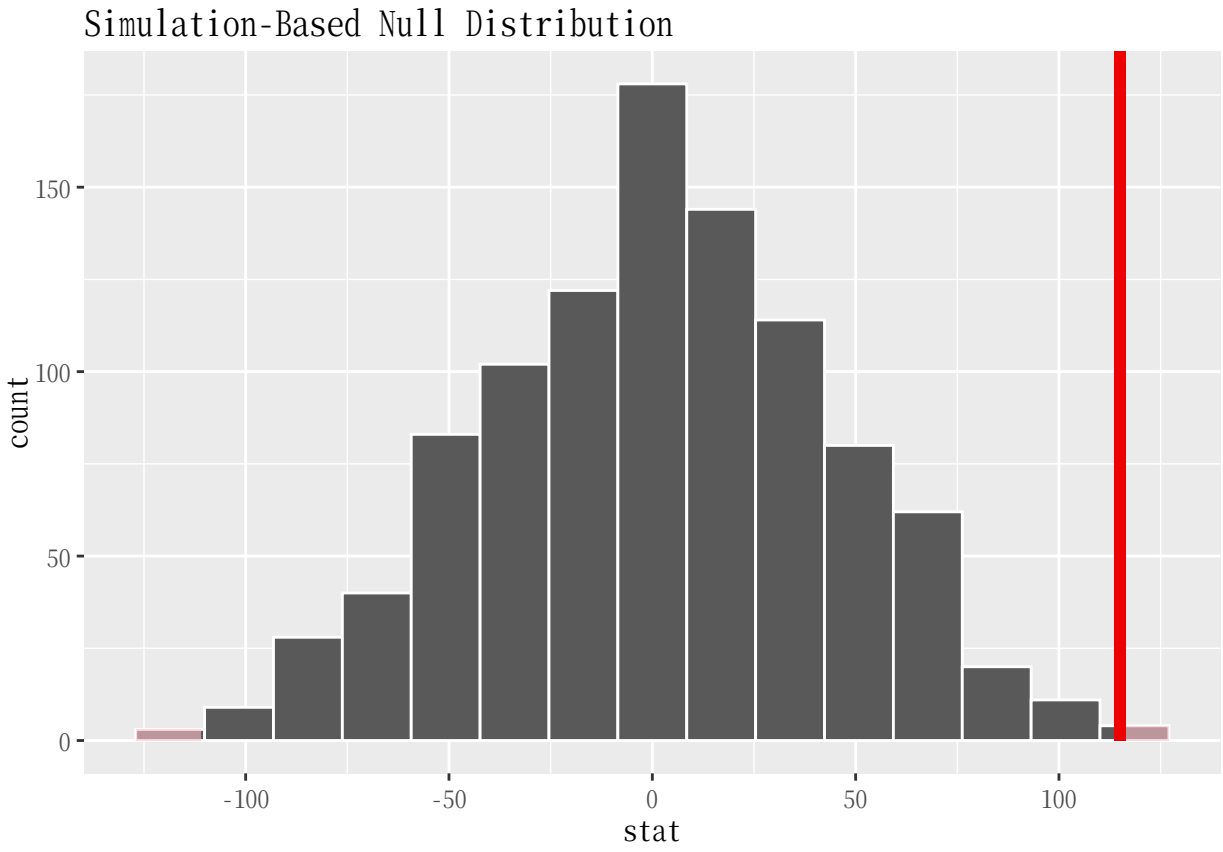
```
null_dist |>
  get_p_value(obs_stat = obs_stat,
    direction = "two-sided")
```

```
## # A tibble: 1 x 1
##   p_value
##     <dbl>
## 1   0.004
```

这是独立两组均值比较的置换检验方法。置换检验不需要通常的基于理论的独立两样本 t 检验方法要求两组服从正态分布以及方差相等等前提，仍需要假定两组独立。所以置换检验比独立两样本 t 检验更为适用。

作两均值统计量差的零假设下分布直方图，并标出 p 值对应的面积：

```
null_dist |>
  visualise() +
  shade_p_value(obs_stat = obs_stat,
    direction = "two-sided")
```



infer 也提供了作独立两样本 t 检验的函数 `t_test()`，如：

```
da_bank |>
  t_test(
    Balance ~ Shop,
    order = c("Cherry Grove", "Beechmont"),
    direction = "two-sided")
```

```
## # A tibble: 1 x 7
##   statistic  t_df p_value alternative estimate lower_ci upper_ci
##       <dbl> <dbl>   <dbl> <chr>          <dbl>    <dbl>    <dbl>
## 1      2.96  47.8 0.00483 two.sided       115.     36.8     193.
```

30.12.3 成对均值比较

成对均值比较，实际是成对均值的差值与 0 的比较。

以前面两种工艺时间的比较为例。使用单样本的 bootstrap 检验。

计算成对的差值，以及差值的均值：

```
da_proc <- Matched |>
  mutate(time_diff = `Method 1` -`Method 2`)
diff_mean <- da_proc |>
  specify(response = time_diff) |>
  calculate(stat = "mean")
diff_mean
```

```
## Response: time_diff (numeric)
## # A tibble: 1 x 1
##    stat
##   <dbl>
## 1   0.3
```

生成 bootstrap 样本：

```
boot_dist <- da_proc |>
  specify(response = time_diff) |>
  hypothesize(null = "point", mu = 0) |>
  generate(reps = 1000) |>
  calculate(stat = "mean")
```

```
## Setting `type = "bootstrap"` in `generate()`.
```

计算双侧 p 值：

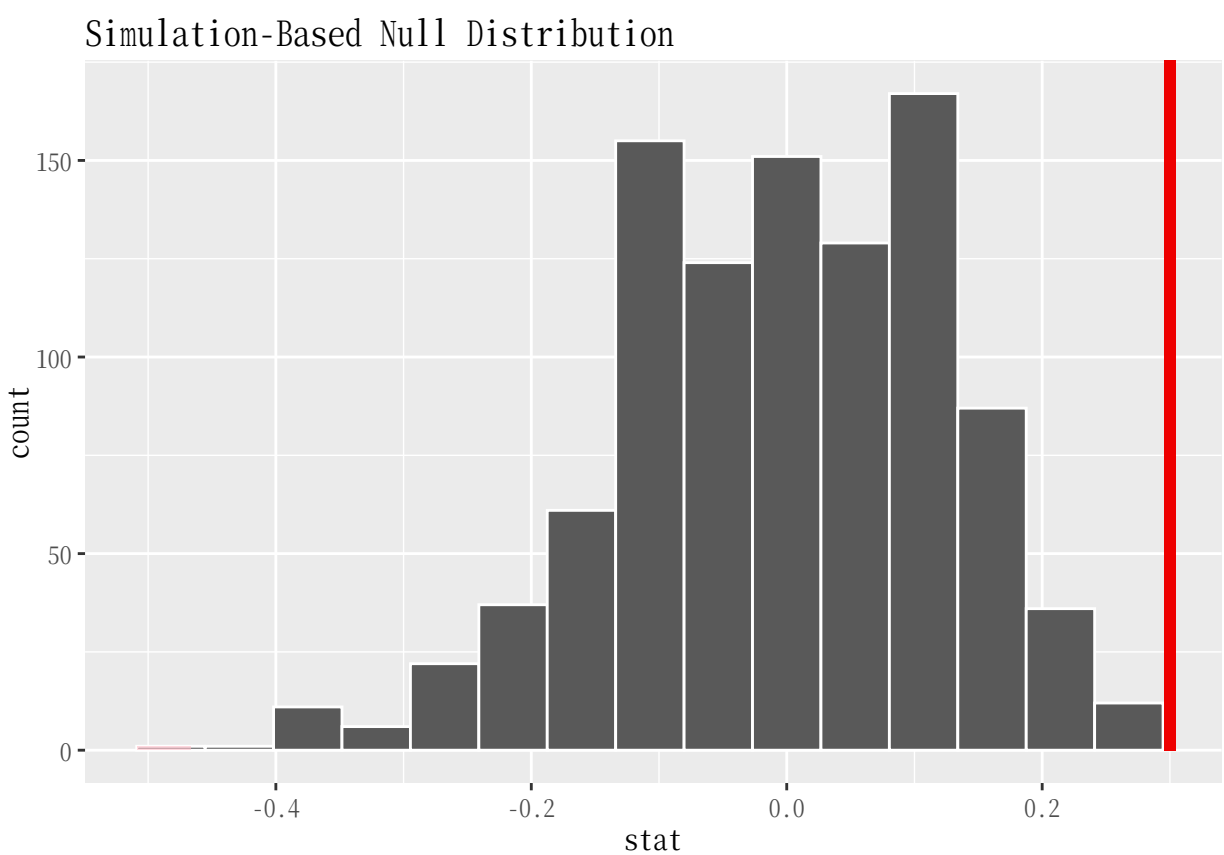
```
boot_dist |>
  get_p_value(obs_stat = diff_mean,
    direction = "two-sided")
```

```
## Warning: Please be cautious in reporting a p-value of 0. This result is an
## approximation based on the number of `reps` chosen in the `generate()` step.
## See `?get_p_value()` for more information.
```

```
## # A tibble: 1 x 1
##   p_value
##     <dbl>
## 1       0
```

图示：

```
boot_dist |>
  visualize() +
  shade_p_value(obs_stat = diff_mean,
    direction = "two-sided")
```



30.12.4 单样本比例的比较

为了使用随机模拟方法，需要有二值（是、否）的观测样本。可以从汇总统计反向生成这样的样本。

考虑前面的高尔夫培训女生比例的例子。按汇总数据生成原始数据：

```
FemGolf <- tibble(
  gender = rep(c("F", "M"), c(100, 300)))
```

计算比例统计量：

```
p_stat <- FemGolf |>
  specify(response = gender, success = "F") |>
  calculate(stat = "prop")
p_stat
```

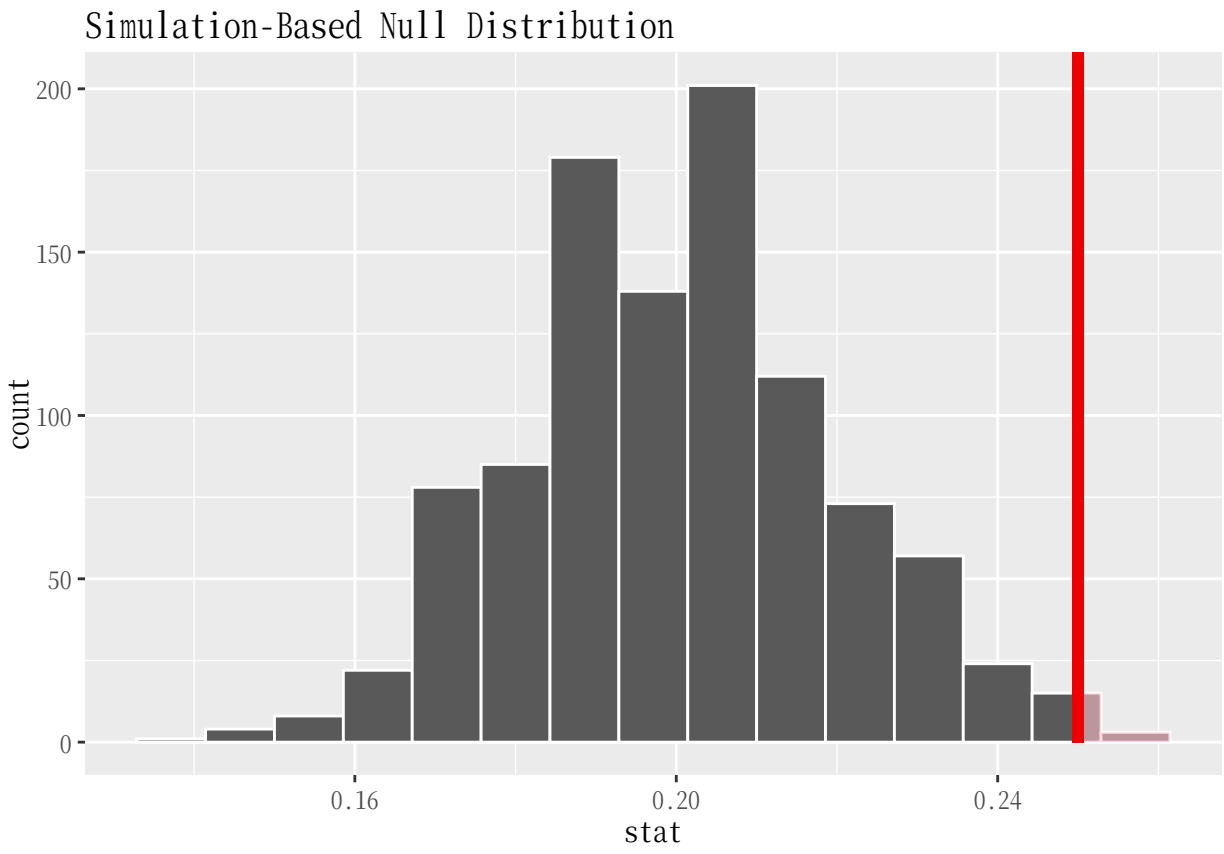
```
## Response: gender (factor)
## # A tibble: 1 x 1
##    stat
##   <dbl>
## 1  0.25
```

生成满足零假设的多组样本（真实比例为 0.20 的样本），从每个样本计算比例统计量并作图：

```
null_dist <- FemGolf |>
  specify(response = gender, success = "F") |>
  hypothesize(null = "point", p = 0.20) |>
  generate(reps = 1000) |>
  calculate(stat = "prop")
```

```
## Setting `type = "draw"` in `generate()`.
```

```
null_dist |>
  visualise() +
  shade_p_value(obs_stat = p_stat, direction = "greater")
```



显示 p 值：

```
null_dist |>
  get_p_value(obs_stat = p_stat, direction = "greater")
```

```
## # A tibble: 1 x 1
##   p_value
##     <dbl>
## 1   0.007
```

在 0.05 水平下拒绝零假设，认为女性学员比例显著增加了。

用 bootstrap 和 bootstrap 样本的分位数计算双侧置信区间：

```
FemGolf |>
  specify(response = gender, success = "F") |>
  generate(reps = 1000, type="bootstrap") |>
  calculate(stat = "prop") |>
  get_confidence_interval(level = 0.95,
    type = "percentile")
```

```
## # A tibble: 1 x 2
##   lower_ci upper_ci
##      <dbl>    <dbl>
## 1    0.210    0.290
```

用零假设下样本和标准误差计算双侧置信区间：

```
FemGolf |>
  specify(response = gender, success = "F") |>
  hypothesize(null = "point", p = 0.20) |>
  generate(reps = 1000, type="draw") |>
  calculate(stat = "prop") |>
  get_confidence_interval(level = 0.95,
    type = "se",
    point_estimate = p_stat)
```

```
## # A tibble: 1 x 2
##   lower_ci upper_ci
##      <dbl>    <dbl>
## 1    0.212    0.288
```

30.12.5 独立两样本比例的比较

如果只有汇总数据，为了使用随机模拟方法（再抽样方法），还需要制作原始观测数据集。

考虑前面的报税代理分理处的错误率比较问题。制作样本数据集：

```
da_tax <- tibble(
  shop = factor(rep(c(1,2), c(250, 300))),
  error = factor(rep(c("yes", "no", "yes", "no"),
    c(35, 250-35, 27, 300-27)),
    levels = c("yes", "no")) )
```

计算错误率差的统计量：

```
obs_stat <- da_tax |>
  specify(error ~ shop, success = "yes") |>
  calculate(stat = "diff in props",
    order = c(1, 2))
obs_stat
```

```
## Response: error (factor)
## Explanatory: shop (factor)
## # A tibble: 1 x 1
##    stat
##   <dbl>
## 1  0.05
```

上面的 specify 用了 y ~ g 的格式给出了分组变量 g 和二值变量 y，并用 success 指定了二值变量 y 作为两点分布观测值时，哪一个值当作成功。

用随机重排方法产生多个模拟样本，对每个样本计算比例差统计量，作为零假设下的统计量样本。所谓随机重排，是指保持样本量不变，两个分理处的观测数 n_1 和 n_2 不变，但将所有 $n = n_1 + n_2$ 个观测随机地重新标 n_1 个为分理处 1，n_2 个为分理处 2。这样的样本就代表两个分理处是不区分的，即代表了零假设成立情况下的样本。

```
null_dist <- da_tax |>
  specify(error ~ shop, success = "yes") |>
  hypothesize(null = "independence") |>
  generate(reps = 1000) |>
  calculate(stat = "diff in props",
    order = c(1, 2))
```

```
## Setting `type = "permute"` in `generate()`.
```

上面的程序的 hypothesize 中 null = "independence" 表示要检验的零假设是两组比较问题，对比例比较问题，零假设就是独立两组的比例相等。

双侧检验：

```
null_dist |>
  get_p_value(obs_stat = obs_stat,
    direction = "two-sided")
```

```
## # A tibble: 1 x 1
##   p_value
##     <dbl>
## 1   0.064
```

这种检验方法称为 “置换检验”(permutation test)。

利用比例差的统计量值，以及从零假设下置换检验方法生成的统计量分布计算到标准误差，计算比例差的置信区间：

```
null_dist |>
  get_confidence_interval(
    level = 0.95,
    type = "se",
    point_estimate = obs_stat)
```

```
## # A tibble: 1 x 2
##   lower_ci upper_ci
##      <dbl>    <dbl>
## 1 -0.00310    0.103
```

为了计算置信区间，也可以使用 bootstrap 方法产生样本，用 bootstrap 样本的统计量抽样分布的分位数计算置信区间：

```
da_tax |>
  specify(error ~ shop, success = "yes") |>
  generate(reps = 1000, type = "bootstrap") |>
  calculate(stat = "diff in props",
    order = c(1, 2)) |>
  get_confidence_interval(
    level = 0.95,
    type = "percentile")
```

```
## # A tibble: 1 x 2
##   lower_ci upper_ci
##      <dbl>    <dbl>
## 1 -0.00209    0.109
```

或利用比例差统计量和从 bootstrap 样本中估计的标准误差计算置信区间：

```
da_tax |>
  specify(error ~ shop, success = "yes") |>
  generate(reps = 1000, type = "bootstrap") |>
  calculate(stat = "diff in props",
    order = c(1, 2)) |>
  get_confidence_interval(
    level = 0.95,
    type = "se",
    point_estimate = obs_stat)
```

```
## # A tibble: 1 x 2
##   lower_ci upper_ci
##      <dbl>    <dbl>
## 1 -0.00516    0.105
```

30.12.6 拟合优度检验

!!! 当前 infer 扩展包 (1.2.1.9001) 的拟合优度代码可能有误，卡方统计量计算错误!!

考虑前面的 5 个候选人，甲和乙支持率相等的零假设。通过汇总数据生成原始数据集：

```
da_poll <-
  tibble(support = rep(
    c("A", "B", "Other"),
    c(300, 200, 500)))
```

计算拟合优度卡方统计量：

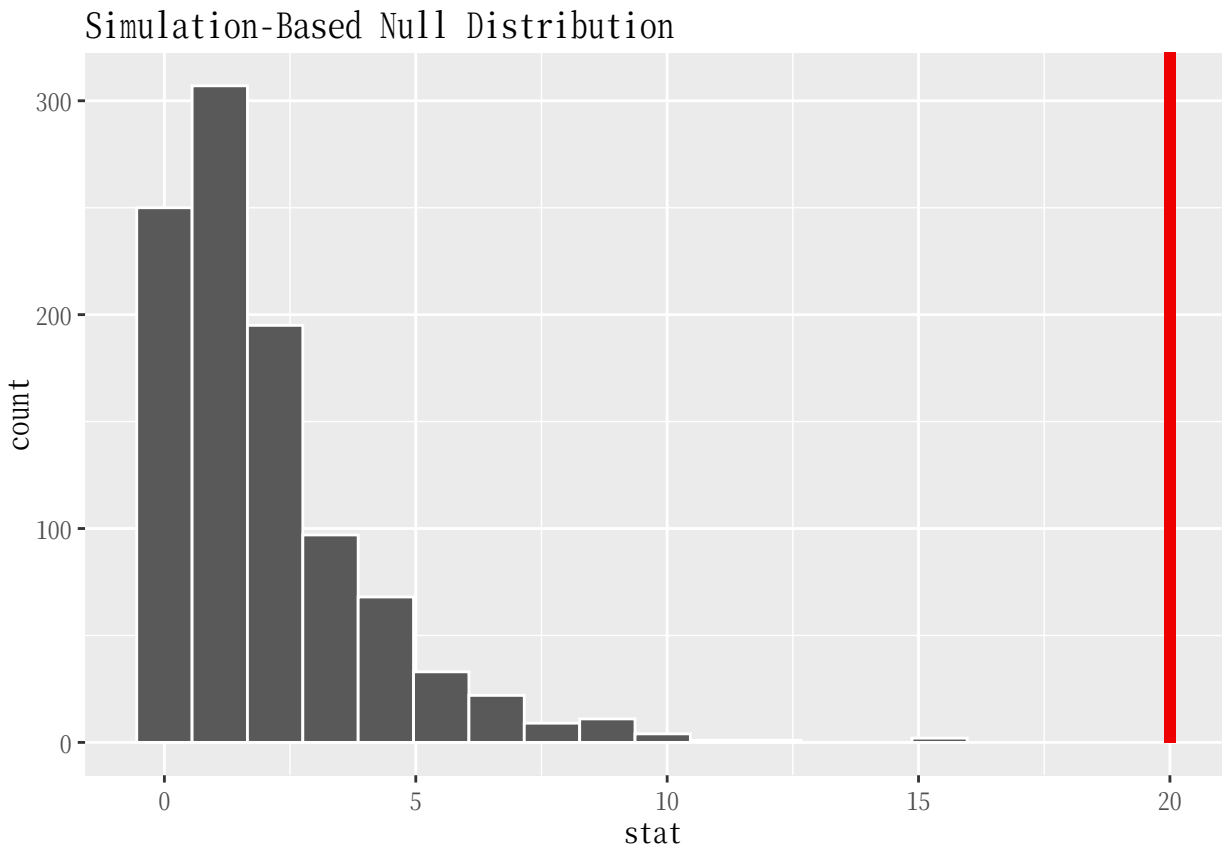
```
chisq_stat <- da_poll |>
  specify(response = support) |>
  hypothesize(null = "point", p = c(
    "A" = 0.25,
    "B" = 0.25,
    "Other" = 0.5  )) |>
  calculate(stat = "Chisq")
chisq_stat
```

```
## Response: support (factor)
## Null Hypothesis: point
## # A tibble: 1 x 1
##    stat
##   <dbl>
## 1    20
```

生成零假设下的样本：

```
null_dist <- da_poll |>
  specify(response = support) |>
  hypothesize(null = "point", p = c(
    "A" = 0.25,
    "B" = 0.25,
```

```
    "Other" = 0.5  )) |>
  generate(reps = 1000, type = "draw") |>
  calculate(stat = "Chisq")
null_dist |>
  visualize() +
  shade_p_value(
    obs_stat = chisq_stat,
    direction = "greater"  )
```



计算 p 值：

```
null_dist |>
  get_p_value(
    obs_stat = chisq_stat,
    direction = "greater"  )
```

```
## Warning: Please be cautious in reporting a p-value of 0. This result is an
## approximation based on the number of `reps` chosen in the `generate()` step.
## See `?get_p_value()` for more information.
```

```
## # A tibble: 1 x 1
##   p_value
##     <dbl>
## 1       0
```

30.12.7 列联表独立性检验

独立两样本比例比较可以看成是列联表独立性检验的特例。考虑前面的不同性别对啤酒种类偏好差异的例子。为了使用随机模拟方法，从表格数据生成具体的观测数据集：

```
d1 <- expand_grid(
  gender = factor(
    c("Male", "Female"), levels=c("Male", "Female")),
  beer = factor(
    c("Light", "Regular", "Dark"),
    levels=c("Light", "Regular", "Dark"))) |>
  mutate(n = c(t(ctab.beer)))
da_beer <- tibble(
  gender = rep(d1$gender, d1$n),
  beer = rep(d1$beer, d1$n))
da_beer |> count(gender, beer)
```

```
## # A tibble: 6 x 3
##   gender beer        n
##   <fct>  <fct>   <int>
## 1 Male   Light      20
## 2 Male   Regular    40
## 3 Male   Dark       20
## 4 Female Light      30
## 5 Female Regular    30
## 6 Female Dark       10
```

计算卡方统计量:

```
chisq_stat <- da_beer |>
  specify(beer ~ gender) |>
  calculate(stat = "Chisq")
chisq_stat
```

```
## Response: beer (factor)
```

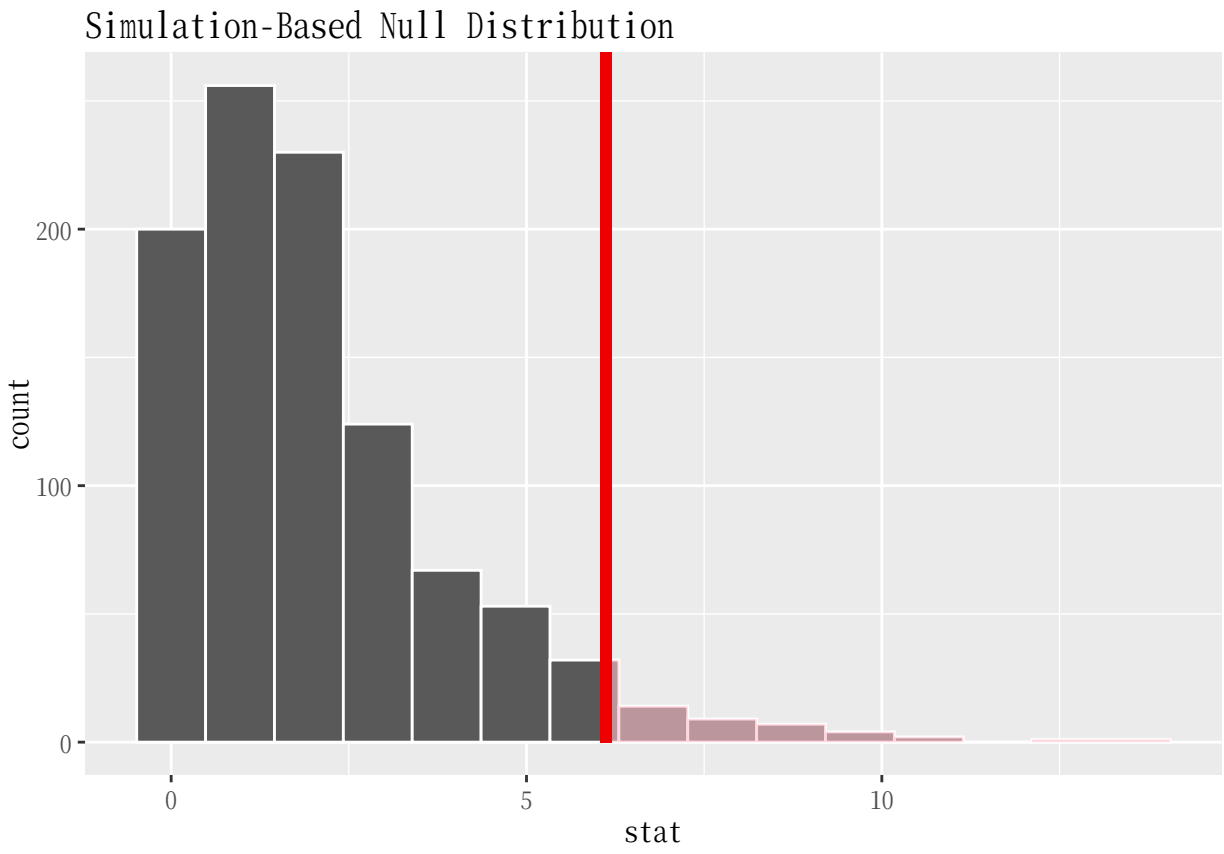
```
## Explanatory: gender (factor)
## # A tibble: 1 x 1
##    stat
##   <dbl>
## 1  6.12
```

用置换检验，将数据中的性别随机重排，生成多组重抽样样本，对每个样本计算卡方统计量，作为独立零假设下统计量分布的近似：

```
null_dist <- da_beer |>
  specify(beer ~ gender) |>
  hypothesize(null = "independence") |>
  generate(reps = 1000) |>
  calculate(stat = "Chisq")
```

```
## Setting `type = "permute"` in `generate()`.
```

```
null_dist |>
  visualize() +
  shade_p_value(
    obs_stat = chisq_stat,
    direction = "greater"
  )
```



计算 p 值：

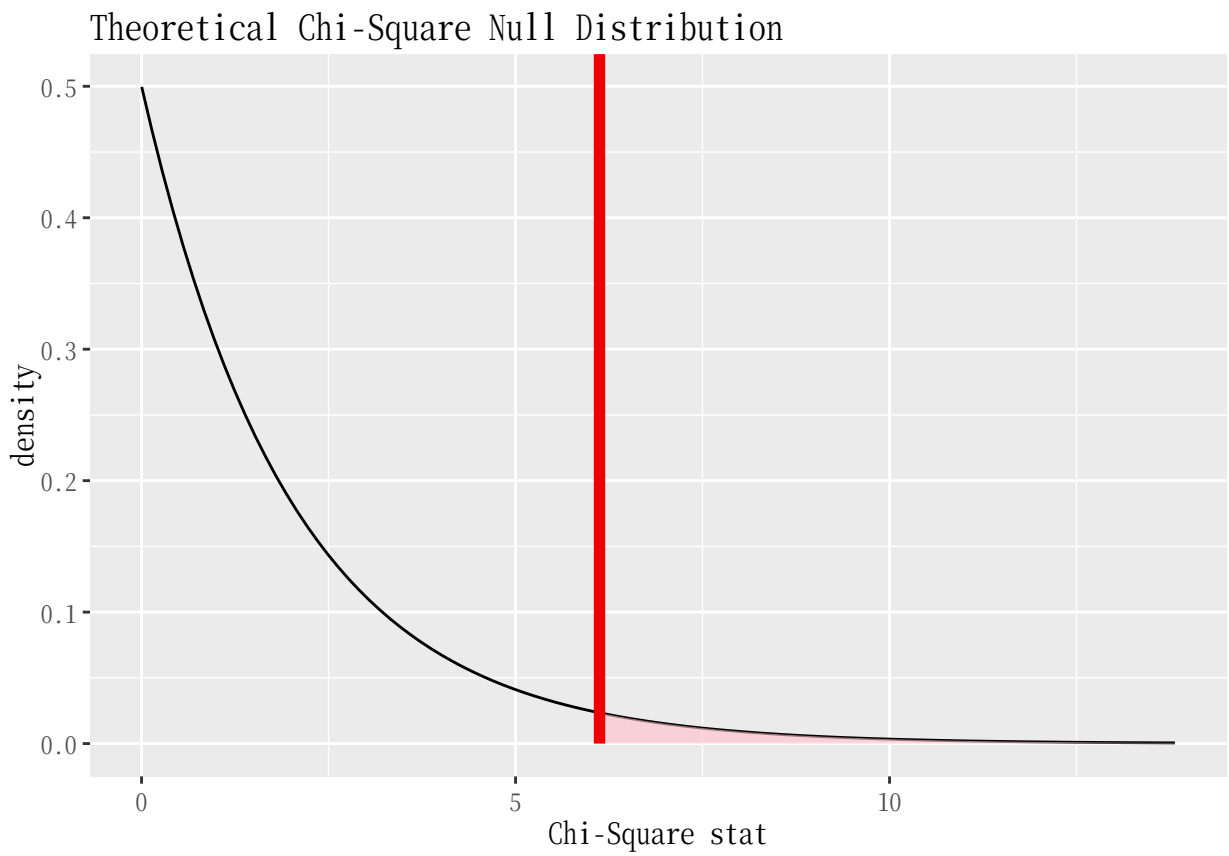
```
null_dist |>
  get_p_value(
    obs_stat = chisq_stat,
    direction = "greater"
  )
```

```
## # A tibble: 1 x 1
##   p_value
##     <dbl>
## 1   0.048
```

也可以使用理论卡方分布检验：

```
null_dist_th <- da_beer |>
  specify(beer ~ gender) |>
  assume(distribution = "Chisq")
null_dist_th |>
  visualize() +
```

```
  shade_p_value(
    obs_stat = chisq_stat,
    direction = "greater"
  )
```




```
null_dist_th |>
  get_p_value(
    obs_stat = chisq_stat,
    direction = "greater"
  )
```

```
## # A tibble: 1 x 1
##   p_value
##     <dbl>
## 1  0.0468
```

或:

```
da_beer |>
  chisq_test(beer ~ gender)
```

```
## # A tibble: 1 x 3
##   statistic chisq_df p_value
##       <dbl>    <int>   <dbl>
## 1      6.12        2  0.0468
```

Chapter 31

R 相关与回归

本章所用例子数据下载：

- data/reg-data.RData

31.1 相关分析

考虑连续型随机变量之间的关系。相关系数定义为

$$\rho(X,Y)=\frac{E\left[(X-EX)(Y-EY)\right]}{\sqrt{\mathrm{Var}(X)\mathrm{Var}(Y)}}$$

又称 Pearson 相关系数。

$-1\le\rho\le 1$。ρ 接近于 $+1$ 表示 X 和 Y 有正向的相关；ρ 接近于 -1 表示 X 和 Y 有负向的相关。

相关系数代表的是**线性**相关性，对于 X 和 Y 的其它相关可能反映不出来，比如 $X\sim\mathrm{N}(0,1)$，$Y=X^2$，有 $\rho(X,Y)=0$。

给定样本 $(X_i,Y_i),i=1,2,\ldots,n$，样本相关系数为

$$r=\frac{\sum_{i=1}^n(X_i-\bar{X})(Y_i-\bar{Y})}{\sqrt{\sum_{i=1}^n(X_i-\bar{X})^2\sum_{i=1}^n(Y_i-\bar{Y})^2}}$$

用散点图和散点图矩阵直观地查看变量间的相关。

例如，线性相关的模拟数据的散点图:

```
set.seed(1)
nsamp <- 30
x <- runif(nsamp, -10, 10)
y <- 20 + 0.5*x + rnorm(nsamp,0,0.5)
plot(x, y)
```

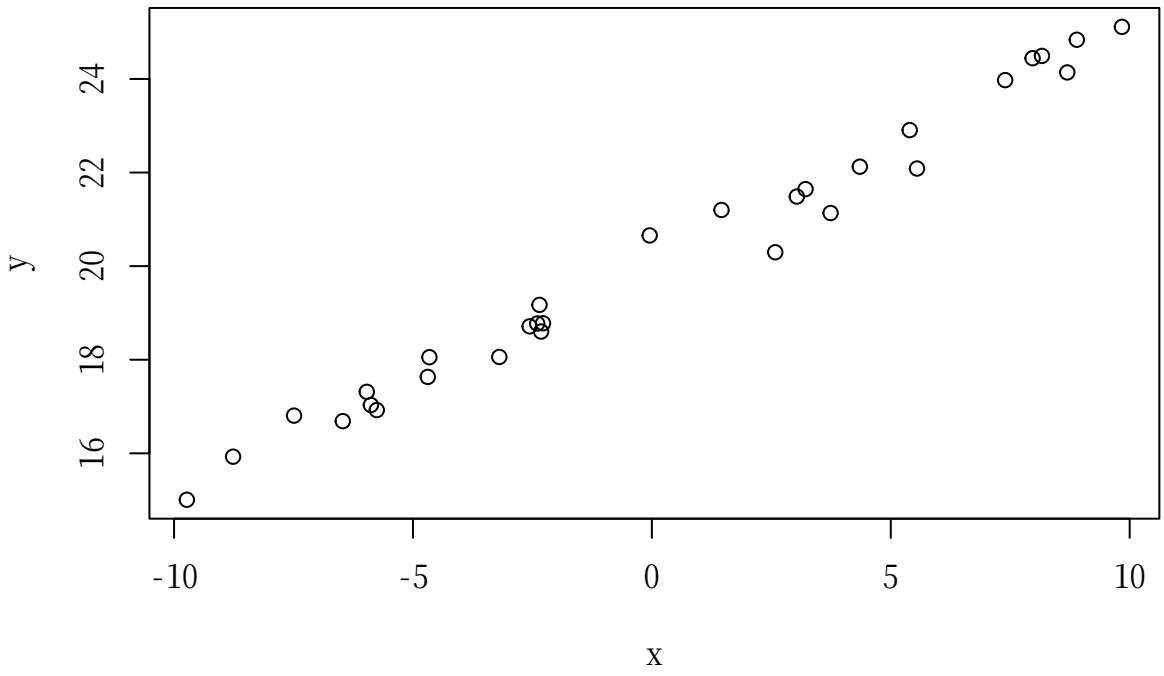


图 31.1: 线性相关数据

二次曲线相关的模拟数据散点图:

```
set.seed(1)
y2 <- 0.5*x^2 + rnorm(nsamp,0,2)
plot(x, y2)
```

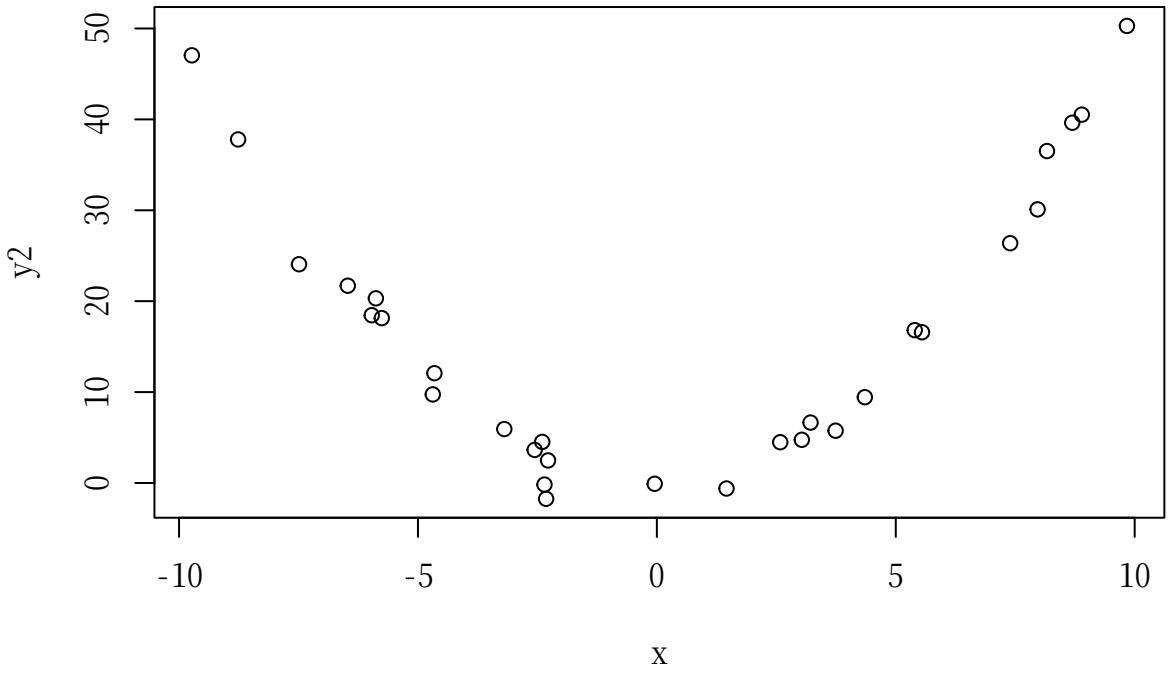


图 31.2: 二次曲线相关数据

指数关系的例子:

```
set.seed(1)
y3 <- exp(0.2*(x+10)) + rnorm(nsamp,0,2)
plot(x, y3)
```

对数关系的例子:

```
set.seed(1)
y4 <- log(10*(x+12)) + rnorm(nsamp,0,0.1)
plot(x, y4)
```

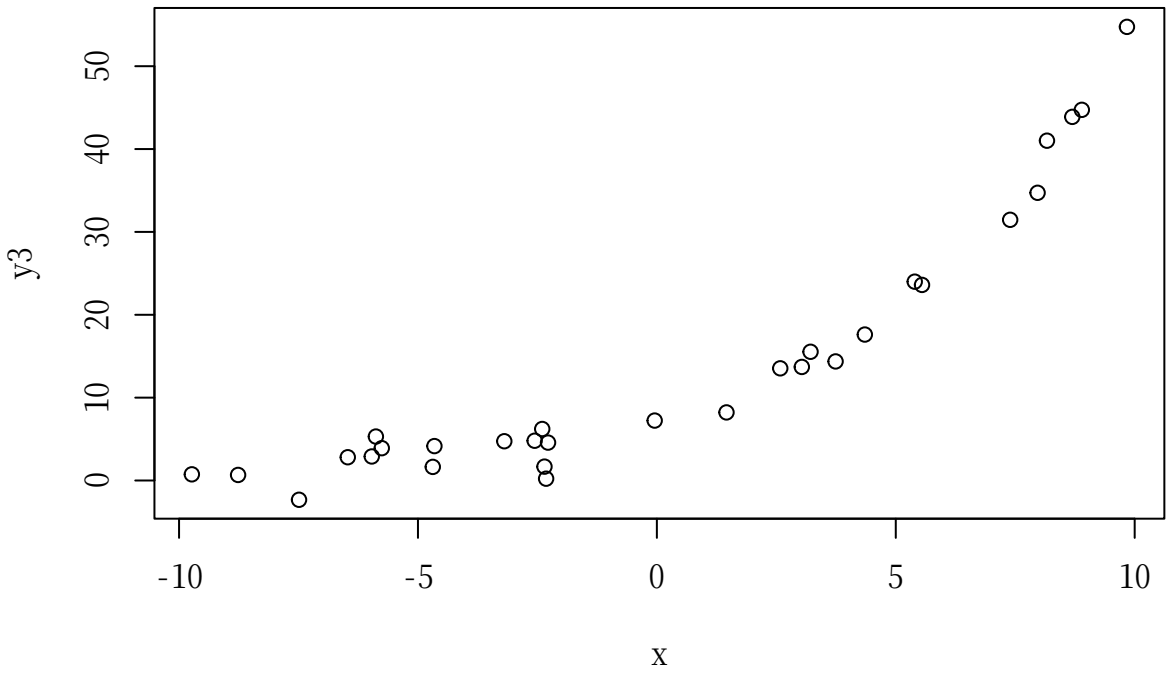


图 31.3: 指数关系相关数据

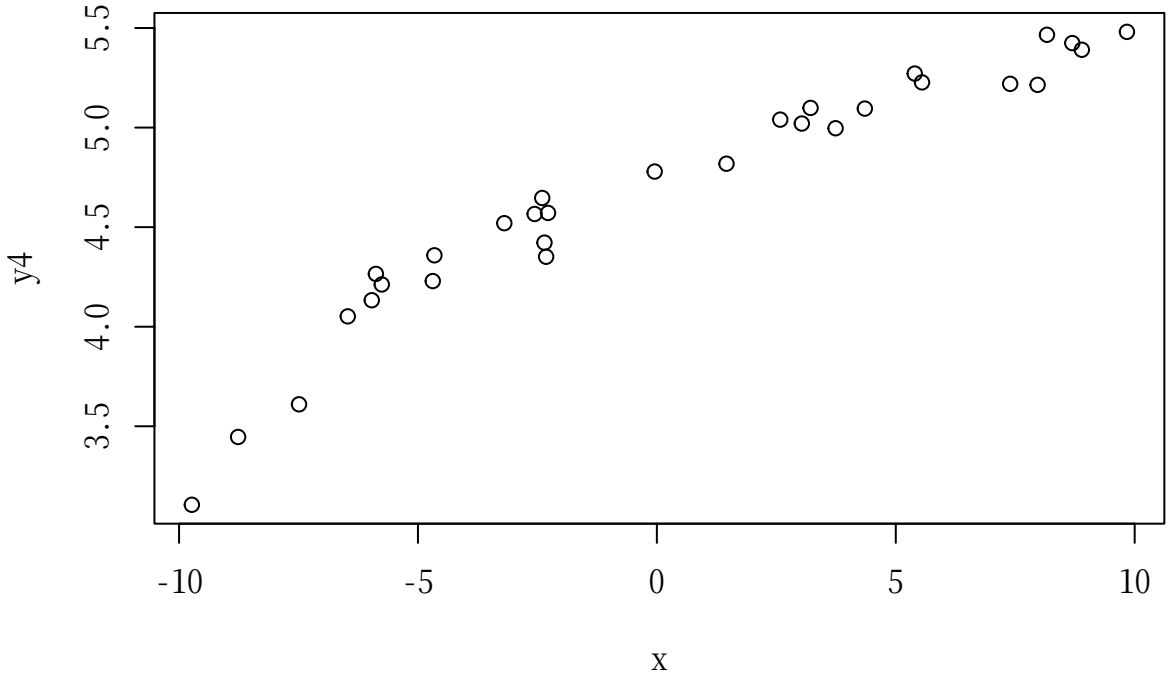


图 31.4: 对数关系相关数据

31.1.1 相关系数的性质

- 取值 $-1 \le r \le 1$。
- $r > 0$ 表示正线性相关；$r < 0$ 表示负线性相关。
- 当 $r = 1$ 时 $(x_i, y_i), i = 1, \dots, n$ 这 n 个点完全在一条正斜率的直线上，属于确定性关系。
- 当 $r = -1$ 时 $(x_i, y_i), i = 1, \dots, n$ 这 n 个点完全在一条负斜率的直线上，属于确定性关系。
- 计算相关系数时与 x, y 的次序或记号无关。
- 如果对变量 x 或者 y 分别乘以倍数，再分别加上不同的平移量，相关系数不变。即：$a + bx$ 与 $c + dy$ 的相关系数，等于 x, y 的相关系数。这样，相关系数不受单位、量纲的影响。相关系数是无量纲的。
- 如果 x 和 y 之间是非线性相关，相关系数不一定能准确反映其关系，只能作为一定程度的近似。

设变量 X 和 Y 的样本存放于 R 向量 `x` 和 `y` 中，用 `cor(x,y)` 计算样本相关系数。

```
set.seed(1)
x <- runif(30, 0, 10)
xx <- seq(0, 10, length.out = 100)
y <- 40 - (x-7)^2 + rnorm(30)
yy <- 40 - (xx-7)^2
plot(x, y, pch=16)
lines(xx, yy)
```

```
cor(x, y)
```

```
## [1] 0.8244374
```

```
x <- runif(30, 0, 10)
xx <- seq(0, 10, length.out = 100)
y <- 40 - (x-5)^2 + rnorm(30)
yy <- 40 - (xx-5)^2
plot(x, y, pch=16)
lines(xx, yy)
```

```
cor(x, y)
```

```
## [1] -0.042684
```

```
x <- runif(30, 0, 10)
xx <- seq(0, 10, length.out = 100)
y <- 40*exp(-x/2) + rnorm(30)
```

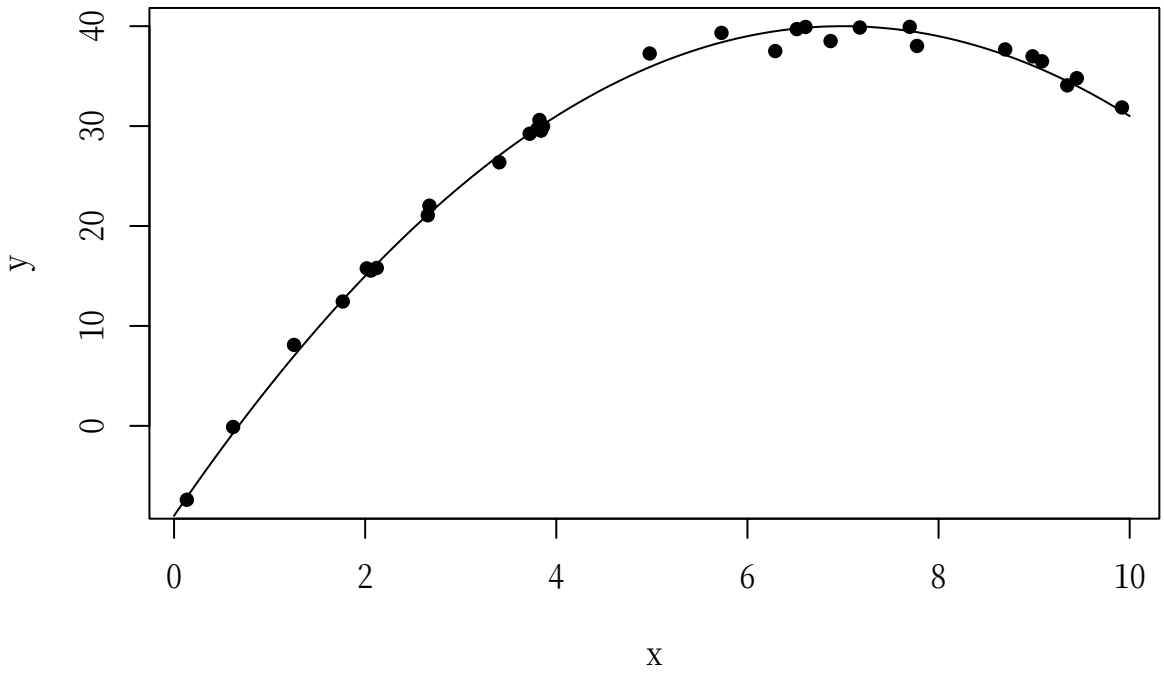


图 31.5: 曲线相关数据的相关系数例 1

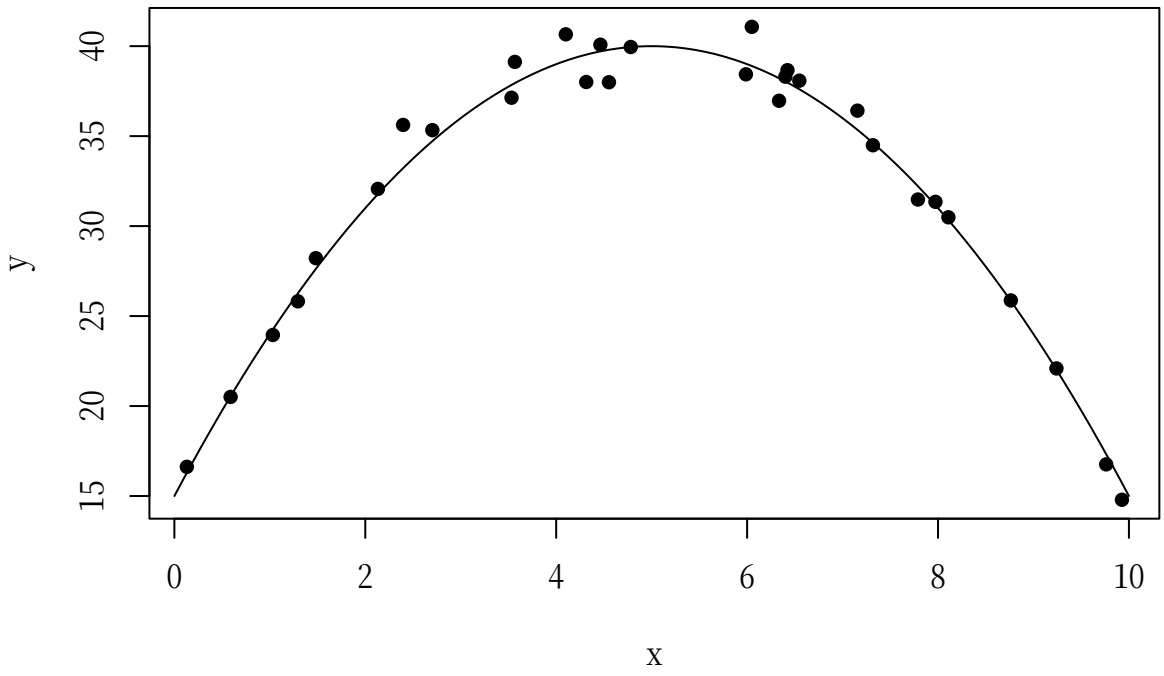


图 31.6: 曲线相关数据的相关系数例 2

```
yy <- 40*exp(-xx/2)
plot(x, y, pch=16)
lines(xx, yy)
```

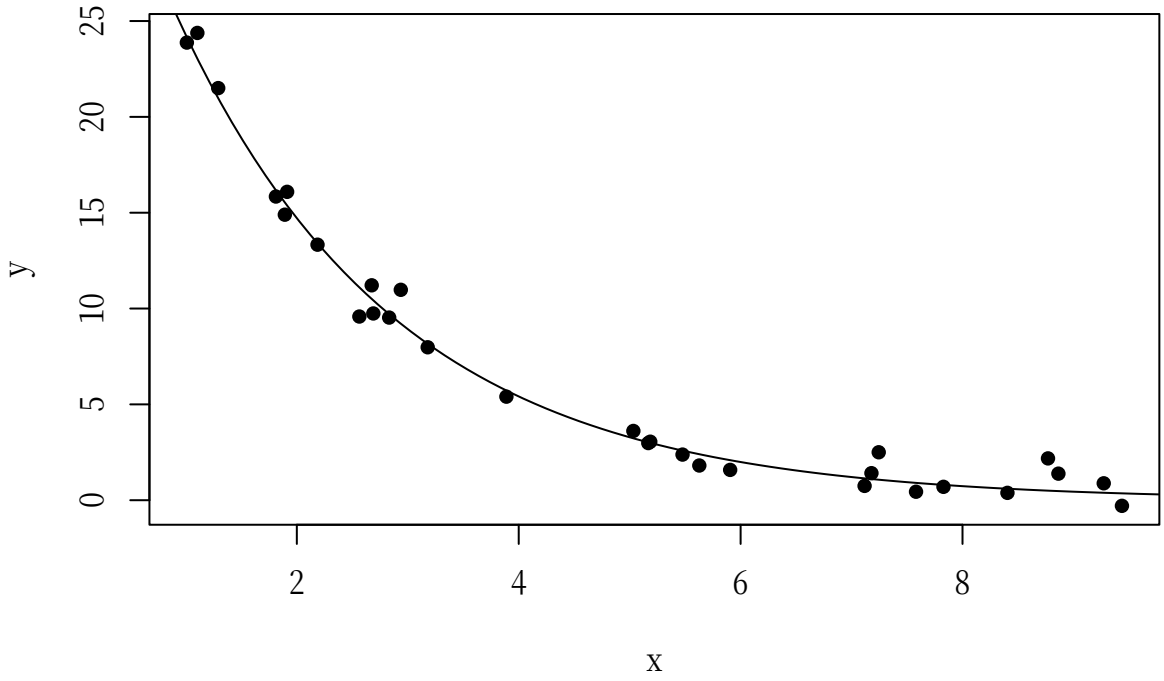


图 31.7: 曲线相关数据的相关系数例 3

```
cor(x, y)
```

```
## [1] -0.8841117
```

31.1.2 相关与因果

相关系数是从数据中得到的 x 和 y 两个变量关系的一种描述，不能直接引申为因果关系。

例如，增加身高，不一定增长体重。

实际中有许多错误理解相关与因果性的例子。例如：研究发现喝咖啡的人群比不喝咖啡的人群心脏病发病率高。于是断言喝咖啡导致心脏病危险增加。进一步研究更多的因素发现，喝咖啡放糖很多的人心脏病发病率才增高了。

31.1.3 相关系数大小

- 相关系数绝对值在 0.8 以上认为高度相关。
- 在 0.5 到 0.8 之间认为中度相关。
- 在 0.3 到 0.5 之间认为低度相关。
- 在 0.3 以下认为不相关或相关性很弱以至于没有实际价值。
- 当然，在特别重要的问题中，只要经过检验显著不等于零的相关都认为是有意义的。

31.1.4 相关系数的检验

相关系数 r 是总体相关系数 ρ 的估计。

$$H_0 : \rho = 0 \longleftrightarrow H_a : \rho \neq 0$$

检验统计量

$$t = \frac{r\sqrt{n-2}}{\sqrt{1-r^2}}$$

当总体 (X, Y) 服从二元正态分布且 $H_0 : \rho = 0$ 成立时，t 服从 $t(n-2)$ 分布。设 t 统计量值为 t_0，p 值为

$$P(|t(n-2)| > |t_0|)$$

在 R 中用 `cor.test(x,y)` 计算检验。

例如：

```
cor.test(d.class$height, d.class$weight)
```

```
##
##  Pearson's product-moment correlation
##
## data:  d.class$height and d.class$weight
## t = 7.5549, df = 17, p-value = 7.887e-07
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.7044314 0.9523101
## sample estimates:
##       cor
## 0.8777852
```

得身高与体重的相关系数为 0.88，检验的 p 值为 `7.887e-07`，95% 置信区间为 $[0.70, 0.95]$。

可以用 ggstatsplot 包将相关系数检验进行图示：

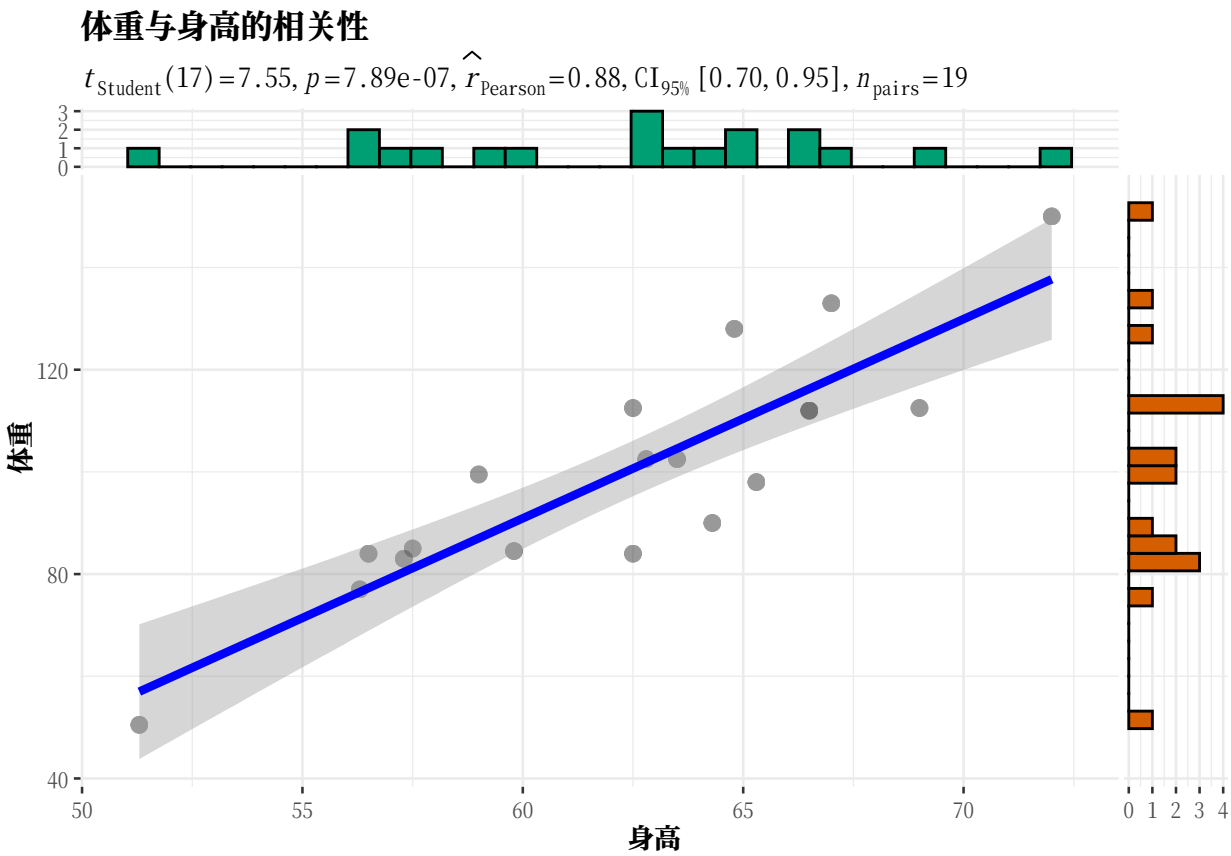
```
library(ggstatsplot)
```

```
## You can cite this package as:
##      Patil, I. (2021). Visualizations with statistical details: The 'ggstatsplot' approach.
##      Journal of Open Source Software, 6(61), 3167, doi:10.21105/joss.03167
```

```
ggscatterstats(
  data  = d.class,
  x     = height,
  y     = weight,
  xlab  = " 身高",
  ylab  = " 体重",
  title = " 体重与身高的相关性"
)
```

```
## Registered S3 method overwritten by 'ggside':
##   method from
##   +.gg   ggplot2
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



再例如:

```
set.seed(1)
nsamp <- 30
x <- runif(nsamp, -10, 10)
y <- 0.5*x^2 + rnorm(nsamp,0,2)
cor.test(x, y)
```

```
##
##  Pearson's product-moment correlation
##
## data:  x and y
## t = 0.93076, df = 28, p-value = 0.3599
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.1994815  0.5021658
## sample estimates:
##       cor
## 0.1732378
```

得 x 与 y 的样本相关系数为 0.17, p 值为 0.36，不显著。

31.1.4.1 相关系数检验的功效和样本量

pwr 包的 `pwr.r.test()` 可以计算相关系数检验的功效和样本量。用相关系数绝对值作为效应大小，常用值为：

- 小：0.1;
- 中: 0.3;
- 大: 0.5。

比如，样本量为 30，检验 “大” 的效应，计算其功效：

```
library(pwr)
```

```
## Warning: package 'pwr' was built under R version 4.2.2
```

```
cohen.ES(test = "r", size = "large")
```

```
##
##      Conventional effect size from Cohen (1982)
##
##            test = r
##            size = large
##     effect.size = 0.5
```

```
pwr.r.test(
  r = 0.5,
  n = 30,
  alternative = "two.sided"
)
```

```
##
##      approximate correlation power calculation (arctangh transformation)
##
##               n = 30
##               r = 0.5
##       sig.level = 0.05
##           power = 0.8250298
##     alternative = two.sided
```

功效为 83%。

检验 “小” 的效应，功效为 80%，需要的样本量：

```
cohen.ES(test = "r", size = "small")
```

```
##
##      Conventional effect size from Cohen (1982)
##
##            test = r
##            size = small
##     effect.size = 0.1
```

```
pwr.r.test(
  r = 0.1,
  power = 0.80,
  alternative = "two.sided"
)
```

```
##
##      approximate correlation power calculation (arctangh transformation)
##
##               n = 781.7516
##               r = 0.1
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
```

需要的样本量为 782。

31.1.5 相关阵

如果 x 是一个仅包含数值型列的数据框，则 cor(x) 计算样本相关系数矩阵；var(x) 计算样本协方差阵。

corrgram::corrgram() 可以绘制相关系数矩阵的图形，用颜色和阴影浓度代表相关系数正负和绝对值大小，用饼图中阴影部分大小代表相关系数绝对值大小。如：

```
library(corrgram)
corrgram(
  baseball[,c("Assists","Atbat","Errors","Hits","Homer","logSal",
           "Putouts","RBI","Runs","Walks","Years")],
```

```
  order=TRUE, main="Baseball data PC2/PC1 order",
  lower.panel=panel.shade, upper.panel=panel.pie)
```

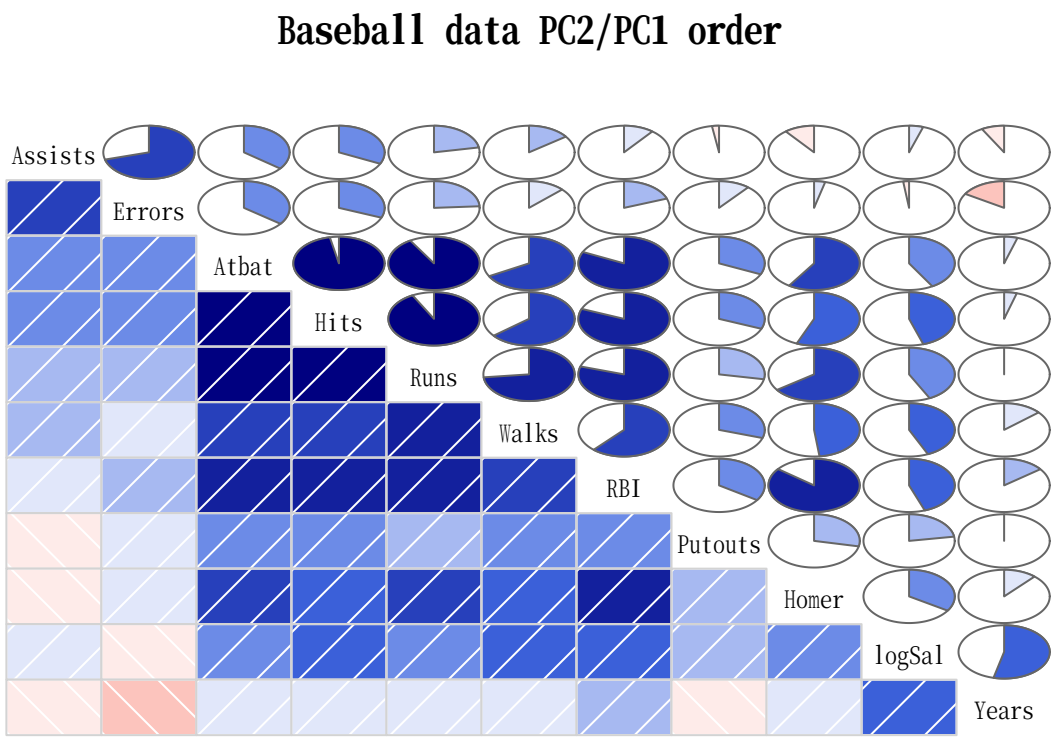


图 31.8: 相关阵图形

其中选项 `order=TRUE` 重排各列的次序使得较接近的列排列在相邻位置。

ggcorrplot 包也提供了类似的功能：

```
library(ggcorrplot)
baseball |>
  select(all_of(c(
  "Assists","Atbat","Errors","Hits","Homer","logSal",
  "Putouts","RBI","Runs","Walks","Years"))) |>
  cor(use="pairwise.complete.obs") |>
  ggcorrplot(hc.order=TRUE)
```

ggstatsplot 包也提供了这样的功能，并且进行检验：

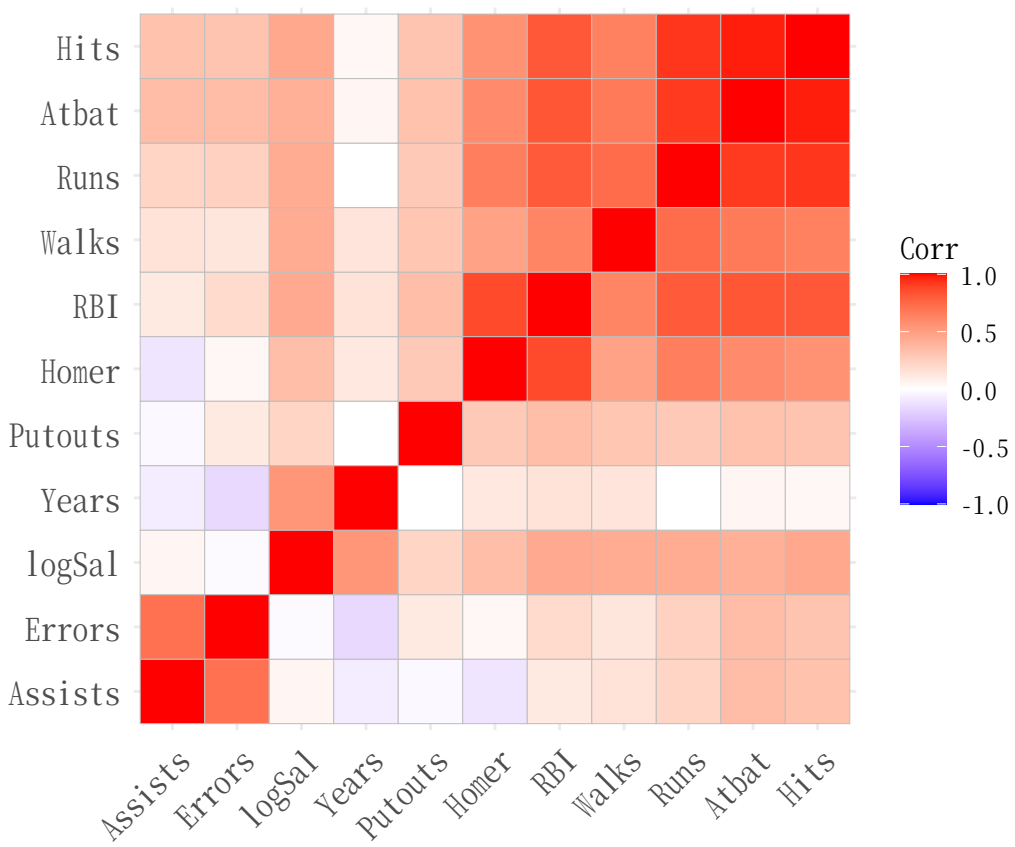


图 31.9: 相关阵图形（ggcorrplot）

```
library(ggstatsplot)
ggcorrmat(
  data     = baseball[,c("Assists","Atbat","Errors","Hits","Homer","logSal",
             "Putouts","RBI","Runs","Walks","Years")]
)
```

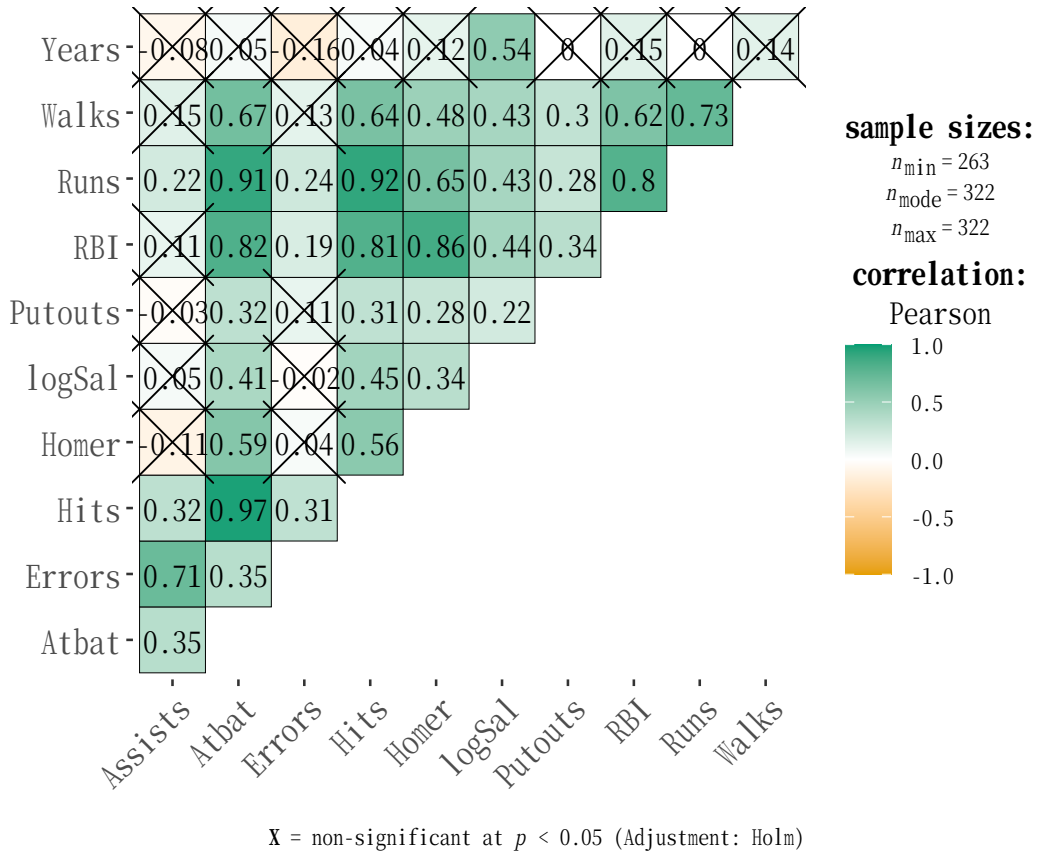


图 31.10: 相关阵图形（ggstatsplot）

31.1.6 Spearman 秩相关系数

Spearman 秩相关系数也成为 Spearman rho 系数，是两个变量的秩统计量的相关系数。计算如：

```
cor.test(d.class$height, d.class$weight,
  method="spearman")
```

```
## Warning in cor.test.default(d.class$height, d.class$weight, method =
## "spearman"): Cannot compute exact p-value with ties
```

```
##
```

```
##  Spearman's rank correlation rho
##
## data:  d.class$height and d.class$weight
## S = 164.43, p-value = 2.983e-06
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.8557611
```

秩相关系数值为 0.8558。

31.1.7 Kendall tau 系数

当变量 (X, Y) 正相关性很强时，任意两个观测的 X 值的大小顺序应该与 Y 值的大小顺序相同；如果独立，一对观测的 X 值比较和 Y 值比较顺序相同与顺序相反的数目应该基本相同。Kandall tau 系数也是取值于 $[-1, 1]$ 区间，用这样的思想表示两个变量的相关性和正负。

如：

```r
cor.test(d.class$height, d.class$weight,
  method="kendall")
```

```
## Warning in cor.test.default(d.class$height, d.class$weight, method = "kendall"):
## Cannot compute exact p-value with ties
```

```
##
##  Kendall's rank correlation tau
##
## data:  d.class$height and d.class$weight
## z = 4.2136, p-value = 2.513e-05
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##       tau
## 0.7142984
```

Kendall tau 系数的值为 0.7143。

31.2 一元回归分析

31.2.1 模型

考虑两个变量 Y 与 X 的关系，希望用 X 值的变化解释 Y 值的变化。X 称为自变量 (independent variable)，Y 称为因变量 (response variable)。

假设模型

$$Y = a + bX + \varepsilon, \quad \varepsilon \sim \mathrm{N}(0, \sigma^2)$$

在回归分析理论中通常假定 X 是非随机的，因变量 Y 的随机性来自误差项 ε。实际应用中 X 常常也是随机的。

设观测值为 $(X_i, Y_i), i = 1, 2, \ldots, n$，假设观测值满足上模型。即

$$Y_i = a + bX_i + \varepsilon_i, \quad \varepsilon_i \text{ iid } \sim \mathrm{N}(0, \sigma^2).$$

31.2.2 最小二乘法

直观上看，要找最优的直线 $y = a + bx$ 使得直线与观测到的点最接近。例如：

```
set.seed(1)
nsamp <- 30
x <- runif(nsamp, -10, 10)
y <- 20 + 0.5*x + rnorm(nsamp,0,0.5)
plot(x, y)
abline(lm(y ~ x), col="red", lwd=2)
```

a, b 的解用**最小二乘法**得到:

$$\min_{a,b} \sum_{i=1}^{n} (y_i - a - bx_i)^2$$

最小二乘解表达式为

$$\begin{aligned}
\hat{b} =& \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2} = r_{xy} \frac{S_y}{S_x} \\
\hat{a} =& \bar{y} - \hat{b}\bar{x}
\end{aligned}$$

其中 r_{xy} 为 X 与 Y 的样本相关系数，S_x 与 S_y 分别为 X 和 Y 的样本标准差。

随机误差方差 σ^2 的估计取为

$$\hat{\sigma}^2 = \frac{1}{n-2} \sum_i (y_i - \hat{a} - \hat{b}x_i)^2$$

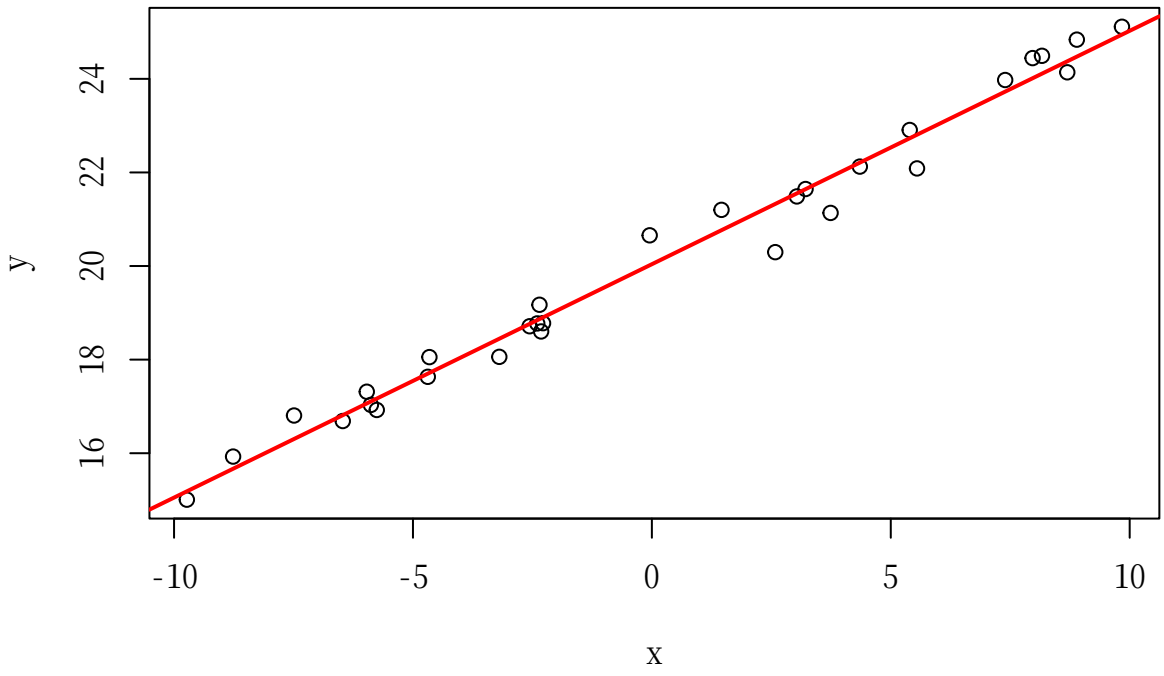


图 31.11: 一元线性回归最小二乘估计

标准误差: 为衡量 $\hat{a}$ 和 $\hat{b}$ 的估计精度，计算 $\mathrm{SE}(\hat{a})$ 和 $\mathrm{SE}(\hat{b})$。

估计结果:

- 系数估计 $\hat{a}, \hat{b}$ 以及相应的标准误差估计 $\mathrm{SE}(\hat{a})$ 和 $\mathrm{SE}(\hat{b})$;
- 拟合值 (预测值)
$$\hat{y}_i = \hat{a} + \hat{b}x_i, \quad i = 1, 2, \dots, n$$
- 残差
$$e_i = y_i - \hat{y}_i, \quad i = 1, 2, \dots, n$$
- $\mathrm{SSE} = \sum_i e_i^2$ 称为残差平方和，残差平方和小则拟合优。

31.2.3 回归有效性

31.2.3.1 拟合优度指标

按照最小二乘估计公式，只要 $x_1, x_2, \dots, x_n$ 不全相等，不论 x, y 之间有没有相关关系，都能计算出参数估计值。得到的回归直线与观测数据的散点之间的接近程度代表了回归结果的优劣。

残差平方和
$$\mathrm{SSE} = \sum_{i=1}^{n} (y_i - \hat{a} - \hat{b}x_i)^2$$
残差平方和越小，说明回归直线与观测数据点吻合得越好。

y 是因变量，因变量的数据的离差平方和
$$\mathrm{SST} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$
代表了因变量的未知变动大小，需要对这样的变动用自变量进行解释，称 SST 为总平方和。

可以证明如下平方和分解公式：
$$\mathrm{SST} = \mathrm{SSR} + \mathrm{SSE}$$
其中 SSR 称为回归平方和。

令 $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$ 称为第 i 个拟合值，回归平方和为
$$\mathrm{SSR} = \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2 = \hat{b}^2 \sum_{i=1}^{n} (x_i - \bar{x})^2$$

在总平方和中，回归平方和是能够用回归斜率与自变量的变动解释的部分，残差平方和是自变量不能解释的变动。分解中，回归平方和越大，误差平方和越小，拟合越好。令
$$R^2 = \frac{\mathrm{SSR}}{\mathrm{SST}} = 1 - \frac{\mathrm{SSE}}{\mathrm{SST}}$$
称为回归的复相关系数，或判定系数。

$0 \le R^2 \le 1$。判定系数越大，回归拟合越好。$R^2 = 1$ 时，观测点完全落在一条直线上面。一元回归中 R^2 是 x 和 y 的样本相关系数的平方。在多元回归时只能用复相关系数。

估计误差项方差 σ^2 为

$$\hat{\sigma}^2 = \frac{1}{n-2}\sum_{i=1}^{n}(y_i - \hat{a} - \hat{b}x_i)^2 = \frac{1}{n-2}\mathrm{SSE}$$

$\hat{\sigma}$ 是 σ 的估计，R 的回归结果中显示为 "residual standard error"（残差标准误差）。$\hat{\sigma}$ 是残差 $e_i = y_i - \hat{y}_i$ 的标准差的一个粗略估计。

31.2.3.2 线性关系显著性检验

当 $b = 0$ 时，模型退化为 $Y = a + \varepsilon$，X 不出现在模型中，说明 Y 与 X 不相关。检验

$$H_0 : b = 0 \longleftrightarrow H_\mathrm{a} : b \neq 0$$

取统计量

$$F = \frac{\mathrm{SSR}}{\mathrm{SSE}/(n-2)}$$

检验 p 值为 $P(F(1, n-2) > c)$(设 c 为 F 统计量的值)。取检验水平 α, p 值小于等于 α 时拒绝 H_0，认为 y 与 x 有显著的线性相关关系，否则认为 y 与 x 没有显著的线性相关关系。

也可以使用 t 统计量

$$t = \frac{\hat{b}}{\mathrm{SE}(\hat{b})}$$

其中

$$\mathrm{SE}(\hat{b}) = \hat{\sigma}/\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

设 t 统计量的值为 c，检验 p 值为 $P(|t(n-2)| > |c|)$。

31.2.4 R 程序

设数据保存在数据框 d 中，变量名为 y 和 x，用 R 的 `lm()` 函数计算回归，如:

```
set.seed(1)
nsamp <- 30
x <- runif(nsamp, -10, 10)
y <- 20 + 0.5*x + rnorm(nsamp,0,0.5)
d <- data.frame(x=x, y=y)
lm1 <- lm(y ~ x, data=d); lm1
```

```
##
## Call:
```

```
## lm(formula = y ~ x, data = d)
##
## Coefficients:
## (Intercept)            x
##     20.0388       0.4988
```

结果只有回归系数。需要用 summary() 显示较详细的结果。如

```
summary(lm1)
```

```
##
## Call:
## lm(formula = y ~ x, data = d)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.03030 -0.16436 -0.05741  0.29511  0.64046
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 20.03883    0.07226   277.3   <2e-16 ***
## x            0.49876    0.01244    40.1   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3956 on 28 degrees of freedom
## Multiple R-squared:  0.9829, Adjusted R-squared:  0.9823
## F-statistic:  1608 on 1 and 28 DF,  p-value: < 2.2e-16
```

结果中 $H_0 : b = 0$ 的检验结果 p 值为 $< 2e - 16$。这个结果可以从输出最后一行的 F 统计量检验结果看到，也可以从稀疏估计中自变量 x 所在行的 t 检验结果看到。

又如对 d.class 数据集，建立体重对身高的回归方程:

```
lm2 <- lm(weight ~ height, data=d.class)
summary(lm2)
```

```
##
## Call:
## lm(formula = weight ~ height, data = d.class)
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.6807  -6.0642   0.5115   9.2846  18.3698
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -143.0269    32.2746  -4.432 0.000366 ***
## height         3.8990     0.5161   7.555 7.89e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.23 on 17 degrees of freedom
## Multiple R-squared:  0.7705, Adjusted R-squared:  0.757
## F-statistic: 57.08 on 1 and 17 DF,  p-value: 7.887e-07
```

身高对体重的影响是显著的 (p 值 `7.89e-7`)。

31.2.5 回归诊断

除了系数的显著性检验以外，对误差项是否符合回归模型假定中的独立性、方差齐性、正态性等，可以利用回归残差进行一系列的回归诊断。还可以计算一些异常值、强影响点的诊断。

对 n 组观测值，回归模型为

$$y_i = a + bx_i + \varepsilon_i$$

拟合值为

$$\hat{y}_i = \hat{a} + \hat{b}x_i$$

残差（residual）为

$$e_i = y_i - \hat{y}_i$$

残差相当于模型中的随机误差，但仅在模型假定成立时才是随机误差的合理估计。所以残差能够反映违反模型假定的情况。

e_i 有量纲，不好比较大小。定义标准化残差（内部学生化残差）：

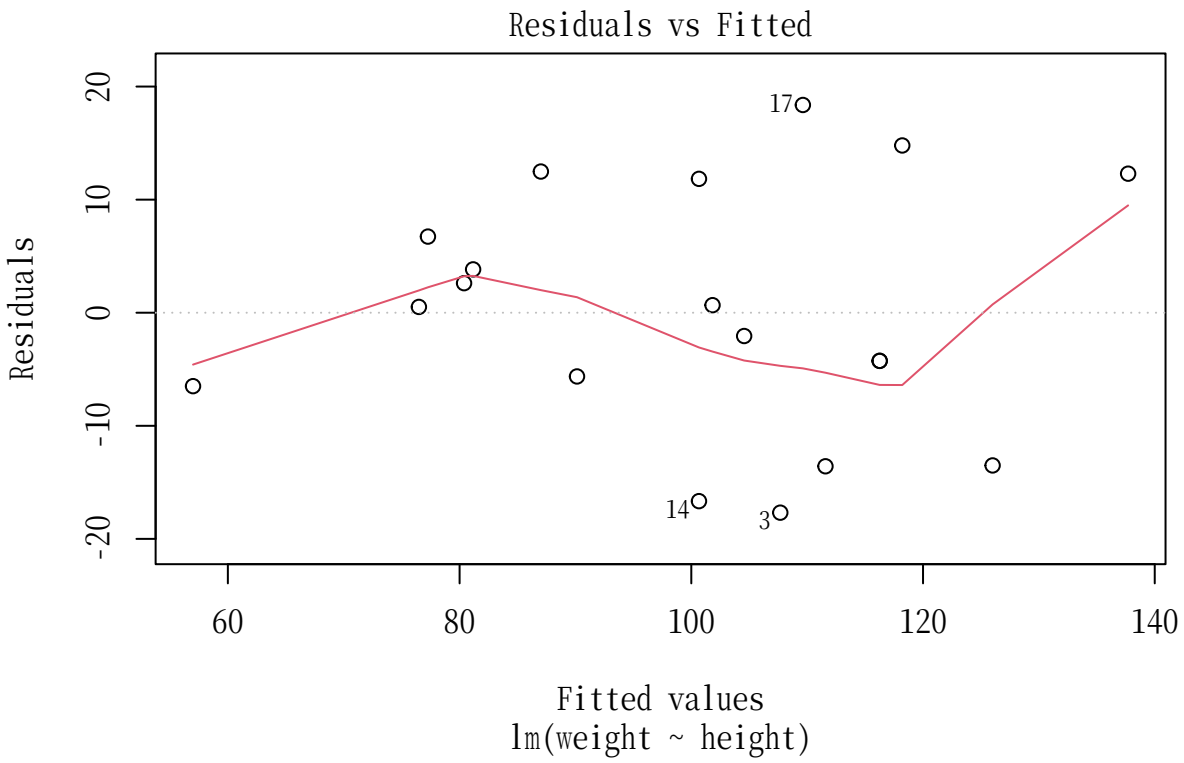
$$r_i = \frac{e_i}{S_e\sqrt{1 - \frac{1}{n} - \frac{(x_i - \bar{x})^2}{\sum_{k=1}^{n}(x_k - \bar{x})^2}}}$$

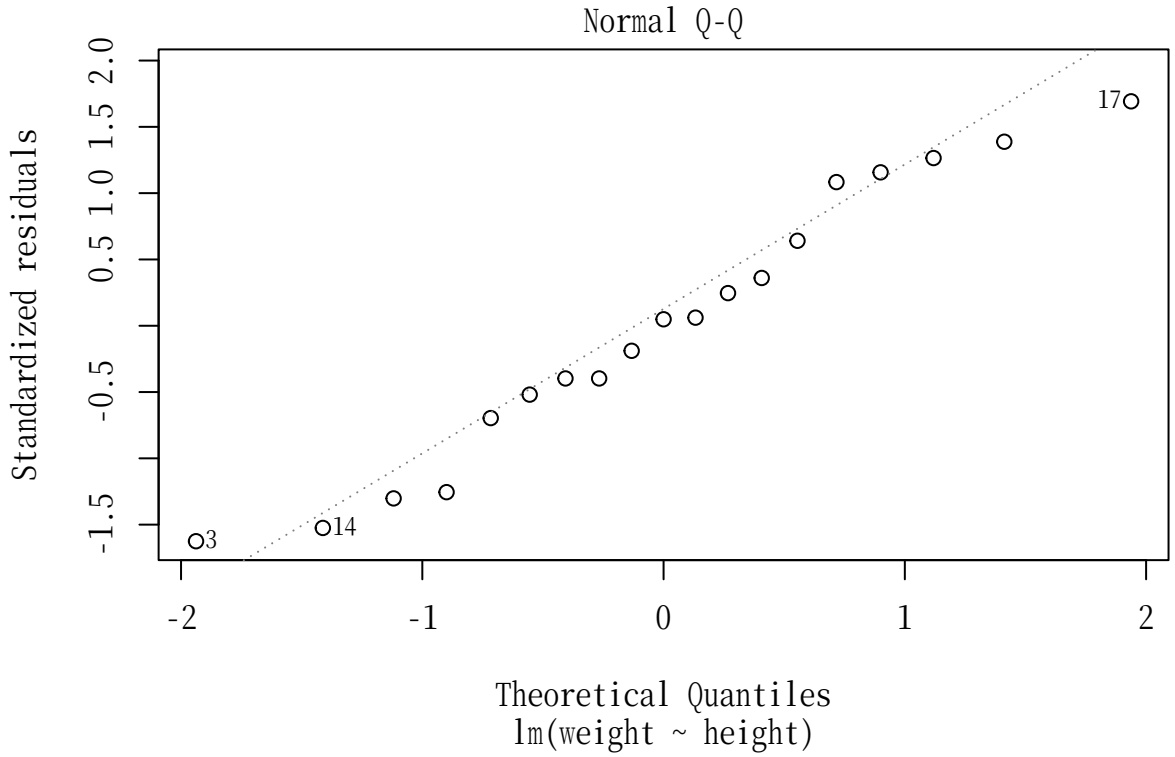
样本量较大时标准化残差近似服从标准正态分布，于是取值于 $[-2, 2]$ 范围外的点是可疑的离群点。

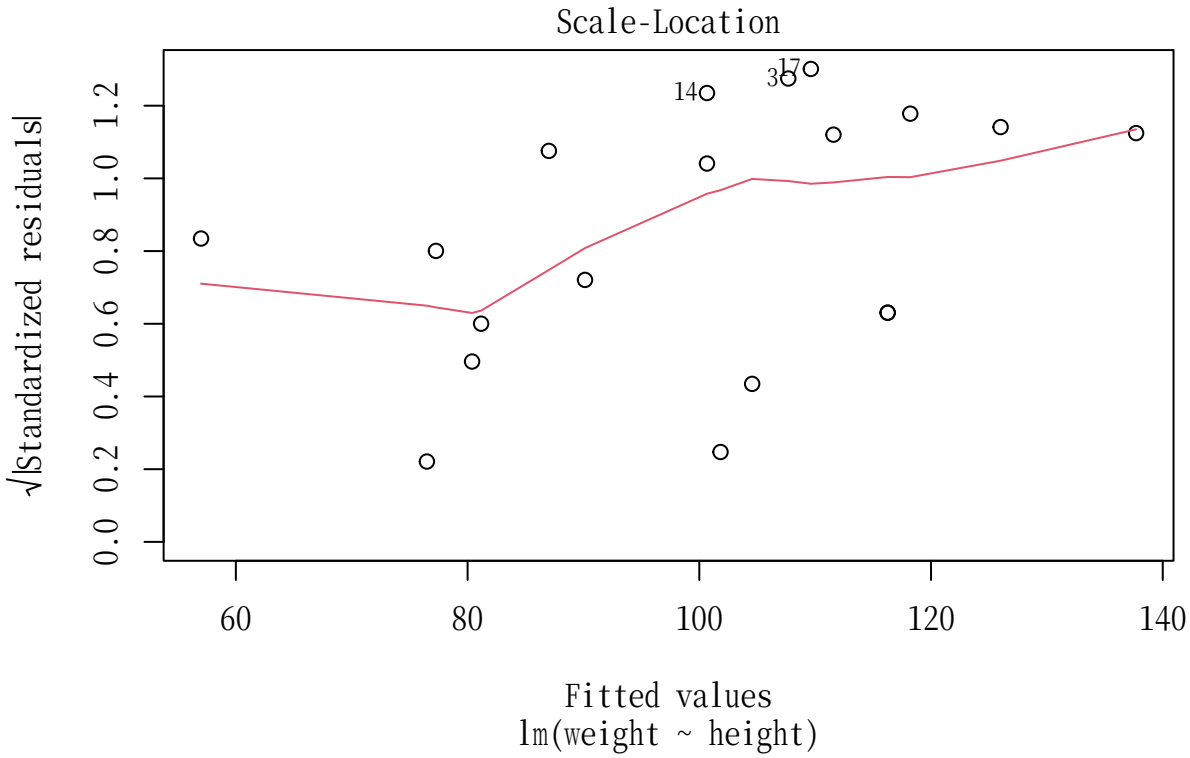
R 中用 `residuals()` 从回归结果计算残差，用 `rstandard()` 从回归结果计算标准化残差。

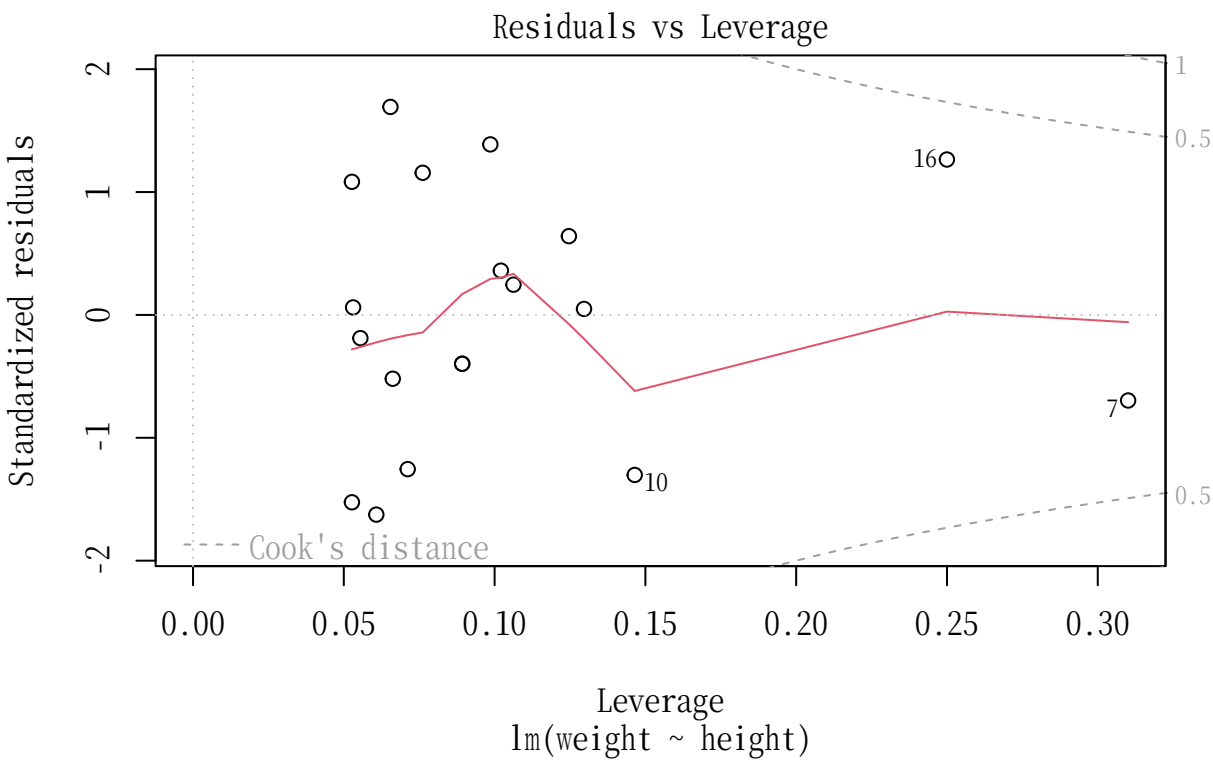
R 中 `plot(lmres)`(lmres 为回归结果变量) 可以做简单回归诊断。

```
plot(lm2)
```


Residuals vs Fitted
Residuals
20
10
0
-10
-20
17
14
3
60
80
100
120
140
Fitted values
lm(weight ~ height)


Normal Q-Q
Standardized residuals
-1.5
-0.5
0.5
1.0
1.5
2.0
3
14
17
-2
-1
0
1
2
Theoretical Quantiles
lm(weight ~ height)


Scale-Location
√|Standardized residuals|
0.0
0.2
0.4
0.6
0.8
1.0
1.2
14
3
17
60
80
100
120
140
Fitted values
lm(weight ~ height)



第一幅图是残差对预测值散点图，散点应该随机在 0 线上下波动，不应该有曲线模式、分散程度增大模式、特别突出的离群点等情况。

第二幅图是残差的正态 QQ 图，散点接近于直线时可以认为模型误差项的正态分布假定是合理的。

第三幅图是误差大小 (标准化残差绝对值的平方根) 对拟合值的图形，可以判断方差齐性假设 (方差 σ^2 不变)。

第四个幅是残差对杠杆量图，并叠加了 Cook 距离等值线。杠杆量代表了回归自变量对结果的影响大小，超过 $4/n$ 的值是需要重视的。Cook 距离考察删去第 i 个观测对回归结果的影响，如果有超出两条虚线范围之外的点，就可能是强影响点。

显示残差与标准化残差：

```
knitr::kable(cbind(" 残差"=residuals(lm2),
    " 标准化残差"=rstandard(lm2)),
    digits=2)
```

残差	标准化残差
6.73	0.64
-13.58	-1.26
-17.68	-1.63
0.51	0.05
-5.64	-0.52
-4.26	-0.40
-6.49	-0.70
11.84	1.08
0.67	0.06
-13.51	-1.30
-2.06	-0.19
14.79	1.39
2.61	0.25
-16.66	-1.52
12.48	1.16
12.30	1.26
18.37	1.69
3.83	0.36
-4.26	-0.40

可以用 e_i 或 r_i 作为纵坐标，x_i 或者 $\hat{y}_i$ 作为横坐标，作残差图。

下面给出残差图的几种常见的缺陷：

非线性:

```
set.seed(1)
x <- runif(30, 0, 10)
xx <- seq(0, 10, length.out = 100)
y <- 40 - (x-7)^2 + rnorm(30)
yy <- 40 - (xx-7)^2
lms1 <- lm(y ~ x)
opar <- par(mfrow=c(1,2))
plot(x, y, pch=16, main=" 数据和真实模型")
lines(xx, yy)
plot(x, rstandard(lms1), main=" 使用线性回归的残差")
```

```
par(opar)
```

异方差:

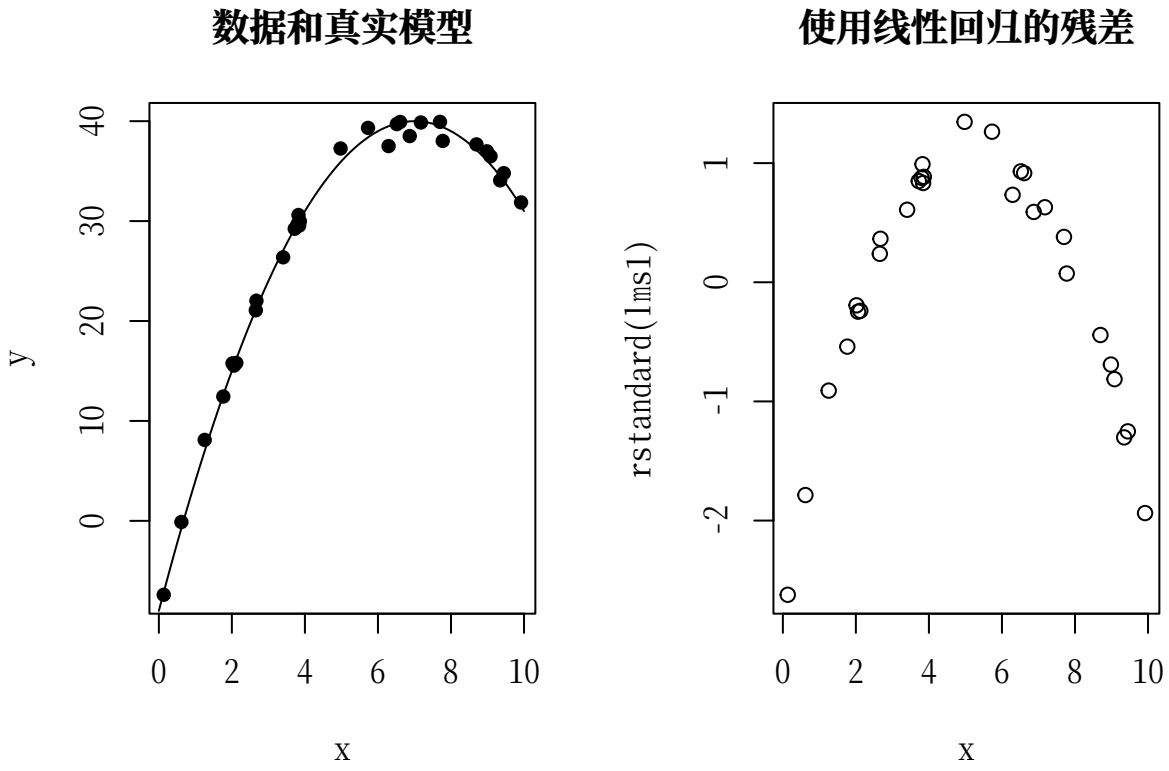
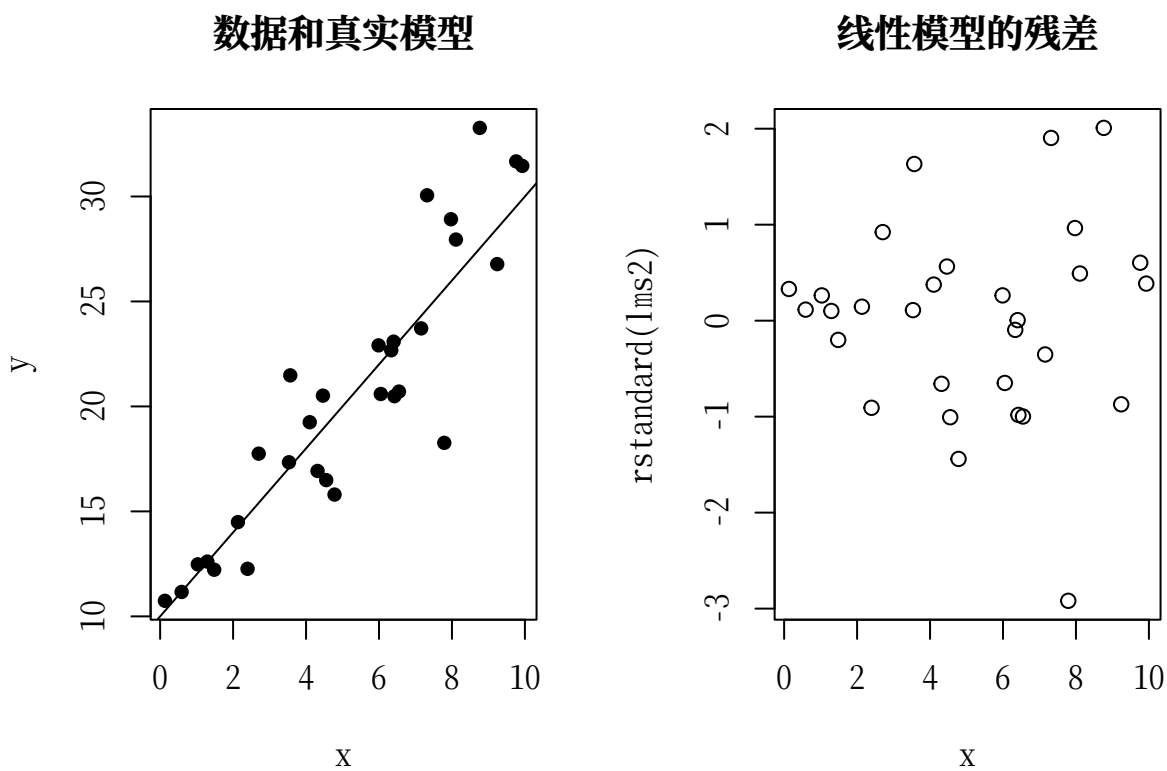


图 31.12: 残差中非线性模式

```
x <- sort(runif(30, 0, 10))
y <- 10 + 2*x + rnorm(30)*(seq(30)/5)
lms2 <- lm(y ~ x)
opar <- par(mfrow=c(1,2))
plot(x, y, pch=16, main=" 数据和真实模型")
abline(a=10, b=2)
plot(x, rstandard(lms2), main=" 线性模型的残差")
```

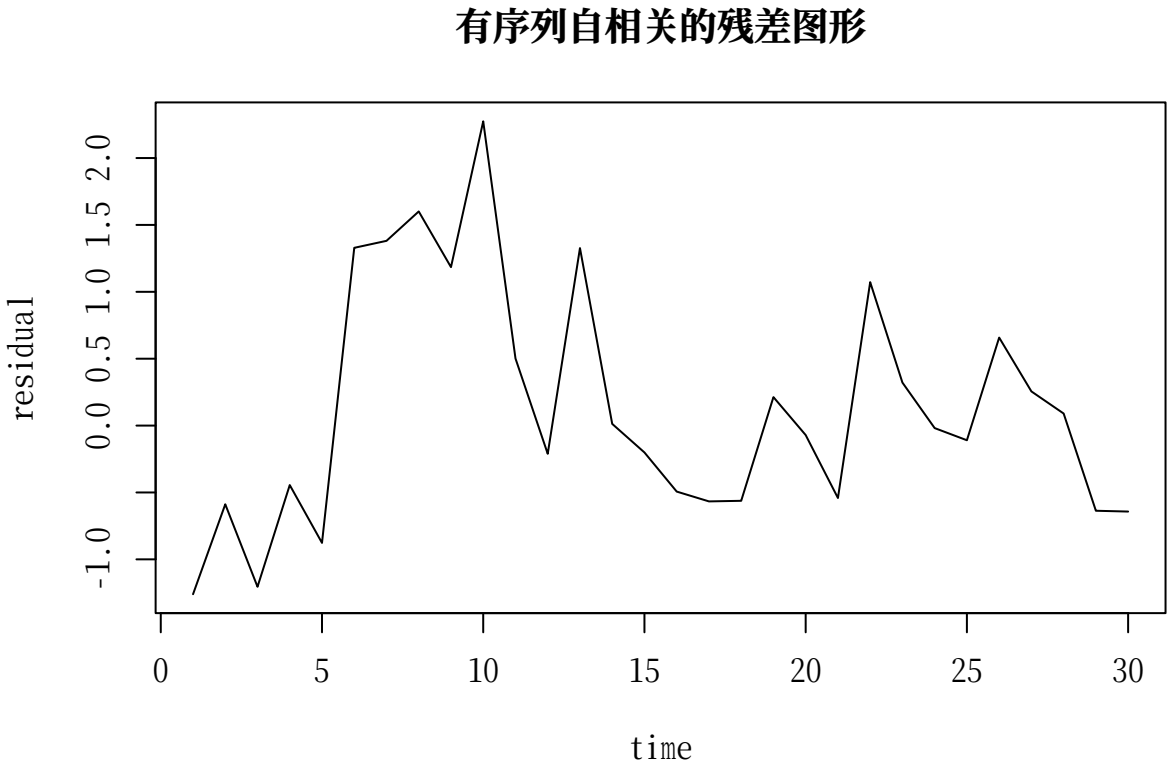



```
par(opar)
```

序列自相关：

当数据随时间变化时，还需要考虑前后是否有序列自相关。自相关残差图形例子：

```
ar1 <- arima.sim(list(ar=c(0.5)), n=30)
plot(1:30, ar1[], type="l", xlab="time", ylab="residual",
     main=" 有序列自相关的残差图形")
```



图中横坐标是观测序号。

`car::ncvTest()` 检验方差齐性。零假设是方差齐性成立。如

```
car::ncvTest(lm2)
```

```
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 1.664307, Df = 1, p = 0.19702
```

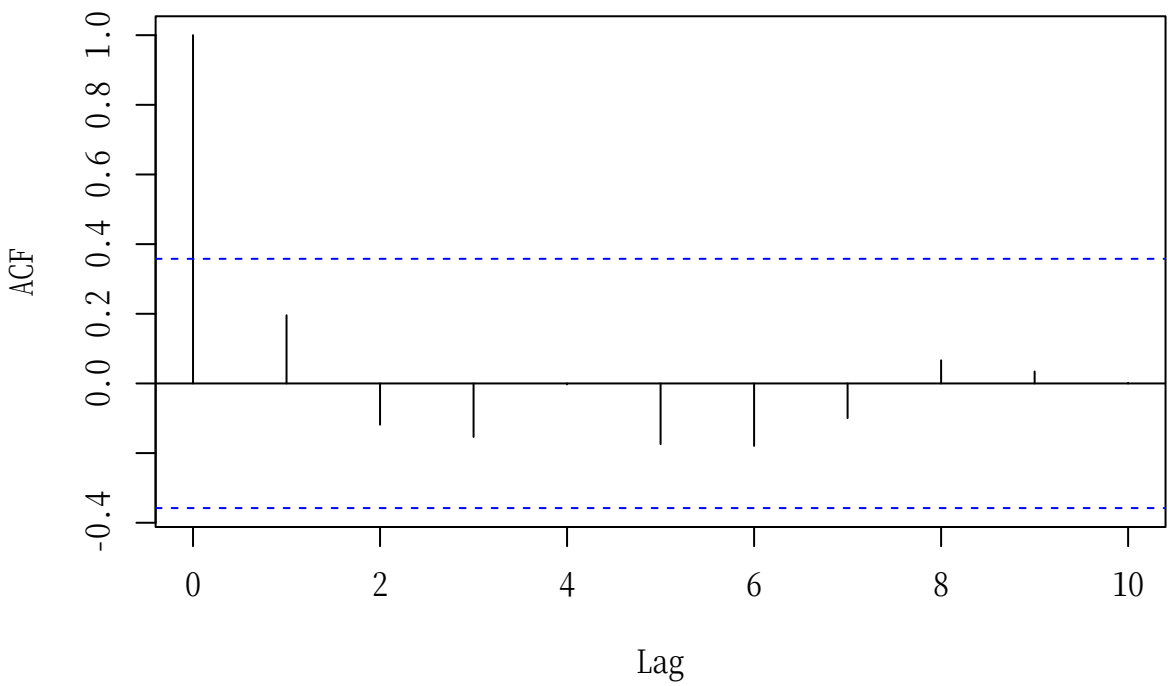
用 Durbin-Watson 检验（DW 检验）可以检验残差中是否有序列自相关，零假设是没有序列自相关。R 中用 `car::durbinWatsonTest()` 检验。序列自相关检验仅当数据中数据是沿时间等间隔记录时有意义。如

```
car::durbinWatsonTest(lm1)
```

```
##  lag Autocorrelation D-W Statistic p-value
##    1       0.1959139       1.57293    0.24
##  Alternative hypothesis: rho != 0
```

也可以对回归残差绘制 ACF 图，如果除了横坐标 0 之外都落在两条水平界限内则认为没有序列自相关，如果有明显超出界限的就认为有序列自相关。如

```
acf(residuals(lm1), lag.max = 10, main="")
```



31.2.6 预测区间

设 X 取 x_0，Y 的预测值为 $\hat{y}_0 = \hat{a} + \hat{b}x_0$。置信水平为 $1-\alpha$ 的预测区间为

$$\hat{y}_0 \pm \lambda\hat{\sigma}\sqrt{1 + \frac{1}{n} + \frac{(\bar{x} - x_0)^2}{\sum_i (x_i - \bar{x})^2}}$$

其含义是 $P(y_0 \in \text{预测区间}) = 1-\alpha$。$\lambda$ 是标准正态分布双侧 α 分位数 `qnorm(1 - alpha/2)`。

用 `predict(lmres)` 得到 $\hat{y}_i, i = 1, \ldots, n$ 的值，`lmres` 表示回归结果变量。

用 `predict(lmres, interval="prediction")` 同时得到预测的置信区间，需要的话加入 `level=` 选项设定置信度。

回归的置信区间有两种，上述的区间是 “预测区间”(CLI)，使得 $P(y_0 \in \text{预测区间}) = 1-\alpha$; 另一种区间称为均值的置信区间 (CLM)，使得 $P(Ey_0 \in \text{均值置信区间}) = 1-\alpha$。

31.2.7 控制

如果需要把 Y 的值控制在 $[y_l, y_u]$ 范围内，问如何控制 X 的范围，可以求解 x_0 的范围使上面的置信区间包含在 $[y_l, y_u]$ 内。

近似地可以解不等式

$$\begin{aligned}\hat{a} + \hat{b}x_0 - z_{1-\frac{\alpha}{2}}\hat{\sigma} \geq & y_l \\ \hat{a} + \hat{b}x_0 + z_{1-\frac{\alpha}{2}}\hat{\sigma} \leq & y_u\end{aligned}$$

其中 $z_{1-\frac{\alpha}{2}}$ 为标准正态分布双侧 α 分位数（用 `qnorm(1-alpha/2)` 计算）。

Chapter 32

R 多元回归

建模步骤：

- 确定要研究的因变量，以及可能对因变量有影响并且数据可以获得的自变量集合；
- 假定因变量 y 与 p 个自变量之间为线性相关关系，建立模型；
- 对模型进行评估和检验；
- 判断模型中是否存在多重共线性，如果存在，应进行处理；
- 预测；
- 残差分析。

32.1 模型

模型

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \varepsilon$$

其中

- y: 因变量，随机变量；
- $x_1, \ldots, x_p$: 自变量，非随机；
- ε: 随机误差项，随机变量；
- β_0: 截距项；
- β_j: 对应于 x_j 的斜率项；
- 模型表明因变量 y 近似等于自变量的线性组合；
- $E\varepsilon = 0$, $\mathrm{Var}(\varepsilon) = \sigma^2$。

对 n 组观测数据，有

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i, \quad i = 1, 2, \ldots, n$$

对其中的随机误差项 ε_i, $i = 1, 2, \ldots, n$，假定：

- 期望为零，服从正态分布；
- 方差齐性: 方差为 σ^2 与自变量值无关；
- 相互独立。

总之，$\varepsilon_1, \varepsilon_2, \dots, \varepsilon_n$ 相互独立，服从 $\mathrm{N}(0, \sigma^2)$ 分布。

数据格式如：

$$\left(\begin{array}{cccc|c} x_1 & x_2 & \cdots & x_p & y \\ \hline x_{11} & x_{12} & \cdots & x_{1p} & y_1 \\ x_{21} & x_{22} & \cdots & x_{2p} & y_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} & y_n \end{array}\right)$$

R 中回归数据放在一个数据框中，有一列 y 和 p 列自变量。

根据模型有

$$Ey = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p.$$

32.2 参数估计

未知的参数有回归系数 $\beta_0, \beta_1, \dots, \beta_p, \sigma^2$，另外误差项也是未知的。

模型估计仍使用最小二乘法，得到系数估计 $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p$ 及误差方差估计 $\hat{\sigma}^2$。把系数估计代入到模型中，写出如下的估计的多元线性回归方程：

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p.$$

对第 i 组观测值，将自变量值代入估计的回归方程中得拟合值

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}.$$

$e_i = y_i - \hat{y}_i$ 称为残差。

回归参数估计，用残差最小作为目标，令

$$Q = \sum_{i=1}^{n} \left[y_i - (\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) \right]^2$$

取使得 Q 达到最小的 $(\beta_0, \beta_1, \dots, \beta_p)$ 作为参数估计，称为**最小二乘估计**，记为 $(\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p)$。称得到的最小值为 SSE，$\sigma^2$ 的估计为

$$\hat{\sigma}^2 = \frac{1}{n-p-1}\mathrm{SSE} = \frac{1}{n-p-1}\sum_{i=1}^{n} e_i^2.$$

32.3 R 的多元回归程序

在 R 中用 `lm(y ~ x1 + x2 + x3, data=d)` 这样的程序来做多元回归，数据集为 d，自变量为 x1,x2,x3 三列。

例 32.1. 考虑 d.class 数据集中体重对身高和年龄的回归。

```
lm3 <- lm(weight ~ height + age, data=d.class)
summary(lm3)
```

```
##
## Call:
## lm(formula = weight ~ height + age, data = d.class)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -17.962  -6.010  -0.067   7.553  20.796
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -141.2238    33.3831  -4.230 0.000637 ***
## height         3.5970     0.9055   3.973 0.001093 **
## age            1.2784     3.1101   0.411 0.686492
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.51 on 16 degrees of freedom
## Multiple R-squared:  0.7729, Adjusted R-squared:  0.7445
## F-statistic: 27.23 on 2 and 16 DF,  p-value: 7.074e-06
```

得到的回归模型可以写成

$$\widehat{\text{weight}} = -141.2 + 3.597 \times \text{height} + 1.278 \times \text{age}$$

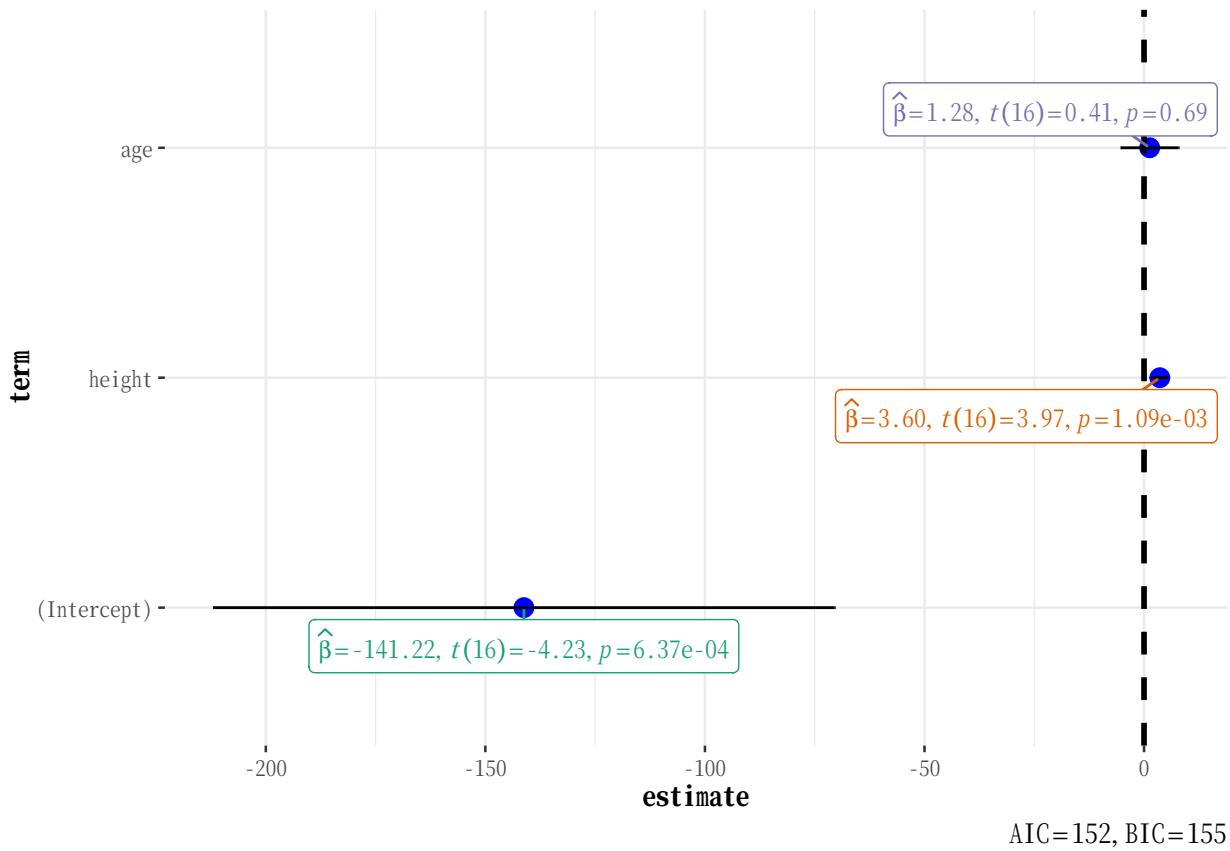
其中身高的系数 3.597 表示，如果两个学生年龄相同，则身高增加 1 个单位（这里是英寸），体重平均增加 3.597 个单位（这里是磅）。注意在仅有身高作为自变量时，系数为 3.899。年龄的系数 1.278 也类似解释，在两个学生身高相同时，如果一个学生年龄大 1 岁，则此学生的体重平均多 1.278 个单位。

回归系数可以用 ggstatsplot 包图示如下：

```
library(ggstatsplot)
```

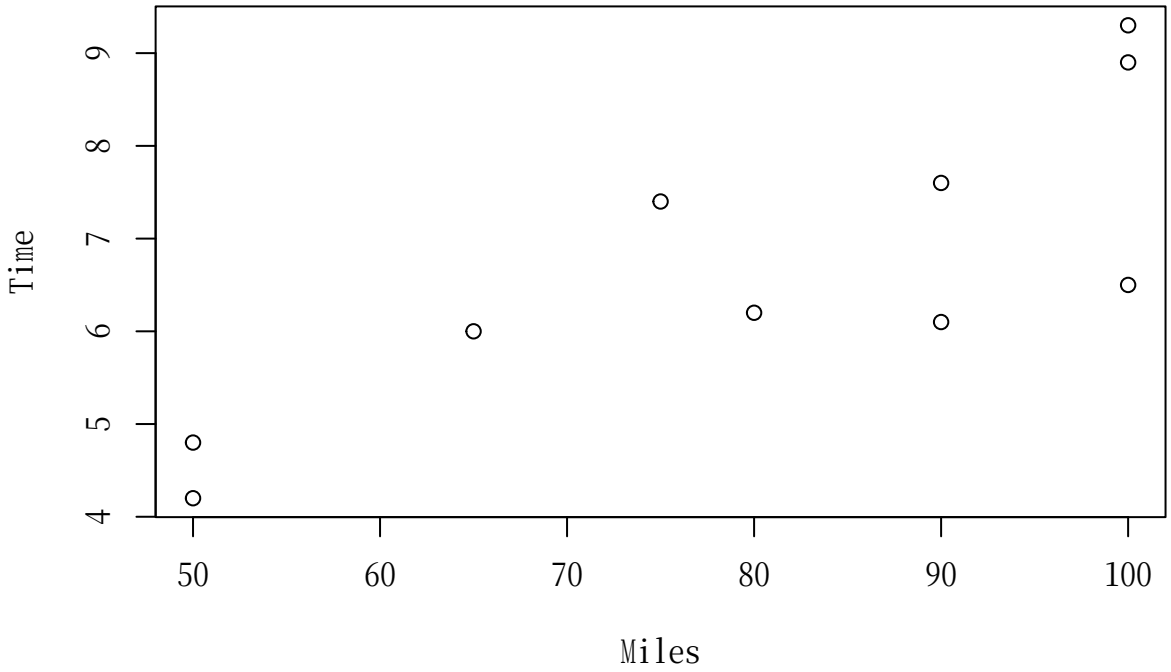
```
## You can cite this package as:
##      Patil, I. (2021). Visualizations with statistical details: The 'ggstatsplot' approach.
##      Journal of Open Source Software, 6(61), 3167, doi:10.21105/joss.03167
```

```
ggcoefstats(lm3)
```



例 32.2. Butler Trucking Company 是一个短途货运公司。老板想研究每个驾驶员每天的行驶时间的影响因素。随机抽取了 10 名驾驶员一天的数据。考虑行驶里程 (Miles) 对行驶时间的影响。

```
with(Butler, plot(Miles, Time))
```



从散点图看，行驶里程与行驶时间有线性相关关系。作一元线性回归：

```
lmbut01 <- lm(Time ~ Miles, data=Butler)
summary(lmbut01)
```

```
##
## Call:
## lm(formula = Time ~ Miles, data = Butler)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.5565 -0.4913  0.1783  0.7120  1.2435
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.27391    1.40074   0.909  0.38969
## Miles        0.06783    0.01706   3.977  0.00408 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 1.002 on 8 degrees of freedom
## Multiple R-squared:  0.6641, Adjusted R-squared:  0.6221
## F-statistic: 15.81 on 1 and 8 DF,  p-value: 0.00408
```

这里 Miles 的系数解释为：行驶里程增加 1 英里，时间平均增加 0.068 小时。老板想进一步改进对行驶时间的预测，增加“派送地点数”（Deliveries）作为第二个自变量。作多元回归：

```
lmbut02 <- lm(Time ~ Miles + Deliveries, data=Butler)
summary(lmbut02)
```

```
##
## Call:
## lm(formula = Time ~ Miles + Deliveries, data = Butler)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.79875 -0.32477  0.06333  0.29739  0.91333
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.868701   0.951548  -0.913 0.391634
## Miles        0.061135   0.009888   6.182 0.000453 ***
## Deliveries   0.923425   0.221113   4.176 0.004157 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5731 on 7 degrees of freedom
## Multiple R-squared:  0.9038, Adjusted R-squared:  0.8763
## F-statistic: 32.88 on 2 and 7 DF,  p-value: 0.0002762
```

这里 Miles 的系数从一元回归时的 0.068 改成了 0.061，解释为：在派送地点数相同的条件下，行驶里程每增加 1 英里，行驶时间平均增加 0.061 小时。

Deliveries 的系数解释为：在行驶里程相同的条件下，派送地点数每增加一个地点，行驶时间平均增加 0.92 小时。

32.4 模型的检验

32.4.1 复相关系数平方

将总平方和分解为：

$$\mathrm{SST} = \mathrm{SSR} + \mathrm{SSE},$$

其中

- 总平方和
$$\mathrm{SST} = \sum_{i=1}^{n}(y_i - \bar{y})^2$$
;
- 回归平方和
$$\mathrm{SSR} = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$$
是能够用回归系数和自变量的变量解释的因变量变化；
- 残差平方和
$$\mathrm{SSE} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$
是回归模型不能解释的因变量变化。

回归平方和越大，残差平方和越小，回归拟合越好。定义复相关系数平方（判定系数）

$$R^2 = \frac{\mathrm{SSR}}{\mathrm{SST}} = 1 - \frac{\mathrm{SSE}}{\mathrm{SST}},$$

则 $0 \le R^2 \le 1$。R^2 越大，说明数据中的自变量拟合因变量值拟合越好。

但是，“拟合好” 不是唯一标准。仅考虑拟合好，可能产生很复杂的仅对建模用数据有效但是对其它数据无效的模型，这称为 “过度拟合”。

定义调整的（修正的）复相关系数平方：

$$R^{*2} = 1 - (1 - R^2)\frac{n-1}{n-p-1},$$

克服了 R^2 的部分缺点。

在体重与身高、年龄的模型结果中，$R^2 = 0.7729$，$R^{*2} = 0.7445$。

32.4.2 残差标准误差

模型中 ε 的方差 σ^2 的估计为 $\hat{\sigma}_e^2$,

$$\hat{\sigma}^2 = \frac{1}{n-p-1}\mathrm{SSE},$$

$\hat{\sigma}$ 是随机误差的标准差 σ 的估计量，称为 “残差标准误差”（Residual standard error）。这是残差 $e_i = y_i - \hat{y}_i$ 的标准差的一个较粗略的近似估计。

在体重与身高、年龄的模型结果中，$\hat{\sigma} = 11.51$。

32.4.3 线性关系检验

为了检验整个回归模型是否都无效，考虑假设检验:

$$H_0 : \beta_1 = \cdots = \beta_p = 0,$$

当 H_0 成立时模型退化成 $y = \beta_0 + \varepsilon$，$y$ 与 $x_1, x_2, \ldots, x_p$ 之间不再具有线性相关关系。

取统计量为

$$F = \frac{\mathrm{SSR}/p}{\mathrm{SSE}/(n-p-1)},$$

在模型前提条件都满足且 H_0 成立时 F 服从 $F(p, n-p-1)$ 分布。计算右侧 p 值，给出检验结论。

当检验显著时，各斜率项不全为零。结果不显著时，当前回归模型不能使用。

在体重与身高、年龄的模型结果中，$F = 27.23$，p 值为 7.074×10^{-6}，在 0.05 水平下显著，模型有意义。

32.4.4 线性关系检验的功效

pwr 包可以计算上述 F 检验的功效，对立假设是至少有一个斜率项不等于 0，计算功效时针对的效应大小定义为 R^2 的如下变换：

$$f_2 = \frac{R^2}{1-R^2}.$$

体重对年龄、身高的回归获得的 $R^2 = 0.7729$。按此 R^2 对应的效应大小，计算功效：

```
library(pwr)
```

```
## Warning: package 'pwr' was built under R version 4.2.2
```

```
R2 <- 0.7729
pwr.f2.test(
  u = 2,              # 斜率项个数
  v = 19 - 2 - 1,
  # 19 为观测数，v 为 F 统计量的分母自由度
  f2 = R2 / (1 - R2), # 效应大小
  sig.level = 0.01)   # 检验水平
```

```
##
##      Multiple regression power calculation
##
##               u = 2
##               v = 16
##              f2 = 3.403347
##       sig.level = 0.01
##           power = 0.99995
```

功效为 100%。这是因为 R^2 很大。

计算达到 80% 功效所需样本量：

```
pwr.f2.test(
  u = 2,              # 斜率项个数
  power = 0.80,       # 功效
  f2 = R2 / (1 - R2), # 效应大小
  sig.level = 0.01)   # 检验水平
```

```
##
##      Multiple regression power calculation
##
##               u = 2
##               v = 6.324025
##              f2 = 3.403347
##       sig.level = 0.01
##           power = 0.8
```

样本量为 $n = u + v + 1 = 9.32$ 即只需要 10 个观测。

32.4.5 单个斜率项的显著性检验

为了检验某一个自变量 X_j 是否对因变量的解释有贡献（在模型中已经包含了其它自变量的情况下），检验

$$H_0 : \beta_j = 0,$$

如果不拒绝 H_0，相当于 x_j 可以不出现在模型中。但是，在多元线性回归中这不代表 x_j 与 y 之间没有线性相关，可能是由于其它的自变量包含了 x_j 中的信息。

检验使用如下 t 统计量

$$t = \frac{\hat{\beta}_j}{\mathrm{SE}(\hat{\beta}_j)},$$

在模型前提条件都满足且 H_0 成立时 t 服从 $t(n-p-1)$ 分布。给定检验水平 α，计算双侧 p 值，做出检验结论。

对单个回归系数的检验，检验水平可取得略高，如 0.10, 0.15。

在体重与身高、年龄的模型结果中，身高的斜率项显著性 p 值为 0.001, 在 0.15 水平下显著；体重的斜率项显著性 p 值为 0.686，在 0.15 水平下不显著，可考虑从模型中去掉体重变量。

32.5 回归自变量筛选

例 32.3. 考虑 19 个学生的体重与身高、年龄的回归模型。

在 19 个学生的体重与身高、年龄的线性回归模型结果中，发现关于年龄的系数为零的检验 p 值为 0.686, 不显著，说明在模型中已经包含身高的情况下，年龄不提供对体重的额外信息。但是如果体重对年龄单独建模的话，年龄的影响还是显著的：

```
lm4 <- lm(weight ~ age, data=d.class)
summary(lm4)
```

```
##
## Call:
## lm(formula = weight ~ age, data = d.class)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -23.349  -7.609  -5.260   7.945  42.847
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -50.493     33.290  -1.517 0.147706
## age           11.304      2.485   4.548 0.000285 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.74 on 17 degrees of freedom
## Multiple R-squared:  0.5489, Adjusted R-squared:  0.5224
## F-statistic: 20.69 on 1 and 17 DF,  p-value: 0.0002848
```

模型中不显著的自变量应该逐一剔除。可以用 `step` 函数进行逐步回归变量选择，如:

```
lm5 <- step(lm(weight ~ height + age + sex, data=d.class))
```

```
## Start:  AIC=94.93
## weight ~ height + age + sex
##
##          Df Sum of Sq    RSS     AIC
## - age     1    113.76 1957.8  94.067
## <none>                1844.0  94.930
## - sex     1    276.09 2120.1  95.581
## - height  1   1020.61 2864.6 101.299
##
## Step:  AIC=94.07
```

```
## weight ~ height + sex
##
##          Df Sum of Sq    RSS     AIC
## - sex     1     184.7 2142.5  93.780
## <none>                1957.8  94.067
## - height  1    5696.8 7654.6 117.974
##
## Step:  AIC=93.78
## weight ~ height
##
##          Df Sum of Sq    RSS    AIC
## <none>                2142.5  93.78
## - height  1    7193.2 9335.7 119.75
```

```
summary(lm5)
```

```
##
## Call:
## lm(formula = weight ~ height, data = d.class)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.6807  -6.0642   0.5115   9.2846  18.3698
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -143.0269    32.2746  -4.432 0.000366 ***
## height         3.8990     0.5161   7.555 7.89e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.23 on 17 degrees of freedom
## Multiple R-squared:  0.7705, Adjusted R-squared:  0.757
## F-statistic: 57.08 on 1 and 17 DF,  p-value: 7.887e-07
```

例 32.4. 餐馆营业额的回归建模分析。

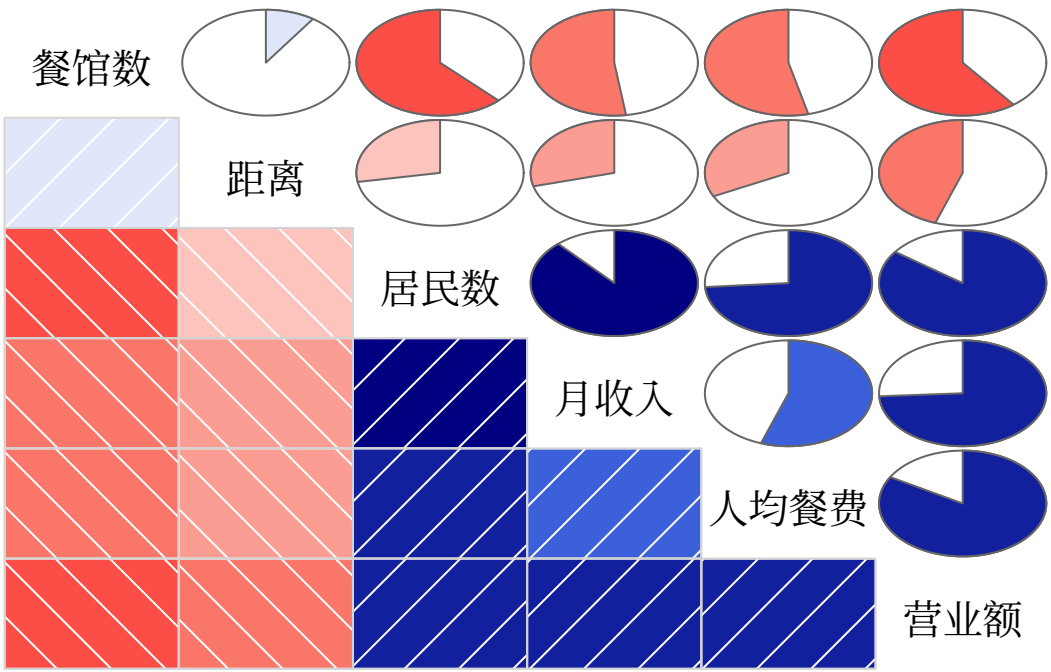
考虑另一个例子数据。在数据框 Resturant 中包含 25 个餐馆的一些信息，包括：

- y, 营业额, 日均营业额（万元）
- x_1, 居民数, 周边居民人数（万人）

- x_2, 人均餐费, 用餐平均支出 (元/人)
- x_3, 月收入, 周边居民月平均收入（元）
- x_4, 餐馆数, 周边餐馆数（个）
- x_5, 距离, 距市中心距离（km）

对营业额进行回归建模研究。变量间的相关系数图：

```
corrgram::corrgram(
  Resturant, order=TRUE,
  lower.panel=corrgram::panel.shade,
  upper.panel = corrgram::panel.pie,
  text.panel = corrgram::panel.txt)
```




```
lmrst01 <- lm(`营业额` ~ `居民数` + `人均餐费` + `月收入` + `餐馆数` + `距离`,
              data=Resturant)
summary(lmrst01)
```

```
##
## Call:
```

```
## lm(formula = 营业额 ~ 居民数 + 人均餐费 + 月收入 +
##     餐馆数 + 距离, data = Resturant)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.7204  -6.0600   0.7152   3.2144  21.4805
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.2604768 10.4679833   0.407  0.68856
## 居民数       0.1273254  0.0959790   1.327  0.20037
## 人均餐费     0.1605660  0.0556834   2.884  0.00952 **
## 月收入       0.0007636  0.0013556   0.563  0.57982
## 餐馆数      -0.3331990  0.3986248  -0.836  0.41362
## 距离        -0.5746462  0.3087506  -1.861  0.07826 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.65 on 19 degrees of freedom
## Multiple R-squared:  0.8518, Adjusted R-squared:  0.8128
## F-statistic: 21.84 on 5 and 19 DF,  p-value: 2.835e-07
```

在 0.15 水平上只有人均餐费和到市中心距离是显著的。进行逐步回归筛选：

```
lmrst02 <- step(lmrst01)
```

```
## Start:  AIC=123.39
## 营业额 ~ 居民数 + 人均餐费 + 月收入 + 餐馆数 +
##     距离
##
##            Df Sum of Sq    RSS    AIC
## - 月收入    1     35.96 2189.0 121.81
## - 餐馆数    1     79.17 2232.2 122.30
## <none>                  2153.0 123.39
## - 居民数    1    199.42 2352.4 123.61
## - 距离      1    392.54 2545.6 125.58
## - 人均餐费  1    942.22 3095.2 130.47
##
## Step:  AIC=121.81
## 营业额 ~ 居民数 + 人均餐费 + 餐馆数 + 距离
```

```
##
##            Df Sum of Sq    RSS    AIC
## - 餐馆数    1     78.22 2267.2 120.69
## <none>                  2189.0 121.81
## - 距离      1    445.69 2634.7 124.44
## - 人均餐费  1    925.88 3114.9 128.63
## - 居民数    1   1133.27 3322.3 130.24
##
## Step:  AIC=120.69
## 营业额 ~ 居民数 + 人均餐费 + 距离
##
##            Df Sum of Sq    RSS    AIC
## <none>                  2267.2 120.69
## - 距离      1    404.28 2671.5 122.79
## - 人均餐费  1   1050.90 3318.1 128.21
## - 居民数    1   1661.83 3929.0 132.43
```

```
summary(lmrst02)
```

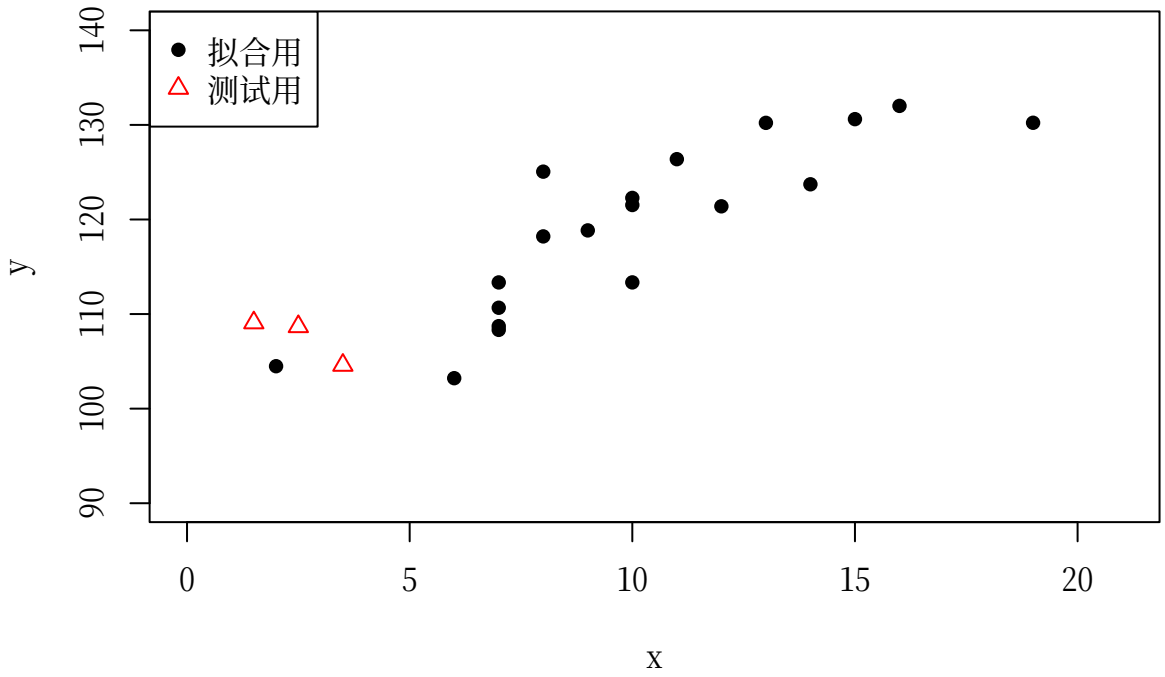
```
##
## Call:
## lm(formula = 营业额 ~ 居民数 + 人均餐费 + 距离, data = Resturant)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -14.027  -5.361  -1.560   2.304  23.001
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.68928    6.25242  -0.270  0.78966
## 居民数       0.19022    0.04848   3.923  0.00078 ***
## 人均餐费     0.15763    0.05052   3.120  0.00518 **
## 距离        -0.56979    0.29445  -1.935  0.06656 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.39 on 21 degrees of freedom
## Multiple R-squared:  0.8439, Adjusted R-squared:  0.8216
## F-statistic: 37.85 on 3 and 21 DF,  p-value: 1.187e-08
```

最后进保留了周边居民人数、人均餐费、到市中心距离三个自变量，删去了周边居民月收入和周边餐馆数两个自变量。

32.6 过度拟合示例

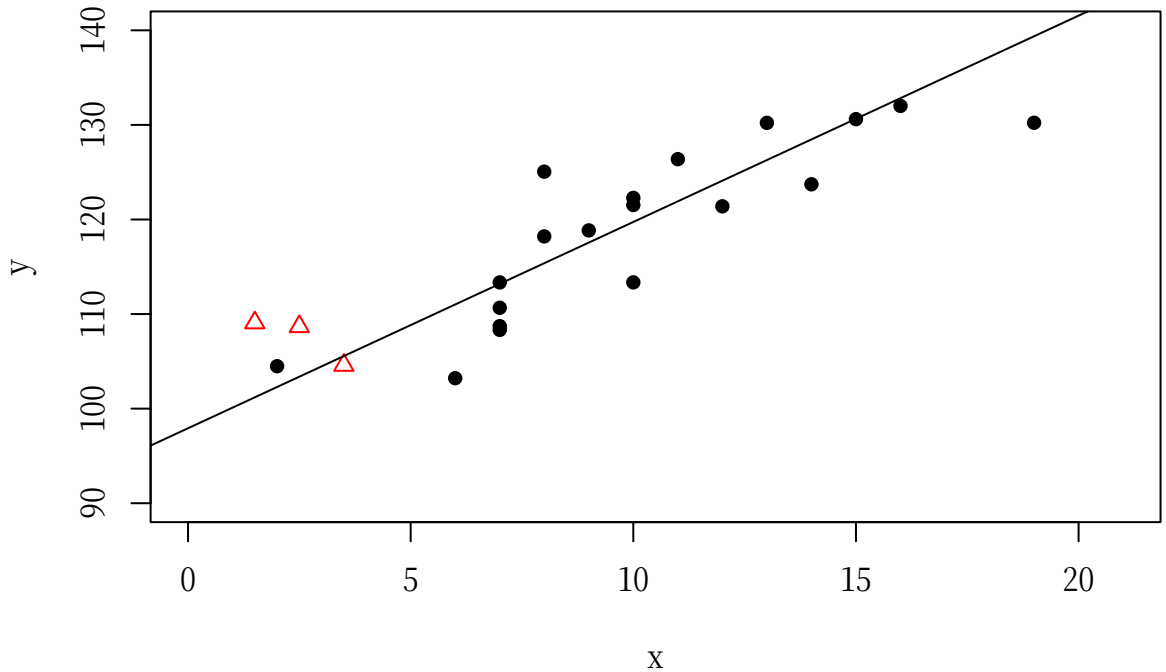
R^2 代表了模型对数据的拟合程度，模型中加入的自变量越多，R^2 越大。是不是模型中的自变量越多越好？自变量过多时可能会发生 “过度拟合”，这时用来建模的数据都拟合误差很小，但是模型很难有合理解释，对新的数据的预测效果很差甚至于完全错误。下面给出模拟数据例子。

```
set.seed(10)
n <- 20
x <- sample(1:n, size=n, replace=TRUE)
a <- 100
b <- 2
sigma <- 5
y <- a + b*x + rt(n, 4)*sigma
xnew <- c(1.5, 2.5, 3.5)
ynew <- a + b*xnew + rnorm(length(xnew), 0, sigma)
plot(x, y, pch=16, xlim=c(0, n+1), ylim=c(90,140))
points(xnew, ynew, pch=2, col="red")
legend("topleft", pch=c(16,2), col=c("black", "red"),
       legend=c(" 拟合用", " 测试用"))
```



作线性回归：

```
plot(x, y, pch=16, xlim=c(0, n+1), ylim=c(90,140))
points(xnew, ynew, pch=2, col="red")
lmof1 <- lm(y ~ x)
abline(lmof1)
```



回归系数：

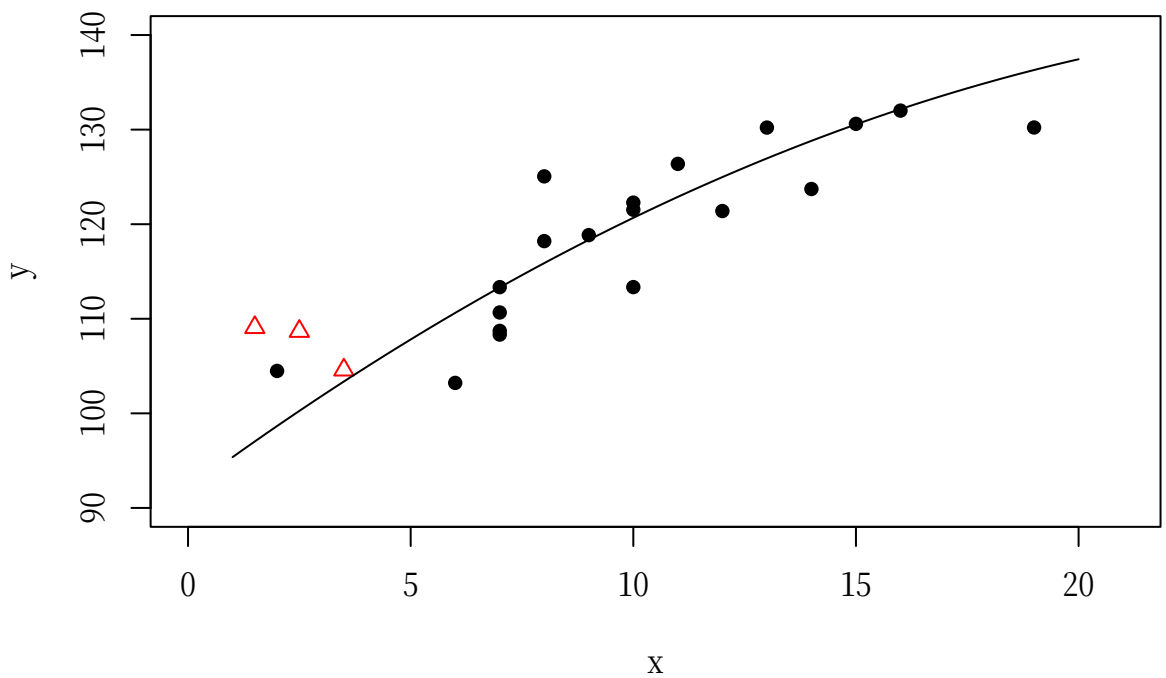
```
summary(lmof1)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.1457 -4.5412  0.0573  2.6144 14.6213
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  97.9248     3.6616  26.744 6.07e-16 ***
## x             2.1814     0.3314   6.583 3.49e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.93 on 18 degrees of freedom
```

```
## Multiple R-squared:  0.7065, Adjusted R-squared:  0.6902
## F-statistic: 43.33 on 1 and 18 DF,  p-value: 3.493e-06
```

作二次多项式回归：

```
plot(x, y, pch=16, xlim=c(0, n+1), ylim=c(90,140))
points(xnew, ynew, pch=2, col="red")
lmof2 <- lm(y ~ x + I(x^2))
xx <- seq(1, n, length=100)
yy <- predict(lmof2, newdata=data.frame(x=xx))
lines(xx, yy)
```



回归系数：

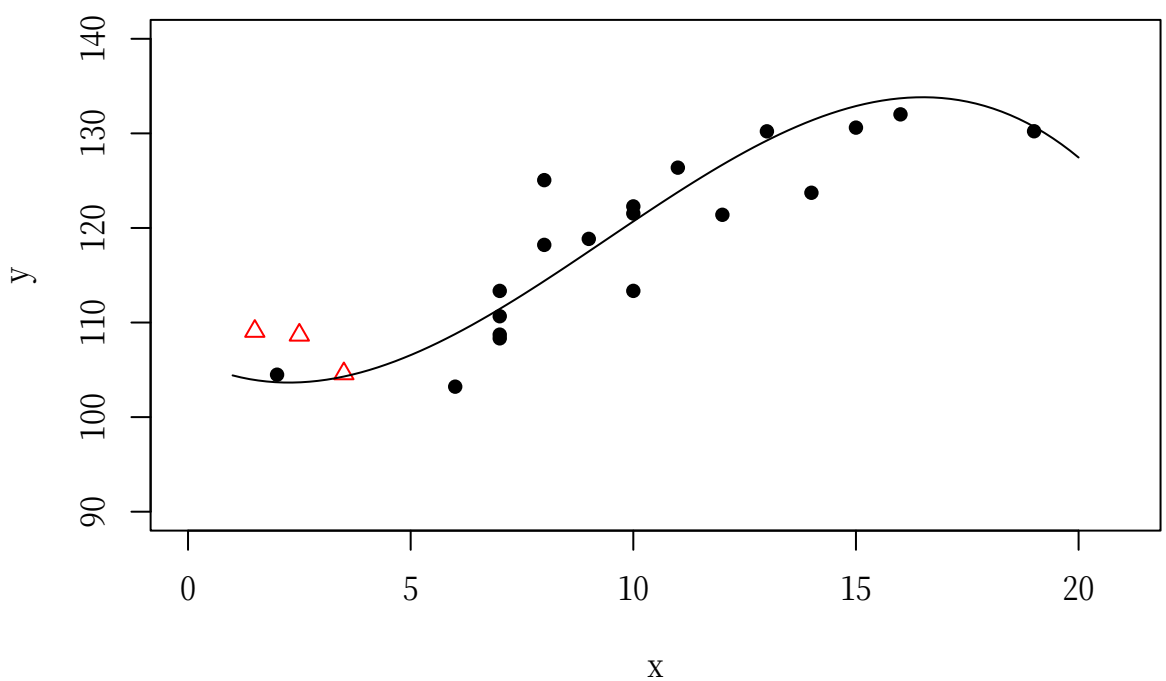
```
summary(lmof2)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2))
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.4005 -4.6897  0.0461  2.5595 14.7168
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 91.95984    7.52881  12.214 7.67e-10 ***
## x            3.46829    1.45573   2.383   0.0291 *
## I(x^2)      -0.05971    0.06575  -0.908   0.3765
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.959 on 17 degrees of freedom
## Multiple R-squared:  0.7201, Adjusted R-squared:  0.6872
## F-statistic: 21.87 on 2 and 17 DF,  p-value: 1.993e-05
```

这个回归结果出现了多重共线性问题。也已经过度拟合。

作三次多项式回归：



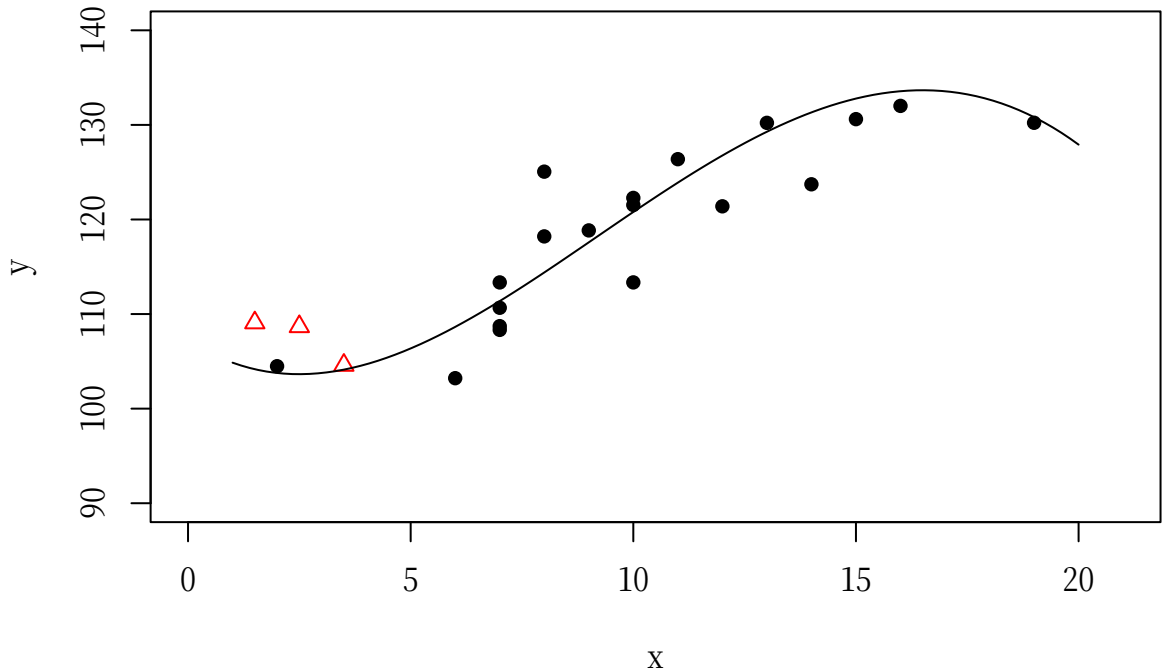
回归系数：

```
summary(lmof3)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2) + I(x^3))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.6352 -2.8162  0.1635  1.6680 12.3784
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 106.19613   10.37451  10.236 1.98e-08 ***
## x            -2.35124    3.40146  -0.691   0.4993
## I(x^2)        0.58944    0.35317   1.669   0.1146
## I(x^3)       -0.02094    0.01122  -1.867   0.0804 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.566 on 16 degrees of freedom
## Multiple R-squared:  0.7702, Adjusted R-squared:  0.7271
## F-statistic: 17.87 on 3 and 16 DF,  p-value: 2.32e-05
```

这个回归结果出现了多重共线性问题。也已经过度拟合。

作四次多项式回归：



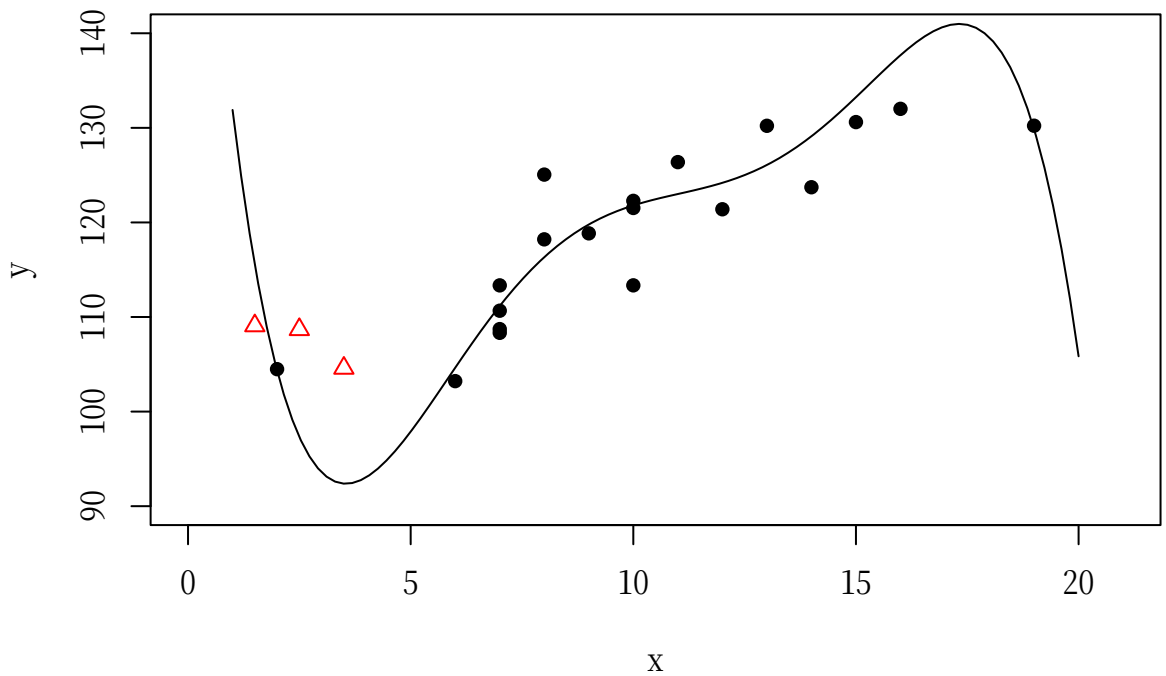
回归系数：

```
summary(lmof4)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.5998 -2.7462  0.0614  1.6083 12.4813
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.072e+02  1.803e+01   5.947 2.68e-05 ***
## x           -3.043e+00  1.047e+01  -0.291    0.775
## I(x^2)       7.201e-01  1.898e+00   0.379    0.710
## I(x^3)      -3.020e-02  1.325e-01  -0.228    0.823
## I(x^4)       2.192e-04  3.126e-03   0.070    0.945
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.747 on 15 degrees of freedom
## Multiple R-squared:  0.7702, Adjusted R-squared:  0.709
## F-statistic: 12.57 on 4 and 15 DF,  p-value: 0.0001098
```

作五次多项式回归：



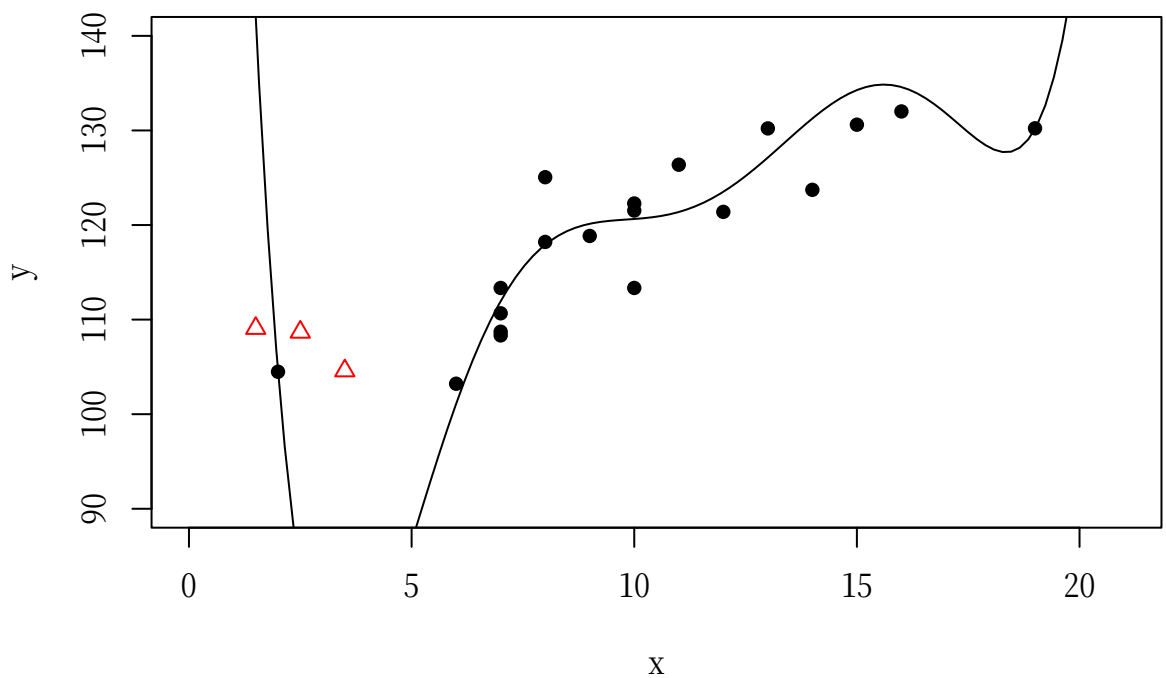
已经明显过度拟合。这时可以看出对三个测试点（图中三角形点）中最左边一个的预测效果极差。

回归系数：

```
summary(lmof5)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.4506 -2.6704 -0.3829  1.9584 12.0281
```

```
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 184.694327  52.805353   3.498  0.00355 **
## x           -68.766010  43.505251  -1.581  0.13628
## I(x^2)       17.817606  11.162736   1.596  0.13277
## I(x^3)       -1.948770   1.242420  -1.569  0.13908
## I(x^4)        0.097357   0.062647   1.554  0.14248
## I(x^5)       -0.001818   0.001171  -1.552  0.14289
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.495 on 14 degrees of freedom
## Multiple R-squared:  0.804,  Adjusted R-squared:  0.734
## F-statistic: 11.48 on 5 and 14 DF,  p-value: 0.0001494
```



六次多项式回归：

明显过度拟合。可以看出对三个测试点（图中三角形点）中最左边一个的预测效果极差。

回归系数：

```
summary(lmof6)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.5413 -2.7415  0.0219  1.7613 10.9967
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.032e+02  1.833e+02   2.199   0.0466 *
## x           -2.773e+02  1.732e+02  -1.601   0.1334
## I(x^2)       8.500e+01  5.517e+01   1.541   0.1474
## I(x^3)      -1.217e+01  8.318e+00  -1.463   0.1671
## I(x^4)       9.007e-01  6.495e-01   1.387   0.1888
## I(x^5)      -3.332e-02  2.538e-02  -1.313   0.2120
## I(x^6)       4.871e-04  3.921e-04   1.242   0.2360
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.391 on 13 degrees of freedom
## Multiple R-squared:  0.8248, Adjusted R-squared:  0.7439
## F-statistic:  10.2 on 6 and 13 DF,  p-value: 0.000278
```

32.7 残差诊断

先复习一些回归理论。将模型写成矩阵形式

$$Y = X\beta + \varepsilon,$$

Y 为 $n \times 1$ 向量, X 为 $n \times p$ 矩阵（$n > p$）, 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)。

假设矩阵 X 列满秩，系数的估计为

$$\hat{\beta} = \left(X^T X\right)^{-1} X^T Y,$$

拟合值 (或称预报值) 向量为

$$\hat{Y} = X\left(X^T X\right)^{-1} X^T Y = HY,$$

其中 $H = X\left(X^TX\right)^{-1}X^T$ 是 R^n 空间的向量向 X 的列张成的线性空间 $\mu(X)$ 投影的投影算子矩阵, 叫做**帽子矩阵**。设 $H = \left(h_{ij}\right)_{n\times n}$。

拟合残差向量为

$$e = Y - \hat{Y} = (I-H)Y,$$

满足

$$\begin{aligned} Ee =& 0, \\ \mathrm{Var}(e) =& \sigma^2(I-H). \end{aligned}$$

残差平方和为

$$\mathrm{ESS} = e^Te = \sum_{i=1}^n \left(Y_i - \hat{Y}_i\right)^2.$$

误差项方差的估计 (要求设计阵 X 满秩) 为均方误差 (MSE)

$$\hat{\sigma}^2 = \mathrm{MSE} = \frac{1}{n-p}\mathrm{ESS}$$

（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 X 满秩, $\hat{\beta}$ 和 $\hat{\sigma}^2$ 分别是 β 和 σ^2 的无偏估计。

系数估计的方差阵

$$\mathrm{Var}(\hat{\beta}) = \sigma^2\left(X^TX\right)^{-1}.$$

回归残差及其方差为

$$\begin{aligned} e_i =& y_i - \hat{y}_i, \quad i = 1,2,\dots,n, \\ \mathrm{Var}(e_i) =& \sigma^2(1-h_{ii}) \quad (h_{ii}\text{是}H\text{的主对角线元素}). \end{aligned}$$

若 `lmres` 是 R 中 `lm()` 的回归结果，用 `residuals(lmres)` 可以求残差。

把 e_i 除以其标准差估计，称为**标准化残差**，或**内部学生化残差**：

$$r_i = \frac{e_i}{s\sqrt{1-h_{ii}}}, \quad i = 1,2,\dots,n$$

其中 $s = \hat{\sigma}$。r_i 渐近服从正态分布。

若 `lmres` 是 R 中 `lm()` 的回归结果，用 `rstandard(lmres)` 可以求标准化残差。

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 $s_{(i)}^2$，则**外部学生化残差**为

$$t_i = \frac{e_i}{s_{(i)}\sqrt{1-h_{ii}}}$$

t_i 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）。其中 $s_{(i)}$ 有简单公式：

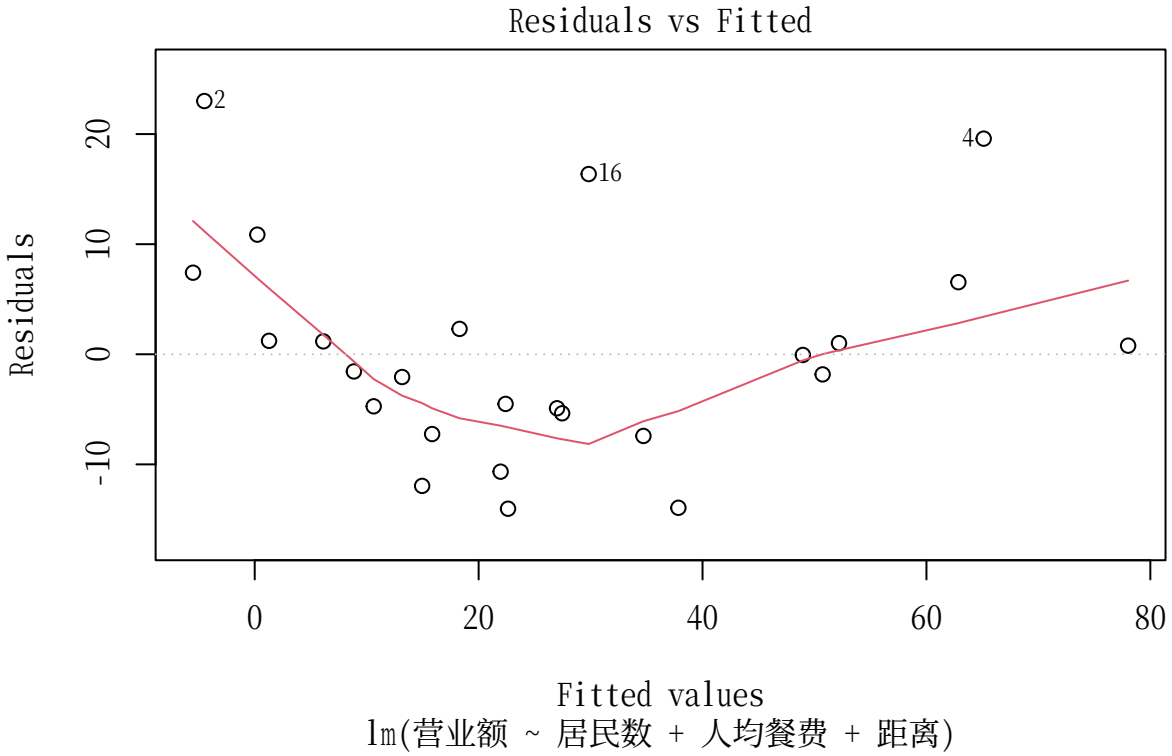
$$s_{(i)}^2 = \frac{n-p-r_i^2}{n-p-1}\hat{\sigma}^2$$

若 `lmres` 是 R 中 `lm()` 的回归结果，用 `rstudent(lmres)` 可以求外部学生化残差。

在 R 中，与一元回归的诊断类似。用 `plot()` 作 4 个残差诊断图。可以用 `which=1` 指定仅作第一幅图。

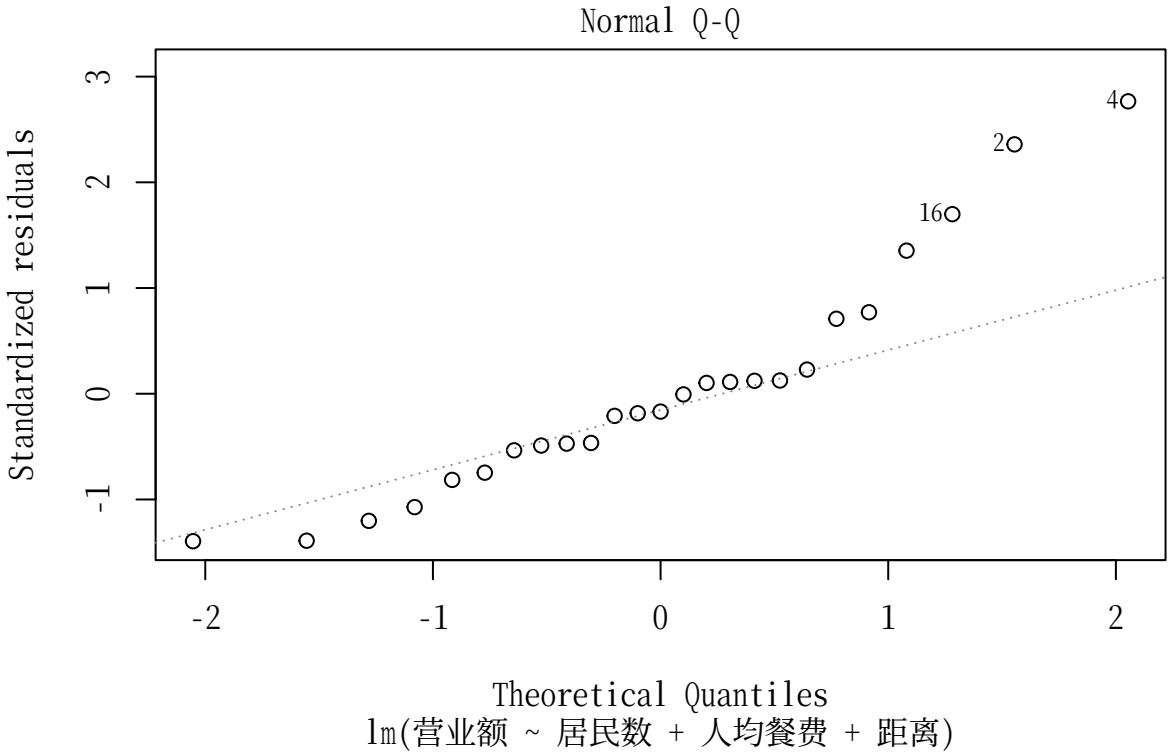
如餐馆营业额例子的残差诊断：

```
plot(lmrst02, which=1)
```



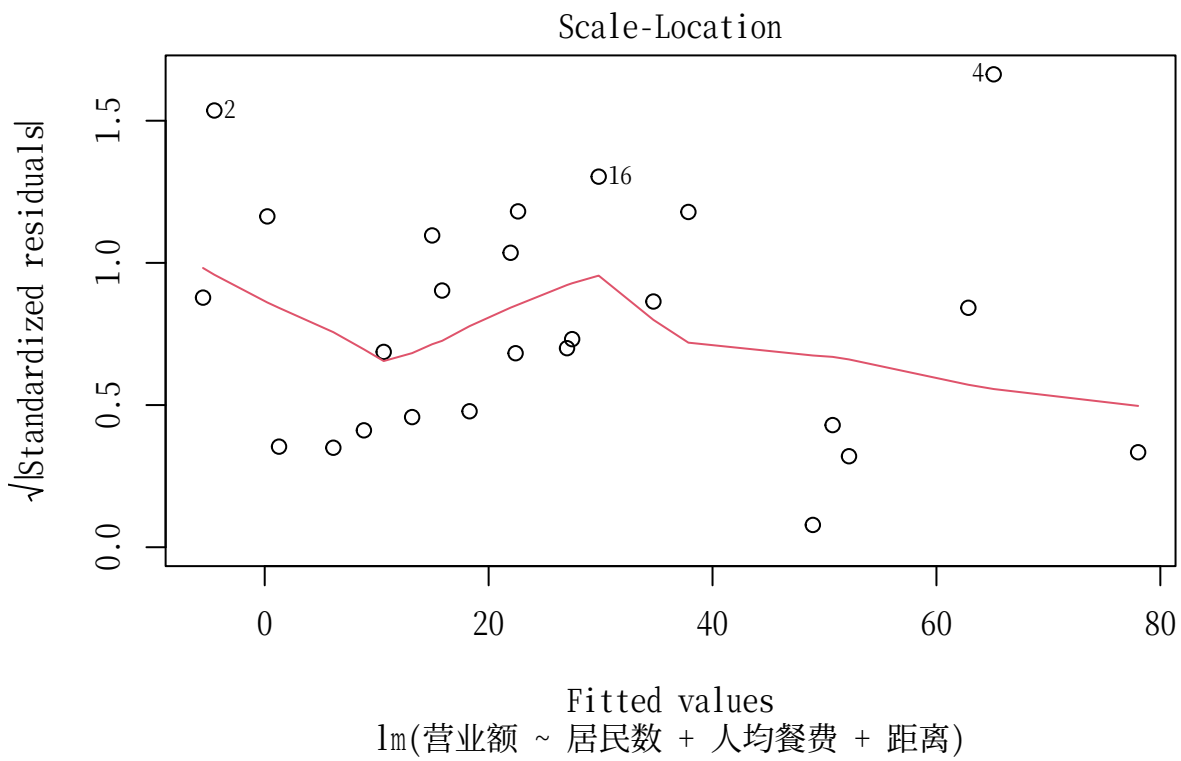
上图是残差对拟合值的散点图，可以查看有无非线性。有轻微的非线性关系。

```
plot(lmrst02, which=2)
```



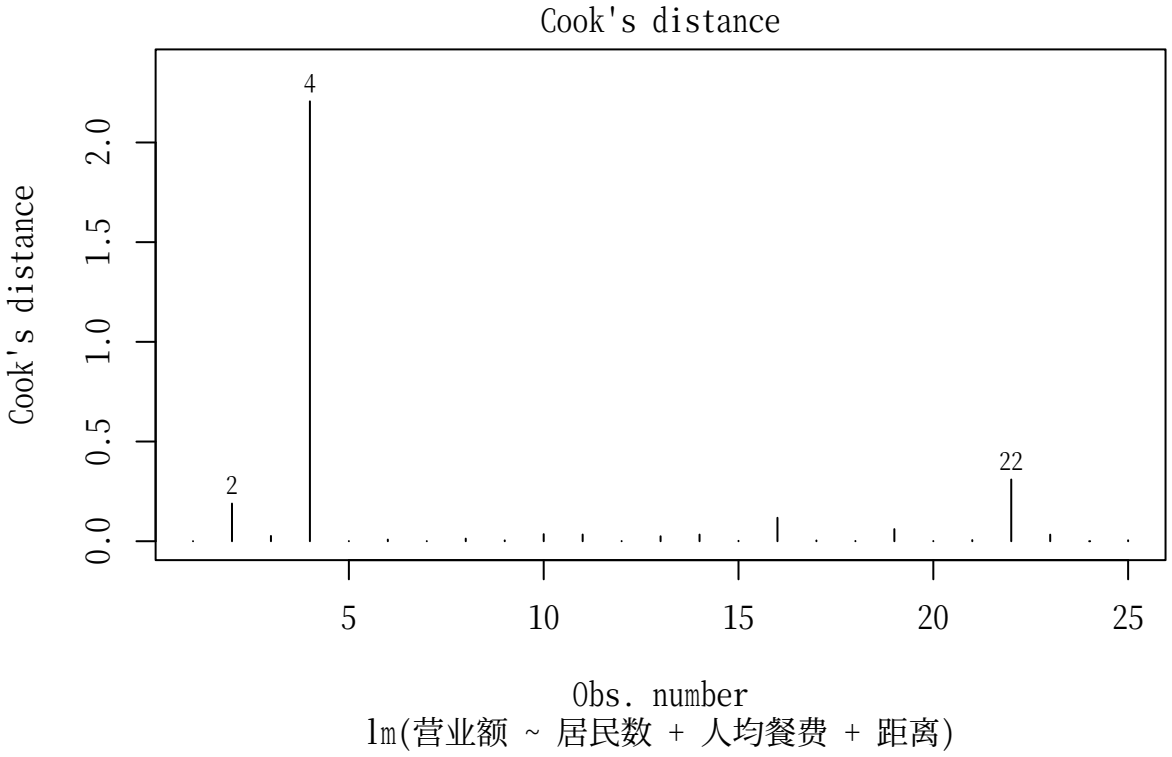
上图是残差的正态 QQ 图，查看残差是否正态分布。残差分布略有右偏，不算太严重。

```
plot(lmrst02, which=3)
```



上图是标准化残差绝对值平方根对拟合值的散点图，可以查看是否有异方差。没有明显的异方差。

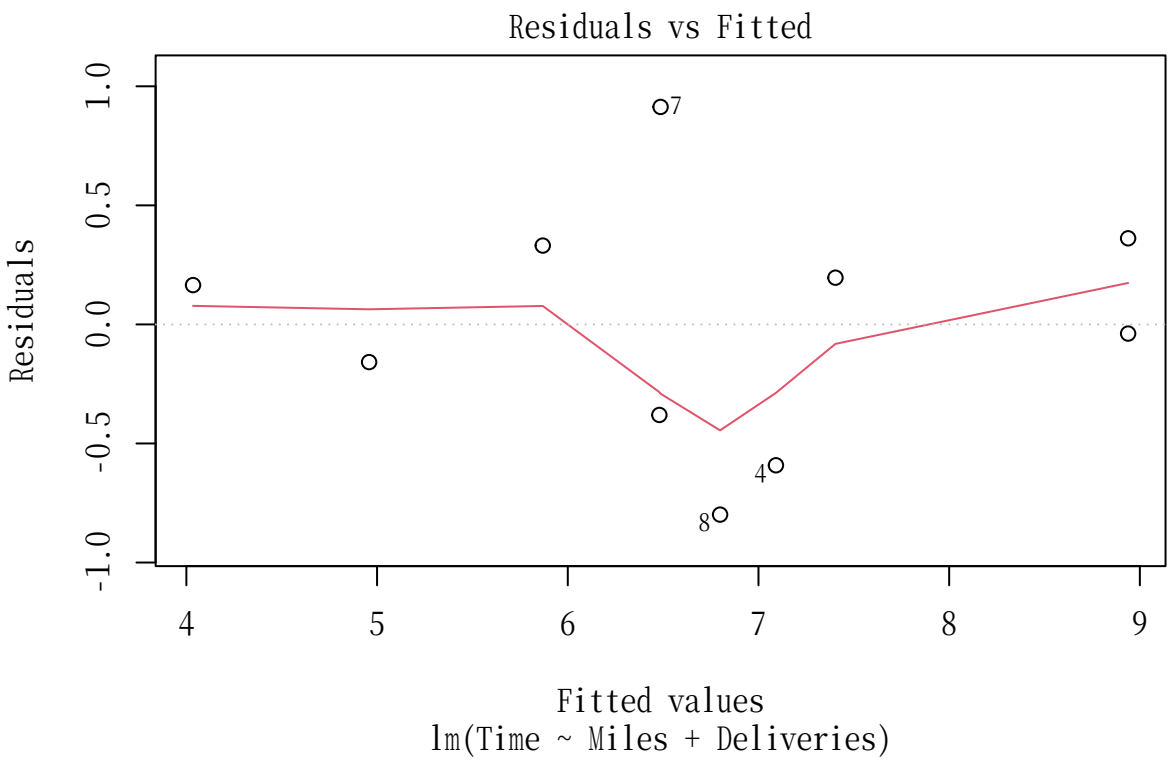
```
plot(lmrst02, which=4)
```

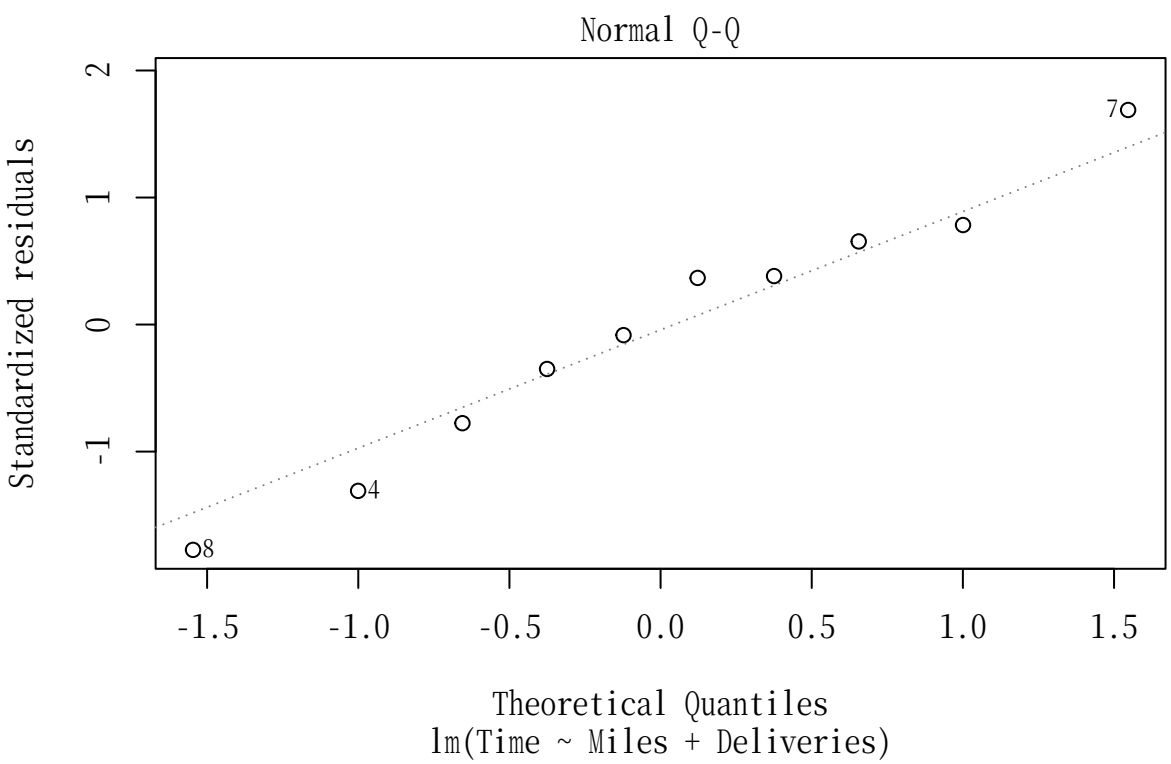


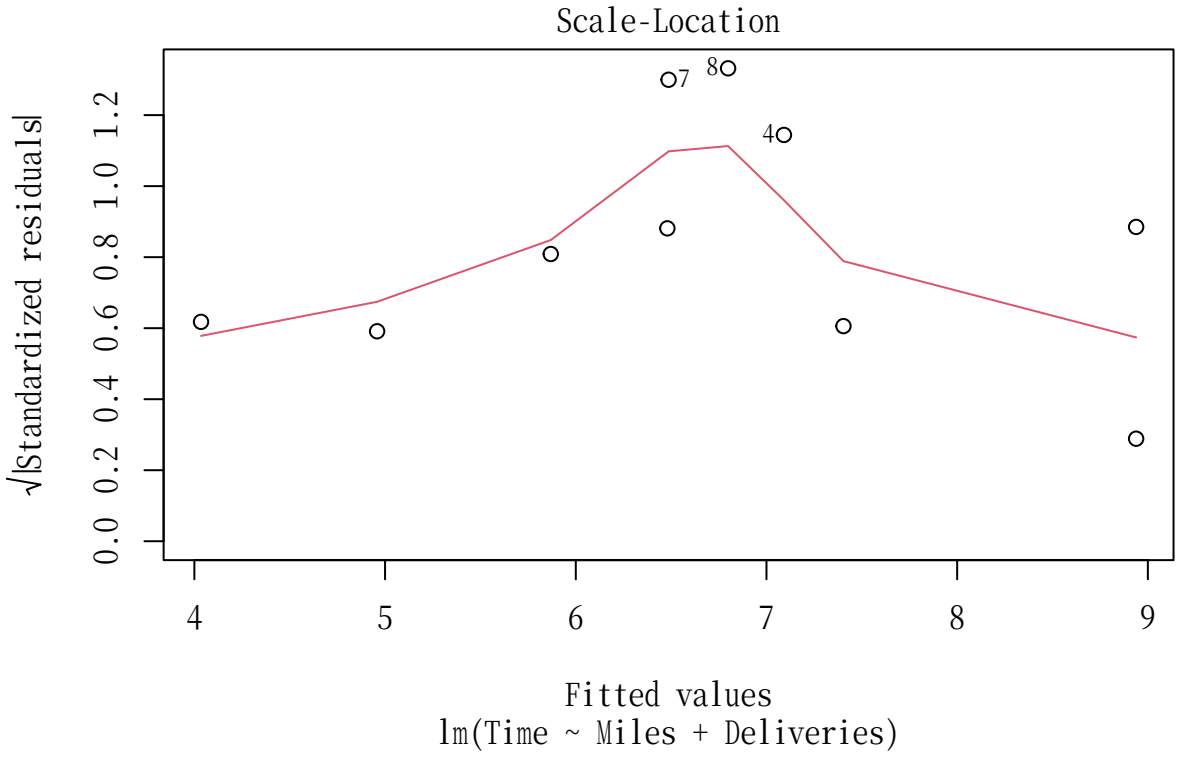
上图用来查看强影响点，4 号观测是一个强影响点。

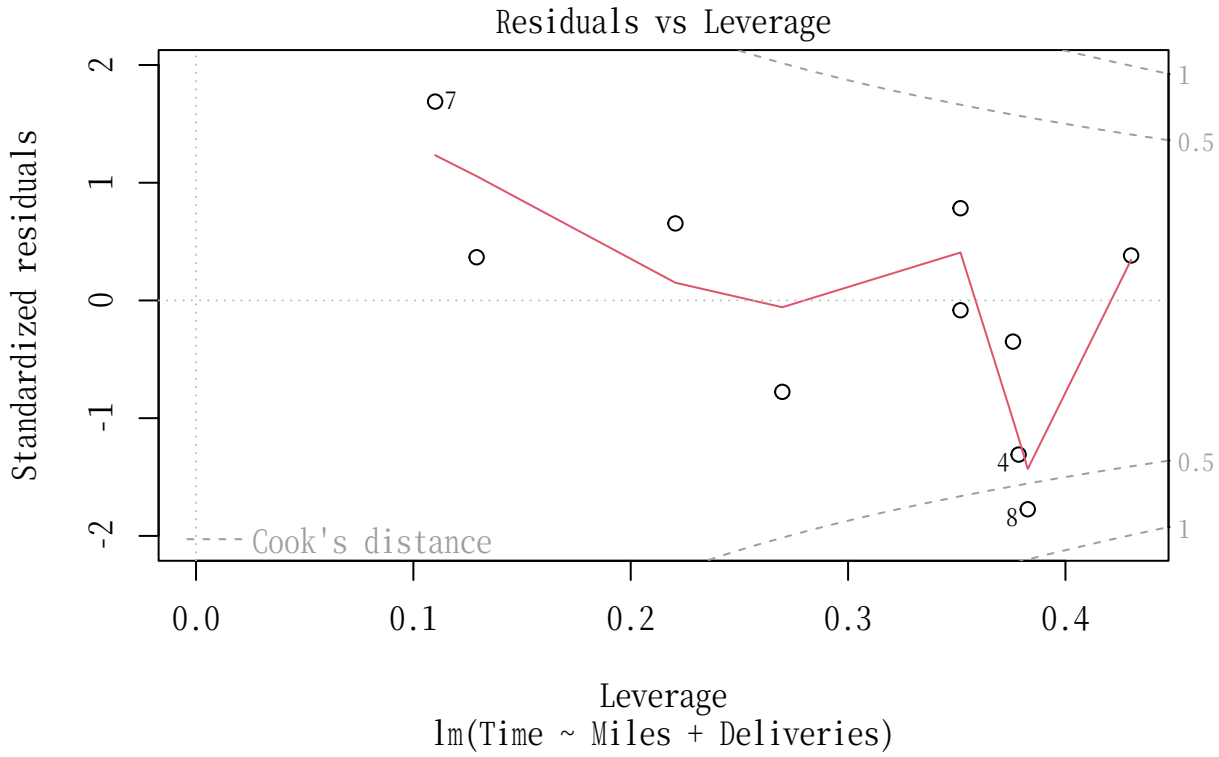
货运公司例子的多元回归的残差诊断：

```
plot(lmbut02)
```


Residuals vs Fitted
Residuals
1.0
0.5
0.0
-0.5
-1.0
7
4
8
4
5
6
7
8
9
Fitted values
lm(Time ~ Miles + Deliveries)


Normal Q-Q
Standardized residuals
2
1
0
-1
7
4
8
-1.5
-1.0
-0.5
0.0
0.5
1.0
1.5
Theoretical Quantiles
lm(Time ~ Miles + Deliveries)


Scale-Location
√|Standardized residuals|
0.0 0.2 0.4 0.6 0.8 1.0 1.2
4 5 6 7 8 9
7
8
4
Fitted values
lm(Time ~ Miles + Deliveries)


Residuals vs Leverage
Standardized residuals
2 1 0 -1 -2
0.0 0.1 0.2 0.3 0.4
7
4
8
1
0.5
0.5
1
Cook's distance
Leverage
lm(Time ~ Miles + Deliveries)

有一定的异方差倾向，但是数据量不大就不做处理。

32.8 多重共线性

狭义的多重共线性（multicollinearity）：自变量的数据存在线性组合近似地等于零，使得解线性方程组求解回归系数时结果不稳定，回归结果很差。

广义的多重共线性：自变量之间存在较强的相关性，这样自变量是联动的，互相之间有替代作用。甚至于斜率项的正负号都因为这种替代作用而可能是错误的方向。

例如，餐馆营业额例子中，F 检验显著，5 个自变量如果用在一元回归中斜率项都显著，但是在多元回归中，在 0.15 水平下仅有人均餐费和到市中心的距离的系数是显著的，月收入、餐馆数、居民数的系数不显著。

但是实际上，单独使用这三个自变量作一元线性回归，结果都是显著的，比如单独以月收入为自变量进行一元回归：

```
summary(lm(`营业额` ~ `月收入`, data=Resturant))
```

```
##
## Call:
## lm(formula = 营业额 ~ 月收入, data = Resturant)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -32.151 -10.725  -0.696   6.033  47.819
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.484580   5.955478   0.081    0.936
## 月收入      0.004995   0.000944   5.291 2.27e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.88 on 23 degrees of freedom
## Multiple R-squared:  0.549,  Adjusted R-squared:  0.5294
## F-statistic: 27.99 on 1 and 23 DF,  p-value: 2.273e-05
```

如何识别多重共线性？

- 如果两个自变量之间的相关系数显著地不等于零，这两个自变量就有广义的共线性。
- 如果线性关系的 F 检验显著但是单个回归系数都不显著，可能是由于多重共线性。
- 如果有单个回归系数显著但是 F 检验不显著，可能是由于多重共线性。

- 如果某些回归系数的正负号与通常的认识相反，可能是由于多重共线性。

将第 i 个自变量 x_i 作为因变量，用其它的 $p-1$ 个自变量作为自变量作多元线性回归，得到一个复相关系数平方 R_i^2，这个 R_i^2 接近于 1 时 x_i 与其他自变量之间存在多重共线性。令 x_i 的容忍度（tolerance）等于 $1-R_i^2$，容忍度接近于 0 时存在多重共线性。

容忍度小于 0.1 时多重共线性为严重程度。

称容忍度的倒数为方差膨胀因子（VIF），VIF 大于 10 或者大于 5 作为严重的多重共线性。

为了计算 VIF, 首先把矩阵 X^TX 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。

car 包的 `vif()` 函数计算方差膨胀因子。

如：

```
car::vif(lmrst01)
```

```
##    居民数 人均餐费    月收入    餐馆数      距离
## 8.233159 2.629940 5.184365 1.702361 1.174053
```

可以认为变量 “居民数” 和 “月收入” 有共线问题。

做共线诊断还可以用条件数 (Conditional Index): 这是一个正数，用来衡量 $(X^TX)^{-1}$ 的稳定性，定义为 X^TX 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。

解决多重共线性问题，最简单的方法是回归自变量选择，剔除掉有严重共线性的自变量，这些自变量的信息可以由其他变量代替。还可以对自变量作变换，如用主成分分析降维。可以用收缩方法如岭回归、lasso 回归等。

32.9 强影响点分析

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。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）。

若 `lmres` 是 R 中 `lm()` 的回归结果，用 `hatvalues(lmres)` 可以求杠杆值。

考察外学生化残差 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。

若 `lmres` 是 R 中 `lm()` 的回归结果，用 `rstudent(lmres)` 可以求外学生化残差。

Cook 距离统计量

$$D_i = \frac{1}{p} r_i^2 \frac{h_{ii}}{1-h_{ii}}$$

用来度量估计模型时是否使用第 i 个观测对回归系数 β 的估计的影响。超过 $\frac{4}{n}$ 的值需要注意。若 `lmres` 是 R 中 `lm()` 的回归结果，用 `cooks.distance(lmres)` 可以求 Cook 距离。

R 中的强影响点诊断函数还有 `dfbetas()`, `dffits()`, `covratio()`。

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。

32.10 协方差分析

32.10.1 哑变量

回归分析的因变量是连续型的，服从正态分布。回归分析的自变量是数值型的，可以连续取值也可以离散取值。但是，如果自变量是类别变量，用简单的 1,2,3 编码并不能正确地表现不同类别的作用。可以将类别变量转换成一个或者多个取 0、1 值的变量，称为哑变量（dummy variables）或虚拟变量。如果模型中既有连续型自变量又有分类自变量，称这样的模型为协方差分析模型。

32.10.2 二值哑变量与平行线模型

如果 f 是一个二值分类变量，将其中一个类编码为 1，另一个类编码为零。例如，$f=1$ 表示处理组，$f=0$ 表示对照组。这样编码后二值分类变量可以直接用在回归模型中。

例如

$$Ey = \beta_0 + \beta_1 x + \beta_2 f$$

相当于

$$Ey = \begin{cases} \beta_0 + \beta_1 x, & \text{对照组} \\ (\beta_0 + \beta_2) + \beta_1 x, & \text{处理组.} \end{cases}$$

这样的模型叫做平行线模型，处理组的数据与对照组的数据服从斜率相同、截距不同的一元线性回归模型。比每个组单独建模更有效。在 R 建模函数（如 `lm()`）中将这样的公式写成 `y ~ x + f`，这里 `f` 是一个因子的主效应，对于二值因子，用模型截距项（β_0）表示对应于水平 0 的截距，用 `f` 的系数（β_2）加模型截距项表示对应于水平 1 的截距。

如果检验：

$$H_0 : \beta_2 = 0$$

则不显著时，可以认为在处理组和对照组中 y 与 x 的关系没有显著差异。

进一步考虑

$$Ey = \beta_0 + \beta_1 x + \beta_2 f + \beta_3 fx$$

相当于

$$Ey = \begin{cases} \beta_0 + \beta_1 x, & \text{对照组} \\ (\beta_0 + \beta_2) + (\beta_1 + \beta_3)x, & \text{处理组} \end{cases}$$

比每个组单独建立回归模型更有效。在 R 建模函数中这个模型可以写成 `y ~ x + f + x:f` 或者 `y ~ x*f`，其中 `x` 对应于系数 β_1, `f` 对应于系数 β_2，`f:x` 对应于系数 β_3。

可以检验

$$H_0 : \beta_3 = 0$$

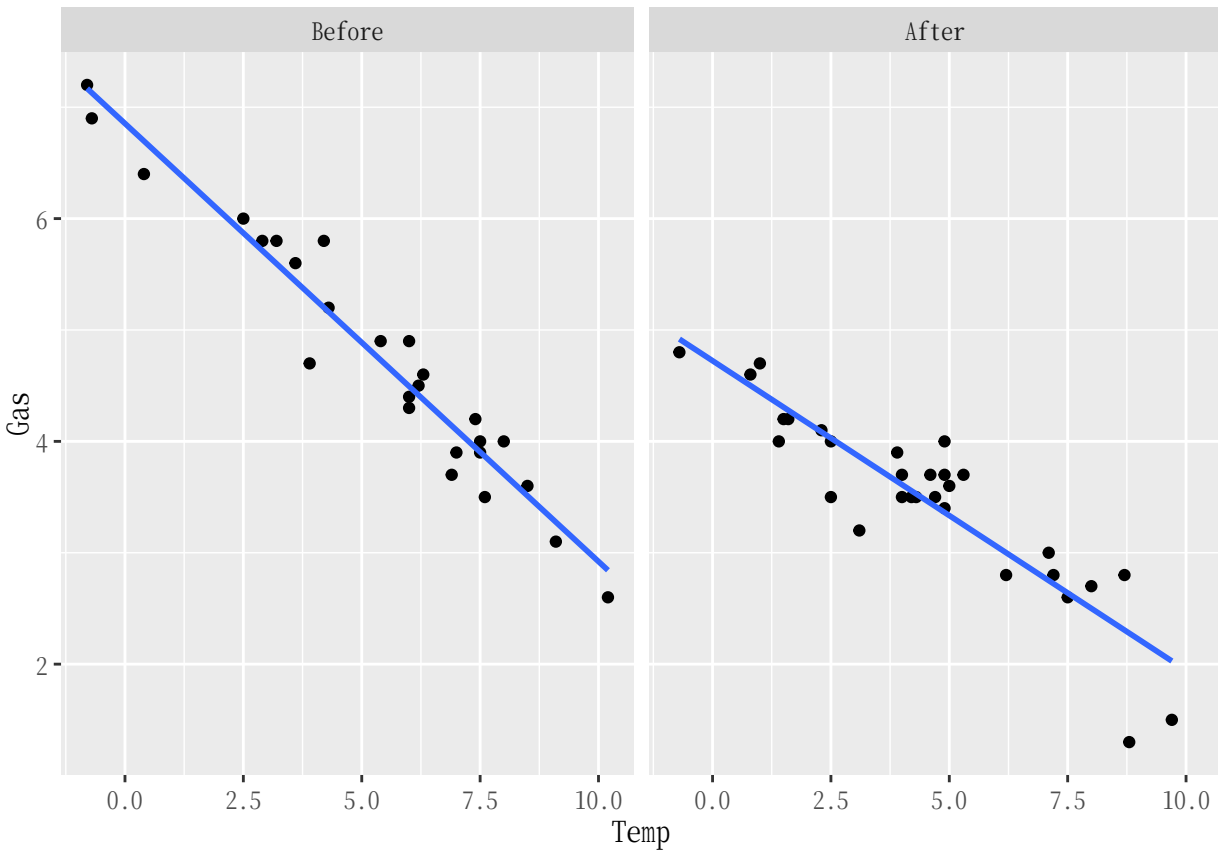
不显著时可以用平行线模型。

例 32.5. 考虑 MASS 包的 whiteside 数据。

因变量 `Gas` 是天然气用量，自变量包括一个因子 `Insul`，表示是否安装了隔热层 (Before 和 After)，一个连续型变量 `Temp` 为室外温度。按是否安装了隔热层分别作图并显示线性回归拟合线：

```
data(whiteside, package="MASS")
ggplot(data=whiteside, mapping = aes(
  x = Temp, y = Gas )) +
  geom_point() +
  geom_smooth(method="lm", se=FALSE) +
  facet_wrap(~ Insul, ncol=2)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



从图上看，两组的截距项有明显差距，斜率的差距不一定显著。

拟合平行线模型：

```
lm.gas1 <- lm(Gas ~ Temp + Insul,
  data = whiteside)
summary(lm.gas1)
```

```
##
## Call:
## lm(formula = Gas ~ Temp + Insul, data = whiteside)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.74236 -0.22291  0.04338  0.24377  0.74314
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.55133    0.11809   55.48   <2e-16 ***
## Temp        -0.33670    0.01776  -18.95   <2e-16 ***
```

```
## InsulAfter  -1.56520    0.09705  -16.13   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3574 on 53 degrees of freedom
## Multiple R-squared:  0.9097, Adjusted R-squared:  0.9063
## F-statistic: 267.1 on 2 and 53 DF,  p-value: < 2.2e-16
```

以 “Before” 为基线，所以 Before 组的截距为 6.55，After 组的截距为 $6.55 - 1.57$。

设定两组分别有不同斜率：

```r
lm.gas2 <- lm(Gas ~ Temp + Insul + Insul:Temp,
  data = whiteside)
summary(lm.gas2)
```

```
##
## Call:
## lm(formula = Gas ~ Temp + Insul + Insul:Temp, data = whiteside)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.97802 -0.18011  0.03757  0.20930  0.63803
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      6.85383    0.13596  50.409  < 2e-16 ***
## Temp            -0.39324    0.02249 -17.487  < 2e-16 ***
## InsulAfter      -2.12998    0.18009 -11.827 2.32e-16 ***
## Temp:InsulAfter  0.11530    0.03211   3.591 0.000731 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.323 on 52 degrees of freedom
## Multiple R-squared:  0.9277, Adjusted R-squared:  0.9235
## F-statistic: 222.3 on 3 and 52 DF,  p-value: < 2.2e-16
```

交叉项也显著，说明应该使用不同斜率项的模型。

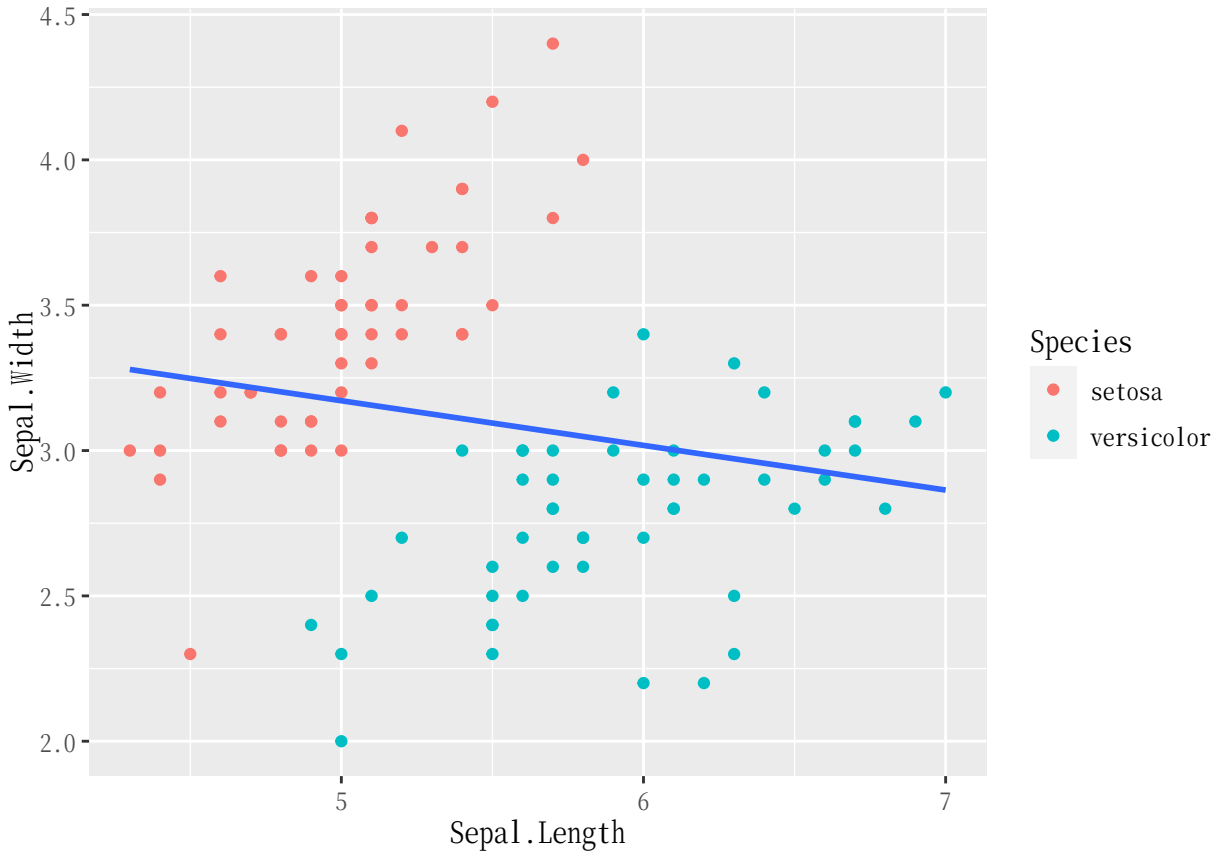
有些例子中如果忽略了分类变量，结论可能是错误的。

例 32.6. 考察 iris 数据中花萼长宽之间的关系。数据中有三个品种的花，仅考虑其中的 setosa 和 versicolor 两个品种。

下面的程序做了散点图和回归直线。

```
d.iris2 <- iris |>
  select(Sepal.Length, Sepal.Width, Species) |>
  filter(Species %in% c("setosa", "versicolor")) |>
  mutate(Species = factor(Species, levels=c("setosa", "versicolor")))
ggplot(data=d.iris2, mapping = aes(
  x = Sepal.Length, y = Sepal.Width )) +
  geom_point(mapping = aes(color = Species)) +
  geom_smooth(method="lm", se=FALSE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



从图形看，回归结果花萼长、宽是负相关的，这明显不合理，出现了“悖论”。实际是因为两个品种的样本混杂在一起了。这个回归的程序和结果如下：

```
lm.iris1 <- lm(Sepal.Width ~ Sepal.Length, data=d.iris2)
summary(lm.iris1)
```

```
##
```

```
## Call:
## lm(formula = Sepal.Width ~ Sepal.Length, data = d.iris2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.17136 -0.26382 -0.04468  0.29966  1.33618
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   3.93951    0.40621   9.698 5.47e-16 ***
## Sepal.Length -0.15363    0.07375  -2.083   0.0398 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4709 on 98 degrees of freedom
## Multiple R-squared:  0.04241,    Adjusted R-squared:  0.03263
## F-statistic:  4.34 on 1 and 98 DF,  p-value: 0.03984
```

回归结果在 0.05 水平下显著。这也给出了一个例子，即回归结果检验显著，并不能说明模型就是合理的。

我们采用平行线模型，加入 Species 分类变量作为回归自变量：

```
lm.iris2 <- lm(Sepal.Width ~ Sepal.Length + Species,
  data=d.iris2)
summary(lm.iris2)
```

```
##
## Call:
## lm(formula = Sepal.Width ~ Sepal.Length + Species, data = d.iris2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.88917 -0.14677 -0.02517  0.16643  0.64444
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        1.06519    0.32272   3.301  0.00135 **
## Sepal.Length       0.47200    0.06398   7.377 5.51e-11 ***
## Speciesversicolor -1.09696    0.08170 -13.427  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.2799 on 97 degrees of freedom
## Multiple R-squared:  0.665,  Adjusted R-squared:  0.6581
## F-statistic: 96.28 on 2 and 97 DF,  p-value: < 2.2e-16
```

现在花萼长度变量的系数为正而且高度显著。两个品种区别的变量 `Speciesversicolor` 表示 versicolor 品种取 1，setosa 取 0 的哑变量。versicolor 品种的花萼宽度与 setosa 相比在长度相等的情况下平均较小。

考虑斜率也不同的模型：

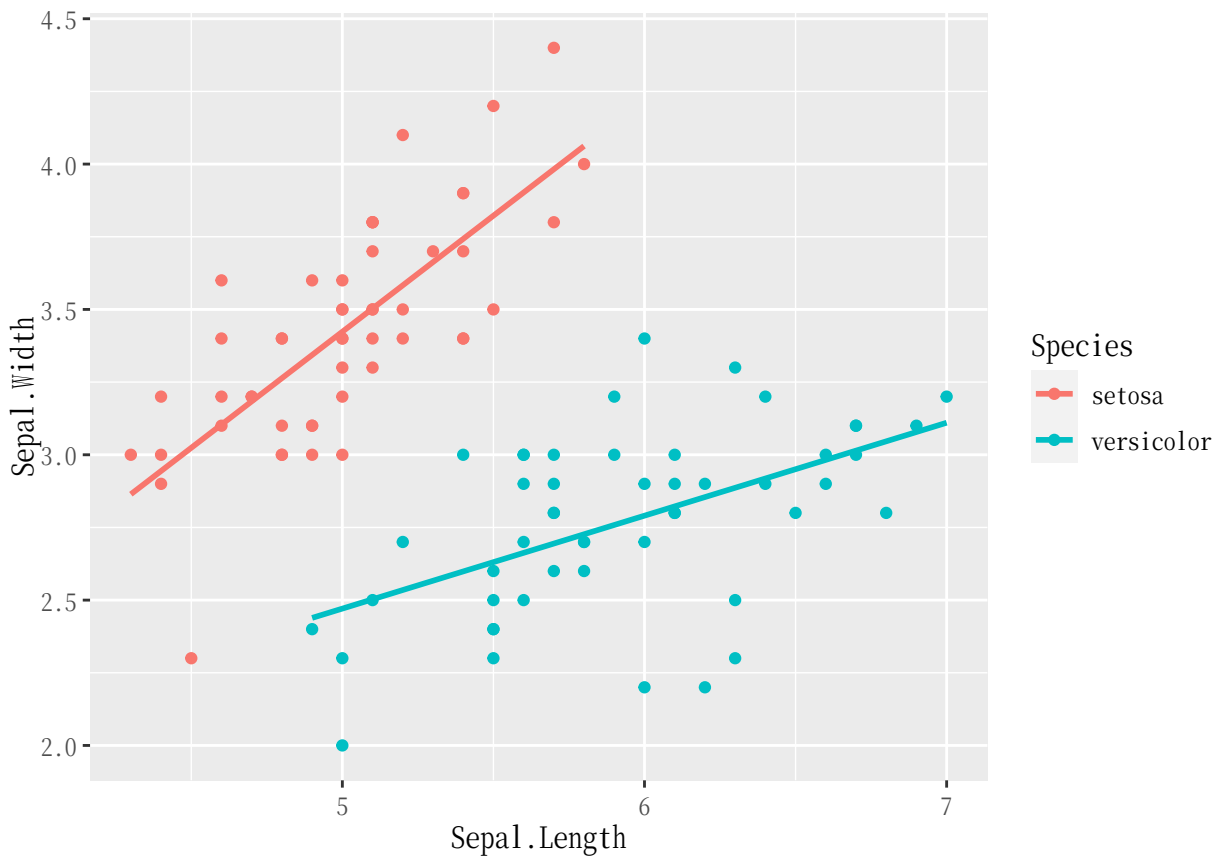
```r
lm.iris3 <- lm(Sepal.Width
  ~ Sepal.Length + Species + Species:Sepal.Length,
  data=d.iris2)
summary(lm.iris3)
```

```
##
## Call:
## lm(formula = Sepal.Width ~ Sepal.Length + Species + Species:Sepal.Length,
##     data = d.iris2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.72394 -0.16281 -0.00306  0.15936  0.60954
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     -0.5694     0.5352  -1.064 0.290049
## Sepal.Length                     0.7985     0.1067   7.487 3.41e-11 ***
## Speciesversicolor                1.4416     0.6891   2.092 0.039069 *
## Sepal.Length:Speciesversicolor  -0.4788     0.1292  -3.707 0.000351 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2632 on 96 degrees of freedom
## Multiple R-squared:  0.707,  Adjusted R-squared:  0.6978
## F-statistic:  77.2 on 3 and 96 DF,  p-value: < 2.2e-16
```

结果中交互作用项 `Speciesversicolor:Sepal.Length` 是显著的，这提示两组的斜率不同。下面的图形是对两个品种分别拟合一条回归直线的结果。

```
ggplot(data=d.iris2, mapping = aes(
  x = Sepal.Length, y = Sepal.Width, color = Species)) +
  geom_point() +
  geom_smooth(method="lm", se=FALSE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



32.10.3 多值哑变量

如果变量 f 是一个有 k 分类的变量，可以将 f 变成 $k-1$ 个取 0、1 值的哑变量 $z_1, z_2, \dots, z_{k-1}$。

例如，f 取三种不同的类别，可以转换为两个哑变量 z_1, z_2, 常用的编码方式为：

$$
\begin{aligned}
f = 1 &\Leftrightarrow (z_1, z_2) = (0, 0), \\
f = 2 &\Leftrightarrow (z_1, z_2) = (1, 0), \\
f = 3 &\Leftrightarrow (z_1, z_2) = (0, 1)
\end{aligned}
$$

这是将 $f = 1$ 看成对照组。

在 R 中，用 `model.matrix(y ~ x1 + f, data=d)` 可以生成将因子 `f` 转换成哑变量后的包含因变量和自变量的矩阵。详见32.18。

注意，使用多值的分类变量作为自变量时，一定要显式地将其转换成因子类型再加入到公式中，否则用数值表示的分类变量会被当作普通连续型自变量。

例 32.7. 以 iris 数据为例，用 Petal.Width 为因变量，Petal.Lenth 和 Species 为自变量，建立有三个分组的平行线模型。

```
lm.iris <- lm(Petal.Width ~ Species + Petal.Length,
  data = iris)
summary(lm.iris)
```

```
##
## Call:
## lm(formula = Petal.Width ~ Species + Petal.Length, data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.63706 -0.07779 -0.01218  0.09829  0.47814
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -0.09083    0.05639  -1.611    0.109
## Speciesversicolor  0.43537    0.10282   4.234 4.04e-05 ***
## Speciesvirginica   0.83771    0.14533   5.764 4.71e-08 ***
## Petal.Length       0.23039    0.03443   6.691 4.41e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1796 on 146 degrees of freedom
## Multiple R-squared:  0.9456, Adjusted R-squared:  0.9445
## F-statistic: 845.5 on 3 and 146 DF,  p-value: < 2.2e-16
```

这里分类变量 Species 有 3 个水平，用两个哑变量 `Speciesversicolor` 和 `Speciesvirginica` 表示，这两个变量分别是类别 versicolor 和 virginica 的示性函数值。关于哑变量检验依赖于所使用的对照，这里采用的是默认的 `contr.treatment()` 对照，以 3 个水平中第一个水平 setosa 为基线，`summary()` 关于哑变量 `Speciesversicolor` 的系数的检验对应与零假设 “versicolor 和 setosa 的模型截距项相同”，关于关于哑变量 `Speciesvirginica` 的系数的检验对应与零假设 “virginica 和 setosa 的模型截距项相同”。为了检验分类变量 Species 的主效应在模型中是否显著，应该使用 `anova()` 函数：

```
anova(lm.iris)
```

```
## Analysis of Variance Table
```

```
##
## Response: Petal.Width
##               Df Sum Sq Mean Sq  F value    Pr(>F)
## Species        2 80.413  40.207 1245.887 < 2.2e-16 ***
## Petal.Length   1  1.445   1.445   44.775 4.409e-10 ***
## Residuals    146  4.712   0.032
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

这个结果就显示了 Species 的主效应在模型中有显著影响。

`anova()` 进行 F 检验，对因子和包含因子的交互作用效应检验效应的显著性而非单个哑变量的显著性；另外，`anova()` 进行的是序贯的检验称为第一类检验，比如，上面结果关于 `Species` 的结果是包含自变量 `Species` 的模型与仅有截距项的模型比较的 F 检验，而关于 `Petal.Length` 的结果是包含自变量 `Species` 和 `Petal.Length` 的模型与仅包含自变量 `Species` 的模型比较的 F 检验。

`summary()` 函数结果中的 t 检验则是进行的第三类检验，这是包含某个变量（或哑变量）的模型与去掉此变量的模型的比较。

也可以利用下面比较嵌套模型的方法，参见32.11。

32.11 嵌套模型的比较

如果两个多元线性回归模型，第一个模型中的自变量都在第二个模型中，且第二个模型具有更多的自变量，可以通过对第二个模型的参数施加约束（如某些斜率项等于零）变成第一个模型，则称第一个模型嵌套在第二个模型中。

例如：第一个模型中自变量为 x_1, x_2, x_5, 第二个模型自变量为 x_1, x_2, x_3, x_4, x_5，则第一个模型嵌套在第二个模型中。

又如：第一个模型自变量为 x_1, x_2, 第二个模型自变量为 $x_1, x_2, x_1^2, x_2^2, x_1x_2$，则第一个模型嵌套在第二个模型中。这时令 $x_3 = x_1^2$, $x_4 = x_2^2$, $x_5 = x_1x_2$，第二个模型变成有 5 个自变量的多元线性回归模型。

在嵌套的模型中，相对而言自变量多的模型叫做完全模型（full model），自变量少的模型叫做简化模型（reduced model）。

完全模型是否比简化模型更好？在回归模型选择中贯彻一个 “精简性” 原则（Ocam’s razor）：在对建模数据拟合效果相近的情况下，越简单的模型越好。

例如：全模型是

$$Ey = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4,$$

精简模型是

$$Ey = \beta_0 + \beta_1 x_1 + \beta_2 x_2,$$

判断两个模型是否没有显著差异，只要检验：

$$H_0 : \beta_3 = \beta_4 = 0,$$

这可以构造一个方差分析 F 检验，

设全模型有 t 个自变量，模型得到的回归残差平方和为 SSE_f; 设精简模型有 s 个自变量（$s < t$），模型得到的回归残差平方和为 SSE_r；检验零假设为多出的自变量对应的系数都等于零。检验统计量为

$$F = \frac{(\mathrm{SSE}_r - \mathrm{SSE}_f)/(t-s)}{\mathrm{SSE}_f/(n-t-1)}$$

在全模型成立且 H_0 成立时，F 服从 $F(t-s, n-t-1)$ 分布。计算右侧 p 值。

在 R 中用 `anova()` 函数比较两个嵌套的线性回归结果可以进行这样的方差分析 F 检验。

例 32.8. 餐馆营业额回归模型的比较。

`lmrst01` 是完全模型，包含 5 个自变量；`lmrst02` 是嵌套的精简模型，包含居民数、人均餐费、到市中心距离共 3 个自变量。用 `anova()` 函数可以检验多出的变量是否有意义：

```
anova(lmrst01, lmrst02)
```

```
## Analysis of Variance Table
##
## Model 1: 营业额 ~ 居民数 + 人均餐费 + 月收入 + 餐馆数 +
##     距离
## Model 2: 营业额 ~ 居民数 + 人均餐费 + 距离
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     19 2153.0
## 2     21 2267.2 -2   -114.17 0.5038 0.6121
```

模型 p 值为 0.61，在 0.05 水平下不拒绝多出的月收入和餐馆数的系数全为零的零假设，两个模型的效果没有显著差异，应选择更精简的模型。

32.12 非嵌套模型的比较

方差分析检验仅能比较嵌套模型。对不同模型计算 AIC，取 AIC 较小的模型，这可以对非嵌套的模型进行比较选择。R 中用 `AIC()` 函数比较两个回归结果的 AIC 值。

如：

```
AIC(lmrst01, lmrst02)
```

```
##         df      AIC
## lmrst01  7 196.3408
## lmrst02  5 193.6325
```

精简模型 `lmrst02` 的 AIC 更小，是更好的模型。

AIC 和 BIC 基于最大似然估计的似然函数值，所以最好使用最大似然估计，对线性模型，系数的最大似然估计与最小二乘估计相同，但是方差的最大似然估计与最小二乘估计不同。目前 `lm()` 仅支持最小二乘估计。

32.13 参数线性组合的检验

$H_0: \beta_j = 0$ 这样关于系数的检验，以及 $H_0: \beta_1 = \cdots = \beta_p = 0$ 这样的检验，是关于回归系数线性组合的检验的特例。

32.13.1 可估性、对照

对一般的线性模型

$$Y = X\beta + \varepsilon,$$

其中 $\varepsilon \sim \mathrm{N}(0, \sigma^2 I)$，$X$ 称为设计阵，β 为回归系数。如果设计阵 $X_{n\times p}$ 列满秩，则

$$\widehat{\beta} = (X^T X)^{-1} X^T Y$$

是 β 的无偏估计，$\mathrm{Var}(\widehat{\beta}) = \sigma^2 (X^T X)^{-1}$，对任意 $c \neq 0 \in \mathbb{R}^p$，都有 $E(c^T \widehat{\beta}) = c^T \beta$。

如果存在 $a \in \mathbb{R}^n$ 使得 $E(a^T Y) = c^T \beta$, 则称 $c^T \beta$ 线性可估。当设计阵 X 列满秩时所有的 $c^T \beta$ 都是线性可估的。

但是，如果 X 非列满秩，则存在 $c \in \mathbb{R}^p$ 使得 $c^T \beta$ 没有无偏估计。事实上，只要 $c \neq 0$ 使得 $Xc = 0$，则 $c^T \beta$ 非线性可估。$c^T \beta$ 线性可估，当且仅当 c 等于设计阵 X 的行向量的线性组合，即 c 属于设计阵 X 的行向量张成的 $\mathbb{R}^p$ 的线性子空间。参见 (陈家鼎、孙山泽、李东风、刘力平, 2015) P.169 定理 3.3。

在利用 R 软件进行线性模型建模时，通常都保证了设计阵 X 列满秩，比如，当自变量中有分类变量时，K 个水平的分类变量一般都用 $K-1$ 个取 0、1 值的哑变量来表示，这样设计阵保持列满秩。设 $c \in \mathbb{R}^p$ 使得 $1^T c = 0$，即 $c_1 + \cdots + c_p = 0$，称 c 为参数的一个**对照**向量。考虑检验问题

$$H_0: c_1\beta_1 + \cdots + c_p\beta_p = a.$$

32.13.2 t 检验法和嵌套模型方法

设 c 是一个对照，零假设

$$H_0: c^T \beta = a$$

可以用如下的 H_0 下服从 t 分布的统计量检验：

$$t = \frac{c^T \widehat{\beta} - a}{\widehat{\sigma}\sqrt{c^T (X^T X)^{-1} c}}.$$

其中 $\widehat{\sigma} = \sqrt{\mathrm{MSE}}$。R 的 multcomp 包的 lhgt 函数提供了这样的检验。也可以建立满足约束条件 $c^T \beta = a$ 的模型，用 `anova()` 函数比较两个模型。下面用模拟数据说明。

例 32.9. 对模型

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon,$$

考虑检验

$$H_0 : \beta_1 = \beta_2.$$

解答： 令 $\beta = (\beta_0, \beta_1, \beta_2)^T$, $c = (0, 1, -1)^T$，H_0 可以写成 $c^T\beta = 0$。我们产生一个模拟数据，检验这样的假设。

```
library(multcomp, quietly = TRUE, warn.conflicts = FALSE)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

##
## Attaching package: 'TH.data'

## The following object is masked from 'package:MASS':
##
##     geyser
```

```
sim.lt <- function(n=30,
    betas = c(100, 2, 2)){
  x1 <- sample(1:20, size=n, replace=TRUE)
  x2 <- sample(1:20, size=n, replace=TRUE)
  eps <- rnorm(n, 0, 10)

  y <- betas[1] + betas[2]*x1 + betas[3]*x2 + eps
  data.frame(x1=x1, x2=x2, y=y)
}
set.seed(101)
d.lt <- sim.lt(betas = c(100, 2, 2))
lm.lt1 <- lm(y ~ x1 + x2, data=d.lt)
summary(lm.lt1)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2, data = d.lt)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.5569  -6.1377  -0.7288   5.1499  18.6338
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  99.7713     4.9133  20.306  < 2e-16 ***
## x1            1.9394     0.3188   6.084 1.70e-06 ***
## x2            2.1445     0.3054   7.021 1.51e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.633 on 27 degrees of freedom
## Multiple R-squared:  0.7565, Adjusted R-squared:  0.7384
## F-statistic: 41.94 on 2 and 27 DF,  p-value: 5.227e-09
```

上面产生的模拟数据中设定了 $\beta_1 = \beta_2$。用 multcomp 包的 lhgt 函数检验 $\beta_1 = \beta_2$:

```
multcomp::glht(lm.lt1,
  linfct = rbind(
    c(0, 1, -1) ) ) |> summary()
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: lm(formula = y ~ x1 + x2, data = d.lt)
##
## Linear Hypotheses:
##        Estimate Std. Error t value Pr(>|t|)
## 1 == 0  -0.2051     0.4350  -0.471    0.641
## (Adjusted p values reported -- single-step method)
```

结果不显著，不拒绝零假设。

产生 $\beta_1 \neq \beta_2$ 的模拟样本进行检验:

```
d.ltb <- sim.lt(betas = c(100, 1, 2))
lm.lt2 <- lm(y ~ x1 + x2, data=d.ltb)
summary(lm.lt2)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2, data = d.ltb)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -22.104  -7.204   1.210   5.866  20.987
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  98.5311     4.8156  20.461  < 2e-16 ***
## x1            0.8861     0.3435   2.579   0.0157 *
## x2            2.2535     0.3254   6.925 1.93e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.15 on 27 degrees of freedom
## Multiple R-squared:  0.7081, Adjusted R-squared:  0.6865
## F-statistic: 32.75 on 2 and 27 DF,  p-value: 6.026e-08
```

```r
multcomp::glht(lm.lt2,
  linfct = rbind(
    c(0, 1, -1) ) ) |> summary()
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: lm(formula = y ~ x1 + x2, data = d.ltb)
##
## Linear Hypotheses:
##        Estimate Std. Error t value Pr(>|t|)
## 1 == 0  -1.3674     0.5222  -2.619   0.0143 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

结果显著。

这样的检验问题也可以利用建立嵌套模型解决。在 $\beta_1 = \beta_2$ 约束下，模型变成

$$y = \beta_0 + \beta_1(x_1 + x_2) + \varepsilon,$$

只要用 anova() 比较两个模型。

$\beta_1 = \beta_2$ 的模拟数据的检验：

```
lm.lt1b <- lm(y ~ I(x1 + x2), data=d.lt)
summary(lm.lt1b)
```

```
##
## Call:
## lm(formula = y ~ I(x1 + x2), data = d.lt)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -19.0341  -5.8075  -0.6999   5.6555  18.0947
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  99.7301     4.8438  20.589  < 2e-16 ***
## I(x1 + x2)    2.0464     0.2206   9.276 4.93e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.498 on 28 degrees of freedom
## Multiple R-squared:  0.7545, Adjusted R-squared:  0.7457
## F-statistic: 86.04 on 1 and 28 DF,  p-value: 4.926e-10
```

```
anova(lm.lt1b, lm.lt1)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ I(x1 + x2)
## Model 2: y ~ x1 + x2
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     28 2526.1
## 2     27 2505.5  1    20.624 0.2223 0.6411
```

结果不显著。

对 $\beta_1 \neq \beta_2$ 的模拟数据检验：

```r
lm.lt2b <- lm(y ~ I(x1 + x2), data=d.ltb)
summary(lm.lt2b)
```

```
##
## Call:
## lm(formula = y ~ I(x1 + x2), data = d.ltb)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -29.5424  -8.1704  -0.0426   6.2143  22.8775
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  97.5681     5.2800  18.479  < 2e-16 ***
## I(x1 + x2)    1.6002     0.2298   6.964 1.43e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.16 on 28 degrees of freedom
## Multiple R-squared:  0.634,  Adjusted R-squared:  0.6209
## F-statistic:  48.5 on 1 and 28 DF,  p-value: 1.426e-07
```

```r
anova(lm.lt2b, lm.lt2)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ I(x1 + x2)
## Model 2: y ~ x1 + x2
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     28 3486.8
## 2     27 2780.6  1    706.23 6.8576 0.0143 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

两个模型有显著差异。注意尽管 $\beta_1 = \beta_2$ 不成立，但是在 $\beta_1 = \beta_2$ 约束下建立的模型也显著，这提示我们，仅从回归模型结果显著不能说明模型就是正确的。

32.13.3 多个对照的检验

设矩阵 $H_{k\times p}$ 是由对照行向量组成的矩阵，行秩为 k，考虑检验

$$H_0 : H\beta = a. \tag{32.1}$$

取 H 为 I_p 的第 j 行且 $a=0$ 时，就是 $H_0:\beta_j=0$ 的检验；取 H 为 I_p 且 $a=0$ 时，就是 $H_0:\beta_1=\cdots=\beta_p=0$ 的检验。

可以在约束(32.1)下建立约束模型，并与无约束的模型用 F 检验进行比较。如果要检验 $H_0:H\beta=0$（即上面的 $a=0$），F 统计量为

$$F=\frac{\hat{\beta}^T H^T[H(X^TX)^{-1}H^T]^{-1}H\hat{\beta}}{\text{MSE}} \tag{32.2}$$

其中

$$\text{MSE}=\frac{1}{n-p}(Y-X\hat{\beta})^T(Y-X\hat{\beta})=\frac{1}{n-p}\sum_{i=1}^{n}(y_i-\hat{\beta}_1x_{i1}-\cdots-\hat{\beta}_px_{ip})^2.$$

p 是设计阵 X 的列数，在所有自变量都是普通连续变量时，p 等于自变量个数加 1。参见 (陈家鼎、孙山泽、李东风、刘力平, 2015) P.195 式 (5.17)

如果要检验 $H_0:H\beta=a$，其中 H 是 $k>1$ 行的对照矩阵，则不能使用单独检验每个对照的方法，因为这样会产生多重比较问题，使得第一类错误概率放大。可以使用(32.2)这样的 F 检验。

在 R 中对参数线性组合进行检验时，一种办法是建立满足约束的模型，用比较嵌套模型的 `anova()` 函数进行检验。

关于因子自变量的效应之间的比较，可以采用适当的对照函数编码，参见32.19。

如果要对多个对照同时进行检验但希望给出每个检验的结果，可以使用 R 的 multcomp 包，该包使用多元 t 分布对(32.1)的每一行进行检验，并控制总错误率。因为控制了总错误率，所以只要所有的对照中至少一个有显著差异，就可以认为零假设 $H_0:H\beta=a$ 被否定。参见 (Bretz et al., 2011) 节 3.1。

例 32.10. 考虑例32.4餐馆营业额的问题。检验

$$H_0:\beta_3=\beta_4=0,$$

即周边居民月收入和附近其它餐馆数的效应。

可以用上面的嵌套模型的 `anova()` 函数检验，也可以用 multcomp 包的多元 t 分布方法进行检验：

```
multcomp::glht(lmrst01,
  linfct = rbind(
    c(0, 0, 1, 0, 0, 0),
    c(0, 0, 0, 1, 0, 0) )) |> summary()
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
```

```
##
## Fit: lm(formula = 营业额 ~ 居民数 + 人均餐费 + 月收入 +
##     餐馆数 + 距离, data = Resturant)
##
## Linear Hypotheses:
##         Estimate Std. Error t value Pr(>|t|)
## 1 == 0 0.1605660  0.0556834   2.884   0.0183 *
## 2 == 0 0.0007636  0.0013556   0.563   0.8112
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

程序中 `linfct` 输入了两个对照行向量，分别进行检验并控制总错误率。注意对照向量维数等于设计阵列数，而设计阵列数一般等于自变量个数加 1，其中第一列一般都是与截距项配合的一列 1。

32.14 拟合与预测

32.14.1 拟合

有了参数最小二乘估计后，对建模用的每个数据点计算

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip},$$

称为拟合值（fitted value）。

得到回归模型结果 lmres 后，要对原数据框中的观测值做预测，只要使用 `predict(lmres)`。

32.14.2 点预测

为了使用得到的模型结果 lmres 对新数据做预测，建立包含了自变量的一组新的观测值的数据框 dp，用 `predict(lmres, newdata=dp)` 做预测。

如餐馆营业额的选择自变量的回归模型 `lmrst02` 的拟合值：

```
predict(lmrst02)
```

```
##          1          2          3          4          5          6          7
## 52.1892541 -4.5010247 21.9626575 65.1136964  6.1284803 22.4083426  1.2783244
##          8          9         10         11         12         13         14
## 34.7189934 10.6275869 37.8433996 62.8524888 18.2959990 -5.5101343 14.9558131
##         15         16         17         18         19         20         21
```

```
##  8.8598588 29.8309866 78.0159456 13.1673581 15.8469287 50.7274217 27.4608977
##         22         23         24         25
##  0.2331759 22.6274030 48.9610796 27.0050675
```

如果是一元回归，一般还画数据的散点图并画回归直线。多元回归的图形无法在二维表现出来。

有了估计的回归方程后，对一组新的自变量值 $(x_{01}, \dots, x_{0p})$，可以计算对应的因变量的预测值：

$$\hat{y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_{01} + \cdots + \hat{\beta}_p x_{0p}$$

在 R 中，设 `lmres` 保存了回归结果，`newd` 是一个保存了新的自变量值的数据框，此数据框结构与原建模用数据框类似但是自变量与原来不同，且不需要有因变量。这时用 `predict(lm1, data=newd)` 预测。

例如，利用包含居民数、人均餐费、到市中心距离的模型 `lmrst02`，求居民数 =50(万居民)，人均餐费 =100(元)，距市中心 10 千米的餐馆的日均营业额：

```
predict(
  lmrst02,
  newdata=data.frame(
    `居民数`=50, `人均餐费`=100, `距离`=10
  ))
```

```
##        1
## 17.88685
```

预测的日均营业额为 17.9 千元。

函数 `expand.grid()` 可以对若干个变量的指定值，生成包含所有组合的数据框，如：

```
newd <- expand.grid(
  `居民数`=c(60, 140),
  `人均餐费`=c(50, 130),
  `距离`=c(6, 16))
newd
```

```
##   居民数 人均餐费 距离
## 1     60       50    6
## 2    140       50    6
## 3     60      130    6
## 4    140      130    6
## 5     60       50   16
## 6    140       50   16
## 7     60      130   16
## 8    140      130   16
```

```
predict(lmrst02, newdata=newd)
```

```
##         1         2         3         4         5         6         7         8
## 14.186636 29.404103 26.797108 42.014574  8.488759 23.706225 21.099230 36.316696
```

32.14.3 均值的置信区间

对 $Ey = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$, 可以计算置信度为 $1-\alpha$ 的置信区间，称为均值的置信区间。

在 `predict()` 中加选项 `interval="confidence"`, 用 `level=` 指定置信度，可以计算均值的置信区间。

如

```
predict(
  lmrst02, interval="confidence", level=0.95,
  newdata=data.frame(
    `居民数`=50, `人均餐费`=100, `距离`=10
  ))
```

```
##        fit      lwr      upr
## 1 17.88685 10.98784 24.78585
```

其中 `fit` 是预测值，`lwr` 和 `upr` 分别是置信下限和置信上限。

32.14.4 个别值的预测区间

对 $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \varepsilon$, 可以计算置信度为 $1-\alpha$ 的预测区间，称为预测区间，预测区间比均值的置信区间要宽。

在 `predict()` 中加选项 `interval="prediction"`, 用 `level=` 指定置信度，可以计算预测区间。

如

```
predict(
  lmrst02, interval="prediction", level=0.95,
  newdata=data.frame(
    `居民数`=50, `人均餐费`=100, `距离`=10
  ))
```

```
##        fit       lwr      upr
## 1 17.88685 -4.795935 40.56963
```

其中 `fit` 是预测值，`lwr` 和 `upr` 分别是预测下限和预测上限。

32.15 利用线性回归模型做曲线拟合

某些非线性关系可以通过对因变量和自变量的简单变换变成线性回归模型。

例 32.11. 彩色显影中，染料光学密度 Y 与析出银的光学密度 x 有如下类型的关系

$$Y \approx Ae^{-B/x}, \quad B > 0.$$

这不是线性关系。两边取对数得

$$\ln Y \approx \ln A - B\frac{1}{x}$$

令

$$Y^* = \ln Y, \qquad x^* = \frac{1}{x}$$

则

$$Y^* \approx \ln A - Bx^*$$

为线性关系。

从 n 组数据 $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ 得到变换的数据 $(x_1^*, y_1^*), (x_2^*, y_2^*), \dots, (x_n^*, y_n^*)$。对变换后的数据建立线性回归方程

$$\hat{y}^* = \hat{a} + \hat{b}x^*$$

反变换得

$$\hat{A} = e^{\hat{a}}, \qquad \hat{B} = -b$$

则有

$$\hat{Y} = \hat{A}e^{-\hat{B}/x}$$

例 32.12. 考虑一个钢包容积的例子。

炼钢钢包随使用次数增加而容积增大。测量了 13 组这样的数据:

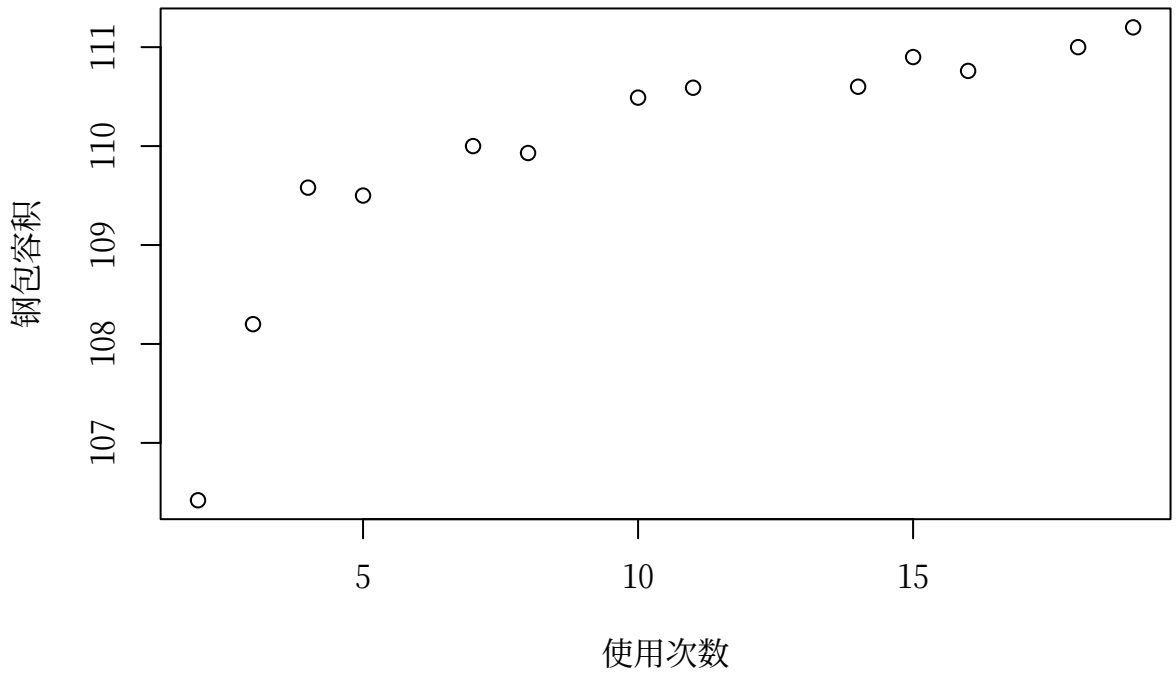
```
SteelBag <- data.frame(
  x = c(2, 3, 4, 5, 7, 8, 10,
        11, 14, 15, 16, 18, 19),
  y = c(106.42, 108.20, 109.58, 109.50, 110.0,
        109.93, 110.49, 110.59, 110.60, 110.90,
        110.76, 111.00, 111.20)
)
```

```
knitr::kable(SteelBag)
```

x	y
2	106.42
3	108.20
4	109.58
5	109.50
7	110.00
8	109.93
10	110.49
11	110.59
14	110.60
15	110.90
16	110.76
18	111.00
19	111.20

散点图：

```
with(SteelBag, plot(
  x, y, xlab=" 使用次数", ylab=" 钢包容积"
))
```



散点图呈现非线性。

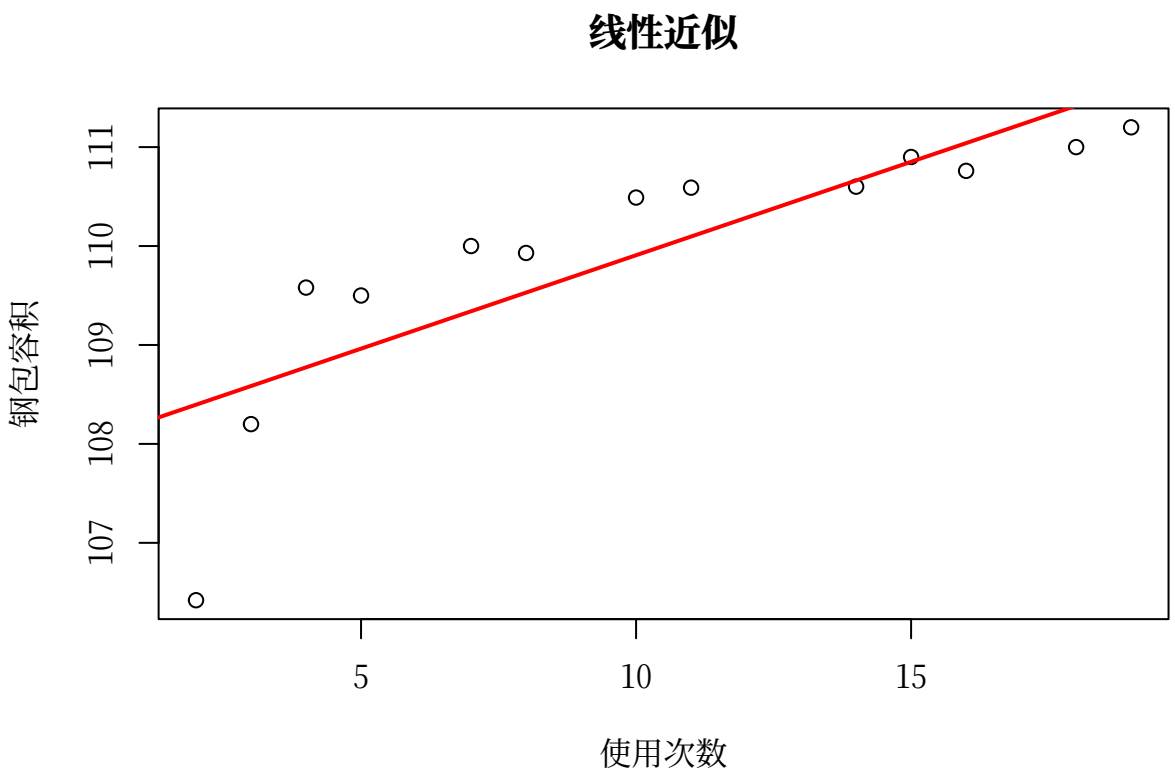
用线性回归近似：

```
lmsb1 <- lm(y ~ x, data=SteelBag)
summary(lmsb1)
```

```
##
## Call:
## lm(formula = y ~ x, data = SteelBag)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.97615 -0.38502  0.04856  0.53724  0.80611
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 108.01842    0.44389 243.346  < 2e-16 ***
## x             0.18887    0.03826   4.937 0.000445 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7744 on 11 degrees of freedom
## Multiple R-squared:  0.689,  Adjusted R-squared:  0.6607
## F-statistic: 24.37 on 1 and 11 DF,  p-value: 0.000445
```

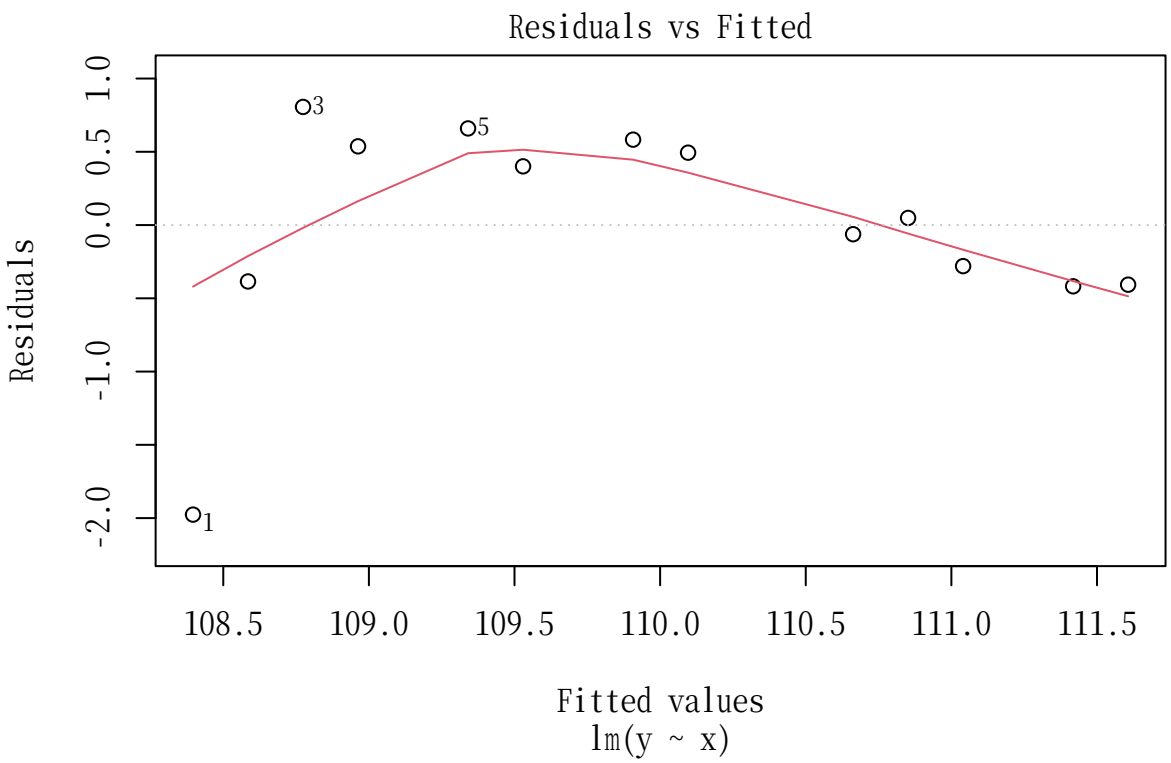
结果显著。$R^2 = 0.69$。拟合图：

```
with(SteelBag, plot(
  x, y, xlab=" 使用次数", ylab=" 钢包容积",
  main=" 线性近似"
))
abline(lmsb1, col="red", lwd=2)
```



残差诊断：

```
plot(lmsb1, which=1)
```



残差图呈现非线性。

用双曲线模型：

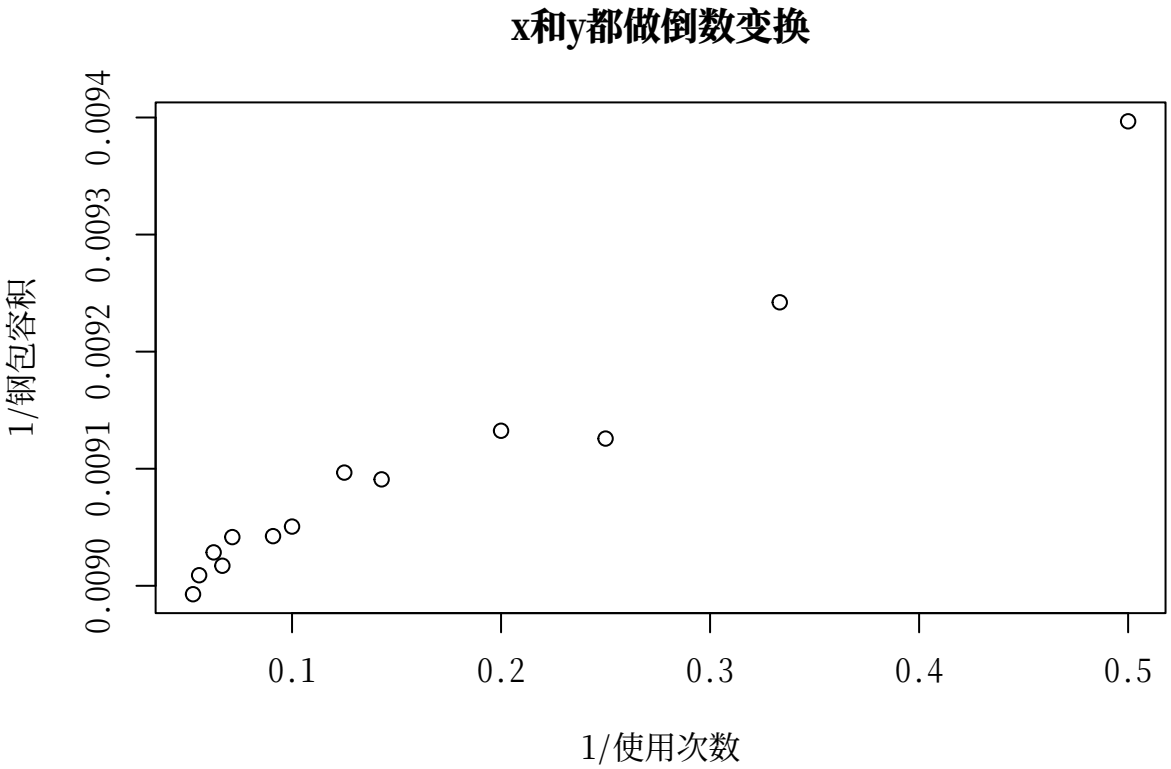
$$\frac{1}{y} \approx a + b\frac{1}{x}$$

令 $x^* = 1/x, y^* = 1/y$，化为线性模型

$$y^* \approx a + bx^*$$

(x^*, y^*) 的散点图：

```
with(SteelBag, plot(
  1/x, 1/y, xlab="1/使用次数", ylab="1/钢包容积",
  main="x 和 y 都做倒数变换"
))
```



y^* 对 x^* 的回归：

```
lmsb2 <- lm(I(1/y) ~ I(1/x), data=SteelBag)
summary(lmsb2)
```

```
##
## Call:
## lm(formula = I(1/y) ~ I(1/x), data = SteelBag)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -4.817e-05 -3.686e-06  4.000e-07  1.008e-05  2.642e-05
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 8.967e-03  8.371e-06 1071.14  < 2e-16 ***
## I(1/x)      8.292e-04  4.118e-05   20.14 4.97e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 1.903e-05 on 11 degrees of freedom
## Multiple R-squared:  0.9736, Adjusted R-squared:  0.9712
## F-statistic: 405.4 on 1 and 11 DF,  p-value: 4.97e-10
```

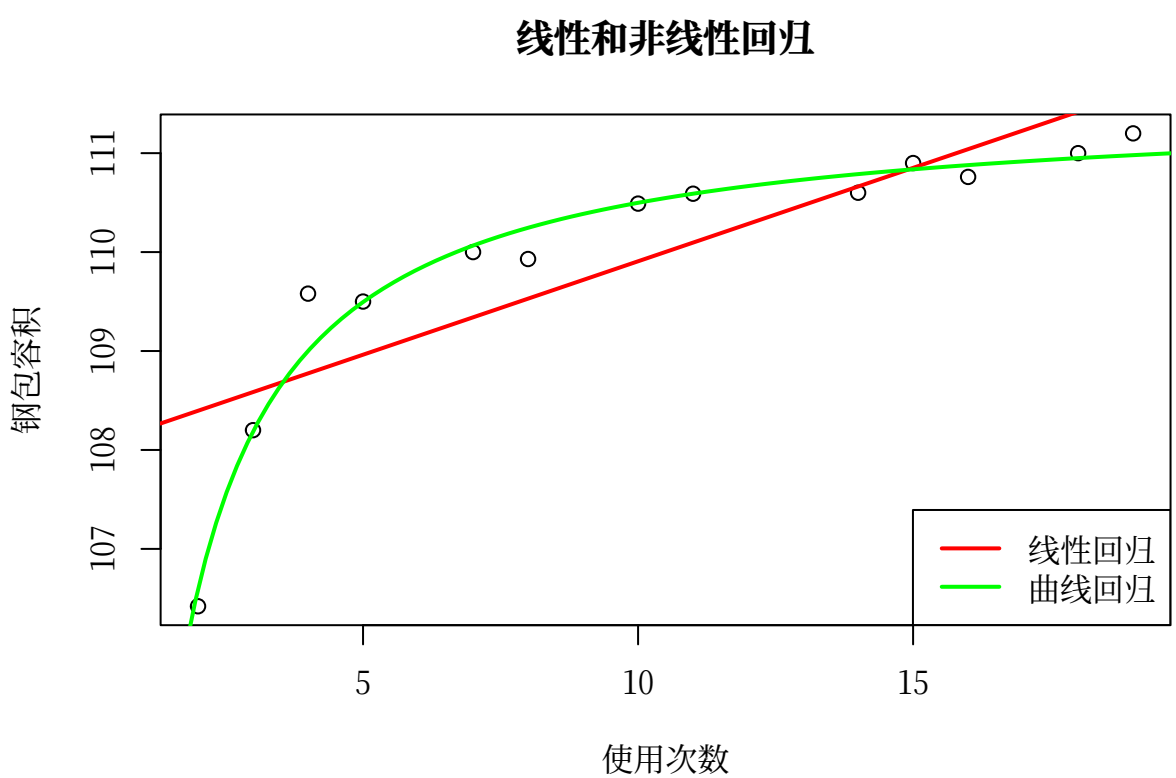
结果显著。解得 $\hat{a} = 0.008967$, $\hat{b} = 0.0008292$, 经验公式为

$$\frac{1}{\hat{y}} = 0.008967 + 0.0008292\frac{1}{x}$$

R^2 从线性近似的 0.69 提高到了 0.97。

拟合图:

```
with(SteelBag, plot(
  x, y, xlab=" 使用次数", ylab=" 钢包容积",
  main=" 线性和非线性回归"
))
abline(lmsb1, col="red", lwd=2)
curve(1/(0.008967 + 0.0008292/x), 1, 20,
      col="green", lwd=2, add=TRUE)
legend("bottomright", lty=1, lwd=2,
       col=c("red", "green"),
       legend=c(" 线性回归", " 曲线回归"))
```



例 32.13. 考虑 Reynolds, Inc. 销售业绩数据分析问题。

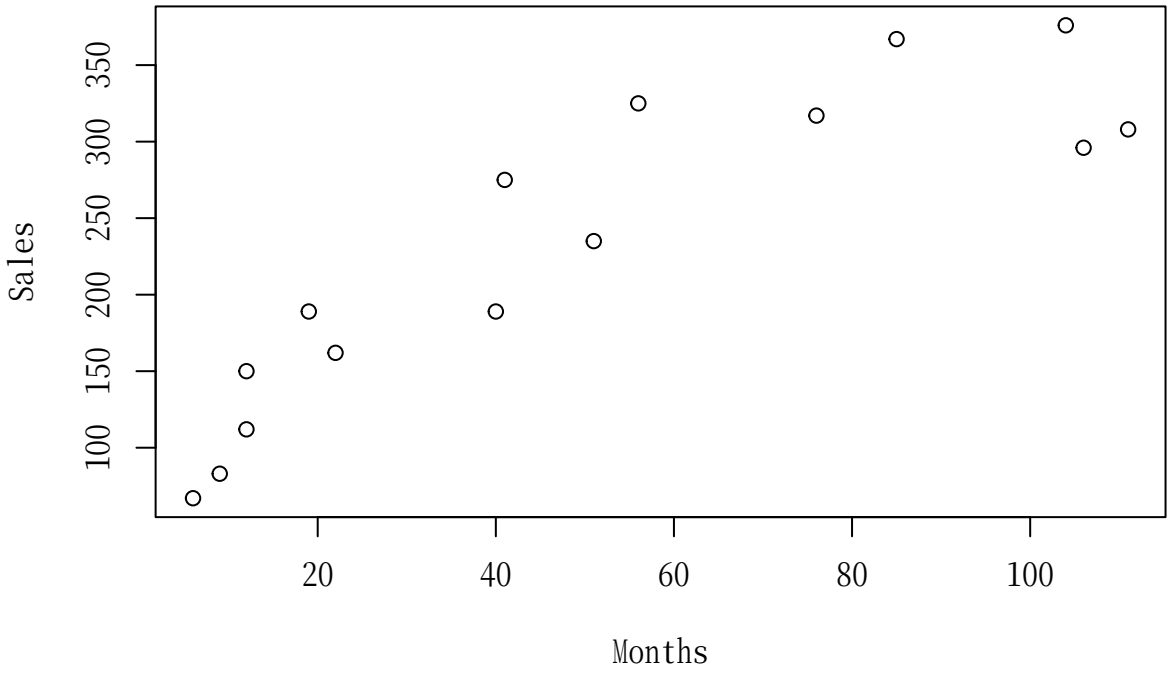
Reynolds, Inc. 是一个工业和试验室量具厂商。为研究销售业务员的业绩，考察业务员从业年限 (Months, 单位：月) 与其销售的电子量具数量 (Sales) 的关系。随机抽查了 15 名业务员。

```
knitr::kable(Reynolds)
```

Months	Sales
41	275
106	296
76	317
104	376
22	162
12	150
85	367
111	308
40	189
51	235
9	83
12	112
6	67
56	325
19	189

散点图：

```
with(Reynolds, plot(Months, Sales))
```



散点图呈现非线性。

用线性近似：

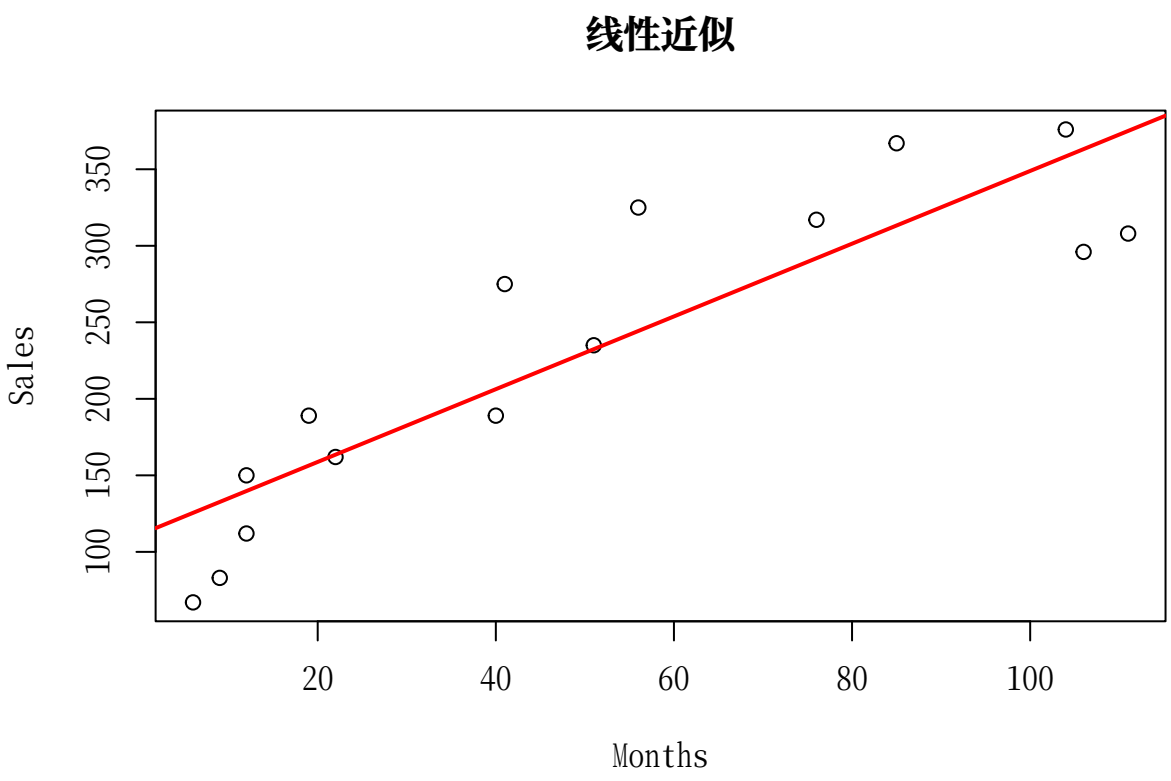
```
lmre1 <- lm(Sales ~ Months, data=Reynolds)
summary(lmre1)
```

```
##
## Call:
## lm(formula = Sales ~ Months, data = Reynolds)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -67.166 -38.684   2.557  28.875  80.673
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 111.2279    21.6280   5.143 0.000189 ***
## Months        2.3768     0.3489   6.812 1.24e-05 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 49.52 on 13 degrees of freedom
## Multiple R-squared:  0.7812, Adjusted R-squared:  0.7643
## F-statistic: 46.41 on 1 and 13 DF,  p-value: 1.239e-05
```

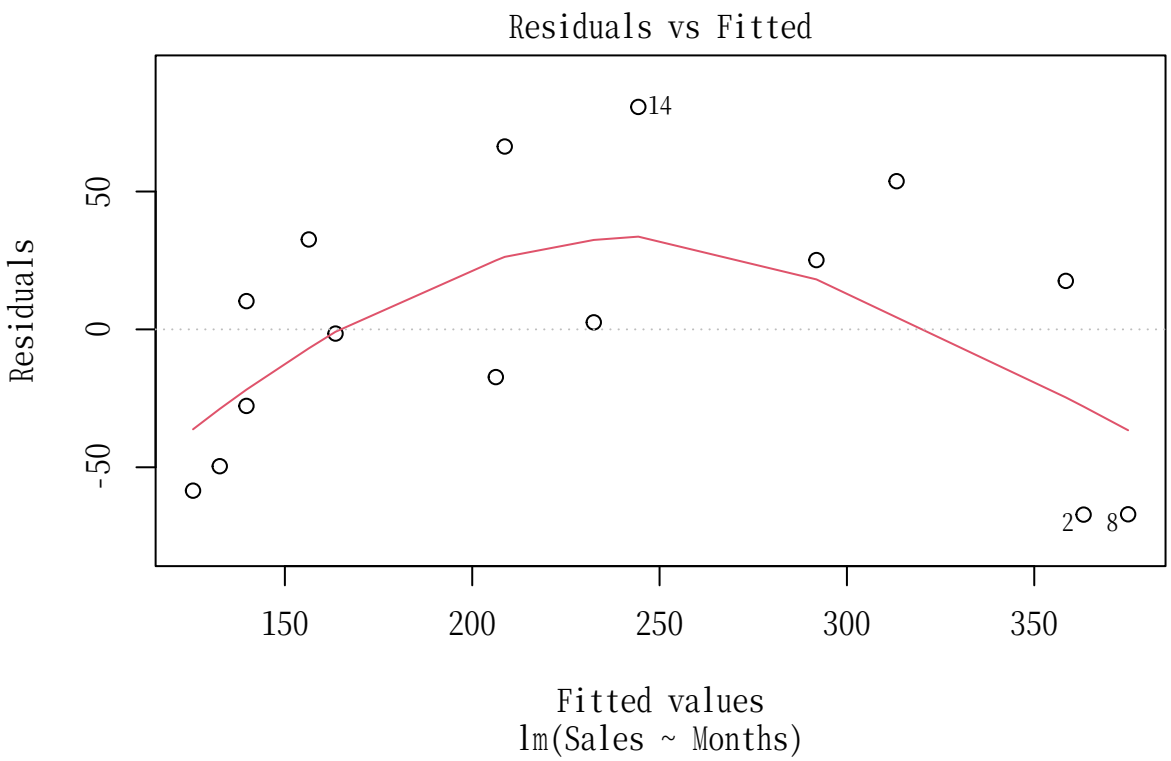
结果显著。$R^2 = 0.78$。拟合图：

```
with(Reynolds, plot(Months, Sales, main=" 线性近似"))
abline(lmre1, col="red", lwd=2)
```



残差诊断：

```
plot(lmre1, which=1)
```



残差图有明显的非线性。

考虑最简单的非线性模型：

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \varepsilon$$

令 $x_1 = x$, $x_2 = x^2$，有

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$$

是二元线性回归模型。作二次多项式回归：

```
lmre2 <- lm(Sales ~ Months + I(Months^2), data=Reynolds)
summary(lmre2)
```

```
##
## Call:
## lm(formula = Sales ~ Months + I(Months^2), data = Reynolds)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -54.963 -16.691  -6.242  31.996  43.789
##
```

```
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 45.347579  22.774654   1.991  0.06973 .
## Months       6.344807   1.057851   5.998 6.24e-05 ***
## I(Months^2) -0.034486   0.008948  -3.854  0.00229 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 34.45 on 12 degrees of freedom
## Multiple R-squared:  0.9022, Adjusted R-squared:  0.8859
## F-statistic: 55.36 on 2 and 12 DF,  p-value: 8.746e-07
```

模型显著。R^2 从线性近似的 0.78 提高到 0.90。x^2 项的系数的显著性检验 p 值为 0.002，显著不等于零，说明二次项是必要的。

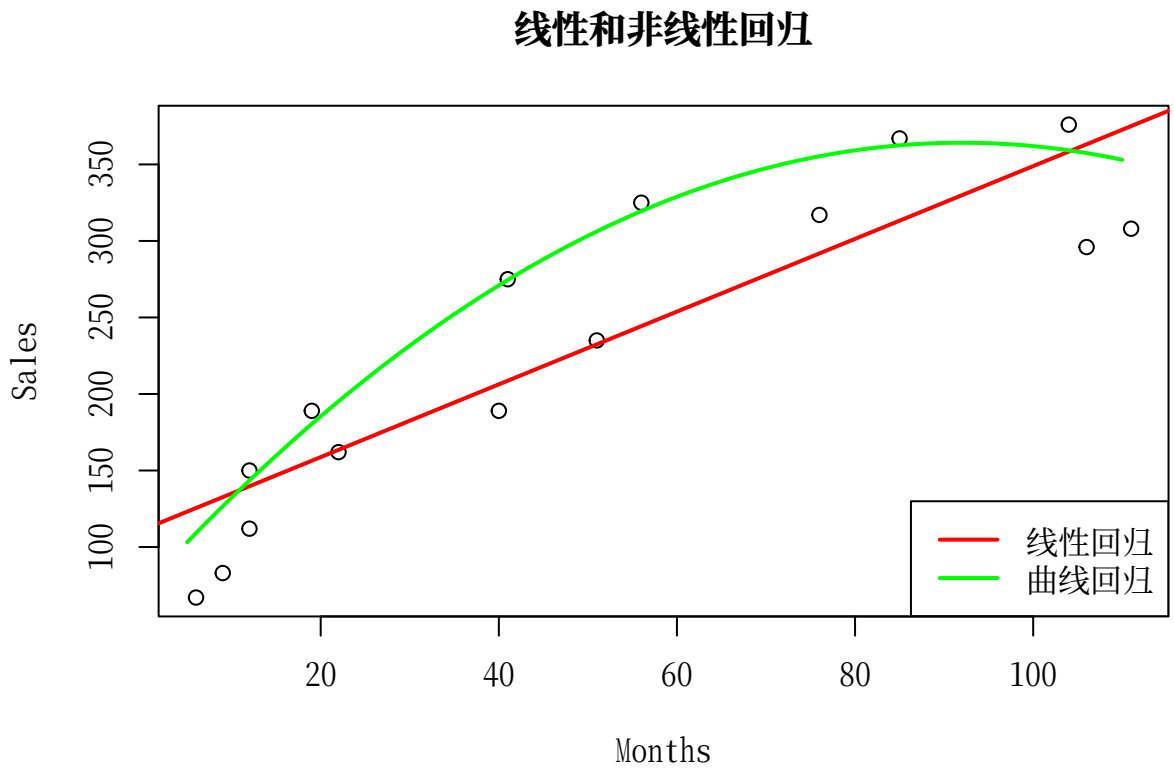
这样添加二次项容易造成 x 与 x^2 之间的共线性，所以添加中心化的二次项：

$$x_2 = (x - 60)^2$$

```
lmre3 <- lm(Sales ~ Months + I((Months-60)^2), data=Reynolds)
summary(lmre3)
```

```
##
## Call:
## lm(formula = Sales ~ Months + I((Months - 60)^2), data = Reynolds)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -54.963 -16.691  -6.242  31.996  43.789
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)        169.495742  21.331978   7.946 4.03e-06 ***
## Months               2.206535   0.246744   8.943 1.18e-06 ***
## I((Months - 60)^2)  -0.034486   0.008948  -3.854  0.00229 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 34.45 on 12 degrees of freedom
## Multiple R-squared:  0.9022, Adjusted R-squared:  0.8859
## F-statistic: 55.36 on 2 and 12 DF,  p-value: 8.746e-07
```

```
with(Reynolds, plot(Months, Sales, main=" 线性和非线性回归"))
abline(lmre1, col="red", lwd=2)
curve(196.50 + 2.2065*x - 0.03449*(x-60)^2, 5, 110,
      col="green", lwd=2, add=TRUE)
legend("bottomright", lty=1, lwd=2,
       col=c("red", "green"),
       legend=c(" 线性回归", " 曲线回归"))
```



32.16 分组建立多个模型

有时希望将数据按照一个或者几个分类变量分组，每一组分别建立模型，并将模型结果统一列表比较。broom 包可以用来将模型结果转换成规范的数据框格式。

以肺癌病人数据为例，建立 v1 对 v0 和 age 的多元线性回归模型：

```
print(d.cancer)
```

```
## # A tibble: 34 x 6
##       id   age sex   type     v0     v1
```

```
##    <dbl> <dbl> <chr> <chr> <dbl>  <dbl>
##  1     1    70 F     腺癌   26.5   2.91
##  2     2    70 F     腺癌  135.   35.1
##  3     3    69 F     腺癌  210.   74.4
##  4     4    68 M     腺癌   61    35.0
##  5     5    67 M     鳞癌  238.  128.
##  6     6    75 F     腺癌  330.  112.
##  7     7    52 M     鳞癌  105.   32.1
##  8     8    71 M     鳞癌   85.2  29.2
##  9     9    68 M     鳞癌  102.   22.2
## 10    10    79 M     鳞癌   65.5  21.9
## # ... with 24 more rows
```

```
lmgr01 <- lm(v1 ~ v0 + age, data = d.cancer)
summary(lmgr01)
```

```
##
## Call:
## lm(formula = v1 ~ v0 + age, data = d.cancer)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -42.757 -11.010  -2.475  11.907  52.941
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.90818   33.14895   0.239    0.814
## v0           0.43288    0.05564   7.780 1.79e-07 ***
## age         -0.12846    0.53511  -0.240    0.813
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 21.76 on 20 degrees of freedom
##   (11 observations deleted due to missingness)
## Multiple R-squared:  0.7683, Adjusted R-squared:  0.7451
## F-statistic: 33.16 on 2 and 20 DF,  p-value: 4.463e-07
```

用 broom 包的 `tidy()` 函数可以将系数估计结果转换成合适的 tibble 数据框格式:

```
library(broom)
tidy(lmgr01)
```

```
## # A tibble: 3 x 5
##   term        estimate std.error statistic     p.value
##   <chr>          <dbl>     <dbl>     <dbl>       <dbl>
## 1 (Intercept)    7.91     33.1        0.239 0.814
## 2 v0             0.433     0.0556     7.78  0.000000179
## 3 age           -0.128     0.535     -0.240 0.813
```

用 broom 包的 `augment()` 函数可以获得拟合值、残差等每个观测的回归结果：

```
knitr::kable(augment(lmgr01), digits=2)
```

.rownames	v1	v0	age	.fitted	.resid	.hat	.sigma	.cooksd	.std.resid
1	2.91	26.51	70	10.39	-7.48	0.13	22.25	0.01	-0.37
2	35.08	135.48	70	57.56	-22.48	0.06	21.68	0.03	-1.07
3	74.44	209.74	69	89.84	-15.40	0.10	22.01	0.02	-0.75
4	34.97	61.00	68	25.58	9.39	0.08	22.21	0.01	0.45
5	128.34	237.75	67	102.22	26.12	0.14	21.37	0.09	1.29
6	112.34	330.24	75	141.23	-28.89	0.33	20.81	0.43	-1.62
7	32.10	104.90	52	46.64	-14.54	0.13	22.04	0.03	-0.72
8	29.15	85.15	71	35.65	-6.50	0.08	22.27	0.00	-0.31
9	22.15	101.65	68	43.18	-21.03	0.05	21.77	0.02	-0.99
10	21.94	65.54	79	26.13	-4.19	0.22	22.30	0.00	-0.22
11	12.33	125.31	55	55.09	-42.76	0.10	19.79	0.16	-2.07
12	99.44	224.36	54	98.09	1.35	0.23	22.32	0.00	0.07
13	2.30	12.93	55	6.44	-4.14	0.12	22.30	0.00	-0.20
14	23.96	40.21	75	15.68	8.28	0.18	22.23	0.01	0.42
15	7.39	12.58	61	5.52	1.87	0.10	22.32	0.00	0.09
16	112.58	231.04	76	98.16	14.42	0.16	22.03	0.03	0.72
17	91.62	172.13	65	74.07	17.55	0.07	21.93	0.02	0.83
18	13.95	39.26	66	16.42	-2.47	0.09	22.32	0.00	-0.12
20	122.45	161.00	63	69.51	52.94	0.06	18.47	0.14	2.51
21	68.35	105.26	67	44.87	23.48	0.05	21.63	0.02	1.11
22	5.25	13.25	51	7.09	-1.84	0.16	22.32	0.00	-0.09
23	3.34	18.70	49	9.71	-6.37	0.18	22.27	0.01	-0.32
24	50.36	60.23	49	27.69	22.67	0.17	21.59	0.09	1.14

用 broom 包的 `glance()` 函数可以将回归的复相关系数平方、F 检验 p 值等整体结果做成一行的数据框：

```
knitr::kable(glance(lmgr01), digits=2)
```

r.squared	adj.r.squared	sigma	statistic	p.value	df	logLik	AIC	BIC	deviance	df.residual	nobs
0.77	0.75	21.76	33.16	0	2	-101.87	211.74	216.28	9470.34	20	23

下面将男病人与女病人分别建模，并以表格形式合并分组的建模结果。用 dplyr 包的 `group_by()` 函数分组，用 tidyr 包的 `nest()` 函数可以将分组后的数据框转换成一个新的数据框，新数据框中有一列是列表类型，每个元素是一个子数据框：

```
d.cancer %>%
  group_by(sex) %>%
  nest()
```

```
## # A tibble: 2 x 2
## # Groups:   sex [2]
##   sex   data
##   <chr> <list>
## 1 F     <tibble [13 x 5]>
## 2 M     <tibble [21 x 5]>
```

这样，可以用 `purr::map()` 函数对每一组分别建模，建模结果可以借助 broom 包制作成合适的数据框格式，然后用 `unnest()` 函数将不同组的结果合并成一个大数据框，如：

```
d.cancer %>%
  group_by(sex) %>%
  nest() %>%
  mutate(model = map(data, function(df)
    summary(lm(v1 ~ v0 + age, data=df)))) %>%
  mutate(tidyr = map(model, tidy)) %>%
  ungroup() %>%
  dplyr::select(sex, tidyr) %>%
  unnest(tidyr)
```

```
## # A tibble: 6 x 6
##   sex   term        estimate std.error statistic    p.value
##   <chr> <chr>          <dbl>     <dbl>     <dbl>      <dbl>
## 1 F     (Intercept)  171.     147.         1.16  0.310
## 2 F     v0             0.485    0.136      3.56  0.0235
## 3 F     age           -2.74     2.39      -1.15  0.316
## 4 M     (Intercept)  -17.2     30.5       -0.563 0.583
## 5 M     v0             0.486    0.0657     7.39  0.00000528
## 6 M     age            0.204    0.484      0.421 0.681
```

当需要分组拟合许多个模型时，这种方法比较方便。

32.17 R 语言公式界面

R 语言继承了来自 S 语言的公式界面，用以描述统计模型中因变量和自变量的关系，并有相应的将自变量群组转换为相应的线性模型设计阵的默认规则。

R 语言的线性回归 (`lm`)、方差分析 (`aov`)、广义线性模型 (`glm`)、线性混合模型 (`nlme::lme`) 等回归类建模函数都使用公式 (formula) 界面描述因变量与自变量之间的关系。公式都包含符号 `~`，在 `~` 左边是因变量，在 `~` 右边有一到多个自变量或者固定效应，各个自变量（效应）之间用加号 `+` 连接；两个自变量之间的交互作用效用之间以冒号`:` 连接。

32.17.1 公式中的运算符

`+` 和 `~` 是公式中最基本的运算符，还有其它一些辅助的运算符用来简化公式格式。下面给出一些仅使用基本运算符的公式的例子，其中 `y` 表示因变量，`x1` 是连续型自变量，`f1`, `f2`, `f3` 是因子：

- `y ~ x1`：一元线性回归；
- `y ~ 1 + x1`: 显式表明有截距项;
- `y ~ 0 + x1`: 表明没有截距项；
- `y ~ f1 + x1`: 仅有因子主效应的协方差分析，或平行线回归模型;
- `y ~ f1 + x1 - 1`: 也是平行线模型但是 `f1` 每个水平都直接有截距项估计；
- `y ~ f1 + x1 + f1:x1`: 包含因子主效应以及因子与连续型自变量的交互作用，即截距和斜率都不同的模型;
- `y ~ -1 + f1 + f1:x1`, 也是因子不同水平下都关于 `x1` 有不同截距和斜率的模型，而且对每个水平都直接有截距项和斜率项估计；
- `y ~ f1 + f2 + f1:f2`: 包含主效应和交互作用效应的两因素方差分析；
- `y ~ f1 + f1:f3`: 包含 `f1` 主效应和嵌套在 `f1` 内部的 `f3` 效应的方差分析；
- `y ~ x1 + f1 + f2 + x1:f1 + x1:f2 + f1:f2`: 包括三个主效应和所有交互效应的模型。

在公式中，将截距项称为 0 阶项，主效应称为 1 阶项，两个自变量的交互作用称为 2 阶项，等等。

公式中一般自动包含截距项，可以用一个 `0` 或者`-1` 项表示没有截距项。

公式中还可以用如下的操作符对公式进行简化描述：

- `*`: 两个因子的主效应以及交互作用效应，如 `f1 * f2` 等价于 `f1 + f2 + f1:f2`。
- `f3 %in% f1`: 表示因子 `f3` 嵌套在因子 `f1` 内部，如 `f1 + f3 %in% f1` 等价于 `f1 + f1:f3`，这时，`f1` 的效应照常估计，由于因子的哑变量编码通常会去掉第一个水平作为基线，所以 K 个水平 `f1` 的主效应仅显示后 $K-1$ 个水平的效应，而 `f3` 的效应是针对 `f1` 的 K 个水平的每一个都显示。
- `f1 / f3`：表示 `f1` 主效应以及 `f3` 嵌套在 `f1` 内的交互作用，等价于 `f1 + f1:f3`。

- `f1 / x1`: 表示 `f1` 主效应和 `x1` 嵌套在 `f1` 内，实际效果是 `f1` 的每组对 `x1` 有不同的截距项和不同的斜率项，不同的截距项实际是由 `f1` 主效应提供的，等效于 `f1 + f1:x`。`-1 + f1 / x1` 则不使用共同的截距项，使得每个 `f1` 水平对 `x1` 有单独的截距和斜率，其中的截距的解释更直接。
- `^`：将若干项的和进行平方表示这些项的主效应以及两两的交互作用，更高的方幂可以表示更多因子的交互作用，如 `(x1 + f1 + f2)^2` 表示这三项的主效应以及两两的交互作用效应，注意 `x1:x1`, `f1:f1` 并没有意义。
- `-`: 扣除某一项，如 `(x1 + f1 + f2)^2 - f1:f2` 会扣除本来会包括的 `f1` 和 `f2` 的交互作用效应。

32.17.2 公式中的复合项

在公式中还可以用自变量的变换函数、自变量的多项式、基本样条等函数，以及用 `I()` 保护的一般自变量表达式。如：

- 用 `sqrt(x1)`, `log(x1)`, `exp(x1)` 等表示自变量或因变量的某种函数变换；
- 用 `orderd(x1, breaks)` 表示将连续型自变量用指定的分点转换成有序因子；
- 用 `poly(x1, 2)` 表示 `x1` 的一次项和二次项（零次项即截距项一般与模型截距项合并）；
- 用 `bs(x1, df=3)` 表示 `x1` 的基本样条基展开。

因为自变量的计算公式也用到加减乘除这些符号，而这些符号在公式界面中有另外的解释，所以用计算表达式表示某个公式项时，应将这个公式项用 `I()` 保护，如 `I(x1 + x2*x3)` 是直接计算 `x1 + x2*x3` 的值作为一个效应而不是表示四个公式效应。

可以用 `formula()` 函数显式地生成公式，用 `update()` 函数修改公式。如：

```
form1 <- formula(y ~ (x1 + x2 + x3)^2)
form2 <- update(form1, . ~ . - x2:x3)
```

`update()` 中的 `.` 表示公式原来的左侧或右侧内容。

32.17.3 term 类型

在 `lm()` 等函数使用公式时，会产生一个 `term` 类型的对象，这个对象与输入数据框配合就可以产生模型框 (model frame) 和设计阵。term 对象保存了公式用到的变量、公式中各个项、有无截距项等信息，如果公式中用了 `*`，`^` 这样的简写，term 对象会将其转换成用 `:` 表达的形式。

32.18 R 的设计阵

在线性模型应用中，给定了公式和输入数据框后，将这两者合并成为一个模型框 (model frame)，然后将所有连续型自变量、因子、交互作用效应进行编码，得到一个设计阵，这就是线性模型 $y = X\beta + \epsilon$ 中的 X 矩阵。实际上，这些转换一般不需要建模用户自己操作，而是由 `lm()`, `glm()`, `nlme::lme()` 等函数自动在后台完成。但是，当效应比较复杂时，为了能够更好地理解估计问题并看懂估计结果，有必要了解这些背后的转换。

这些转换的步骤为：从公式生成 terms 对象，将 terms 对象与输入数据框转换成模型框，将模型框转换成设计阵。

当所有自变量都是连续型变量时，如果有截距项，这时设计阵就是每个自变量的观测值占一列，并在第一列有代表截距项的全为 1 的一列。这与一般的多元线性回归教材的做法一致。

32.18.1 生成模型框

模型框是一个和数据框类似的 R 对象，是将公式所需的变量从输入数据框中提取计算而得，并将公式以 terms 对象的形式作为属性保存。除了各种建模函数自动生成以外，可以用 `model.frame()` 显式地生成模型框，此函数输入 `formula`, `data`, `subset` 和 `na.action` 自变量。`na.action` 默认为 `na.omit`，即删除有缺失的观测；类似的 `na.exclude` 在建模时删除有缺失的观测但计算残差和预测值时可对有缺失的观测计算。

例如，nmleU 包的 armd.wide 数据框包含了 ARMD 眼科疾病的治疗数据，如下的程序生成了某个模型的模型框：

```
data(armd.wide, package="nlmeU")
```

公式：

```
armd.fm1 <- formula(
  visual52 ~                    # 因变量，连续型
    sqrt(line0) +               # 连续型自变量的函数变换
    factor(lesion) +            # 有四个水平的因子
    treat.f * log(visual24) +
    # 两水平的因子与上颚连续型自变量函数变换的主效应和交互作用效应
    poly(visual0, 2))           # 连续型自变量的一次和二次项
```

模型框：

```
armd.mf1 <- model.frame(
  armd.fm1,                             # 输入公式
  data = armd.wide,                     # 输入数据框
  subset = !(subject %in% c("1", "2")), # 取子集
  na.action = na.exclude, # 缺失值处理策略
  SubjectId = subject)    # 可以额外定义变量
class(armd.mf1)
```

```
## [1] "data.frame"
```

```
## 原始数据的观测行数和变量数
dim(armd.wide)
```

```
## [1] 240  10
```

```r
## 注意模型框的观测行数少于输入数据框，因为取了子集并进行了缺失值处理
dim(armd.mf1)
```

```
## [1] 189   7
```

```r
names(armd.mf1)
```

```
## [1] "visual52"         "sqrt(line0)"      "factor(lesion)"   "treat.f"
## [5] "log(visual24)"    "poly(visual0, 2)" "(SubjectId)"
```

将模型框作为数据框显示：

```r
head(armd.mf1, 3)
```

```
##   visual52 sqrt(line0) factor(lesion) treat.f log(visual24) poly(visual0, 2).1
## 4       68    3.605551              2 Placebo      4.158883        0.052346233
## 6       42    3.464102              3  Active      3.970292        0.017581526
## 7       65    3.605551              1 Placebo      4.276666        0.039309468
##   poly(visual0, 2).2 (SubjectId)
## 4       -0.005443471           4
## 6       -0.046084343           6
## 7       -0.024394420           7
```

可见模型框提取了原始连续型变量，计算了连续型变量的变换，原样提取了因子而没有生成哑变量，计算了 `poly()` 需要的一次项和二次项，提取了因子的主效应但是没有计算因子和连续型变量的交互作用效应。注意前面 `names(armd.mf1)` 的结果为 7 项，但是显示的模型框中有 8 列，这是因为 `poly(visual0, 2)` 生成了两列，作为一个两列的矩阵保存在模型框中。在 `model.frame()` 调用中用 `SubjectId=subjectid` 将公式中没有出现的变量 `subjectid` 改名为 `(SubjectId)` 复制到了输出的模型框中（注意名字带有括号），这种做法还可以将不在设计阵中用到的变量包含进模型框中，比如模型的权重、位移等。

因为因子需要变成哑变量，还需要计算交互作用效应，所以模型框是一个中间结果，不能直接用来进行线性模型计算。

32.18.2 模型框中 terms 对象的属性

数据框中包含了公式转换出来的 terms 对象，以模型框的属性的形式保存。因为公式有对应的数据框作为参考，所以可以知道公式中哪些是连续型变量，哪些是因子，所以 terms 对象包含了比公式本身更详细的信息。如：

```
armd.tm1 <- attr(armd.mf1, "terms")
class(armd.tm1)
```

```
## [1] "terms"   "formula"
```

terms 对象的属性中保存公式和模型框的重要信息，其属性有：

```
names(attributes(armd.tm1))
```

```
##  [1] "variables"    "factors"      "term.labels"  "order"        "intercept"
##  [6] "response"     "class"        ".Environment" "predvars"     "dataClasses"
```

```
attr(armd.tm1, "variables")
```

```
## list(visual52, sqrt(line0), factor(lesion), treat.f, log(visual24),
##     poly(visual0, 2))
```

variable 属性是模型框的各变量的名字，注意其中 `poly(visual0, 2)` 代表一个两列矩阵。

```
attr(armd.tm1, "factors")
```

```
##                  sqrt(line0) factor(lesion) treat.f log(visual24)
## visual52                   0              0       0             0
## sqrt(line0)                1              0       0             0
## factor(lesion)             0              1       0             0
## treat.f                    0              0       1             0
## log(visual24)              0              0       0             1
## poly(visual0, 2)           0              0       0             0
##                  poly(visual0, 2) treat.f:log(visual24)
## visual52                         0                     0
## sqrt(line0)                      0                     0
## factor(lesion)                   0                     0
## treat.f                          0                     1
## log(visual24)                    0                     1
## poly(visual0, 2)                 1                     0
```

factors 属性保存一个矩阵，表现模型框中各项出现在公式的哪些项中，用 0 表示不出现，1 表示出现而且对因子使用对照方式编码，2 表示出现而且对因子使用全部哑变量编码（这适用于仅有高阶项没有相应低阶项情形）。即使没有截距项因子的主效应也用 1 表示，这时第一个因子用全部哑变量编码。

```
attr(armd.tm1, "term.labels")
```

```
## [1] "sqrt(line0)"           "factor(lesion)"        "treat.f"
## [4] "log(visual24)"         "poly(visual0, 2)"      "treat.f:log(visual24)"
```

term.labels 属性是将公式展开后各项的标签，这会将 `*`, `^` 的简写展开，但因子和 `poly()` 等在设计阵中需要变成多列的不会有多个标签。

```
attr(armd.tm1, "order")
```

```
## [1] 1 1 1 1 1 2
```

orders 属性是 terms.label 各项对应的阶数，主效应是 1 阶，两个变量的交互作用效用是 2 阶。

```
attr(armd.tm1, "intercept")
```

```
## [1] 1
```

虽然公式中没有显式地写出模型的截距项，但默认有截距项，上面结果 1 就是表示截距项。

```
attr(armd.tm1, "response")
```

```
## [1] 1
```

response 属性保存因变量在模型框中的次序，0 表示公式中没有因变量。

```
attr(armd.tm1, "predvars")
```

```
## list(visual52, sqrt(line0), factor(lesion), treat.f, log(visual24),
##     poly(visual0, 2, coefs = list(alpha = c(54.9541666666667,
##     50.5097520799239), norm2 = c(1, 240, 52954.4958333333, 16341393.4347853
##     ))))
```

predvars 列出了模型因变量和自变量，其中 `poly()` 给出了所用的正交多项式参数，在利用模型进行预测时可能需要利用这个属性中保存的参数。

```
attr(armd.tm1, "dataClasses")
```

```
##         visual52      sqrt(line0)   factor(lesion)          treat.f
##        "numeric"        "numeric"         "factor"         "factor"
##    log(visual24) poly(visual0, 2)      (SubjectId)
##        "numeric"      "nmatrix.2"         "factor"
```

dataClasses 属性给出了模型框中变量的数据类型。注意其中 `poly(visual0, 2)` 保存了一个两列的数值型矩阵。

32.18.3 模型框中 poly 等变量

在上面的例子中，公式中的 poly(visual0, 2) 在模型框中生成了一个两列的矩阵。poly() 函数会生成自变量的正交多项式系数，而使用的正交多项式是与输入自变量值有关的。splines 包提供的 bs() 和 ns() 函数也需要从自变量值计算节点，所以也是与输入自变量有关的。与之相对照，sqrt(x) 等变换则总是执行固定的函数变换。

poly() 函数在计算所用到的正交多项式系数时会使用输入数据框的所有观测，不会按照 model.frame() 函数中的 subset 选项和 na.action 选项进行子集处理。计算出的参数会保存在 terms 对象的 predvars 属性中。如果对原始输入数据中的 visual0 计算得到的带有参数的 poly() 函数，可以发现结果是两列的矩阵，分别是一次多项式变换和二次多项式变换，每列的列和等于 0，从而与截距项（零次项）正交，每列的平方和等于 1（标准化），两列之间正交。

32.18.4 生成设计阵

为了能够利用线性模型计算方法进行模型估计，需要将模型框中的因子变成可计算的哑变量或者对照矩阵，交互作用效应计算出来。R 的建模函数如 lm() 会在后台自动完成这样的转换，为了理解估计过程以及输出的模型参数估计结果，可以用 model.matrix() 函数直接产生设计阵进行学习。另外，R 中某些包（如机器学习类）不支持公式界面，必须自己输入可直接计算的自变量矩阵（设计阵），这也需要调用 model.matrix() 完成。

model.matrix() 以公式和模型框为输入，输出数值型的设计阵：

```
armd.dm1 <- model.matrix(
  armd.fm1, armd.mf1)
dim(armd.dm1)
```

```
## [1] 189  10
```

生成的设计阵与模型框观测行数相同，列数增加了。各列为：

```
colnames(armd.dm1)
```

```
##  [1] "(Intercept)"                 "sqrt(line0)"
##  [3] "factor(lesion)2"             "factor(lesion)3"
##  [5] "factor(lesion)4"             "treat.fActive"
##  [7] "log(visual24)"               "poly(visual0, 2)1"
##  [9] "poly(visual0, 2)2"           "treat.fActive:log(visual24)"
```

使用列名的简写将前几行列表显示：

```
knitr::kable(head(armd.dm1),
  col.names = abbreviate(colnames(armd.dm1)),
  digits=2)
```

	(In)	s(0)	f()2	f()3	f()4	tr.A	l(24	p(0,2)1	p(0,2)2	t.A:
4	1	3.61	1	0	0	0	4.16	0.05	-0.01	0.00
6	1	3.46	0	1	0	1	3.97	0.02	-0.05	3.97
7	1	3.61	0	0	0	0	4.28	0.04	-0.02	0.00
8	1	2.83	0	1	0	0	3.61	-0.07	-0.01	0.00
9	1	3.46	1	0	0	1	3.99	0.02	-0.05	3.99
12	1	3.00	0	0	0	1	3.30	-0.04	-0.04	3.30

设计阵的第一列是 (Intercept)，模型截距项，是一列 1。因子 factor(lesion) 有 4 个水平，用三个哑变量表示；因子 factor(treat.f) 有两个水平，用一个哑变量表示；交互作用 treat.f:log(visual24) 是两水平的因子和连续型变量的交互，等于一个零壹变量和一个连续型变量的乘积，在设计阵中变量名为 treat.fActive:log(visual24)，即处理组的示性函数与 log(visual24) 的乘积。

设计阵有一个属性 assign，表示设计阵的每一列来自模型框 term 对象的哪一项：

```
attr(armd.dm1, "assign")
```

```
##  [1] 0 1 2 2 2 3 4 5 5 6
```

```
attr(armd.tm1, "term.labels")
```

```
## [1] "sqrt(line0)"           "factor(lesion)"        "treat.f"
## [4] "log(visual24)"         "poly(visual0, 2)"      "treat.f:log(visual24)"
```

这表明设计阵的第 1 列截距项在 term.labels 没有对应的项，设计阵的第 2 列 sqrt(line0) 对应于 term.labels 的同名的第 1 项，设计阵的第 3-5 列 factor(lesion)2、factor(lesion)3、factor(lesion)4 对应于 term.labels 的第 2 项 factor(lesion)，设计阵的第 6 列 treat.fActive 对应于 term.labels 的第 3 项 treat.f，设计阵的第 7 列 log(visual24) 对应于 term.labels 的同名的第 4 项，设计阵的第 8-9 列 poly(visual0, 2)1、poly(visual0, 2)2 对应于的第 5 项 poly(visual0, 2)，设计阵的第 10 列 treat.fActive:log(visual24) 对应于 term.labels 的第 6 项 treat.f:log(visual24)。

因子和含有因子的交互作用需要进行编码，其编码方式存放在设计阵的属性 contrasts 中：

```
attr(armd.dm1, "contrasts")
```

```
## $`factor(lesion)`
## [1] "contr.treatment"
##
## $treat.f
## [1] "contr.treatment"
```

32.19 R 对照矩阵

因子的主效应反映在设计阵中，需要使得因子的每个水平有自己对截距项的不同贡献。例如，最简单的仅有一个 K 水平因子 a 作为自变量的模型为

$$y_{ji} = \mu + \alpha_j + e_{ji}, i = 1, 2, \dots, n_i; j = 1, 2, \dots, K, \tag{32.3}$$

其中 y_{ji} 是自变量因子在第 j 水平的第 i 个观测。这时，设计阵可以使用 $X = (x_{ij})_{n\times(K+1)}$，其中第一列都等于 1，对应于参数 μ，第 $j+1$ 列取因子 a 的第 j 水平的示性函数值，对应于参数 α_j，$x_{i,j+1} = 1$ 当且仅当 y_{ij} 属于第 j 组，否则取 0。这样的设计阵的问题是 X 的第一列都等于 1 而其余列的和等于第一列，使得设计阵不满秩。记这个设计阵为 $X = (1\ X_a)$，其列数为 $K+1$ 但列秩为 K。为此，取一个 $K \times (K-1)$ 的 “对照矩阵”C_a，取新的设计阵为 $X^* = (1\ X_a C_a)$，则 X^* 为 $n \times K$ 列满秩矩阵。

关于对照矩阵的一个必要条件是 $(1\ C_a)$ 构成满秩 $K \times K$ 方阵，这个条件也经常是充分条件。

原来的矩阵形式模型为

$$Y = (1\ \ X_a)\begin{pmatrix}\mu\\ \alpha\end{pmatrix} + e,$$

其中 $\alpha = (\alpha_1, \dots, \alpha_K)^T$。改成 X^* 后，除了 μ 以外关于因子 a 就只能有 $K-1$ 个参数 α^*，矩阵形式的模型变成

$$Y = (1\ \ X_a C_a)\begin{pmatrix}\mu\\ \alpha^*\end{pmatrix} + e,$$

所以要求

$$\alpha = C_a \alpha^*.$$

实际上，模型(32.3)是不可辨识的，为了模型可估，必须对参数添加某些约束。如果存在非零 c_a 向量使得 $c_a^T C_a = 0$，则对模型参数 α 添加如下约束：

$$c_a^T \alpha = c_a^T C_a \alpha = 0,$$

于是模型(32.3)在添加上式的约束后变得可辨识，同时因为 C_a 列满秩且 $C_a\alpha = \alpha^*$，所以有

$$C_a^T C_a \alpha^* = C_a^T \alpha,\ \alpha^* = (C_a^T C_a)^{-1} C_a^T \alpha,$$

即原始的参数 α 可以与新的参数 α^* 互相唯一表示。记 $C_a^+ = (C_a^T C_a)^{-1} C_a^T$，则

$$\alpha = C_a \alpha^*, \quad \alpha^* = C_a^+ \alpha.$$

其中 C_a 是 $K \times (K-1)$ 矩阵，C^+ 是 $(K-1) \times K$ 矩阵，矩阵 C_a 称为因子 a 的对照矩阵。

R 中提供的对照函数有 `contr.treatment()`, `contr.sum()`, `contr.helmert()`, `contr.poly()`, `contr.SAS()`。除了 `contr.treatment()` 以外，R 中的对照矩阵一般都以 $1^T C_a = 0$ 作为约束。

`options()$contrasts` 可以访问因子和有序因子的默认使用的对照函数，对因子默认使用 `contr.treatment()`，对有序因子默认使用 `contr.poly()`：

```
options("contrasts")
```

```
## $contrasts
##         unordered           ordered
## "contr.treatment"      "contr.poly"
```

对照函数默认生成 $K \times (K-1)$ 矩阵。其中 K 是因子的水平个数。

以 3 个水平的因子为例，在有公共截距项的情况下 3 个水平的因子需要用两个哑变量编码：

```
contr.treatment(3)
```

```
##   2 3
## 1 0 0
## 2 1 0
## 3 0 1
```

这个矩阵表明因子水平 1 用 $(0,0)$ 表示，水平 2 用 $(1,0)$ 编码，水平 3 用 $(0,1)$ 编码。对 α 的约束相当于 $(1,0,0)\alpha = \alpha_1 = 0$，显然 $(1,0,0)C_a = (0,0)$, C_a 表示上面的对照矩阵。

在这种参数设置下，设截距项为 β_0，两个哑变量的系数为 β_1, β_2，则水平 1 的均值为 β_0，水平 2 的均值为 $\beta_0+\beta_1$，水平 3 的均值为 $\beta_0+\beta_2$。检验 $H_0: \beta_1 = 0$ 可以检验水平 2 与水平 1 是否有显著差异，检验 $H_0: \beta_2 = 0$ 可以检验水平 3 与水平 1 是否有显著差异，没有直接检验水平 2 与水平 3 有无显著差异的方法。

可以用如下程序给出用旧参数 α 表示新参数 $\alpha^* = C_a^+\alpha$ 的变换矩阵 C_a^+：

```
contr.treatment(3) |> MASS::ginv() |> MASS::fractions()
```

```
##      [,1] [,2] [,3]
## [1,] 0    1    0
## [2,] 0    0    1
```

这说明 $\beta_1 = \alpha_2, \beta_2 = \alpha_3$，约束为 $\alpha_1 = 0$。

`contr.treatment()` 默认与第一个水平比较，可以加 `base=` 选项指定与另外一个水平比较：

```
contr.treatment(3, base=2)
```

```
##   1 3
## 1 1 0
## 2 0 0
## 3 0 1
```

这是水平 1 和水平 3 分别与水平 2 比较。

`contr.SAS()` 是 `contr.treatment()` 设定最后一个水平为基线的变种。

`contr.sum()` 函数生成的对照：

```
contr.sum(3)
```

```
##   [,1] [,2]
## 1    1    0
## 2    0    1
## 3   -1   -1
```

这个对照矩阵对应的参数约束向量为 $c_a = 1$，即要求 $\alpha_1 + \alpha_2 + \alpha_3 = 0$，这也是方差分析模型常用的约束。设截距项为 β_0，两个哑变量的系数为 β_1, β_2，哑变量系数分别代表水平 1 和水平 2 与模型总平均值的比较。事实上，对水平 1，因变量均值为 $\beta_0 + \beta_1$, 对水平 2，因变量均值为 $\beta_0 + \beta_2$, 对水平 3，因变量均值为 $\beta_0 - (\beta_1 + \beta_2)$, 这时 $\beta_1, \beta_2, -\beta_1 - \beta_2$ 恰好相当于单因素方差分析模型中三个水平的主效应，β_0 相当于总平均值。

新参数用老参数表示的矩阵 C^+ 为：

```
contr.sum(3) |> MASS::ginv() |> MASS::fractions()
```

```
##      [,1] [,2] [,3]
## [1,]  2/3 -1/3 -1/3
## [2,] -1/3  2/3 -1/3
```

这说明 $\beta_1 = (2\alpha_1 - (\alpha_2 + \alpha_3))/3 = \alpha_1 - \frac{1}{3}(\alpha_1 + \alpha_2 + \alpha_3)$, $\beta_2 = \alpha_2 - \frac{1}{3}(\alpha_1 + \alpha_2 + \alpha_3)$，分别表示前两个水平与三个水平均值的平均值的比较，仅在均衡设计时个水平的平均值 $\frac{1}{3}(\alpha_1 + \alpha_2 + \alpha_3)$ 才有意义。

`contr.helmert()` 将水平 2 与水平 1 比较，将水平 3 与前 2 个水平的平均值比较，以此类推。参数约束向量仍为 $c_a = 1$。

```
contr.helmert(3)
```

```
##   [,1] [,2]
## 1   -1   -1
## 2    1   -1
## 3    0    2
```

新参数用旧参数表示的矩阵 C^+ 为：

```
contr.helmert(3) |> MASS::ginv() |> MASS::fractions()
```

```
##      [,1] [,2] [,3]
## [1,] -1/2  1/2    0
## [2,] -1/6 -1/6  1/3
```

这里 $\beta_1 = \alpha_2 - \alpha_1$ 是水平 2 与水平 1 的比较，$\beta_2 = \frac{1}{3}[\alpha_3 - \frac{1}{2}(\alpha_1 + \alpha_2)]$ 是水平 3 与前两个水平均值的比较。

`contr.poly()` 利用正交多项式产生对照。

```
contr.poly(3)
```

```
##                  .L         .Q
## [1,] -7.071068e-01  0.4082483
## [2,] -7.850462e-17 -0.8164966
## [3,]  7.071068e-01  0.4082483
```

在某些特殊情况下可能需要用 k 个哑变量表示有 k 个水平的因子，比如，模型中没有截距项时。在对照函数中设置选项 `contrasts=FALSE` 可以生成 $k \times k$ 的对照矩阵。如

```
contr.treatment(3, contrasts=FALSE)
```

```
##   1 2 3
## 1 1 0 0
## 2 0 1 0
## 3 0 0 1
```

如果有两个因子 a, b，分别有 K, J 个水平，没有交互作用时，只要将每个因子利用对照矩阵将其原始设计阵 X_a，X_b 转换成 $n \times (K-1)$ 矩阵 $X_a C_a$ 和 $n \times (J-1)$ 矩阵 $X_b C_b$。如果有交互作用效应 $a : b$，则只要将 $X_a C_a$ 的 $K-1$ 列与 $X_b C_b$ 的 $J-1$ 列两两进行对应元素乘法，得到 $(K-1) \times (J-1)$ 列，这些列就作为设计阵中对应于两个因子的交互作用效应的部分。

如果一个因子 a 和一个连续变量 x 作交互作用 $a : x$，只要将因子 a 产生的 $K-1$ 列 $X_a C_a$ 的每一列与 x 产生的一列作对应元素乘法，得到新的 $K-1$ 列，就可以作为 a 和 x 的交互作用在设计阵中的部分。这相当于在因子 a 的不同水平下连续变量 x 有不同的斜率项。

两个连续变量 x, z 的交互作用 $x : z$ 就是将两列的对应元素相乘产生一列，相当于增加了一个新变量 $v = xz$。

可以用如下方法对某个因子附加非默认的对照函数：

```
lesion.f <- factor(armd.wide$lesion)
contrasts(lesion.f) <- contr.sum(4)
```

提取因子的对照矩阵：

```
contrasts(lesion.f)
```

```
##   [,1] [,2] [,3]
## 1    1    0    0
## 2    0    1    0
## 3    0    0    1
## 4   -1   -1   -1
```

如果模型是均衡设计的方差分析，则使用默认的 `contr.treatment()` 或 `contr.poly()` 可以使设计阵各列正交，这样各回归系数的估计值不相关。

32.20 R 信息提取函数

前面演示过可以用 `summary()` 从回归结果中显示参数估计、检验等信息，用 `anova()` 比较嵌套的模型，用 `AIC()` 比较模型。这些函数称为信息提取函数，这里对信息提取函数进行简单列举，设 `lmr` 为 `lm()` 的输出结果，`summ` 为 `summary()` 的结果。

关于参数估计：

- `summary(lmr)`：结果包括参数估计、标准误差、t 检验、F 检验、复相关系数平方等；
- `coef(lmr)`: 提取回归系数，返回一个向量；
- `coef(summ)`: 返回包括参数估计、标准误差、t 统计量、p 值的矩阵；
- `vcov(lmr)`: 返回系数估计向量的协方差阵估计；
- `confint(lmr)`：返回系数估计向量的置信区间，默认置信度为 95%；
- `summ$sigma`：返回误差项标准差估计 $\hat{\sigma}$。

如果使用了最大似然估计 (ML) 或者限制最大似然估计 (REML)，可以提取：

- `logLik(lmr)`：最大似然估计对应的似然函数值；
- `logLik(lmr, REML=TRUE)`：限制最大似然估计对应的限制似然函数值；
- `AIC(lmr)`, `BIC(lmr)`：AIC 或者 BIC 值。

关于拟合值与残差：

- `fitted(lmr)` 或 `predict(lmr)`: 拟合值；

- `residuals(lmr)`：原始的残差；
- `rstandard(lmr)`：内部学生化残差；
- `rstudent(lmr)`: 外部学生化残差；
- `predict(lmr, interval="confidence")`：拟合值及均值的置信区间；
- `predict(lmr, interval="prediction")`：拟合值及观测值的置信区间。

Chapter 33

R 非参数回归

33.1 模型

线性回归模型可以看成非线性回归模型的特例:

$$Y = f(X) + \varepsilon$$

其中 $f(x)$ 为未知的回归函数。

参数方法：假定 $f(x)$ 具有某种形式，如

- $f(x) = a + bx$: 线性回归；
- $f(x) = a + bx + cx^2$: 二次多项式回归；
- $f(x) = Ae^{bx}$: 指数模型，等等。

二次多项式回归可以令 $X_1 = x, X_2 = x^2$，变成二元回归模型来解决。指数模型可以令 $z = \ln Y$, 模型化为 $z = a + bx$。有一些曲线模型可以通过变换化为线性回归。

在多元情形，一般的非线性回归模型为

$$Y = f(x_1, x_2, \dots, x_p) + \varepsilon$$

但是这样可选的模型就过于复杂，难以把握。比较简单的是不考虑变量之间交互作用的可加模型：

$$Y = \sum_{j=1}^{p} f_j(x_j) + \varepsilon$$

其中 $f_j(\cdot)$ 是未知的回归函数，需要满足一定的光滑性条件。

$f_j(\cdot)$ 可以是参数形式的，比如二次多项式、三次多项式、阶梯函数等。较好的一种选择是使用三次样条函数。

所谓参数回归，是指回归函数 $f(\cdot)$ 有预先确定的公式，仅需要估计 $f(\cdot)$ 的未知参数；非参数回归，就是 $f(\cdot)$ 没有预先确定的公式，$f(\cdot)$ 的形式本身也依赖于输入的样本 (x_i, y_i), $i = 1, 2, \dots, n$。下面描述的核回归就是这样典型的非参数回归，样条平滑、样条函数回归一般也看作是非参数回归。

33.2 样条平滑

为了得到一般性的 Y 与 X 的曲线关系 $f(x)$ 的估计，可以使用样条函数。三次样条函数将实数轴用若干个**节点** (knots) $\{z_k\}$ 分成几段，每一段上 $\hat{f}(x)$ 为三次多项式，函数在节点处有连续的二阶导数。样条函数是光滑的分段三次多项式。通常我们使用“自然样条函数”，在最左边和最右边这两段为线性函数。

用样条函数估计 $f(x)$ 的准则是曲线接近各观测值点 (x_i, y_i)，同时曲线足够光滑。

在 R 中用 `stats::smooth.spline` 函数进行样条曲线拟合。取每个自变量 x_i 处为一个节点，对于给定的某个光滑度/模型复杂度系数值 λ，求函数 $f(x)$ 使得

$$\min_{f(\cdot)} \left\{ \sum_i [y_i - f(x_i)]^2 + \lambda \int [f''(x)]^2 \, dx \right\} \tag{33.1}$$

目标函数中第一项越小，拟合误差越小，但是也越容易产生过度拟合；目标函数中第二项度量了函数 $f(\cdot)$ 的不光滑性，是一个对模型复杂度的惩罚项，λ 为调节参数，λ 越大，惩罚越重，所得的曲线越光滑。`smooth.spline()` 函数可以通过交叉验证方法自动取得一个对于预测最优的光滑参数 λ 值。

问题(33.1)的理论解称为光滑样条，是一个自然样条函数，节点为输入的 x_i 的所有不同值。此理论解与 §33.4直接指定这些节点进行样条函数回归的结果不同，是依赖于调节参数 λ 进行了不光滑性惩罚的结果。调节参数 λ 可以用交叉验证方法选择，只要在 `smooth.spline()` 函数中指定 `cv=TRUE`。也可以指定 `spar` 选项或者 `df` 选项，这里 `df` 代表“等效自由度”，是模型复杂度的一个度量方式。

下面的程序模拟产生了一个非线性回归数据集，然后用 `stats::smooth.spline()` 进行样条平滑估计：

```
set.seed(1)
nsamp <- 30
x <- runif(nsamp, -10, 10)
xx <- seq(-10, 10, length.out=100)
x <- sort(x)
y <- 10*sin(x/10*pi)^2 + rnorm(nsamp,0,0.3)
plot(x, y)
curve(10*sin(x/10*pi)^2, -10, 10, add=TRUE, lwd=2)

library(splines)
res <- smooth.spline(x, y)
lines(spline(res$x, res$y), col="red")
res2 <- loess(y ~ x, degree=2, span=0.3)
lines(xx, predict(res2, newdata=data.frame(x=xx)),
      col="blue")
legend("top", lwd=c(2,1,1),
       col=c("black", "red", "blue"),
       legend=c(" 真实函数关系", " 样条平滑结果", " 局部线性拟合"))
```

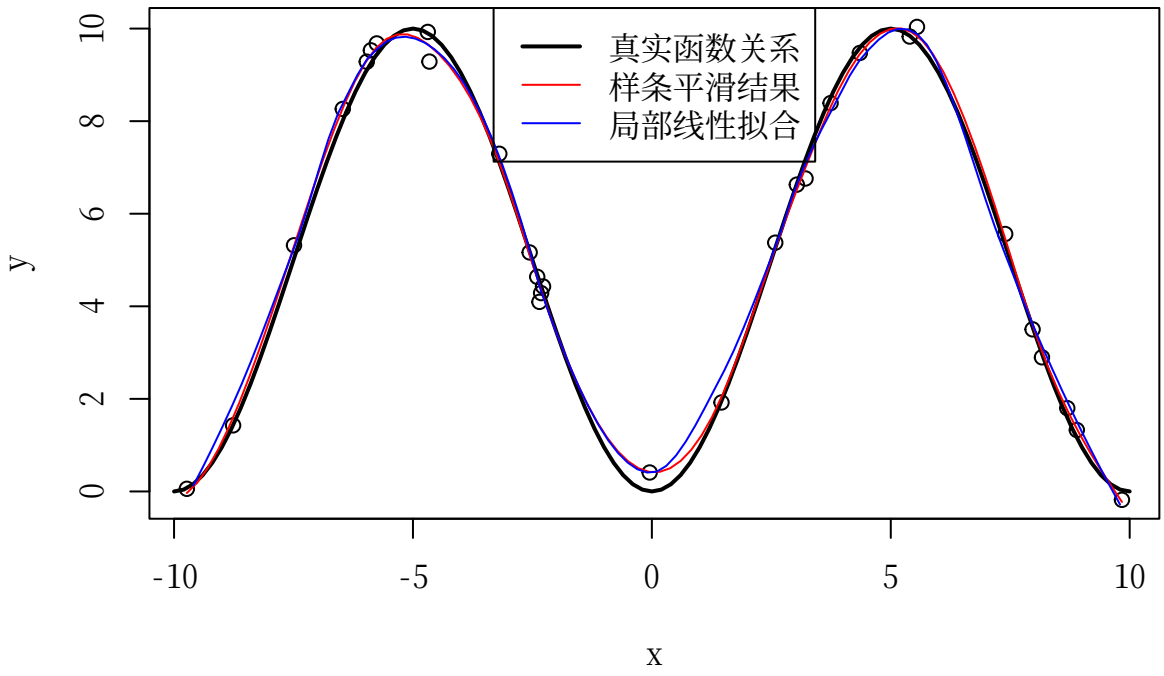


图 33.1: 曲线平滑示例

其中 res 是 `stats::smooth.splines()` 的输出，其元素 y 为拟合值，元素 x 为拟合值对应的自变量值。

R 函数 `spline(x,y)` 不是做样条平滑，而是做样条插值，它输入一组 x, y 坐标，用样条插值算法在原始的自变量 x 范围内产生等间隔距离的格子点值，输出包含格子点上的样条插值 x 和 y 坐标的列表。样条平滑曲线不需要穿过输入的各个散点，但是插值则需要穿过输入的各个散点。

R 函数 `approx(x,y)` 是另一个插值函数，它用线性插值方法产生线性插值后在等间隔的横坐标上的坐标值。

33.3 局部多项式曲线平滑

另一种非参数的拟合非线性回归曲线 $f(x)$ 的方法是对每个自变量 x 处选一个局部的小邻域，用这个小邻域附近的 (x_i, y_i) 点拟合一个局部的零次、一次或二次多项式，用拟合的局部多项式在 x 处的值作为回归函数在 x 处的值的估计。

局部零次多项式的方法叫做核回归，公式为

$$\hat{f}(x) = \frac{\sum_{i=1}^{n} K\left(\frac{x-X_i}{h}\right) Y_i}{\sum_{i=1}^{n} K\left(\frac{x-X_i}{h}\right)}$$

其中 $K(\cdot)$ 称为核函数，是类似标准正态密度这样的中间高两边低的可以用来加权的函数，比如，双三次核函数：

$$K(x) = \begin{cases} \left(1-|x|^3\right)^3 & |x| \le 1 \\ 0 & \text{其它} \end{cases}$$

```
kernel.dcube <- function(u){
  y <- numeric(length(u))
  sele <- (abs(u) < 1)
  y[sele] <- (1 - abs(u[sele])^3)^3
  y
}
curve(kernel.dcube, -1.5, 1.5, main=" 双三次核函数")
```

参数 h 称为窗宽，h 越大，参与平均的点越多，曲线越光滑，回归复杂度越低。h 选取可以用交叉验证方法。

R 扩展包 KernSmooth 的函数 `locpoly(x, y, degree=1, bandwidth=0.25)` 可以计算核回归，其中 `bandwidth` 是输入的窗宽，函数 `dpill(x,y)` 可以帮助确定窗宽。如 `locpoly(x, y, degree=1, bandwidth=dpill(x,y))`。stats 包的 `ksmooth()` 函数也可以计算核回归。

当局部拟合的是一次或者二次多项式时，这种曲线拟合方法叫做局部多项式回归。对每个自变量 x 的值，给每对输入自变量、因变量 (x_i, y_i) 加权重 $\frac{1}{h}K(\frac{x-x_i}{h})$，然后拟合加权的线性回归或者二次多项式回归，得到 x 处的回归函数值作为 $f(x)$ 的估计。如果核函数 $K(\cdot)$ 是紧支集函数（仅在一个闭区间上非零），则每个 x 处仅距离 x 较近的观测值的权重非零。实际计算时，可以对 $\{x_i\}$ 取值范围内的一个等间隔格子点集合的 x 计算 $f(x)$ 的估计值。

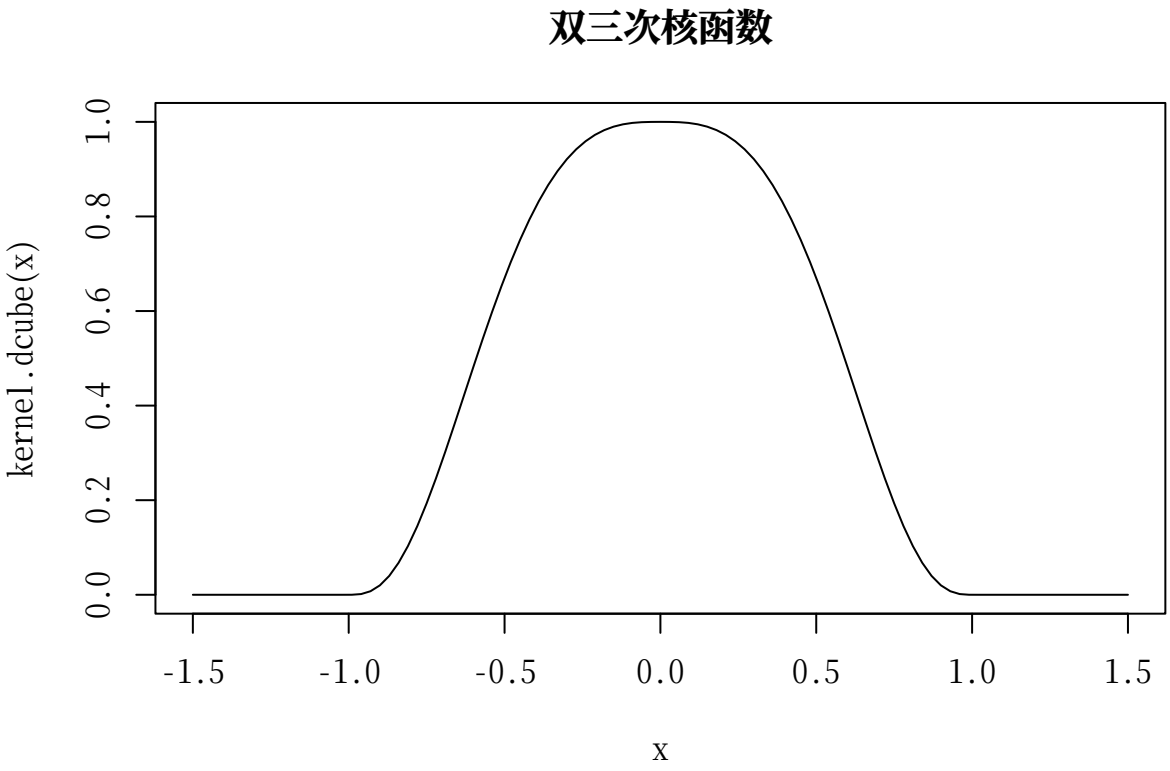


图 33.2: 双三次核函数

R 函数 `loess(y ~ x, degree=1, span=0.75)` 拟合局部线性函数，用 `span=` 控制结果的光滑度，起到了类似窗宽 h 的作用。`span` 是局部拟合所用的点的比例，取值于 $[0,1]$ 内，越接近于 1 结果越光滑。可以用指定 `span`，也可以用交叉验证方法确定 `span`，但 `loess()` 函数没有提供交叉验证功能。取 `degree=2` 拟合局部二次多项式函数。例子见图33.1。

33.4 样条函数回归

可以用指定节点集合的样条函数作为回归函数估计。m 个节点的三次样条函数需要 $m+4$ 个参数。因为每段需要 4 个参数，$m+1$ 段需要 $4m+4$ 个参数，而在 m 个节点上连续、一阶导数连续、二阶导数连续构成三个约束条件，所以参数个数为 $m+4$ 个。

自然样条函数假定函数在最左边一段和最右边一段为线性函数，这样 m 个节点需要 $m+2$ 个参数。R 的 `lm()` 函数中对自变量 `x` 指定 `ns(x)` 可以对输入的 `x` 指定作自然样条变换，在 `ns()` 中可以用 `df=` 指定自由度作为曲线复杂度的度量。

给定输入 (x_i, y_i), $i=1,2,\dots,n$ 后如何估计函数 $f(\cdot)$？计算时，用了基函数的思想将问题转化成了线性模型。根据 `df` 的值可以确定样条函数节点个数 K，然后可以选择 K 个节点 $z_k, k=1,2,\dots,K$，并令

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \sum_{k=1}^{K} \beta_{k+3} h(x, z_k),$$

其中

$$h(x,z) = \begin{cases} (x-z)^3, & x > z, \\ 0, & x \le z. \end{cases}$$

这样就可以用线性模型的估计方法估计出未知参数 $\beta_0, \dots, \beta_{K+3}$，从而得到样条函数估计。

如何决定节点个数 K 以及节点位置？节点越多，函数 $f(\cdot)$ 越复杂，计算量也越大。`lm()` 中 `ns()` 和 `bs()` 函数可以用 `df=` 选项指定一个等效自由度，等效自由度越大，模型越复杂，曲线光滑程度越低，`df` 值相当于多元线性回归中的自变量个数，决定了节点的个数。比如，`df=4` 的复杂度类似于有 4 个自变量的线性回归模型，它有 3 个内部节点，将 x 轴分为 4 段，拟合结果构成一个分 4 段的自然样条函数。不人为指定时，也可以通过交叉验证方法得到 `df` 的值。

节点的位置最好在回归曲线变化剧烈的位置多取，在变化缓慢的地方少取，但是这个要求很难实现，所以实际上是在输入的 x 变量值中取等分位间隔的 `df` 个点。

例如，下面的程序中用了自由度为 4 和自由度为 8 的样条函数进行回归估计：

```
set.seed(1)
nsamp <- 30
x <- runif(nsamp, -10, 10)
xx <- seq(-10, 10, length.out=100)
x <- sort(x)
y <- 10*sin(x/10*pi)^2 + rnorm(nsamp,0,0.3)
```

```
plot(x, y)
curve(10*sin(x/10*pi)^2, -10, 10, add=TRUE, lwd=2)

res <- lm(y ~ ns(x, df=4))
lines(xx, predict(res, newdata=data.frame(x=xx)),
      col="red")
res2 <- lm(y ~ ns(x, df=8))
lines(xx, predict(res2, newdata=data.frame(x=xx)),
      col="blue")
legend("top", lwd=c(2,1,1),
       col=c("black", "red", "blue"),
       legend=c(" 真实函数关系", "ns(df=4)", "ns(df=8)"))
```

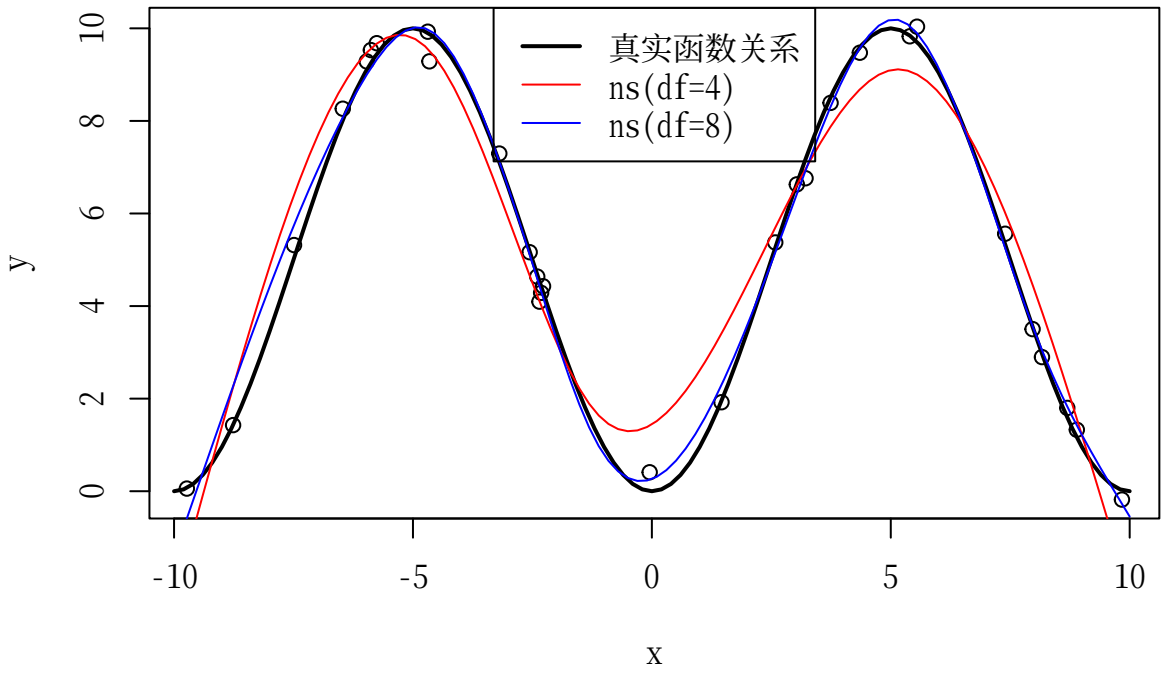


图 33.3: 自然样条回归示例

在多元回归中也可以用 `ns(x)` 对单个自变量引入非线性。

33.5 线性可加模型

虽然在用 `lm()` 作多元回归时可以用 `ns()`、`poly()` 等函数引入非线性成分，但需要指定复杂度参数。对可加模型

$$Y = \sum_{j=1}^{p} f_j(x_j) + \varepsilon$$

最好能从数据中自动确定 $f_j(\cdot)$ 的复杂度（光滑度）参数。

R 扩展包 mgcv 的 `gam()` 函数可以执行这样的可加模型的非参数回归拟合。模型中可以用 `s(x)` 指定 `x` 的样条平滑，用 `lo(x)` 指定 `x` 的局部多项式平滑。具体的计算是迭代地对每个自变量 x_j 进行，估计 x_j 的平滑函数 $f_j(\cdot)$ 时，采用数据 $\tilde{y} = Y - \sum_{k \neq j} f_k(x_k)$，迭代估计到结果基本不变为止。

可加模型的优点是可以对多个自变量进行非线性的模型估计，同时可加性使得每个自变量对因变量的解释很容易，缺点是不容易考虑自变量之间的交互作用效应。为了考虑交互作用效应可以使用基于树的随机森林或提升法 (boosting)。

例如，MASS 包的 rock 数据框是石油勘探中 12 块岩石样本分别产生 4 个切片得到的测量数据，共 48 个观测，因变量是渗透率 (permeability)，自变量为 area, peri, shape。

先作线性回归：

```
lm.rock <- lm(log(perm) ~ area + peri + shape, data=rock)
summary(lm.rock)
```

```
##
## Call:
## lm(formula = log(perm) ~ area + peri + shape, data = rock)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.8092 -0.5413  0.1734  0.6493  1.4788
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.333e+00  5.487e-01   9.720 1.59e-12 ***
## area         4.850e-04  8.657e-05   5.602 1.29e-06 ***
## peri        -1.527e-03  1.770e-04  -8.623 5.24e-11 ***
## shape        1.757e+00  1.756e+00   1.000    0.323
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8521 on 44 degrees of freedom
## Multiple R-squared:  0.7483, Adjusted R-squared:  0.7311
```

```
## F-statistic:  43.6 on 3 and 44 DF,  p-value: 3.094e-13
```

其中 shape 变量不显著。可能的原因有：

- shape 与因变量没有关系；
- 样本量不足；
- shape 与因变量之间有非线性关系，该非线性无法用线性近似。

用样条平滑的 gam() 建模：

```
library(mgcv)
```

```
## Loading required package: nlme

##
## Attaching package: 'nlme'

## The following object is masked from 'package:dplyr':
##
##     collapse

## This is mgcv 1.8-40. For overview type 'help("mgcv-package")'.
```

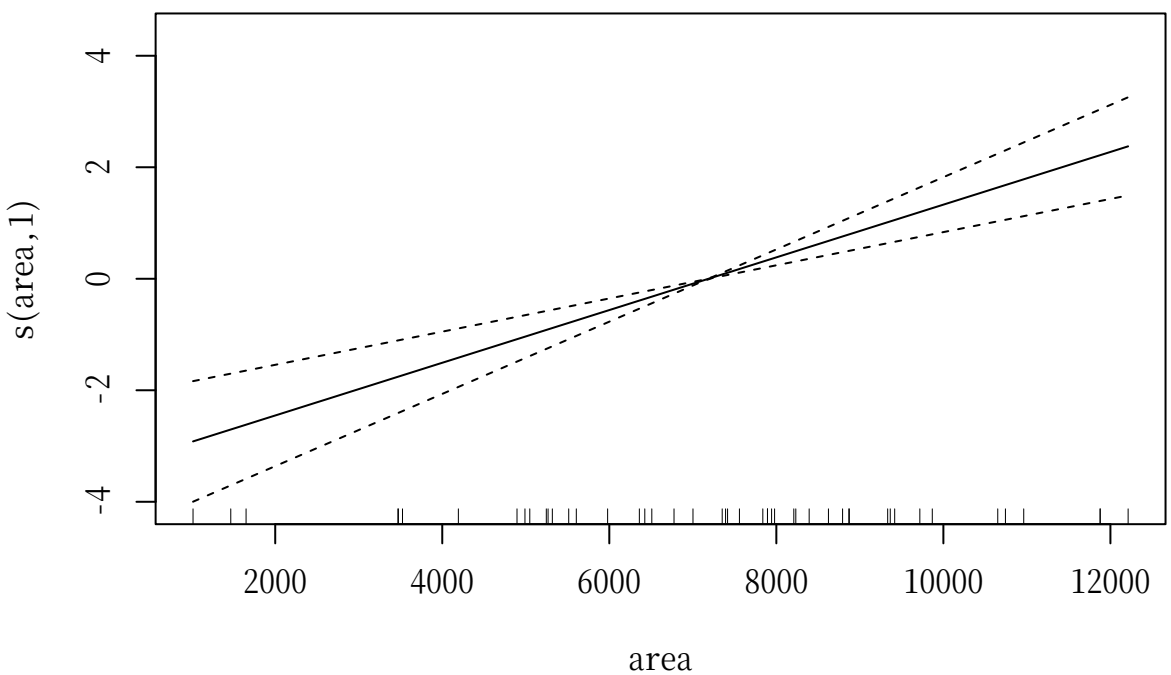
```
gam.rock1 <- gam(
  log(perm) ~ s(area) + s(peri) + s(shape), data=rock)
summary(gam.rock1)
```

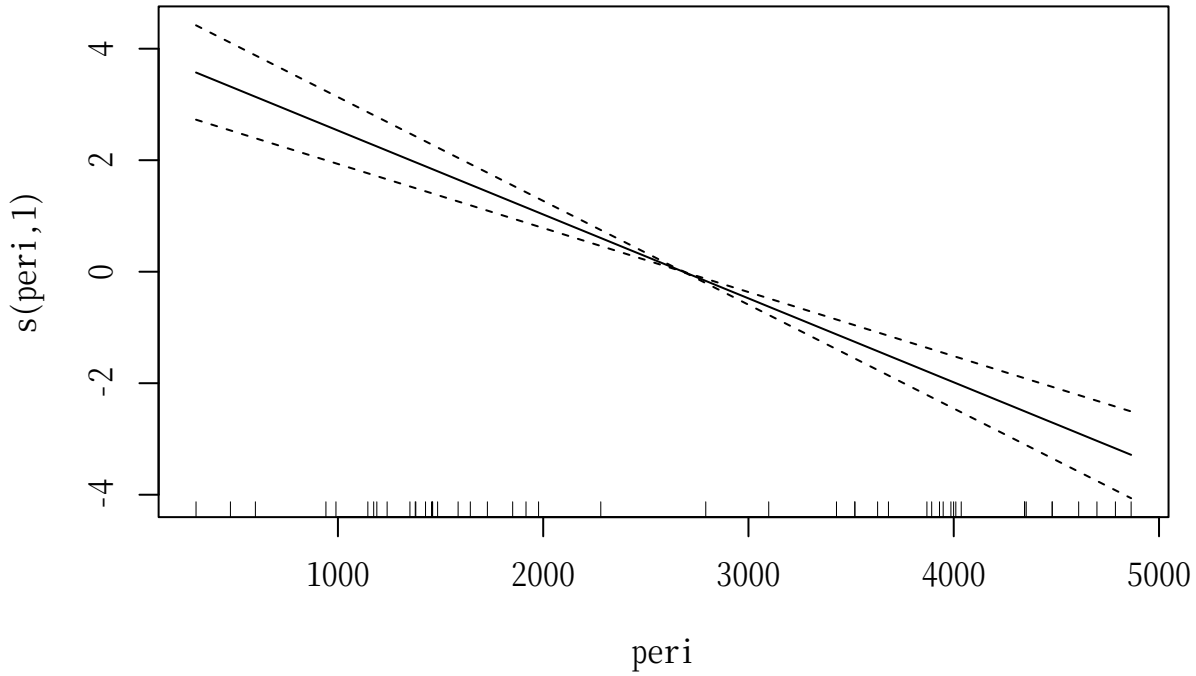
```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## log(perm) ~ s(area) + s(peri) + s(shape)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   5.1075     0.1222   41.81   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

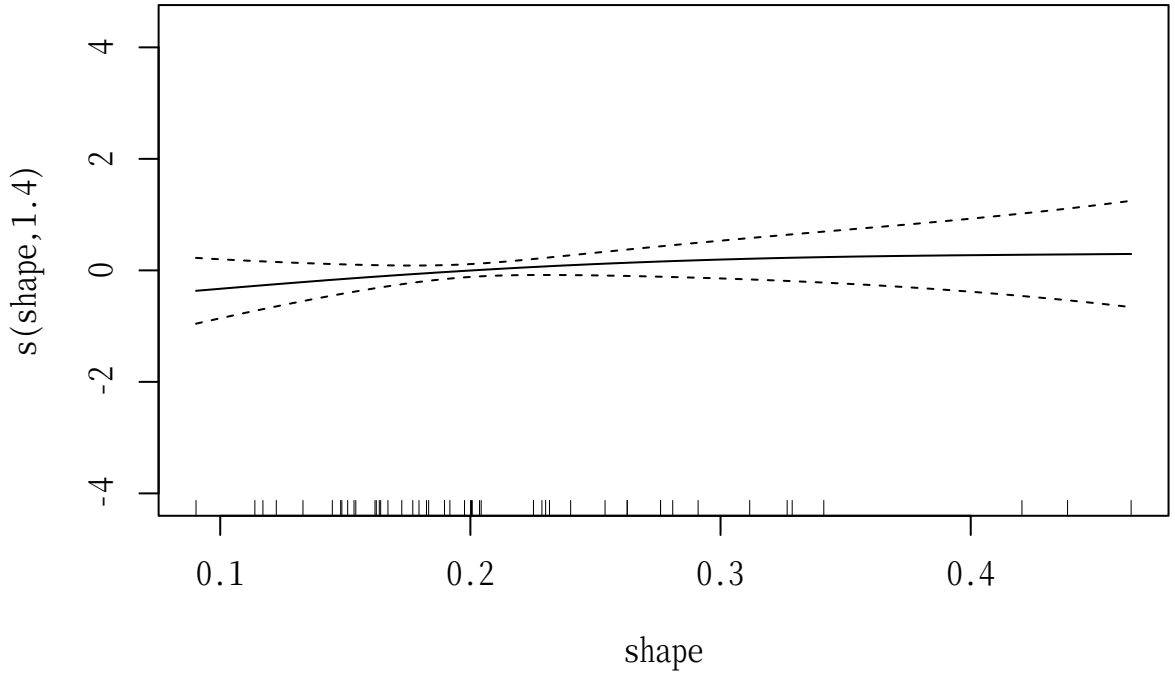
```
## Approximate significance of smooth terms:
##             edf Ref.df     F  p-value
## s(area)  1.000  1.000 29.13 3.07e-06 ***
## s(peri)  1.000  1.000 71.30  < 2e-16 ***
## s(shape) 1.402  1.705  0.58    0.437
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.735   Deviance explained = 75.4%
## GCV = 0.78865  Scale est. = 0.71631   n = 48
```

对 gam() 回归结果做单个变量的曲线拟合图：

```
plot(gam.rock1)
```




s(peri,1)
4
2
0
-2
-4
1000
2000
3000
4000
5000
peri


s(shape,1.4)
4
2
0
-2
-4
0.1
0.2
0.3
0.4
shape

其中对应每个自变量的曲线表示控制其它自变量情况下因变量与此自变量的关系。可以看出三条曲线都基本不需要非线性。比较两个模型：

```
anova(lm.rock, gam.rock1)
```

```
## Analysis of Variance Table
##
## Model 1: log(perm) ~ area + peri + shape
## Model 2: log(perm) ~ s(area) + s(peri) + s(shape)
##   Res.Df    RSS      Df Sum of Sq      F Pr(>F)
## 1 44.000 31.949
## 2 43.598 31.230 0.40237   0.71914 2.4951  0.125
```

结果也不显著，说明线性模型是可取的。

Chapter 34

R 方差分析

34.1 单因素方差分析

单因素方差分析可以看成基础统计中两样本 t 检验的一个推广，要比较试验观测值的某个因变量（称为“指标”）按照一个分组变量（称为“因素”）分组后，各组的因变量均值有无显著差异。

设因素 A 将所有观测分为 m 个组，每组对因变量进行 r 次观测，且各次观测相互独立，模型为

$$\begin{aligned} y_{ij} =& \mu_i + e_{ij}, i = 1, 2, \dots, m,\ j = 1, 2, \dots, r. \\ & e_{ij} \text{ iid } \mathrm{N}(0, \sigma^2). \end{aligned}$$

希望研究如下问题：

$$H_0:\ \mu_1 = \mu_2 = \cdots = \mu_m.$$

如果拒绝 H_0，希望找出哪些组的均值两两之间是有显著差异的。

这样的问题称为单因素**方差分析**问题，“单因素” 是指按照一种分组方式分组。主要模型中假定各个观测独立，误差 e_{ij} 服从正态分布，且误差的方差相等。

方差分析在最初是采用平方和分解的方式进行检验，但是随着模型变得更复杂，现在处理方差分析问题一般都采用线性模型的一般理论进行处理。

单因素方差分析问题可以推广到不同组的重复观测次数不同的情形。

34.1.1 单因素方差分析示例

例子：考虑如下的方差分析问题。某工厂要比较三种不同组装工艺 (A, B, C) 的工作效率，将 15 名工人随机分为 3 个组，每个组采用一种工艺，一周后各组的成品数如下：

A	B	C
58	58	48
64	69	57
55	71	59
66	64	47
67	68	49

我们需要长表格式，进行转换：

```
d.manu <- d.manu.w |>
  pivot_longer(A:C,
    names_to = "grp",
    values_to = "y") |>
  mutate(grp = factor(grp, levels=c("A", "B", "C"))) |>
  arrange(grp)
knitr::kable(d.manu)
```

grp	y
A	58
A	64
A	55
A	66
A	67
B	58
B	69
B	71
B	64
B	68
C	48
C	57
C	59
C	47
C	49

用 aov() 进行方差分析：

```
aov.manu <- aov(y ~ grp, data=d.manu)
summary(aov.manu)
```

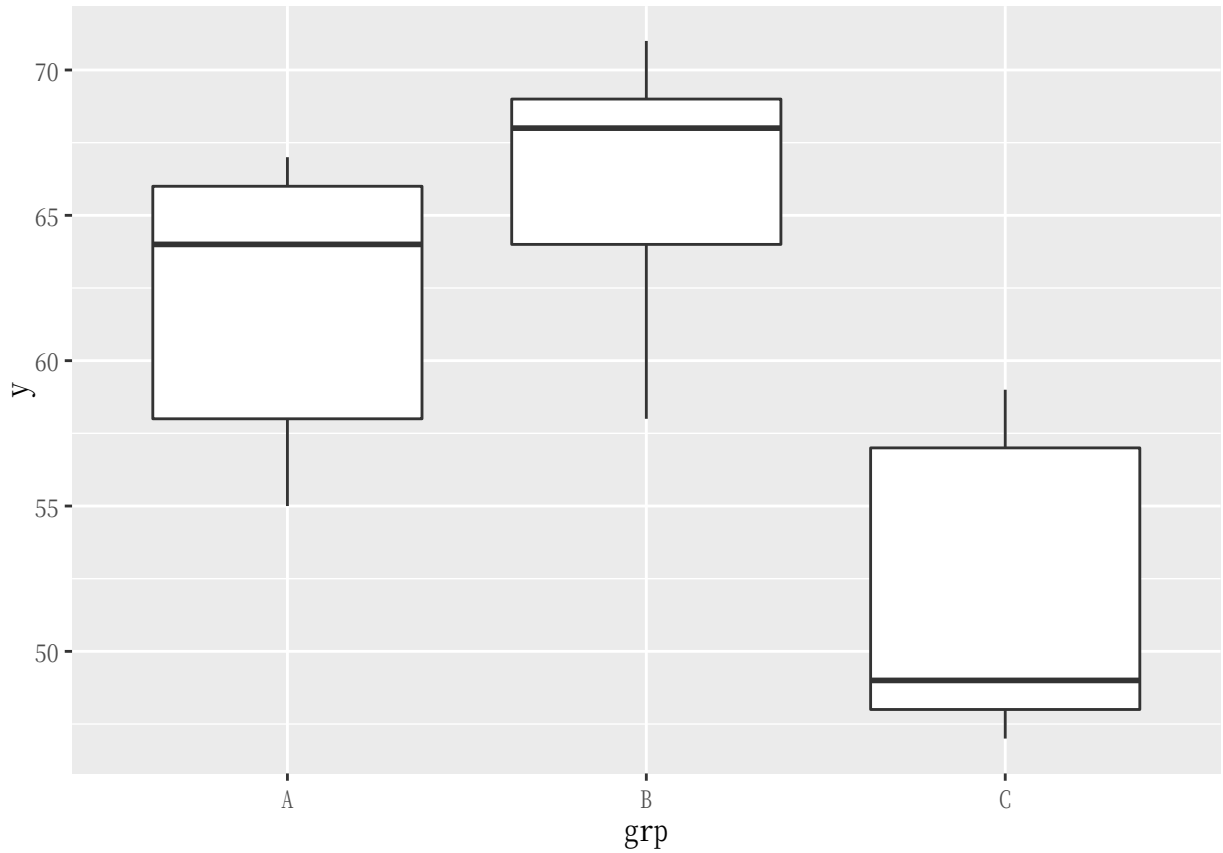
```
##             Df Sum Sq Mean Sq F value  Pr(>F)
## grp          2    520  260.00   9.176 0.00382 **
```

```
## Residuals   12    340   28.33
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

主效应 grp（分组）的 F 检验的 p 值为 0.004，若检验水平为 0.05 则分组效应显著，各组之间有显著差异。

可以用并列盒形图比较各组的取值，并检查分布对称性和方差齐性：

```
ggplot(data = d.manu, mapping = aes(
  x = grp, y = y)) +
  geom_boxplot()
```



从图形看，C 组的工作效率显著低于 A 组和 B 组，A、B 两组之间差异不明显。

为了检查各组方差是否相等，可以进行 Bartlett 检验：

```
bartlett.test(y ~ grp, data = d.manu)
```

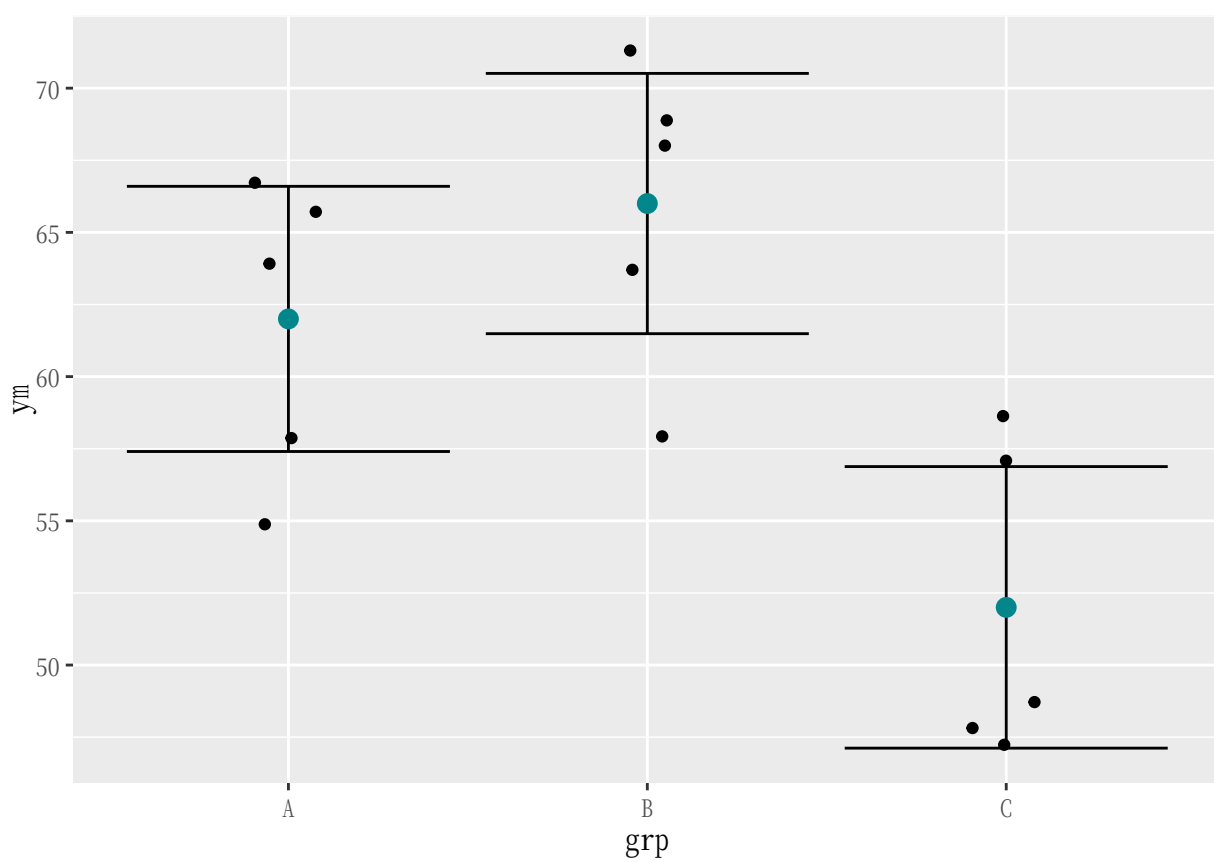
```
##
##  Bartlett test of homogeneity of variances
##
```

```
## data:  y by grp
## Bartlett's K-squared = 0.024476, df = 2, p-value = 0.9878
```

结果表明可以认为各组的方差相等。

除了各组并列盒形图，也可以做点图并将各组的均值、均值置信区间作图，如：

```
d2.manu <- d.manu |>
  group_by(grp) |>
  summarise(ym = mean(y),
    ystd = sd(y),
    yrad = 1.96 * ystd / sqrt(n()))
ggplot(data=d2.manu) +
  geom_errorbar(data = d2.manu,
    mapping = aes(
      x = grp, ymin = ym - yrad, ymax = ym + yrad  )) +
  geom_point(mapping = aes(
    x = grp, y = ym ), size=3, color="turquoise4") +
  geom_jitter(data=d.manu, mapping=aes(
    x = grp, y = y ), width=0.1)
```



可以看出 C 的误差条（均值置信区间）低于 A 和 B 的误差条，而 A 和 B 的误差条有交集。

34.1.2 功效与样本量计算

对均衡设计的单因素方差分析，可以用 pwr 包的 `pwr.anova.test()` 函数计算功效和样本量。设 k 个组的因变量（指标）值共同的标准差为 σ_{within}，而各个组的均值的样本标准差为 σ_{between}(但是使用 k 作为分母而不是 $k-1$ 作为分母)，定义要检验的效应大小为

$$f = \frac{\sigma_{\text{between}}}{\sigma_{\text{within}}}.$$

典型的效应大小值为：

- 小：0.1;
- 中: 0.25;
- 大: 0.4。

考虑前面比较三种工艺的例子，取检验水平 0.05，每组样本量 $n = 5$，要检验中等的效应大小，功效为：

```
library(pwr)
```

```
## Warning: package 'pwr' was built under R version 4.2.2
```

```
cohen.ES(test = "anov", size = "medium")
```

```
##
##      Conventional effect size from Cohen (1982)
##
##            test = anov
##            size = medium
##     effect.size = 0.25
```

```
pwr.anova.test(
  k = 3,    # 分组数
  f = 0.25, # 效应大小
  sig.level = 0.05,
  n = 5 )
```

```
##
##      Balanced one-way analysis of variance power calculation
##
##               k = 3
```

```
##                n = 5
##                f = 0.25
##        sig.level = 0.05
##            power = 0.1095297
##
## NOTE: n is number in each group
```

功效只有 11%。计算达到 80% 功效所需样本量：

```
library(pwr)
cohen.ES(test = "anov", size = "medium")
```

```
##
##      Conventional effect size from Cohen (1982)
##
##             test = anov
##             size = medium
##     effect.size = 0.25
```

```
pwr.anova.test(
  k = 3,    # 分组数
  f = 0.25, # 效应大小
  sig.level = 0.05,
  power = 0.80 )
```

```
##
##      Balanced one-way analysis of variance power calculation
##
##                k = 3
##                n = 52.3966
##                f = 0.25
##        sig.level = 0.05
##            power = 0.8
##
## NOTE: n is number in each group
```

需要每组 53 个试验值。

我们计算一下比较工艺例子中实际数据反映出来的效应大小。

```
sigs <- d.manu |>
  group_by(grp) |>
  summarise(
    ng = n(),
    gmean = mean(y),
    gstd = sd(y),
    .groups = "drop") |>
  summarise(
    sig_within = sqrt(sum(gstd^2) / 3),
    sig_between = sd(gmean) * sqrt(2/3)  )
fv <- sigs |>
  mutate(f = sig_between / sig_within) |>
  pull()
fv
```

```
## [1] 1.106133
```

按这个效应计算检验的功效：

```
pwr.anova.test(
  k = 3,    # 分组数
  f = fv,   # 效应大小
  sig.level = 0.05,
  n = 5 )
```

```
##
##      Balanced one-way analysis of variance power calculation
##
##               k = 3
##               n = 5
##               f = 1.106133
##       sig.level = 0.05
##           power = 0.9292112
##
## NOTE: n is number in each group
```

功效有 93%。计算 80% 功效需要的样本量：

```
pwr.anova.test(
  k = 3,    # 分组数
  f = fv,   # 效应大小
  sig.level = 0.05,
  power = 0.80 )
```

```
##
##      Balanced one-way analysis of variance power calculation
##
##               k = 3
##               n = 3.82216
##               f = 1.106133
##       sig.level = 0.05
##           power = 0.8
##
## NOTE: n is number in each group
```

每组 4 各试验即可。

基本 R 的 power.anova.test() 也能计算功效，需要输入组内方差、组间方差，而且计算组间方差时用的是各组均值方差使用 $k-1$（组数）作为分母的公式。如：

```
power.anova.test(
  groups = 3,
  within.var = sigs$sig_within^2,
  between.var = sigs$sig_between^2 * 3 / 2,
  sig.level = 0.05,
  n = 5 )
```

```
##
##      Balanced one-way analysis of variance power calculation
##
##          groups = 3
##               n = 5
##     between.var = 52
##      within.var = 28.33333
##       sig.level = 0.05
##           power = 0.9292112
##
## NOTE: n is number in each group
```

和 pwr.anova.test() 输出的结果相同。

34.1.3 多重比较

为了找到各组两两之间是否有显著差异，可以进行两两的独立两样本 t 检验，但这样不能利用共同的模型参数，进行多次重复检验也会使得总第一类错误概率变得比较高，发生过度拟合。

进行多个假设检验（如均值比较）的操作称为“多重比较”（multiple comparison，或 multiple testing），多次检验会使得总第一类错误概率增大。以两两比较为例，当比较次数很多时，即使所有的组之间都没有真实的差异，很多个两两比较也会找到差异最大的一对，使得发现的的显著差异有很大可能性不是真实存在的。

为此，可以进行一些调整，使得报告的检验 p 值能够控制总第一类错误概率。multcomp 包的 `glht()` 函数可以对方差分析结果进行多重比较并控制总错误率，一种方法是利用 Tukey 的 HSD(Honest Significant Difference) 方法，程序如下：

```
library(multcomp, quietly=TRUE)
aov.manu <- aov(y ~ grp, data=d.manu)
glht(aov.manu, linfct = mcp(grp = "Tukey")) |>
  summary()
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Multiple Comparisons of Means: Tukey Contrasts
##
##
## Fit: aov(formula = y ~ grp, data = d.manu)
##
## Linear Hypotheses:
##            Estimate Std. Error t value Pr(>|t|)
## B - A == 0    4.000      3.367   1.188  0.48200
## C - A == 0  -10.000      3.367  -2.970  0.02899 *
## C - B == 0  -14.000      3.367  -4.159  0.00357 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

Tukey HSD 检验的结果是，在 0.05 水平下，A 和 B 没有显著差异，C 与 A、B 均有显著差异。

34.1.4 方差不相等情形

方差分析模型要求误差项独立同正态分布 $N(0, \sigma^2)$，这意味着各组的因变量方差相等。实际中不同组的因变量可能有不同的方差。R 提供了 `oneway.test()` 函数作为独立两样本 t 检验的 Welch 方法的推广，可以不要求方差相等。如

```
oneway.test(y ~ grp, data=d.manu)
```

```
##
##  One-way analysis of means (not assuming equal variances)
##
## data:  y and grp
## F = 8.1976, num df = 2.0000, denom df = 7.9914, p-value = 0.01159
```

p 值为 0.011，在 0.05 水平下显著，说明三种生产工艺的时间有显著差异。

为了进行多重比较，可以进行两两 t 检验并不使用合并的标准差估计，使用 Holm 方法进行 p 值调整以控制总错误率：

```
with(d.manu,
  pairwise.t.test(y, grp, pool.sd=FALSE,
    p.adjust.method="holm"))
```

```
##
##  Pairwise comparisons using t tests with non-pooled SD
##
## data:  y and grp
##
##   A     B
## B 0.258 -
## C 0.039 0.010
##
## P value adjustment method: holm
```

在 0.05 水平下，A 和 B 没有显著差异，C 与 A 和 B 都有显著差异。

34.1.5 非参数方差分析

如果各组的因变量（指标）分布严重偏离正态，则单因素方差分析所依据的 F 检验会有很大的误差。这时，可以使用非参数方法，Kruskal-Wallis 检验是独立两样本比较的 Wilcoxon 秩和检验的推广。如：

```
kruskal.test(y ~ grp, data=d.manu)
```

```
##
##  Kruskal-Wallis rank sum test
##
```

```
## data:  y by grp
## Kruskal-Wallis chi-squared = 7.7677, df = 2, p-value = 0.02057
```

检验 p 值为 0.02，在 0.05 水平下拒绝零假设，认为各组之间有显著差异。

34.2 协方差分析

在单因素方差分析问题中，有可能存在一个连续型的解释变量，对因变量（指标）有影响，希望在排除此连续型解释变量影响后对各组进行比较。这相当于如下的模型：

$$y_{ij} = \mu_i + \beta x_{ij} + e_{ij},$$

其中 x_{ij} 为已知值的解释变量。

例子考虑 multcomp 包的 litter 数据。为研究某种药物对出生小数体重的影响，将若干怀孕小鼠随机分配到 4 个不同的处理组，变量 `dose` 表示这 4 个组，其取值是母鼠服药的剂量。研究的因变量（指标）是母鼠所生产的幼鼠的平均体重 (weight)，要排除协变量 `gesttime`（怀孕时间）的影响。这个数据的各组试验数不相同，频数统计为：

```r
data(litter, package="multcomp")
litter |>
  count(dose) |>
  knitr::kable()
```

dose	n
0	20
5	19
50	18
500	17

进行协方差分析：

```r
aov.lit1 <- aov(weight ~ gesttime + dose,
  data = litter)
summary(aov.lit1)
```

```
##             Df Sum Sq Mean Sq F value  Pr(>F)
## gesttime     1  134.3  134.30   8.049 0.00597 **
## dose         3  137.1   45.71   2.739 0.04988 *
## Residuals   69 1151.3   16.69
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

从结果中 dose 的检验结果看，排除协变量 gesttime 影响后，在 0.05 水平下，四个处理组之间有显著差异。

将服药的三个组的均值与不服药（dose=0）的组比较的检验如下：

```
contr <- rbind(" 服药与不服药比较" = c(3, -1, -1, -1))
glht(aov.lit1, linfct = mcp(dose = contr)) |>
  summary()
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Multiple Comparisons of Means: User-defined Contrasts
##
##
## Fit: aov(formula = weight ~ gesttime + dose, data = litter)
##
## Linear Hypotheses:
##                        Estimate Std. Error t value Pr(>|t|)
## 服药与不服药比较 == 0    8.284      3.209   2.581    0.012 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

检验的零假设为 $H_0: \alpha_1 = (\alpha_2 + \alpha_3 + \alpha_4)/3$, 其中 α_j 表示第 j 个组的均值，α_1 对应于 dose=0 的组，即不服药的组。检验结果表明在 0.05 水平下，服药的和不服药的指标均值有显著差异。

34.3 两因素方差分析

两因素方差分析，最简单的情形是有两个分组变量（称为因素）A 和 B，A 有 s 个分类，B 有 t 个分类，对 A 和 B 的每一种搭配 (i, j)，重复试验 r 次得到观测值 y_{ijk}，设各次观测值相互独立。模型为

$$y_{ijk} = \mu_{ij} + e_{ijk},\ i = 1, \dots, s,\ j = 1, \dots, t,\ k = 1, \dots, r,$$

误差项 e_{ijk} 独立同正态分布 N(0,σ^2)。

一般将上述模型用另外的参数表示成：

$$y_{ijk} = \mu + \alpha_i + \beta_j + \gamma_{ij} + e_{ijk},$$

其中 α_i 表示因素 A 的“主效应”，β_j 表示因素 B 的主效应，γ_{ij} 表示因素 A 和因素 B 的交互作用效应。这样模型中的参数是冗余的，一般会加限制，比如设 $\alpha_1 = 0$，$\beta_1 = 0$，$\gamma_{i1} = \gamma_{1j} = 0$。

二元方差分析的问题是分别检验两个因素的主效应是否显著（不等于零），交互作用效应是否存在（不等于零）。

34.3.1 两因素方差分析计算示例

以 R 的 datasets 包的 ToothGrowth 数据为例。考虑维生素 C 对豚鼠的成牙质细胞生长的影响，因变量（指标）`len` 是某牙齿长度测量值，考虑两种不同的维生素 C 类型 (变量 `supp`，取值为 OJ 或 VC)，以及三种剂量 (变量 `dose`, 取值为 0.5, 1.0, 2.0)。需要将这两个因素都转换成 R 的因子：

```
data(ToothGrowth, package="datasets")
d.tooth <- ToothGrowth |>
  mutate(supp = factor(supp, levels=c("OJ", "VC")),
         dose = factor(dose, levels=c(0.5, 1.0, 2.0)))
d.tooth |>
  count(supp, dose)
```

```
##   supp dose  n
## 1   OJ  0.5 10
## 2   OJ    1 10
## 3   OJ    2 10
## 4   VC  0.5 10
## 5   VC    1 10
## 6   VC    2 10
```

可见这个试验是均衡有重复完全试验设计。

进行二元方差分析：

```
aov.to1 <- aov(len ~ dose + supp + dose:supp,
  data = d.tooth)
summary(aov.to1)
```

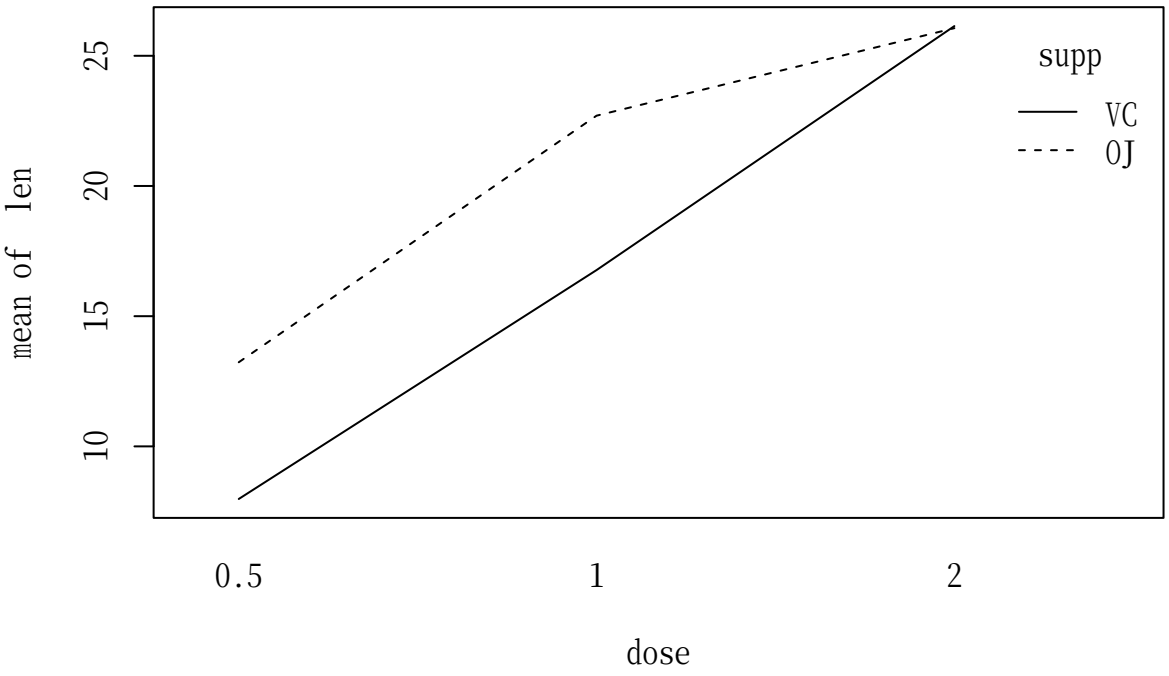
```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## dose         2 2426.4  1213.2  92.000  < 2e-16 ***
## supp         1  205.4   205.4  15.572 0.000231 ***
## dose:supp    2  108.3    54.2   4.107 0.021860 *
## Residuals   54  712.1    13.2
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

在 0.05 水平下，两个主效应和交互作用效应都显著。

34.3.2 交互作用图

R 函数 `interaction.plot()` 可以绘制表示交互作用的图形。此图形以因变量为 y 坐标，以第一个因素的水平为横坐标作折线图，但根据第二个因素的不同水平分组作图。如果不同组的折线图基本平行则没有交互作用效应，否则提示有交互作用效应。程序如：

```
with(d.tooth,
  interaction.plot(dose, supp, len) )
```



图形结果提示有交互作用效应。

Chapter 35

线性混合模型

35.1 介绍

在基本回归分析模型中，假定回归误差项独立同分布，方差相等，还经常假定误差项服从正态分布。在实际应用回归分析建模时，还经常遇到模型误差项方差不相等或者误差项之间不独立的情形。比如，如果每个观测来自于一个群体的平均值，设群体中的个体方差相同，则每个观测的误差方差正比于群体中个体的个数的倒数，不等于常数。又比如，医学研究中每个受试者有多次随访的数值型观测值，则每个受试者的各次测量值是相关的，不同受试者之间可以认为是独立的，这样模型误差不是相互独立的。

考虑如下的线性模型：

$$Y = X\beta + \varepsilon, \tag{35.1}$$

$$\varepsilon \sim \mathrm{N}(0, \sigma^2 W^{-1}). \tag{35.2}$$

其中 X 为 $n \times p$ 非随机的自变量矩阵，β 是 p 维回归系数向量，ε 为 n 维随机误差向量，W 已知，σ^2 未知。

因为标准的线性模型要求 $W = I$，所以这个模型是线性模型的推广。

考虑这个模型的估计问题。设存在矩阵 A 使得 $W^{-1} = AA^T$，则

$$A^{-1}Y = A^{-1}X\beta + A^{-1}\varepsilon,$$

令 $Y^* = A^{-1}Y$, $X^* = A^{-1}X$, $\varepsilon^* = A^{-1}\varepsilon$，则

$$Y^* = X^*\beta + \varepsilon^*,$$

其中 $\mathrm{Var}(\varepsilon^*) = \sigma^2 I$。在矩阵可逆条件下 β 的最小二乘估计为

$$\hat{\beta} = (X^{*T}X^*)^{-1}X^{*T}Y^* = (X^TWX)^{-1}X^TWY.$$

这个公式称为加权最小二乘公式。

模型可以进一步推广。考虑

$$Y = X\beta + Z\alpha + \varepsilon, \tag{35.3}$$

$$\varepsilon \sim \mathrm{N}(0, \sigma^2 W^{-1}), \tag{35.4}$$

$$\alpha \sim \mathrm{N}(0, \Sigma). \tag{35.5}$$

其中 Y 为可观测的 n 维因变量向量，$X_{n\times p}$, $Z_{n\times q}$ 已知，$\beta_{p\times 1}$ 为未知的非随机回归系数向量，称为**固定效应**；$\alpha_{q\times 1}$ 为未知的随机向量，称为**随机效应**，Σ 未知；ε 为未知的随机误差向量，σ^2 未知，W 已知。称此模型为**线性混合模型**。

在估计模型时，Σ 必须是对称非负定阵，一般有一定的参数结构，所以一般会分解为

$$\Sigma = \sigma^2 \Lambda_\theta \Lambda_\theta^T,$$

其中 θ 为待估参数。

因为随机效应 α 未知但一般不需要估计，所以可以将其作用混合到因变量 Y 的方差结构中，得到

$$\begin{aligned} EY =& X\beta, \\ \mathrm{Var}(Y) =& \sigma^2 W^{-1} + Z\Sigma Z^T = \sigma^2 (W^{-1} + (Z\Lambda_\theta)(Z\Lambda_\theta)^T). \end{aligned}$$

线性混合效应模型适用的数据包括：

- 集簇数据，比如，同一学科的不同授课教师的学生成绩；
- 重复量测，比如多个受试者每人都多次测量某一指标；
- 纵向数据，比如多个病人的多次不同时间的随访观测；
- 多元观测，比如多个病人每人都有多个生理指标。

本章理论和例子参考了 (Andrzej Galecki, 2013)。

本章主要使用 nmleU 包的 armd 数据。ARMD（年龄相关性黄斑变性）是一种老年人眼科疾病，会使得患病的人逐渐失明。为了研究某种新药的疗效，收录了 240 位病人，随机分为治疗组与对照组，并在入组时以及 4、12、24、52 周时对每位病人测量视力指标。这样的数据称为纵向数据，每位病人都测量了 5 个不同时间的因变量值，但这 5 个时间是共同的。有些病人的随访有缺失，并且有些是在中间缺失的。每个病人的各次视力指标都与入组时的基础测量值有关。

```
data(armd, package="nlmeU")
str(armd)
```

```
## 'data.frame':    867 obs. of  8 variables:
##  $ subject : Factor w/ 234 levels "1","2","3","4",..: 1 1 2 2 2 2 3 3 3 4 ...
##  $ treat.f : Factor w/ 2 levels "Placebo","Active": 2 2 2 2 2 2 1 1 1 1 ...
##  $ visual0 : int  59 59 65 65 65 65 40 40 40 67 ...
##  $ miss.pat: Factor w/ 8 levels "----","---X",..: 4 4 1 1 1 1 2 2 2 1 ...
```

```
##  $ time.f  : Ord.factor w/ 4 levels "4wks"<"12wks"<..: 1 2 1 2 3 4 1 2 3 1 ...
##   ..- attr(*, "contrasts")= num [1:4, 1:3] -0.5222 -0.3023 0.0275 0.797 0.565 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:4] "4wks" "12wks" "24wks" "52wks"
##   .. .. ..$ : chr [1:3] ".L" ".Q" ".C"
##  $ time    : num  4 12 4 12 24 52 4 12 24 4 ...
##  $ visual  : int  55 45 70 65 65 55 40 37 17 64 ...
##  $ tp      : num  1 2 1 2 3 4 1 2 3 1 ...
```

35.2 加权线性回归

35.2.1 加权已知情形

对于某些回归问题的数据，各个观测的因变量方差不相等。设模型为：

$$\begin{aligned} y_i =& \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i, \\ & \varepsilon_i\text{相互独立，零均值，方差为}\frac{\sigma^2}{w_i},\ i = 1, 2, \ldots, n. \end{aligned}$$

这个模型违反了线性回归模型要求误差项方差相等的要求，但可以通过简单的变换变成标准的线性回归问题。令 $y_i^* = \sqrt{w_i} y_i$, $x_{ij}^* = \sqrt{w_i} x_{ij}$, $\varepsilon_i^* = \sqrt{w_i}\varepsilon_i$，则模型变成：

$$\begin{aligned} y_i^* =& \beta_0 + \beta_1 x_{i1}^* + \cdots + \beta_p x_{ip}^* + \varepsilon_i^*, \\ & \varepsilon_i^*\text{相互独立，零均值，方差为}\sigma^2,\ i = 1, 2, \ldots, n. \end{aligned}$$

于是最小二乘估计为：

$$\begin{aligned} & \min \sum_{i=1}^n \left[y_i^* - \left(\beta_0 + \beta_1 x_{i1}^* + \cdots + \beta_p x_{ip}^* \right) \right]^2 \\ =& \min \sum_{i=1}^n w_i \left[y_i - \left(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} \right) \right]^2, \end{aligned}$$

用向量记号表示的估计公式为

$$\widehat{\beta} = (X^T W X)^{-1} X^T W Y,$$

其中 $W = \mathrm{diag}(w_1, \ldots, w_n)$。

常见的加权线性回归问题是每个观测都是某一类数据的平均值，已知参与平均的个数但没有具体的原始数据。在 `lm()` 函数中用 `weights=` 指定权重，如果已知每个观测是 `n` 个观测的平均，可以指定 `weights=n`，用数据集中的变量 `n` 保存每个观测代表的原始观测个数。

考虑 nlmeU 包的 armd 数据框。共有 240 为受试者分为处理组和对照组，在第 4、12、24、52 周的安排了视力的随访测量，这些随访的测量值可能有缺失。将每个受试者的随访测量值进行平均，利用视力的平均测量值比较处理组与对照组，以基线的视力测量值为协变量，进行加权回归时以参与平均的测量值个数作为权重（这是一种近似，因为有序列相关可能）。程序如下：

```
data(armd, package="nlmeU")
```

```
dar <- armd |>
  group_by(subject) |>
  mutate(meanvis = mean(visual, na.rm=TRUE),
         nfu = sum(!is.na(visual))) |>
  filter(nfu > 0)
lm1.armd <- lm(meanvis ~ treat.f + visual0,
  weights = nfu, data=dar)
summary(lm1.armd)
```

```
##
## Call:
## lm(formula = meanvis ~ treat.f + visual0, data = dar, weights = nfu)
##
## Weighted Residuals:
##     Min      1Q  Median      3Q     Max
## -52.506 -12.042   3.502  13.797  43.560
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    4.34194    1.31211   3.309 0.000975 ***
## treat.fActive -3.27536    0.66054  -4.959 8.55e-07 ***
## visual0        0.83199    0.02227  37.354  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 19 on 864 degrees of freedom
## Multiple R-squared:  0.6233, Adjusted R-squared:  0.6224
## F-statistic: 714.8 on 2 and 864 DF,  p-value: < 2.2e-16
```

加权回归也可以用 nlme 包的 gls() 函数计算，gls() 可以进行误差项方差可变或者相关的广义最小二乘估计。如：

```
library(nlme)
```

```
##
## Attaching package: 'nlme'

## The following object is masked from 'package:dplyr':
```

```
##
##     collapse
```

```r
lm2.armd <- gls(meanvis ~ treat.f + visual0,
  weights = varFixed(value = ~ I(1/nfu)), data=dar)
summary(lm2.armd)
```

```
## Generalized least squares fit by REML
##   Model: meanvis ~ treat.f + visual0
##   Data: dar
##        AIC      BIC    logLik
##   6422.203 6441.249 -3207.102
##
## Variance function:
##  Structure: fixed weights
##  Formula: ~I(1/nfu)
##
## Coefficients:
##                   Value Std.Error  t-value p-value
## (Intercept)    4.341935 1.3121127  3.30912   0.001
## treat.fActive -3.275356 0.6605417 -4.95859   0.000
## visual0        0.831994 0.0222731 37.35417   0.000
##
##  Correlation:
##               (Intr) trt.fA
## treat.fActive -0.260
## visual0       -0.938  0.024
##
## Standardized residuals:
##        Min         Q1        Med         Q3        Max
## -2.7637685 -0.6338519  0.1843284  0.7262547  2.2928684
##
## Residual standard error: 18.99789
## Degrees of freedom: 867 total; 864 residual
```

其中 `weights()` 选项可以指定各种表示方差结构的函数，这里的 `varFixed()` 函数指定一个自变量作为误差方差的倍数，是权重的倒数。

35.2.2 方差依赖于均值的情形

比上述加权已知的情形略复杂一些的，是 ε_i 的方差为如下函数：

$$\lambda_i = \sigma^2 \lambda(\mu_i),$$

其中 $\mu_i = x^i\beta$，x^i 表示第 i 个观测的自变量。这时，可以用 IRLS(Iteratively Re-Weighted Least Square) 算法估计模型参数：

1. 给定 β 的初值 $\beta^{(0)}$，令迭代计数器 $k=0$；
2. 令 $\mu_i^{(k)} = x^i\beta^{(k)}$，$\hat{\lambda}_i^{(k)} = \lambda(\mu_i^{(k)})$;
3. 取权重 $w_i = 1/\hat{\lambda}_i^{(k)}$，用加权最小二乘方法估计参数 β，作为 $\beta^{(k+1)}$；
4. 令 $k \leftarrow k+1$;
5. 重复第 2 至第 4 步直到收敛。
6. 计算最终的权重，用加权最小二乘方法估计参数 σ^2。

nmle 包的 `gls()` 函数支持这样的方差结构，并可以进行参数估计和推断。

35.3 线性混合模型

35.3.1 问题介绍

在医学研究等领域问题中，经常收集多个受试者在多个不同时间的研究数据，试图介绍某一因变量如何受到各种连续型以及分类型自变量的影响。因为每人有沿时间收集的多个因变量值，所以这样的问题有些像是多元时间序列问题，但是，收集数据的时间点不一定是完全对齐的，不同受试者的时间点个数不一定相同，所以经典的等时间间隔的多元时间序列模型如向量自回归无法使用。另外，时间序列模型也不擅长分析外生自变量的影响。

这种问题更适用于回归类的模型，只不过要将回归模型中误差项独立同分布等方差假定放松，因为同一个受试者在不同时间的因变量值是相关的，不同受试者的因变量值仍可以认为独立。在这样的模型中，用来解释因变量的连续型和分类型自变量的作用称为**固定效应**，个体之间的差别的影响称为**随机效应**，这样既有固定效应又有随机效应的线性模型称为**线性混合模型**或者线性混合效应模型。

以 nmleU 包的 ARMD 数据为例，这个数据中有 240 名 ARMD（年龄相关性黄斑变性），随机分为治疗组与对照组，并在入组时以及 4、12、24、52 周时对每位病人测量视力指标，这样的数据称为纵向数据，研究目的是考察治疗的有效性以及病变随时间的变换规律。以入组后的 4 次视力测量值为因变量，以处理效应、时间效应等为固定效应，以不同病人作为随机效应，可以用线性混合模型建模。

35.3.2 理论模型

设数据按某个分组变量分为 N 组，不同组之间相互独立，第 i 组有 n_i 个观测，模型写成

$$y_i = X_i\beta + Z_i b_i + \varepsilon_i. \tag{35.6}$$

模型(35.6)中：

- X_i 是由连续型自变量与因子自变量构造的回归设计阵，β 代表这些自变量的作用（固定效应），相当于经典的线性模型中的回归系数向量。
- b_i 是一个未知的随机向量，代表第 i 组特有的个体效应（随机效应），不同组的 b_i 相互独立，Z_i 为 $n_i \times q$ 的设计阵，保存与 q 个随机效应对应的自变量值，$Z_i b_i$ 可以描述第 i 组特有的相关性。Z_i 可以使用 X_i 的部分列。
- ε_i 是第 i 组的随机误差，仍是分量之间独立的且在不同组之间独立的。

在模型中，固定效应 β 是需要估计和推断的，而随机效应 b_i 则一般不需要估计。关于 b_i 和 ε_i 还进行如下的分布假定：

$$b_i \sim \mathrm{N}_q(0, \sigma^2 D), \quad \varepsilon_i \sim \mathrm{N}_{n_i}(0, \sigma^2 R_i), \quad b_i\text{与}\varepsilon_i\text{相互独立}.$$

不同组的 (b_i, ε_i) 之间也相互独立。

(35.6)是对第 i 组的模型。将所有组的 y_i 合并成一个向量 y，长度为 $n = \sum_{i=1}^N n_i$，将所有设计阵 X_i 纵向合并成一个 $n \times p$ 矩阵 X，将所有随机效应 b_i 合并成一个向量 b，长度为 Nq，将所有对应于 b_i 的设计阵 Z_i 按分块对角矩阵方式合并为一个 $n \times (Nq)$ 矩阵 Z，将所有 ε_i 合并成一个向量 ε，模型可以统一写成

$$y = X\beta + Zb + \varepsilon,$$

其中

$$b \sim \mathrm{N}(0, \sigma^2 \tilde{D}), \quad \varepsilon \sim \mathrm{N}(0, \sigma^2 R),$$

b 和 ε 相互独立，$\tilde{D}$ 和 R 为分块对角矩阵，$\tilde{D} = \mathrm{diag}(D, D, \dots, D)$, $R = \mathrm{diag}(R_1, \dots, R_N)$。

随机效应和误差项的这种分块对角矩阵形式的方差阵结构，是来源于假定不同组之间相互独立。如果有两层分组，内层分组嵌套在外层分组中，这时可以引入外层与内层的两种随机效应，在组间仍可假定相互独立，随机效应和误差项的方差阵仍保持对角矩阵形式。如果存在两种分组但分组之间存在交互则会违背这样的假定。

35.3.3 因变量分布

在每一组中，若将 b_i 看作已知，y_i 的条件分布服从多元正态分布，有

$$\begin{aligned} E(y_i|b_i) \equiv \mu_i =& X_i\beta + Z_i b_i, \\ \mathrm{Var}(y_i|b_i) =& \sigma^2 R_i. \end{aligned}$$

一般形式的线性混合模型包含太多的未知参数，无法估计，所以需要对随机效应和误差项的方差结构进行简化，假定其仅依赖于少数几个未知参数。对于一个协方差阵 Σ，可以将其分解为标准差与相关阵两部分：

$$\Sigma = \Lambda \Xi \Lambda,$$

其中 Ξ 为 Σ 对应的相关系数阵，Λ 是各分量的标准差组成的对角阵。这样，可以分别对标准差和相关系数进行参数化表示。

y_i 边缘分布为多元正态，

$$\begin{aligned} E(y_i) =& X_i\beta, \\ \mathrm{Var}(y_i) =& \sigma^2 \left(Z_i D Z_i^T + R_i\right). \end{aligned}$$

从给定 b_i 的条件分布与边缘分布来看，固定效应系数 β 起到了相同的作用，可以将其看成总体（所有受试者）所共有的特性。

35.3.4 估计方法

可以基于 y_i 的边缘分布写出似然函数，进行最大似然估计或限制最大似然估计 (REML)。

最大似然估计实际是根据 $\{y_i\}$ 的联合分布进行的最大似然估计。但是最大似然估计给出的方差、协方差不是无偏估计，在小样本情形影响更大。

REML(Restricted Maximum Likelihood) 是先用误差项独立同分布假设下的最小二乘方法估计出固定效应参数，求出残差并利用残差估计方差结构，这实际上是对因变量观测值向量进行了 $n-p$ 个线性独立的变换，这些线性变换向量与设计阵的 p 列正交，这样可以得到方差、协方差的无偏估计；再假定方差结构已知写出似然函数重新估计固定效应参数。因为是分两步进行的估计，所以两个用 REML 估计的嵌套模型，仅固定效应相同时可比，可以用 REML 得到的似然比检验比较其随机效应的差异。

另一方面，如果两个嵌套模型的随机效应相同，固定效应不同，则可以用最大似然估计得到的似然比检验比较两个模型的固定效应的差异。

似然比检验计算精简模型和完全模型的对数似然函数值最大值 (ML 或 REML) 的差的 -2 倍作为检验统计量，在无显著差异的零假设下近似服从自由度等于参数个数之差的卡方分布，统计量值超过右侧临界值时拒绝零假设。

35.3.5 板材摩擦力数据建模

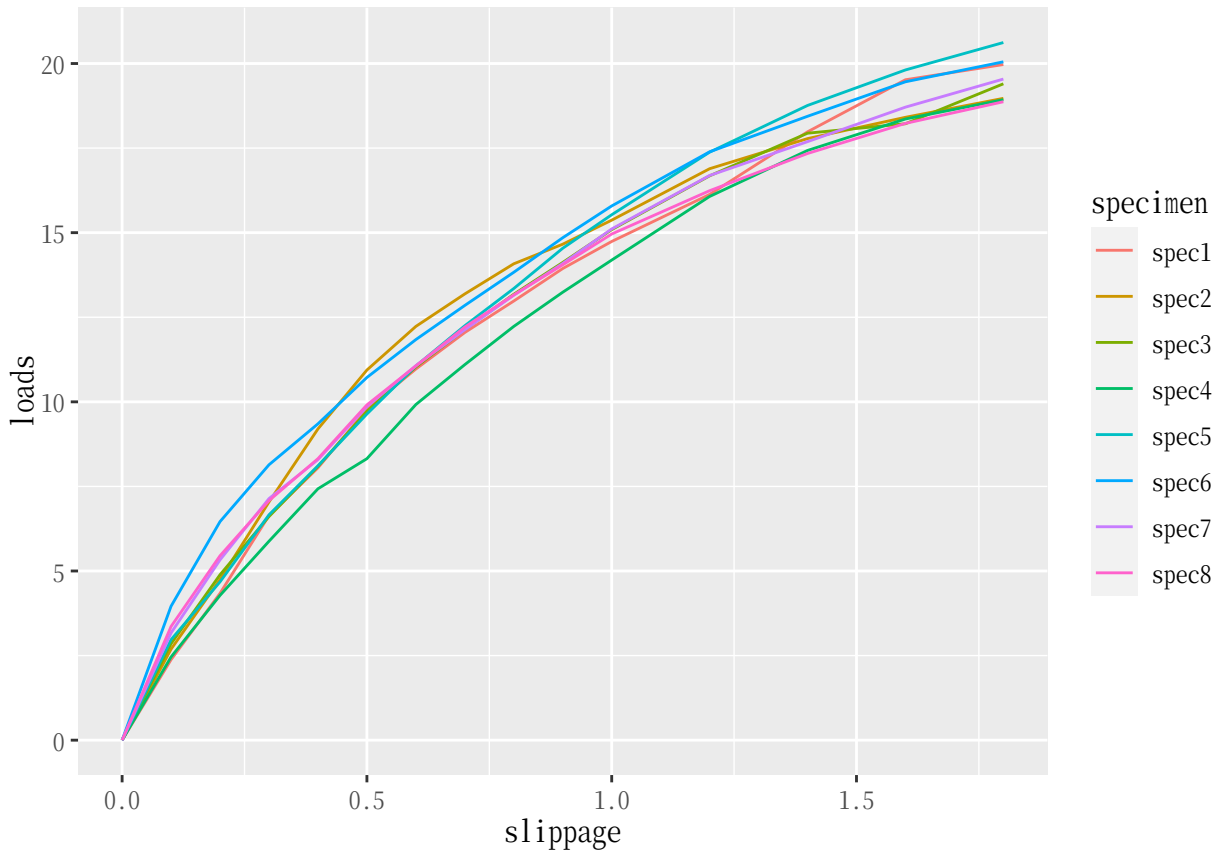
ARMD 数据比较复杂，我们先用一个比较简单的数据进行讲解。数据框 `d.timber` 是研究不同类型板材在固定夹具中滑动一定程度所需的力的大小测量值的数据，包含了对 8 块板材样本的 15 次观测，每次测量实现不同的滑动程度，记录所需的力的大小，变量 specimen 表示 8 块不同板材，变量 slippage 表示要达到的滑动程度，变量 loads 表示达到要求的滑动程度所需要的力的大小。共有 8×15 个观测。部分观测如：

specimen	slippage	loads
spec1	0.0	0.00
spec1	0.1	2.38
spec1	0.2	4.34
spec1	0.3	6.64
spec1	0.4	8.05
spec2	0.0	0.00
spec2	0.1	2.69
spec2	0.2	4.75
spec2	0.3	7.04
spec2	0.4	9.20

考虑不同板材之间所需的力可能有差异，但是我们不关心这些具体的差异，我们主要关心 loads 和 slippage 的关系。

先作图显示：

```
d.timber |>
  ggplot(aes(x = slippage, y = loads, color = specimen)) +
  geom_line()
```



loads 和 slippage 的关系存在非线性，我们用二次项来表示非线性。模型为：

$$
\begin{aligned}
\text{loads}_{ij} =& \beta_0 + b_i + \beta_1 \text{slippage} + \beta_2 \text{slippage}^2 + \epsilon_{ij}, \\
& i = 1, 2, \ldots, 8,\ j = 1, 2, \ldots, 15.
\end{aligned}
$$

其中 ϵ_{ij} 独立同 $\mathrm{N}(0, \sigma_e^2)$ 分布，b_i 独立同 $\mathrm{N}(0, \sigma_b^2)$ 分布，$\{b_i\}$ 与 $\{\epsilon_{ij}\}$ 相互独立。

于是，

$$
\begin{aligned}
\text{loads}_{ij}|b_i \sim& \mathrm{N}(\beta_0 + b_i, \sigma_e^2), \\
\text{loads}_{ij} \sim& \mathrm{N}(\beta_0, \sigma_b^2 + \sigma_e^2), \\
\mathrm{Cov}(\text{loads}_{ij}, \text{loads}_{ik}) =& \sigma_b^2,\ j \neq k, \\
\mathrm{Corr}(\text{loads}_{ij}, \text{loads}_{ik}) =& \frac{\sigma_b^2}{\sigma_b^2 + \sigma_e^2}, \\
\mathrm{Cov}(\text{loads}_{ij}, \text{loads}_{mk}) =& 0,\ i \neq m.
\end{aligned}
$$

即同一板材样品针对不同滑动程度的力测量值之间的相关系数为常数，不同样品之间独立，称这样的方差结构为复合对称 (compound symmetry)。这样的方差结构比较简单，在纵向数据情形下不够准确，因为时间相近的同样品测量值的相关系数更高。

β_0 是 slippage 等于 0 时 loads 的基准值，应该等于 0，但是因为每个样品在 slippage 等于 0 时的值是 $\beta_0 + b_i$，所以我们不限制 $\beta_0 = 0$。b_i 是第 i 样品与其它样品所需力的差异的表现，我们不关心 b_i 的具体值。

用 nlme 包建模如下：

```
## Linear mixed-effects model fit by REML
##   Data: d.timber
##        AIC      BIC    logLik
##   221.3987 235.2096 -105.6994
##
## Random effects:
##  Formula: ~1 | specimen
##         (Intercept)  Residual
## StdDev:   0.4250922 0.5320369
##
## Fixed effects:  loads ~ slippage + I(slippage^2)
##                   Value Std.Error  DF   t-value p-value
## (Intercept)    0.943394 0.1918903 110   4.91632       0
## slippage      19.889105 0.3219437 110  61.77820       0
## I(slippage^2) -5.429453 0.1761170 110 -30.82867       0
##  Correlation:
##               (Intr) slippg
## slippage      -0.521
## I(slippage^2)  0.435 -0.959
##
## Standardized Within-Group Residuals:
##        Min         Q1        Med         Q3        Max
## -3.0335318 -0.5491485 -0.1409554  0.5915032  2.3797394
##
## Number of Observations: 120
## Number of Groups: 8
```

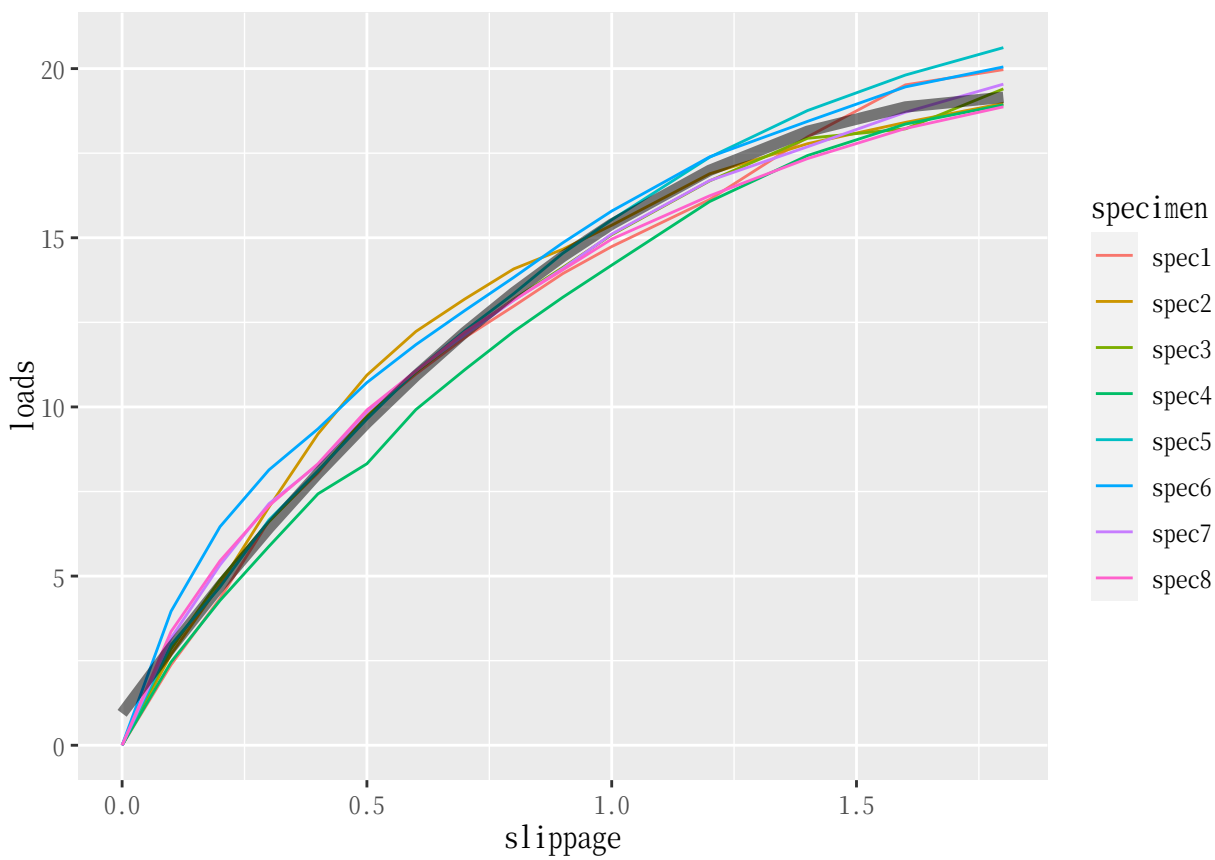
程序中 `random =` 是随机效应的设置，`| specimen` 表明按照样品分组，各组之间独立，每组内部有随机效应，`~ 1` 表示随机效应中仅有一个随机的截距项 (b_i)。

输出中 `(Intercept)`, `slippage` 和 `I(slippage^2)` 部分是固定效应估计，与线性回归模型中类似项的解释相同。

输出结果中随机效应部分，给出了随机截距项的方差估计 $\sigma_b^2 = 0.9467^2$, 随机误差的方差估计 $\sigma_e^2 = 0.5334236^2$。

计算模型预测值并作图：

```
slippage <- d.timber |>
  filter(specimen == "spec1") |>
  arrange(slippage) |>
  pull(slippage)
d.timbp <- tibble(slippage=slippage) |>
  mutate(loads = predict(
    timb.lme1, level = 0,
    newdata = tibble(slippage=slippage)))
ggplot() +
  geom_line(data = d.timber, mapping = aes(
    x = slippage, y = loads, color = specimen)) +
  geom_line(data = d.timbp, mapping = aes(
    x = slippage, y = loads ),
    size = 2, alpha = 0.5)
```



进一步地，考虑每个样品的 slippage 一次项系数也增加一个随机效应，二次项系数仍仅有固定效应，模型变成：

$$\begin{aligned}\text{loads}_{ij} =& \beta_0 + b_i + (\beta_1 + c_i)\text{slippage} + \beta_2\text{slippage}^2 + \epsilon_{ij},\\ & i = 1, 2, \ldots, 8,\ \ j = 1, 2, \ldots, 15.\end{aligned}$$

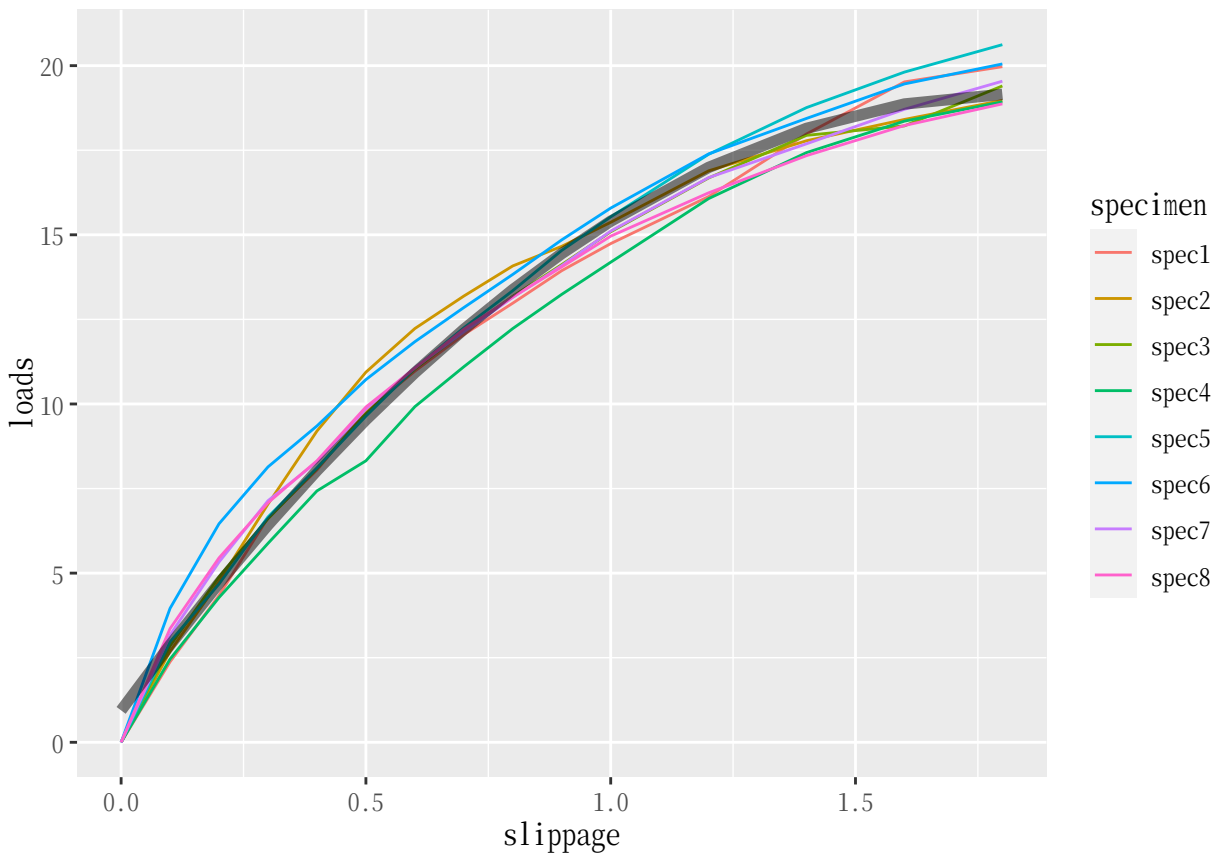
其中 ϵ_{ij} 独立同 $\mathrm{N}(0,\sigma_e^2)$ 分布，b_i 独立同 $\mathrm{N}(0,\sigma_b^2)$ 分布，c_i 独立同 $\mathrm{N}(0,\sigma_c^2)$ 分布，$\{b_i,c_i\}$ 与 $\{\epsilon_{ij}\}$ 相互独立。

建模程序：

```
timb.lme2 <- lme(
  loads ~ slippage + I(slippage^2),
  random = ~ 1 + slippage | specimen,
  data = d.timber)
summary(timb.lme2)
```

```
## Linear mixed-effects model fit by REML
##   Data: d.timber
##        AIC      BIC    logLik
##   219.2344 238.5696 -102.6172
##
## Random effects:
##  Formula: ~1 + slippage | specimen
##  Structure: General positive-definite, Log-Cholesky parametrization
##             StdDev    Corr
## (Intercept) 0.5314334 (Intr)
## slippage    0.3563649 -0.599
## Residual    0.4988500
##
## Fixed effects:  loads ~ slippage + I(slippage^2)
##                   Value Std.Error  DF   t-value p-value
## (Intercept)    0.943394 0.2186693 110   4.31425       0
## slippage      19.889105 0.3271010 110  60.80418       0
## I(slippage^2) -5.429453 0.1651313 110 -32.87961       0
##  Correlation:
##               (Intr) slippg
## slippage      -0.594
## I(slippage^2)  0.358 -0.885
##
## Standardized Within-Group Residuals:
##         Min          Q1         Med          Q3         Max
## -3.49458462 -0.50446350 -0.02813348  0.59525502  2.18850561
##
## Number of Observations: 120
## Number of Groups: 8
```

```
d.timbp2 <- tibble(slippage=slippage) |>
  mutate(loads = predict(
    timb.lme2, level = 0,
    newdata = tibble(slippage=slippage)))
ggplot() +
  geom_line(data = d.timber, mapping = aes(
    x = slippage, y = loads, color = specimen)) +
  geom_line(data = d.timbp2, mapping = aes(
    x = slippage, y = loads ),
    size = 2, alpha = 0.5)
```



结果中随机效应部分显示了随机截距项方差估计为 $\sigma_b^2 = 0.5314^2$，随机一次项斜率方差估计 $\sigma_c^2 = 0.3564^2$，$\mathrm{Corr}(b_i, c_i) = -0.599$，模型误差项方差 $\sigma_e^2 = 0.4989^2$。

为了检验随机斜率项的必要性，因为 `timb.lme1` 和 `timb.lme2` 都是用默认的 REML 估计的，而这两个模型采用了相同的固定效应，所以可以用似然比检验法比较其随机效应部分：

```
anova(timb.lme1, timb.lme2)
```

```
##           Model df      AIC      BIC    logLik   Test  L.Ratio p-value
```

```
## timb.lme1     1  5 221.3987 235.2096 -105.6994
## timb.lme2     2  7 219.2344 238.5696 -102.6172 1 vs 2 6.164316  0.0459
```

p 值 0.0459 在 0.05 水平下显著，所以认为带有随机一次项斜率的模型更优。

35.3.6 ARMD 数据建模

考虑对 ARMD 数据的分析。主要的研究目标是考察处理组与对照组的差异，以及视力随时间的变化。

35.3.6.1 主效应和随机截距项

先考虑如下的主效应模型：

$$\text{VISUAL}_{it} = \beta_0 + \beta_1 \times \text{VISUAL0}_i + \beta_2 \times \text{TIME}_{it} + \beta_3 \times \text{TREAT}_i + b_{0i} + \varepsilon_{it},$$

这里 i 是病人编码，t 是视力测量的时间序号 (1,2,3,4)，VISUAL_{it} 是第 i 号病人在第 t 次随访的视力测量值，β_0 是固定效应中的截距项，VISUAL0_i 是第 i 号病人在随机化分组前的视力的基线测量值，β_1 属于固定效应，TIME_{it} 是 VISUAL_{it} 的测量时间，取值于 $4, 12, 24, 52$ 周，β_2 属于固定效应；TREAT_i 是处理组的示性函数，第 i 病人属于处理组时为 1，处于对照组时为 0，β_3 属于固定效应。b_{0i} 是第 i 病人特有的截距项修正，期望为 0，是随机效应，此项存在使得同一病人的各观测之间有一个相等的正相关系数。ε_{it} 是随机误差项。

固定效应 β_3 体现了处理组与对照组的均值差异，此项等于零的检验如果显著，说明两组之间有显著差异。

因为处理组和对照组关于时间 TIME_{it} 的斜率相等，都是 β_2，所以这个模型的固定效应部分是一个平行线模型。

设 ε_{it} 独立同分布 $\text{N}(0, \sigma_e^2)$，b_i 独立同分布 $\text{N}(0, \sigma_b^2)$，$\{\varepsilon_{ij}\}$ 与 $\{b_i\}$ 相互独立，则

$$\begin{aligned}
\text{Var}(\text{VISUAL}_{it}) =& \sigma_b^2 + \sigma_e^2, \\
\text{Cov}(\text{VISUAL}_{it}, \text{VISUAL}_{it'}) =& \sigma_b^2, (t \neq t') \\
\text{Corr}(\text{VISUAL}_{it}, \text{VISUAL}_{it'}) =& \frac{\sigma_b^2}{\sigma_b^2 + \sigma_e^2},
\end{aligned}$$

即同一受试者不同时间的测量值之间的相关系数为常数，称这样的方差结构为复合对称 (compound symmetry)。

用 R 程序计算如下：

```
library(nlme)
data(armd, package="nlmeU")
lme1 <- lme(
  visual ~ visual0 + time + treat.f,
  random = ~ 1 | subject,
  data = armd)
summary(lme1)
```

```
## Linear mixed-effects model fit by REML
##   Data: armd
##        AIC      BIC    logLik
##   6587.202 6615.764 -3287.601
##
## Random effects:
##  Formula: ~1 | subject
##         (Intercept) Residual
## StdDev:    8.967864 8.637692
##
## Fixed effects:  visual ~ visual0 + time + treat.f
##                   Value Std.Error  DF    t-value p-value
## (Intercept)    9.788402 2.6586549 632   3.681712  0.0003
## visual0        0.826563 0.0446365 231  18.517678  0.0000
## time          -0.235355 0.0167632 632 -14.039982  0.0000
## treat.fActive -3.476393 1.3184773 231  -2.636673  0.0089
##  Correlation:
##               (Intr) visul0 time
## visual0       -0.928
## time          -0.136 -0.002
## treat.fActive -0.269  0.026  0.015
##
## Standardized Within-Group Residuals:
##         Min          Q1         Med          Q3         Max
## -4.10527500 -0.39197459  0.03661312  0.54464543  2.92150255
##
## Number of Observations: 867
## Number of Groups: 234
```

程序中 `random =` 是随机效应的设置，`| subject` 表明按照病人编码分组，各组之间独立，每组内部有随机效应，`~ 1` 表示随机效应中仅有一个随机的截距项（b_{0i}）。

结果中对 `treat.fActive` 的检验就是关于 $H_0: \beta_3 = 0$ 的检验，p 值为 0.0087，在 0.05 水平下显著，两组之间有显著差异。但处理组的视力更差。

输出结果中随机效应部分，给出了随机截距项 b_i 的方差估计 $\hat{\sigma}_b^2 = 8.97^2$, 随机误差 ε_{ij} 的方差估计 $\hat{\sigma}_e^2 = 8.64^2$。

结果给出了固定效应参数估计 $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3$ 之间的相关系数的估计，如 $\hat{\beta}_1$ 和 $\hat{\beta}_2$ 的相关系数估计为 -0.002。

35.3.6.2 固定效应中不同斜率

处理组和对照组的视力随时间的变化的斜率不一定相等，所以考虑如下的带有交叉项的模型：

$$\begin{aligned}\text{VISUAL}_{it} =& \beta_0 + \beta_1 \times \text{VISUAL0}_i + \beta_2 \times \text{TIME}_{it} + \beta_3 \times \text{TREAT}_i \\ &+ \beta_4 \times \text{TIME}_{it} \times \text{TREAT}_i + b_{0i} + \varepsilon_{it},\end{aligned}$$

这样，处理组关于 TIME_{it} 的斜率为 $\beta_2 + \beta_4$，对照组则为 β_2。

用 R 程序计算如下：

```
lme2 <- update(lme1, . ~ . + treat.f:time)
summary(lme2)
```

或直接写成：

```
lme2 <- lme(
  visual ~ visual0 + time + treat.f + treat.f:time,
  random = ~ 1 | subject,
  data = armd)
summary(lme2)
```

```
## Linear mixed-effects model fit by REML
##   Data: armd
##        AIC      BIC    logLik
##   6591.971 6625.286 -3288.986
##
## Random effects:
##  Formula: ~1 | subject
##         (Intercept) Residual
## StdDev:    8.978212 8.627514
##
## Fixed effects:  visual ~ visual0 + time + treat.f + treat.f:time
##                        Value Std.Error  DF   t-value p-value
## (Intercept)         9.288078 2.6818888 631  3.463260  0.0006
## visual0             0.826440 0.0446670 231 18.502244  0.0000
## time               -0.212216 0.0229295 631 -9.255150  0.0000
## treat.fActive      -2.422000 1.4999667 231 -1.614703  0.1077
## time:treat.fActive -0.049591 0.0335617 631 -1.477594  0.1400
##  Correlation:
##                    (Intr) visul0 time   trt.fA
## visual0            -0.920
```

```
## time                -0.185 -0.003
## treat.fActive       -0.295  0.022  0.335
## time:treat.fActive   0.126  0.002 -0.683 -0.476
##
## Standardized Within-Group Residuals:
##         Min          Q1         Med          Q3         Max
## -4.18750513 -0.39692515  0.03204783  0.55138252  2.95132118
##
## Number of Observations: 867
## Number of Groups: 234
```

默认的参数估计方法是 REML。

这里的检验是第三类检验，即有某项的模型与仅排除该项的模型的比较。用 `anova()` 则选择进行第一类检验，即依次增加项的检验：

```
anova(lme2)
```

```
##               numDF denDF  F-value p-value
## (Intercept)       1   631 5399.285  <.0001
## visual0           1   231  343.825  <.0001
## time              1   631  196.499  <.0001
## treat.f           1   231    6.942   0.009
## time:treat.f      1   631    2.183   0.140
```

因为交叉项 `time:treat.fActive` 在最后面，所以其第三类检验和第一类检验相同。

可以用 `anova()` 比较两个嵌套模型，为了比较随机效应相同、固定效应不同的模型，需要两个模型都使用最大似然估计。如：

```
lme1.ml <- update(lme1, method="ML")
lme2.ml <- update(lme2, method="ML")
anova(lme1.ml, lme2.ml)
```

```
##         Model df      AIC      BIC    logLik   Test  L.Ratio p-value
## lme1.ml     1  6 6579.854 6608.444 -3283.927
## lme2.ml     2  7 6579.672 6613.027 -3282.836 1 vs 2 2.181979  0.1396
```

取不同斜率有一定作用。

35.3.6.3 随机斜率

考虑每个人的视力随时间变化分别有不同的斜率。模型公式为

$$\begin{aligned}\text{VISUAL}_{it} =& \beta_0 + \beta_1 \times \text{VISUAL0}_i + \beta_2 \times \text{TIME}_{it} + \beta_3 \times \text{TREAT}_i \\ &+ \beta_4 \times \text{TIME}_{it} \times \text{TREAT}_i + b_{0i} + b_{1i}\text{TIME}_{it} + \varepsilon_{it}.\end{aligned}$$

用 R 程序计算如下：

```
lme3 <- update(lme2, random = ~ 1 + time | subject)
summary(lme3)
```

```
## Linear mixed-effects model fit by REML
##   Data: armd
##        AIC      BIC    logLik
##   6453.824 6496.657 -3217.912
##
## Random effects:
##  Formula: ~1 + time | subject
##  Structure: General positive-definite, Log-Cholesky parametrization
##             StdDev    Corr
## (Intercept) 7.0666328 (Intr)
## time        0.2729987 0.143
## Residual    6.7442472
##
## Fixed effects:  visual ~ visual0 + time + treat.f + treat.f:time
##                        Value Std.Error  DF   t-value p-value
## (Intercept)         4.768536 2.3195550 631  2.055798  0.0402
## visual0             0.908896 0.0391891 231 23.192601  0.0000
## time               -0.215285 0.0316627 631 -6.799343  0.0000
## treat.fActive      -2.290098 1.1835442 231 -1.934949  0.0542
## time:treat.fActive -0.056374 0.0462155 631 -1.219803  0.2230
##  Correlation:
##                    (Intr) visul0 time   trt.fA
## visual0            -0.934
## time               -0.073  0.003
## treat.fActive      -0.274  0.026  0.138
## time:treat.fActive  0.050 -0.002 -0.685 -0.207
##
## Standardized Within-Group Residuals:
```

```
##         Min          Q1         Med          Q3         Max
## -3.93038550 -0.33999935  0.04188037  0.46186462  2.98622721
##
## Number of Observations: 867
## Number of Groups: 234
```

这里程序中 random 参数的 ~ 1 + time 中的 time 表示每个病人关于 time 自变量有单独的斜率项，此斜率项为均值等于零的随机变量。

lme2 和 lme3 的固定效应相同，仅随机效应有差别，当固定效应相同时用 anova() 函数比较两个模型的随机效应部分，应使用 REML 估计。程序如：

```
anova(lme2, lme3)
```

```
##      Model df      AIC      BIC    logLik   Test  L.Ratio p-value
## lme2     1  7 6591.971 6625.286 -3288.986
## lme3     2  9 6453.824 6496.657 -3217.912 1 vs 2 142.1469  <.0001
```

结果表明加入 time 的随机效应斜率项是有意义的。从 AIC 结果来看也是 lme3 较好。

现在固定效应中的交叉项显著性不高，去掉此项：

```
lme4 <- update(lme3, . ~ . - treat.f:time)
summary(lme4)
```

```
## Linear mixed-effects model fit by REML
##   Data: armd
##        AIC      BIC  logLik
##   6448.999 6487.083 -3216.5
##
## Random effects:
##  Formula: ~1 + time | subject
##  Structure: General positive-definite, Log-Cholesky parametrization
##             StdDev    Corr
## (Intercept) 7.0799183 (Intr)
## time        0.2734706 0.138
## Residual    6.7423971
##
## Fixed effects:  visual ~ visual0 + time + treat.f
##                   Value Std.Error  DF    t-value p-value
## (Intercept)    4.927439 2.3181699 632   2.125573  0.0339
```

```
## visual0        0.908509 0.0392129 231  23.168651  0.0000
## time          -0.241733 0.0230894 632 -10.469443  0.0000
## treat.fActive -2.593006 1.1585178 231  -2.238210  0.0262
##  Correlation:
##               (Intr) visul0 time
## visual0       -0.935
## time          -0.054  0.003
## treat.fActive -0.270  0.026 -0.006
##
## Standardized Within-Group Residuals:
##         Min          Q1         Med          Q3         Max
## -3.94548685 -0.34966982  0.04283532  0.45701780  2.95567436
##
## Number of Observations: 867
## Number of Groups: 234
```

从 AIC 来看此模型较优。

lme3 和 lme4 随机效应相同而固定效应不同，为了用 `anova()` 检验固定效应中交叉项，应使用最大似然估计：

```
lme3.ml <- update(lme3, method="ML")
lme4.ml <- update(lme4, method="ML")
anova(lme3.ml, lme4.ml)
```

```
##         Model df      AIC      BIC    logLik   Test  L.Ratio p-value
## lme3.ml     1  9 6442.015 6484.900 -3212.008
## lme4.ml     2  8 6441.515 6479.635 -3212.757 1 vs 2 1.499852  0.2207
```

可见固定效应中处理组和对照组在 time 上的不同斜率是不必要的，关于 time 可以使用平行线模型。

35.3.6.4 更多问题

ARMD 属于纵向数据，同一个受试者在不同时间的测量值，一般时间比较接近的测量值的相关系数较大，时间间隔远的测量值的相关系数较小，可以假设方差结构具有这样的特点。

ARMD 数据中有一些缺失测量，大部分是在最后一次或者最后两次，称为提前退出 (dropouts)，提前退出如果与分组有关或者与视力有关，模型就需要将提前退出因素考虑在内。

35.3.7 Orthodont 数据建模

nlme 包中有一个例子数据 Orthodont，是 27 名儿童（16 名男生，11 名女生）在 8、10、12、14 岁测量的某个口腔距离（脑下垂体到翼上颌裂的距离）。这是纵向数据，同一名儿童的不同年龄的观测是相关的；这 27 名儿童看成是从一个很大的总体中抽样得到的，并不关系每一个儿童具体的情况。

```r
library(nlme)
```

```
##
## Attaching package: 'nlme'

## The following object is masked from 'package:dplyr':
##
##     collapse
```

```r
data(Orthodont, package = "nlme")
```

35.3.7.1 主效应和误差项等方差

考虑年龄增长时距离增长，并考虑性别的差异。用平行线模型。

```r
orth.lme1 <- lme(
  distance ~ age + Sex,
  random = ~ 1 | Subject,
  data=Orthodont)
summary(orth.lme1)
```

```
## Linear mixed-effects model fit by REML
##   Data: Orthodont
##        AIC      BIC    logLik
##   447.5125 460.7823 -218.7563
##
## Random effects:
##  Formula: ~1 | Subject
##         (Intercept) Residual
## StdDev:    1.807425 1.431592
##
## Fixed effects:  distance ~ age + Sex
##                 Value Std.Error DF   t-value p-value
## (Intercept) 17.706713 0.8339225 80 21.233044  0.0000
## age          0.660185 0.0616059 80 10.716263  0.0000
## SexFemale   -2.321023 0.7614168 25 -3.048294  0.0054
##  Correlation:
##           (Intr) age
## age       -0.813
```

```
## SexFemale -0.372  0.000
##
## Standardized Within-Group Residuals:
##         Min          Q1         Med          Q3         Max
## -3.74889609 -0.55034466 -0.02516628  0.45341781  3.65746539
##
## Number of Observations: 108
## Number of Groups: 27
```

年龄和性别的固定效应都显著。

35.3.7.2 加入交互作用

增加年龄和性别的交互作用效应，即不同性别使用不同的年龄斜率项：

```r
orth.lme2 <- update(orth.lme1,
  . ~ . + Sex:age)
summary(orth.lme2)
```

```
## Linear mixed-effects model fit by REML
##   Data: Orthodont
##        AIC      BIC    logLik
##   445.7572 461.6236 -216.8786
##
## Random effects:
##  Formula: ~1 | Subject
##         (Intercept) Residual
## StdDev:    1.816214 1.386382
##
## Fixed effects:  distance ~ age + Sex + age:Sex
##                   Value Std.Error DF   t-value p-value
## (Intercept)   16.340625 0.9813122 79 16.651810  0.0000
## age            0.784375 0.0775011 79 10.120823  0.0000
## SexFemale      1.032102 1.5374208 25  0.671321  0.5082
## age:SexFemale -0.304830 0.1214209 79 -2.510520  0.0141
##  Correlation:
##               (Intr) age    SexFml
## age           -0.869
## SexFemale     -0.638  0.555
## age:SexFemale  0.555 -0.638 -0.869
```

```
##
## Standardized Within-Group Residuals:
##          Min           Q1          Med           Q3          Max
## -3.59804400 -0.45461690  0.01578365  0.50244658  3.68620792
##
## Number of Observations: 108
## Number of Groups: 27
```

orth.lme1 和 orth.lme2 方差结构相同，固定效应不同，需要用最大似然估计对应的似然比检验进行比较：

```
orth.lme1b <- update(orth.lme1, method = "ML")
orth.lme2b <- update(orth.lme2, method = "ML")
anova(orth.lme1b, orth.lme2b)
```

```
##            Model df      AIC      BIC    logLik   Test  L.Ratio p-value
## orth.lme1b     1  5 444.8565 458.2671 -217.4282
## orth.lme2b     2  6 440.6391 456.7318 -214.3195 1 vs 2 6.217427  0.0126
```

不同性别使用不同的 age 斜率项是有意义的。因为 age 和 SexFemale 的交互项系数为负值且显著，说明女性的 age 斜率项显著地低于男性，即女生的因变量随年龄的增长率低于男生。

35.3.7.3 不同性别的误差项使用不同方差

如果将男、女分两组分别建模，可以发现其随机误差部分的方差估计相差较大。如下的程序可以设定男、女两组的误差项使用不同的方差，而随机截距项仍假设相同的方差：

```
orth.lme3 <- update(orth.lme2,
  weights = varIdent(form = ~ 1 | Sex))
summary(orth.lme3)
```

也可以直接写成

```
orth.lme3 <- lme(
  distance ~ age + Sex + Sex:age,
  random = ~ 1 | Subject,
  weights = varIdent(form = ~ 1 | Sex),
  data = Orthodont)
summary(orth.lme3)
```

```
## Linear mixed-effects model fit by REML
```

```
##   Data: Orthodont
##        AIC      BIC    logLik
##   429.2205 447.7312 -207.6102
##
## Random effects:
##  Formula: ~1 | Subject
##         (Intercept) Residual
## StdDev:     1.84757 1.669823
##
## Variance function:
##  Structure: Different standard deviations per stratum
##  Formula: ~1 | Sex
##  Parameter estimates:
##      Male    Female
## 1.0000000 0.4678944
## Fixed effects:  distance ~ age + Sex + Sex:age
##                   Value Std.Error DF   t-value p-value
## (Intercept)   16.340625 1.1450945 79 14.270111  0.0000
## age            0.784375 0.0933459 79  8.402883  0.0000
## SexFemale      1.032102 1.4039842 25  0.735124  0.4691
## age:SexFemale -0.304830 0.1071828 79 -2.844016  0.0057
##  Correlation:
##               (Intr) age    SexFml
## age           -0.897
## SexFemale     -0.816  0.731
## age:SexFemale  0.781 -0.871 -0.840
##
## Standardized Within-Group Residuals:
##         Min          Q1         Med          Q3         Max
## -3.00556474 -0.63419474  0.01890475  0.55016878  3.06446971
##
## Number of Observations: 108
## Number of Groups: 27
```

程序中 `weights` 参数用来指定非独立同分布情形的误差项方差结构，其中 `varIdent()` 用来规定在某个分类变量的每一类中独立且方差相同的方差结构，这里 `~ 1 | Sex` 则指定了按性别分成两组，每一组内的误差项假定是相互独立且方差相等的，两组的误差项方差不相等。

在输出结果中，显示了不同层 (stratum)，这里是男性和女性两组的误差项方差比例关系，比例为 $1:0.47$。

lme2 和 lme3 的固定效应相同，方差结构不同且构成嵌套关系，可以用基于 REML 的似然比检验进行比较：

```
anova(orth.lme2, orth.lme3)
```

```
##           Model df      AIC      BIC    logLik   Test  L.Ratio p-value
## orth.lme2     1  6 445.7572 461.6236 -216.8786
## orth.lme3     2  7 429.2205 447.7312 -207.6102 1 vs 2 18.53677  <.0001
```

两个模型有显著差异，说明男性组与女性组使用不同的误差项方差是必要的。

35.3.7.4 删去不同性别的不同截距

上面结果中 Sex 的主效应不显著，可以删去此主效应，使得固定效应中男女的截距项相同而斜率项不同。程序如下：

```
orth.lme4 <- update(orth.lme3,
  . ~ . - Sex)
summary(orth.lme4)
```

```
## Linear mixed-effects model fit by REML
##   Data: Orthodont
##        AIC      BIC    logLik
##   430.2763 446.2001 -209.1381
##
## Random effects:
##  Formula: ~1 | Subject
##         (Intercept) Residual
## StdDev:    1.842395 1.665563
##
## Variance function:
##  Structure: Different standard deviations per stratum
##  Formula: ~1 | Sex
##  Parameter estimates:
##      Male    Female
## 1.0000000 0.4685187
## Fixed effects:  distance ~ age + age:Sex
##                   Value Std.Error DF   t-value p-value
## (Intercept)   17.026899 0.6611240 79 25.754472   0e+00
## age            0.734206 0.0635120 79 11.560109   0e+00
## age:SexFemale -0.238642 0.0580447 79 -4.111355   1e-04
##  Correlation:
##               (Intr) age
## age           -0.761
```

```
## age:SexFemale  0.305 -0.693
##
## Standardized Within-Group Residuals:
##        Min         Q1        Med         Q3        Max
## -3.1173385 -0.6522050  0.0218480  0.5116131  3.0887448
##
## Number of Observations: 108
## Number of Groups: 27
```

比较两个模型：

```
orth.lme3b <- update(orth.lme3, method="ML")
orth.lme4b <- update(orth.lme4, method="ML")
anova(orth.lme3b, orth.lme4b)
```

```
##            Model df      AIC      BIC    logLik   Test   L.Ratio p-value
## orth.lme3b     1  7 423.3524 442.1273 -204.6762
## orth.lme4b     2  6 421.9128 438.0056 -204.9564 1 vs 2 0.5604856  0.4541
```

可见两个模型没有显著差异。

35.4 非线性混合模型介绍

待完成。

Chapter 36

广义线性模型

36.1 模型

36.1.1 介绍

线性回归模型

$$Y = a + bx + \varepsilon,\ \varepsilon \sim \mathrm{N}(0, \sigma^2)$$

可以写成

$$\begin{aligned} Y|x \sim & \mathrm{N}(g(x), \sigma^2), \\ g(x) = & a + bx. \end{aligned}$$

这样，因变量 Y 与自变量 $x_1, x_2, \dots, x_p$ 的更一般的模型，可以写成如下形式的**广义线性模型**：

$$\begin{aligned} Y \sim & F(y; \mu) \\ g(\mu) = & \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p. \end{aligned}$$

其中 $F(\cdot, \mu)$ 是参数为 μ 的某种分布，通常取 $\mu = EY$，$g(\cdot)$ 为某个线性或非线性函数，称为联系函数 (link function)。模型也可以写成

$$Y|(x_1, \dots, x_p) \sim F(y;\, g^{-1}(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)).$$

一个广义线性模型有如下三个部分：

- 随机部分，给出因变量 Y 及其条件概率分布 $Y \sim F(\cdot; \mu)$；
- 系统部分，给出模型自变量的线性函数 $\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$；
- 联系函数 $g(\cdot)$，将因变量分布参数（一般是均值）μ 与自变量线性组合 $\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$ 联系在一起。

在广义线性模型中，因变量的分布不一定是正态分布，因变量的期望值 EY 不一定是自变量的线性函数，因变量的方差不一定是恒定值。

典型的因变量分布有二项分布、泊松分布等。

本章内容参考了 Paul Roback 和 Julie Legler 的教材《超越多元线性回归–用 R 语言进行广义线性模型和多水平模型应用》(Roback & Legler, 2021)：

- https://bookdown.org/roback/bookdown-BeyondMLR/

36.1.2 指数族分布

设随机变量 Y 的概率密度函数或概率质量函数 $f(y)$ 可以表达为

$$f(y;\theta)=\exp\left\{a(y)b(\theta)+c(\theta)+d(y)\right\},$$

且 $f(\cdot;\theta)$ 的支撑集不依赖于 θ，则称 Y 服从**单参数指数族分布**。

单参数指数族分布与广义线性模型有密切联系，其中 $b(\cdot)$ 称为经典联系函数，作为广义线性模型中 $g(\cdot)$ 的一种重要选择。

对单参数指数族分布随机变量 Y，计算可得

$$EY=-\frac{c'(\theta)}{b'(\theta)},\quad \mathrm{Var}(Y)=\frac{b''(\theta)c'(\theta)-c''(\theta)b'(\theta)}{[b'(\theta)]^3}.$$

单参数指数族分布包含了许多常见分布，如方差已知的正态分布、泊松分布、二项分布等。

例如，考虑一元线性回归模型

$$Y=a+bx+\varepsilon,\ \varepsilon\sim \mathrm{N}(0,\sigma^2),$$

正态 $\mathrm{N}(\mu,\sigma^2)$（其中 σ^2 已知）的密度函数写成单参数指数族为

$$\begin{aligned}f(y;\mu)=&\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{1}{2\sigma^2}(y-\mu)^2\right\}\\=&\exp\left\{\left[\frac{y}{\sigma^2}\mu\right]+\left[-\frac{\mu^2}{2\sigma^2}\right]+\left[-\frac{1}{2}\ln 2\pi-\frac{1}{2}\ln\sigma^2-\frac{1}{2\sigma^2}y^2\right]\right\}\\=&\exp\{a(y)b(\theta)+c(\theta)+d(y)\}.\end{aligned}$$

这里参数为 $\theta=\mu$。一元线性回归模型可以写成

$$\begin{aligned}Y\sim&\mathrm{N}(\theta,\sigma^2),\\g(\theta)=&\theta=a+bx,\end{aligned}$$

其中 $g(\theta)=\theta$ 是恒等函数，这是经典联系函数 $b(\theta)=\theta$。

泊松分布的概率质量函数为

$$f(y;\lambda)=\frac{\lambda^y}{y!}e^{-\lambda}=\exp\{y\ln\lambda-\lambda-\ln(y!)\},\ y=0,1,\dots,$$

是单参数指数族分布。参数为 $\lambda=EY$。经典联系函数为 $g(\theta)=\ln\theta$，一元泊松回归模型可以写成

$$\begin{aligned}Y\sim&\mathrm{Poisson}(\theta),\\g(\theta)=&\ln(\theta)=a+bx.\end{aligned}$$

二项分布 $\mathrm{B}(n,p)$ 当 n 已知，以 p 为参数时，是单参数指数族分布，其概率质量函数为

$$\begin{aligned}
f(y;p) =& \binom{n}{y} p^y (1-p)^{n-y},\ y=0,1,\dots,n \\
=& \binom{n}{y} (1-p)^n \left(\frac{p}{1-p}\right)^y \\
=& \exp\left\{ y \ln \frac{p}{1-p} + n\ln(1-p) + \ln \binom{n}{y} \right\} \\
=& \exp\{a(y)b(\theta) + c(\theta) + d(y)\},
\end{aligned}$$

这里 $\theta = p = \frac{1}{n}EY$, 其中 n 已知。$b(p) = \ln \frac{p}{1-p}$，称 $\ln \frac{p}{1-p}$ 为 $\mathrm{logit}(p)$，是二项分布的经典联系函数。

$$\begin{aligned}
\mathrm{logit}(p) =& \ln \frac{p}{1-p},\ p \in (0,1), \\
\mathrm{logit}^{-1}(u) =& \frac{e^u}{1+e^u} = \frac{1}{1+e^{-u}} \in (0,1),\ u \in (-\infty,\infty).
\end{aligned}$$

一元逻辑斯谛回归模型可以写成

$$\begin{aligned}
Y \sim& \mathrm{B}(n,\theta), \\
g(\theta) =& \mathrm{logit}(\theta) = a + bx.
\end{aligned}$$

对于以上的三个分布，使用经典联系函数将因变量分布参数与自变量线性组合联系起来可以给出一些粗浅的解释。

- 对于正态分布因变量，因为正态分布可以在 $(-\infty,\infty)$ 取值，其期望值也可以在 $(-\infty,\infty)$ 取值，所以其期望值 μ 可以不进行变换地用自变量线性组合 $a+bx$ 表示。
- 对于泊松分布因变量，泊松分布仅取非负整数值，其期望 λ 仅取正值，所以不能直接用自变量线性组合 $a+bx$ 表示，因为 $a+bx$ 可能取到负值；而期望 λ 用 $\exp\{a+bx\}$ 表示则不会出现 λ 取负值的问题。
- 对于二项分布因变量，其分布参数 p 仅在 $(0,1)$ 区间内取值，所以不能直接用自变量线性组合 $a+bx$ 表示，因为 $a+bx$ 可能取到负值或者大于 1 的值，而 $\mathrm{logit}^{-1}(a+bx)$ 则可以将 $(-\infty,\infty)$ 映射到 $(0,1)$ 内，保证了 p 的取值在允许范围内。

36.2 泊松回归

36.2.1 模型

泊松分布是一种常见的取非负整数值的随机变量分布，经常用于计数变量的模型。如果某种事件在短时间内发生的概率很低，发生与否独立，则该时间在某一时间区间内发生的次数服从泊松分布。参数为 λ 的泊松分布的概率质量函数为

$$f(y;\lambda) = \frac{\lambda^y}{y!} e^{-\lambda},\ y=0,1,\dots.$$

设随机变量 Y 的分布受到自变量 $x_1,\dots,x_p$ 的影响，在给定 $x_1,\dots,x_p$ 的条件下 Y 服从泊松分布，则可以用泊松回归描述 Y 与 $x_1,\dots,x_p$ 的关系：

$$\begin{aligned}
Y \sim& \mathrm{Poisson}(\lambda), \\
g(\lambda) =& \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p.
\end{aligned}$$

其中 $g(\lambda)$ 经常取为经典联系函数 $\ln\lambda$。

实际上，即使给定 $x_1,\dots,x_p$ 后 Y 服从泊松分布，参数 λ 与 $x_1,\dots,x_p$ 的关系也不一定恰好被 $g(\lambda)=\beta_0+\beta_1x_1+\cdots+\beta_px_p$ 确定，泊松回归模型是 $Y|(x_1,\dots,x_p)$ 分布的一种简化模型。

泊松回归的假定：

- $Y|(x_1,\dots,x_p)$ 服从泊松分布；
- 不同的观测相互独立；
- $Y|(x_1,\dots,x_p)$ 的分布均值与分布方差相等，这一点与线性回归不同；
- $g(\lambda)$ 与 $x_1,\dots,x_p$ 服从线性关系，这是对模型的简化假设。

36.2.2 最大似然估计

考虑单个自变量的泊松回归，设 $Y_i\sim\text{Poisson}(\lambda_i)$, $\ln\lambda_i=\beta_0+\beta_1x_i$，于是单个观测 (x_i,y_i) 对于似然函数的贡献为

$$\frac{\lambda_i^{y_i}}{y_i!}e^{-\lambda_i},$$

对于对数似然函数的贡献为

$$y_i\ln\lambda_i-\ln y_i!-\lambda_i=y_i(\beta_0+\beta_1x_i)-\ln y_i!-e^{\beta_0+\beta_1x_i},$$

其中 $-\ln y_i!$ 在对数似然函数中是常数项，所以对数似然函数为

$$\ell(\beta_0,\beta_1)=\text{常数}+\sum_{i=1}^n\left\{y_i(\beta_0+\beta_1x_i)-e^{\beta_0+\beta_1x_i}\right\}.$$

可以用 Newton-Raphson 迭代方法进行最大似然估计的数值求解，并且可以使用 Fisher 得分方法改造 Newton-Raphson 方法，具体方法是在迭代中用负信息阵的当前估计值代替对数似然函数的海色阵。这样还可以从最大值点处的信息阵的逆矩阵给出参数估计的协方差阵估计。这种利用 Fisher 得分的 Newton-Raphson 迭代方法，等价于迭代再加权最小二乘方法，每步迭代是一个加权最小二乘计算，加权按照当前的方差倒数来计算。

36.2.3 偏差与偏差分析

线性回归模型使用残差平方和以及方差分析评估模型的拟合优劣。因为广义线性模型是对因变量分布参数的拟合而非直接对因变量值的拟合，所以没有原来的残差以及残差平方和的概念，而是用偏差 (deviance) 代替。

设要评估的模型的对数似然函数在最大似然估计处的值为 ℓ_M，为了比较，考虑一个参数最复杂的“饱和模型”，使得因变量得到最大程度的解释。例如，在方差分析中对因素的每种组合都单独使用一个参数。理论上，在饱和模型中，应该每一组不同的自变量组合都单独设置一个因变量独立参数值，但是在 R 统计软件 `glm()` 计算时实际是对每个观测单独使用参数。计算饱和模型的对数似然函数的最大值 ℓ_S。这样，饱和模型的自由参数（独立参数）个数是最多的。

要评估的模型的偏差 (deviance) 定义为

$$\text{Dev}=-2[\ell_M-\ell_S],$$

偏差一定是非负的，对于某些广义线性模型，如果模型 M 很好地解释了数据，则模型 M 与饱和模型 S 应该效果相近，在一定条件下偏差近似服从卡方分布，自由度等于两个模型独立参数的个数差，当偏差不超过卡方的 $1-\alpha$ 分位数时可以认为模型 M 是充分的。

类似于线性模型中的平方和分解，偏差也可以进行分解。比如，有两个模型 M_0 和 M_1，M_0 是 M_1 的特例，即 M_0 是 M_1 的参数取特定值或满足特定约束后的模型，称 M_1 为完全模型，称 M_0 为简化模型，称这两个模型有嵌套关系。这时，为了在 M_1 的假设下检验 M_0 是否成立，计算似然比统计量

$$-2[\ell_0-\ell_1]=-2[\ell_0-\ell_S]+2[\ell_1-\ell_S]=\mathrm{Dev}_0-\mathrm{Dev}_1,$$

当 M_0 与 M_1 效果相同的零假设成立时统计量服从卡方分布，自由度等于两个模型的独立参数个数差，这个统计量可以用两个模型的偏差之差计算。

36.2.4 残差分析

对于泊松回归，也可以定义残差，但是不像线性回归那样只有观测值减去预测值这一种，因为广义线性模型中自变量预测的是因变量的期望值而非因变量本身。

以一元泊松回归为例。响应残差 (response residual) 定义为 $e_i=y_i-\hat{\mu}_i$，而 $\hat{\mu}_i$ 估计为 $g^{-1}(\hat{\beta}_0+\hat{\beta}_1x_i)$，其中 $g(\cdot)$ 是联系函数，对泊松回归一般取为 ln，所以 $\hat{\mu}_i=\exp(\hat{\beta}_0+\hat{\beta}_1x_i)$。

定义标准化残差或 Pearson 残差为

$$e_i^{(P)}=\frac{y_i-\hat{\mu}_i}{\sqrt{\hat{\mu}_i}}.$$

定义偏差残差 (deviance residual) 为

$$e_i^{(D)}=\mathrm{sgn}(y_i-\hat{\mu}_i)\left\{2\left[y_i(\ln y_i-\ln\hat{\mu}_i)-(y_i-\hat{\mu}_i)\right]\right\}^{1/2}.$$

其平方和恰好等于模型的残差偏差 (residual deviance)。偏差残差一般与 Pearson 残差近似相等。

对 `glm()` 结果，可以用 `residuals()` 函数提取各种残差，指定 `type="response"` 或`"pearson"`、`"deviance"`，缺省为 `type="deviance"`。

36.2.5 菲律宾家庭人数研究案例

36.2.5.1 数据与探索性分析

本例子来自 (Roback & Legler, 2021)§4.4。数据是菲律宾统计局的 2015 年调查数据，从 4 万户家庭数据中随机抽取了 1500 个家庭的子集。因变量是家庭中除了户主以外的人数，自变量包括户主年龄、家庭贫困与否、所在地区。家庭贫困与否，用住房屋顶是否主要使用廉价材料来表征。因为因变量是计数的，所以考虑使用泊松回归作为初始的模型。泊松分布参数 λ 是除了户主以外的家庭人数的期望值。

读入数据并展示前几行：

```
dfp <- read_csv(
  "data/fHH1.csv",
  col_types = cols(
    .default = col_double(),
    location = col_factor(
      levels = c("CentralLuzon", "DavaoRegion",
                 "IlocosRegion", "MetroManila", "Visayas")),
    roof = col_factor(
      levels = c("Predominantly Light/Salvaged Material",
                 "Predominantly Strong Material"))  ),
  col_select = c(-1))
```

```
## New names:
## * `` -> `...1`
```

```
knitr::kable(head(dfp, 5))
```

location	age	total	numLT5	roof
CentralLuzon	65	0	0	Predominantly Strong Material
MetroManila	75	3	0	Predominantly Strong Material
DavaoRegion	54	4	0	Predominantly Strong Material
Visayas	49	3	0	Predominantly Strong Material
MetroManila	74	3	0	Predominantly Strong Material

数据包含了 1500 个家庭的数据，每行为一个家庭，每行中各变量分别为：

- `location`：所在区域，有 5 个不同取值；
- `age`：户主年龄；
- `total`：除了户主以外的人数；
- `numLT5`：家庭中低于 5 岁的儿童个数；
- `roof`：房顶类型，`"Predominantly Light/Salvaged Material"` 代表家庭比较贫困，`"Predominantly Strong Material"` 则代表非贫困。

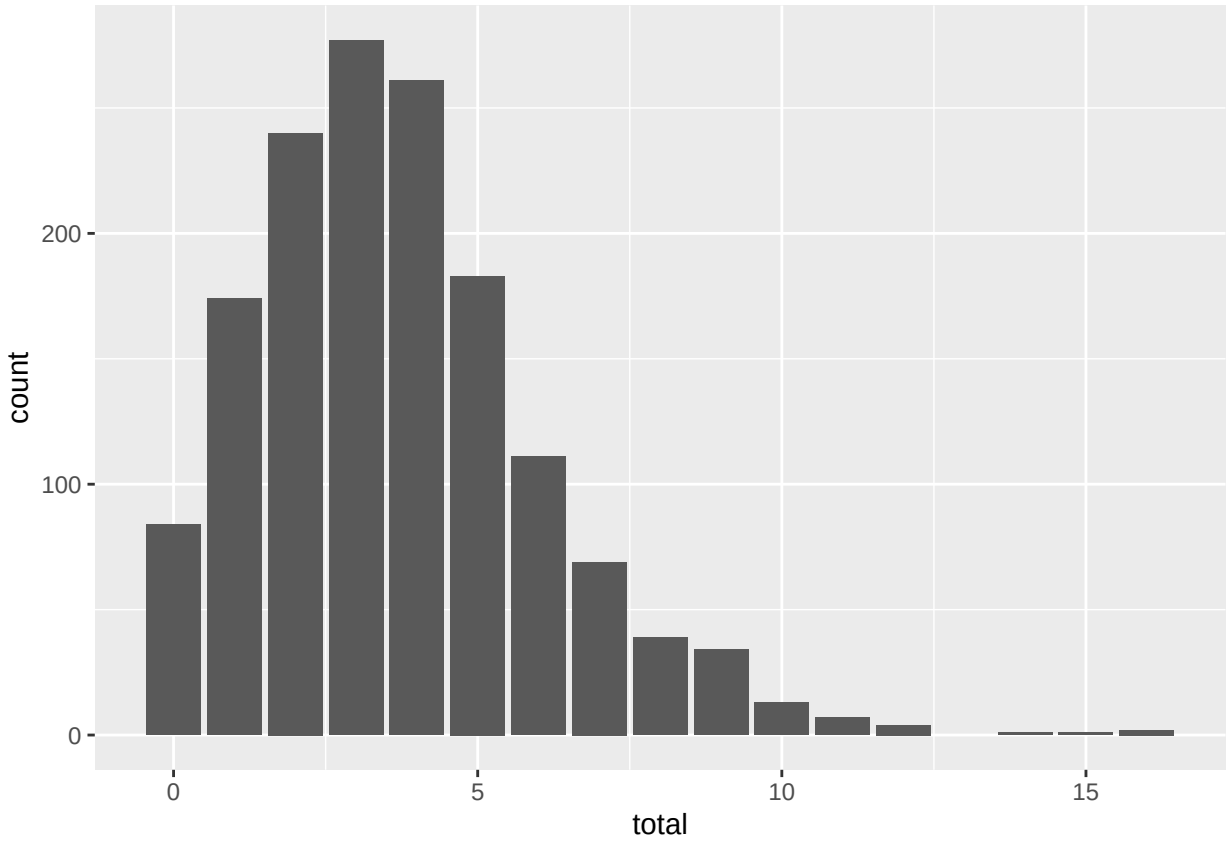
因变量 `total` 是一个取非负整数值的数值型变量，这里之所以扣除了户主，是因为泊松分布理论上可以取 0，而包括户主后人数至少等于 1。对 `total` 变量进行简单数值概括：

```
summary(dfp[["total"]])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   2.000   3.000   3.685   5.000  16.000
```

平均值为 3.685，中位数为 3，最大值为 16。因为 `total` 是离散取值的，不适合作一般的直方图，用条形图表现其分布：

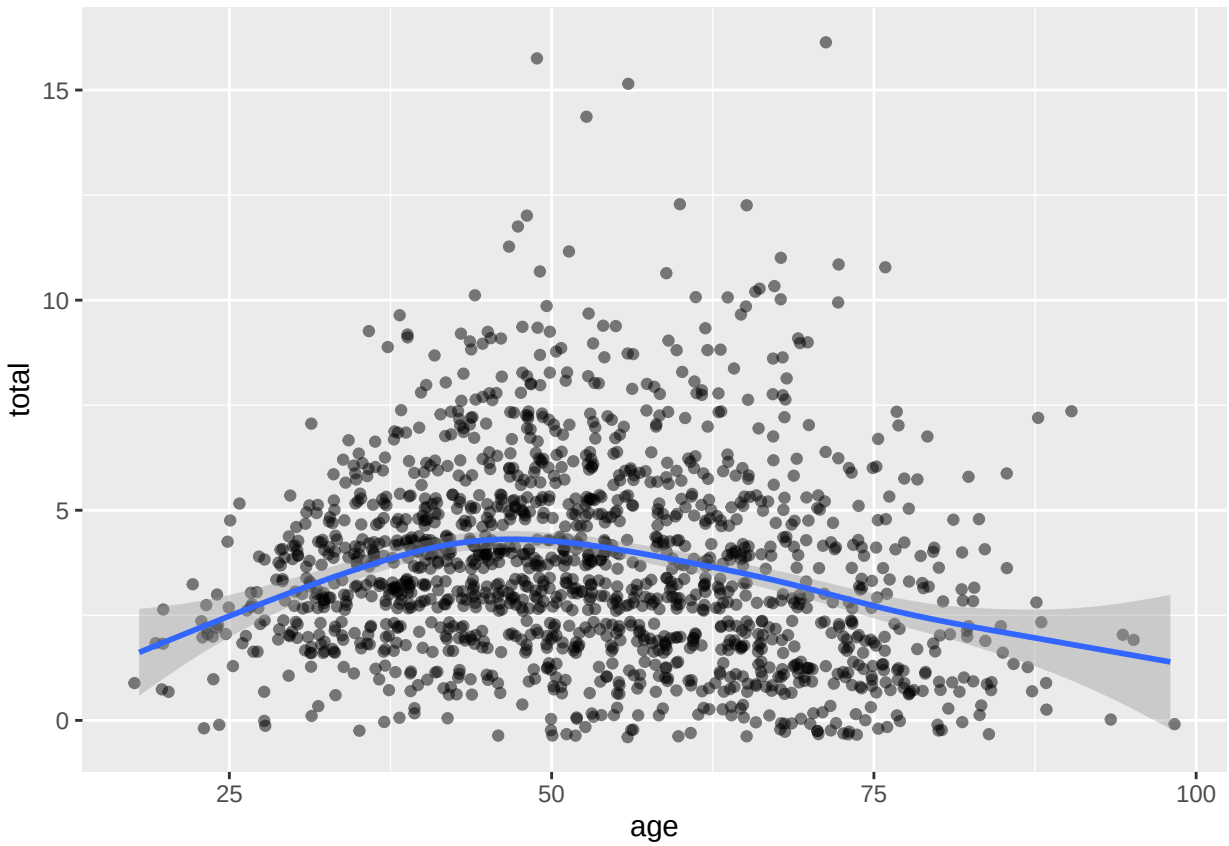
```
ggplot(data = dfp, mapping = aes(
  x = total )) +
  geom_bar()
```



为考察户主年龄与家庭人数的关系，作两者的散点图：

```
ggplot(data = dfp, mapping = aes(
  x = age, y = total )) +
  geom_jitter(alpha = 0.5) +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



其中的拟合曲线并不符合 $\exp\{a + b \cdot \text{age}\}$ 的形状。所以有可能需要增加 age 的二次项以改善模型。

从数据无法确定各个观测之间是否独立，这里先假定独立，如果抽样方案不能保证各个家庭独立，可以采用观测之间带有相关性的模型。

36.2.5.2 单个自变量的泊松回归建模

基本 R 软件的配套（默认安装）扩展包 stats 提供了 `glm()` 函数，用来进行广义线性模型建模。对家庭人口数问题，以 `total` 为因变量，以户主年龄 `age` 为自变量，先建立仅有一个自变量的模型：

```
glmp1 <- glm(total ~ age,
    family = poisson, data = dfp)
summary(glmp1)
```

```
##
## Call:
## glm(formula = total ~ age, family = poisson, data = dfp)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -2.9079  -0.9637  -0.2155   0.6092   4.9561
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.5499422  0.0502754  30.829  < 2e-16 ***
## age         -0.0047059  0.0009363  -5.026 5.01e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 2362.5  on 1499  degrees of freedom
## Residual deviance: 2337.1  on 1498  degrees of freedom
## AIC: 6714
##
## Number of Fisher Scoring iterations: 5
```

结果模型可以写成

$$\begin{aligned}\text{total} \sim &\text{Poisson}(\lambda),\\ \ln\lambda =&1.55-0.0047\text{age}.\end{aligned}$$

模型指出随着户主年龄增加，平均人口数会减少。从 `summary()` 函数给出的 `age` 变量的系数的 z 检验 p 值看，年龄对平均人口数是有显著影响的。

为了获得 β_1 参数的置信区间，可以利用 `confint()` 函数：

```
confint(glmp1)
```

```
## Waiting for profiling to be done...
```

```
##                    2.5 %       97.5 %
## (Intercept)  1.451170100  1.648249185
## age         -0.006543163 -0.002872717
```

可以用 `level` 参数指定置信度，默认值为 0.95。因为 $\log\lambda=\beta_0+\beta_1x$，所以 $\lambda=\exp\{\beta_0+\beta_1x\}$，当 x 增加一个单位时，λ 变化为原来的 e^{β_1} 倍；所以，当年龄增加一岁时，按模型解释，平均人口数会降低到原来的 $e^{-0.0047}=99.5\%$ 倍，即平均减少 0.5%，变化的倍数的 95% 置信区间为：

```
exp(confint(glmp1))
```

```
## Waiting for profiling to be done...
```

```
##                  2.5 %    97.5 %
## (Intercept) 4.2681057 5.1978713
## age         0.9934782 0.9971314
```

即减少 0.7% 到 0.3%。

36.2.5.3 与空模型的比较

可以建立一个没有自变量的模型作为基准模型，称这个模型为空模型 (null model)：

```r
glmp2 <- glm(total ~ 1,
    family = poisson, data = dfp)
summary(glmp2)
```

```
##
## Call:
## glm(formula = total ~ 1, family = poisson, data = dfp)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7147  -0.9619  -0.3687   0.6495   4.7285
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.30418    0.01345   96.96   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 2362.5  on 1499  degrees of freedom
## Residual deviance: 2362.5  on 1499  degrees of freedom
## AIC: 6737.4
##
## Number of Fisher Scoring iterations: 5
```

可以看出，结果中 `Residual deviance` 和 `Null deviance` 相同，也和前面包含自变量 age 的模型结果中的 `Null deviance` 相同。

在线性回归中我们用残差平方和 (SSE) 表示模型的拟合优度，更一般的广义线性模型中不一定能使用残差平方和，而是改用偏差 (deviance)。偏差等于定义要评估的模型与饱和模型的对数似然函数最大值之差的负二倍。通过比较

两个嵌套模型的残差偏差，可以判断复杂的模型是否与简单的模型有显著差异。饱和模型是自由参数最多的模型，对每个自变量组合都设置一个自由参数；空模型是仅包含截距项的模型，是自由参数最少的模型。

空模型结果可以写成公式

$$\text{total} \sim \text{Poisson}(e^{1.30} = 3.68).$$

以有 age 自变量的模型为完全模型，以没有自变量的模型为简化模型，用 `anova()` 函数进行偏差检验：

```
anova(glmp2, glmp1, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: total ~ 1
## Model 2: total ~ age
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1      1499     2362.5
## 2      1498     2337.1  1   25.399 4.661e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

结果给出了残差偏差的差值 (Deviance)，在简化模型与完全模型相同的零假设下，偏差近似服从卡方分布，自由度为两个模型的自由参数个数之差，这里自由度为 1，结果显著，说明加入 age 变量是有意义的。

36.2.5.4 增加年龄的二次项

从探索性数据分析结果发现年龄对平均人口数的影响并不是单调的，而是在中间达到最大值。所以有可能需要增加年龄的二次项。平均年龄约为 53，为了避免共线问题，增加 $(\text{age} - 53)^2$ 项：

```
glmp3 <- glm(total ~ age + I((age - 53)^2),
    family = poisson, data = dfp)
summary(glmp3)
```

```
##
## Call:
## glm(formula = total ~ age + I((age - 53)^2), family = poisson,
##     data = dfp)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9068  -0.9261  -0.1048   0.5773   5.1731
##
```

```
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      1.657e+00  5.596e-02  29.611  < 2e-16 ***
## age             -4.196e-03  1.033e-03  -4.062 4.87e-05 ***
## I((age - 53)^2) -7.083e-04  6.406e-05 -11.058  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 2362.5  on 1499  degrees of freedom
## Residual deviance: 2200.9  on 1497  degrees of freedom
## AIC: 6579.8
##
## Number of Fisher Scoring iterations: 5
```

这里二次项的 z 检验的 p 值也是显著的，说明增加二次项是有意义的。与仅有一次项的模型进行偏差比较：

```
anova(glmp1, glmp3, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: total ~ age
## Model 2: total ~ age + I((age - 53)^2)
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1      1498     2337.1
## 2      1497     2200.9  1   136.15 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

结果也是显著的。模型写成数学公式为

$$\log \lambda = 1.66 - 0.0042\text{age} - 0.00071(\text{age} - 53)^2.$$

36.2.5.5 增加更多自变量

其它自变量对家庭人口数也可以有影响。先增加 location 变量，这是一个有 5 个水平的因子，建模时会被转化为 4 个二值哑变量：

```r
glmp41 <- glm(total ~ age + I((age - 53)^2) + location,
    family = poisson, data = dfp)
summary(glmp41)
```

```
##
## Call:
## glm(formula = total ~ age + I((age - 53)^2) + location, family = poisson,
##     data = dfp)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9781  -0.9278  -0.1149   0.5924   5.0424
##
## Coefficients:
##                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)           1.5892291  0.0666770  23.835  < 2e-16 ***
## age                  -0.0041112  0.0010348  -3.973 7.09e-05 ***
## I((age - 53)^2)      -0.0007026  0.0000642 -10.944  < 2e-16 ***
## locationDavaoRegion  -0.0193872  0.0537827  -0.360  0.71849
## locationIlocosRegion  0.0609820  0.0526598   1.158  0.24685
## locationMetroManila   0.0544801  0.0472012   1.154  0.24841
## locationVisayas       0.1121092  0.0417496   2.685  0.00725 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 2362.5  on 1499  degrees of freedom
## Residual deviance: 2187.8  on 1493  degrees of freedom
## AIC: 6574.7
##
## Number of Fisher Scoring iterations: 5
```

summary() 给出的关于因子 location 的检验结果是基于各个哑变量的，如果要检验整个因子，可以使用偏差来比较：

```r
anova(glmp3, glmp41, test = "Chisq")
```

```
## Analysis of Deviance Table
##
```

```
## Model 1: total ~ age + I((age - 53)^2)
## Model 2: total ~ age + I((age - 53)^2) + location
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1      1497     2200.9
## 2      1493     2187.8  4   13.144  0.01059 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

p 值为 0.01，在 0.05 水平下显著。

继续增加表征贫困与否的变量 roof，这是一个二值变量：

```
glmp42 <- glm(total ~ age + I((age - 53)^2) + location + roof,
    family = poisson, data = dfp)
summary(glmp42)
```

```
##
## Call:
## glm(formula = total ~ age + I((age - 53)^2) + location + roof,
##     family = poisson, data = dfp)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.9900  -0.9281  -0.1070   0.5912   5.0255
##
## Coefficients:
##                                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)                       1.547e+00  7.698e-02  20.102  < 2e-16 ***
## age                              -4.160e-03  1.036e-03  -4.017  5.9e-05 ***
## I((age - 53)^2)                  -7.034e-04  6.419e-05 -10.958  < 2e-16 ***
## locationDavaoRegion              -1.655e-02  5.384e-02  -0.307  0.75855
## locationIlocosRegion              6.299e-02  5.269e-02   1.195  0.23194
## locationMetroManila               5.322e-02  4.721e-02   1.127  0.25967
## locationVisayas                   1.168e-01  4.196e-02   2.784  0.00537 **
## roofPredominantly Strong Material 4.752e-02  4.359e-02   1.090  0.27564
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 2362.5  on 1499  degrees of freedom
```

```
## Residual deviance: 2186.6  on 1492  degrees of freedom
## AIC: 6575.5
##
## Number of Fisher Scoring iterations: 5
```

summary() 给出的结果表明在已有年龄和区域自变量的前提下，增加 roof 自变量对模型没有显著意义。

尝试增加低于五岁的儿童人数的自变量：

```r
glmp43 <- glm(total ~ age + I((age - 53)^2) + location + numLT5,
    family = poisson, data = dfp)
summary(glmp43)
```

```
##
## Call:
## glm(formula = total ~ age + I((age - 53)^2) + location + numLT5,
##     family = poisson, data = dfp)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6909  -0.8391  -0.1610   0.5856   3.5535
##
## Coefficients:
##                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)           1.248e+00  6.919e-02  18.035   <2e-16 ***
## age                   2.839e-04  1.056e-03   0.269    0.788
## I((age - 53)^2)      -7.809e-04  6.475e-05 -12.059   <2e-16 ***
## locationDavaoRegion  -6.309e-02  5.386e-02  -1.171    0.241
## locationIlocosRegion  8.834e-03  5.277e-02   0.167    0.867
## locationMetroManila   1.800e-02  4.726e-02   0.381    0.703
## locationVisayas       2.371e-02  4.204e-02   0.564    0.573
## numLT5                3.729e-01  1.632e-02  22.853   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 2362.5  on 1499  degrees of freedom
## Residual deviance: 1736.8  on 1492  degrees of freedom
## AIC: 6125.6
##
```

```
## Number of Fisher Scoring iterations: 5
```

增加低龄儿童数自变量后，location 变量变得不再显著，用偏差进行检验：

```
glmp43b <- glm(total ~ age + I((age - 53)^2) + numLT5,
    family = poisson, data = dfp)
summary(glmp43b)
```

```
##
## Call:
## glm(formula = total ~ age + I((age - 53)^2) + numLT5, family = poisson,
##     data = dfp)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6678  -0.8414  -0.1672   0.5970   3.5762
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)      1.256e+00  5.965e-02  21.057   <2e-16 ***
## age              2.483e-04  1.053e-03   0.236    0.814
## I((age - 53)^2) -7.817e-04  6.465e-05 -12.092   <2e-16 ***
## numLT5           3.742e-01  1.617e-02  23.139   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 2362.5  on 1499  degrees of freedom
## Residual deviance: 1740.8  on 1496  degrees of freedom
## AIC: 6121.6
##
## Number of Fisher Scoring iterations: 5
```

```
anova(glmp43b, glmp43, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: total ~ age + I((age - 53)^2) + numLT5
## Model 2: total ~ age + I((age - 53)^2) + location + numLT5
```

```
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1      1496     1740.8
## 2      1492     1736.8  4   4.0023   0.4057
```

检验结果表明加入 numLT5 后 location 可以取消。但是，numLT5 本身就是一个人口数变量，用作解释家庭人口数的自变量并不太合适。

最终应该选择的模型是包含了年龄、年龄二次项、location 变量的模型。

36.2.6 发病率模型

流行病学研究中，经常要研究某种病的发病率如何受到自变量的影响。虽然这属于二项分布，但是因为发病率都比较低，可以用泊松分布作为近似模型。设 T 为总人数和累计时间（如人年单位），λ 为 T 对应的平均发病人数，则针对参数

$$\rho = \lambda/T$$

建模，这样才可以对人口数不同的地区使用统一的模型。利用泊松回归，需要写成

$$\log(\lambda) = \beta_0 + \beta_1 x + \log(T),$$

`glm()` 公式中可以用 `offset()` 表示系数固定等于 1 的自变量。

例 36.1. ISwR 包的 eba1977 包含类丹麦的 4 个城市肺癌发病率数据，以年龄和城市为自变量建模。

```
data(eba1977, package = "ISwR")
str(eba1977)
```

```
## 'data.frame':    24 obs. of  4 variables:
##  $ city : Factor w/ 4 levels "Fredericia","Horsens",..: 1 2 3 4 1 2 3 4 1 2 ...
##  $ age  : Factor w/ 6 levels "40-54","55-59",..: 1 1 1 1 2 2 2 2 3 3 ...
##  $ pop  : int  3059 2879 3142 2520 800 1083 1050 878 710 923 ...
##  $ cases: int  11 13 4 5 11 6 8 7 11 15 ...
```

用泊松回归建模：

```
glm.eba <- glm(
  cases ~ city + age + offset(log(pop)),
  family = poisson,
  data = eba1977)
summary(glm.eba)
```

```
##
## Call:
```

```
## glm(formula = cases ~ city + age + offset(log(pop)), family = poisson,
##     data = eba1977)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.63573  -0.67296  -0.03436   0.37258   1.85267
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -5.6321     0.2003 -28.125  < 2e-16 ***
## cityHorsens  -0.3301     0.1815  -1.818   0.0690 .
## cityKolding  -0.3715     0.1878  -1.978   0.0479 *
## cityVejle    -0.2723     0.1879  -1.450   0.1472
## age55-59      1.1010     0.2483   4.434 9.23e-06 ***
## age60-64      1.5186     0.2316   6.556 5.53e-11 ***
## age65-69      1.7677     0.2294   7.704 1.31e-14 ***
## age70-74      1.8569     0.2353   7.891 3.00e-15 ***
## age75+        1.4197     0.2503   5.672 1.41e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 129.908  on 23  degrees of freedom
## Residual deviance:  23.447  on 15  degrees of freedom
## AIC: 137.84
##
## Number of Fisher Scoring iterations: 5
```

注意数据 1968-1971 共 4 年的数据，所以模型的发病率 ρ 应理解为 4 年发病率。从残差偏差看模型拟合基本充分，实际上 p 值为：

```r
1 - pchisq(23.447, 15)
```

```
## [1] 0.07509937
```

因为两个自变量都是类别变量，所以系数估计中的检验仅是针对与基线的比较，不能作为某个自变量是否显著的检验。可以用 `anova()` 进行检验，但 `anova()` 执行第一类检验，仅最后一个变量的结果才是排除单个变量的模型与完整模型的比较。用 `drop1()` 可以逐个检验变量是否显著，作第三类检验：

```r
drop1(glm.eba, test="Chisq")
```

```
## Single term deletions
##
## Model:
## cases ~ city + age + offset(log(pop))
##        Df Deviance    AIC     LRT Pr(>Chi)
## <none>      23.447 137.84
## city    3   28.307 136.69   4.859   0.1824
## age     5  126.515 230.90 103.068   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

其中 age 最为显著，city 也可以保留。

为了在各年龄组比较，我们建立一个所有年龄组有各自截距的模型：

```r
glm.eba2 <- glm(
  cases ~ -1 + age + city + offset(log(pop)),
  family = poisson,
  data = eba1977)
summary(glm.eba2)
```

```
##
## Call:
## glm(formula = cases ~ -1 + age + city + offset(log(pop)), family = poisson,
##     data = eba1977)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.63573  -0.67296  -0.03436   0.37258   1.85267
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## age40-54     -5.6321     0.2003 -28.125   <2e-16 ***
## age55-59     -4.5311     0.2073 -21.861   <2e-16 ***
## age60-64     -4.1135     0.1869 -22.005   <2e-16 ***
## age65-69     -3.8644     0.1841 -20.993   <2e-16 ***
## age70-74     -3.7752     0.1897 -19.897   <2e-16 ***
## age75+       -4.2124     0.2082 -20.233   <2e-16 ***
```

```
## cityHorsens  -0.3301     0.1815  -1.818   0.0690 .
## cityKolding  -0.3715     0.1878  -1.978   0.0479 *
## cityVejle    -0.2723     0.1879  -1.450   0.1472
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 50361.048  on 24  degrees of freedom
## Residual deviance:    23.447  on 15  degrees of freedom
## AIC: 137.84
##
## Number of Fisher Scoring iterations: 5
```

从 age 的系数看，肺癌发病率随年龄而上升，但 75 岁以上组又恢复到了 60 岁时的发病率。

对于发病率的泊松回归模型，自变量的系数 β 代表自变量每增加 1，对数发病率会增加 β，从而发病率会增加到原来的 e^{β} 倍。显示自变量的系数所代表的发病率变化倍数：

```r
cbind(Multple=coef(glm.eba), confint(glm.eba)) |>
  exp() |> round(4)
```

```
## Waiting for profiling to be done...
```

```
##             Multple  2.5 %  97.5 %
## (Intercept)  0.0036 0.0024  0.0052
## cityHorsens  0.7189 0.5027  1.0259
## cityKolding  0.6897 0.4756  0.9950
## cityVejle    0.7616 0.5251  1.0990
## age55-59     3.0072 1.8430  4.9010
## age60-64     4.5659 2.9072  7.2363
## age65-69     5.8574 3.7483  9.2489
## age70-74     6.4036 4.0430 10.2119
## age75+       4.1357 2.5229  6.7624
```

两个自变量都是因子，都是和基线水平的比较，比如，age55-59 的倍数 3.0072，是与基线水平 age40-54 的发病率比值。从相邻年龄组之间的倍数变化看，年龄每提高 5 岁，肺癌发病率平均提高 60% 到 300%，尤其以从 44-54 年龄组到 55-60 年龄组的危险率增加最多。

36.2.7 其它问题

在泊松回归这样的因变量为计数变量问题中，可能会出现如下的问题：

- 零过多数据（zero inflated data）。数据中存在比泊松分布应有的个数更多的零。这一般是来自于两个总体的混合，其中一个总体服从泊松分布，而另一个总体总是等于零。
- 过度离势（over dispersion）问题。数据中因变量的方差显著地高于泊松分布应有的方差。这可能来自于缺少了重要的解释变量导致不同组被混在一起变成了混合泊松分布，或者观测的独立性不成立，等等。如果存在过度离势，参数估计的标准误差会过于低估，导致显著性检验结果过于乐观，使得本来不显著的自变量变得显著。

36.3 逻辑斯谛回归

36.3.1 模型

当因变量 Y 是零壹变量（二值变量）时，即 Y 表示分两类的类别，用取值 1 和 0 表示，将取值 1 称为成功，我们关心的是成功概率 $p = P(Y = 1)$。这是一个 $[0, 1]$ 区间内的值。如果把 Y 当作一般因变量做线性回归，会给出不合理的结果，比如负值，另外线性回归假定误差项为正态分布在这里也不适用。

如果 Y 是 m 次试验中成功的次数，这时可以设 Y 服从二项分布 $\mathrm{B}(m, p)$，关心的是成功概率 p，也可以归入相同的模型。

为此考虑对应于二项分布的广义线性回归模型：

$$\begin{aligned} Y \sim & B(m, p) \\ g(p) = & \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k. \end{aligned}$$

其中 m 为试验次数，Y 为 m 次试验中成功次数，p 为给定 $x_1, \ldots, x_k$ 条件下的成功概率。一般取联系函数 $g(p)$ 为 logit 函数

$$\mathrm{logit}(p) = \ln\left(\frac{p}{1-p}\right).$$

此模型称为逻辑斯谛回归模型。$\frac{p}{1-p}$ 是成功概率与失败概率的比值，称为发生比 (odds，或优势)。$\ln \frac{p}{1-p}$ 称为对数发生比。

逻辑斯谛回归模型有如下的模型假定：

- 因变量表示成败型结果，为零壹变量或者已知试验次数中成功次数；
- 各个观测独立；
- 如果某个观测的试验次数为 m，成功概率为 p，则因变量方差为 $mp(1-p)$，方差不等于常数值，与期望值 mp 有关系；
- 对数发生比 $\ln \frac{p}{1-p}$ 与自变量之间为线性关系。

比如，仅有一个自变量 x 时，数据可以是 $(x_i, y_i), i = 1, 2, \ldots, n$，其中 y_i 仅取 0 或取 1，模型为

$$\begin{aligned} y_i \sim & \mathrm{b}(1, p_i), \\ \mathrm{logit}(p_i) = & a + b x_i,\ i = 1, 2, \ldots, n. \end{aligned}$$

数据也可以是 $(x_i, y_i, z_i), i = 1, 2, \ldots, n$，其中 y_i 代表第 i 个观测的成功数，z_i 代表失败数，模型为

$$\begin{aligned} y_i \sim & \mathrm{B}(m_i, p_i), \\ \mathrm{logit}(p_i) = & a + bx_i,\ i = 1, 2, \ldots, n. \end{aligned}$$

其中 $m_i = y_i + z_i$ 是第 i 个观测的试验次数，是非随机的正整数值。

36.3.2 最大似然估计

先考虑观测的因变量是二值因变量情形。数据为 (x_i, y_i), $i = 1, \ldots, n$，一个观测对似然函数的贡献为

$$p_i^{y_i}(1 - p_i)^{1-y_i},$$

其中

$$p_i = \mathrm{logit}^{-1}(a + bx_i) = \frac{e^{a+bx_i}}{1 + e^{a+bx_i}},$$

对数似然函数为

$$\begin{aligned} \ell(a, b) = & \sum_{i=1}^{n} \{y_i \ln p_i + (1 - y_i) \ln(1 - p_i)\} \\ = & \sum_{i=1}^{n} \left\{ y_i \ln \frac{p_i}{1 - p_i} + \ln(1 - p_i) \right\} \\ = & \sum_{i=1}^{n} \left\{ y_i(a + bx_i) - \ln(1 + e^{a+bx_i}) \right\}. \end{aligned}$$

若观测是二项数据 (x_i, m_i, y_i)，其中 $y_i \sim \mathrm{B}(m_i, p_i)$，

$$p_i = \mathrm{logit}^{-1}(a + bx_i),$$

则第 i 个观测对似然函数的贡献为

$$\binom{m_i}{y_i} p_i^{y_i}(1 - p_i)^{m_i - y_i},$$

对于对数似然函数的贡献为

$$\begin{aligned} & \ln \binom{m_i}{y_i} + y_i \ln p_i + (m_i - y_i) \ln(1 - p_i) \\ = & \ln \binom{m_i}{y_i} + y_i \ln \frac{p_i}{1 - p_i} + m_i \ln(1 - p_i) \\ = & \ln \binom{m_i}{y_i} + y_i \ln(a + bx_i) - m_i \ln(1 + e^{a+bx_i}), \end{aligned}$$

忽略与待估参数 a, b 无关的常数项后，对数似然函数为因变量的对数似然函数为

$$\ell(a, b) = \sum_{i=1}^{n} [y_i \ln p_i + (m_i - y_i) \ln(1 - p_i)] \tag{36.1}$$

$$= \sum_{i=1}^{n} \left[y_i \ln(a + bx_i) - m_i \ln(1 + e^{a+bx_i}) \right]. \tag{36.2}$$

需要用数值迭代算法求最大似然估计。若样本为满足模型的随机样本，最大似然估计是相合估计，渐近有效估计，具有渐近正态分布。

得到最大似然估计 $(\hat{a}, \hat{b})$ 后，一般用信息阵的逆矩阵估计其协方差阵，信息阵为：

$$I(a,b) = -E\left[\frac{\partial^2 \ell(a,b)}{\partial(a,b)\partial(a,b)^T}\right],$$

在用数值迭代算法计算最大似然估计时，一般会得到最大值点处的对数似然函数的海色阵，加上负号并求逆矩阵，就可以作为参数协方差阵估计。

有了参数估计的协方差阵估计，就得到了参数估计的标准误差。设参数估计为 $\hat{\theta}$，估计的方差开平方根作为其抽样分布标准差估计，称为标准误差，记为 $\mathrm{SE}(\hat{\theta})$。令

$$Z = \frac{\hat{\theta}}{\mathrm{SE}(\hat{\theta})},$$

可以用 Z 统计量检验 $H_0 : \theta = 0$，在 H_0 成立、大样本且满足适当正则性条件时 Z 近似服从标准正态分布。也可以用 Z^2 近似服从 $\chi^2(1)$ 分布进行检验。这样对回归系数进行的检验称为 Wald 类型的检验。此检验对中小样本有一定缺陷，当参数的实际数值绝对值较大时，容易错判成不显著。另一种可用的方法是下面所述的利用偏差统计量的方法。

36.3.3 偏差分析

当因变量是多次试验的二项分布结果时，虽然可以计算比例与模型预测概率的差，但残差平方和与线性正态情形的回归模型的残差平方和不同，所以用偏差 (deviance) 代替了残差平方和。

对 $y_i \sim \mathrm{B}(m_i, p_i)$，设 $\hat{p}_i = \mathrm{logit}^{-1}(\hat{a} + \hat{b}x_i)$, $\hat{y}_i = m_i\hat{p}_i$，定义逻辑斯谛回归的偏差为

$$G^2 = 2\sum_{i=1}^{n}\left[y_i \log\left(\frac{y_i}{\hat{y}_i}\right) + (m_i - y_i)\log\left(\frac{m_i - y_i}{m_i - \hat{y}_i}\right)\right].$$

当因变量观测值 y_i 与预测值 $\hat{y}_i$ 都很接近时，偏差 G^2 的值比较小；当模型不准确时，偏差的值比较大。记此模型为 M，设模型 M 有 p 个回归系数，模型 M 拟合充分时一定条件下偏差服从自由度为 $n-p$ 的卡方分布。下面解释上述偏差公式。

在 R 软件的 `glm()` 函数计算时，“饱和模型”（称为模型 S）就是对每个观测估计一个不同的 $\hat{y}_i$，即 $\hat{p}_i = y_i/m_i$, $\hat{y}_i = y_i$。这样有 n 个参数。但是要注意，饱和模型使用 n 个参数，默认假定了各个观测的自变量组合都是互不相同的，因为某一个特定的自变量组合如果在多个观测中重复出现应该给出相同的预测值，R 中并没有自动进行各个观测的自变量组合互不相同的验证。

在模型 S 下，最大化的对数似然函数值为(36.1)式中 $p_i = y_i/m_i$; 在模型 M 下，最大化的对数似然函数值为(36.1)式

中 $p_i = \hat{p}_i$; 于是两个模型的对数似然函数差的 -2 倍为

$$
\begin{aligned}
G^2 =& -2\left[\max\ln(L_M) - \max\ln(L_S)\right] \\
=& -2\sum_{i=1}^{n}\left[y_i\log\hat{p}_i + (m_i - y_i)\log(1-\hat{p}_i)\right] \\
& +2\sum_{i=1}^{n}\left[y_i\log\frac{y_i}{m_i} + (m_i - y_i)\log(1-\frac{y_i}{m_i})\right] \\
=& 2\sum_{i=1}^{n}\left[y_i\frac{y_i}{\hat{y}_i} + (m_i - y_i)\log(\frac{m_i - y_i}{m_i - \hat{y}_i})\right]
\end{aligned}
$$

其中 $\hat{y}_i = m_i\hat{p}_i$ 是模型 M 的预测值。可见偏差是模型 M 与饱和模型 S 比较的对数似然比统计量，相应的检验是似然比检验。在 “模型 M 与模型 S 相同” 的零假设成立时，G^2 在大样本情况下近似服从 $\chi^2(n-p)$ 分布，p 为回归参数 β 的维数，在上述推导中有两个回归参数 a, b，所以 $p = 2$。在 R 软件输出中偏差显示为 “残差偏差”(residual deviance)。可以将残差偏差按照 $n-p$ 自由度的卡方分布计算右侧 p 值，因为是将模型与饱和模型比较，所以 p 值小于检验水平 α 时认为模型拟合不足。

如果有两个模型 M 和 C，C 是 M 的特例（比如令 M 中某个参数等于 0），称这两个模型有嵌套关系，模型 C 嵌套于 M 中。所有模型都是饱和模型 S 的嵌套模型。这时，计算 M 和 C 的残差偏差 G_M^2 和 G_C^2，则其差值 $G_M^2 - G_C^2$ 在两个模型相同的零假设下近似服从卡方分布，自由度等于两个模型的参数个数之差。可以用这样的方法检验两个嵌套模型是否有显著差异。在 R 中用 `anova()` 函数加 `test="Chisq"` 选项进行这样的检验。

R 的输出结果有 `null deviance`，这是空模型（仅有截距项、没有自变量的模型）的残差偏差，即空模型与完全模型的比较。`null deviance` 是所有模型的 `residual deviance` 中最大的。将模型偏差（残差偏差）与空模型的偏差进行比较，就可以检验所有自变量的系数（除截距项以外的回归系数）同时为 0 的零假设。显然，空模型嵌套在所有其它模型中。

因为偏差统计量的近似卡方分布是大样本性质，有文献认为，偏差统计量在自变量都是类别变量，数据为列联表数据，且每个单元格观测数都达到 5 或 10 以上时比较准确；如果自变量中有连续测量的自变量，使得每个单元格重复试验次数很少，偏差统计量的分布不一定准确。

36.3.4 足球罚球得分概率案例

考虑足球比赛罚球得分的概率，研究得分概率是否与罚球球队当前是否比分领先有关系。若干次罚球的汇总表格数据如下：

	得分	未得分	总数
领先	22	2	24
不领先	141	39	180
总数	163	41	204

不同于概率统计中用概率表示可能性，在比赛、赌博中经常使用发生比 (odds，或称优势、几率、赔率) 的概念。在本例中，罚球时比分领先的情况下，罚球成功了 22 例，失败了 2 例，则罚球成功的发生比是 $22:2 = 11:1 = 1:0.09$，其含义是每罚球失败一次，平均有 11 次成功；每罚球成功一次，平均有 0.0909 次失败。从概率论的角度，以 p 为罚球成功概率，则发生比是 $\frac{p}{1-p}$。

当罚球的球队比分不领先时，罚球成功的发生比为 $141:39=3.62:1=1:0.28$，这个发生比远低于领先的情况。计算领先时的发生比和不领先时的发生比的比值：

$$\frac{22/2}{141/39}=3.04,$$

称这个比例为优势比 (odds ratio)，可以看出比分领先情况下的罚球成功概率相对较高。

发生比是一个取值于 $(0,\infty)$ 的正数，优势比也是取值于 $(0,\infty)$ 的正数，都是以 1 作为比较的基准值。发生比为 1，表明成功和失败概率相等，发生比越大，成功概率越大。优势比等于 1，表明两个组的成功概率相等；优势比越大，表明第一个组的成功概率越大。但是，两个组的发生比等于 3.04，不能说第一个组的成功概率是第二个组成功概率的 3.04 倍，因为设两个组的成功概率分别为 p_1,p_2，则两个组的优势比等于

$$\frac{p_1/(1-p_1)}{p_2/(1-p_2)}=\frac{p_1}{p_2}\frac{1-p_2}{1-p_1},$$

比两个组的概率之比 $\frac{p_1}{p_2}$ 还多了 $\frac{1-p_2}{1-p_1}$ 这一项。

定义自变量

$$X=\begin{cases}1, & \text{罚球球队比分领先},\\ 0, & \text{比分不领先},\end{cases}$$

则问题用逻辑回归表示，可以写成

$$\begin{aligned}Y\sim&\mathrm{B}(m,p),\\ \ln\frac{p}{1-p}=&\beta_0+\beta_1X.\end{aligned}$$

问题的数据比较简单，可以看成只有 $X=0$ 和 $X=1$ 两个观测，于是当 $X=0$(罚球球队比分不领先时)，记这时罚球成功概率为 p_0，则

$$\ln\frac{p_0}{1-p_0}=\beta_0;$$

当 $X=1$，即罚球球队比分领先时，记这时罚球成功概率为 p_1，则

$$\ln\frac{p_1}{1-p_1}=\beta_0+\beta_1;$$

由此可以解释 β_1 的含义为：

$$\begin{aligned}\beta_1=&(\beta_0+\beta_1)-\beta_0=\ln\frac{p_1}{1-p_1}-\ln\frac{p_0}{1-p_0}\\ =&\ln\frac{\frac{p_1}{1-p_1}}{\frac{p_0}{1-p_0}},\end{aligned}$$

其中 $\frac{\frac{p_1}{1-p_1}}{\frac{p_0}{1-p_0}}$ 是比分领先相对于比分不领先时的优势比 (odds ratio)，所以 e^{β_1} 等于优势比。

用二项分布的分布概率可以写出模型的似然函数为

$$\begin{aligned}L(\beta_0,\beta_1)=&\binom{24}{22}p_1^{22}(1-p_1)^2\binom{180}{141}p_0^{141}(1-p_1)^{39}\\ \propto&\left(\frac{e^{\beta_0+\beta_1}}{1+e^{\beta_0+\beta_1}}\right)^{22}\left(1-\frac{e^{\beta_0+\beta_1}}{1+e^{\beta_0+\beta_1}}\right)^{2}\left(\frac{e^{\beta_0}}{1+e^{\beta_0}}\right)^{141}\left(1-\frac{e^{\beta_0}}{1+e^{\beta_0}}\right)^{39}.\end{aligned}$$

可以用数值优化方法求得最大似然估计 $\hat{\beta}_0 = 1.2852$, $\hat{\beta}_1 = 1.1127$。从而优势比的估计为 $e^{1.1127} = 3.04$，与直接计算优势比完全相同。

使用 R 的 `glm()` 函数来计算这个例子。将因变量输入为由成功数与失败数两列组成的矩阵。

```
dga <- tibble(
  lead = factor(c("Lead", "Not Lead"),
    levels = c("Not Lead", "Lead")),
  score = c(22, 141),
  fail = c(2, 39) )
glmga1 <- glm(cbind(score, fail) ~ lead,
  family = binomial,
  data = dga)
summary(glmga1)
```

```
##
## Call:
## glm(formula = cbind(score, fail) ~ lead, family = binomial, data = dga)
##
## Deviance Residuals:
## [1]  0  0
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   1.2852     0.1809   7.104 1.22e-12 ***
## leadLead      1.1127     0.7604   1.463    0.143
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2.7930e+00  on 1  degrees of freedom
## Residual deviance: 4.2188e-15  on 0  degrees of freedom
## AIC: 11.789
##
## Number of Fisher Scoring iterations: 4
```

在这个例子中，饱和模型只需要给领先和不领先分别指定得分概率，有两个自由参数；要拟合的模型也是恰好有 β_0 和 β_1 两个参数，所以两个模型是等价的，残差偏差 (Residual deviance) 等于负二倍的最大对数似然函数值的差，等于零。空模型的偏差（Null deviance）有一个自由度，所谓空模型是只有截距项 β_0 的模型，有一个自由参数，所以和饱和模型相比相差一个自由度。

领先与不领先时的罚球得分概率是否有显著差异？$H_0: \beta_1 = 0$ 代表优势比等于 1，这时两种情况的得分概率相同。虽然优势比达到 3.04，但是程序结果中系数 β_1 的显著性 p 值只有 0.143，在 0.05 水平下不显著。这里的标准误差是利用最大似然估计得到的信息矩阵估计给出的估计，z 统计量和相应 p 值利用了参数估计的渐近正态分布。

用如下方法产生优势比与优势比的置信区间：

```
exp(cbind(OR = coef(glmga1), confint(glmga1)))
```

```
## Waiting for profiling to be done...
```

```
##                   OR     2.5 %    97.5 %
## (Intercept) 3.615385 2.5637929  5.221335
## leadLead    3.042553 0.8453435 19.505910
```

可见领先相对于非领先的优势比为 3.04, 95% 置信区间为 $(0.85, 19.51)$，置信区间包含 1 说明两组的罚球成功概率没有显著差异。

因为这是一个四格表数据，我们还可以用 Fisher 精确检验比较领先与非领先的罚球成功概率，或将优势比与 1 比较：

```
fisher.test(as.matrix(dga[,2:3]))
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  as.matrix(dga[, 2:3])
## p-value = 0.1758
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##   0.693731 27.710569
## sample estimates:
## odds ratio
##   3.029845
```

Fisher 精确检验是基于多项分布，得到的 p 值与逻辑斯谛回归结果相近但不相等。

36.3.5 餐馆评分问题

数据框 Michelin 中包含了 Zagat Survey 2006 考察的 164 家法餐馆的顾客评分和是否被米其林推荐收录。数据中每一行是一个评分值对应的数据，变量 `Food` 是 Zagat 评分，取值从 15 到 28，`InMichelin` 是某一评分被米其林收录家数，`NotInMichelin` 是未被米其林收录家数，`mi` 是该评分的总家数，`proportion` 是收录比例。

以是否收录作为因变量，以 Zagat 评分作为自变量，建立逻辑斯谛回归：

```r
glm.mic1 <- glm(
  cbind(InMichelin,NotInMichelin) ~ Food,
  family=binomial,
  data=MichelinFood)
summary(glm.mic1)
```

```
##
## Call:
## glm(formula = cbind(InMichelin, NotInMichelin) ~ Food, family = binomial,
##     data = MichelinFood)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4850  -0.7987  -0.1679   0.5913   1.5889
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -10.84154    1.86236  -5.821 5.84e-09 ***
## Food          0.50124    0.08768   5.717 1.08e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 61.427  on 13  degrees of freedom
## Residual deviance: 11.368  on 12  degrees of freedom
## AIC: 41.491
##
## Number of Fisher Scoring iterations: 4
```

自变量 Food 的系数为 0.5012，检验结果显著。

结果中残差偏差 11.4，自由度 12，残差偏差与自由度相比不大，说明模型拟合较好。

优势比和优势比置信区间：

```r
cbind(OR = coef(glm.mic1), confint(glm.mic1)) |>
  exp() |> round(4)
```

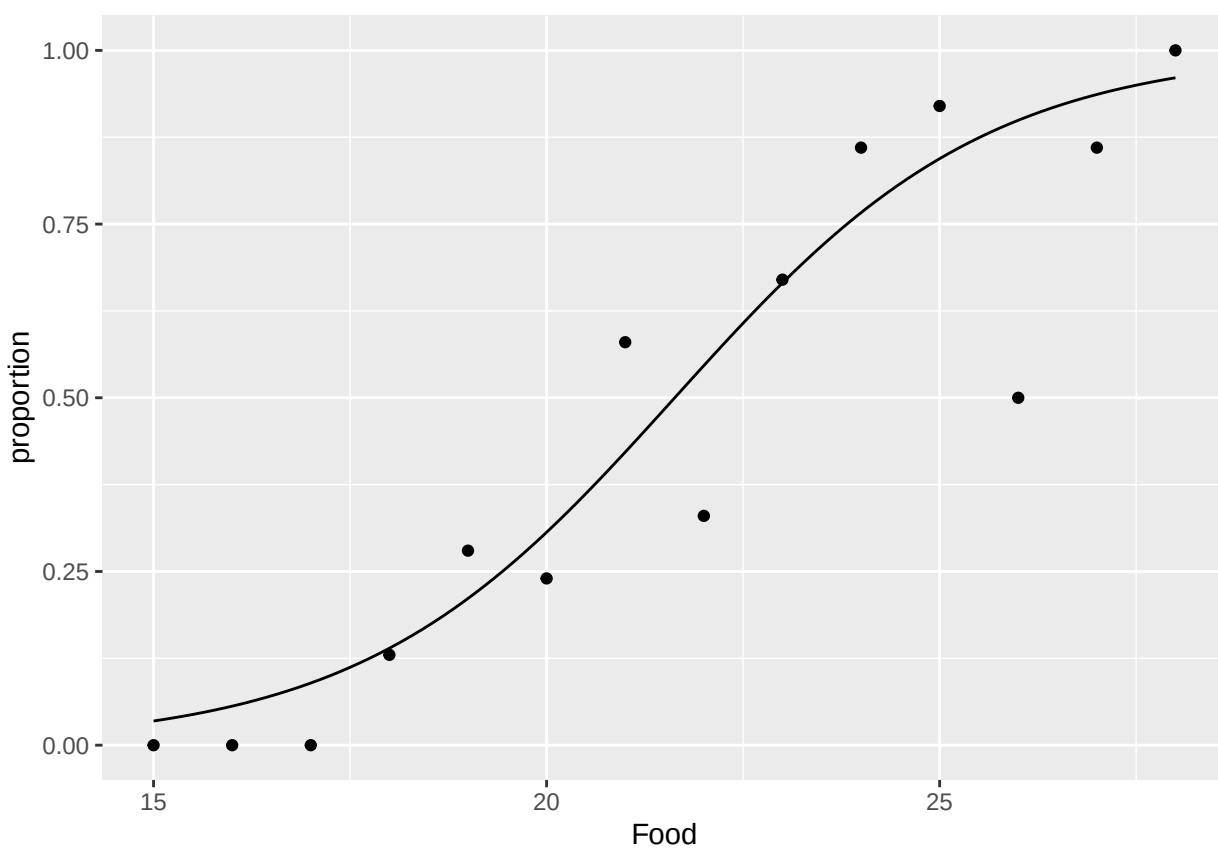
```
## Waiting for profiling to be done...
```

```
##                  OR  2.5 % 97.5 %
## (Intercept) 0.0000 0.0000 0.0006
## Food        1.6508 1.4065 1.9871
```

自变量 Food 的优势比为 1.65，表示 Zaga 评分每提高 1 分，被米其林收录的发生比将提高到原来的 $e^{0.5012} = 1.65$ 倍，即发生比提高 65%；优势比的置信区间为 $(1.41, 1.99)$ 不包含 1，所以优势比与 1 有显著差异。

因为数据是二项数据，有重复，所以我们可以将数据中的收录比例与模型预测的收录概率进行对照，作图表示：

```
dmicpr <- tibble(
  Food = seq(15, 28, length.out=100))
dmicpr <- dmicpr |>
  mutate(p = predict(glm.mic1, newdata = dmicpr,
  type = "response"))
ggplot(data = MichelinFood, mapping = aes(
  x = Food, y = proportion)) +
  geom_point() +
  geom_line(data = dmicpr, mapping = aes(
    x = Food, y = p  ))
```



36.3.6 修建铁路意见投票情况案例

考虑在某地的 4 个不同区域进行关于是否修建铁路的意见征询投票的数据分析，赞同修建铁路的比例可能受到所在区域到要修建的铁路的距离 (单位：英里) 的影响，距离越远越容易反对修建，可能受到投票人中黑人比例的影响。数据如下：

黑人百分比	距离	赞成票数	总票数
58.40	17	61	110
92.40	7	0	15
18.28	15	4	42
59.38	0	1790	1804
55.51	7	0	15
58.40	12	16	68
83.39	2	31	42
94.54	9	87	154
88.54	0	107	118
77.19	4	90	102
38.75	7	6	23

仅使用距离作为自变量进行逻辑斯谛回归：

```
glmrr1 <- glm(
  cbind(YesVotes, NumVotes - YesVotes) ~ distance,
  family = binomial, data = drr)
summary(glmrr1)
```

```
##
## Call:
## glm(formula = cbind(YesVotes, NumVotes - YesVotes) ~ distance,
##     family = binomial, data = drr)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -6.793  -4.753  -2.852  -1.625   9.097
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.30927    0.11313   29.25   <2e-16 ***
## distance    -0.28758    0.01302  -22.09   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 988.45  on 10  degrees of freedom
## Residual deviance: 318.44  on  9  degrees of freedom
## AIC: 360.73
##
## Number of Fisher Scoring iterations: 5
```

模型说明距离对支持比例起到负面作用，距离每增加一英里，支持修建铁路的发生比降低到原来的 $e^{-0.28758} = 75\%$。此自变量是显著的。

因为数据总共 11 个观测，所以 R 程序将完全模型的参数个数定为 11，空模型是仅有截距项的模型，参数个数为 1，空模型的偏差（null deviance）的自由度为 $11 - 1 = 10$。上面的模型有两个回归系数 β_0, β_1，参数个数为 2，所以模型的偏差 (Residual deviance) 自由度为 $11 - 2 = 9$。与自由度相比，偏差统计量的值很大，说明模型拟合不足。

仅使用黑人比例建立逻辑斯谛回归模型：

```r
glmrr2 <- glm(
  cbind(YesVotes, NumVotes - YesVotes) ~ pctBlack,
  family = binomial, data = drr)
summary(glmrr2)
```

```
##
## Call:
## glm(formula = cbind(YesVotes, NumVotes - YesVotes) ~ pctBlack,
##     family = binomial, data = drr)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -12.7774   -8.4961   -7.5295   -0.7781   18.4292
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.565957   0.307642   8.341   <2e-16 ***
## pctBlack    -0.009091   0.004679  -1.943    0.052 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 988.45  on 10  degrees of freedom
```

```
## Residual deviance: 984.74  on  9  degrees of freedom
## AIC: 1027
##
## Number of Fisher Scoring iterations: 6
```

系数为 -0.009091，模型说明黑人比例对支持比例有负面影响，黑人比例每增加一个百分点，支持修建铁路的发生比降低到原来的 $e^{-0.009091} = 99\%$。从 Z 检验（Wald 类型检验）结果看显著性 p 值 0.052 显著性不明显。

使用两个自变量进行逻辑斯谛回归建模：

```r
glmrr3 <- glm(
  cbind(YesVotes, NumVotes - YesVotes)
  ~ pctBlack + distance,
  family = binomial, data = drr)
summary(glmrr3)
```

```
##
## Call:
## glm(formula = cbind(YesVotes, NumVotes - YesVotes) ~ pctBlack +
##     distance, family = binomial, data = drr)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -7.0509  -5.3271  -3.7259  -0.2912   8.7718
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  4.222021   0.296963  14.217  < 2e-16 ***
## pctBlack    -0.013227   0.003897  -3.394 0.000688 ***
## distance    -0.291735   0.013100 -22.270  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 988.45  on 10  degrees of freedom
## Residual deviance: 307.22  on  8  degrees of freedom
## AIC: 351.51
##
## Number of Fisher Scoring iterations: 5
```

注意这时回归系数的解释受到自变量之间的相关性的影响。pctBlack 的系数 -0.013227，对应于优势比 $e^{-0.013227} = 0.987$，需要解释成控制距离自变量后，黑人比例每增加一个单位，发生比降低 1.3%，与仅有黑人比例的模型相比，对支持发生比的影响与原来方向相同但影响更大了一些。在控制了黑人比例后，距离自变量对支持发生比的影响与原来接近。

从模型的残差偏差来看，如果模型很好地拟合观测数据，残差偏差应该近似服从卡方分布，自由度也在输出结果中给出，上述结果中过大的残差偏差 (residual deviance) 说明模型拟合不足。改进的方向是考虑两个自变量的交互作用，考虑方差膨胀模型 (over dispresion)，等等。

可以用残差偏差的卡方检验方法比较有或者没有黑人比例自变量的模型：

```
anova(glmrr1, glmrr3, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: cbind(YesVotes, NumVotes - YesVotes) ~ distance
## Model 2: cbind(YesVotes, NumVotes - YesVotes) ~ pctBlack + distance
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1         9     318.44
## 2         8     307.22  1   11.222 0.0008083 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

结果显著，说明增加黑人比例是有作用的。

逻辑斯谛回归采用了二项分布，估计和统计推断依据二项分布及其大样本近似进行。实际建模中，有时数据会有过度离势 (over dispersion) 问题，即因变量成功个数的实际方差大于二项分布理论方差。设第 i 个观测有 m_i 次试验，成功次数为 y_i，估计的成功概率为 $\hat{p}_i$，则 y_i 的拟合值为 $m_i\hat{p}_i$，方差应该约为 $m_i\hat{p}_i(1-\hat{p}_i)$，如果实际数据中 y_i 与拟合值的差距表现得远大于上述方差，则称数据有过度离势。

R 提供了一种简单的拟合过度离势的方法，是利用偏差统计量与自由度比值估计过度离势参数。这时，实际的似然函数已经偏离了理论上的二项分布。在 R 的 glm() 程序中用 family=quasibinomial 指定这种方法。例如：

```
glmrr11 <- glm(
  cbind(YesVotes, NumVotes - YesVotes) ~ distance + pctBlack,
  family = quasibinomial, data = drr)
summary(glmrr11)
```

```
##
## Call:
## glm(formula = cbind(YesVotes, NumVotes - YesVotes) ~ distance +
##     pctBlack, family = quasibinomial, data = drr)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -7.0509  -5.3271  -3.7259  -0.2912   8.7718
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.22202    1.99031   2.121   0.0667 .
## distance    -0.29173    0.08780  -3.323   0.0105 *
## pctBlack    -0.01323    0.02612  -0.506   0.6262
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasibinomial family taken to be 44.9194)
##
##     Null deviance: 988.45  on 10  degrees of freedom
## Residual deviance: 307.22  on  8  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 5
```

结果中的 Dispersion parameter 就是过度离势参数，估计为 44.9194，远大于 1。采用过度离势调整后，自变量 “黑人比例” 不再是显著的。

36.3.7 癌症病人康复概率自变量选择

文件 “data/remiss.csv” 中保存了癌症病人康复的数据，变量 remiss 为康复与否 (1 为康复，0 为未康复)，另外的 6 个变量为可能影响康复概率的自变量。研究这些自变量对康复概率的影响，并筛选最有解释能力的自变量。

读入数据:

```r
d.remiss <- read_csv("data/remiss.csv")
```

```
## Rows: 27 Columns: 7
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (7): remiss, cell, smear, infil, li, blast, temp
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
knitr::kable(d.remiss)
```

remiss	cell	smear	infil	li	blast	temp
1	0.80	0.83	0.66	1.9	1.100	0.996
1	0.90	0.36	0.32	1.4	0.740	0.992
0	0.80	0.88	0.70	0.8	0.176	0.982
0	1.00	0.87	0.87	0.7	1.053	0.986
1	0.90	0.75	0.68	1.3	0.519	0.980
0	1.00	0.65	0.65	0.6	0.519	0.982
1	0.95	0.97	0.92	1.0	1.230	0.992
0	0.95	0.87	0.83	1.9	1.354	1.020
0	1.00	0.45	0.45	0.8	0.322	0.999
0	0.95	0.36	0.34	0.5	0.000	1.038
0	0.85	0.39	0.33	0.7	0.279	0.988
0	0.70	0.76	0.53	1.2	0.146	0.982
0	0.80	0.46	0.37	0.4	0.380	1.006
0	0.20	0.39	0.08	0.8	0.114	0.990
0	1.00	0.90	0.90	1.1	1.037	0.990
1	1.00	0.84	0.84	1.9	2.064	1.020
0	0.65	0.42	0.27	0.5	0.114	1.014
0	1.00	0.75	0.75	1.0	1.322	1.004
0	0.50	0.44	0.22	0.6	0.114	0.990
1	1.00	0.63	0.63	1.1	1.072	0.986
0	1.00	0.33	0.33	0.4	0.176	1.010
0	0.90	0.93	0.84	0.6	1.591	1.020
1	1.00	0.58	0.58	1.0	0.531	1.002
0	0.95	0.32	0.30	1.6	0.886	0.988
1	1.00	0.60	0.60	1.7	0.964	0.990
1	1.00	0.69	0.69	0.9	0.398	0.986
0	1.00	0.73	0.73	0.7	0.398	0.986

建立包含所有 6 个自变量的模型：

```
glmr1 <- glm(
  remiss ~ cell+smear+infil+li+blast+temp,
  family=binomial, data=d.remiss)
summary(glmr1)
```

```
##
## Call:
```

```
## glm(formula = remiss ~ cell + smear + infil + li + blast + temp,
##     family = binomial, data = d.remiss)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.95165  -0.66491  -0.04372   0.74304   1.67069
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  58.0385    71.2364   0.815   0.4152
## cell         24.6615    47.8377   0.516   0.6062
## smear        19.2936    57.9500   0.333   0.7392
## infil       -19.6013    61.6815  -0.318   0.7507
## li            3.8960     2.3371   1.667   0.0955 .
## blast         0.1511     2.2786   0.066   0.9471
## temp        -87.4339    67.5735  -1.294   0.1957
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 34.372  on 26  degrees of freedom
## Residual deviance: 21.751  on 20  degrees of freedom
## AIC: 35.751
##
## Number of Fisher Scoring iterations: 8
```

可以用 drop1() 函数选择要删去的变量：

```
drop1(glmr1, test="Chisq")
```

```
## Single term deletions
##
## Model:
## remiss ~ cell + smear + infil + li + blast + temp
##        Df Deviance    AIC    LRT Pr(>Chi)
## <none>      21.751 35.751
## cell    1   22.071 34.071 0.3202  0.57147
## smear   1   21.869 33.869 0.1180  0.73121
## infil   1   21.857 33.857 0.1065  0.74417
```

```
## li      1   26.095 38.095 4.3446  0.03713 *
## blast   1   21.755 33.755 0.0044  0.94716
## temp    1   23.877 35.877 2.1264  0.14478
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

先删除 blast:

```
glmr2 <- update(glmr1, . ~ . - blast)
summary(glmr2)
```

```
##
## Call:
## glm(formula = remiss ~ cell + smear + infil + li + temp, family = binomial,
##     data = d.remiss)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.93753  -0.65855  -0.04328   0.75287   1.65475
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   57.128     69.977   0.816    0.414
## cell          24.180     47.257   0.512    0.609
## smear         18.370     56.218   0.327    0.744
## infil        -18.477     59.260  -0.312    0.755
## li             3.987      1.902   2.097    0.036 *
## temp         -86.137     64.785  -1.330    0.184
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 34.372  on 26  degrees of freedom
## Residual deviance: 21.755  on 21  degrees of freedom
## AIC: 33.755
##
## Number of Fisher Scoring iterations: 8
```

```r
drop1(glmr2, test="Chisq")
```

```
## Single term deletions
##
## Model:
## remiss ~ cell + smear + infil + li + temp
##        Df Deviance    AIC    LRT Pr(>Chi)
## <none>      21.755 33.755
## cell    1   22.073 32.073 0.3175 0.573134
## smear   1   21.869 31.869 0.1141 0.735530
## infil   1   21.858 31.858 0.1029 0.748323
## li      1   30.216 40.216 8.4606 0.003629 **
## temp    1   24.198 34.199 2.4435 0.118016
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

删去 infil:

```r
glmr3 <- update(glmr2, . ~ . - infil)
summary(glmr3)
```

```
##
## Call:
## glm(formula = remiss ~ cell + smear + li + temp, family = binomial,
##     data = d.remiss)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.87933  -0.66813  -0.07052   0.78408   1.72472
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  70.0994    58.7634   1.193   0.2329
## cell          9.8507     7.8263   1.259   0.2082
## smear         0.9124     2.9600   0.308   0.7579
## li            3.9052     1.8167   2.150   0.0316 *
## temp        -85.4447    64.2143  -1.331   0.1833
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 34.372  on 26  degrees of freedom
## Residual deviance: 21.858  on 22  degrees of freedom
## AIC: 31.858
##
## Number of Fisher Scoring iterations: 7
```

```r
drop1(glmr3, test="Chisq")
```

```
## Single term deletions
##
## Model:
## remiss ~ cell + smear + li + temp
##        Df Deviance    AIC    LRT Pr(>Chi)
## <none>      21.858 31.858
## cell    1   24.477 32.477 2.6185 0.105622
## smear   1   21.953 29.953 0.0954 0.757449
## li      1   30.434 38.434 8.5757 0.003407 **
## temp    1   24.292 32.292 2.4345 0.118695
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

删去 smear:

```r
glmr4 <- update(glmr3, . ~ . - smear)
summary(glmr4)
```

```
##
## Call:
## glm(formula = remiss ~ cell + li + temp, family = binomial, data = d.remiss)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.02043  -0.66313  -0.08323   0.81282   1.65887
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   67.634     56.888   1.189   0.2345
## cell           9.652      7.751   1.245   0.2130
```

```
## li             3.867      1.778   2.175   0.0297 *
## temp         -82.074     61.712  -1.330   0.1835
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 34.372  on 26  degrees of freedom
## Residual deviance: 21.953  on 23  degrees of freedom
## AIC: 29.953
##
## Number of Fisher Scoring iterations: 7
```

```
drop1(glmr4, test="Chisq")
```

```
## Single term deletions
##
## Model:
## remiss ~ cell + li + temp
##        Df Deviance    AIC    LRT Pr(>Chi)
## <none>      21.953 29.953
## cell    1   24.648 30.648 2.6945 0.100698
## li      1   30.829 36.829 8.8752 0.002891 **
## temp    1   24.341 30.341 2.3874 0.122321
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

剩下的 cell, li, temp 这三个变量可以保留。

可用直接使用逐步回归进行自变量筛选:

```
glmr5 <- step(glm(
  remiss ~ cell + smear + infil + li + blast + temp,
  family=binomial, data=d.remiss))
```

```
## Start:  AIC=35.75
## remiss ~ cell + smear + infil + li + blast + temp
##
##         Df Deviance    AIC
## - blast  1   21.755 33.755
## - infil  1   21.857 33.857
```

```
## - smear  1   21.869 33.869
## - cell   1   22.071 34.071
## <none>       21.751 35.751
## - temp   1   23.877 35.877
## - li     1   26.095 38.095
##
## Step:  AIC=33.76
## remiss ~ cell + smear + infil + li + temp
##
##         Df Deviance    AIC
## - infil  1   21.858 31.858
## - smear  1   21.869 31.869
## - cell   1   22.073 32.073
## <none>       21.755 33.755
## - temp   1   24.198 34.199
## - li     1   30.216 40.216
##
## Step:  AIC=31.86
## remiss ~ cell + smear + li + temp
##
##         Df Deviance    AIC
## - smear  1   21.953 29.953
## <none>       21.858 31.858
## - temp   1   24.292 32.292
## - cell   1   24.477 32.477
## - li     1   30.434 38.434
##
## Step:  AIC=29.95
## remiss ~ cell + li + temp
##
##        Df Deviance    AIC
## <none>      21.953 29.953
## - temp  1   24.341 30.341
## - cell  1   24.648 30.648
## - li    1   30.829 36.829
```

```
summary(glmr5)
```

```
##
## Call:
```

```
## glm(formula = remiss ~ cell + li + temp, family = binomial, data = d.remiss)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.02043  -0.66313  -0.08323   0.81282   1.65887
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   67.634     56.888   1.189   0.2345
## cell           9.652      7.751   1.245   0.2130
## li             3.867      1.778   2.175   0.0297 *
## temp         -82.074     61.712  -1.330   0.1835
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 34.372  on 26  degrees of freedom
## Residual deviance: 21.953  on 23  degrees of freedom
## AIC: 29.953
##
## Number of Fisher Scoring iterations: 7
```

36.4 多水平有序因变量的逻辑斯谛回归

逻辑斯谛回归是因变量为二值变量时的广义线性模型。在有些应用问题中，因变量是有序变量，比如满意程度取“低”、“中”、“高”，这时不应该分别对“中对低”、“高对低”用二值的逻辑斯谛回归建模，因为“高”和“中”之间也是可比的，分别建模的结果不一定能保证这个次序。

采用如下的理论模型。设某个潜在的连续取值的因变量为 Z，服从如下的逻辑斯谛分布：

$$F(x;\eta) = \text{logit}^{-1}(x-\eta),$$

易见 $EZ = \eta$。设观测到的取有序因子值的因变量 Y 是将 Z 的值按照如下的分点

$$\zeta_0 = -\infty < \zeta_1 < \cdots < \zeta_{K-1} < \zeta_K = +\infty$$

将实数轴分为 K 段，各段分别编码为 $1, 2, \ldots, K$，则

$$\begin{aligned} P(Y \le k) =& P(Z \le \zeta_k) = \text{logit}^{-1}(\zeta_k - \eta), \\ \text{logit}[P(Y \le k)] =& \zeta_k - \eta. \end{aligned}$$

设潜在变量 Z 的均值 η 依赖于自变量的线性组合，则有如下的有序因子值的因变量的 “比例发生比逻辑斯谛回归”(POLR) 模型：

$$\text{logit}[P(Y \leq k \mid x)] = \zeta_k - \beta x, k = 2, 3, \ldots, K. \tag{36.3}$$

这里 $k = 1$ 作为一个基线值。因为 ζ_k 未知，所以回归的截距项也归入到 $\{\zeta_k\}$ 中。模型左边是因变量类别为 k 以及低于 k 的概率的对数发生比，如果自变量的值不变，对于不同的 k 这些对数发生比只差一个常数，发生比只差一个倍数，所以称为 “比例发生比逻辑斯谛回归” 模型；因为是对 $P(Y \leq k)$ 从 $k = 1, 2, \ldots, K$ 依次建模，所以也称为累积逻辑斯谛回归模型。

注意(36.3)式左侧是小于等于的概率的一个单调增函数，右侧用了自变量线性组合的相反数，所以正系数代表了当自变量值增大时因变量取较高类别的概率变大。

MASS 包的 `polr()` 函数可以建立这种模型。以 MASS 包的 housing 数据为例，这是一个有四个分类变量的交叉频数表，调查房主满意度有关情况，四个分类变量是：

- `Sat`：房主满意度，取值为 Low, Medium, High，是有序因子;
- `Infl`：房主对物业服务影响的评价，取值为 Low, Medium, High;
- `Type`：房屋类型，包括 Tower, Atrium, Apartment, Terrace;
- `Cont`：住户之间的交往情况，取值为 Low, High。

housing 是长表格式，每个分类变量占一列，另外 `Freq` 列保存单元格频数。

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
data(housing, package="MASS")
options(contrasts = c("contr.treatment", "contr.poly"))
house.fm1 <- Sat ~ Infl + Type + Cont
house.plr <- polr(
  house.fm1,
  weights = Freq, data = housing)
summary(house.plr)
```

```
##
## Re-fitting to get Hessian
```

```
## Call:
## polr(formula = house.fm1, data = housing, weights = Freq)
##
## Coefficients:
##                 Value Std. Error t value
## InflMedium     0.5664    0.10465   5.412
## InflHigh       1.2888    0.12716  10.136
## TypeApartment -0.5724    0.11924  -4.800
## TypeAtrium    -0.3662    0.15517  -2.360
## TypeTerrace   -1.0910    0.15149  -7.202
## ContHigh       0.3603    0.09554   3.771
##
## Intercepts:
##             Value   Std. Error t value
## Low|Medium  -0.4961  0.1248    -3.9739
## Medium|High  0.6907  0.1255     5.5049
##
## Residual Deviance: 3479.149
## AIC: 3495.149
```

在结果中, `Intercepts` 是潜在因变量 Z 的分点 $\zeta_1, \ldots, \zeta_{K-1}$，即 Y 的各组之间的 Z 分界点，这里 Y 有三个组，所以给出了两个分界点。

Infl(物业服务影响) 分低、中、高三档，以 “低” 为基线，两个系数为正值，可以看出物业服务影响越大，高满意度概率越大。

住房类型分 Tower, Atrium, Apartment, Terrace 等类型，以 Tower 为基线，从系数看，Apartment, Atrium 和 Terrace 都相对于 Tower 满意度较低，满意度次高的是 Atrium，最低的是 Terrace。

住户交往情况分为低、高两类，从系数看，高的交往对应较高的满意度。

使用 anova 进行每个自变量的显著性检验：

```
house.plr2 <- update(house.plr, . ~ . - Infl)
anova(house.plr2, house.plr)
```

```
## Likelihood ratio tests of ordinal regression models
##
## Response: Sat
##                Model Resid. df Resid. Dev   Test    Df LR stat. Pr(Chi)
## 1        Type + Cont      1675   3587.389
## 2 Infl + Type + Cont      1673   3479.149 1 vs 2     2 108.2392       0
```

物业服务影响 Infl 显著。

```
house.plr3 <- update(house.plr, . ~ . - Type)
anova(house.plr3, house.plr)
```

```
## Likelihood ratio tests of ordinal regression models
##
## Response: Sat
##                Model Resid. df Resid. Dev   Test    Df LR stat.     Pr(Chi)
## 1        Infl + Cont      1676   3535.059
## 2 Infl + Type + Cont      1673   3479.149 1 vs 2     3 55.91008 4.39071e-12
```

住房类型 Type 显著。

```
house.plr4 <- update(house.plr, . ~ . - Cont)
anova(house.plr4, house.plr)
```

```
## Likelihood ratio tests of ordinal regression models
##
## Response: Sat
##                Model Resid. df Resid. Dev   Test    Df LR stat.      Pr(Chi)
## 1        Infl + Type      1674   3493.456
## 2 Infl + Type + Cont      1673   3479.149 1 vs 2     1 14.30621 0.0001553518
```

住户之间交往情况 Cont 显著。

还可以考虑加入交互作用效应的模型。

Chapter 37

整洁建模

tidymodels 包提供了针对统计建模用户简便易用的建模功能和使用规范。参见

- https://www.tmwr.org/

37.1 建模一般步骤

在建模之前，需要先获取数据。统计建模用的数据一般都保存为数据框格式。

必须对数据有充分的了解，否则在后续使用模型时，就无法判断模型所对应的各种理论假定是否满足，比如，数据是否能够认为是独立的，数据是否没有偏差地代表了所要研究的对象。

获得数据后，一般需要花费时间进行“数据清理”。数据中可能有许多错误、不一致、缺失内容，都需要进行整理修改。对缺失数据，常见的办法是仅保留没有任何缺失内容的案例，但是这常常会使得保留的数据量太小，另外有可能删去的内容使得保留的数据带有偏差。另一种对待缺失数据的方法是进行插补，这又有简单插补，如用均值代替，与利用相关性插补等方法。

在建模之前，应增加对数据的直观了解，比如，数据的收集方式，每个属性（变量）的可取值范围，变量分布情况，变量之间的关系，观测有没有明显分组聚集，等等。这些研究称为探索性数据分析（EDA）。

随后的工作是建模、评估、使用。模型大致可以分为:

- 描述性模型，如描述统计量，直方图，散点图，散点图局部曲线拟合。
- 推断性模型，如置信区间、假设检验、贝叶斯决策分析等。推断性模型的结论一般是带有概率判断的。
- 预测性模型，如回归分析、判别分析等。主要关心预测的精度。有些模型基于数学方程，如微分方程、回归模型，有些模型所需的模型假设很少，如最近邻回归。

建模的时候，一般不仅仅考虑一个模型，而是比较多个模型；对同一个模型，也有一些模型参数需要进行调整使得模型性能最优，比如乘法最小二乘中的光滑性系数的选择，提升法 (boosting) 中学习速度和停止时间的选择。模型比较和参数调优经常使用将数据交叉划分来评估性能的方法。

在研究模型表现时，有可能会发现一些数据中的不足，这时有可能需要对数据进行一些新的变换，或进一步地去收集更多的数据。

在比较模型并筛选出少数较优模型后，应对模型的性能进行更客观的判断，如果是预报问题，这通常需要使用建模时未使用过的数据来评估模型的表现。

Chapter 38

统计学习介绍

统计学习介绍的主要参考书: (James et al., 2013): Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani(2013) An Introduction to Statistical Learning: with Applications in R, Springer.

调入需要的扩展包:

```
library(leaps) # 全子集回归
library(ISLR) # 参考书对应的包
library(glmnet) # 岭回归和 lasso
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```

```
library(tree) # 树回归
library(randomForest) # 随机森林和装袋法
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(MASS)
library(gbm) # boosting
```

```
## Loaded gbm 2.1.8.1
```

```
library(e1071) # svm
```

38.1 统计学习的基本概念和一般步骤

38.1.1 统计学习的基本概念和方法

统计学习（statistical learning），也有数据挖掘（data mining），机器学习（machine learning）等称呼。主要目的是用一些计算机算法从大量数据中发现知识。方兴未艾的数据科学就以统计学习为重要支柱。方法分为有监督（supervised）学习与无监督（unsupervised）学习。

无监督学习方法如聚类问题、购物篮问题、主成分分析等。

有监督学习即统计中回归分析和判别分析解决的问题，现在又有树回归、树判别、随机森林、lasso、支持向量机、神经网络、贝叶斯网络、排序算法等许多方法。

无监督学习在给了数据之后，直接从数据中发现规律，比如聚类分析是发现数据中的聚集和分组现象，购物篮分析是从数据中找到更多的共同出现的条目（比如购买啤酒的用户也有较大可能购买火腿肠）。

有监督学习方法众多。通常，需要把数据分为训练样本和检验样本，训练样本的因变量（数值型或分类型）是已知的，根据训练样本中自变量和因变量的关系训练出一个回归函数，此函数以自变量为输入，可以输出因变量的预测值。

训练出的函数有可能是有简单表达式的（例如，logistic 回归）、有参数众多的表达式的（如神经网络），也有可能是依赖于所有训练样本而无法写出表达式的（例如 k 近邻分类）。

38.1.2 偏差与方差折衷

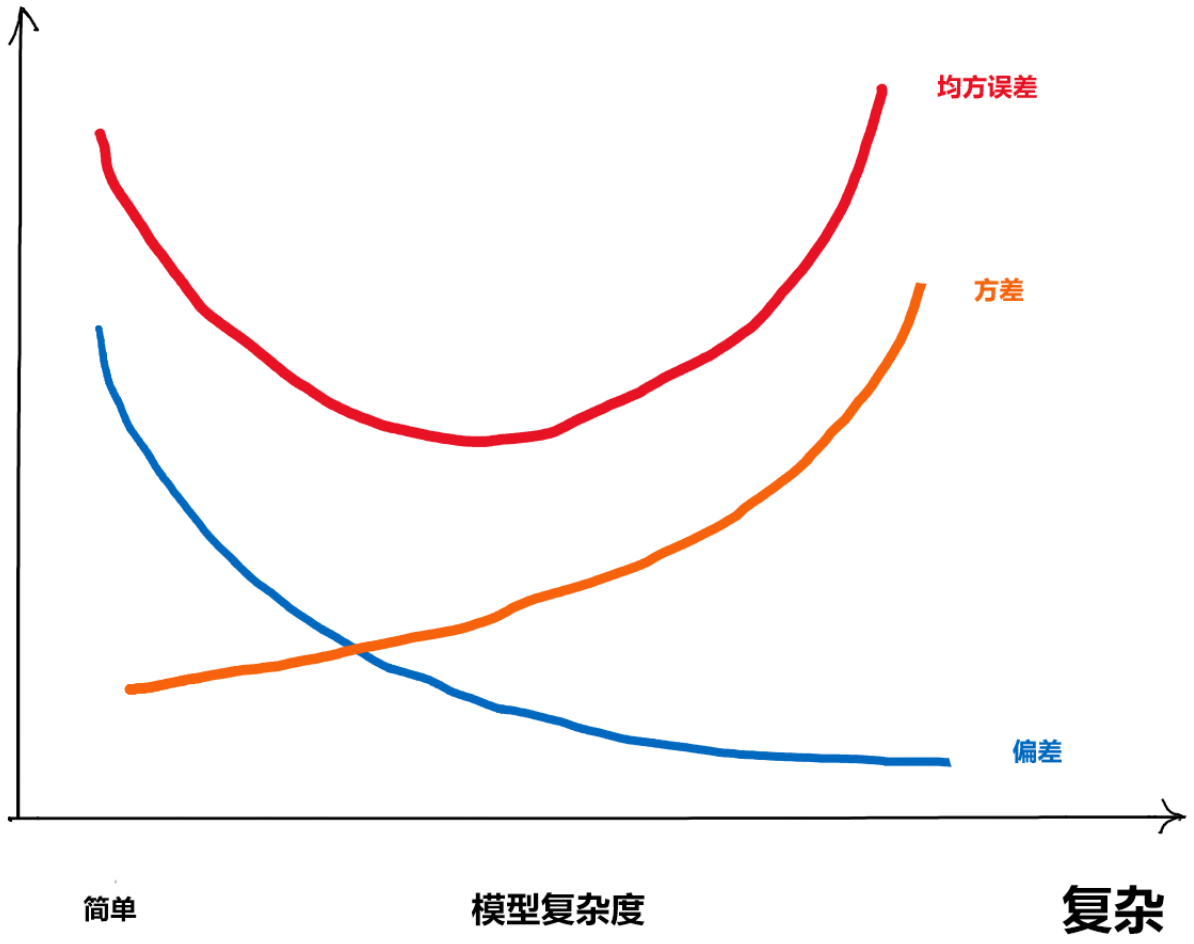
对回归问题，经常使用均方误差 $E|Ey-\hat{y}|^2$ 来衡量精度。对分类问题，经常使用分类准确率等来衡量精度。易见 $E|Ey-\hat{y}|^2=\mathrm{Var}(\hat{y})+(E\hat{y}-Ey)^2$，所以均方误差可以分解为

$$\text{均方误差}=\text{方差}+\text{偏差}^2,$$

训练的回归函数如果仅考虑对训练样本解释尽可能好，就会使得估计结果方差很大，在对检验样本进行计算时因方差大而导致很大的误差，所以选取的回归函数应该尽可能简单。

如果选取的回归函数过于简单而实际上自变量与因变量关系比较复杂，就会使得估计的回归函数偏差比较大，这样在对检验样本进行计算时也会有比较大的误差。

所以，在有监督学习时，回归函数的复杂程度是一个很关键的量，太复杂和太简单都可能导致差的结果，需要找到一个折衷的值。



复杂程度在线性回归中就是自变量个数，在一元曲线拟合中就是曲线的不光滑程度。在其它指标类似的情况下，简单的模型更稳定、可解释更好，所以统计学特别重视模型的简化。

38.1.3 交叉验证

即使是在从训练样本中修炼（估计）回归函数时，也需要适当地选择模型的复杂度。仅考虑对训练数据的拟合程度是不够的，这会造成过度拟合问题。

为了相对客观地度量模型的预报误差，假设训练样本有 n 个观测，可以留出第一个观测不用，用剩余的 $n-1$ 个观测建模，然后预测第一个观测的因变量值，得到一个误差；对每个观测都这样做，就可以得到 n 个误差。这样的方法叫做留一法。

更常用的是五折或十折交叉验证。假设训练集有 n 个观测，将其均分成 10 分，保留第一份不用，将其余九份合并在一起用来建模，然后预报第一份；对每一份都这样做，也可以得到 n 个误差，这叫做十折 (ten-fold) 交叉验证方法。

因为要预报的数据没有用来建模，交叉验证得到的误差估计更准确。

38.1.4 一般步骤

一个有监督的统计学习项目，大致上按如下步骤进行：

- 将数据拆分为训练集和测试集；
- 在训练集上比较不同的模型，可以用交叉验证方法比较；
- 使用训练集上找到的最优模型，对测试集进行测试；
- 在整个数据集上使用找到的最优模型，但预测效果的指标使用测试集上得到的预测效果指标。

38.2 Hitters 数据分析

考虑 ISLR 包的 Hitters 数据集。此数据集有 322 个运动员的 20 个变量的数据，其中的变量 Salary（工资）是我们关心的。变量包括:

```
names(Hitters)
```

```
##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"       "Walks"     "Years"     "CAtBat"
```

数据集的详细变量信息如下:

```
str(Hitters)
```

```
## 'data.frame':    322 obs. of  20 variables:
##  $ AtBat    : int  293 315 479 496 321 594 185 298 323 401 ...
##  $ Hits     : int  66 81 130 141 87 169 37 73 81 92 ...
##  $ HmRun    : int  1 7 18 20 10 4 1 0 6 17 ...
##  $ Runs     : int  30 24 66 65 39 74 23 24 26 49 ...
##  $ RBI      : int  29 38 72 78 42 51 8 24 32 66 ...
##  $ Walks    : int  14 39 76 37 30 35 21 7 8 65 ...
##  $ Years    : int  1 14 3 11 2 11 2 3 2 13 ...
##  $ CAtBat   : int  293 3449 1624 5628 396 4408 214 509 341 5206 ...
##  $ CHits    : int  66 835 457 1575 101 1133 42 108 86 1332 ...
##  $ CHmRun   : int  1 69 63 225 12 19 1 0 6 253 ...
##  $ CRuns    : int  30 321 224 828 48 501 30 41 32 784 ...
##  $ CRBI     : int  29 414 266 838 46 336 9 37 34 890 ...
##  $ CWalks   : int  14 375 263 354 33 194 24 12 8 866 ...
##  $ League   : Factor w/ 2 levels "A","N": 1 2 1 2 2 1 2 1 2 1 ...
##  $ Division : Factor w/ 2 levels "E","W": 1 2 2 1 1 2 1 2 2 1 ...
##  $ PutOuts  : int  446 632 880 200 805 282 76 121 143 0 ...
##  $ Assists  : int  33 43 82 11 40 421 127 283 290 0 ...
```

```
##  $ Errors   : int  20 10 14 3 4 25 7 9 19 0 ...
##  $ Salary   : num  NA 475 480 500 91.5 750 70 100 75 1100 ...
##  $ NewLeague: Factor w/ 2 levels "A","N": 1 2 1 2 2 1 1 1 2 1 ...
```

希望以 Salary 为因变量，查看其缺失值个数:

```
sum( is.na(Hitters$Salary) )
```

```
## [1] 59
```

为简单起见，去掉有缺失值的观测：

```
d <- na.omit(Hitters); dim(d)
```

```
## [1] 263  20
```

38.2.1 回归自变量选择

38.2.1.1 最优子集选择

用 leaps 包的 `regsubsets()` 函数计算最优子集回归，办法是对某个试验性的子集自变量个数 $\hat{p}$ 值，都找到 $\hat{p}$ 固定情况下残差平方和最小的变量子集，这样只要在这些不同 $\hat{p}$ 的最优子集中挑选就可以了。挑选可以用 AIC、BIC 等方法。

可以先进行一个包含所有自变量的全集回归：

```
regfit.full <- regsubsets(Salary ~ ., data=d, nvmax=19)
reg.summary <- summary(regfit.full)
reg.summary
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = d, nvmax = 19)
## 19 Variables  (and intercept)
##            Forced in Forced out
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
## Years          FALSE      FALSE
```

```
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 19
## Selection Algorithm: exhaustive
##           AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI CWalks LeagueN Divi
## 1  ( 1 )  " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "     " "
## 2  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "     " "
## 3  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "     " "
## 4  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "     "*"
## 5  ( 1 )  "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "     "*"
## 6  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*"  " "    " "     "*"
## 7  ( 1 )  " "   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "   " "  " "    " "     "*"
## 8  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"   " "  "*"    " "     "*"
## 9  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"  "*"    " "     "*"
## 10  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"  "*"    " "     "*"
## 11  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"  "*"    "*"     "*"
## 12  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"  "*"    "*"     "*"
## 13  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"  "*"    "*"     "*"
## 14  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"  "*"    "*"     "*"
## 15  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    "*"   " "    "*"   "*"  "*"    "*"     "*"
## 16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*"  "*"    "*"     "*"
## 17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*"  "*"    "*"     "*"
## 18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   " "    "*"   "*"  "*"    "*"     "*"
## 19  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"   "*"  "*"    "*"     "*"
```

这里用 `nvmax=` 指定了允许所有的自变量都参加，缺省行为是限制最多个数的。上述结果表格中每一行给出了固定 $\hat{p}$ 条件下的最优子集。

试比较这些最优模型的 BIC 值:

```
reg.summary$bic
```

```
##  [1]  -90.84637 -128.92622 -135.62693 -141.80892 -144.07143 -147.91690 -145.25594 -147.61525 -145.
```

```
plot(reg.summary$bic)
```

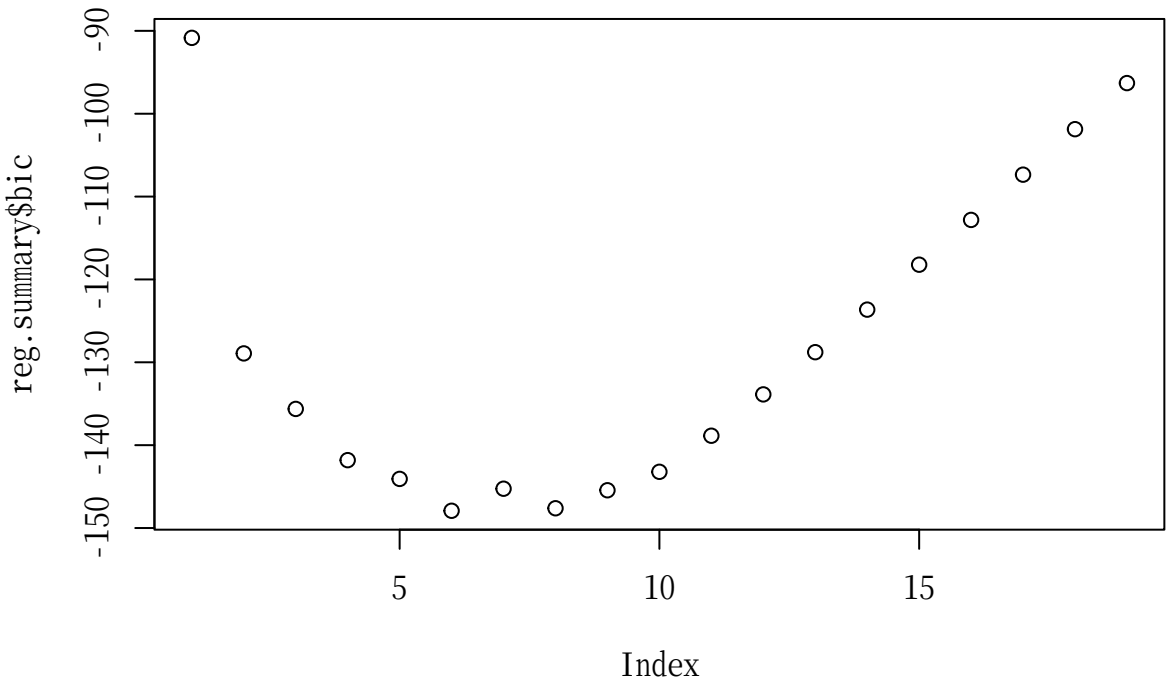


图 38.1: Hitters 数据最优子集回归 BIC

其中 $\hat{p} = 6, 8$ 的值相近，都很低，取 $\hat{p} = 6$。用 `coef()` 加 `id=6` 指定第六种子集：

```
coef(regfit.full, id=6)
```

```
##  (Intercept)        AtBat         Hits        Walks         CRBI    DivisionW      PutOuts
##   91.5117981   -1.8685892    7.6043976    3.6976468    0.6430169 -122.9515338    0.2643076
```

这种方法实现了选取 BIC 最小的自变量子集。

38.2.1.2 逐步回归方法

在用 `lm()` 做了全集回归后，把全集回归结果输入到 `step()` 函数中可以执行逐步回归。如:

```r
lm.full <- lm(Salary ~ ., data=d)
print(summary(lm.full))
```

```
##
## Call:
## lm(formula = Salary ~ ., data = d)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -907.62 -178.35  -31.11  139.09 1877.04
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  163.10359   90.77854   1.797 0.073622 .
## AtBat         -1.97987    0.63398  -3.123 0.002008 **
## Hits           7.50077    2.37753   3.155 0.001808 **
## HmRun          4.33088    6.20145   0.698 0.485616
## Runs          -2.37621    2.98076  -0.797 0.426122
## RBI           -1.04496    2.60088  -0.402 0.688204
## Walks          6.23129    1.82850   3.408 0.000766 ***
## Years         -3.48905   12.41219  -0.281 0.778874
## CAtBat        -0.17134    0.13524  -1.267 0.206380
## CHits          0.13399    0.67455   0.199 0.842713
## CHmRun        -0.17286    1.61724  -0.107 0.914967
## CRuns          1.45430    0.75046   1.938 0.053795 .
## CRBI           0.80771    0.69262   1.166 0.244691
## CWalks        -0.81157    0.32808  -2.474 0.014057 *
## LeagueN       62.59942   79.26140   0.790 0.430424
## DivisionW   -116.84925   40.36695  -2.895 0.004141 **
## PutOuts        0.28189    0.07744   3.640 0.000333 ***
## Assists        0.37107    0.22120   1.678 0.094723 .
## Errors        -3.36076    4.39163  -0.765 0.444857
## NewLeagueN   -24.76233   79.00263  -0.313 0.754218
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 315.6 on 243 degrees of freedom
## Multiple R-squared:  0.5461, Adjusted R-squared:  0.5106
## F-statistic: 15.39 on 19 and 243 DF,  p-value: < 2.2e-16
```

```r
lm.step <- step(lm.full)
```

```
## Start:  AIC=3046.02
## Salary ~ AtBat + Hits + HmRun + Runs + RBI + Walks + Years +
##     CAtBat + CHits + CHmRun + CRuns + CRBI + CWalks + League +
##     Division + PutOuts + Assists + Errors + NewLeague
##
##             Df Sum of Sq      RSS    AIC
## - CHmRun     1      1138 24201837 3044.0
## - CHits      1      3930 24204629 3044.1
## - Years      1      7869 24208569 3044.1
## - NewLeague  1      9784 24210484 3044.1
## - RBI        1     16076 24216776 3044.2
## - HmRun      1     48572 24249272 3044.6
## - Errors     1     58324 24259023 3044.7
## - League     1     62121 24262821 3044.7
## - Runs       1     63291 24263990 3044.7
## - CRBI       1    135439 24336138 3045.5
## - CAtBat     1    159864 24360564 3045.8
## <none>                   24200700 3046.0
## - Assists    1    280263 24480963 3047.1
## - CRuns      1    374007 24574707 3048.1
## - CWalks     1    609408 24810108 3050.6
## - Division   1    834491 25035190 3052.9
## - AtBat      1    971288 25171987 3054.4
## - Hits       1    991242 25191941 3054.6
## - Walks      1   1156606 25357305 3056.3
## - PutOuts    1   1319628 25520328 3058.0
##
## Step:  AIC=3044.03
## Salary ~ AtBat + Hits + HmRun + Runs + RBI + Walks + Years +
##     CAtBat + CHits + CRuns + CRBI + CWalks + League + Division +
##     PutOuts + Assists + Errors + NewLeague
##
##             Df Sum of Sq      RSS    AIC
## - Years      1      7609 24209447 3042.1
## - NewLeague  1     10268 24212106 3042.2
## - CHits      1     14003 24215840 3042.2
## - RBI        1     14955 24216793 3042.2
```

```
## - HmRun      1     52777 24254614 3042.6
## - Errors     1     59530 24261367 3042.7
## - League     1     63407 24265244 3042.7
## - Runs       1     64860 24266698 3042.7
## - CAtBat     1    174992 24376830 3043.9
## <none>                   24201837 3044.0
## - Assists    1    285766 24487603 3045.1
## - CRuns      1    611358 24813196 3048.6
## - CWalks     1    645627 24847464 3049.0
## - Division   1    834637 25036474 3050.9
## - CRBI       1    864220 25066057 3051.3
## - AtBat      1    970861 25172699 3052.4
## - Hits       1   1025981 25227819 3052.9
## - Walks      1   1167378 25369216 3054.4
## - PutOuts    1   1325273 25527110 3056.1
##
## Step:  AIC=3042.12
## Salary ~ AtBat + Hits + HmRun + Runs + RBI + Walks + CAtBat +
##     CHits + CRuns + CRBI + CWalks + League + Division + PutOuts +
##     Assists + Errors + NewLeague
##
##             Df Sum of Sq      RSS    AIC
## - NewLeague  1      9931 24219377 3040.2
## - RBI        1     15989 24225436 3040.3
## - CHits      1     18291 24227738 3040.3
## - HmRun      1     54144 24263591 3040.7
## - Errors     1     57312 24266759 3040.7
## - Runs       1     63172 24272619 3040.8
## - League     1     65732 24275178 3040.8
## <none>                   24209447 3042.1
## - CAtBat     1    266205 24475652 3043.0
## - Assists    1    293479 24502926 3043.3
## - CRuns      1    646350 24855797 3047.1
## - CWalks     1    649269 24858716 3047.1
## - Division   1    827511 25036958 3049.0
## - CRBI       1    872121 25081568 3049.4
## - AtBat      1    968713 25178160 3050.4
## - Hits       1   1018379 25227825 3050.9
## - Walks      1   1164536 25373983 3052.5
## - PutOuts    1   1334525 25543972 3054.2
```

```
##
## Step:  AIC=3040.22
## Salary ~ AtBat + Hits + HmRun + Runs + RBI + Walks + CAtBat +
##     CHits + CRuns + CRBI + CWalks + League + Division + PutOuts +
##     Assists + Errors
##
##            Df Sum of Sq      RSS    AIC
## - RBI       1     15800 24235177 3038.4
## - CHits     1     15859 24235237 3038.4
## - Errors    1     54505 24273883 3038.8
## - HmRun     1     54938 24274316 3038.8
## - Runs      1     62294 24281671 3038.9
## - League    1    107479 24326856 3039.4
## <none>                  24219377 3040.2
## - CAtBat    1    261336 24480713 3041.1
## - Assists   1    295536 24514914 3041.4
## - CWalks    1    648860 24868237 3045.2
## - CRuns     1    661449 24880826 3045.3
## - Division  1    824672 25044049 3047.0
## - CRBI      1    880429 25099806 3047.6
## - AtBat     1    999057 25218434 3048.9
## - Hits      1   1034463 25253840 3049.2
## - Walks     1   1157205 25376583 3050.5
## - PutOuts   1   1335173 25554550 3052.3
##
## Step:  AIC=3038.4
## Salary ~ AtBat + Hits + HmRun + Runs + Walks + CAtBat + CHits +
##     CRuns + CRBI + CWalks + League + Division + PutOuts + Assists +
##     Errors
##
##            Df Sum of Sq      RSS    AIC
## - CHits     1     13483 24248660 3036.5
## - HmRun     1     44586 24279763 3036.9
## - Runs      1     54057 24289234 3037.0
## - Errors    1     57656 24292833 3037.0
## - League    1    108644 24343821 3037.6
## <none>                  24235177 3038.4
## - CAtBat    1    252756 24487934 3039.1
## - Assists   1    294674 24529851 3039.6
## - CWalks    1    639690 24874868 3043.2
```

```
## - CRuns     1    693535 24928712 3043.8
## - Division  1    808984 25044161 3045.0
## - CRBI      1    893830 25129008 3045.9
## - Hits      1   1034884 25270061 3047.4
## - AtBat     1   1042798 25277975 3047.5
## - Walks     1   1145013 25380191 3048.5
## - PutOuts   1   1340713 25575890 3050.6
##
## Step:  AIC=3036.54
## Salary ~ AtBat + Hits + HmRun + Runs + Walks + CAtBat + CRuns +
##     CRBI + CWalks + League + Division + PutOuts + Assists + Errors
##
##            Df Sum of Sq      RSS    AIC
## - HmRun     1     40487 24289148 3035.0
## - Errors    1     51930 24300590 3035.1
## - Runs      1     79343 24328003 3035.4
## - League    1    114742 24363402 3035.8
## <none>                  24248660 3036.5
## - Assists   1    283442 24532103 3037.6
## - CAtBat    1    613356 24862016 3041.1
## - Division  1    801474 25050134 3043.1
## - CRBI      1    903248 25151908 3044.2
## - CWalks    1   1011953 25260613 3045.3
## - Walks     1   1246164 25494824 3047.7
## - AtBat     1   1339620 25588280 3048.7
## - CRuns     1   1390808 25639469 3049.2
## - PutOuts   1   1406023 25654684 3049.4
## - Hits      1   1607990 25856650 3051.4
##
## Step:  AIC=3034.98
## Salary ~ AtBat + Hits + Runs + Walks + CAtBat + CRuns + CRBI +
##     CWalks + League + Division + PutOuts + Assists + Errors
##
##            Df Sum of Sq      RSS    AIC
## - Errors    1     44085 24333232 3033.5
## - Runs      1     49068 24338215 3033.5
## - League    1    103837 24392985 3034.1
## <none>                  24289148 3035.0
## - Assists   1    247002 24536150 3035.6
## - CAtBat    1    652746 24941894 3040.0
```

```
## - Division  1    795643 25084791 3041.5
## - CWalks    1    982896 25272044 3043.4
## - Walks     1   1205823 25494971 3045.7
## - AtBat     1   1300972 25590120 3046.7
## - CRuns     1   1351200 25640348 3047.2
## - CRBI      1   1353507 25642655 3047.2
## - PutOuts   1   1429006 25718154 3048.0
## - Hits      1   1574140 25863288 3049.5
##
## Step:  AIC=3033.46
## Salary ~ AtBat + Hits + Runs + Walks + CAtBat + CRuns + CRBI +
##     CWalks + League + Division + PutOuts + Assists
##
##            Df Sum of Sq      RSS    AIC
## - Runs      1     54113 24387345 3032.0
## - League    1     91269 24424501 3032.4
## <none>                  24333232 3033.5
## - Assists   1    220010 24553242 3033.8
## - CAtBat    1    650513 24983746 3038.4
## - Division  1    799455 25132687 3040.0
## - CWalks    1    971260 25304493 3041.8
## - Walks     1   1239533 25572765 3044.5
## - CRBI      1   1331672 25664904 3045.5
## - CRuns     1   1361070 25694302 3045.8
## - AtBat     1   1378592 25711824 3045.9
## - PutOuts   1   1391660 25724892 3046.1
## - Hits      1   1649291 25982523 3048.7
##
## Step:  AIC=3032.04
## Salary ~ AtBat + Hits + Walks + CAtBat + CRuns + CRBI + CWalks +
##     League + Division + PutOuts + Assists
##
##            Df Sum of Sq      RSS    AIC
## - League    1    113056 24500402 3031.3
## <none>                  24387345 3032.0
## - Assists   1    280689 24668034 3033.1
## - CAtBat    1    596622 24983967 3036.4
## - Division  1    780369 25167714 3038.3
## - CWalks    1    946687 25334032 3040.1
## - Walks     1   1212997 25600342 3042.8
```

```
## - CRuns     1   1334397 25721742 3044.1
## - CRBI      1   1361339 25748684 3044.3
## - PutOuts   1   1455210 25842555 3045.3
## - AtBat     1   1522760 25910105 3046.0
## - Hits      1   1718870 26106215 3047.9
##
## Step:  AIC=3031.26
## Salary ~ AtBat + Hits + Walks + CAtBat + CRuns + CRBI + CWalks +
##     Division + PutOuts + Assists
##
##            Df Sum of Sq      RSS    AIC
## <none>                  24500402 3031.3
## - Assists   1    313650 24814051 3032.6
## - CAtBat    1    534156 25034558 3034.9
## - Division  1    798473 25298875 3037.7
## - CWalks    1    965875 25466276 3039.4
## - CRuns     1   1265082 25765484 3042.5
## - Walks     1   1290168 25790569 3042.8
## - CRBI      1   1326770 25827172 3043.1
## - PutOuts   1   1551523 26051925 3045.4
## - AtBat     1   1589780 26090181 3045.8
## - Hits      1   1716068 26216469 3047.1
```

```
print(lm.step)
```

```
##
## Call:
## lm(formula = Salary ~ AtBat + Hits + Walks + CAtBat + CRuns +
##     CRBI + CWalks + Division + PutOuts + Assists, data = d)
##
## Coefficients:
## (Intercept)        AtBat         Hits        Walks       CAtBat        CRuns         CRBI       CW
##    162.5354      -2.1687       6.9180       5.7732      -0.1301       1.4082       0.7743      -0.
```

最后保留了 10 个自变量。

38.2.1.3 划分训练集与测试集

在整个数据集中随机选取一部分作为训练集，其余作为测试集。下面的程序把原始数据一分为二：

```
set.seed(1)
train <- sample(nrow(d), size = round(nrow(d)/2))
test <- -train
```

仅用训练集估计模型。为了在测试集上用模型进行预报并估计预测均方误差，需要自己写一个预测函数：

```
predict.regsubsets <- function(object, newdata, id, ...){
  form <- as.formula(object$call[[2]])
  mat <- model.matrix(form, newdata)
  coefi <- coef(object, id=id)
  xvars <- names(coefi)
  mat[, xvars] %*% coefi
}
```

然后，对每个子集大小，用最优子集在测试集上进行预报，计算均方误差：

```
regfit.best <- regsubsets( Salary ~ ., data=d[train,], nvmax=19 )
val.errors <- rep(as.numeric(NA), 19)
for(i in 1:19){
  #pred <- predict.regsubsets(regfit.best, newdata=d[test,], id=i)
  pred <- predict(regfit.best, newdata=d[test,], id=i)
  val.errors[i] <- mean( (d[test, 'Salary'] - pred)^2 )
}
print(val.errors)
```

```
##  [1] 188190.9 163306.2 152365.0 164857.0 152100.7 147120.0 148833.0 155546.5 167429.2 169949.1 173
```

```
best.id <- which.min(val.errors); best.id
```

```
## [1] 6
```

用测试集得到的最优子集大小为 6。模型子集和回归系数为:

```
coef(regfit.best, id=best.id)
```

```
##  (Intercept)        Walks       CAtBat        CHits       CHmRun    DivisionW      PutOuts
##  179.4442609    4.1205817   -0.5508342    2.1670021    2.3479409 -126.3067258    0.1840943
```

38.2.1.4 用 10 折交叉验证方法选择最优子集

下列程序对数据中每一行分配一个折号：

```
k <- 10
set.seed(1)
folds <- sample(1:k, nrow(d), replace=TRUE)
```

下面，对 10 折中每一折都分别当作测试集一次，得到不同子集大小的均方误差：

```
cv.errors <- matrix( as.numeric(NA), k, 19, dimnames=list(NULL, paste(1:19)) )
for(j in 1:k){ # 对
  best.fit <- regsubsets(Salary ~ ., data=d[folds != j,], nvmax=19)
  for(i in 1:19){
    pred <- predict( best.fit, d[folds==j,], id=i)
    cv.errors[j, i] <- mean( (d[folds==j, 'Salary'] - pred)^2 )
  }
}
head(cv.errors)
```

```
##              1         2         3         4         5         6         7         8         9
## [1,]  98623.24 115600.61 120884.31 113831.63 120728.51 122922.93 155507.25 137753.36 149198.01 153
## [2,] 155320.11 100425.87 168838.35 159729.47 145895.71 123555.25 119983.35  96609.16  99057.32  80
## [3,] 124151.77  68833.50  69392.29  77221.37  83802.82  70125.41  68997.77  64143.70  65813.14  65
## [4,] 232191.41 279001.29 294568.10 288765.81 276972.83 260121.22 276413.09 259923.88 270151.18 263
## [5,] 115397.35  96807.44 108421.66 104933.55  99561.69  86103.05  89345.61  87693.15  91631.88  88
## [6,] 103839.30  75652.50  69962.31  58291.91  65893.45  64215.56  65800.88  61413.45  60200.70  59
```

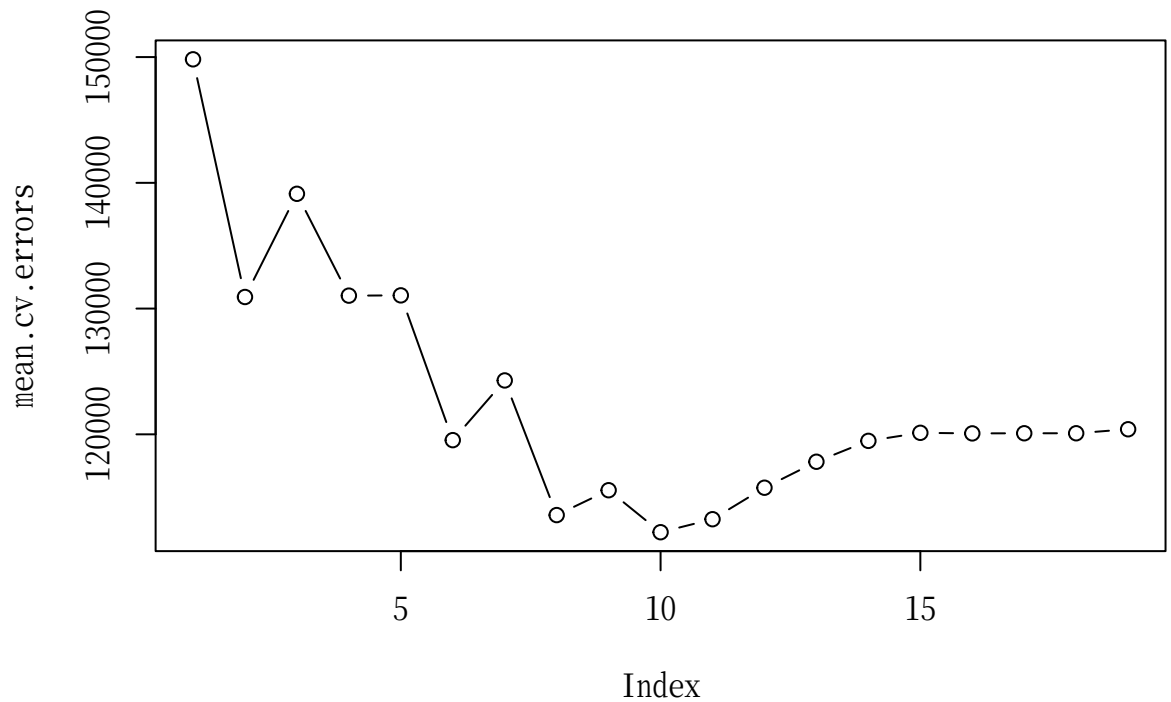
cv.errors 是一个 10×19 矩阵，每行对应一折作为测试集的情形，每列是一个子集大小，元素值是测试均方误差。

对每列的 10 个元素求平均，可以得到每个子集大小的平均均方误差:

```
mean.cv.errors <- apply(cv.errors, 2, mean)
mean.cv.errors
```

```
##        1        2        3        4        5        6        7        8        9       10       11
## 149821.1 130922.0 139127.0 131028.8 131050.2 119538.6 124286.1 113580.0 115556.5 112216.7 113251.2
```

```
best.id <- which.min(mean.cv.errors)
plot(mean.cv.errors, type='b')
```



这样找到的最优子集大小是 10。用这种方法找到最优子集大小后，可以对全数据集重新建模但是选择最优子集大小为 10:

```
reg.best <- regsubsets(Salary ~ ., data=d, nvmax=19)
coef(reg.best, id=best.id)
```

```
##  (Intercept)        AtBat         Hits        Walks       CAtBat        CRuns         CRBI        C
##  162.5354420   -2.1686501    6.9180175    5.7732246   -0.1300798    1.4082490    0.7743122   -0.83
```

事实上，划分训练集和验证集与交叉验证方法经常联合运用。取一个固定的较小规模的测试集，此测试集不用来作子集选择，对训练集用交叉验证方法选择最优子集，然后再测试集上验证。

38.2.2 岭回归

当自变量个数太多时，模型复杂度高，可能有过度拟合，模型不稳定。

一种方法是对较大的模型系数施加二次惩罚，把最小二乘问题变成带有二次惩罚项的惩罚最小二乘问题：

$$\min \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_{i1} - \cdots - \beta_p x_{ip}\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2.$$

这比通常最小二乘得到的回归系数绝对值变小，但是求解的稳定性增加了，避免了共线问题。

实际上, 与线性模型 $Y = X\beta + \varepsilon$ 的普通最小二乘解 $\hat{\beta} = (X^T X)^{-1} X^T Y$ 相比，岭回归问题的解为

$$\tilde{\beta} = (X^T X + sI)^{-1} X^T Y$$

其中 I 为单位阵，$s > 0$ 与 λ 有关。

λ 称为调节参数，λ 越大，相当于模型复杂度越低。适当选择 λ 可以在方差与偏差之间找到适当的折衷，从而减小预测误差。

由于量纲问题，在不同自变量不可比时，数据集应该进行标准化。

用 R 的 glmnet 包计算岭回归。用 `glmnet()` 函数，指定参数 `alpha=0` 时执行的是岭回归。用参数 `lambda=` 指定一个调节参数网格，岭回归将在这些调节参数上计算。用 `coef()` 从回归结果中取得不同调节参数对应的回归系数估计，结果是一个矩阵，每列对应于一个调节参数。

仍采用上面去掉了缺失值的 Hitters 数据集结果 d。

如下程序把回归的设计阵与因变量提取出来：

```
x <- model.matrix(Salary ~ ., d)[,-1]
y <- d$Salary
```

岭回归涉及到调节参数 λ 的选择，为了绘图，先选择 λ 的一个网格：

```
grid <- 10^seq(10, -2, length=100)
```

用所有数据针对这样的调节参数网格计算岭回归结果，注意 `glmnet()` 函数允许调节参数 λ 输入多个值：

```
ridge.mod <- glmnet(x, y, alpha=0, lambda=grid)
dim(coef(ridge.mod))
```

```
## [1]  20 100
```

`glmnet()` 函数默认对数据进行标准化。
`coef()` 的结果是一个矩阵，每列对应一个调节参数值。

38.2.2.1 划分训练集与测试集

如下程序把数据分为一半训练、一半测试：

```
set.seed(1)
train <- sample(nrow(x), size = nrow(x)/2)
test <- (-train)
y.test <- y[test]
```

仅用测试集建立岭回归：

```
ridge.mod <- glmnet(x[train,], y[train], alpha=0, lambda=grid, thresh=1E-12)
```

用建立的模型对测试集进行预测，并计算调节参数等于 4 时的均方误差：

```
ridge.pred <- predict( ridge.mod, s=4, newx=x[test,] )
mean( (ridge.pred - y.test)^2 )
```

```
## [1] 142199.2
```

如果用因变量平均值作预测，这是最差的预测：

```
mean( (mean(y[train]) - y.test)^2 )
```

```
## [1] 224669.9
```

$\lambda = 4$ 的结果要好得多。事实上，取 λ 接近正无穷时模型就相当于用因变量平均值预测。取 $\lambda = 0$ 就相当于普通最小二乘回归（但是 `glmnet()` 是对输入数据要做标准化的）。

38.2.2.2 用 10 折交叉验证选取调节参数

仍使用训练集，但训练集再进行交叉验证。`cv.glmnet()` 函数可以执行交叉验证。

```
set.seed(1)
cv.out <- cv.glmnet(x[train,], y[train], alpha=0)
plot(cv.out)
```

```
bestlam <- cv.out$lambda.min
```

这样获得了最优调节参数 $\lambda = 326.0827865$。用最优调节参数对测试集作预测，得到预测均方误差：

```
ridge.pred <- predict(ridge.mod, s=bestlam, newx=x[test,])
mean( (ridge.pred - y.test)^2 )
```

```
## [1] 139856.6
```

结果比 $\lambda = 4$ 略有改进。

最后，用选取的最优调节系数对全数据集建模，得到相应的岭回归系数估计：

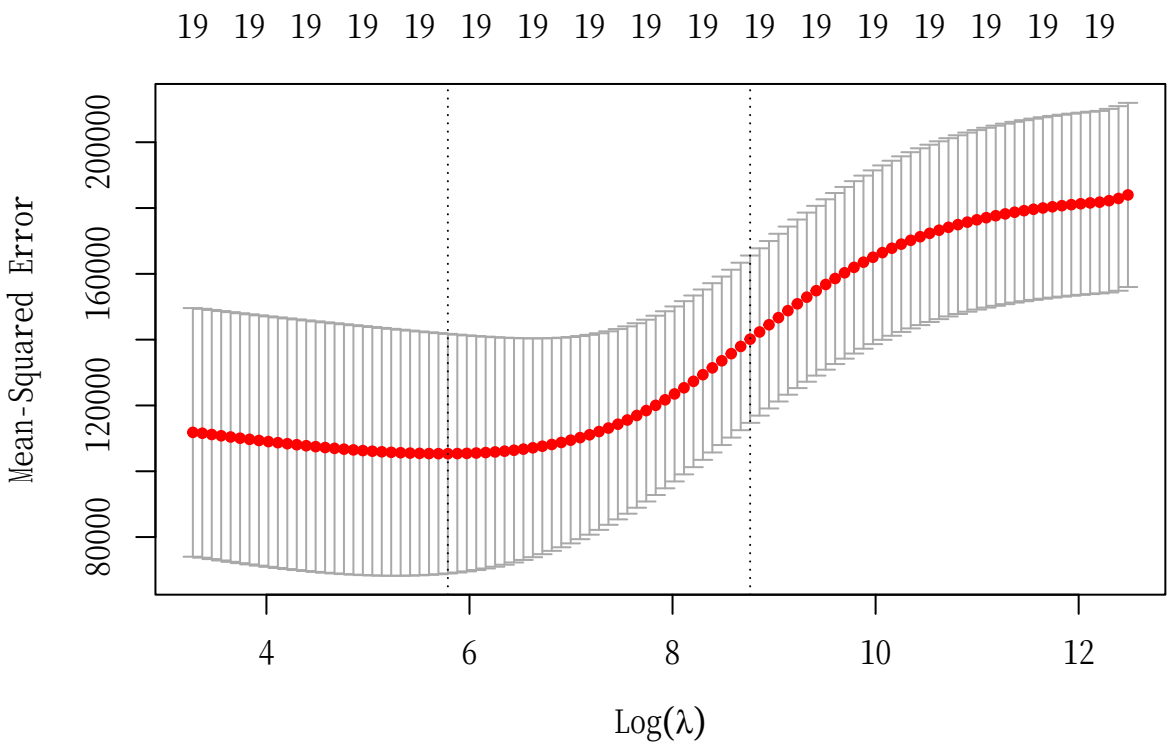


图 38.2: Hitters 数据岭回归参数选择

```
out <- glmnet(x, y, alpha=0)
predict(out, type='coefficients', s=bestlam)[1:20,]
```

```
## (Intercept)        AtBat         Hits        HmRun         Runs          RBI        Walks
## 15.44383120   0.07715547   0.85911582   0.60103106   1.06369007   0.87936105   1.62444617   1.352
```

38.2.3 Lasso 回归

另一种对回归系数的惩罚是 L_1 惩罚：

$$\min \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_{i1} - \cdots - \beta_p x_{ip}\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|. \tag{38.1}$$

奇妙地是，适当选择调节参数 λ，可以使得部分回归系数变成零，达到了即减小回归系数的绝对值又挑选重要变量子集的效果。

事实上，(38.1)等价于约束最小值问题

$$\min \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_{i1} - \cdots - \beta_p x_{ip}\right)^2 \quad \text{s.t.}$$
$$\sum_{j=1}^{p} |\beta_j| \le s$$

其中 s 与 λ 一一对应。这样的约束区域是带有顶点的凸集，而目标函数是二次函数，最小值点经常在约束区域顶点达到，这些顶点是某些坐标等于零的点。见图38.3。

```
knitr::include_graphics("figs/lasso-min.png")
```

对于每个调节参数 λ，都应该解出(38.1)的相应解，记为 $\widehat{\beta}(\lambda)$。幸运的是，不需要对每个 λ 去解最小值问题(38.1)，存在巧妙的算法使得问题的计算量与求解一次最小二乘相仿。

通常选取 λ 的格子点，计算相应的惩罚回归系数。用交叉验证方法估计预测的均方误差。选取使得交叉验证均方误差最小的调节参数（一般 R 函数中已经作为选项）。

用 R 的 glmnet 包计算 lasso。用 `glmnet()` 函数，指定参数 `alpha=1` 时执行的是 lasso。用参数 `lambda=` 指定一个调节参数网格，lasso 将输出这些调节参数对应的结果。对回归结果使用 `plot()` 函数可以画出调节参数变化时系数估计的变化情况。

仍使用 gmlnet 包的 `glmnet()` 函数计算 Lasso 回归，指定一个调节参数网格（沿用前面的网格）：

```
lasso.mod <- glmnet(x[train,], y[train], alpha=1, lambda=grid)
plot(lasso.mod)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm): collapsing to unique 'x' v
```

对 lasso 结果使用 `plot()` 函数可以绘制延调节参数网格变化的各回归系数估计，横坐标不是调节参数而是调节参数对应的系数绝对值和，可以看出随着系数绝对值和增大，实际是调节参数变小，更多地自变量进入模型。

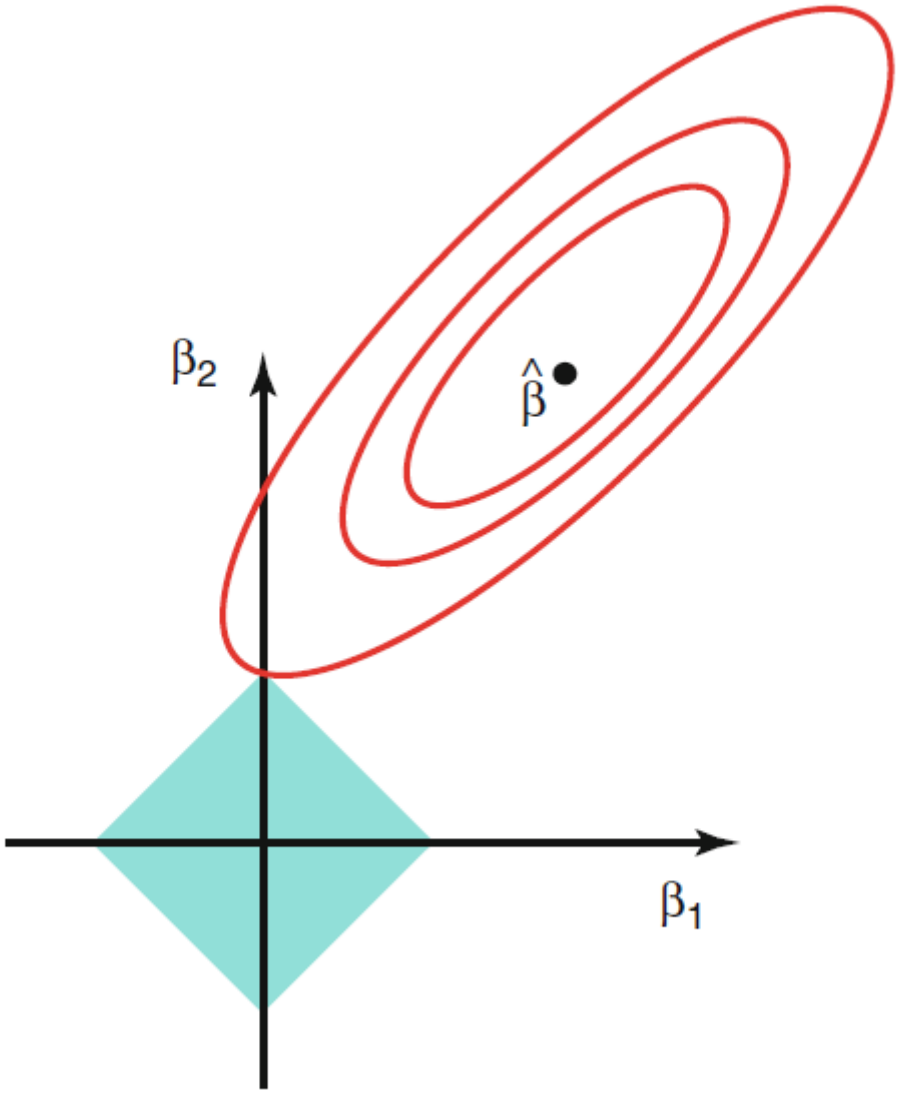


图 38.3: Lasso 约束优化问题图示

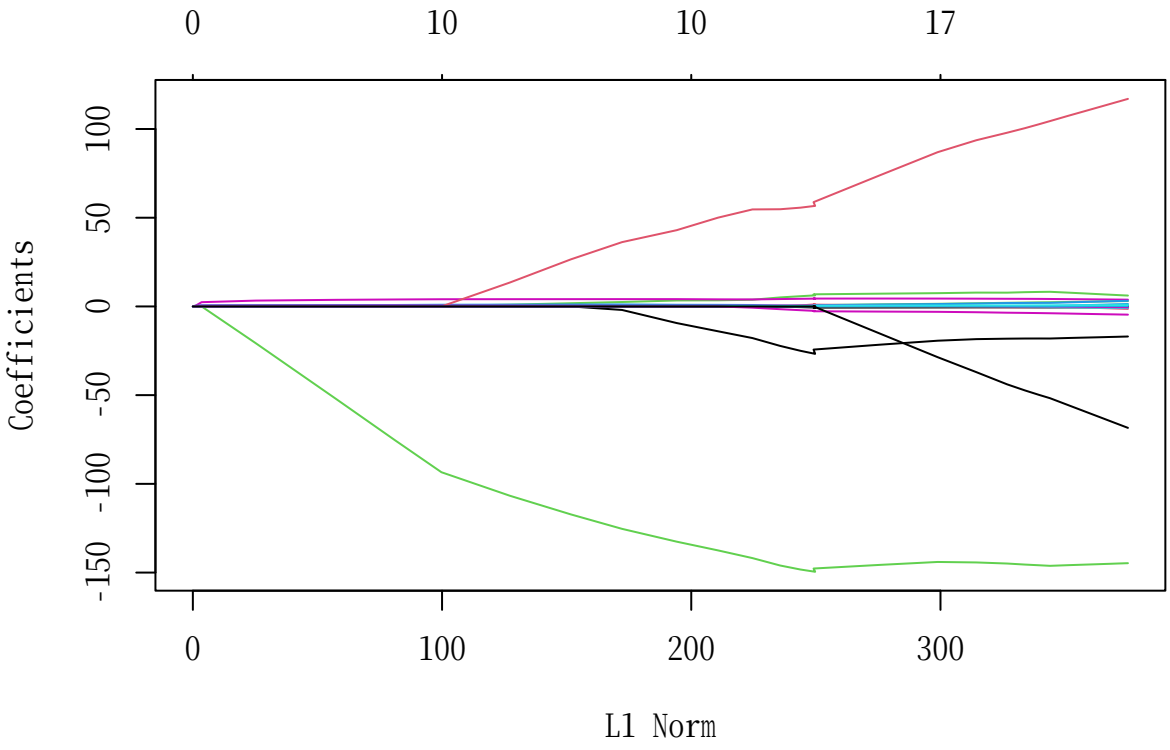
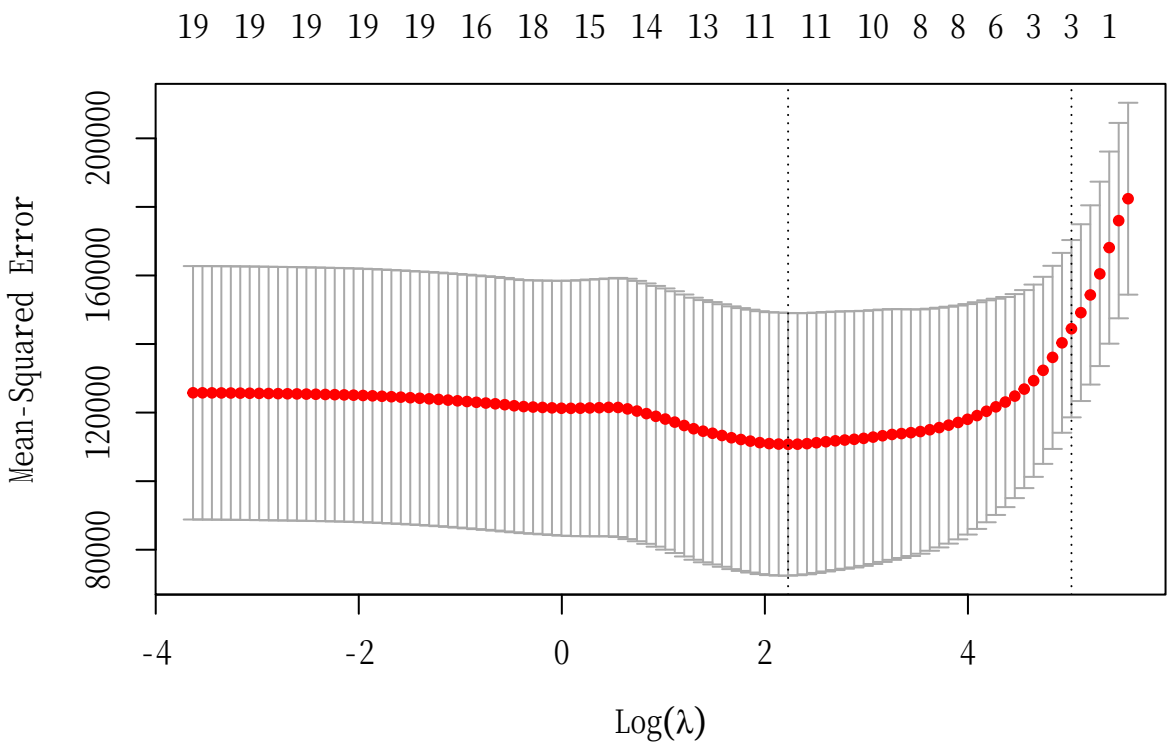


图 38.4: Hitters 数据 lasso 轨迹

38.2.3.1 用交叉验证估计调节参数

按照前面划分的训练集与测试集，仅使用训练集数据做交叉验证估计最优调节参数：

```
set.seed(1)
cv.out <- cv.glmnet(x[train,], y[train], alpha=1)
plot(cv.out)
```




```
bestlam <- cv.out$lambda.min; bestlam
```

```
## [1] 9.286955
```

得到调节参数估计后，对测试集计算预测均方误差：

```
lasso.pred <- predict(lasso.mod, s=bestlam, newx=x[test,])
mean( (lasso.pred - y.test)^2 )
```

```
## [1] 143673.6
```

这个效果比岭回归效果略差。

为了充分利用数据，使用前面获得的最优调节参数，对全数据集建模：

```
out <- glmnet(x, y, alpha=1, lambda=grid)
lasso.coef <- predict(out, type='coefficients', s=bestlam)[1:20,]; lasso.coef
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs           RBI         Walks
##    1.27479059   -0.05497143    2.18034583    0.00000000    0.00000000    0.00000000    2.29192406
```

```
lasso.coef[lasso.coef != 0]
```

```
##   (Intercept)         AtBat          Hits         Walks         Years        CHmRun         CRuns
##    1.27479059   -0.05497143    2.18034583    2.29192406   -0.33806109    0.02825013    0.21628385
```

选择的自变量子集有 11 个自变量。

38.2.4 树回归的简单演示

决策树方法按不同自变量的不同值，分层地把训练集分组。每层使用一个变量，所以这样的分组构成一个二叉树表示。为了预测一个观测的类归属，找到它所属的组，用组的类归属或大多数观测的类归属进行预测。这样的方法称为**决策树** (decision tree)。决策树方法既可以用于判别问题，也可以用于回归问题，称为回归树。

决策树的好处是容易解释，在自变量为分类变量时没有额外困难。但预测准确率可能比其它有监督学习方法差。

改进方法包括装袋法 (bagging)、随机森林 (random forests)、提升法 (boosting)。这些改进方法都是把许多棵树合并在一起，通常能改善准确率但是可解释性变差。

对 Hitters 数据，用 Years 和 Hits 作因变量预测 log(Salaray)。

仅取 Hitters 数据集的 Salary, Years, Hits 三个变量，并仅保留完全观测：

```
d <- na.omit(Hitters[,c('Salary', 'Years', 'Hits')])
print(str(d))
```

```
## 'data.frame':    263 obs. of  3 variables:
##  $ Salary: num  475 480 500 91.5 750 ...
##  $ Years : int  14 3 11 2 11 2 3 2 13 10 ...
##  $ Hits  : int  81 130 141 87 169 37 73 81 92 159 ...
##  - attr(*, "na.action")= 'omit' Named int [1:59] 1 16 19 23 31 33 37 39 40 42 ...
##   ..- attr(*, "names")= chr [1:59] "-Andy Allanson" "-Billy Beane" "-Bruce Bochte" "-Bob Boone" ..
## NULL
```

建立完整的树:

```
tr1 <- tree(log(Salary) ~ Years + Hits, data=d)
```

剪枝为只有 3 个叶结点:

```
tr1b <- prune.tree(tr1, best=3)
```

显示树:

```
print(tr1b)
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 263 207.20 5.927
##   2) Years < 4.5 90  42.35 5.107 *
##   3) Years > 4.5 173  72.71 6.354
##     6) Hits < 117.5 90  28.09 5.998 *
##     7) Hits > 117.5 83  20.88 6.740 *
```

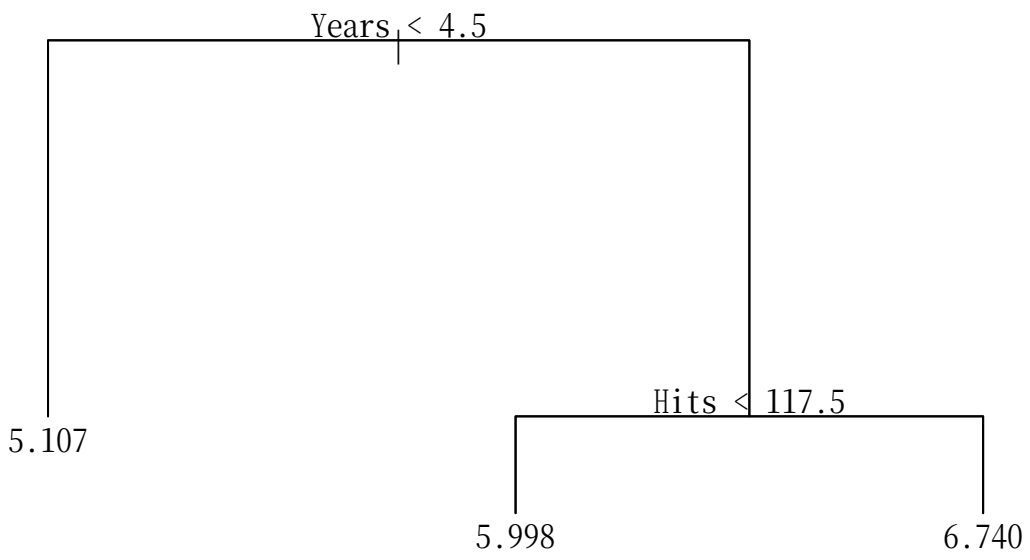
显示概括:

```
print(summary(tr1b))
```

```
##
## Regression tree:
## snip.tree(tree = tr1, nodes = c(6L, 2L))
## Number of terminal nodes:  3
## Residual mean deviance:  0.3513 = 91.33 / 260
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.24000 -0.39580 -0.03162  0.00000  0.33380  2.55600
```

做树图:

```
plot(tr1b); text(tr1b, pretty=0)
```



38.2.5 树回归

把数据随机地分成一半训练集，一半测试集：

```
d <- na.omit(Hitters)
set.seed(1)
train <- sample(nrow(d), size=round(nrow(d)/2))
test <- (-train)
```

对训练集，建立未剪枝的树：

```
tr1 <- tree(log(Salary) ~ ., data=d, subset=train)
plot(tr1); text(tr1, pretty=0)
```

对训练集上的未剪枝树用交叉验证方法寻找最优大小：

```
cv1 <- cv.tree(tr1)
print(cv1)
```

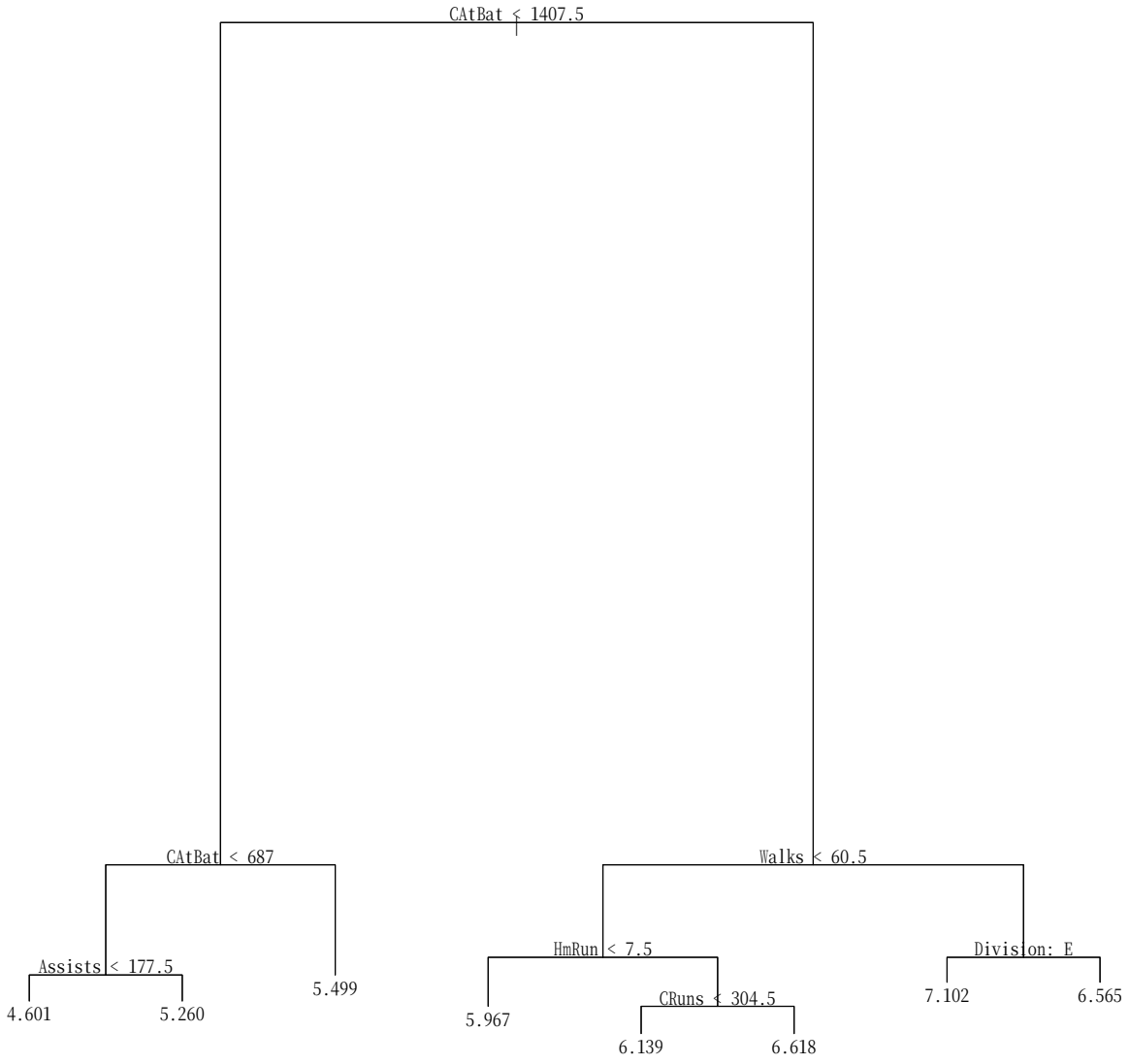
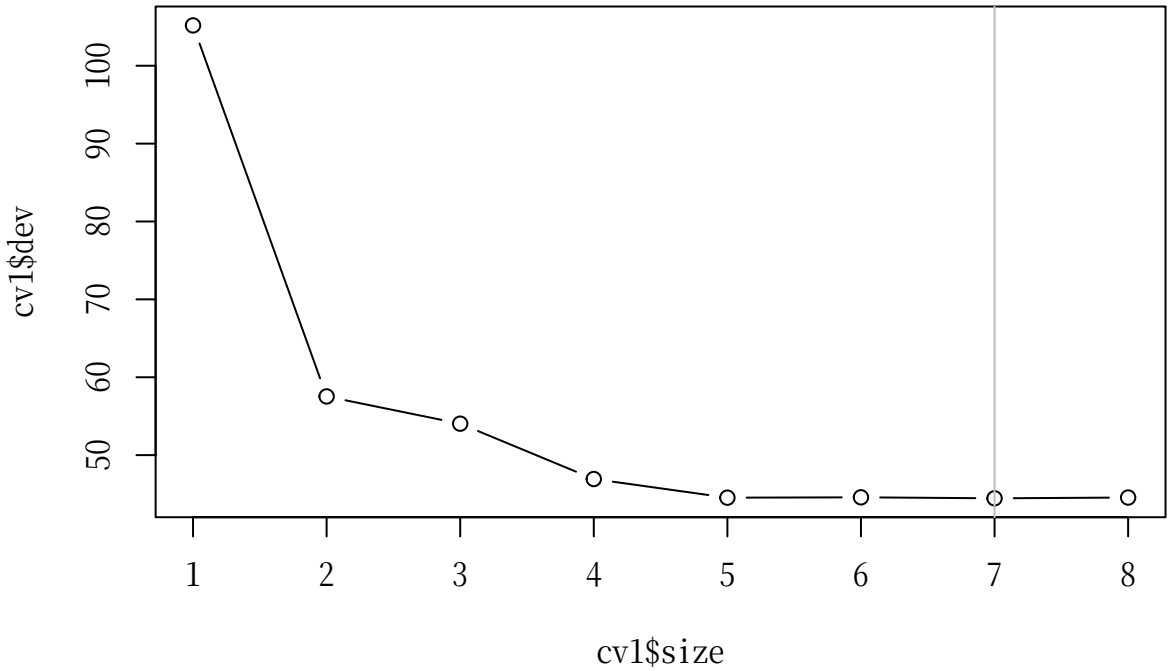


图 38.5: Hitters 数据训练集未剪枝树

```
## $size
## [1] 8 7 6 5 4 3 2 1
##
## $dev
## [1]  44.55223  44.45312  44.57906  44.53469  46.93001  54.03823  57.53660 105.17743
##
## $k
## [1]      -Inf  1.679266  1.750440  1.836204  3.300858  6.230249  7.420672 56.727362
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
plot(cv1$size, cv1$dev, type='b')
best.size <- cv1$size[which.min(cv1$dev)[1]]
abline(v=best.size, col='gray')
```



最优大小为 7。获得训练集上构造的树剪枝后的结果：

```
best.size <- 4
tr1b <- prune.tree(tr1, best=best.size)
```

在测试集上计算预测均方误差:

```
pred.test <- predict(tr1b, newdata=d[test,])
test.mse <- mean( (d[test, 'Salary'] - exp(pred.test))^2 )
test.mse
```

```
## [1] 128224.1
```

如果用训练集的因变量平均值估计测试集的因变量值，均方误差为:

```
worst.mse <- mean( (d[test, 'Salary'] - mean(d[train, 'Salary']))^2 )
worst.mse
```

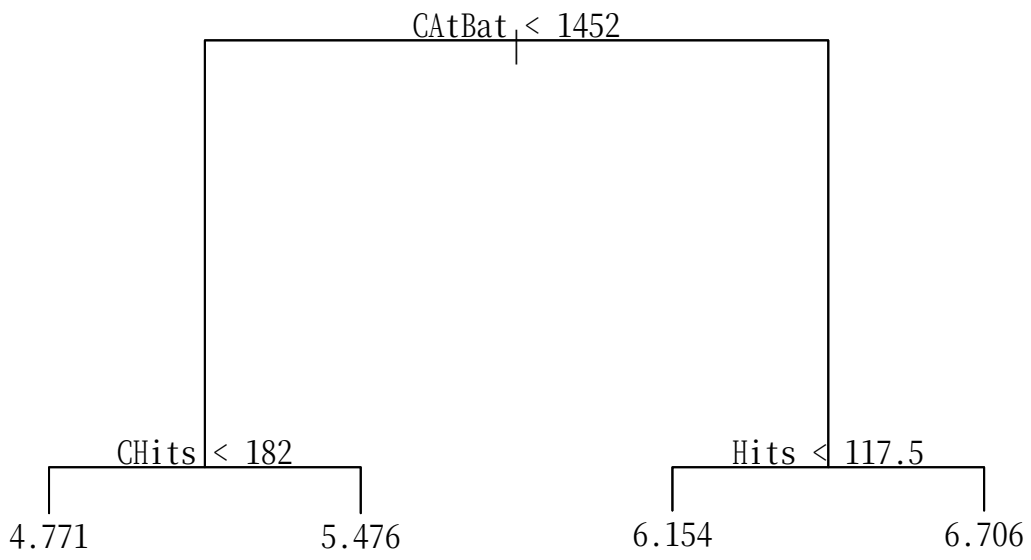
```
## [1] 224692.1
```

用所有数据来构造未剪枝树：

```
tr2 <- tree(log(Salary) ~ ., data=d)
```

用训练集上得到的子树大小剪枝：

```
tr2b <- prune.tree(tr2, best=best.size)
plot(tr2b); text(tr2b, pretty=0)
```



38.2.6 装袋法

判别树在不同的训练集、测试集划分上可以产生很大变化，说明其预测值方差较大。利用 bootstrap 的思想，可以随机选取许多个训练集，把许多个训练集的模型结果平均，就可以降低预测值的方差。

办法是从一个训练集中用有放回抽样的方法抽取 B 个训练集，设第 b 个抽取的训练集得到的回归函数为 $\hat{f}^{*b}(\cdot)$, 则最后的回归函数是这些回归函数的平均值:

$$\hat{f}_{\text{bagging}}(x) = \frac{1}{B}\sum_{b=1}^{b}\hat{f}^{*b}(x)$$

这称为装袋法 (bagging)。装袋法对改善判别与回归树的精度十分有效。

装袋法的步骤如下：

- 从训练集中取 B 个有放回随机抽样的 bootstrap 训练集，B 取为几百到几千之间。
- 对每个 bootstrap 训练集，估计未剪枝的树。
- 如果因变量是连续变量，对测试样品，用所有的树的预测值的平均值作预测。
- 如果因变量是分类变量，对测试样品，可以用所有树预测类的多数投票决定预测值。

装袋法也可以用来改进其他的回归和判别方法。

装袋后不能再用图形表示，模型可解释性较差。但是，可以度量自变量在预测中的重要程度。在回归问题中，可以计算每个自变量在所有 B 个树种平均减少的残差平方和的量，以此度量其重要度。在判别问题中，可以计算每个自变量在所有 B 个树种平均减少的基尼系数的量，以此度量其重要度。

除了可以用测试集、交叉验证方法以外，还可以使用袋外观测预测误差。用 bootstrap 再抽样获得多个训练集时每个 bootstrap 训练集总会遗漏一些观测，平均每个 bootstrap 训练集会遗漏三分之一的观测。对每个观测，大约有 $B/3$ 棵树没有用到此观测，可以用这些树的预测值平均来预测此观测，得到一个误差估计，这样得到的均方误差估计或错判率称为**袋外观测估计**（OOB 估计）。好处是不用很多额外的工作。

对训练集用装袋法：

```
bag1 <- randomForest(log(Salary) ~ ., data=d, subset=train, mtry=ncol(d)-1, importance=TRUE)
bag1
```

```
##
## Call:
##  randomForest(formula = log(Salary) ~ ., data = d, mtry = ncol(d) -      1, importance = TRUE, sub
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 19
##
##           Mean of squared residuals: 0.2549051
##                     % Var explained: 67.32
```

注意 `randomForest()` 函数实际是随机森林法，但是当 `mtry` 的值取为所有自变量个数时就是装袋法。

对测试集进行预报:

```
pred2 <- predict(bag1, newdata=d[test,])
test.mse2 <- mean( (d[test, 'Salary'] - exp(pred2))^2 )
test.mse2
```

```
## [1] 89851.48
```

结果与剪枝过的单课树相近。

在全集上使用装袋法：

```
bag2 <- randomForest(log(Salary) ~ ., data=d, mtry=ncol(d)-1, importance=TRUE)
bag2
```

```
##
## Call:
```

```
##  randomForest(formula = log(Salary) ~ ., data = d, mtry = ncol(d) -      1, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 19
##
##           Mean of squared residuals: 0.1894008
##                     % Var explained: 75.95
```

变量的重要度数值和图形：各变量的重要度数值及其图形：

```
importance(bag2)
```

```
##              %IncMSE IncNodePurity
## AtBat     10.4186792     8.9315248
## Hits       8.0033436     7.5938472
## HmRun      3.6180992     1.9689157
## Runs       7.2586283     3.9341954
## RBI        5.9223739     5.9201328
## Walks      7.6449979     6.6988173
## Years      9.4732817     2.2977104
## CAtBat    28.2381456    84.4338845
## CHits     13.8405414    26.2455674
## CHmRun     6.7109109     3.8246805
## CRuns     13.6067783    29.1340423
## CRBI      14.1694017    10.9852537
## CWalks     7.7656943     4.1725799
## League    -1.0964577     0.2146176
## Division   0.5534057     0.2307037
## PutOuts    0.3157195     4.1169655
## Assists   -1.7730978     1.6599765
## Errors     2.3420783     1.6852796
## NewLeague -0.3091532     0.3747445
```

```
varImpPlot(bag2)
```

最重要的自变量是 CAtBats, 其次有 CRuns, CHits 等。

38.2.7 随机森林

随机森林的思想与装袋法类似，但是试图使得参加平均的各个树之间变得比较独立。仍采用有放回抽样得到的多个 bootstrap 训练集，但是对每个 bootstrap 训练集构造判别树时，每次分叉时不考虑所有自变量，而是仅考虑随机

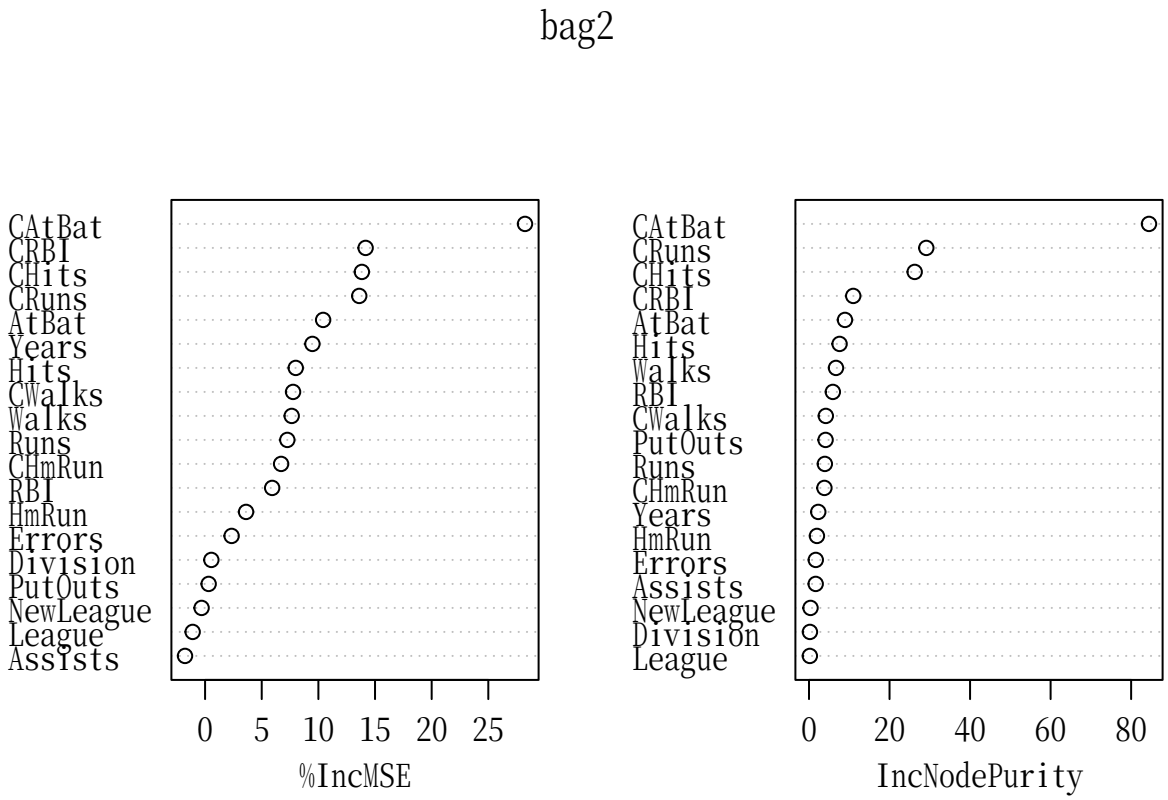


图 38.6: Hitters 数据装袋法的变量重要性结果

选取的一个自变量子集。

对判别树，每次分叉时选取的自变量个数通常取 $m \approx \sqrt{p}$ 个。比如，对 Heart 数据的 13 个自变量，每次分叉时仅随机选取 4 个纳入考察范围。

随机森林的想法是基于正相关的样本在平均时并不能很好地降低方差，独立样本能比较好地降低方差。如果存在一个最重要的变量，如果不加限制这个最重要的变量总会是第一个分叉，使得 B 棵树相似程度很高。随机森林解决这个问题的办法是限制分叉时可选的变量子集。

随机森林也可以用来改进其他的回归和判别方法。

装袋法和随机森林都可以用 R 扩展包 randomForest 的 `randomForest()` 函数实现。当此函数的 `mtry` 参数取为自变量个数时，执行的就是装袋法；`mtry` 取缺省值时，执行随机森林算法。执行随机森林算法时，`randomForest()` 函数在回归问题时分叉时考虑的自变量个数取 $m \approx p/3$，在判别问题时取 $m \approx \sqrt{p}$。

对训练集用随机森林法：

```
rf1 <- randomForest(log(Salary) ~ ., data=d, subset=train, importance=TRUE)
rf1
```

```
##
## Call:
##  randomForest(formula = log(Salary) ~ ., data = d, importance = TRUE,      subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##           Mean of squared residuals: 0.2422914
##                     % Var explained: 68.94
```

当 `mtry` 的值取为缺省值时执行随机森林算法。

对测试集进行预报:

```
pred3 <- predict(rf1, newdata=d[test,])
test.mse3 <- mean( (d[test, 'Salary'] - exp(pred3))^2 )
test.mse3
```

```
## [1] 95455.53
```

结果与剪枝过的单课树、装袋法相近。

在全集上使用随机森林：

```
rf2 <- randomForest(log(Salary) ~ ., data=d, importance=TRUE)
rf2
```

```
##
## Call:
##  randomForest(formula = log(Salary) ~ ., data = d, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##           Mean of squared residuals: 0.1789257
##                     % Var explained: 77.28
```

各变量的重要度数值及其图形：

```
importance(rf2)
```

```
##              %IncMSE IncNodePurity
## AtBat     10.1010786     7.6235571
## Hits       8.3614365     7.9201425
## HmRun      3.7354302     2.4553471
## Runs       7.9446786     4.6566217
## RBI        6.6246470     6.0251832
## Walks      8.8848565     5.9901840
## Years     11.3153391     8.4097999
## CAtBat    18.0377154    41.0851904
## CHits     17.5110322    37.8686798
## CHmRun     8.4476983     6.9078686
## CRuns     14.8793512    30.8423409
## CRBI      14.8308800    20.5339522
## CWalks     9.7578555    14.9467745
## League    -0.6619015     0.3041459
## Division  -0.3583808     0.2954341
## PutOuts    2.4366422     3.4901980
## Assists   -0.0965240     1.8105096
## Errors     1.5007791     1.7068960
## NewLeague  1.0593259     0.3453514
```

```
varImpPlot(rf2)
```

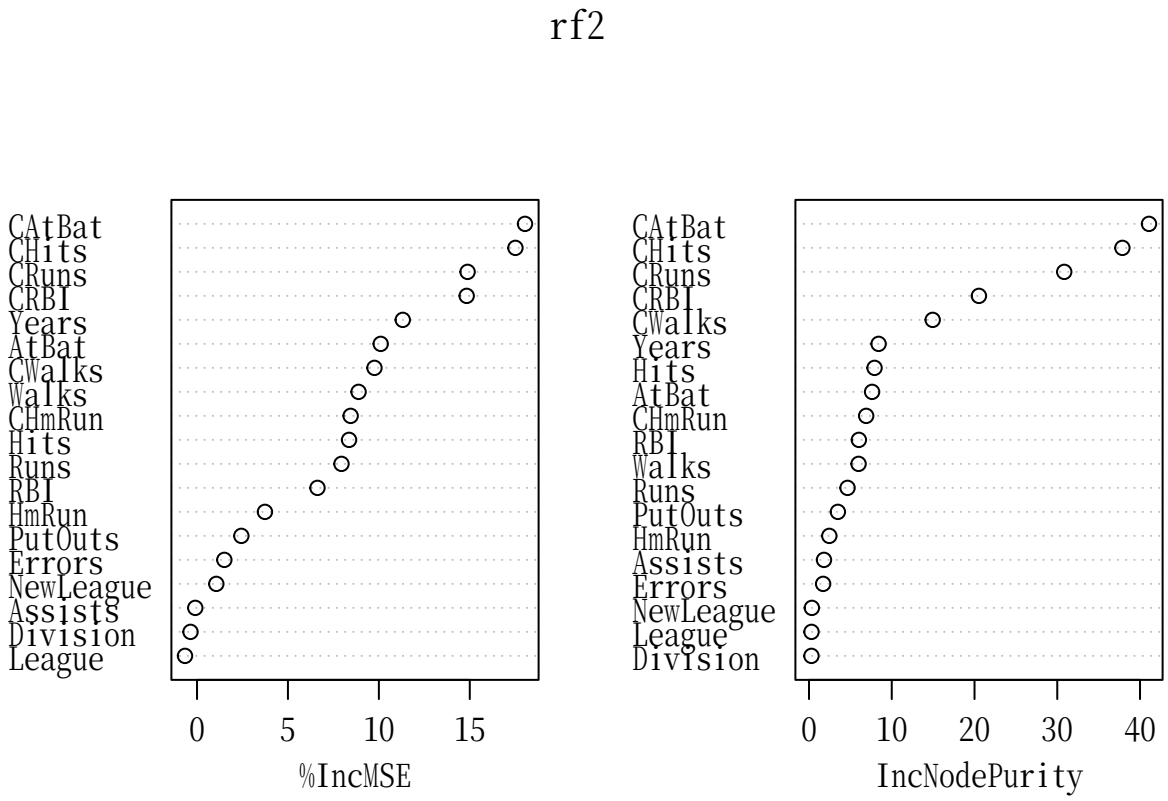


图 38.7: Hitters 数据随机森林法的变量重要度结果

最重要的自变量是 CAtBats, CRuns, CHits, CWalks, CRBI 等。

38.3 Heart 数据分析

Heart 数据是心脏病诊断的数据，因变量 AHD 为是否有心脏病，试图用各个自变量预测（判别）。

读入 Heart 数据集，并去掉有缺失值的观测：

```
Heart <- read.csv(
  "data/Heart.csv", header=TRUE, row.names=1,
  stringsAsFactors=TRUE)
Heart <- na.omit(Heart)
str(Heart)
```

```
## 'data.frame':    297 obs. of  14 variables:
```

```
##  $ Age      : int  63 67 67 37 41 56 62 57 63 53 ...
##  $ Sex      : int  1 1 1 1 0 1 0 0 1 1 ...
##  $ ChestPain: Factor w/ 4 levels "asymptomatic",..: 4 1 1 2 3 3 1 1 1 1 ...
##  $ RestBP   : int  145 160 120 130 130 120 140 120 130 140 ...
##  $ Chol     : int  233 286 229 250 204 236 268 354 254 203 ...
##  $ Fbs      : int  1 0 0 0 0 0 0 0 0 1 ...
##  $ RestECG  : int  2 2 2 0 2 0 2 0 2 2 ...
##  $ MaxHR    : int  150 108 129 187 172 178 160 163 147 155 ...
##  $ ExAng    : int  0 1 1 0 0 0 0 1 0 1 ...
##  $ Oldpeak  : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
##  $ Slope    : int  3 2 2 3 1 1 3 1 2 3 ...
##  $ Ca       : int  0 3 2 0 0 0 2 0 1 0 ...
##  $ Thal     : Factor w/ 3 levels "fixed","normal",..: 1 2 3 2 2 2 2 2 3 3 ...
##  $ AHD      : Factor w/ 2 levels "No","Yes": 1 2 2 1 1 1 2 1 2 2 ...
##  - attr(*, "na.action")= 'omit' Named int [1:6] 88 167 193 267 288 303
##   ..- attr(*, "names")= chr [1:6] "88" "167" "193" "267" ...
```

```
t(summary(Heart))
```

```
##
##      Age         Min.   :29.00      1st Qu.:48.00      Median :56.00      Mean   :54.54      3rd Q
##      Sex         Min.   :0.0000     1st Qu.:0.0000     Median :1.0000     Mean   :0.6768     3rd Q
##        ChestPain asymptomatic:142   nonanginal  : 83   nontypical  : 49   typical     : 23
##     RestBP       Min.   : 94.0      1st Qu.:120.0      Median :130.0      Mean   :131.7      3rd Q
##      Chol        Min.   :126.0      1st Qu.:211.0      Median :243.0      Mean   :247.4      3rd Q
##      Fbs         Min.   :0.0000     1st Qu.:0.0000     Median :0.0000     Mean   :0.1448     3rd Q
##    RestECG       Min.   :0.0000     1st Qu.:0.0000     Median :1.0000     Mean   :0.9966     3rd Q
##     MaxHR        Min.   : 71.0      1st Qu.:133.0      Median :153.0      Mean   :149.6      3rd Q
##     ExAng        Min.   :0.0000     1st Qu.:0.0000     Median :0.0000     Mean   :0.3266     3rd Q
##    Oldpeak       Min.   :0.000      1st Qu.:0.000      Median :0.800      Mean   :1.056      3rd Q
##     Slope        Min.   :1.000      1st Qu.:1.000      Median :2.000      Mean   :1.603      3rd Q
##       Ca         Min.   :0.0000     1st Qu.:0.0000     Median :0.0000     Mean   :0.6768     3rd Q
##         Thal     fixed     : 18     normal    :164     reversable:115
##  AHD             No :160            Yes:137
```

数据下载：Heart.csv

38.3.1 树回归

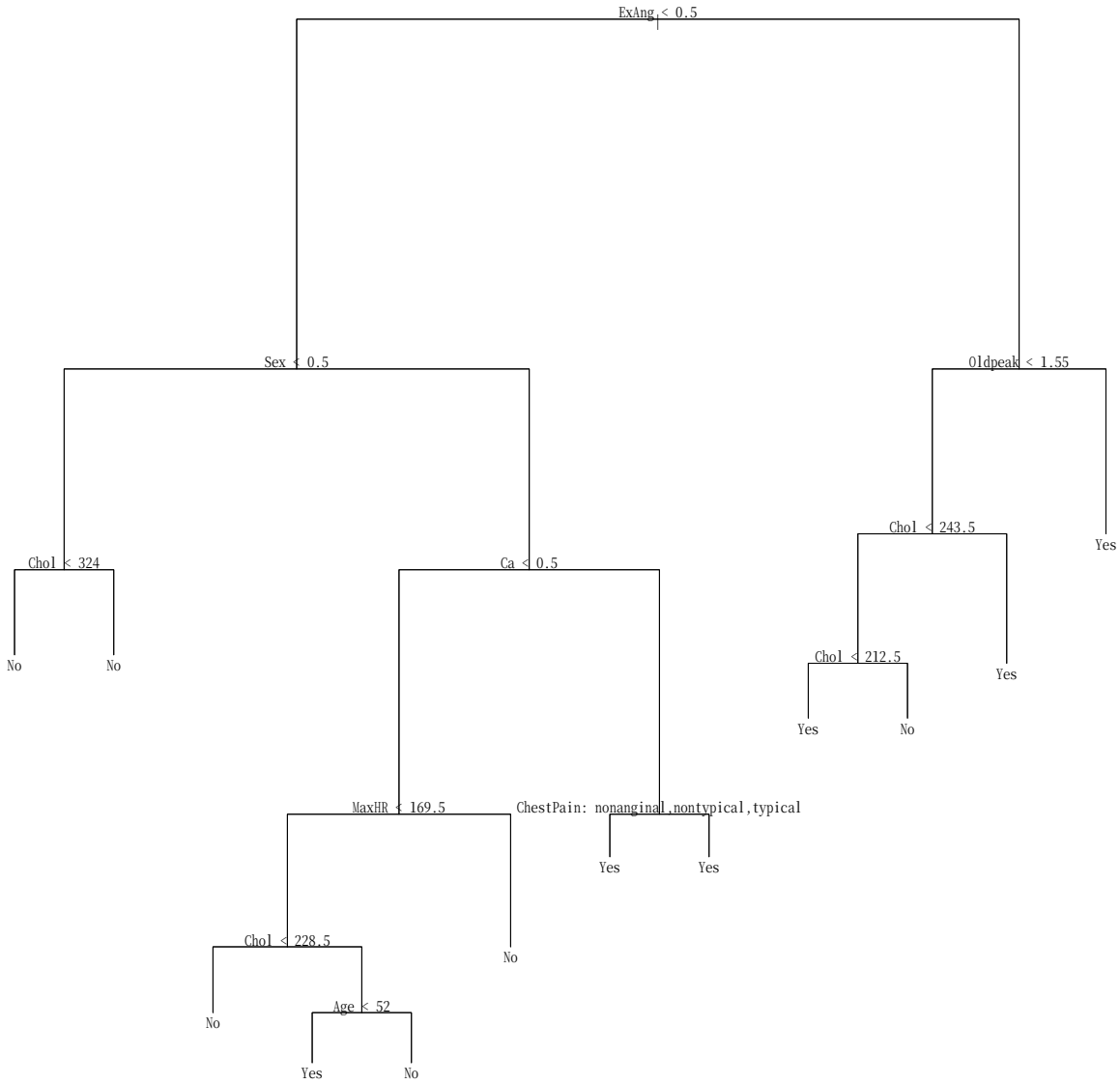
38.3.1.1 划分训练集与测试集

简单地把观测分为一半训练集、一半测试集：

```
set.seed(1)
train <- sample(nrow(Heart), size=round(nrow(Heart)/2))
test <- (-train)
test.y <- Heart[test, 'AHD']
```

在训练集上建立未剪枝的判别树:

```
tr1 <- tree(AHD ~ ., data=Heart[train,])
plot(tr1); text(tr1, pretty=0)
```



38.3.1.2 适当剪枝

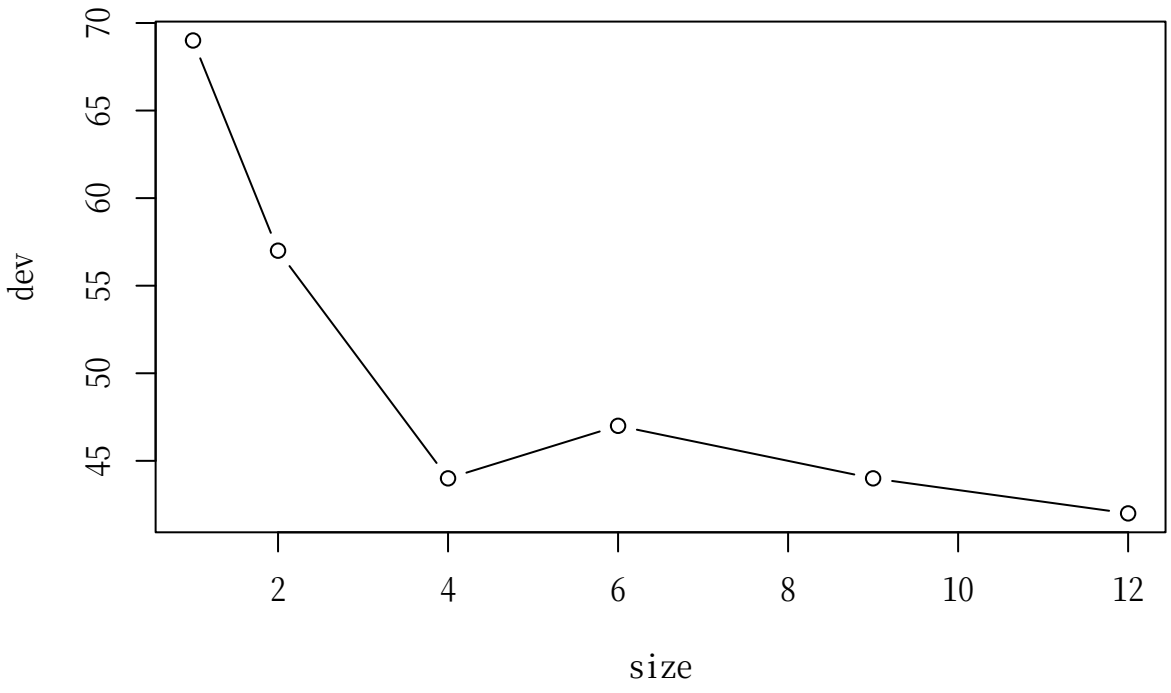
用交叉验证方法确定剪枝保留的叶子个数，剪枝时按照错判率执行：

```
cv1 <- cv.tree(tr1, FUN=prune.misclass)
cv1
```

```
## $size
```

```
## [1] 12  9  6  4  2  1
##
## $dev
## [1] 42 44 47 44 57 69
##
## $k
## [1]      -Inf  0.000000  1.666667  3.000000  7.000000 26.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
plot(cv1$size, cv1$dev, type='b', xlab='size', ylab='dev')
```




```
best.size <- cv1$size[which.min(cv1$dev)]
```

最优的大小是 12。但是从图上看，4 个叶结点已经足够好，所以取为 4。

对训练集生成剪枝结果：

```
best.size <- 4
tr1b <- prune.misclass(tr1, best=best.size)
plot(tr1b); text(tr1b, pretty=0)
```

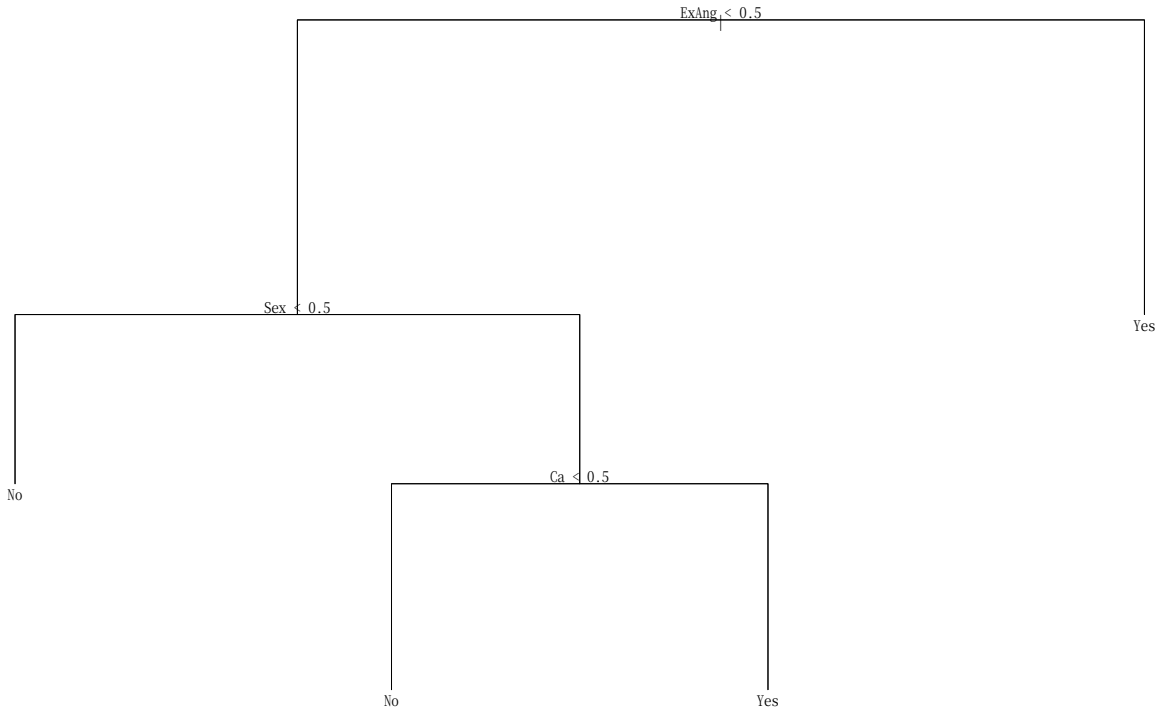


图 38.8: Heart 数据回归树

注意剪枝后树的显示中，内部节点的自变量存在分类变量，这时按照这个自变量分叉时，取指定的某几个分类值时对应分支 Yes，取其它的分类值时对应分支 No。

38.3.1.3 对测试集计算误判率

```
pred1 <- predict(tr1b, Heart[test,], type='class')
tab1 <- table(pred1, test.y); tab1
```

```
##      test.y
## pred1 No Yes
```

```
##   No  56  17
##   Yes 21  55
```

```
test.err <- (tab1[1,2]+tab1[2,1])/sum(tab1[]); test.err
```

```
## [1] 0.2550336
```

对测试集的错判率约 26%。

利用未剪枝的树对测试集进行预测, 一般比剪枝后的结果差:

```
pred1a <- predict(tr1, Heart[test,], type='class')
tab1a <- table(pred1a, test.y); tab1a
```

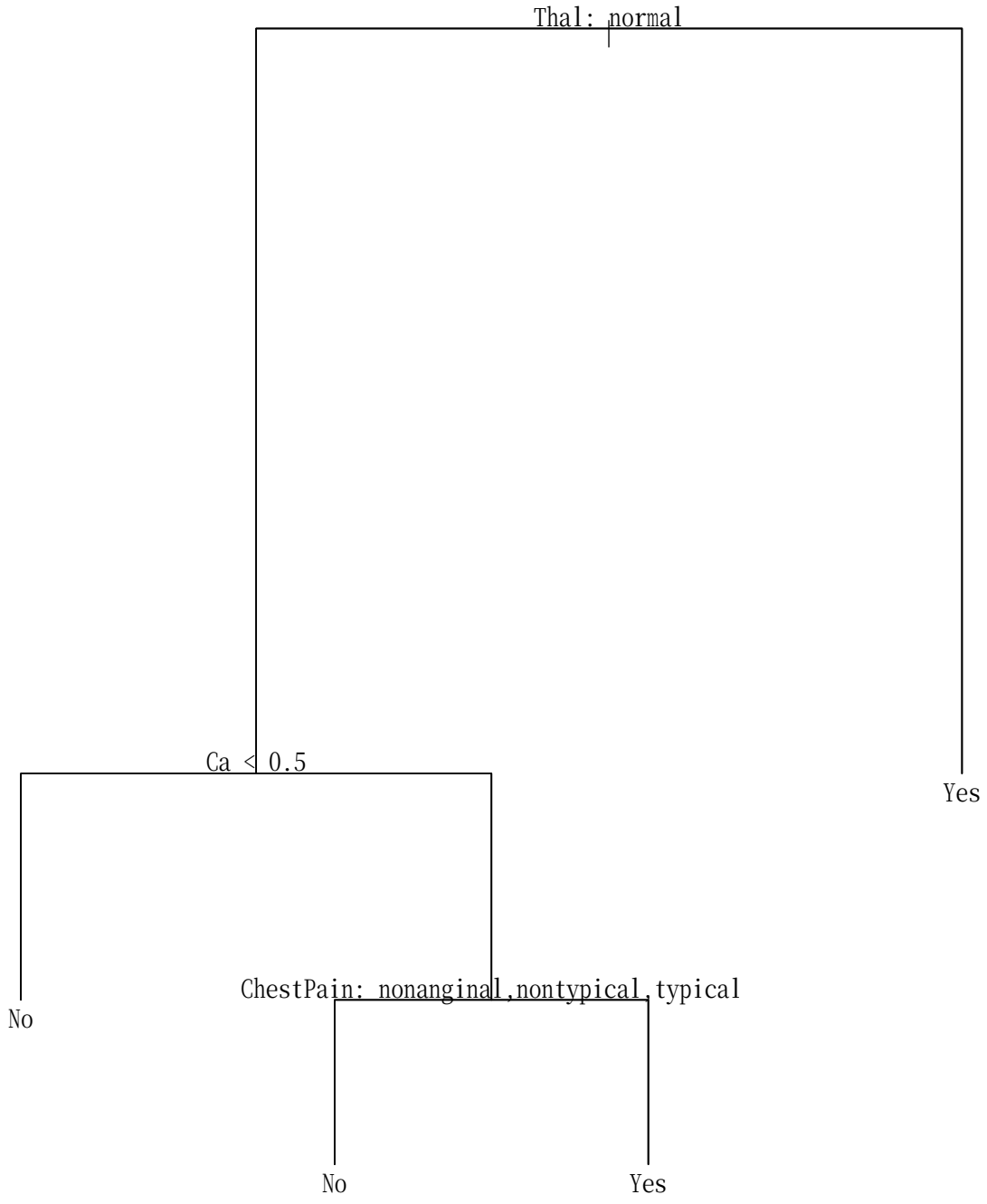
```
##       test.y
## pred1a No Yes
##    No  58  21
##    Yes 19  51
```

```
test.err1a <- (tab1a[1,2]+tab1a[2,1])/sum(tab1a[]); test.err1a
```

```
## [1] 0.2684564
```

38.3.1.4 利用全集数据建立剪枝判别树

```
tr2 <- tree(AHD ~ ., data=Heart)
tr2b <- prune.misclass(tr2, best=best.size)
plot(tr2b); text(tr2b, pretty=0)
```


Thal: normal
Ca < 0.5
Yes
No
ChestPain: nonanginal,nontypical,typical
No
Yes

38.3.2 用装袋法

对训练集用装袋法：

```
bag1 <- randomForest(AHD ~ ., data=Heart, subset=train, mtry=13, importance=TRUE)
bag1
```

```
##
## Call:
##  randomForest(formula = AHD ~ ., data = Heart, mtry = 13, importance = TRUE,      subset = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##         OOB estimate of  error rate: 22.3%
## Confusion matrix:
##     No Yes class.error
## No  71  12   0.1445783
## Yes 21  44   0.3230769
```

注意 `randomForest()` 函数实际是随机森林法，但是当 `mtry` 的值取为所有自变量个数时就是装袋法。袋外观测得到的错判率比较差。

对测试集进行预报:

```
pred2 <- predict(bag1, newdata=Heart[test,])
tab2 <- table(pred2, test.y); tab2
```

```
##      test.y
## pred2 No Yes
##   No  66  17
##   Yes 11  55
```

```
test.err2 <- (tab2[1,2]+tab2[2,1])/sum(tab2[]); test.err2
```

```
## [1] 0.1879195
```

测试集的错判率约为 19%。

对全集用装袋法:

```
bag1b <- randomForest(AHD ~ ., data=Heart, mtry=13, importance=TRUE)
bag1b
```

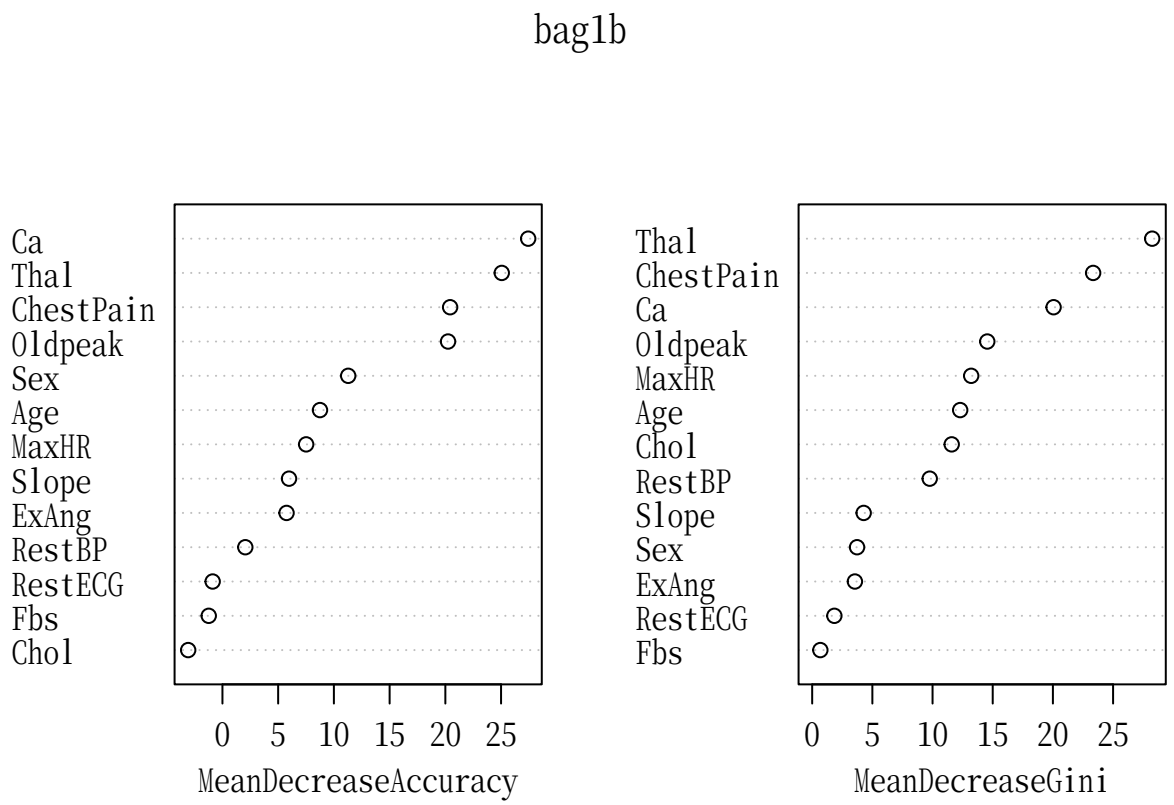
```
##
## Call:
##  randomForest(formula = AHD ~ ., data = Heart, mtry = 13, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##         OOB estimate of  error rate: 20.88%
## Confusion matrix:
##      No Yes class.error
## No  131  29   0.1812500
## Yes  33 104   0.2408759
```

各变量的重要度数值及其图形：

```
importance(bag1b)
```

```
##                   No         Yes MeanDecreaseAccuracy MeanDecreaseGini
## Age        6.5766876  5.12005531            8.7542379       12.2956568
## Sex       11.2077275  4.48390165           11.2853739        3.7278623
## ChestPain 13.0268932 17.89348038           20.4292863       23.3424850
## RestBP     2.6203153  0.05626759            2.0521195        9.7650173
## Chol      -0.8712348 -4.23294461           -3.0733270       11.5911988
## Fbs       -0.6941335 -1.16860850           -1.2288380        0.6775051
## RestECG   -1.4881617  0.23292163           -0.8772267        1.8426038
## MaxHR      7.7625054  2.34660468            7.5122314       13.2101707
## ExAng      2.7926364  5.45108497            5.7525854        3.5491718
## Oldpeak   14.8193517 14.67748373           20.2425364       14.5480191
## Slope      2.5189935  5.73789018            5.9744484        4.2777028
## Ca        23.0513399 18.01671793           27.4320740       20.0564750
## Thal      20.1968435 18.74418431           25.0618361       28.2479833
```

```
varImpPlot(bag1b)
```



最重要的变量是 Thal, ChestPain, Ca。

38.3.3 用随机森林

对训练集用随机森林法：

```
rf1 <- randomForest(AHD ~ ., data=Heart, subset=train, importance=TRUE)
rf1
```

```
##
## Call:
##  randomForest(formula = AHD ~ ., data = Heart, importance = TRUE,      subset = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 21.62%
## Confusion matrix:
##     No Yes class.error
```

```
## No  71  12   0.1445783
## Yes 20  45   0.3076923
```

这里 mtry 取缺省值，对应于随机森林法。

对测试集进行预报:

```
pred3 <- predict(rf1, newdata=Heart[test,])
tab3 <- table(pred3, test.y); tab3
```

```
##      test.y
## pred3 No Yes
##   No  70  16
##   Yes  7  56
```

```
test.err3 <- (tab3[1,2]+tab3[2,1])/sum(tab3[]); test.err3
```

```
## [1] 0.1543624
```

测试集的错判率约为 15%。

对全集用随机森林:

```
rf1b <- randomForest(AHD ~ ., data=Heart,  importance=TRUE)
rf1b
```

```
##
## Call:
##  randomForest(formula = AHD ~ ., data = Heart, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 16.5%
## Confusion matrix:
##      No Yes class.error
## No  140  20   0.1250000
## Yes  29 108   0.2116788
```

各变量的重要度数值及其图形:

```
importance(rf1b)
```

```
##                   No        Yes MeanDecreaseAccuracy MeanDecreaseGini
## Age        7.2380857  5.4451404            9.1859647        12.908917
## Sex       10.1973138  8.0790483           12.6929315         4.938266
## ChestPain 10.4623927 16.7054395           18.8771946        18.218363
## RestBP     1.2157266  1.8875229            2.1025511        10.624864
## Chol      -1.2630538 -0.4285615           -1.3028275        11.470420
## Fbs        0.4417651 -2.6574949           -1.4327524         1.418137
## RestECG   -1.1149040  1.4649476            0.2220661         2.840670
## MaxHR      9.3788412  6.0542618           10.7139921        17.383623
## ExAng      3.3923281  9.8037831            9.3523828         6.715947
## Oldpeak   10.1617047 14.3372404           17.5616061        15.403935
## Slope      2.6703016  9.3147738            8.5774100         6.752552
## Ca        21.1750038 20.2285033           26.7114362        18.388524
## Thal      18.0250446 16.6731737           22.5365589        18.351175
```

```
varImpPlot(rf1b)
```

最重要的变量是 ChestPain, Thal, Ca。

38.4 汽车销量数据分析

Carseats 是 ISLR 包的一个数据集，基本情况如下: `{rstatl-car-summ01, cache=TRUE}` `str(Carseats)` `summary(Carseats)`

把 Salses 变量按照大于 8 与否分成两组，结果存入变量 High，以 High 为因变量作判别分析。

```
d <- na.omit(Carseats)
d$High <- factor(ifelse(d$Sales > 8, 'Yes', 'No'))
dim(d)
```

```
## [1] 400  12
```

38.4.1 判别树

38.4.1.1 全体数据的判别树

对全体数据建立未剪枝的判别树:

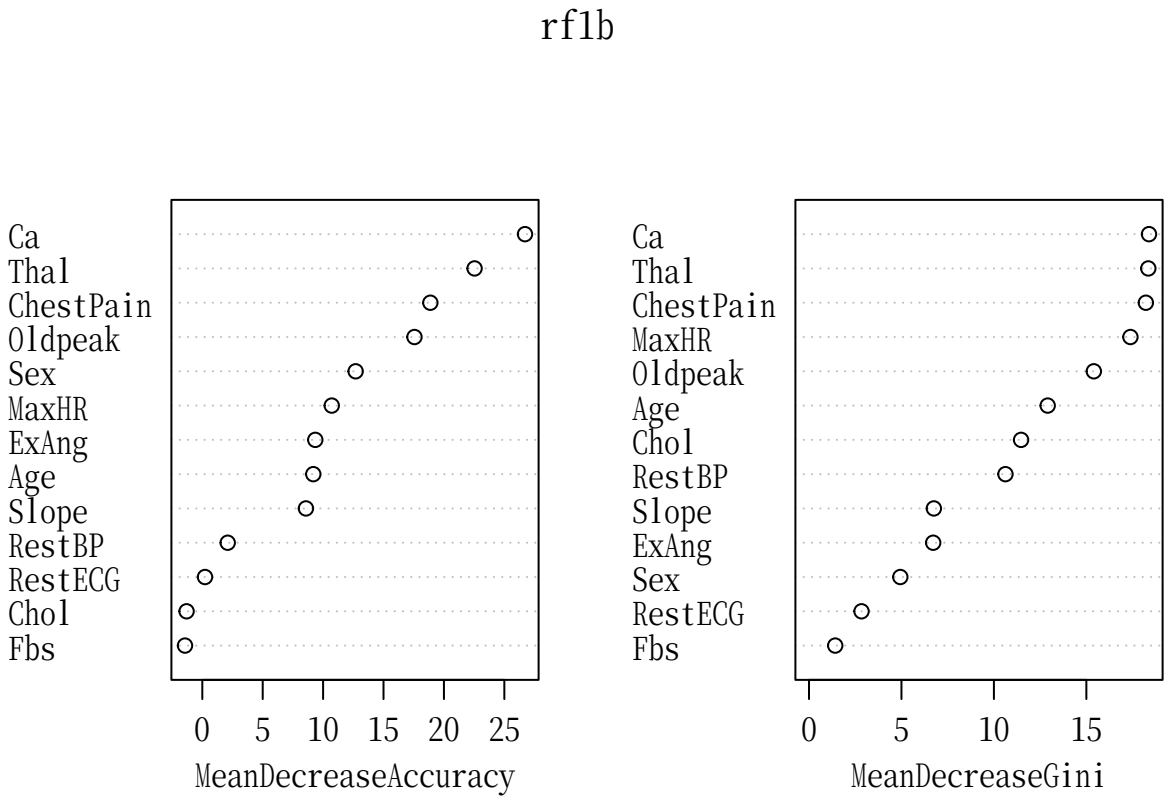
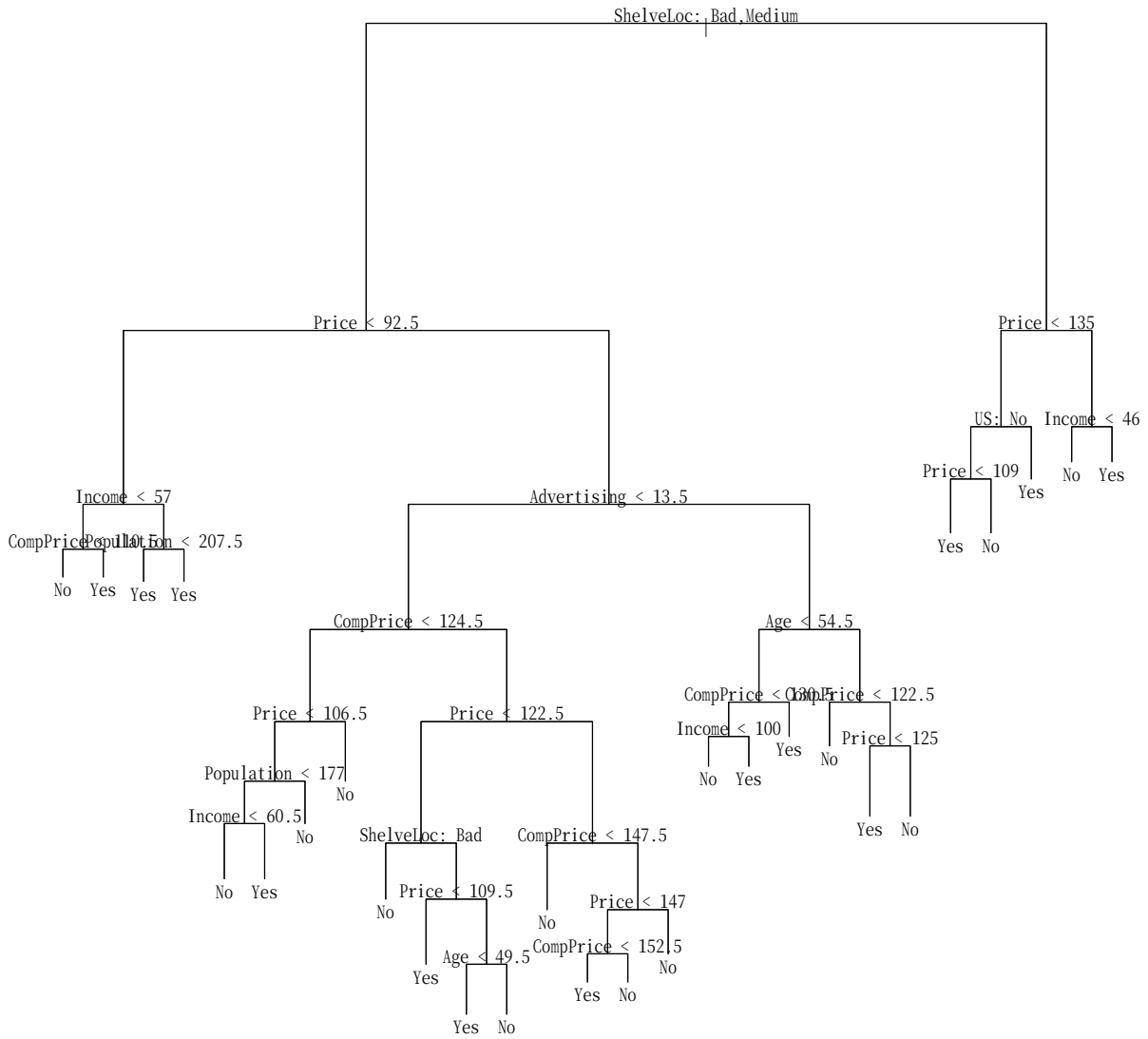


图 38.9: Heart 数据随机森林方法得到的变量重要度

```r
tr1 <- tree(High ~ . - Sales, data=d)
summary(tr1)
```

```
##
## Classification tree:
## tree(formula = High ~ . - Sales, data = d)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"  "Advertising" "Age"
## Number of terminal nodes:  27
## Residual mean deviance:  0.4575 = 170.7 / 373
## Misclassification error rate: 0.09 = 36 / 400
```

```r
plot(tr1)
text(tr1, pretty=0)
```



38.4.1.2 划分训练集和测试集

把输入数据集随机地分一半当作训练集，另一半当作测试集：

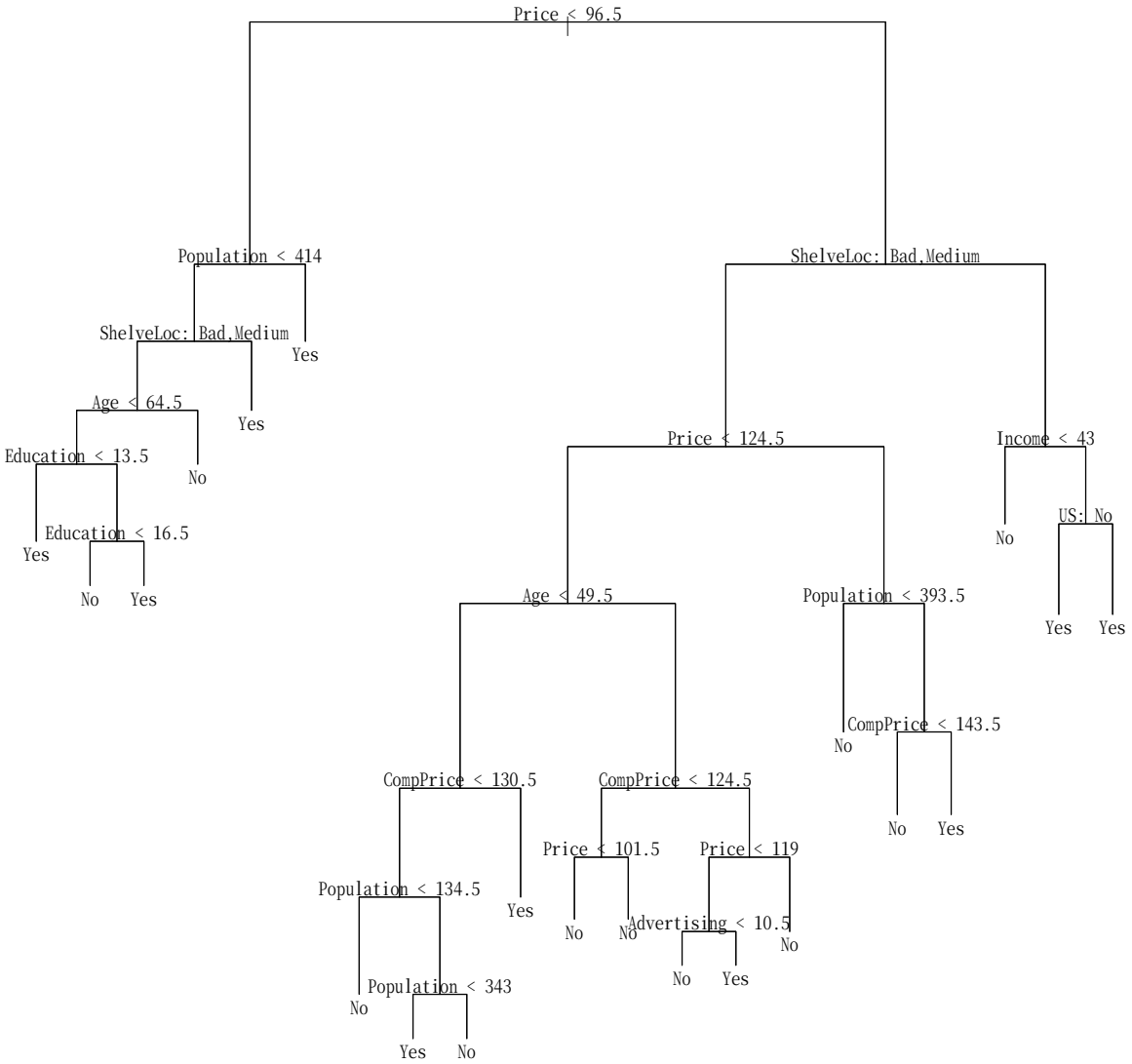
```
set.seed(2)
train <- sample(nrow(d), size=round(nrow(d)/2))
test <- (-train)
test.high <- d[test, 'High']
```

用训练数据建立未剪枝的判别树:

```
tr2 <- tree(High ~ . - Sales, data=d, subset=train)
summary(tr2)
```

```
##
## Classification tree:
## tree(formula = High ~ . - Sales, data = d, subset = train)
## Variables actually used in tree construction:
## [1] "Price"       "Population"  "ShelveLoc"   "Age"         "Education"   "CompPrice"   "Advertisi
## Number of terminal nodes:  21
## Residual mean deviance:  0.5543 = 99.22 / 179
## Misclassification error rate: 0.115 = 23 / 200
```

```
plot(tr2)
text(tr2, pretty=0)
```



用未剪枝的树对测试集进行预测，并计算误判率：

```
pred2 <- predict(tr2, d[test,], type='class')
tab <- table(pred2, test.high); tab
```

```
##      test.high
## pred2  No Yes
##   No  104  33
##   Yes  13  50
```

```
test.err2 <- (tab[1,2] + tab[2,1]) / sum(tab[]); test.err2
```

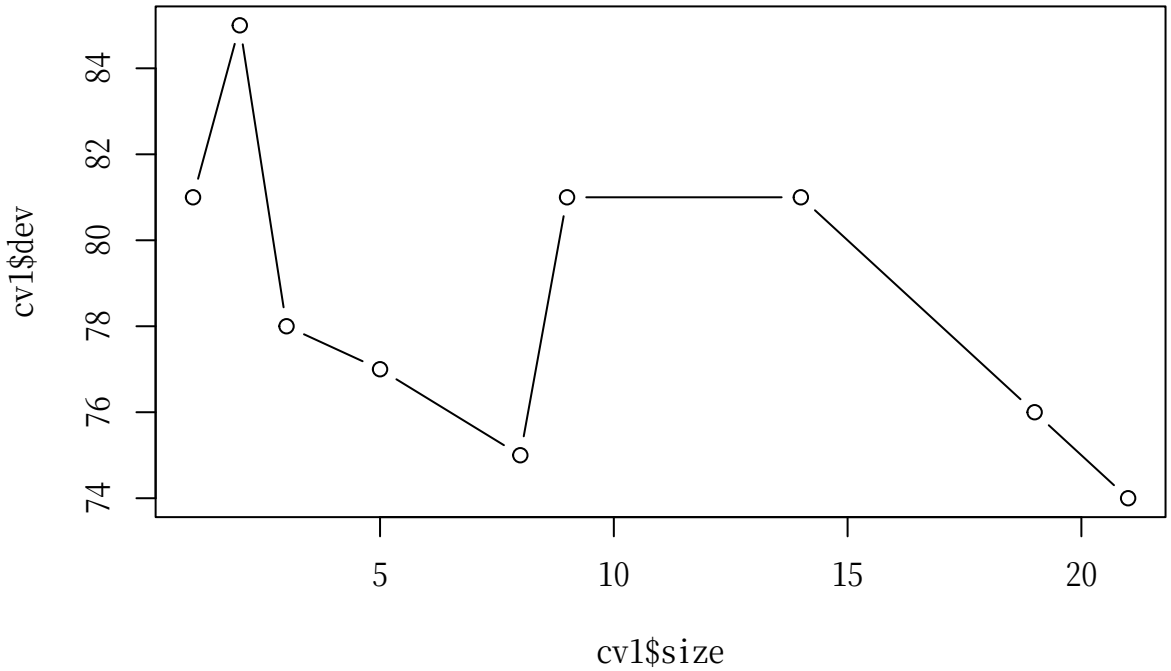
```
## [1] 0.23
```

38.4.1.3 用交叉验证确定训练集的剪枝

```
set.seed(3)
cv1 <- cv.tree(tr2, FUN=prune.misclass)
cv1
```

```
## $size
## [1] 21 19 14  9  8  5  3  2  1
##
## $dev
## [1] 74 76 81 81 75 77 78 85 81
##
## $k
## [1] -Inf  0.0  1.0  1.4  2.0  3.0  4.0  9.0 18.0
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
plot(cv1$size, cv1$dev, type='b')
```




```
best.size <- cv1$size[which.min(cv1$dev)]
```

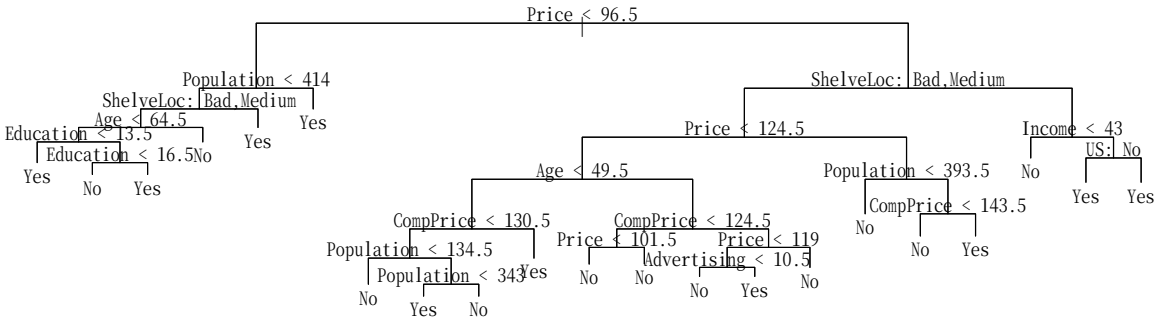
用交叉验证方法自动选择的最佳树大小为 21。

剪枝:

```
tr3 <- prune.misclass(tr2, best=best.size)
summary(tr3)
```

```
##
## Classification tree:
## tree(formula = High ~ . - Sales, data = d, subset = train)
## Variables actually used in tree construction:
## [1] "Price"       "Population"  "ShelveLoc"   "Age"         "Education"   "CompPrice"   "Advertisi
## Number of terminal nodes:  21
## Residual mean deviance:  0.5543 = 99.22 / 179
## Misclassification error rate: 0.115 = 23 / 200
```

```
plot(tr3)
text(tr3, pretty=0)
```



用剪枝后的树对测试集进行预测，计算误判率：

```
pred3 <- predict(tr3, d[test,], type='class')
tab <- table(pred3, test.high); tab
```

```
##      test.high
## pred3  No Yes
##   No  104  32
##   Yes  13  51
```

```
test.err3 <- (tab[1,2] + tab[2,1]) / sum(tab[]); test.err3
```

```
## [1] 0.225
```

38.4.2 随机森林

对训练集用随机森林法：

```
rf4 <- randomForest(High ~ . - Sales, data=d, subset=train, importance=TRUE)
rf4
```

```
##
## Call:
##  randomForest(formula = High ~ . - Sales, data = d, importance = TRUE,      subset = train)
##                Type of random forest: classification
```

```
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 25.5%
## Confusion matrix:
##      No Yes class.error
## No  102  17   0.1428571
## Yes  34  47   0.4197531
```

这里 mtry 取缺省值，对应于随机森林法。

对测试集进行预报:

```
pred4 <- predict(rf4, newdata=d[test,])
tab <- table(pred4, test.high); tab
```

```
##      test.high
## pred4  No Yes
##   No  109  24
##   Yes   8  59
```

```
test.err4 <- (tab[1,2]+tab[2,1])/sum(tab[]); test.err4
```

```
## [1] 0.16
```

注意错判率结果依赖于训练集和测试集的划分，另行选择训练集与测试集可能会得到很不一样的错判率结果。

对全集用随机森林:

```
rf5 <- randomForest(High ~ . - Sales, data=d,  importance=TRUE)
rf5
```

```
##
## Call:
##  randomForest(formula = High ~ . - Sales, data = d, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 18.25%
## Confusion matrix:
```

```
##      No Yes class.error
## No  213  23  0.09745763
## Yes  50 114  0.30487805
```

各变量的重要度数值及其图形：

```
importance(rf5)
```

```
##                     No        Yes MeanDecreaseAccuracy MeanDecreaseGini
## CompPrice   11.0998129  5.4168875          11.65469579        21.820876
## Income       3.2897388  4.7177705           5.75865440        20.384692
## Advertising 10.8093624 16.0263308          18.31175988        23.350563
## Population  -3.1872660 -1.7367798          -3.63402082        15.670307
## Price       30.0864270 28.7929995          37.44125656        43.492787
## ShelveLoc   30.2789749 33.8109594          39.67983055        30.053785
## Age          9.7116826  9.0261373          12.78808426        22.578000
## Education    0.2214031 -0.3203644           0.06365633         9.899447
## Urban        1.3826674  1.4199879           1.98859615         2.128048
## US           3.7289827  5.1909662           6.83788775         3.405420
```

```
varImpPlot(rf5)
```

重要的自变量为 Price, ShelfLoc, 其次有 Age, Advertising, CompPrice, Income 等。

38.5 波士顿郊区房价数据

MASS 包的 Boston 数据包含了波士顿地区郊区房价的若干数据。以中位房价 medv 为因变量建立回归模型。首先把缺失值去掉后存入数据集 d:

```
d <- na.omit(Boston)
```

数据集概况：

```
str(d)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
```

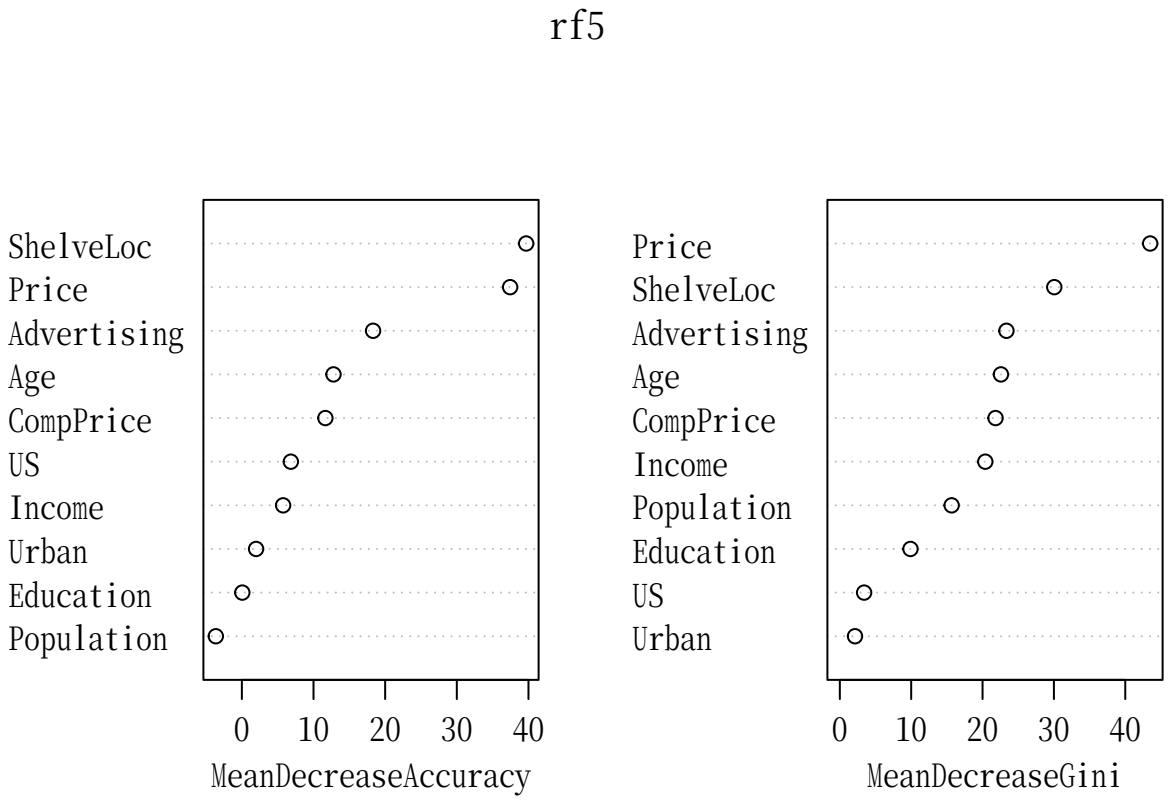


图 38.10: Carseats 数据随机森林法得到的变量重要度

```
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
summary(d)
```

```
##       crim                zn             indus            chas              nox               rm
##  Min.   : 0.00632   Min.   :  0.00   Min.   : 0.46   Min.   :0.00000   Min.   :0.3850   Min.   :3.
##  1st Qu.: 0.08205   1st Qu.:  0.00   1st Qu.: 5.19   1st Qu.:0.00000   1st Qu.:0.4490   1st Qu.:5.
##  Median : 0.25651   Median :  0.00   Median : 9.69   Median :0.00000   Median :0.5380   Median :6.
##  Mean   : 3.61352   Mean   : 11.36   Mean   :11.14   Mean   :0.06917   Mean   :0.5547   Mean   :6.
##  3rd Qu.: 3.67708   3rd Qu.: 12.50   3rd Qu.:18.10   3rd Qu.:0.00000   3rd Qu.:0.6240   3rd Qu.:6.
##  Max.   :88.97620   Max.   :100.00   Max.   :27.74   Max.   :1.00000   Max.   :0.8710   Max.   :8.
```

38.5.1 回归树

38.5.1.1 划分训练集和测试集

```
set.seed(1)
train <- sample(nrow(d), size=round(nrow(d)/2))
test <- (-train)
```

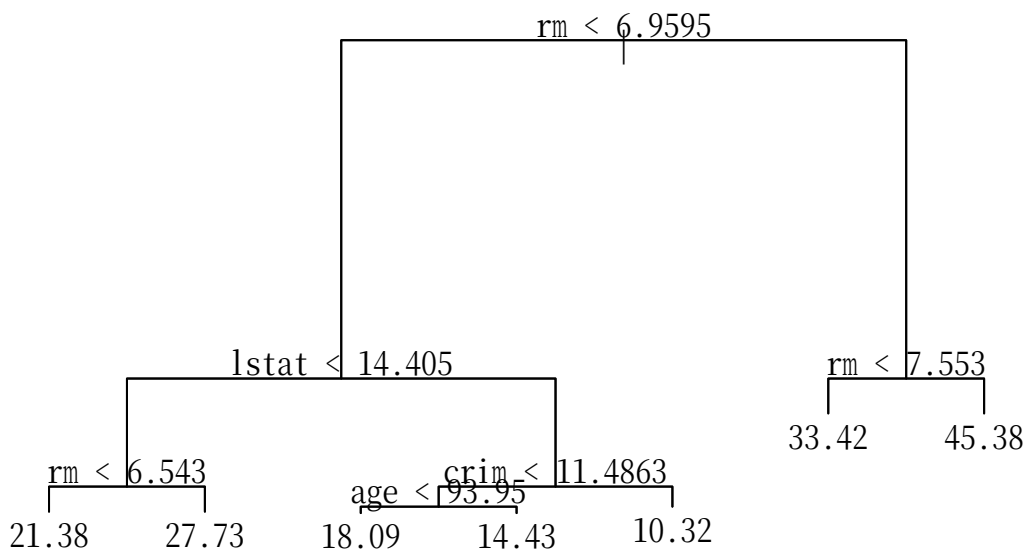
对训练集建立未剪枝的树：

```
tr1 <- tree(medv ~ ., d, subset=train)
summary(tr1)
```

```
##
## Regression tree:
## tree(formula = medv ~ ., data = d, subset = train)
## Variables actually used in tree construction:
## [1] "rm"    "lstat" "crim"  "age"
```

```
## Number of terminal nodes:  7
## Residual mean deviance:  10.38 = 2555 / 246
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -10.1800  -1.7770  -0.1775   0.0000   1.9230  16.5800
```

```
plot(tr1)
text(tr1, pretty=0)
```



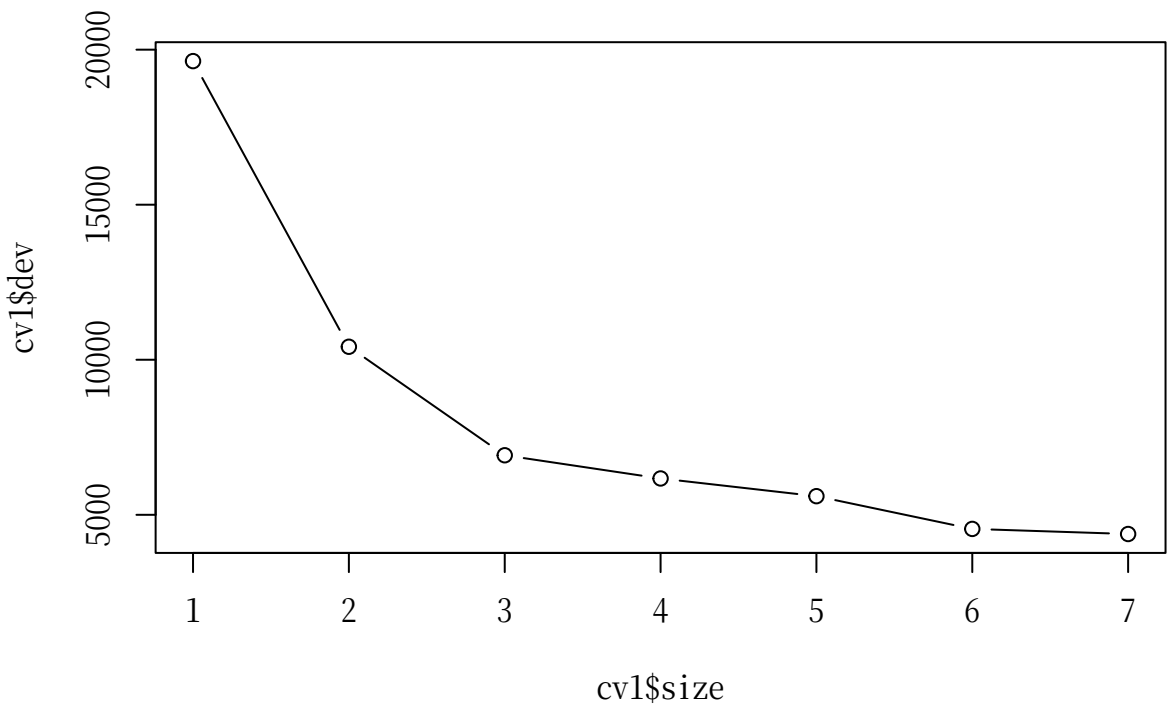
用未剪枝的树对测试集进行预测，计算均方误差：

```
yhat <-predict(tr1, newdata=d[test,])
mse1 <- mean((yhat - d[test, 'medv'])^2)
mse1
```

```
## [1] 35.28688
```

38.5.1.2 用交叉验证方法确定剪枝复杂度

```
cv1 <- cv.tree(tr1)
plot(cv1$size, cv1$dev, type='b')
```

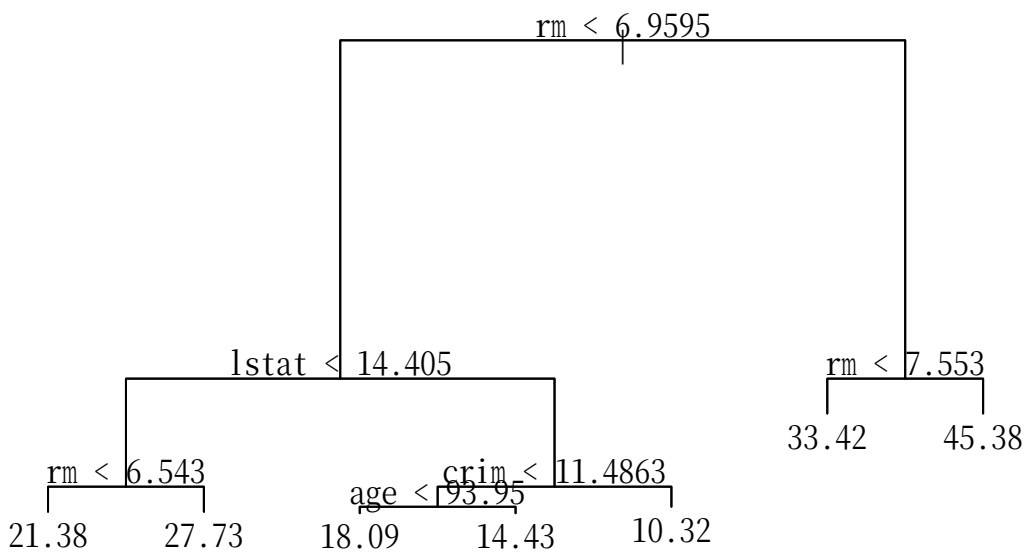



```
best.size <- cv1$size[which.min(cv1$dev)]; best.size
```

```
## [1] 7
```

剪枝并对测试集进行预测：

```
tr2 <- prune.tree(tr1, best=best.size)
plot(tr2)
text(tr2, pretty=0)
```




```
yhat <-predict(tr2, newdata=d[test,])
mse2 <- mean((yhat - d[test, 'medv'])^2)
mse2
```

```
## [1] 35.28688
```

剪枝后效果没有改善。

38.5.2 装袋法

用 randomForest 包计算。当参数 `mtry` 取为自变量个数时按照装袋法计算。对训练集计算。

```
set.seed(1)
bag1 <- randomForest(
  medv ~ ., data=d, subset=train,
  mtry=ncol(d)-1, importance=TRUE)
bag1
```

```
##
```

```
## Call:
##  randomForest(formula = medv ~ ., data = d, mtry = ncol(d) - 1,      importance = TRUE, subset = t
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##           Mean of squared residuals: 11.39601
##                     % Var explained: 85.17
```

在测试集上计算装袋法的均方误差：

```
yhat <- predict(bag1, newdata=d[test,])
mean( (yhat - d[test, 'medv'])^2 )
```

```
## [1] 23.59273
```

比单棵树的结果有明显改善。

38.5.3 随机森林

用 randomForest 包计算。当参数 `mtry` 取为缺省值时按照随机森林方法计算。对训练集计算。

```
set.seed(1)
rf1 <- randomForest(
  medv ~ ., data=d, subset=train,
  importance=TRUE)
rf1
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = d, importance = TRUE,      subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 10.23441
##                     % Var explained: 86.69
```

在测试集上计算随机森林法的均方误差：

```
yhat <- predict(rf1, newdata=d[test,])
mean( (yhat - d[test, 'medv'])^2 )
```

```
## [1] 18.11686
```

比单棵树的结果有明显改善, 比装袋法的结果也好一些。

各变量的重要度数值及其图形：

```
importance(rf1)
```

```
##           %IncMSE IncNodePurity
## crim    15.372334    1220.14856
## zn       3.335435     194.85945
## indus    6.964559    1021.94751
## chas     2.059298      69.68099
## nox     14.009761    1005.14707
## rm      28.693900    6162.30720
## age     13.832143     708.55138
## dis     10.317731     852.33701
## rad      4.390624     162.22597
## tax      7.536563     564.60422
## ptratio  9.333716    1163.39624
## black    8.341316     355.62445
## lstat   27.132450    5549.25088
```

```
varImpPlot(rf1)
```

38.5.4 提升法

提升法 (Boosting) 也是可以用在多种回归和判别问题中的方法。提升法的想法是，用比较简单的模型拟合因变量，计算残差，然后以残差为新的因变量建模，仍使用简单的模型，把两次的回归函数作加权和，得到新的残差后，再以新残差作为因变量建模，如此重复地更新回归函数，得到由多个回归函数加权和组成的最终的回归函数。

加权一般取为比较小的值，其目的是降低逼近速度。统计学习问题中降低逼近速度一般结果更好。

提升法算法:

- [(1)] 对训练集，设置 $r_i = y_i$，并令初始回归函数为 $\hat{f}(\cdot) = 0$。
- [(2)] 对 $b = 1, 2, \ldots, B$ 重复执行：

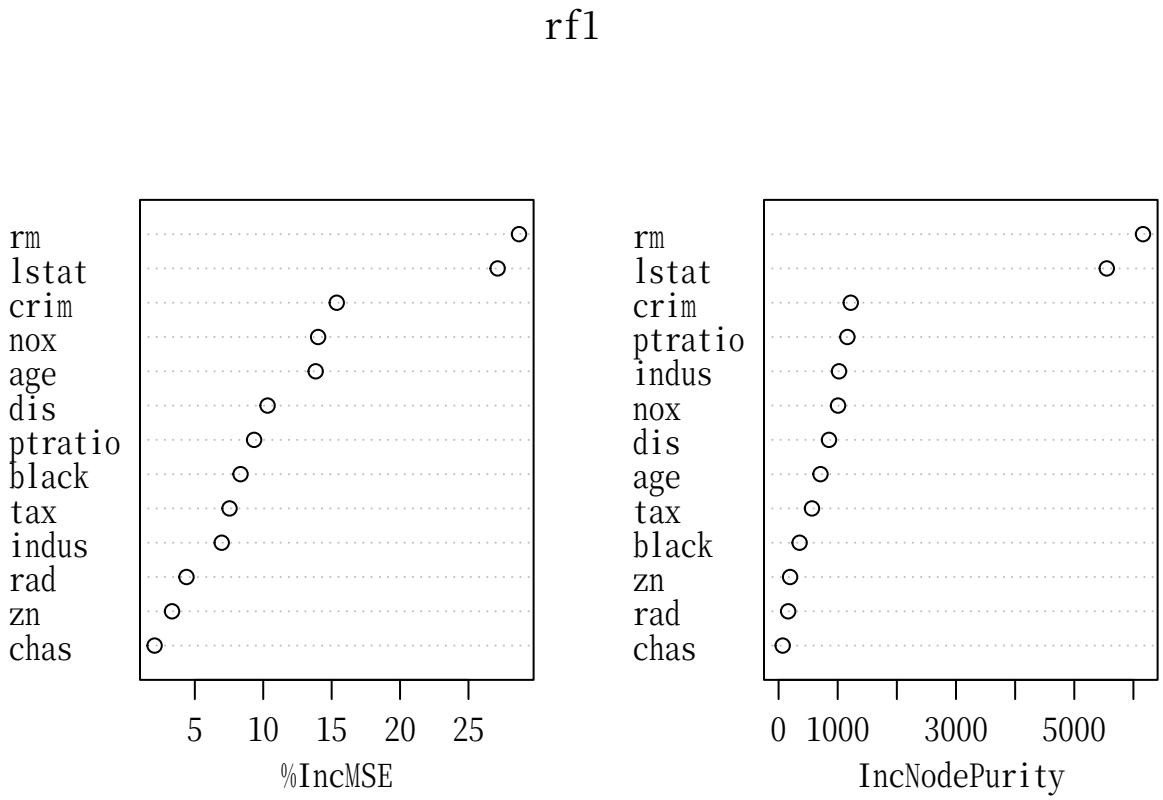


图 38.11: Boston 数据用随机森林法得到的变量重要度

- – [(a)] 以训练集的自变量为自变量，以 r 为因变量，拟合一个仅有 d 个分叉的简单树回归函数，设为 $\hat{f}_b$；

- – [(b)] 更新回归函数，添加一个压缩过的树回归函数:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}_b(x);$$

- – [(c)] 更新残差:

$$r_i \leftarrow r_i - \lambda \hat{f}_b(x_i).$$

- [(3)] 提升法的回归函数为

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}_b(x).$$

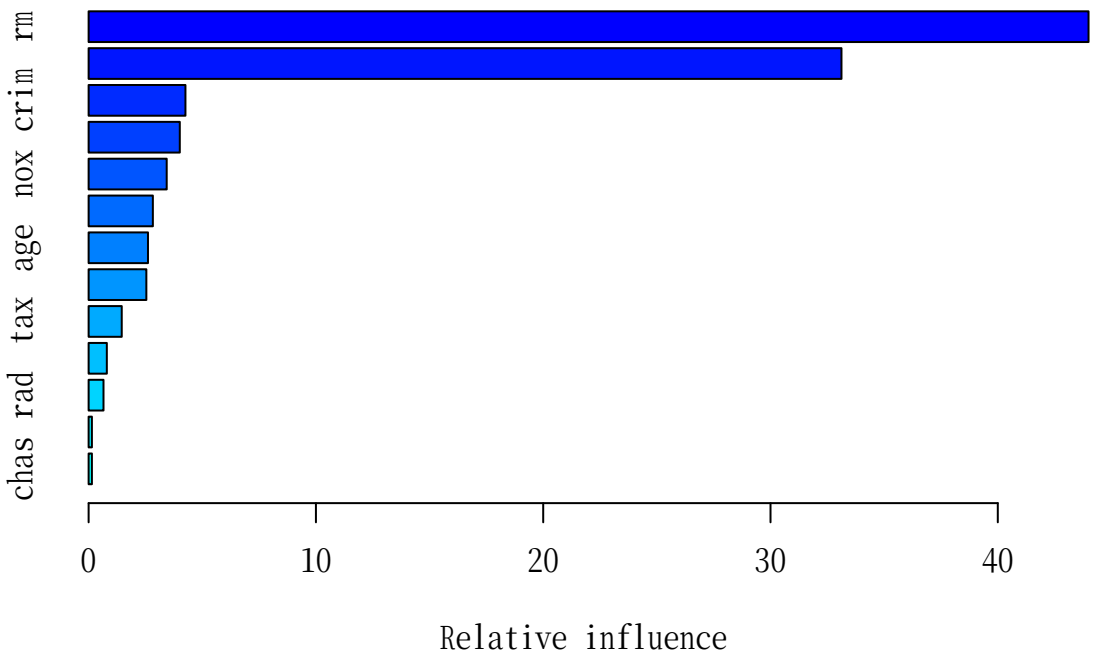
用多少个回归函数做加权和，即 B 的选取问题。取得 B 太大也会有过度拟合，但是只要 B 不太大这个问题不严重。可以用交叉验证选择 B 的值。

收缩系数 λ。是一个小的正数，控制学习速度，经常用 0.01, 0.001 这样的值，与要解决的问题有关。取 λ 很小，就需要取 B 很大。

用来控制每个回归函数复杂度的参数，对树回归而言就是树的大小。一个分叉的树往往就很好。取单个分叉时结果模型是可加模型，没有交互项，这是因为每个加权相加得回归函数都只依赖于单一自变量。$d > 1$ 时就加入了交互项。

使用 gbm 包。在训练集上拟合：

```
set.seed(1)
bst1 <- gbm(medv ~ ., data=d[train,],  distribution='gaussian',  n.trees=5000,  interaction.depth=4)
summary(bst1)
```




```
##              var    rel.inf
## rm            rm 43.9919329
## lstat     lstat 33.1216941
## crim       crim  4.2604167
## dis         dis  4.0111090
## nox         nox  3.4353017
## black     black  2.8267554
## age         age  2.6113938
## ptratio ptratio  2.5403035
## tax         tax  1.4565654
## indus     indus  0.8008740
## rad         rad  0.6546400
## zn           zn  0.1446149
## chas       chas  0.1443986
```

lstat 和 rm 是最重要的变量。

在测试集上预报，并计算均方误差：

```
yhat <- predict(bst1, newdata=d[test,], n.trees=5000)
mean( (yhat - d[test, 'medv'])^2 )
```

```
## [1] 18.84709
```

与随机森林方法结果相近。

如果提高学习速度：

```
bst2 <- gbm(medv ~ ., data=d[train,],  distribution='gaussian',  n.trees=5000,  interaction.depth=4,
yhat <- predict(bst2, newdata=d[test,], n.trees=5000)
mean( (yhat - d[test, 'medv'])^2 )
```

```
## [1] 18.33455
```

均方误差有改善。

38.6 支持向量机方法

支持向量机是 1990 年代有计算机科学家发明的一种有监督学习方法，使用范围较广，预测精度较高。

支持向量机利用了 Hilbert 空间的方法将线性问题扩展为非线性问题。线性的支持向量判别法，可以通过 $\mathbb{R}^p$ 的内积将线性的判别函数转化为如下的表示：

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$$

其中 $\beta_0, \alpha_1, \dots, \alpha_n$ 是待定参数。为了估计参数，不需要用到各 x_i 的具体值，而只需要其两两的内积值，而且在判别函数中只有支持向量对应的 α_i 才非零，记 $\mathcal{S}$ 为支持向量点集，则线性判别函数为

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i \langle x, x_i \rangle$$

支持向量机方法将 $\mathbb{R}^p$ 中的内积推广为如下的核函数值：

$$K(x, x')$$

核函数 $K(x, x')$, $x, x' \in \mathbb{R}^p$ 是度量两个观测点 x, x' 的相似程度的函数。比如，取

$$K(x, x') = \sum_{j=1}^{p} x_j x'_j$$

就又回到了线性的支持向量判别法。

核有多种取法。例如，取

$$K(x,x') = \left\{1 + \sum_{j=1}^{p} x_j x'_j\right\}^d$$

其中 $d > 1$ 为正整数，称为**多项式核**，则结果是多项式边界的判别法，本质上是对线性的支持向量方法添加了高次项和交叉项。

利用核代替内积后，判别法的判别函数变成

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} K(x, x_i)$$

另一种常用的核是**径向核** (radial kernel)，定义为

$$K(x,x') = \exp\left\{-\gamma \sum_{j=1}^{p} (x_j - x'_j)^2\right\}$$

γ 为正常数。当 x 和 x' 分别落在以原点为中心的两个超球面上时，其核函数值不变。

使用径向核时，判别函数为

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \exp\left\{-\gamma \sum_{j=1}^{p} (x_j - x_{ij})^2\right\}$$

对一个待判别的观测 x^*，如果 x^* 距离训练观测点 x_i 较远，则 $K(x^*, x_i)$ 的值很小，x_i 对 x^* 的判别基本不起作用。这样的性质使得径向核方法具有很强的局部性，只有离 x^* 很近的点才对其判别起作用。

为什么采用核函数计算观测两两的 $\binom{n}{2}$ 个核函数值，而不是直接增加非线性项？原因是计算这些核函数值计算量是确定的，而增加许多非线性项，则可能有很大的计算量，而且某些核如径向核对应的自变量空间维数是无穷维的，不能通过添加维度的办法解决。

支持向量机的理论基于再生核希尔伯特空间 (RKHS)，可参见 (Trevor Hastie, 2009) 节 5.8 和节 12.3.3。

38.6.1 支持向量机用于 Heart 数据

考虑心脏病数据 Heart 的判别。共 297 个观测，随机选取其中 207 个作为训练集，90 个作为测试集。

```
set.seed(1)
Heart <- read.csv(
  "data/Heart.csv", header=TRUE, row.names=1,
  stringsAsFactors=TRUE)
d <- na.omit(Heart)
train <- sample(nrow(d), size=207)
test <- -train
d[["AHD"]] <- factor(d[["AHD"]], levels=c("No", "Yes"))
```

定义一个错判率函数：

```
classifier.error <- function(truth, pred){
  tab1 <- table(truth, pred)
  err <- 1 - sum(diag(tab1))/sum(c(tab1))
  err
}
```

38.6.1.1 线性的 SVM

支持向量判别法就是 SVM 取多项式核，阶数 $d = 1$ 的情形。需要一个调节参数 `cost`，`cost` 越大，分隔边界越窄，过度拟合危险越大。

先随便取调节参数 `cost=1` 试验支持向量判别法：

```
res.svc <- svm(AHD ~ ., data=d[train,], kernel="linear", cost=1, scale=TRUE)
fit.svc <- predict(res.svc)
summary(res.svc)
```

```
##
## Call:
## svm(formula = AHD ~ ., data = d[train, ], kernel = "linear", cost = 1, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  79
##
##  ( 38 41 )
##
##
## Number of Classes:  2
##
## Levels:
##  No Yes
```

计算拟合结果并计算错判率：

```r
tab1 <- table(truth=d[train,"AHD"], fitted=fit.svc); tab1
```

```
##      fitted
## truth  No Yes
##   No  105   9
##   Yes  18  75
```

```r
cat("SVC 错判率: ", round((tab1[1,2] + tab1[2,1])/ sum(c(tab1)), 2), "\n")
```

```
## SVC错判率:  0.13
```

e1071 函数提供了 `tune()` 函数，可以在训练集上用十折交叉验证选择较好的调节参数。

```r
set.seed(101)
res.tune <- tune(svm, AHD ~ ., data=d[train,], kernel="linear", scale=TRUE,
                 ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100, 1000)))
summary(res.tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.1542857
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.4450000 0.08509809
## 2 1e-02 0.1695238 0.07062868
## 3 1e-01 0.1542857 0.07006458
## 4 1e+00 0.1590476 0.07793796
## 5 5e+00 0.1590476 0.08709789
## 6 1e+01 0.1590476 0.08709789
## 7 1e+02 0.1590476 0.08709789
## 8 1e+03 0.1590476 0.08709789
```

找到的最优调节参数为 0.1，可以用 `res.tune$best.model` 获得对应于最优调节参数的模型：

```
summary(res.tune$best.model)
```

```
##
## Call:
## best.tune(method = svm, train.x = AHD ~ ., data = d[train, ], ranges = list(cost = c(0.001, 0.01,
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  90
##
##  ( 44 46 )
##
##
## Number of Classes:  2
##
## Levels:
##  No Yes
```

在测试集上测试：

```
pred.svc <- predict(res.tune$best.model, newdata=d[test,])
tab1 <- table(truth=d[test,"AHD"], predict=pred.svc); tab1
```

```
##      predict
## truth No Yes
##   No  43   3
##   Yes 11  33
```

```
cat("SVC 错判率: ", round((tab1[1,2] + tab1[2,1])/ sum(c(tab1)), 2), "\n")
```

```
## SVC错判率:  0.16
```

38.6.1.2 多项式核 SVM

```
res.svm1 <- svm(AHD ~ ., data=d[train,], kernel="polynomial",
                order=2, cost=0.1, scale=TRUE)
fit.svm1 <- predict(res.svm1)
summary(res.svm1)
```

```
##
## Call:
## svm(formula = AHD ~ ., data = d[train, ], kernel = "polynomial", order = 2, cost = 0.1, scale = TR
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.1
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  187
##
##  ( 92 95 )
##
##
## Number of Classes:  2
##
## Levels:
##  No Yes
```

```
tab1 <- table(truth=d[train,"AHD"], fitted=fit.svm1); tab1
```

```
##      fitted
## truth  No Yes
##   No  114   0
##   Yes  82  11
```

```
cat("2 阶多项式核 SVM 错判率: ", round((tab1[1,2] + tab1[2,1])/ sum(c(tab1)), 2), "\n")
```

```
## 2阶多项式核SVM错判率:  0.4
```

尝试找到调节参数 cost 的最优值：

```
set.seed(101)
res.tune2 <- tune(svm, AHD ~ ., data=d[train,], kernel="polynomial",
                  order=2, scale=TRUE,
                  ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100, 1000)))
summary(res.tune2)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     5
##
## - best performance: 0.2130952
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.4500000 0.08022549
## 2 1e-02 0.4500000 0.08022549
## 3 1e-01 0.4111905 0.09215957
## 4 1e+00 0.2185714 0.09094005
## 5 5e+00 0.2130952 0.09790737
## 6 1e+01 0.2180952 0.07948562
## 7 1e+02 0.2807143 0.09539966
## 8 1e+03 0.2807143 0.09539966
```

```
fit.svm2 <- predict(res.tune2$best.model)
tab1 <- table(truth=d[train,"AHD"], fitted=fit.svm2); tab1
```

```
##      fitted
## truth  No Yes
##   No  111   3
##   Yes   4  89
```

```
cat("2 阶多项式核最优参数 SVM 错判率: ", round((tab1[1,2] + tab1[2,1])/ sum(c(tab1)), 2), "\n")
```

```
## 2阶多项式核最优参数SVM错判率:  0.03
```

看这个最优调节参数的模型在测试集上的表现:

```
pred.svm2 <- predict(res.tune2$best.model, d[test,])
tab1 <- table(truth=d[test,"AHD"], predict=pred.svm2); tab1
```

```
##      predict
## truth No Yes
##   No  43   3
##   Yes 10  34
```

```
cat("2 阶多项式核最优参数 SVM 测试集错判率: ", round((tab1[1,2] + tab1[2,1])/ sum(c(tab1)), 2), "\n")
```

```
## 2阶多项式核最优参数SVM测试集错判率:  0.14
```

在测试集上的表现与线性方法相近。

38.6.1.3 径向核 SVM

径向核需要的参数为 γ 值。取参数 gamma=0.1。

```
res.svm3 <- svm(AHD ~ ., data=d[train,], kernel="radial",
                gamma=0.1, cost=0.1, scale=TRUE)
fit.svm3 <- predict(res.svm3)
summary(res.svm3)
```

```
##
## Call:
## svm(formula = AHD ~ ., data = d[train, ], kernel = "radial", gamma = 0.1, cost = 0.1, scale = TRUE
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.1
##
```

```
## Number of Support Vectors:  179
##
##  ( 89 90 )
##
##
## Number of Classes:  2
##
## Levels:
##  No Yes
```

```
tab1 <- table(truth=d[train,"AHD"], fitted=fit.svm3); tab1
```

```
##      fitted
## truth  No Yes
##   No  108   6
##   Yes  26  67
```

```
cat(" 径向核（gamma=0.1, cost=0.1）SVM 错判率: ", round((tab1[1,2] + tab1[2,1])/ sum(c(tab1)), 2), "\n
```

```
## 径向核（gamma=0.1, cost=0.1）SVM错判率:  0.15
```

选取最优 cost, gamma 调节参数:

```
set.seed(101)
res.tune4 <- tune(svm, AHD ~ ., data=d[train,], kernel="radial",
                  scale=TRUE,
                  ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100, 1000),
                              gamma=c(0.1, 0.01, 0.001)))
summary(res.tune4)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   100 0.001
##
## - best performance: 0.1492857
```

```
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-03 0.100 0.4500000 0.08022549
## 2  1e-02 0.100 0.4500000 0.08022549
## 3  1e-01 0.100 0.2235714 0.09912346
## 4  1e+00 0.100 0.1788095 0.08490543
## 5  5e+00 0.100 0.1835714 0.06267781
## 6  1e+01 0.100 0.1835714 0.07375788
## 7  1e+02 0.100 0.1933333 0.09294732
## 8  1e+03 0.100 0.1933333 0.09294732
## 9  1e-03 0.010 0.4500000 0.08022549
## 10 1e-02 0.010 0.4500000 0.08022549
## 11 1e-01 0.010 0.3147619 0.11998992
## 12 1e+00 0.010 0.1647619 0.06992960
## 13 5e+00 0.010 0.1547619 0.07819776
## 14 1e+01 0.010 0.1547619 0.08135598
## 15 1e+02 0.010 0.2126190 0.06443790
## 16 1e+03 0.010 0.2409524 0.08621108
## 17 1e-03 0.001 0.4500000 0.08022549
## 18 1e-02 0.001 0.4500000 0.08022549
## 19 1e-01 0.001 0.4500000 0.08022549
## 20 1e+00 0.001 0.2138095 0.11215945
## 21 5e+00 0.001 0.1695238 0.07062868
## 22 1e+01 0.001 0.1840476 0.08321647
## 23 1e+02 0.001 0.1492857 0.08228019
## 24 1e+03 0.001 0.1640476 0.07494392
```

```r
fit.svm4 <- predict(res.tune4$best.model)
tab1 <- table(truth=d[train,"AHD"], fitted=fit.svm4); tab1
```

```
##      fitted
## truth  No Yes
##   No  107   7
##   Yes  18  75
```

```r
cat(" 径向核最优参数 SVM 错判率: ", round((tab1[1,2] + tab1[2,1])/ sum(c(tab1)), 2), "\n")
```

```
## 径向核最优参数SVM错判率:  0.12
```

看这个最优调节参数的模型在测试集上的表现：

```
pred.svm4 <- predict(res.tune4$best.model, d[test,])
tab1 <- table(truth=d[test,"AHD"], predict=pred.svm2); tab1
```

```
##      predict
## truth No Yes
##   No  43   3
##   Yes 10  34
```

```
cat(" 径向核最优参数 SVM 测试集错判率: ", round((tab1[1,2] + tab1[2,1])/ sum(c(tab1)), 2), "\n")
```

```
## 径向核最优参数SVM测试集错判率:  0.14
```

与线性方法结果相近。

38.7 附录

38.7.1 Hitters 数据

```
knitr::kable(Hitters)
```

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns
-Andy Allanson	293	66	1	30	29	14	1	293	66	1	30
-Alan Ashby	315	81	7	24	38	39	14	3449	835	69	321
-Alvin Davis	479	130	18	66	72	76	3	1624	457	63	224
-Andre Dawson	496	141	20	65	78	37	11	5628	1575	225	828
-Andres Galarraga	321	87	10	39	42	30	2	396	101	12	48
-Alfredo Griffin	594	169	4	74	51	35	11	4408	1133	19	501
-Al Newman	185	37	1	23	8	21	2	214	42	1	30
-Argenis Salazar	298	73	0	24	24	7	3	509	108	0	41
-Andres Thomas	323	81	6	26	32	8	2	341	86	6	32
-Andre Thornton	401	92	17	49	66	65	13	5206	1332	253	784
-Alan Trammell	574	159	21	107	75	59	10	4631	1300	90	702
-Alex Trevino	202	53	4	31	26	27	9	1876	467	15	192
-Andy VanSlyke	418	113	13	48	61	47	4	1512	392	41	205
-Alan Wiggins	239	60	0	30	11	22	6	1941	510	4	309
-Bill Almon	196	43	7	29	27	30	13	3231	825	36	376
-Billy Beane	183	39	3	20	15	11	3	201	42	3	20
-Buddy Bell	568	158	20	89	75	73	15	8068	2273	177	1045
-Buddy Biancalana	190	46	2	24	8	15	5	479	102	5	65
-Bruce Bochte	407	104	6	57	43	65	12	5233	1478	100	643
-Bruce Bochy	127	32	8	16	22	14	8	727	180	24	67
-Barry Bonds	413	92	16	72	48	65	1	413	92	16	72
-Bobby Bonilla	426	109	3	55	43	62	1	426	109	3	55
-Bob Boone	22	10	1	4	2	1	6	84	26	2	9
-Bob Brenly	472	116	16	60	62	74	6	1924	489	67	242
-Bill Buckner	629	168	18	73	102	40	18	8424	2464	164	1008
-Brett Butler	587	163	4	92	51	70	6	2695	747	17	442
-Bob Dernier	324	73	4	32	18	22	7	1931	491	13	291
-Bo Diaz	474	129	10	50	56	40	10	2331	604	61	246
-Bill Doran	550	152	6	92	37	81	5	2308	633	32	349
-Brian Downing	513	137	20	90	95	90	14	5201	1382	166	763
-Bobby Grich	313	84	9	42	30	39	17	6890	1833	224	1033
-Billy Hatcher	419	108	6	55	36	22	3	591	149	8	80
-Bob Horner	517	141	27	70	87	52	9	3571	994	215	545
-Brook Jacoby	583	168	17	83	80	56	5	1646	452	44	219
-Bob Kearney	204	49	6	23	25	12	7	1309	308	27	126
-Bill Madlock	379	106	10	38	60	30	14	6207	1906	146	859
-Bobby Meacham	161	36	0	19	10	17	4	1053	244	3	156
-Bob Melvin	268	60	5	24	25	15	2	350	78	5	34
-Ben Oglivie	346	98	5	31	53	30	16	5913	1615	235	784
-Bip Roberts	241	61	1	34	12	14	1	241	61	1	34
-BillyJo Robidoux	181	41	1	15	21	33	2	232	50	4	20
-Bill Russell	216	54	0	21	18	15	18	7318	1926	46	796

38.7.2 Heart 数据

```
knitr::kable(Heart)
```

Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal
63	1	typical	145	233	1	2	150	0	2.3	3	0	fixed
67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3	normal
67	1	asymptomatic	120	229	0	2	129	1	2.6	2	2	reversabl
37	1	nonanginal	130	250	0	0	187	0	3.5	3	0	normal
41	0	nontypical	130	204	0	2	172	0	1.4	1	0	normal
56	1	nontypical	120	236	0	0	178	0	0.8	1	0	normal
62	0	asymptomatic	140	268	0	2	160	0	3.6	3	2	normal
57	0	asymptomatic	120	354	0	0	163	1	0.6	1	0	normal
63	1	asymptomatic	130	254	0	2	147	0	1.4	2	1	reversabl
53	1	asymptomatic	140	203	1	2	155	1	3.1	3	0	reversabl
57	1	asymptomatic	140	192	0	0	148	0	0.4	2	0	fixed
56	0	nontypical	140	294	0	2	153	0	1.3	2	0	normal
56	1	nonanginal	130	256	1	2	142	1	0.6	2	1	fixed
44	1	nontypical	120	263	0	0	173	0	0.0	1	0	reversabl
52	1	nonanginal	172	199	1	0	162	0	0.5	1	0	reversabl
57	1	nonanginal	150	168	0	0	174	0	1.6	1	0	normal
48	1	nontypical	110	229	0	0	168	0	1.0	3	0	reversabl
54	1	asymptomatic	140	239	0	0	160	0	1.2	1	0	normal
48	0	nonanginal	130	275	0	0	139	0	0.2	1	0	normal
49	1	nontypical	130	266	0	0	171	0	0.6	1	0	normal
64	1	typical	110	211	0	2	144	1	1.8	2	0	normal
58	0	typical	150	283	1	2	162	0	1.0	1	0	normal
58	1	nontypical	120	284	0	2	160	0	1.8	2	0	normal
58	1	nonanginal	132	224	0	2	173	0	3.2	1	2	reversabl
60	1	asymptomatic	130	206	0	2	132	1	2.4	2	2	reversabl
50	0	nonanginal	120	219	0	0	158	0	1.6	2	0	normal
58	0	nonanginal	120	340	0	0	172	0	0.0	1	0	normal
66	0	typical	150	226	0	0	114	0	2.6	3	0	normal
43	1	asymptomatic	150	247	0	0	171	0	1.5	1	0	normal
40	1	asymptomatic	110	167	0	2	114	1	2.0	2	0	reversabl
69	0	typical	140	239	0	0	151	0	1.8	1	2	normal
60	1	asymptomatic	117	230	1	0	160	1	1.4	1	2	reversabl
64	1	nonanginal	140	335	0	0	158	0	0.0	1	0	normal
59	1	asymptomatic	135	234	0	0	161	0	0.5	2	0	reversabl
44	1	nonanginal	130	233	0	0	179	1	0.4	1	0	normal
42	1	asymptomatic	140	226	0	0	178	0	0.0	1	0	normal
43	1	asymptomatic	120	177	0	2	120	1	2.5	2	0	reversabl
57	1	asymptomatic	150	276	0	2	112	1	0.6	2	1	fixed
55	1	asymptomatic	132	353	0	0	132	1	1.2	2	1	reversabl
61	1	nonanginal	150	243	1	0	137	1	1.0	2	0	normal
65	0	asymptomatic	150	225	0	2	114	0	1.0	2	3	reversabl
40	1	typical	140	199	0	0	178	1	1.4	1	0	reversabl

38.7.3 CarSeats 数据

```
knitr::kable(Carseats)
```

Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US
9.50	138	73	11	276	120	Bad	42	17	Yes	Yes
11.22	111	48	16	260	83	Good	65	10	Yes	Yes
10.06	113	35	10	269	80	Medium	59	12	Yes	Yes
7.40	117	100	4	466	97	Medium	55	14	Yes	Yes
4.15	141	64	3	340	128	Bad	38	13	Yes	No
10.81	124	113	13	501	72	Bad	78	16	No	Yes
6.63	115	105	0	45	108	Medium	71	15	Yes	No
11.85	136	81	15	425	120	Good	67	10	Yes	Yes
6.54	132	110	0	108	124	Medium	76	10	No	No
4.69	132	113	0	131	124	Medium	76	17	No	Yes
9.01	121	78	9	150	100	Bad	26	10	No	Yes
11.96	117	94	4	503	94	Good	50	13	Yes	Yes
3.98	122	35	2	393	136	Medium	62	18	Yes	No
10.96	115	28	11	29	86	Good	53	18	Yes	Yes
11.17	107	117	11	148	118	Good	52	18	Yes	Yes
8.71	149	95	5	400	144	Medium	76	18	No	No
7.58	118	32	0	284	110	Good	63	13	Yes	No
12.29	147	74	13	251	131	Good	52	10	Yes	Yes
13.91	110	110	0	408	68	Good	46	17	No	Yes
8.73	129	76	16	58	121	Medium	69	12	Yes	Yes
6.41	125	90	2	367	131	Medium	35	18	Yes	Yes
12.13	134	29	12	239	109	Good	62	18	No	Yes
5.08	128	46	6	497	138	Medium	42	13	Yes	No
5.87	121	31	0	292	109	Medium	79	10	Yes	No
10.14	145	119	16	294	113	Bad	42	12	Yes	Yes
14.90	139	32	0	176	82	Good	54	11	No	No
8.33	107	115	11	496	131	Good	50	11	No	Yes
5.27	98	118	0	19	107	Medium	64	17	Yes	No
2.99	103	74	0	359	97	Bad	55	11	Yes	Yes
7.81	104	99	15	226	102	Bad	58	17	Yes	Yes
13.55	125	94	0	447	89	Good	30	12	Yes	No
8.25	136	58	16	241	131	Medium	44	18	Yes	Yes
6.20	107	32	12	236	137	Good	64	10	No	Yes
8.77	114	38	13	317	128	Good	50	16	Yes	Yes
2.67	115	54	0	406	128	Medium	42	17	Yes	Yes
11.07	131	84	11	29	96	Medium	44	17	No	Yes
8.89	122	76	0	270	100	Good	60	18	No	No
4.95	121	41	5	412	110	Medium	54	10	Yes	Yes
6.59	109	73	0	454	102	Medium	65	15	Yes	No
3.24	130	60	0	144	138	Bad	38	10	No	No
2.07	119	98	0	18	126	Bad	73	17	No	No
7.96	157	53	0	403	124	Bad	58	16	Yes	No

38.7.4 Boston 数据

```
knitr::kable(Boston)
```

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0.00632	18.0	2.31	0	0.5380	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
0.02731	0.0	7.07	0	0.4690	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
0.02729	0.0	7.07	0	0.4690	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
0.03237	0.0	2.18	0	0.4580	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	0.0	2.18	0	0.4580	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2
0.02985	0.0	2.18	0	0.4580	6.430	58.7	6.0622	3	222	18.7	394.12	5.21	28.7
0.08829	12.5	7.87	0	0.5240	6.012	66.6	5.5605	5	311	15.2	395.60	12.43	22.9
0.14455	12.5	7.87	0	0.5240	6.172	96.1	5.9505	5	311	15.2	396.90	19.15	27.1
0.21124	12.5	7.87	0	0.5240	5.631	100.0	6.0821	5	311	15.2	386.63	29.93	16.5
0.17004	12.5	7.87	0	0.5240	6.004	85.9	6.5921	5	311	15.2	386.71	17.10	18.9
0.22489	12.5	7.87	0	0.5240	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	15.0
0.11747	12.5	7.87	0	0.5240	6.009	82.9	6.2267	5	311	15.2	396.90	13.27	18.9
0.09378	12.5	7.87	0	0.5240	5.889	39.0	5.4509	5	311	15.2	390.50	15.71	21.7
0.62976	0.0	8.14	0	0.5380	5.949	61.8	4.7075	4	307	21.0	396.90	8.26	20.4
0.63796	0.0	8.14	0	0.5380	6.096	84.5	4.4619	4	307	21.0	380.02	10.26	18.2
0.62739	0.0	8.14	0	0.5380	5.834	56.5	4.4986	4	307	21.0	395.62	8.47	19.9
1.05393	0.0	8.14	0	0.5380	5.935	29.3	4.4986	4	307	21.0	386.85	6.58	23.1
0.78420	0.0	8.14	0	0.5380	5.990	81.7	4.2579	4	307	21.0	386.75	14.67	17.5
0.80271	0.0	8.14	0	0.5380	5.456	36.6	3.7965	4	307	21.0	288.99	11.69	20.2
0.72580	0.0	8.14	0	0.5380	5.727	69.5	3.7965	4	307	21.0	390.95	11.28	18.2
1.25179	0.0	8.14	0	0.5380	5.570	98.1	3.7979	4	307	21.0	376.57	21.02	13.6
0.85204	0.0	8.14	0	0.5380	5.965	89.2	4.0123	4	307	21.0	392.53	13.83	19.6
1.23247	0.0	8.14	0	0.5380	6.142	91.7	3.9769	4	307	21.0	396.90	18.72	15.2
0.98843	0.0	8.14	0	0.5380	5.813	100.0	4.0952	4	307	21.0	394.54	19.88	14.5
0.75026	0.0	8.14	0	0.5380	5.924	94.1	4.3996	4	307	21.0	394.33	16.30	15.6
0.84054	0.0	8.14	0	0.5380	5.599	85.7	4.4546	4	307	21.0	303.42	16.51	13.9
0.67191	0.0	8.14	0	0.5380	5.813	90.3	4.6820	4	307	21.0	376.88	14.81	16.6
0.95577	0.0	8.14	0	0.5380	6.047	88.8	4.4534	4	307	21.0	306.38	17.28	14.8
0.77299	0.0	8.14	0	0.5380	6.495	94.4	4.4547	4	307	21.0	387.94	12.80	18.4
1.00245	0.0	8.14	0	0.5380	6.674	87.3	4.2390	4	307	21.0	380.23	11.98	21.0
1.13081	0.0	8.14	0	0.5380	5.713	94.1	4.2330	4	307	21.0	360.17	22.60	12.7
1.35472	0.0	8.14	0	0.5380	6.072	100.0	4.1750	4	307	21.0	376.73	13.04	14.5
1.38799	0.0	8.14	0	0.5380	5.950	82.0	3.9900	4	307	21.0	232.60	27.71	13.2
1.15172	0.0	8.14	0	0.5380	5.701	95.0	3.7872	4	307	21.0	358.77	18.35	13.1
1.61282	0.0	8.14	0	0.5380	6.096	96.9	3.7598	4	307	21.0	248.31	20.34	13.5
0.06417	0.0	5.96	0	0.4990	5.933	68.2	3.3603	5	279	19.2	396.90	9.68	18.9
0.09744	0.0	5.96	0	0.4990	5.841	61.4	3.3779	5	279	19.2	377.56	11.41	20.0
0.08014	0.0	5.96	0	0.4990	5.850	41.5	3.9342	5	279	19.2	396.90	8.77	21.0
0.17505	0.0	5.96	0	0.4990	5.966	30.2	3.8473	5	279	19.2	393.43	10.13	24.7
0.02763	75.0	2.95	0	0.4280	6.595	21.8	5.4011	3	252	18.3	395.63	4.32	30.8
0.03359	75.0	2.95	0	0.4280	7.024	15.8	5.4011	3	252	18.3	395.62	1.98	34.9
0.12744	0.0	6.91	0	0.4480	6.770	2.9	5.7209	3	233	17.9	385.41	4.84	26.6

Chapter 39

R 时间序列分析

这里仅仅给出了部分常用的时间序列函数的用法，更详细的说明参见作者的金融时间序列分析讲义。

39.1 基本概念

设 $\mathbb{Z}$ 为整数集，$\{X_t, t \in \mathbb{Z}\}$ 中 X_t 是随机变量，称 $\{X_t\}$ 为时间序列。如果 $\{X_t\}$ 的有限维联合分布不随时间推移而变化，称 $\{X_t\}$ 为**严平稳**时间序列。如果 $\{X_t\}$ 二阶矩有限，期望和方差不随时间而变化，两个时间点之间的协方差只依赖于时间距离而不依赖于具体时间，则称 $\{X_t\}$ 为 **(宽) 平稳**时间序列。

如果 X_t 是随机向量，称 $\{X_t\}$ 为多元（或多维）时间序列。

对宽平稳列 $\{X_t\}$，定义其自协方差函数 (ACVF) 为

$$\gamma_k = \mathrm{Cov}(X_{t+k}, X_t),\ k \in \mathbb{Z}.$$

定义其自相关函数 (ACF) 为

$$\rho_k = \mathrm{corr}(X_{t+k}, X_t) = \frac{\gamma_k}{\gamma_0},\ k \in \mathbb{Z}.$$

偏自相关函数 (PACF) 定义请参考时间序列分析的教材。

如果 $\sum_k |\gamma_k| < \infty$，则平稳列 $\{X_t\}$ 有谱密度 $f(\lambda), \lambda \in [-\pi, \pi]$，$f(\lambda)$ 是非负可积偶函数，使得

$$\begin{aligned}\gamma_k =& \int_{-\pi}^{\pi} e^{ik\lambda} f(\lambda)\, d\lambda,\ k \in \mathbb{Z},\\ f(\lambda) =& \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} \gamma_k e^{-ik\lambda},\ \lambda \in [-\pi, \pi].\end{aligned}$$

若平稳列 $\{\varepsilon_t\}$ 的自协方差函数满足 $\gamma_0 = \sigma^2, \gamma_k = 0, k \neq 0$，称 $\{\varepsilon_t\}$ 为白噪声列。如果均值为零，称为零均值白噪声列，记为 WN(0, σ^2)。

39.2 时间序列数据类型

时间序列的数据可以保存在 R 的向量中，或者保存在 R 的数据框的一列或几列中，对应的时间单独保存或者保存在同一数据框中。

也有一些专门的时间序列数据类型，将时间序列的观测数据与对应的时间同时保存在一个专用的数据结构中。

39.2.1 ts 类型

R 中最基本的时间序列类型是 ts 类型，可以保存一元或者多元时间序列数据，其中的时间必须是等间隔的，比如年数据、月数据、季度数据、日数据，不能在中间有缺失的日期。生成方法如

```
ts(x, start=c(2001, 1), frequency=12)
```

其中 x 是向量或者矩阵，取矩阵值时矩阵的每一列是一个多元时间序列的一个分量。frequency 对月度数据是 12，对季度数据是 4，对年度数据可以缺省（值为 1）。start=c(2001,1) 表示序列开始时间是 2001 年 1 月。如果是年度数据，用如 ts(x, start=2001) 即可。

例如：用函数 ts 把一个向量转换为时间序列。如

```
yd <- ts(
  c(4, 8, 7, 7, 3, 1, 8, 9, 8, 6, 3, 5,
    5, 8, 2, 5, 9, 2, 5, 2, 3, 2, 2, 4),
  frequency=1, start=2001); yd
```

```
## Time Series:
## Start = 2001
## End = 2024
## Frequency = 1
##  [1] 4 8 7 7 3 1 8 9 8 6 3 5 5 8 2 5 9 2 5 2 3 2 2 4
```

```
ym <- ts(
  c(9, 6, 3, 5, 4, 8, 2, 5, 8, 4, 6, 3,
    2, 2, 6, 4, 1, 4, 2, 1, 8, 9, 4, 6),
  frequency=12, start=c(2001,1)); ym
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 2001   9   6   3   5   4   8   2   5   8   4   6   3
## 2002   2   2   6   4   1   4   2   1   8   9   4   6
```

其中 yd 是年数据，从 2001 到 2024 年；ym 是月度数据从 2001 年 1 月到 2002 年 12 月。

对于多元时间序列，可以用 ts 函数把一个矩阵转换为 ts 类型的多元时间序列对象。用 ts.intersect 函数和 ts.union 函数可以把两个或多个时间序列合并为多元时间序列，时间段取交集或者取并集。

为了使用序列数据进行计算、绘图等，可以用 as.vector 把时间序列的数据转换成普通向量。如：

```
as.vector(ym)
```

```
##  [1] 9 6 3 5 4 8 2 5 8 4 6 3 2 2 6 4 1 4 2 1 8 9 4 6
```

```
class(as.vector(ym))
```

```
## [1] "numeric"
```

在 R 中已安装的时间列示例数据有美国泛美航空公司 1949-1960 的国际航班订票数的月度数据（单位：千人），12 年 144 个月。用

```
data(AirPassengers)
```

调入名字空间。类型和属性如下：

```
data(AirPassengers)
attributes(AirPassengers)
```

```
## $tsp
## [1] 1949.000 1960.917   12.000
##
## $class
## [1] "ts"
```

这里时间以年为单位，数据类属为 ts，两个数据值之间的时间间隔为 1/12。

AirPassengers 的具体数值：

```
AirPassengers
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1949 112 118 132 129 121 135 148 148 136 119 104 118
## 1950 115 126 141 135 125 149 170 170 158 133 114 140
## 1951 145 150 178 163 172 178 199 199 184 162 146 166
## 1952 171 180 193 181 183 218 230 242 209 191 172 194
## 1953 196 196 236 235 229 243 264 272 237 211 180 201
```

```
## 1954 204 188 235 227 234 264 302 293 259 229 203 229
## 1955 242 233 267 269 270 315 364 347 312 274 237 278
## 1956 284 277 317 313 318 374 413 405 355 306 271 306
## 1957 315 301 356 348 355 422 465 467 404 347 305 336
## 1958 340 318 362 348 363 435 491 505 404 359 310 337
## 1959 360 342 406 396 420 472 548 559 463 407 362 405
## 1960 417 391 419 461 472 535 622 606 508 461 390 432
```

用 start() 求时间序列的开始点，end() 求时间序列的结束点，frequency() 求采样频率。如

```
start(AirPassengers)
```

```
## [1] 1949    1
```

```
end(AirPassengers)
```

```
## [1] 1960   12
```

```
frequency(AirPassengers)
```

```
## [1] 12
```

aggregate() 函数可以把月度数据加总成年数据。如

```
AP.year <- aggregate(AirPassengers); AP.year
```

```
## Time Series:
## Start = 1949
## End = 1960
## Frequency = 1
##  [1] 1520 1676 2042 2364 2700 2867 3408 3939 4421 4572 5140 5714
```

如果加参数 FUN=mean 可以取均值。

time() 函数对 ts 类型数据返回序列中的每个时间点的时间，结果是一个和原来时间序列形状相同的时间序列。cycle() 函数对月度数据返回序列每个时间点所在的月份, 结果是和原序列时间点相同的一个时间序列。如

```
cy.AP <- cycle(AirPassengers); cy.AP
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1949   1   2   3   4   5   6   7   8   9  10  11  12
## 1950   1   2   3   4   5   6   7   8   9  10  11  12
## 1951   1   2   3   4   5   6   7   8   9  10  11  12
## 1952   1   2   3   4   5   6   7   8   9  10  11  12
## 1953   1   2   3   4   5   6   7   8   9  10  11  12
## 1954   1   2   3   4   5   6   7   8   9  10  11  12
## 1955   1   2   3   4   5   6   7   8   9  10  11  12
## 1956   1   2   3   4   5   6   7   8   9  10  11  12
## 1957   1   2   3   4   5   6   7   8   9  10  11  12
## 1958   1   2   3   4   5   6   7   8   9  10  11  12
## 1959   1   2   3   4   5   6   7   8   9  10  11  12
## 1960   1   2   3   4   5   6   7   8   9  10  11  12
```

window() 函数取出时间序列的一段，如果指定 frequency=TRUE 还可以仅取出某个月（季度）。如

```
AP.Jan <- window(AirPassengers, start=c(1949,1),
                 frequency=TRUE); AP.Jan
```

```
## Time Series:
## Start = 1949
## End = 1960
## Frequency = 1
##  [1] 112 115 145 171 196 204 242 284 315 340 360 417
```

ts.union 可以形成多元时间序列。

39.2.2 xts 类型与常用函数

xts 也是一种时间序列数据类型，既可以保存等间隔时间序列数据，也可以保存不等间隔的时间序列数据，并且 xts 类型的数据访问功能更为方便。读入方法例如

```
xts(x, date)
```

其中 x 是向量、矩阵或数据框，date 是日期或者日期时间。x 取矩阵或者数据框时每列是一个时间序列。如

```
library(xts)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
xts.1 <- xts(
  c(5, 5, 4, 6, 4, 3, 3, 3, 4, 5, 5, 4),
  make_date(2018, 1, 1) + ddays(0:11)); xts.1
```

```
##            [,1]
## 2018-01-01    5
## 2018-01-02    5
## 2018-01-03    4
## 2018-01-04    6
## 2018-01-05    4
## 2018-01-06    3
## 2018-01-07    3
## 2018-01-08    3
## 2018-01-09    4
## 2018-01-10    5
## 2018-01-11    5
## 2018-01-12    4
```

如果数据框中一列是日期时间，其它列是时间序列数据，就可以用 `xts()` 转化成 xts 类型。

其它时间序列类型可以用 `as.xts()` 转换成 xts 类型，如

```
as.xts(ym)
```

```
##            [,1]
## 1月 2001      9
## 2月 2001      6
## 3月 2001      3
## 4月 2001      5
## 5月 2001      4
## 6月 2001      8
## 7月 2001      2
## 8月 2001      5
## 9月 2001      8
## 10月 2001     4
## 11月 2001     6
## 12月 2001     3
## 1月 2002      2
## 2月 2002      2
## 3月 2002      6
## 4月 2002      4
## 5月 2002      1
## 6月 2002      4
## 7月 2002      2
## 8月 2002      1
## 9月 2002      8
## 10月 2002     9
## 11月 2002     4
## 12月 2002     6
```

有了 xts 类型的变量 `x` 后，可以用 `coredata(x)` 返回 `x` 的不包含时间的纯数据；用 `index(x)` 返回 `x` 的时间标签。这两个函数很有用。

xts 类型支持强大的子集提取功能。若 `x` 是 xts 类型，则 `x[i]` 可以取出行子集，`x[i,j]` 可以取出行、列子集，其中 `i` 可以是整数下标，还可以是包含日期时间的字符串向量，日期时间类型的向量等。日期时间字符串的格式为 `CCYY-MM-DD HH:MM:SS`，而且可以省略后面的一部分，其含义是取出前面部分能匹配的所有时间点，这种设计使得取出某年、某月的数据变得十分方便。如

```
data(AirPassengers)
xts.ap <- as.xts(AirPassengers)
xts.ap["1949"]
```

```
##           [,1]
## 1月 1949   112
## 2月 1949   118
```

```
## 3月 1949   132
## 4月 1949   129
## 5月 1949   121
## 6月 1949   135
## 7月 1949   148
## 8月 1949   148
## 9月 1949   136
## 10月 1949  119
## 11月 1949  104
## 12月 1949  118
```

又比如，设 `x` 是某个股票的每分钟的数据，则 `x["2018-01-18 08"]` 取出 2018-1-18 8:00-9:00 之间的所有数据。

也可以用`"from/to"` 的格式指定一个日期时间范围，而且也不需要开始点和结束点恰好有数据，`from` 和 `to` 的时间精度也不需要与数据的实际采样频率相同。省略 `from` 则从头开始，省略 `to` 则直到末尾。如

```
xts.1["2018-01-10/2018-01-14"]
```

```
##            [,1]
## 2018-01-10    5
## 2018-01-11    5
## 2018-01-12    4
```

`first(x, n)` 和 `last(x, n)` 类似于 `head(x, n)` 和 `last(x, n)`，但是对 xts 对象 `x`，`n` 除了可以取正整数值以外，还允许用字符串指定时间长度，允许的单位包括 secs, seconds, mins, minutes, hours, days, weeks, months, quarters, years。比如，取出开头的三个月：

```
first(xts.ap, "3 months")
```

```
##          [,1]
## 1月 1949  112
## 2月 1949  118
## 3月 1949  132
```

字符串中取负值时表示扣除，如扣除开始的 3 个月：

```
start(xts.ap)
```

```
## [1] "1月 1949"
```

```
start(first(xts.ap, "-3 months"))
```

```
## [1] "4月 1949"
```

取出的单位可以是比实际采样频率更大的时间单位，比如，分钟数据可以用 `last(x, "1 day")` 取出最后一天的的所有数据。但是，取出的单位不能比实际采样频率更小，比如，日数据不能用小时单位取出。

xts 的时间下标也可以是一个日期或者日期时间类型的向量。支持的类型包括 Date, POSIXct, chron, yearmon, yearqtr, timeDate。可用 `index(x)` 函数来读取或者修改 xts 类型的时间下标。用 `coredata(x)` 取出不包含日期时间线信息的实际数据。

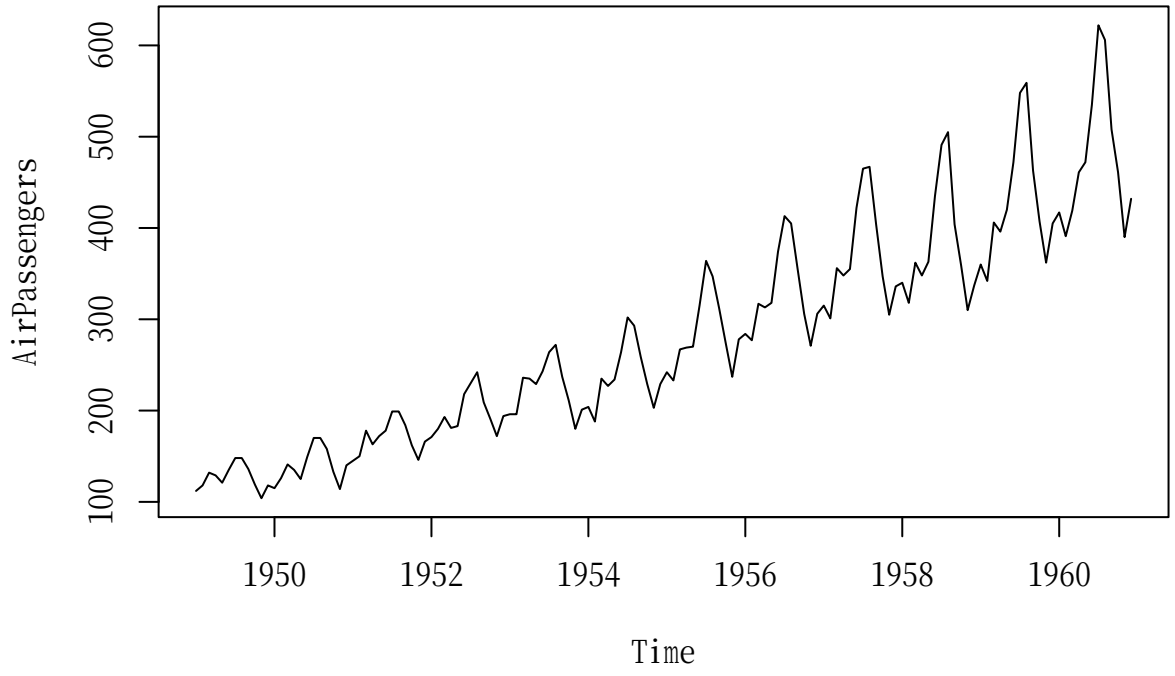
可以用 `to.period(x, period)` 将其 xts 类型的时间序列降频成 `period` 指定的采样频率，如 `to.period(x, period="days")`。

39.3 基本分析与作图

对 ts 和 xts 类型，仍可用 `length()`, `mean()`, `sd()` 等函数计算基本的统计量。

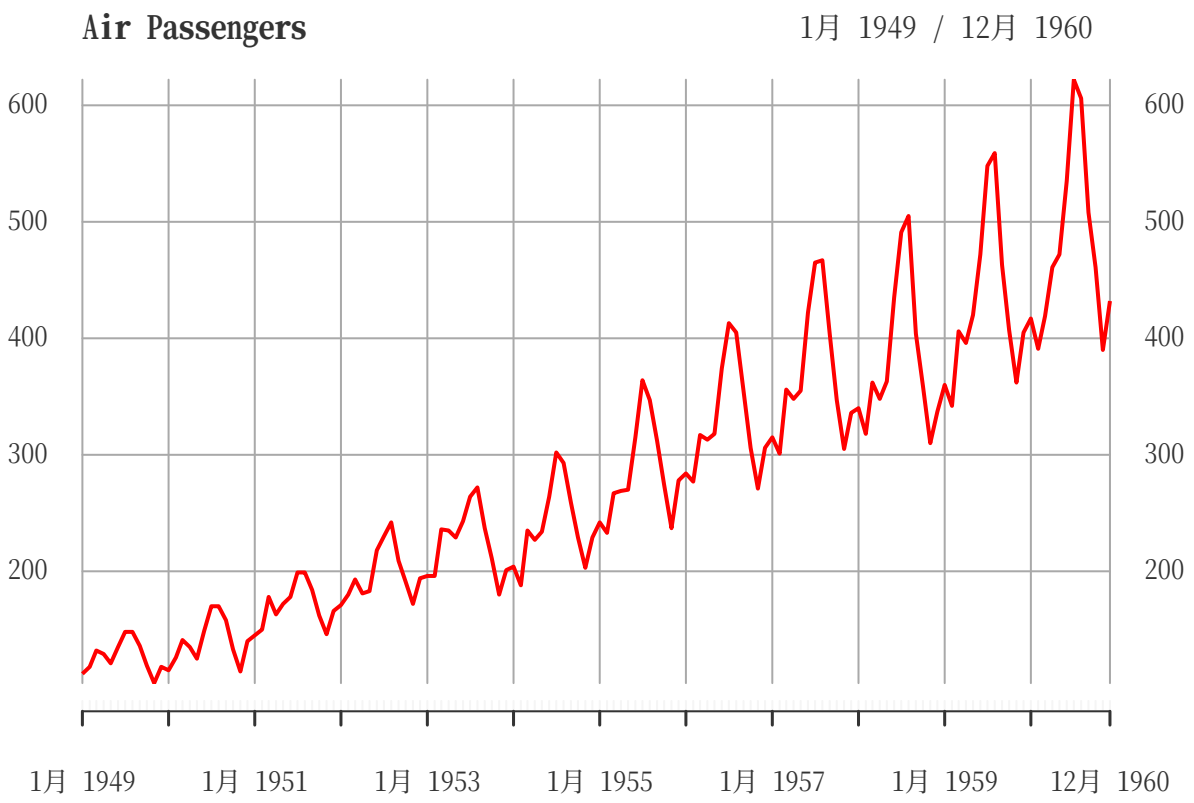
对 ts 类型，`plot()` 函数作曲线图，如：

```
data(AirPassengers)
plot(AirPassengers)
```



对 xts 类型，`plot()` 函数作时间序列曲线图，对多元时间序列，可以选择每个分量单独在一个坐标系中作图也可以重叠在一个坐标系中 (用 `multi.panel = TRUE` 选项)。如

```
xts.ap <- as.xts(AirPassengers)
plot(xts.ap, main="Air Passengers",
     major.ticks="years", minor.ticks=NULL,
     grid.ticks.on="years",
     col="red")
```



对 ts 类型,可以用 `as.vector(x)` 提取不包含日期时间信息的数据,结果是向量;对 xts 类型,可以用 `coredata(x)` 提取不包含日期时间信息的数据，结果不论一元还是多元时间序列都是矩阵。取出数据后，用通常的统计估计、图形方法可以进行一些探索性分析，如直方图 (`hist()`)、盒形图 (`boxplot()`)、QQ 图 (`qqnorm()` 和 `qqline()`)、散点图 (`plot()`) 等。

`stats::lag()` 可以计算滞后序列，对 ts 类型输入，`lag()` 的作用是序列数值不变，但是时间标签增加一个单位或者用 `k=` 指定的间隔。所以，为了得到通常理解的滞后 1 序列，应该使用 `stats::lag(x, k=-1)`，如

```
stats::lag(ts(1:5, start=2001), k=-1)
```

```
## Time Series:
## Start = 2002
## End = 2006
## Frequency = 1
## [1] 1 2 3 4 5
```

`diff(x)` 计算一阶差分，`diff(x, lag, differences)` 计算滞后为 `lag` 的阶数位 `differences` 的差分。

`acf` 和 `pacf` 函数可以作自相关、自协方差和偏自相关函数图，并可返回这些函数的估计值。参见39.4.2.1。

`filter` 函数可以计算递推的或卷积的滤波。

`arima` 可以拟合 ARIMA 模型。`arima.sim` 可以模拟生成 ARIMA 模型的数据。

39.4 ARIMA 建模和模拟

39.4.1 模型公式

AR(p) 模型（零均值）：

$$X_t = a_1 X_{t-1} + a_2 X_{t-2} + \cdots + a_p X_{t-p} + \varepsilon_t,\ \{\varepsilon_t\} \sim \mathrm{WN}(0, \sigma^2),$$

参数为 $a_1, \ldots, a_p, \sigma^2$，满足

$$A(z) = 1 - a_1 z - \cdots - a_p z^p \neq 0,\ \forall |z| \leq 1,\ z \in \mathbb{C},$$

$\mathbb{C}$ 为复数域。

可逆 MA(q) 模型（零均值）：

$$X_t = \varepsilon_t + b_1 \varepsilon_{t-1} + \cdots + b_1 \varepsilon_{t-q},\ \{\varepsilon_t\} \sim \mathrm{WN}(0, \sigma^2),$$

满足

$$B(z) = 1 + b_1 z + \cdots + b_q z^q \neq 0,\ \forall |z| \leq 1,\ z \in \mathbb{C}.$$

零均值可逆 ARMA(p, q) 模型：

$$X_t = a_1 X_{t-1} + a_2 X_{t-2} + \cdots + a_p X_{t-p} + \varepsilon_t + b_1 \varepsilon_{t-1} + \cdots + b_1 \varepsilon_{t-q},\ \{\varepsilon_t\} \sim \mathrm{WN}(0, \sigma^2).$$

记一阶差分为 $\Delta X_t = X_t - X_{t-1}$，$d$ 阶差分为 $\Delta^d X_t = \Delta(\Delta^{d-1} X_t)$，事实上

$$\Delta^d X_t = \sum_{j=0}^{d} C_d^j (-1)^j X_{t-j}.$$

其中 C_d^j 是 k 取 j 的组合个数。

ARIMA(p, d, q) 模型：如果 $\{\Delta^d X_t\}$ 满足零均值可逆 ARMA(p, q) 模型，就称 $\{X_t\}$ 满足 ARIMA(p, d, q) 模型。

39.4.2 模拟

用 `arima.sim()` 函数可以模拟生成 ARIMA 模型的数据，也可以用来模拟 AR、MA、ARMA。`arima.sim()` 输出 ts 类型的时间序列，并且在模型有一阶单位根时会比要求的个数多输出一个值，第一个值为初值 0。

39.4.2.1 AR(4) 模拟例子

考虑如下的 AR(4) 模型：

$$X_t = -0.9 X_{t-1} - 1.4 X_{t-2} - 0.7 X_{t-3} - 0.6 X_{t-4} \tag{39.1}$$

$$+ \varepsilon_t + 0.5 \varepsilon_{t-1} - 0.4 \varepsilon_{t-2}, \quad t \in \mathbb{Z}, \tag{39.2}$$

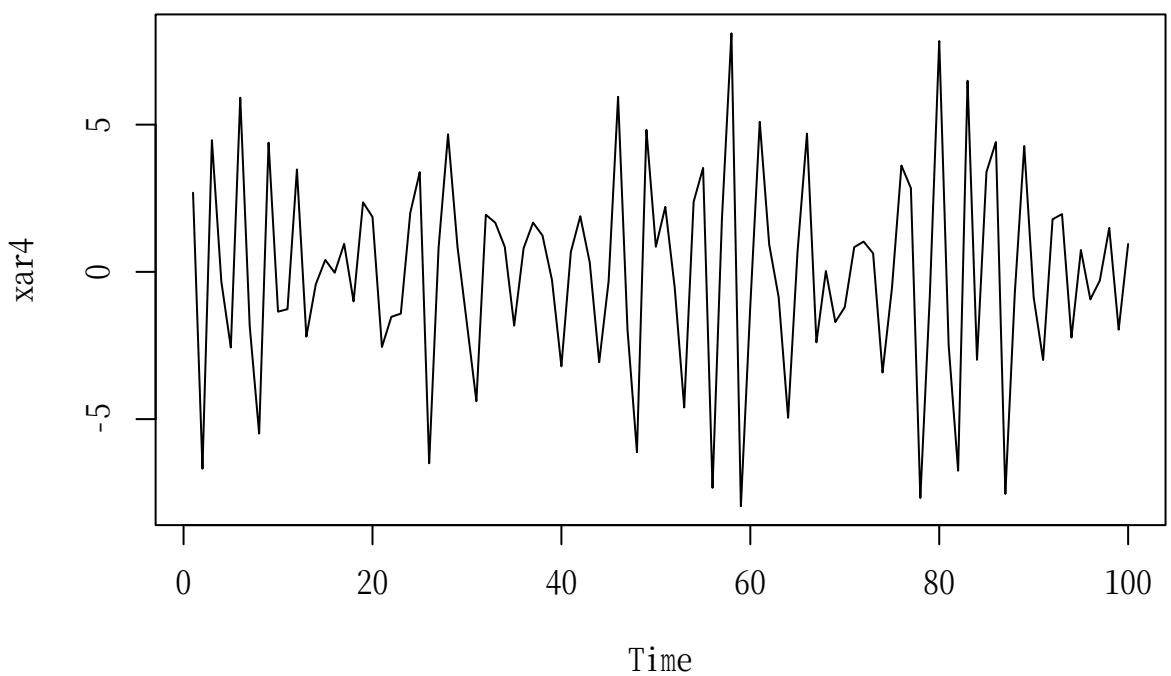
其中 $\{\varepsilon_t\}$ 为 WN(0, 4)。

用如下程序模拟生成长度为 $N = 100$ 的样本：

```
set.seed(101)
xar4 <- arima.sim(
  model=list(ar=c(-0.9, -1.4, -0.7, -0.6)),
  n = 100,
  sd = sqrt(4) )
```

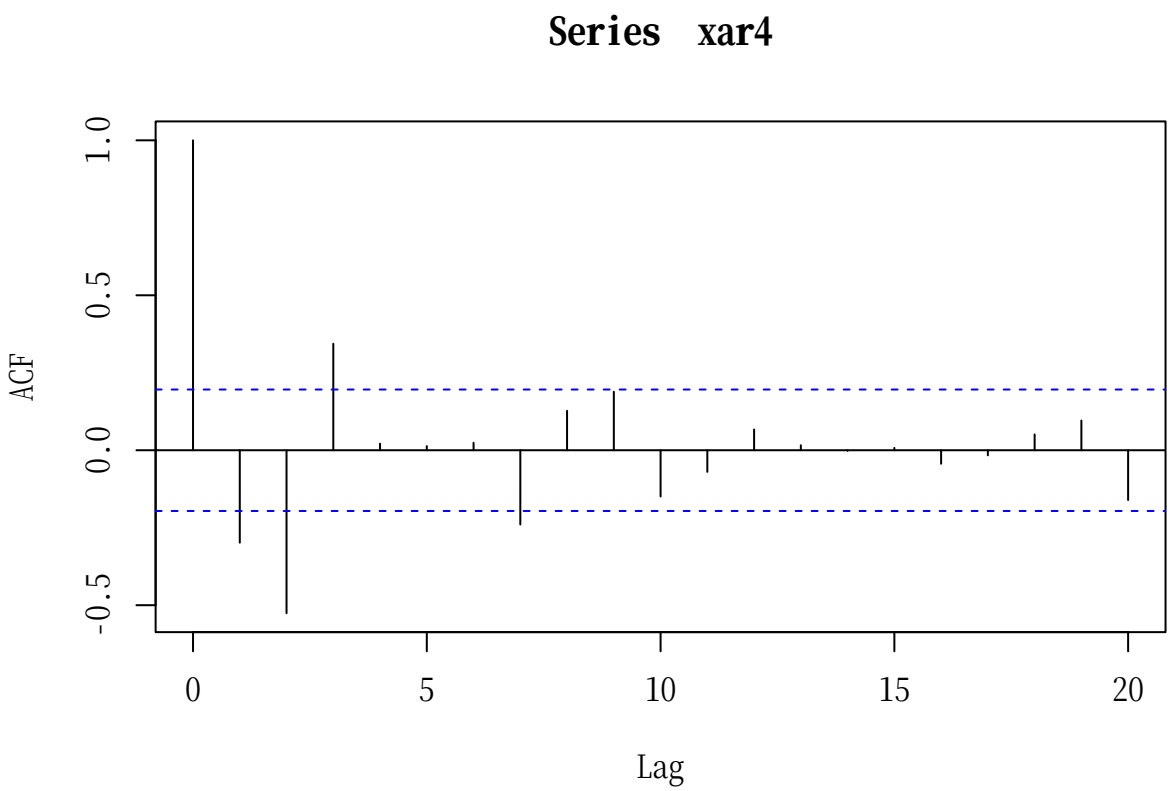
作时间序列图：

```
plot(xar4)
```



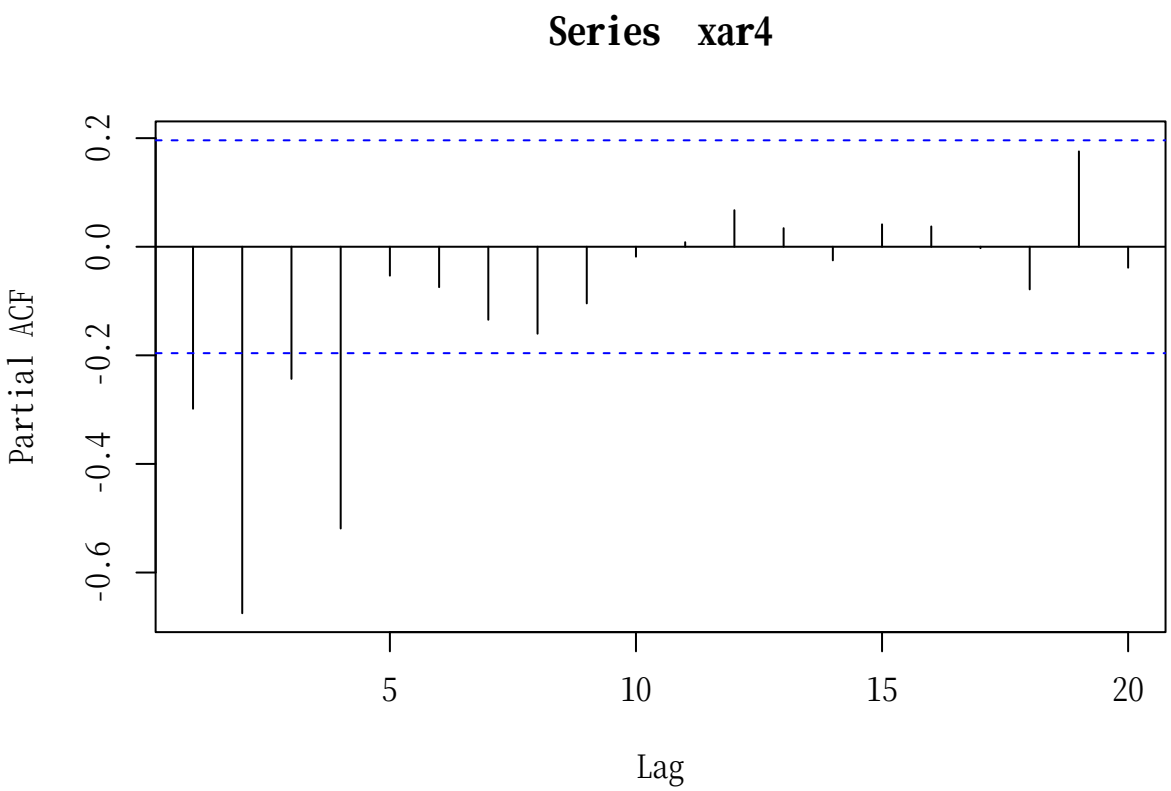
作 ACF 图：

```
acf(xar4)
```



作 PACF 图（偏自相关函数图）：

```
pacf(xar4)
```



acf(x) 加 type 选项和 plot=FALSE 选项可以返回估计的自协方差函数、自相关函数、偏自相关函数。如：

```
## 前 10 个自相关函数值:
round(c( acf(xar4, lag.max=10, type="correlation", plot=FALSE)$acf ), 4)
```

```
##  [1]  1.0000 -0.2984 -0.5258  0.3435  0.0213  0.0134  0.0241 -0.2397  0.1273
## [10]  0.1889 -0.1493
```

```
## 前 10 个自协方差函数值:
round(c( acf(xar4, lag.max=10, type="covariance", plot=FALSE)$acf ), 4)
```

```
##  [1] 11.6502 -3.4762 -6.1260  4.0017  0.2478  0.1566  0.2809 -2.7920  1.4831
## [10]  2.2008 -1.7390
```

```
## 前 9 个偏自相关函数值:
round(c( acf(xar4, lag.max=9, type="partial", plot=FALSE)$acf ), 4)
```

```
## [1] -0.2984 -0.6750 -0.2434 -0.5189 -0.0533 -0.0744 -0.1345 -0.1602 -0.1043
```

注意自相关函数、自协方差函数都是从滞后 0 开始（即 ρ_0, γ_0），而偏自相关函数从滞后 1 开始。

可以用 `polyroot()` 计算多项式的所有复根，输入是从零次项到最高次项的系数。可以用 `Mod()` 或 `abs()` 求复数模，如：

```
a <- c(-0.9, -1.4, -0.7, -0.6)
Mod(polyroot(c(1, -a)))
```

```
## [1] 1.134452 1.137989 1.137989 1.134452
```

四个根都在单位圆外，满足 AR 模型的最小相位条件。

39.4.2.2 MA(2) 模拟例子

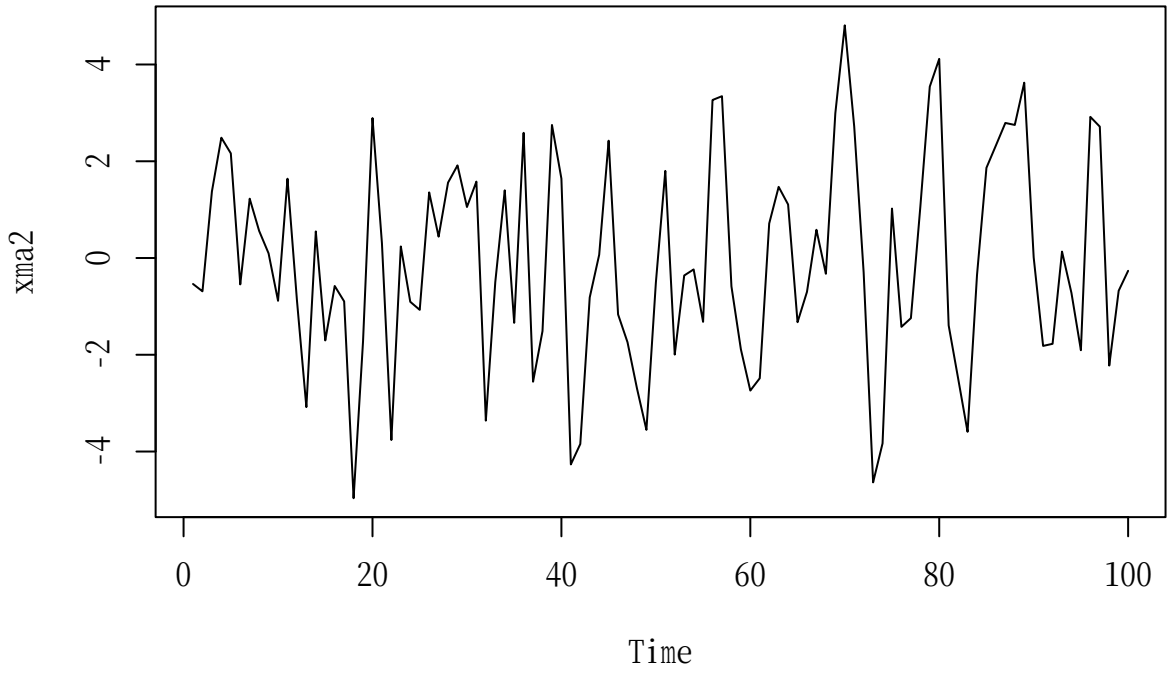
考虑如下的 MA(2) 模型：

$$X_t = \varepsilon_t + 0.5\varepsilon_{t-1} - 0.4\varepsilon_{t-2}, \quad t \in \mathbb{Z}, \tag{39.3}$$

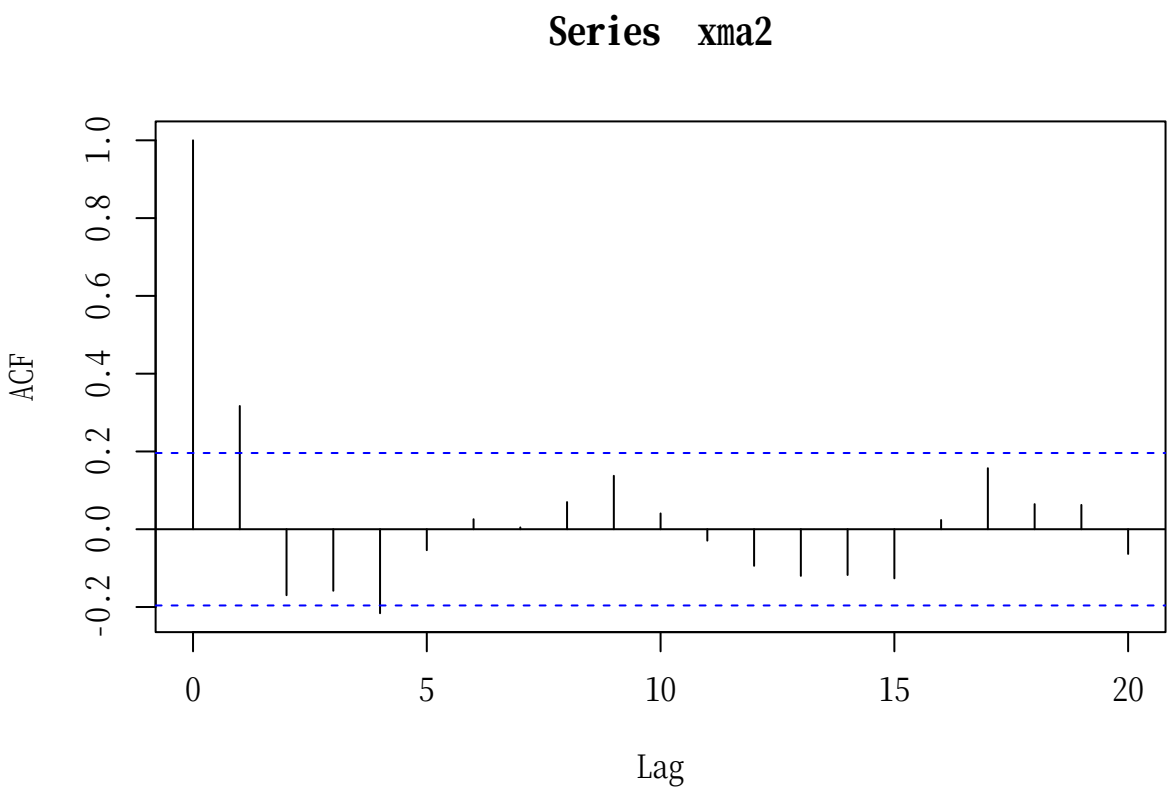
其中 $\{\varepsilon_t\}$ 为 WN(0, 4)。

模拟生成长度为 100 的样本，作时间序列图，ACF，PACF 图：

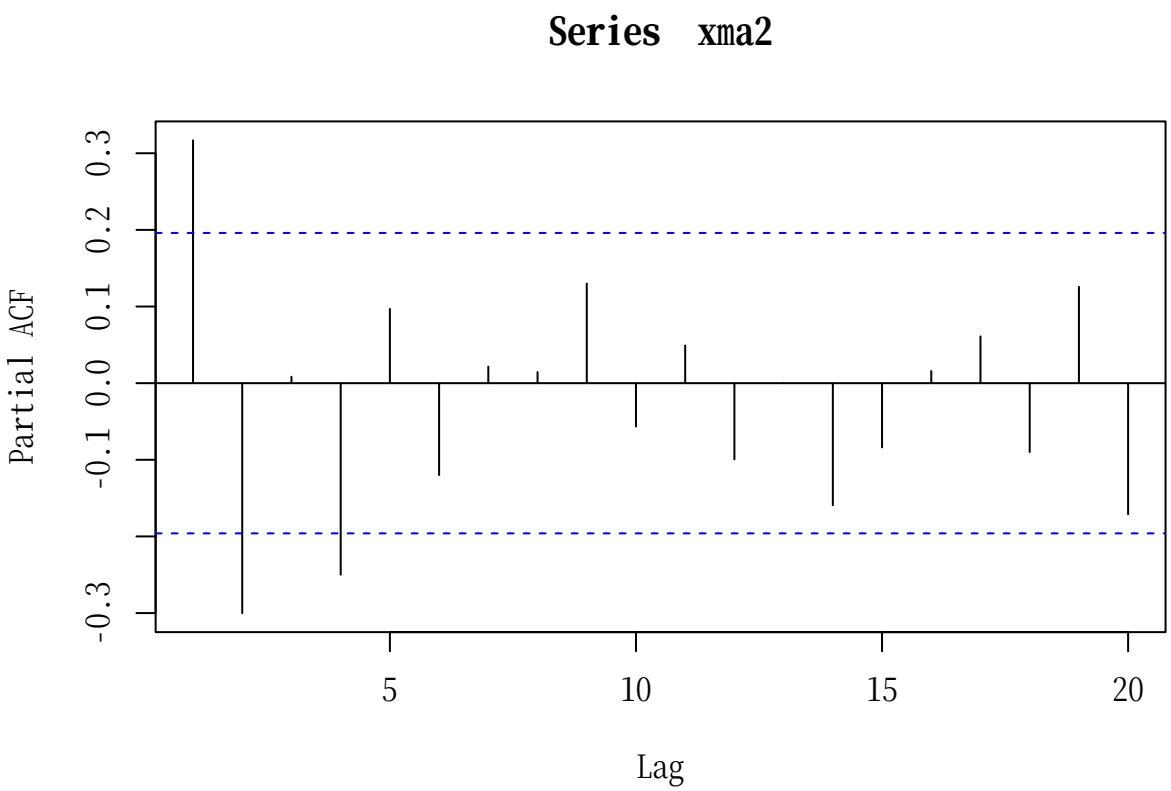
```
set.seed(101)
xma2 <- arima.sim(
  model=list(ma=c(0.5, -0.4)),
  n = 100,
  sd = sqrt(4) )
plot(xma2)
```




```
acf(xma2)
```




```
pacf(xma2)
```



39.4.2.3 ARMA(4,2) 模拟例子

考虑如下的 ARMA(4,2) 模型：

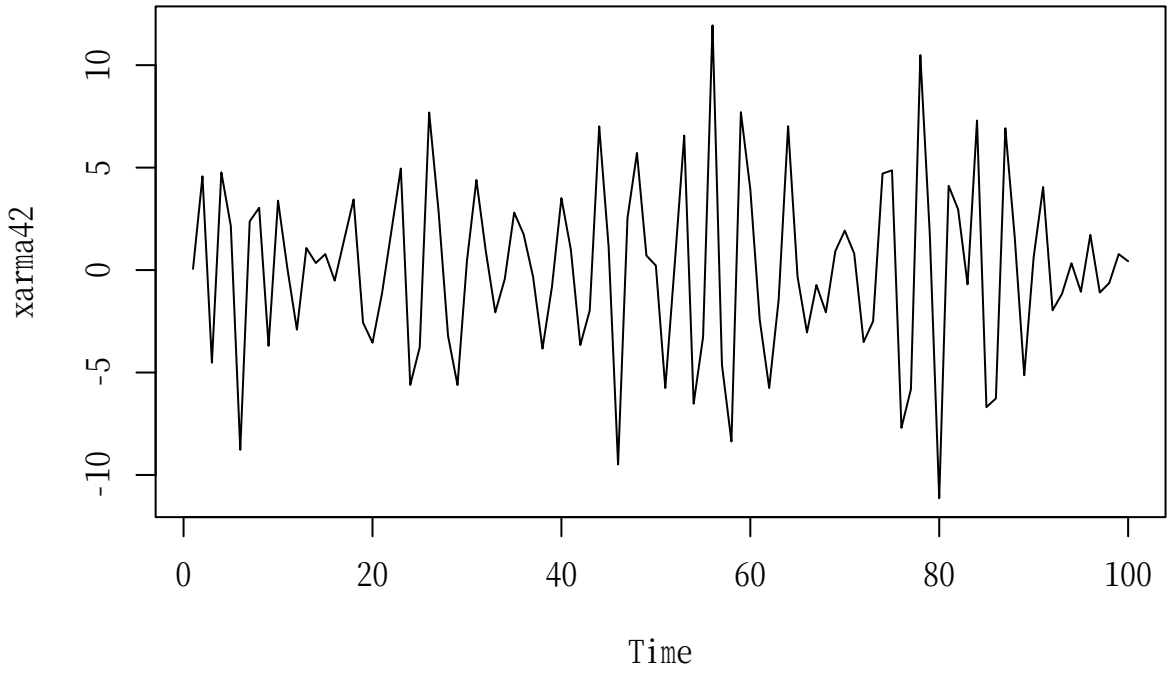
$$X_t = -0.9X_{t-1} - 1.4X_{t-2} - 0.7X_{t-3} - 0.6X_{t-4} \tag{39.4}$$

$$= \varepsilon_t + 0.5\varepsilon_{t-1} - 0.4\varepsilon_{t-2}, \quad t \in \mathbb{Z}, \tag{39.5}$$

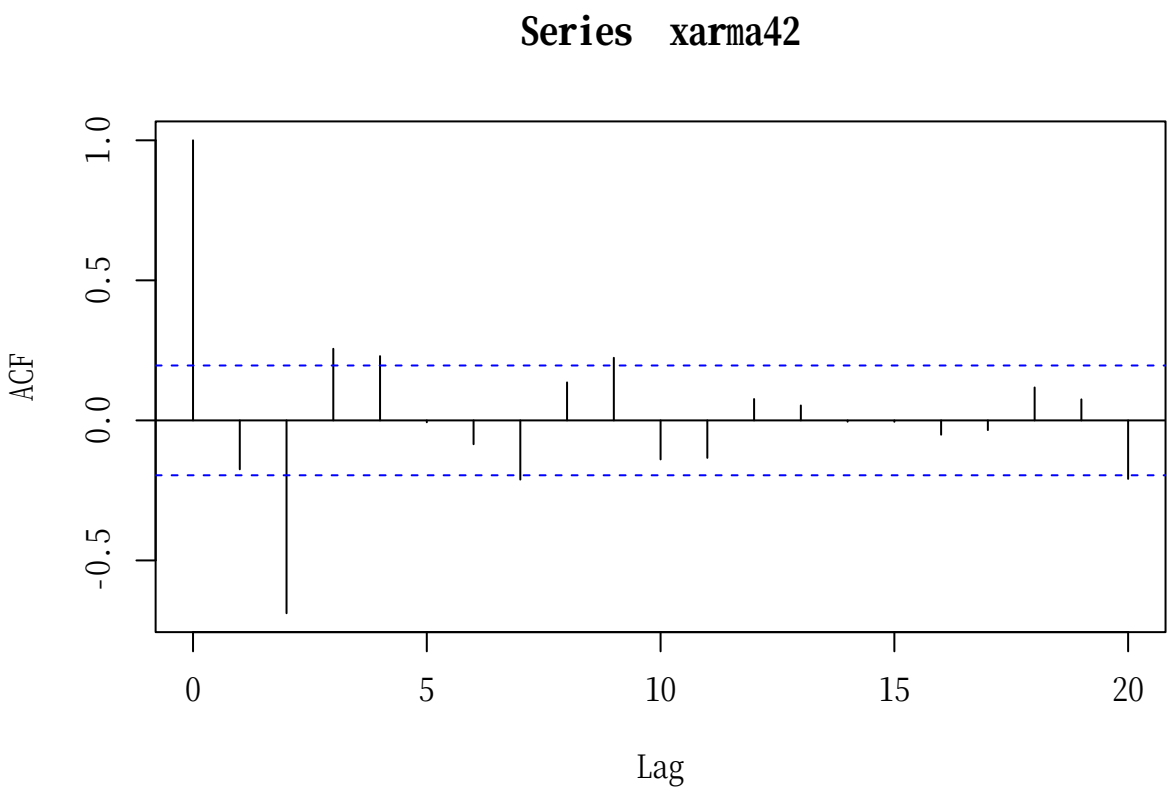
其中 $\{\varepsilon_t\}$ 为 WN(0, 4)。

用如下程序模拟生成长度为 $N = 100$ 的样本，并作时间序列图、ACF 图、PACF 图：

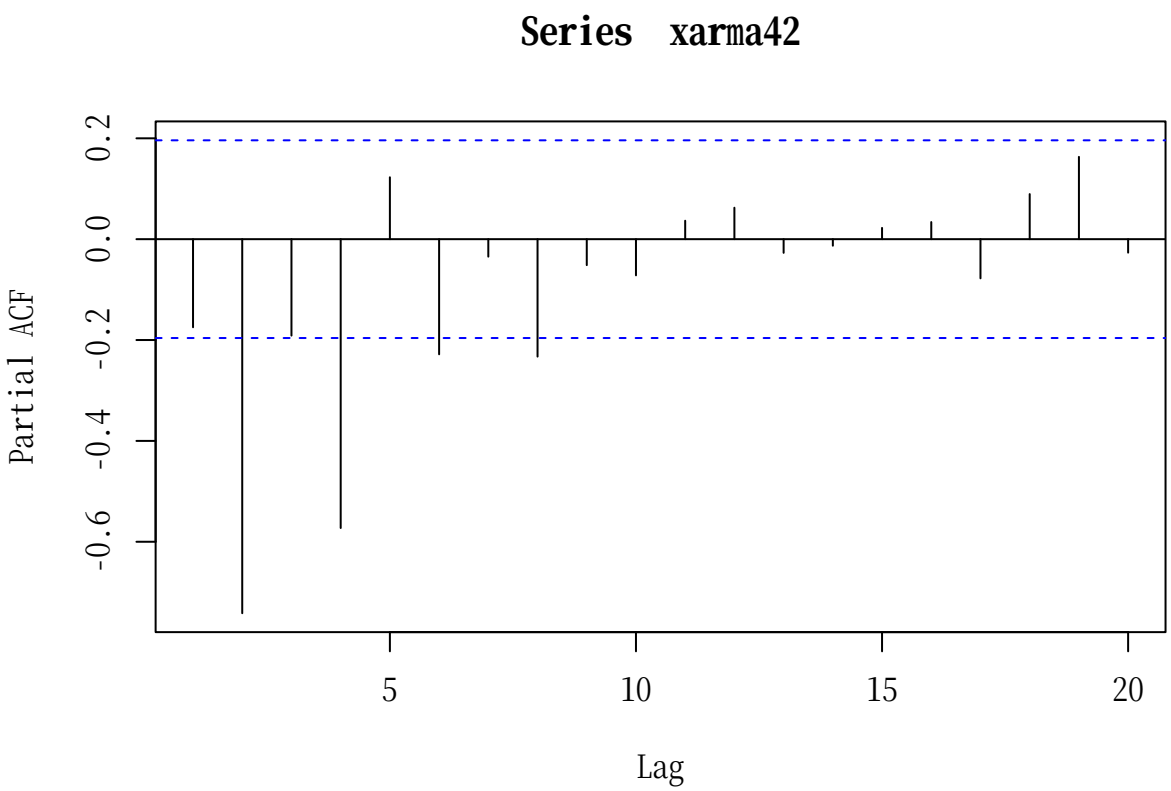
```
set.seed(101)
xarma42 <- arima.sim(
  model=list(
    ar=c(-0.9, -1.4, -0.7, -0.6),
    ma=c(0.5, -0.4)),
  n = 100,
  sd = sqrt(4) )
plot(xarma42)
```




```
acf(xarma42)
```



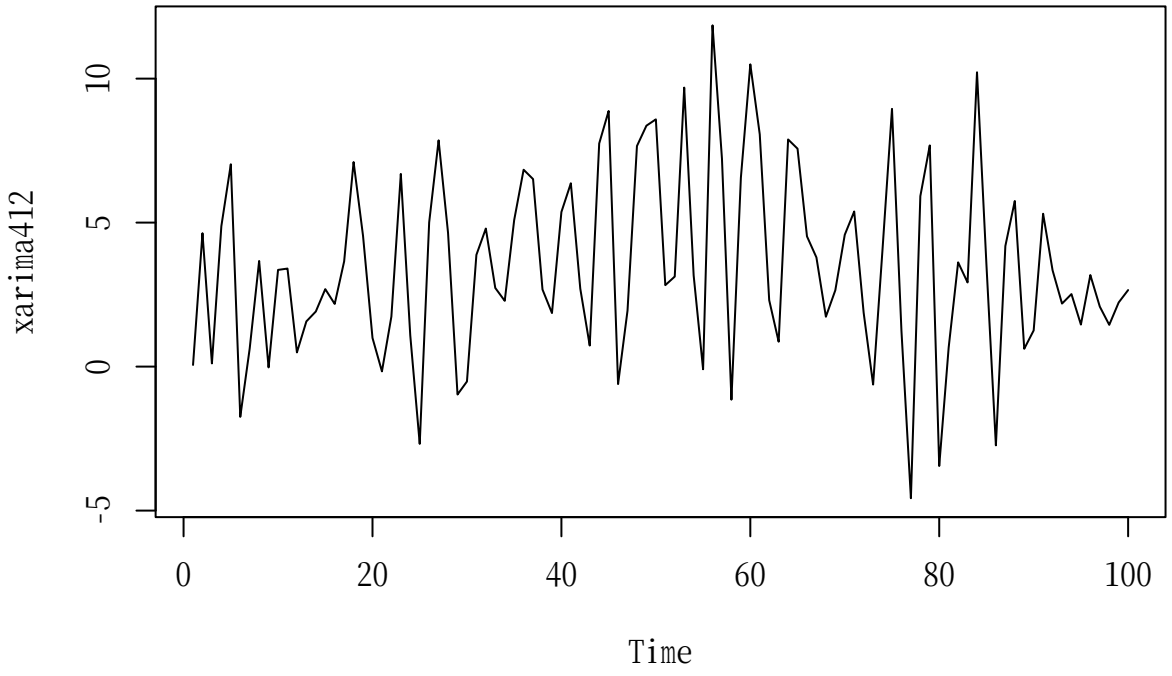

```
pacf(xarma42)
```



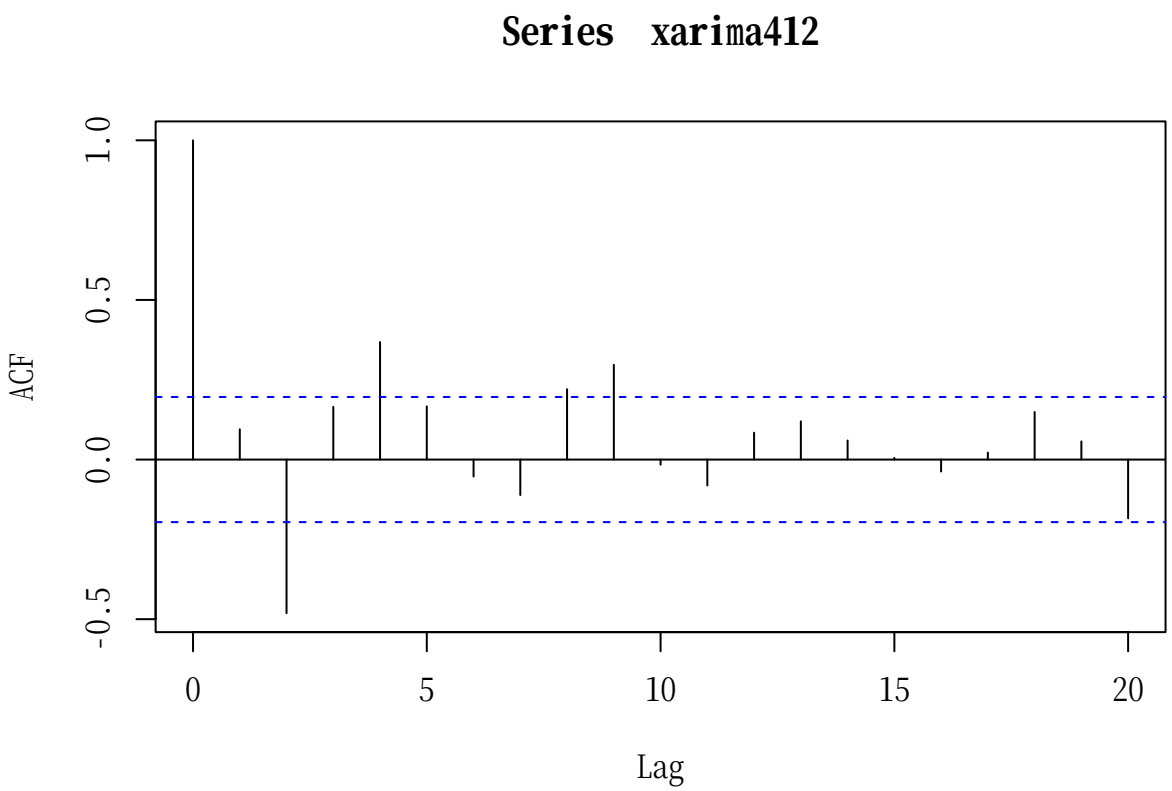
39.4.2.4 ARIMA(4,1,2) 模拟例子

用如下程序模拟生成长度为 $N = 100$ 的 ARIMA(4,1,2) 样本，并作时间序列图、ACF 图、PACF 图。要注意的是，`arima.sim()` 在有一阶差分时会输出 $n + 1$ 个值。

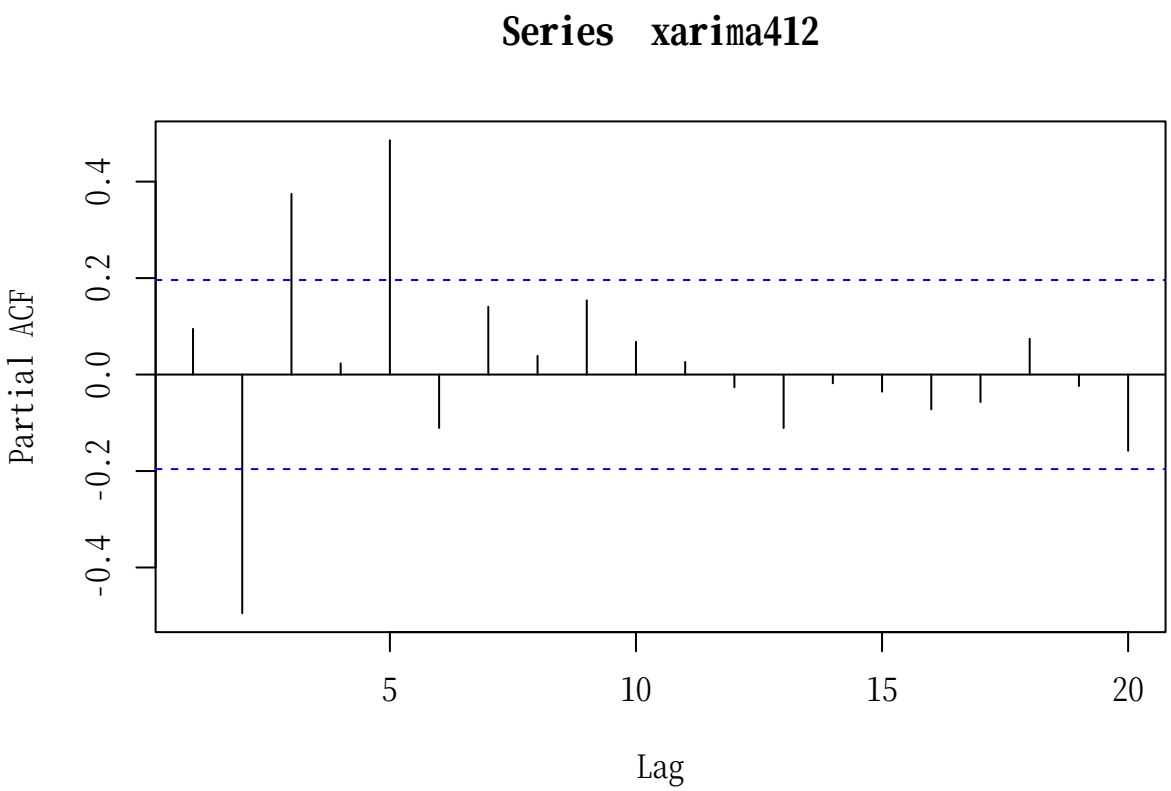
```
set.seed(101)
xarima412 <- arima.sim(
  model=list(
    order = c(4,1,2),
    ar=c(-0.9, -1.4, -0.7, -0.6),
    ma=c(0.5, -0.4)),
  n = 100,
  sd = sqrt(4) )[-1]
ts.plot(xarima412)
```




```
acf(xarima412)
```




```
pacf(xarima412)
```



39.4.3 AR 建模

`stats::ar(x, method="mle")` 可以用最大似然估计方法估计模型参数并用 AIC 定阶。如：

```
resar <- ar(xar4, method="mle"); resar
```

```
##
## Call:
## ar(x = xar4, method = "mle")
##
## Coefficients:
##       1        2        3        4
## -0.7573  -1.2317  -0.5534  -0.5525
##
## Order selected 4  sigma^2 estimated as  3.712
```

结果定阶为 4 阶，模型可以写成：

$$
\begin{aligned}
X_t = & -0.7573X_{t-1} - 1.2317X_{t-2} - 0.5534X_{t-3} - 0.5525X_{t-4} + \varepsilon_t, \\
& \{\varepsilon_t\} \sim \mathrm{WN}(0, 3.712).
\end{aligned}
$$

ar() 的结果是一个列表，元素 order 是阶 p 估计，元素 ar 是 $(a_1, \dots, a_p)$ 估计向量，元素 var.pred 是 σ^2 估计。

39.4.4 ARMA 建模

stats::arima() 可以用最大似然方法估计 AR、MA、ARMA 和 ARIMA 模型，需要人为指定 (p, d, q) 值。如：

```
armares <- arima(
  100 + xarma42, order = c(4,0,2) )
armares
```

```
##
## Call:
## arima(x = 100 + xarma42, order = c(4, 0, 2))
##
## Coefficients:
##           ar1      ar2      ar3      ar4     ma1      ma2  intercept
##       -0.6324  -1.0668  -0.4163  -0.4469  0.3191  -0.6423    99.9989
## s.e.   0.1195   0.1384   0.1350   0.1080  0.1237   0.1267     0.0371
##
## sigma^2 estimated as 3.582:  log likelihood = -209.2,  aic = 434.4
```

模型可以表示为：

$$
\begin{aligned}
EX_t =& 99.9989 \\
Y_t =& X_t - EX_t \\
Y_t =& -0.6324X_{t-1} - 1.0668X_{t-2} - 0.4163X_{t-3} - 0.4469X_{t-4} + \varepsilon_t + 0.3191\varepsilon_{t-1} - 0.6423\varepsilon_{t-2}, \\
\{\varepsilon_t\} \sim& \mathrm{WN}(0, 3.582)
\end{aligned}
$$

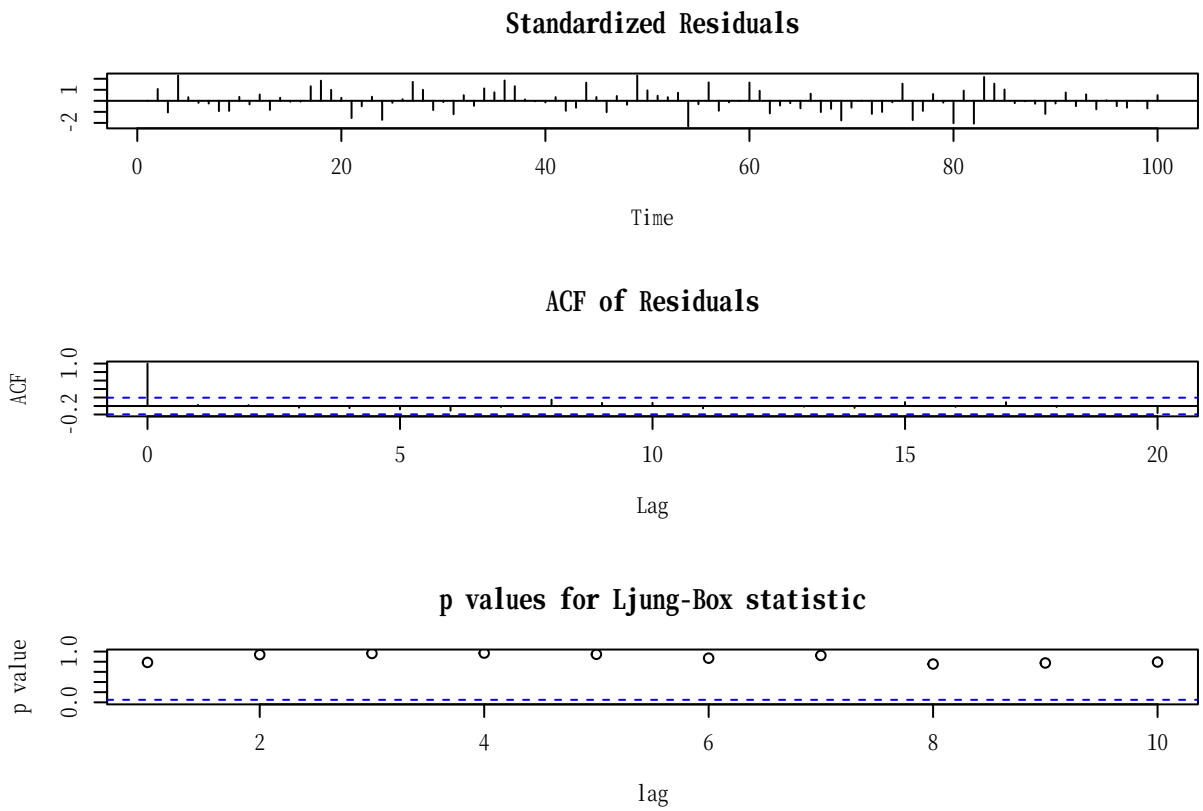
另外，AIC 值为 434.4。可以试验不同的阶，找到 AIC 最低的模型。

arima() 的结果是一个列表，其中 coef 是 $(a_1, \dots, a_p, b_1, \dots, b_q, \mu)$ 的估计。sigma2 是 σ^2 的估计。var.coef 是 $(a_1, \dots, a_p, b_1, \dots, b_q, \mu)$ 的估计的方差阵估计，其对角线的平方根是相应的标准误差。residuals 是估计残差，长度与输入时间序列相同。

39.4.5 模型诊断

将 arima() 的输出结果输入到 tsdiag() 函数，可以进行模型诊断，结果包括标准化残差、残差的 ACF、残差的 Ljung-Box 白噪声检验 p 值，检验对多个滞后值计算，p 值高于横虚线 (0.05 线) 表示模型适合。

```
tsdiag(armares)
```



39.4.6 白噪声检验

Ljung-Box 检验是常用的白噪声检验，零假设为数据来自白噪声列。`Box.test(x, type="Ljung-Box", lag=m)` 执行 Ljung-Box 白噪声检验, `m` 是用到的自相关函数个数。如：

```
Box.test(xarma42, type="Ljung-Box", lag = 10)
```

```
##
##  Box-Ljung test
##
## data:  xarma42
## X-squared = 80.473, df = 10, p-value = 4.056e-13
```

如果要对 ARMA 或者 ARIMA 建模的残差进行白噪声检验，需要加 `fitdf` 选项，取值为 $p+q$，如：

```
Box.test(armares$residuals, type="Ljung-Box", lag = 10, fitdf=4+2)
```

```
##
##  Box-Ljung test
##
## data:  armares$residuals
## X-squared = 6.2553, df = 4, p-value = 0.1809
```

在 0.05 水平下结果不显著，提示模型是合适的。

39.4.7 稀疏系数估计

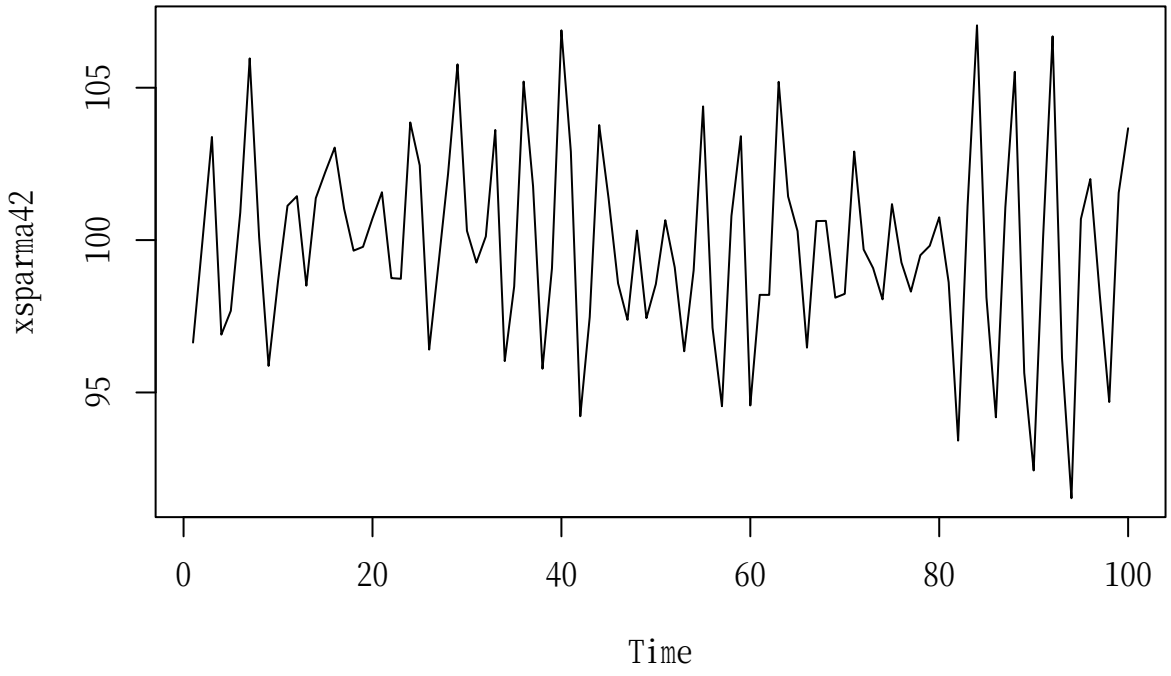
`arima` 函数允许指定某些系数固定为预先确定的值，不从数据中估计。

例如，考虑如下的稀疏系数的 ARMA(4,2) 模型

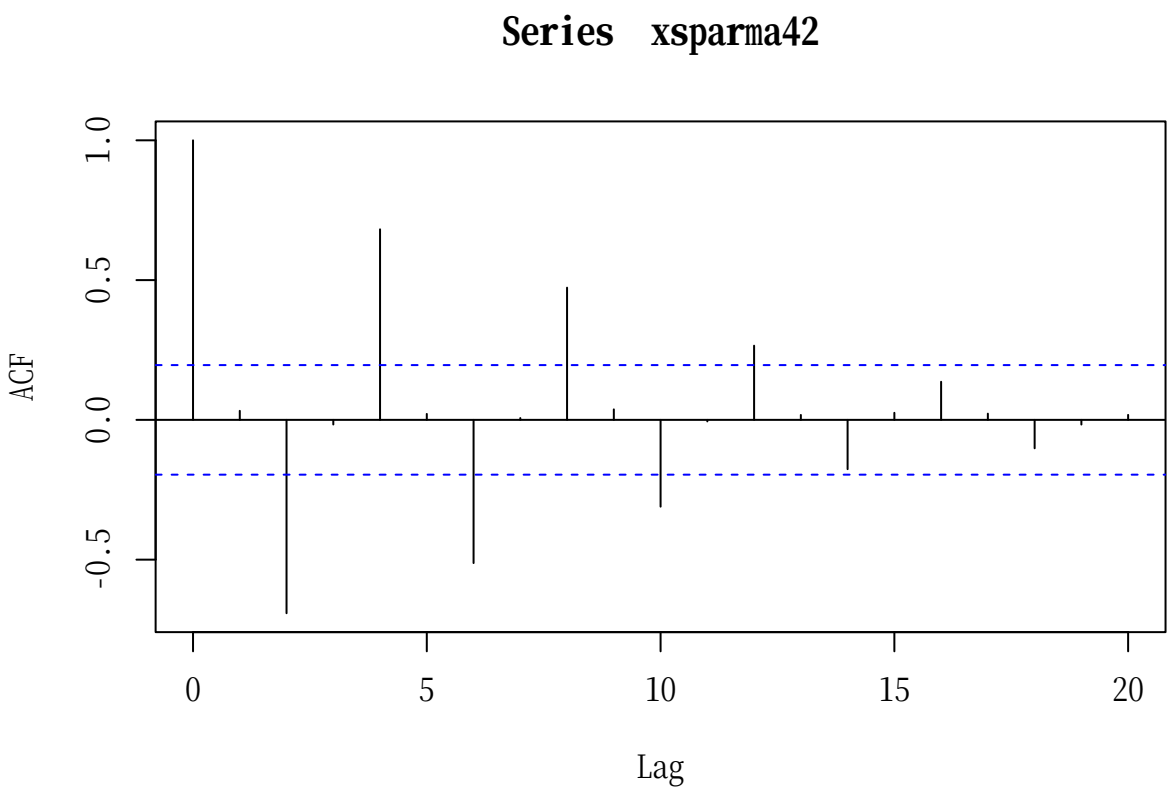
$$
\begin{aligned}
X_t =& 100 + Y_t, \\
Y_t =& 0.7X_{t-4} + \varepsilon_t - 0.4\varepsilon_{t-2}, \\
\varepsilon_t \sim& \mathrm{WN}(0, 4)
\end{aligned}
$$

产生模拟数据：

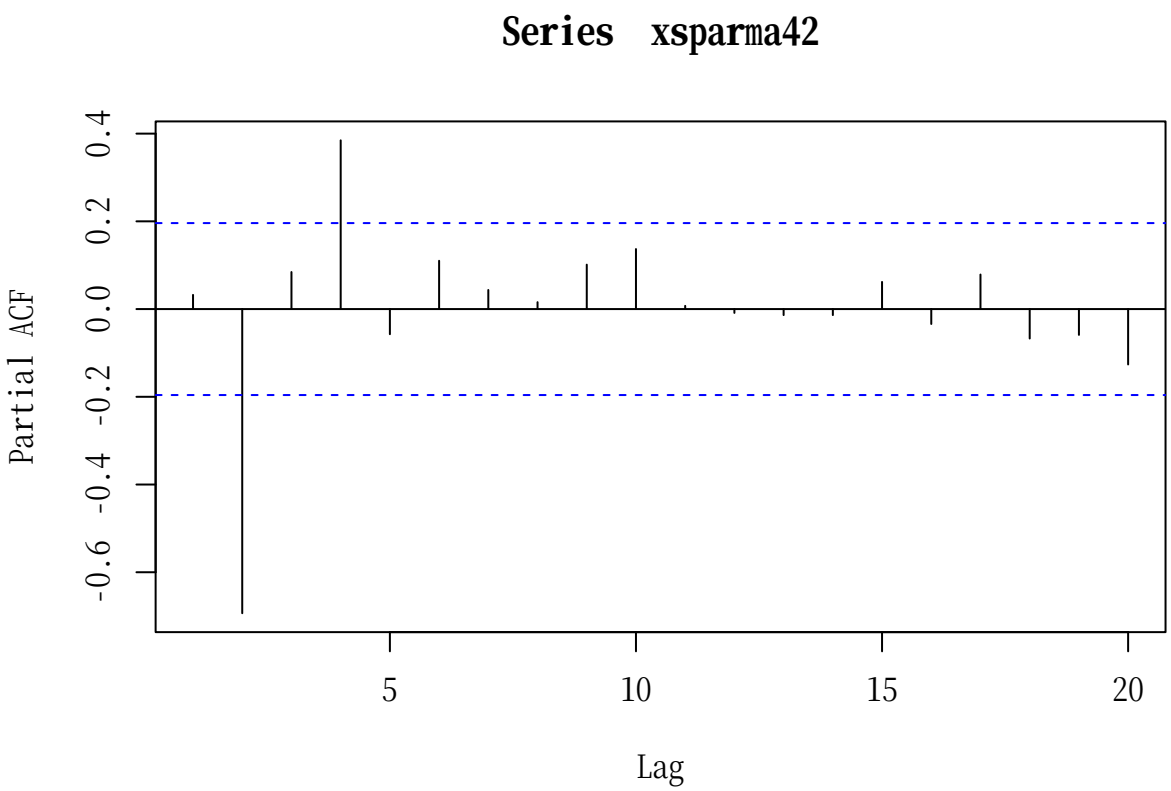
```
set.seed(101)
xsparma42 <- 100 + arima.sim(
  model=list(
    ar=c(0, 0, 0, 0.7),
    ma=c(0, -0.4)),
  n = 100,
  sd = sqrt(4) )
plot(xsparma42)
```




```
acf(xsparma42)
```




```
pacf(xsparma42)
```



拟合 ARMA(4,2) 模型：

```
spres1 <- arima(
  xsparma42, order = c(4,0,2) )
spres1
```

```
##
## Call:
## arima(x = xsparma42, order = c(4, 0, 2))
##
## Coefficients:
##          ar1      ar2     ar3     ar4      ma1      ma2  intercept
##       0.0761  -0.0908  0.0078  0.6437  -0.0276  -0.4091    99.7500
## s.e.  0.1955   0.1963  0.1296  0.1314   0.2733   0.2860     0.3048
##
## sigma^2 estimated as 4.066:  log likelihood = -213.21,  aic = 442.41
```

计算各个系数估计除以标准误差的比值，在系数为 0 的条件下近似服从标准正态分布：

```
zstat.arima <- function(arimares){
  with(
    arimares, {
      y <- numeric(length(coef))
      names(y) <- names(coef)
      y[mask] <- coef[mask] / sqrt(diag(var.coef))
      y
    })
}
round(zstat.arima(spres1), 2)
```

```
##       ar1       ar2       ar3       ar4       ma1       ma2 intercept
##      0.39     -0.46      0.06      4.90     -0.10     -1.43    327.21
```

当近似 Z 统计量值在正负 2 之间时，可认为相应的系数等于零，用如下程序估计稀疏系数的 ARMA 模型：

```
spres2 <- arima(
  xsparma42, order = c(4,0,2),
  fixed = c(0, 0, 0, NA, 0, NA, NA),
  transform.pars = FALSE)
spres2
```

```
##
## Call:
## arima(x = xsparma42, order = c(4, 0, 2), transform.pars = FALSE, fixed = c(0,
##     0, 0, NA, 0, NA, NA))
##
## Coefficients:
##       ar1  ar2  ar3     ar4  ma1      ma2  intercept
##         0    0    0  0.6837    0  -0.5035    99.7432
## s.e.    0    0    0  0.0757    0   0.0927     0.2996
##
## sigma^2 estimated as 4.108:  log likelihood = -213.71,  aic = 435.42
```

```
round(zstat.arima(spres2), 2)
```

```
##       ar1       ar2       ar3       ar4       ma1       ma2 intercept
##      0.00      0.00      0.00      9.03      0.00     -5.43    332.96
```

为了指定某些系数，在 `arima` 中用 `fixed=` 选项，需要指定的系数就输入指定的值，需要从数据中估计的值就输入 `NA`，次序按 `arima` 显示结果时各个系数的次序。

39.4.8 单位根检验

单位根检验是一种平稳性检验，零假设是有单位根，即不平稳；对立假设是平稳。经常使用增强的 Dickey-Fuller 检验 (ADF 检验)。

fUnitRoots 包的 `adfTest()` 函数可以执行单位根 ADF 检验。tseries 包的 `adf.test()` 函数也可以执行单位根 ADF 检验。

注意，ADF 检验都是在拒绝 H_0（显著）时否认有单位根，不显著时承认有单位根。

例如，对模拟的 ARMA(4,2) 序列数据做单位根检验：

```
fUnitRoots::adfTest(100 + xarma42, lags=8, type="c")
```

```
## Warning in fUnitRoots::adfTest(100 + xarma42, lags = 8, type = "c"): p-value
## smaller than printed p-value
```

```
##
## Title:
##  Augmented Dickey-Fuller Test
##
## Test Results:
##   PARAMETER:
##     Lag Order: 8
##   STATISTIC:
##     Dickey-Fuller: -5.2017
##   P VALUE:
##     0.01
##
## Description:
##  Sun Nov  6 21:29:48 2022 by user: Lenovo
```

在 0.05 水平下显著，说明没有单位根。函数中的 `lags` 参数是用来作为对立假设的 AR 模型的阶。可以先尝试拟合 AR 模型，并用适当方法定阶，然后再进行 ADF 检验。

对模拟的 ARIMA(4,1,2) 序列数据检验：

```
fUnitRoots::adfTest(xarima412, lags=8, type="c")
```

```
##
## Title:
##  Augmented Dickey-Fuller Test
##
```

```
## Test Results:
##   PARAMETER:
##     Lag Order: 8
##   STATISTIC:
##     Dickey-Fuller: -1.539
##   P VALUE:
##     0.4856
##
## Description:
##  Sun Nov  6 21:29:48 2022 by user: Lenovo
```

在 0.05 水平下结果不显著，说明有单位根。

选项 `type` 选择基础模型，可以取：

- `"nc"`，表示没有漂移项或截距项；
- `"c"`，表示带有一个漂移项或截距项；
- `"ct"`，表示基础模型中带有 $a + bt$ 这样的线性项。

39.4.9 ARMA 模型的一些自定义函数

这里列出一些与 ARMA 模型等时间序列理论有关的函数，比如，从模型参数计算 Wold 系数，解 Yule-Walker 方程，从模型参数计算协方差函数，逆相关函数计算，ARMA 谱密度绘图，MA 矩估计，Levinson 递推，递推预测，等等。欠缺逆相关法等少数的程序。

AR 模型理论谱密度图：

```r
ar.true.spectrum <- function(
  a, ngrid=256, sigma=1, plot.it=TRUE,
  title="AR True Spectral Density"){
  p <- length(a)
  freqs <- seq(from=0, to=pi, length=ngrid)
  spec <- numeric(ngrid)
  for(ii in seq(ngrid)){
    spec[ii] <- sigma^2 / (2*pi) /
      abs(1 - sum(complex(mod=a, arg=freqs[ii]*seq(p))))^2
  }
  if(plot.it){
    plot(freqs, spec, type='l',
         main=title,
         xlab="frequency", ylab="spectral density",
```

```
        axes=FALSE)
    axis(2)
    axis(1, at=(0:6)/6*pi,
         labels=c(0, expression(pi/6),
           expression(pi/3), expression(pi/2),
           expression(2*pi/3), expression(5*pi/6), expression(pi)))
  }
  list(frequencies=freqs, spectrum=spec,
       ar.coefficients=a, sigma=sigma)
}
```

MA 理论谱密度图:

```
ma.true.spectrum <- function(
  a, ngrid=256, sigma=1,
  tit="True MA Spectral Density",
  plot.it=TRUE){
  p <- length(a)
  freqs <- seq(from=0, to=pi, length=ngrid)
  spec <- numeric(ngrid)
  for(ii in seq(ngrid)){
    spec[ii] <- 1 + sum(complex(mod=a, arg=freqs[ii]*seq(p)))
  }
  spec = sigma^2 / (2*pi) * abs(spec)^2
  if(plot.it){
    plot(freqs, spec, type='l',
         main=tit,
         xlab="frequency", ylab="spectrum",
         axes=FALSE)
    axis(2)
    axis(1, at=(0:6)/6*pi,
         labels=c(0, expression(pi/6),
           expression(pi/3), expression(pi/2),
           expression(2*pi/3), expression(5*pi/6), expression(pi)))
    box()
  }
  invisible(list(frequencies=freqs, spectrum=spec,
            ma.coefficients=a, sigma=sigma))
}
```

利用李雷的矩阵极限方法对 MA 模型参数作矩估计。输入 $\gamma_0, \gamma_1, \ldots, \gamma_q$，输出 MA 参数 $b_1, \ldots, b_q, \sigma^2$ 的估计。

```
ma.solve <- function(gms, k=100){
  q <- length(gms)-1
  if(q==1){
    rho1 <- gms[2] / gms[1]
    b <- (1 - sqrt(1 - 4*rho1^2))/(2*rho1)
    s2 <- gms[1] / (1 + b^2)
    return(list(b=b, s2=s2))
  }

  A <- matrix(0, nrow=q, ncol=q)
  for(j in seq(2,q)){
    A[j-1,j] <- 1
  }
  cc <- numeric(q); cc[1] <- 1
  gamma0 <- gms[1]
  gammas <- numeric(q+k)
  gammas[1:(q+1)] <- gms
  gamq <- gms[-1]
  Gammak <- matrix(0, nrow=k, ncol=k)
  for(ii in seq(k)){
    for(jj in seq(k)){
      Gammak[ii,jj] <- gammas[abs(ii-jj)+1]
    }
  }

  Omk <- matrix(0, nrow=q, ncol=k)
  for(ii in seq(q)){
    for(jj in seq(k)){
      Omk[ii,jj] <- gammas[ii+jj-1+1]
    }
  }
  PI <- Omk %*% solve(Gammak, t(Omk))

  s2 <- gamma0 - c(t(cc) %*% PI %*% cc)
  b <- 1/s2 * c(gamq - A %*% PI %*% cc)
  return(list(b=b, s2=s2))
}
```

ARMA 理论谱密度图：

```
arma.true.spectrum <- function(
  a, b, ngrid=256, sigma=1,
  tit="True ARMA Spectral Density",
  plot.it=TRUE){
  p <- length(a)
  q <- length(b)
  freqs <- seq(from=0, to=pi, length=ngrid)
  spec1 <- numeric(ngrid)
  spec2 <- numeric(ngrid)
  for(ii in seq(ngrid)){
    spec1[ii] <- 1 + sum(complex(mod=b, arg=freqs[ii]*seq(q)))
    spec2[ii] <- 1 - sum(complex(mod=a, arg=freqs[ii]*seq(p)))
  }
  spec = sigma^2 / (2*pi) * abs(spec1)^2 / abs(spec2)^2
  if(plot.it){
    plot(freqs, spec, type='l',
         main=tit,
         xlab="frequency", ylab="spectrum",
         axes=FALSE)
    axis(2)
    axis(1, at=(0:6)/6*pi,
         labels=c(0, expression(pi/6),
           expression(pi/3), expression(pi/2),
           expression(2*pi/3), expression(5*pi/6), expression(pi)))
  }
  box()
  invisible(list(frequencies=freqs, spectrum=spec,
       ar.coefficients=a, ma.coefficients=b,
       sigma=sigma))
}
```

从 ARMA 模型参数计算 Wold 系数，n 是需要计算的个数：

```
arma.Wold <- function(n, a, b=numeric(0)){
  p <- length(a)
  q <- length(b)
  arev <- rev(a)
  psi <- numeric(n)
```

```
  psi[1] <- 1
  for(j in seq(n-1)){
    if(j <= q) bj=b[j]
    else bj=0
    psis <- psi[max(1, j+1-p):j]
    np <- length(psis)
    if(np < p) psis <- c(rep(0,p-np), psis)
    psi[j+1] <- bj + sum(arev * psis)
  }

  psi
}
```

给定 ARMA 模型参数，利用 Wold 系数计算自协方差函数。这里对 Wold 系数进行了截断，nw 值越大，使用的 Wold 系数越多，结果越精确，但计算量也越大。事实上，对 $k \geq q+1$，可以用 Yule-Walker 方程递推计算 γ_k。

```
arma.gamma.by.Wold <- function(n, a, b=numeric(0), sigma=1, nw=100){
  nn <- n + nw
  psi <- arma.Wold(nn, a, b, nw=nw)
  gam <- numeric(n)
  for(ii in seq(0, n-1)){
    gam[ii+1] <- sum(psi[1:(nn-ii)] * psi[(ii+1):nn])
  }
  gam <- (sigma^2) * gam
  gam
}
```

用矩估计法求 ARMA 参数估计，MA 部分用李雷的矩阵极限算法。

```
arma.solve <- function(gms, p, q){
  Gpq <- matrix(0, nrow=p, ncol=p)
  for(ii in seq(p)) for(jj in seq(p)){
    Gpq[ii,jj] <- gms[abs(q + ii - jj)+1]
  }
  gs <- gms[(q+1+1):(q+p+1)]
  a <- solve(Gpq, gs)
  aa <- c(-1, a)

  gys <- numeric(q+1)
  for(k in seq(0, q)){
```

```
    gys[k+1] <- sum(c(outer(0:p,0:p,
                            function(ii,jj) aa[ii+1] * aa[jj+1]
                            * gms[abs(k+jj-ii)+1])))
  }

  res <- ma.solve(gys)
  b <- res$b
  sigma <- sqrt(res$s2)

  list(a=a, b=b, sigma=sigma)
}
```

ARMA 模型矩估计法，其中 MA 部分用逆相关函数法的 R 程序：

```
## 用输入的 $\gamma_0, \dots, \gamma_p$ 求解 AR(p) 模型参数
ar.yw <- function(gam){
  p <- length(gam) - 1
  G <- matrix(0, p, p)
  for(i in 1:p){
    for(j in 1:p){
      G[i,j] = gam[1 + abs(i-j)]
    }
  }
  gri <- gam[-1]
  a <- solve(G, gri)
  sig2 <- sum(gam * c(1, -a))
  list(ar = a, var.pred = sig2)
}

## 从 AR 模型参数计算逆相关函数
ar_racv <- function(a, sig2){
  p <- length(a)
  a <- c(-1, a)
  a2 <- c(a, numeric(p+1))
  res <- numeric(p+1)
  for(k in seq(0, p, by=1)){
    res[k+1] <- sum(a * a2[(k+1):(k+p+1)])
  }
  res/sig2
}
```

```r
## 用长阶自回归方法和逆相关函数方法估计 MA 模型
ma.solve.racv <- function(x, q=1){

  ## 拟合长阶自回归，用来估计逆自协方差函数
  n <- length(x)
  plar <- round(sqrt(n))
  if(plar < q) plar <- q

  mod1 <- ar(x, aic = FALSE, order.max = plar, method="yule-walker")
  a <- mod1$ar
  sigsq <- mod1$var.pred
  gamr <- ar_racv(a, sigsq)[1:(q+1)]
  res2 <- ar.yw(gamr)
  b <- -res2$ar
  sig2 <- 1/res2$var.pred

  list(ma = b, var.pred = sig2)
}


## 用矩估计法估计 ARMA，其中 MA 部分用逆相关函数法
## gms 输入 $\gamma_k$, $k=0,1,\dots,p+q$。
arma.moment.racv <- function(x, p=1, q=1){
  n <- length(x)
  pmax <- round(sqrt(n))
  if(pmax < p+q) pmax <- p+q
  ## 计算样本自协方差函数
  gms <- c(acf(x, lag.max=pmax+p, type="covariance", plot=FALSE)$acf)

  ## 求解 AR 部分
  Gpq <- matrix(0, nrow=p, ncol=p)
  for(ii in seq(p)) for(jj in seq(p)){
    Gpq[ii,jj] <- gms[abs(q + ii - jj)+1]
  }
  gs <- gms[(q+1+1):(q+p+1)]
  a <- solve(Gpq, gs)
  aa <- c(-1, a)
```

```
  ## 从已有的自协方差估计与 AR 参数估计，计算 MA 部分的 $\gamma_j, j=0,1,\dots,pmax$。
  gys <- numeric(pmax+1)
  for(k in seq(0, pmax)){
    gys[k+1] <- sum(c(outer(0:p,0:p,
                            function(ii,jj) aa[ii+1] * aa[jj+1]
                            * gms[abs(k+jj-ii)+1])))
  }

  ## 从 MA 部分的自协方差估计，拟合长阶自回归模型
  res1 <- ar.yw(gys)

  ## 求 MA 部分的逆相关函数
  gamr <- ar_racv(res1$ar, res1$var.pred)[1:(q+1)]

  ## 利用逆相关函数为输入求解 YW 方程，得到 MA 参数估计
  res2 <- ar.yw(gamr)
  b <- -res2$ar
  sig2 <- 1/res2$var.pred
  ##sig2 <- gys[1] / sum(c(1, b)^2)

  list(ar=a, ma=b, var.pred=sig2)
}
```

时间序列递推预测系数计算。输入一个 $n \times n$ 矩阵表示序列非平稳，输入为观测向量的协方差阵；输入一个向量表示序列平稳，输入为平稳列的自协方差函数值 $\gamma_0, \gamma_1, \dots, \gamma_{n-1}$。输出是用新息预报方法对 $Y_1, Y_2, \dots, Y_n$ 进行一步最佳线性预报的系数和均方误差。结果为列表，元素 `theta` 是 $(n-1) \times (n-1)$ 矩阵，第 k 行为预报 Y_{k+1} 的样本新息系数：

$$\hat{Y}_{k+1} = \theta[k,1] \cdot W_k + \theta[k,2] \cdot W_{k-1} + \cdots + \theta[k,k] \cdot W_1$$

$$W_1 = Y_1, W_k = Y_k - \hat{Y}_k, k = 2, 3, \dots, n.$$

元素 `nu` 是 n 个预报均方误差，`nu[k]` 是预报 Y_k 的均方误差。

```
### 非平稳时间序列新息递推预测的系数和一步预测误差方差计算。
### 输入:
###    Gam --- 如果序列非平稳，为自协方差矩阵 n*n
###            如果序列平稳，为自协方差列\gamma_0, \gamma_1, \dots, \gamma_{n-1}
### 输出:
###   用新息预报方法对 Y_1, Y_2, \dots, Y_n 进行一步最佳线性预报的系数和均方误差。
###   theta -- (n-1)*(n-1) 矩阵。第 k 行为预报 Y_{k+1} 的样本新息系数:
###      \hat Y_{k+1} = theta[k,1]*W_k + theta[k,2]*W_{k-1} + \dots + theta[k,k]*W_1
```

```
###      W_1=Y_1, W_k = Y_k - \hat Y_k, k=2,3,\dots,n
###   nu --- n 个元素，nu[k] 为预报 Y_k 的均方误差。
ts.ipred.coef <- function(Gam){
  if(!is.matrix(Gam)){
    n <- length(Gam)
    Gam <- outer(1:n, 1:n,
                 function(i,j) Gam[abs(i-j)+1])
  } else {
    n <- nrow(Gam)
  }
  ## X_1, X_2, \dots, X_{n-1} 的自协方差矩阵为 Gam

  ## 新息预报递推
  theta <- matrix(0, n-1, n-1)
  nu <- numeric(n)
  nu[1] <- Gam[1,1]
  theta[1,1] <- Gam[2,1] / Gam[1,1]
  nu[2] <- Gam[2,2] - theta[1,1]^2 * nu[1]
  for(k in 2:(n-1)){
    theta[k, k] <- Gam[k+1,0+1] / nu[1]
    for(j in 1:(k-1)){
      theta[k,k-j] <- (Gam[k+1,j+1]
                       - sum(theta[j, j:1]*theta[k,k:(k-j+1)]*nu[1:j])) /
                         nu[j+1]
    }
    nu[k+1] <- Gam[k+1,k+1] - sum(theta[k,k:1]^2 * nu[1:k])
  }
  list(theta=theta, nu=nu)
}
```

随时间序列进行递推预报 (新息预报)，假定已知自协方差列（平稳时）或自协方差阵（非平稳时）。输入 x 是时间序列观测数据，可以包含要预报的部分作为对比。输入 gams 向量或矩阵。向量时为平稳序列的自协方差列，矩阵时为非平稳时间序列的自协方差阵。输入 demean 表示是否减去均值再预报。输入 end 表示预报到哪个时间点，这受到 gams 大小的限制，只能预报到与自协方差阵阶数相同的时间点。conf 是计算预测区间的置信度。

```
ts.ipred <- function(x, gams, demean=FALSE, end=length(x), conf=0.95){
  stationary <- !is.matrix(gams)
  res1 <- ts.ipred.coef(gams)
  theta <- res1$theta
  nu <- res1$nu
```

```
  if(demean) {xmean <- mean(x); x <- x - xmean}

  if(!stationary) end <- length(nu)
  W <- numeric(end)
  pred <- numeric(end)
  lb <- numeric(end)
  ub <- numeric(end)
  lam <- qnorm(1 - (1-conf)/2)

  pred[1] <- 0
  W[1] <- x[1]
  for(n in 1:(end-1)){
    pred[n+1] <- sum(theta[n, n:1]*W[1:n])
    W[n+1] <- x[n+1] - pred[n+1]
  }
  lb <- pred - lam*sqrt(nu[1:end])
  ub <- pred + lam*sqrt(nu[1:end])

  if(demean){
    pred <- xmean + pred
    lb <- xmean + lb
    ub <- xmean + ub
  }
  list(x=x, pred=pred, lb=lb, ub=ub, conf=conf)
}
```

对平稳列，已知自协方差列时用 Levinson 递推计算逐个一步预报系数 (Y-W 系数) 和一步预测误差方差。输入 `gams[1:n]` 为 $\gamma_k, k = 0, 1, \dots, n-1$。输出是对 $Y_1, Y_2, \dots, Y_n$ 做滚动向前一步预报所需的系数和均方误差。结果中元素 `coef.YW` 是 $(n-1) \times (n-1)$ 矩阵，第 k 行的 $1:k$ 元素为 $a_{k,1}, a_{k,2}, \dots, a_{k,k}$，用来预报 Y_{k+1}:

$$L(Y_{k+1}|Y_1, Y_2, \dots, Y_k) = a_{k,1}Y_k + a_{k,2}Y_{k-1} + \cdots + a_{k,k}Y_1,\ k = 1, 2, \dots, n-1$$

$$L(Y_1|) = 0$$

结果中 `sigmasq` 是长度为 n 的向量，保存预报 $Y_1, Y_2, \dots, Y_n$ 的一步预报均方误差。

```
Levinson.coef <- function(gams){
  n <- length(gams)

  ## ayw 保存 Y-W 系数 a[i,j], i=1,2,\dots,n-1, j=1,2,\dots,i
  ayw <- matrix(0, n-1, n-1)
```

```
  ## ss 保存一步预报误差方差
  ss <- numeric(n)

  ss[1] <- gams[0+1] ## 预报 Y_1 的均方误差，等于\gamma_0
  ayw[1,1] <- gams[1+1] / gams[0+1] ## 用 Y_1 预报 Y_2 的系数
  ss[2] <- ss[1] * (1 - ayw[1,1]^2) ## 用 Y_1 预报 Y_2 的均方误差
  if(n>2) for(k in 1:(n-2)){
    ## 用 Y_1, \dots, Y_{k+1} 预报 Y_{k+2} 的系数
    ayw[k+1,k+1] <- (gams[(k+1)+1] - sum(ayw[k,1:k] * gams[(k:1)+1])) / ss[k+1]
    ayw[k+1,1:k] <- ayw[k, 1:k] - ayw[k+1,k+1]*ayw[k, k:1]
    ## 用 Y_1, \dots, Y_{k+1} 预报 Y_{k+2} 的均方误差
    ss[k+2] <- ss[k+1] * (1 - ayw[k+1,k+1]^2)
  }

  list(coef.YW=ayw, sigmasq=ss)
}
```

对平稳列用 Levinson 计算一步预测系数 (Yule-Walker 系数) 进行一步预测的程序。设已知自协方差列。输入 x 是平稳时间序列观测数据 $X_1, \dots, X_n$。输入 gams 是理论自协方差列 $\gamma_0, \gamma_1, \dots, \gamma_{m-1}$, $m \le n$。输入 max.lag 表示最多使用多少个历史值预报。缺省使用所有历史值，或理论自协方差列个数减一。输入 end 表示计算一步预测到第 end 时刻为止，默认对每个时间点计算一步预测。输入 demean 选择预报时是否中心化。输入 conf 是计算一步预测区间时的置信度。

输出为一个列表，元素 x 是输入的原始时间序列，pred 是时间序列每个点的一步预报，长度与 x 相同。lb 和 ub 是每个点的预测下限和上限。

```
sts.pred.levinson <- function(x, gams, max.lag=length(gams)-1,
                              end=length(x),
                              demean=FALSE, conf=0.95){
  n <- length(x)
  m <- length(gams)
  if(max.lag > m-1){
    msg <- paste('可用最大历史长度', max.lag,
                 '大于可用理论自协方差个数减一', m-1)
    stop(msg)
  }
  if(demean) {
    xmean <- mean(x)
    x[] <- x - xmean
  }
```

```
  pred <- numeric(end)
  lb <- numeric(end)
  ub <- numeric(end)
  lam <- qnorm(1 - (1-conf)/2)

  res1 <- Levinson.coef(gams)
  ayw <- res1$coef.YW ## (m-1)*(m-1) 矩阵，第 k 行用于 k 个历史进行一步预报。
  ss <- res1$sigmasq  ## 长度为 m 的向量，第 k 个为用 k-1 个历史预报下一步的均方误差。

  pred[1] <- 0
  for(k in seq(end-1)){
    if(k <= max.lag) {
      pred[k+1] <- sum(ayw[k, 1:k] * x[k:1])
    } else {
      pred[k+1] <- sum(ayw[max.lag, 1:max.lag] * x[k:(k-max.lag+1)])
    }
  }
  if(demean) pred[] <- pred + xmean

  ss2 <- c(ss[1:min(end,max.lag+1)],
           rep(ss[max.lag+1], end-min(end,max.lag+1)))

  lb[] <- pred - lam*sqrt(ss2)
  ub[] <- pred + lam*sqrt(ss2)

  list(x=x, pred=pred, lb=lb, ub=ub, conf=conf)
}
```

周期图计算：

```
direct.periodogram <- function(
  y, freqs=seq(0,pi,length=101), demean=FALSE){
  N <- length(y)
  if(demean) y <- y - mean(y)
  II <- numeric(length(freqs))
  for(j in seq(along=freqs)){
    II[j] <- 1/(2*pi*N)*Mod(sum(y * complex(mod=1, arg=freqs[j]*seq(1,N))))^2
  }
```

```
  list(frequencies=freqs, spectrum=II)
}
```

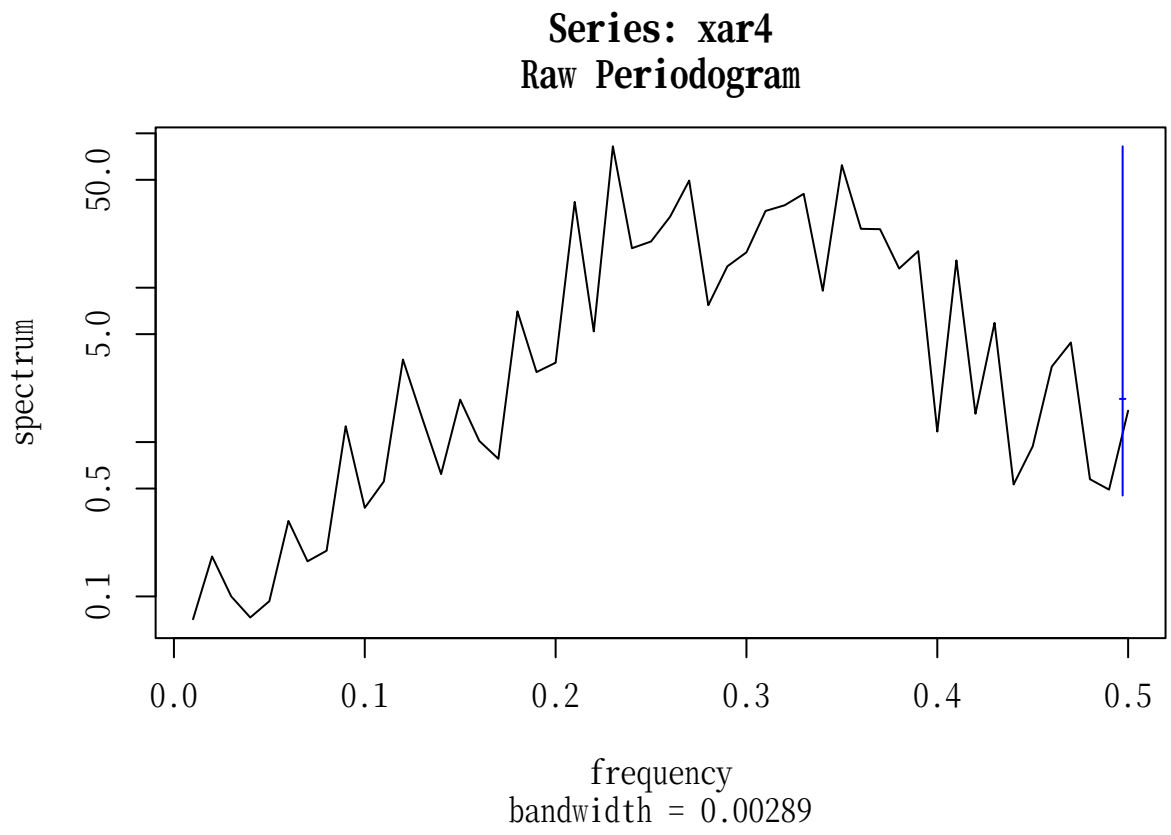
Daniell 加窗谱估计：

```
spec.Daniell <- function(
  y, freqs=seq(0,pi,length=101),
  M = round(2*sqrt(length(y))),
  demean=FALSE){
  N <- length(y)
  if(demean) y <- y - mean(y)
  spec <- numeric(length(freqs))
  gams <- c(acf(y, lag.max=N-1,
                type="covariance", plot=FALSE)$acf)
  jjs <- seq(N-1)
  for(j in seq(along=freqs))
    spec[j] <- 1/(2*pi)*(gams[1] +
                         2*sum(gams[jjs+1] *
                               ( sin(jjs*pi/M) / (jjs*pi/M) ) *
                               cos(jjs*freqs[j])))

  list(frequencies=freqs, spectrum=spec)
}
```

39.5 谱密度估计

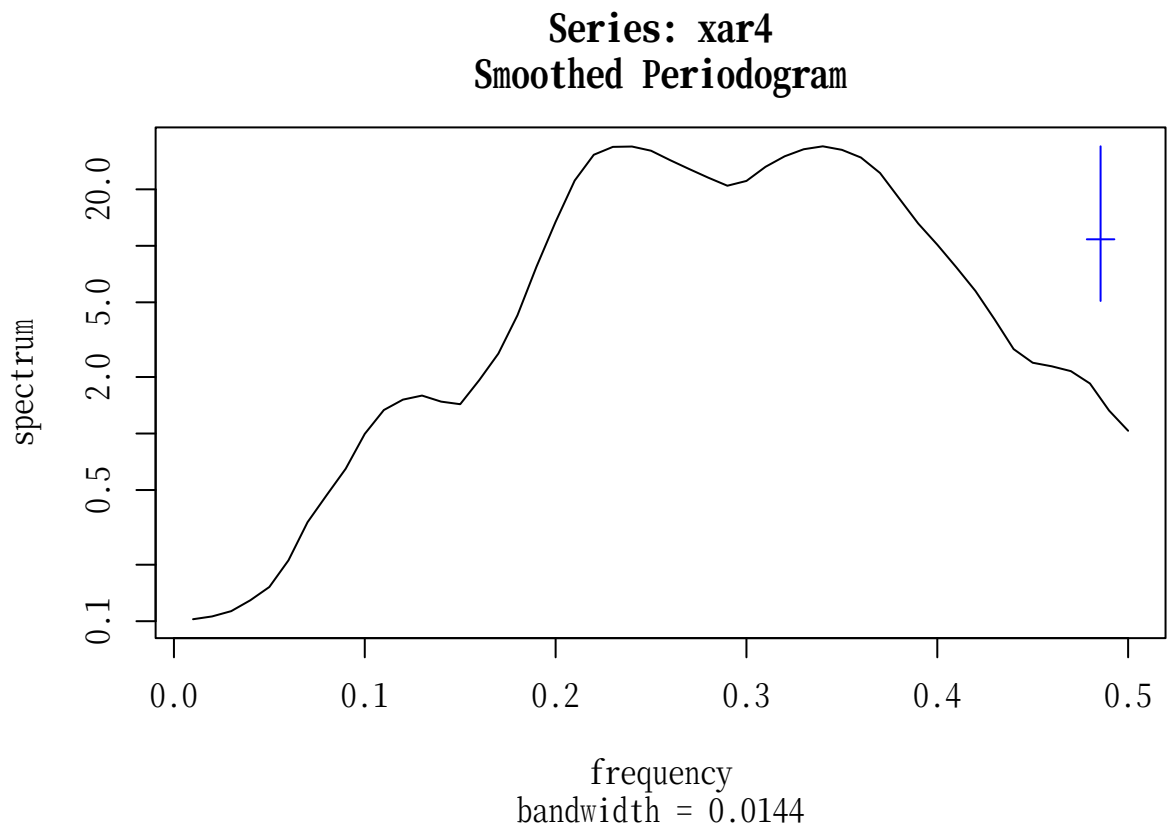
`spec.pgram(x)` 计算周期图。如：

```
spec.pgram(xar4)
```



`spec.pgram()` 可以使用 Daniel 谱窗作加窗谱估计，用 `spans` 给出两个奇数组成的向量作为参数，表示窗宽。如：

```
spec.pgram(xar4, spans=c(3,5))
```



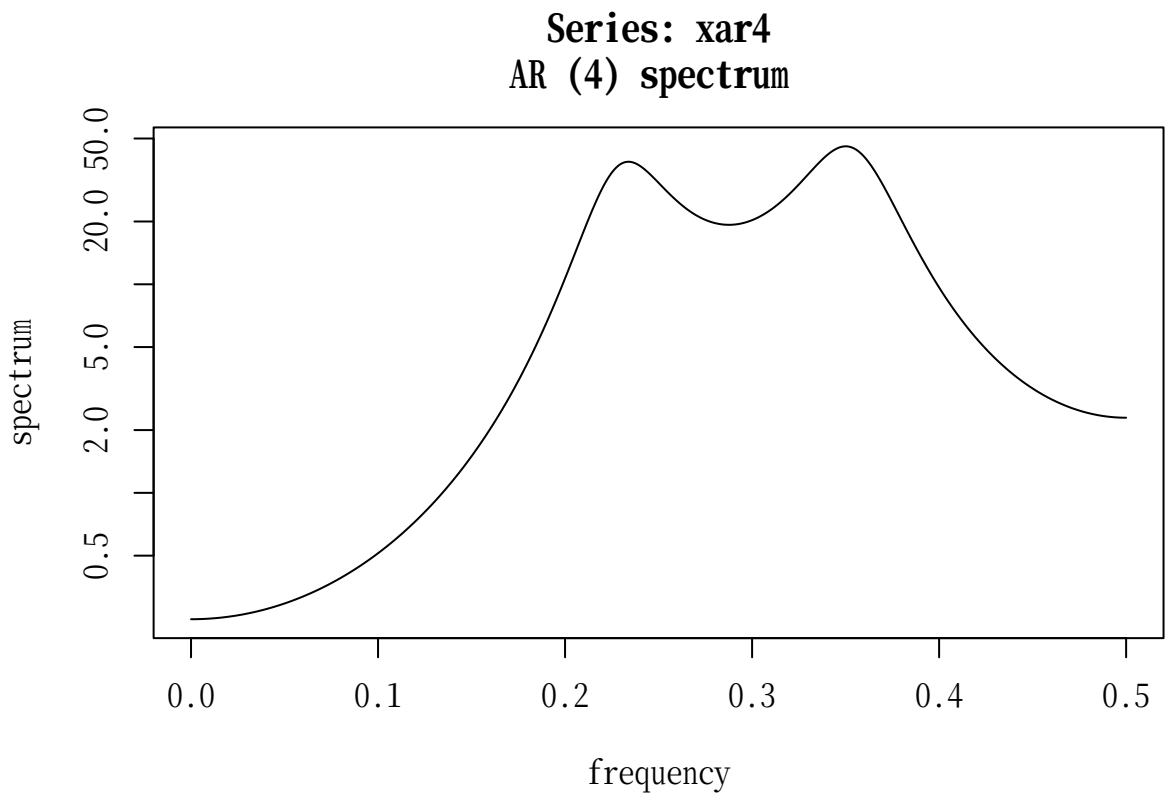
注意结果的横轴是频率而不是角频率，为了转换到角频率需要乘以 2π。输入的模拟 AR(4) 模型的两个谱峰频率为：

```
a <- c(-0.9, -1.4, -0.7, -0.6)
Arg(polyroot(c(1, -a)))/(2*pi)
```

```
## [1]  0.2370723  0.3511244 -0.3511244 -0.2370723
```

用 `spec.ar(x)` 可以做极大熵谱估计（用 AR 模型做谱估计）。如：

```
spec.ar(xar4)
```



因为数据是模拟的 AR(4)，估计的模型也是 AR(4)，所以效果很好。

39.6 GARCH 类模型

待完成。

39.7 多元时间序列基础统计

待完成。

39.8 VAR 模型

待完成。

39.9 协整分析

待完成。

39.10 因果性检验

待完成。

Chapter 40

随机模拟

40.1 随机数

随机模拟是统计研究的重要方法，另外许多现代统计计算方法（如 MCMC）也是基于随机模拟的。R 中提供了多种不同概率分布的随机数函数，可以批量地产生随机数。一些 R 扩展包利用了随机模拟方法，如 boot 包进行 bootstrap 估计。

所谓随机数，实际是“伪随机数”，是从一组起始值（称为**种子**），按照某种递推算法向前递推得到的。所以，从同一种子出发，得到的随机数序列是相同的。

为了得到可重现的结果，随机模拟应该从固定不变的种子开始模拟。用 `set.seed(k)` 指定一个编号为 k 的种子，这样每次从编号 k 种子运行相同的模拟程序可以得到相同的结果。

还可以用 `set.seed()` 加选项 kind= 指定后续程序要使用的随机数发生器名称，用 normal.kind= 指定要使用的正态分布随机数发生器名称。

R 提供了多种分布的随机数函数，如 `runif(n)` 产生 n 个标准均匀分布随机数，`rnorm(n)` 产生 n 个标准正态分布随机数。例如：

```
round(runif(5), 2)
## [1] 0.44 0.56 0.93 0.23 0.22
round(rnorm(5), 2)
## [1] -0.20  1.10 -0.02  0.16  2.02
```

注意因为没有指定种子，每次运行会得到不同的结果。

在 R 命令行运行

```
?Distributions
```

可以查看 R 中提供的不同概率分布。

40.2 sample() 函数

sample() 函数从一个有限集合中无放回或有放回地随机抽取，产生随机结果。

例如，为了设随机变量 X 取值于 $\{正面, 反面\}$, 且 $P(X = 正面) = 0.7 = 1 - P(X = 反面)$, 如下程序产生 X 的 10 个随机抽样值:

```
sample(c('正面', '反面'), size=10,
       prob=c(0.7, 0.3), replace=TRUE)
## [1] " 反面" " 反面" " 反面" " 反面" " 正面"
## [6] " 正面" " 正面" " 正面" " 反面" " 反面"
```

sample() 的选项 size 指定抽样个数，prob 指定每个值的概率，replace=TRUE 说明是有放回抽样。

如果要做无放回等概率的随机抽样，可以不指定 prob 和 replace(缺省是 FALSE)。比如，下面的程序从 1:10 随机抽取 4 个:

```
sample(1:10, size=4)
## [1]  1  5  8 10
```

如果要从 $1:n$ 中等概率无放回随机抽样直到每一个都被抽过，只要用如：

```
sample(10)
## [1]  3  5  9  2 10  7  4  1  6  8
```

这实际上返回了 $1:10$ 的一个重排。

dplyr 包的 sample_n() 函数与 sample() 类似，但输入为数据框，输出为随机抽取的数据框行子集。

40.3 随机模拟示例

40.3.1 估计期望值

设随机变量或随机向量 X 有复杂的分布，使得期望 $\theta = Eh(X)$ 很难计算。如果可以得到 X 的 N 个独立同分布抽样值 $X_1, X_2, \dots, X_N$，则可估计 θ 为：

$$\hat{\theta} = \frac{1}{N}\sum_{i=1}^{N} h(X_i),$$

这时 $E\hat{\theta} = \theta$，估计无偏；均方误差为

$$\begin{aligned}\mathrm{MSE} =& E|\hat{\theta} - \theta|^2 = \mathrm{Var}(\hat{\theta}) = \mathrm{SE}^2 \\ =& \frac{\mathrm{Var}(h(X))}{N} \approx \frac{S_N^2}{N},\end{aligned}$$

其中

$$S_N^2 = \frac{1}{N-1}\sum_{i=1}^{N}(h(X_i)-\hat{\theta})^2$$

是样本方差。由强大数律可知 $N \to \infty$ 时 $\hat{\theta}$ 以概率 1 收敛到 θ，由中心极限定理可知 N 充分大时 $\hat{\theta}$ 有近似正态分布

$$\mathrm{N}(\theta, \mathrm{SE}^2).$$

例如，考虑正方形区域 $\Omega = \{(x,y) : x \in [0,1], y \in [0,1]\}$，以及其中的四分之一扇形 $A = \{(x,y) : (x,y) \in \Omega, x^2+y^2 \le 1\}$。设 X 服从 Ω 上的均匀分布，令

$$Y = \begin{cases} 1, & 当 X \in A, \\ 0, & 其它 \end{cases}$$

则 Y 服从两点分布，概率为

$$p = EY = \frac{\frac{1}{4}\pi 1^2}{1^2} = \frac{\pi}{4}.$$

在 Ω 中投入 $N = 10000$ 个点，得到 N 个 $Y_i, i = 1, 2, \ldots, N$ 的值，估计 π 为

$$\hat{\pi} = 4\bar{Y} = \frac{4}{N}\sum_{i=1}^{N} Y_i.$$

$E\hat{\pi} = \pi$, 估计的根均方误差可估计为

$$\mathrm{RMSE} = \sqrt{E|\hat{\pi}-\pi|^2} = 4\frac{\sqrt{\mathrm{Var}(Y)}}{\sqrt{N}} \approx 4\frac{S_N}{\sqrt{N}}.$$

程序如下：

```
est_pi <- function(N){
  set.seed(101)
  x1 <- runif(N, 0, 1)
  x2 <- runif(N, 0, 1)
  y <- as.numeric(x1^2 + x2^2 <= 1)
  hat_pi <- 4*mean(y)
  se <- 4 * sd(y) / sqrt(N)
  cat("N = ", N, " pi 估计值 =", hat_pi, " SE =", se, "\n")

  invisible(list(N=N, hat_pi = hat_pi, SE = se))
}
est_pi(1E4)
## N =  10000  pi 估计值 = 3.14  SE = 0.01643372
```

估计的标准误差 (SE) 还是比较大，提高到 $N = 10^6$，精度可以增加一位小数：

```
est_pi(1E6)
## N =  1e+06  pi 估计值 = 3.145576  SE = 0.001639408
```

40.3.2 线性回归模拟

考虑如下线性回归模型

$$y = 10 + 2x + \varepsilon,\ \varepsilon \sim \mathrm{N}(0, 0.5^2).$$

假设有样本量 $n = 10$ 的一组样本，R 函数 `lm()` 可以可以得到截距 a, 斜率 b 的估计 $\hat{a}, \hat{b}$, 以及相应的标准误差 $\mathrm{SE}(\hat{a})$, $\mathrm{SE}(\hat{b})$。样本可以模拟产生。

模型中的自变量 x 可以用随机数产生，比如，用 `sample()` 函数从 $1:10$ 中随机有放回地抽取 n 个。模型中的随机误差项 ε 可以用 `rnorm()` 产生。产生一组样本的程序如:

```
n <- 10; a <- 10; b <- 2
x <- sample(1:10, size=n, replace=TRUE)
eps <- rnorm(n, 0, 0.5)
y <- a + b * x + eps
```

如下程序计算线性回归:

```
lm(y ~ x)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##       9.981        1.987
```

如下程序计算线性回归的多种统计量:

```
summary(lm(y ~ x))
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -0.49469 -0.24265 -0.09845  0.15165  0.82853
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   9.9808     0.3119   32.00 9.91e-10 ***
## x             1.9865     0.0548   36.25 3.67e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4012 on 8 degrees of freedom
## Multiple R-squared:  0.9939, Adjusted R-squared:  0.9932
## F-statistic:  1314 on 1 and 8 DF,  p-value: 3.673e-10
```

如下程序返回一个矩阵，包括 a, b 的估计值、标准误差、t 检验统计量、检验 p 值:

```r
summary(lm(y ~ x))$coefficients
```

```
##             Estimate Std. Error  t value     Pr(>|t|)
## (Intercept) 9.980756 0.31190678 31.99916 9.906964e-10
## x           1.986528 0.05479644 36.25288 3.672736e-10
```

如下程序把上述矩阵的前两列拉直成一个向量返回：

```r
c(summary(lm(y ~ x))$coefficients[,1:2])
```

```
## [1] 9.98075574 1.98652843 0.31190678 0.05479644
```

这样得到 $\hat{a}, \hat{b}, \mathrm{SE}(\hat{a}), \mathrm{SE}(\hat{b})$ 这四个值。

用 `replicate`(, 复合语句) 执行多次模拟，返回向量或矩阵结果，返回矩阵时，每列是一次模拟的结果。下面是线性回归整个模拟程序，写成了一个函数。

```r
reg.sim <- function(
  a=10, b=2, sigma=0.5,
  n=10, B=1000){
  set.seed(1)
  resm <- replicate(B, {
      x <- sample(1:10, size=n, replace=TRUE)
      eps <- rnorm(n, 0, 0.5)
      y <- a + b * x + eps
      c(summary(lm(y ~ x))$coefficients[,1:2])
```

```
  })
  resm <- t(resm)
  colnames(resm) <- c('a', 'b', 'SE.a', 'SE.b')
  cat(B, '次模拟的平均值:\n')
  print( apply(resm, 2, mean) )
  cat(B, '次模拟的标准差:\n')
  print( apply(resm, 2, sd) )
}
```

运行测试：

```
set.seed(1)
reg.sim()
```

```
## 1000 次模拟的平均值:
##         a         b      SE.a      SE.b
## 9.9970476 1.9994490 0.3639505 0.0592510
## 1000 次模拟的标准差:
##          a          b       SE.a       SE.b
## 0.37974881 0.06297733 0.11992515 0.01795926
```

可以看出，标准误差作为 $\hat{a}, \hat{b}$ 的标准差估计，与多次模拟得到多个 $\hat{a}, \hat{b}$ 样本计算得到的标准差估计是比较接近的。结果中 $\mathrm{SE}(\hat{a})$ 的平均值为 0.364, 1000 次模拟的 $\hat{a}$ 的样本标准差为 0.380，比较接近；$\mathrm{SE}(\hat{b})$ 的平均值为 0.0593, 1000 次模拟的 $\hat{b}$ 的样本标准差为 0.0630，比较接近。

40.4 bootstrap

随机模拟研究中，可以从理论模型生成许多组独立样本，对每一组样本进行估计，最后评估估计效果。

实际问题中，仅有一组样本，能否利用随机模拟方法评估对这组样本的估计效果？bootstrap 就是这样一种方法，对原始样本进行“再抽样”，得到与原始样本分布相近的许多组样本，称为 bootstrap 样本，对每一个 bootstrap 样本进行估计，用多个样本的估计结果来评估估计效果。

再抽样分为非参数和参数两种方法，其中非参数 bootstrap 对原始数据集进行多次（如 $B = 1000$）重复有放回抽样，每次得到一个同样大小的 bootstrap 数据集，对每个 bootstrap 数据集进行相同的估计过程，用得到的多次估计结果评估参数估计的精度、分布等。参数方法是对原始数据集利用参数方法建模，然后把估计出来的参数当作真实模型参数产生新的多组样本。

40.4.1 boot 扩展包

基本 R 内嵌的 boot 包可以用来对数据集进行 bootstrap 抽样，输入数据集和对数据集进行处理的函数，直接输出对多个 bootstrap 数据集分析的结果。

作为示例，考虑肺癌放疗研究中放疗前后的肿瘤大小 (v0, v1) 的线性回归模型估计参数 $\hat{\beta}_0, \hat{\beta}_1$ 的评估问题。得到的回归方程是：

$$v_1 \approx \hat{\beta}_0 + \hat{\beta}_1 v_0.$$

```
library(tidyverse)
library(boot)
d.cancer <- read_csv(
  "data/cancer.csv",
  locale=locale(encoding="GBK"),
  show_col_types=FALSE)
fa <- function(d, inds){
  dd <- d[inds,]
  lm1 <- lm(v1 ~ v0, data=dd)
  unname(coef(lm1))
}

lm0 <- lm(v1 ~ v0, data=d.cancer)
print(summary(lm0))
```

```
##
## Call:
## lm(formula = v1 ~ v0, data = d.cancer)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -61.536 -10.582  -5.011  11.750  58.621
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.33079    7.26335   0.459     0.65
## v0           0.37576    0.05376   6.990  6.4e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 24.56 on 32 degrees of freedom
## Multiple R-squared:  0.6042, Adjusted R-squared:  0.5919
```

```
## F-statistic: 48.86 on 1 and 32 DF,  p-value: 6.401e-08
```

```
set.seed(101)
btr <- boot(d.cancer, fa, R=1000)
cat("Intercept: bias = ", btr$t0[1] - mean(btr$t[,1]),
    "  SE = ", sd(btr$t[,1]), "\n")
```

```
## Intercept: bias =  0.2877853   SE =  4.78422
```

```
cat("v0 slope: bias = ", btr$t0[2] - mean(btr$t[,2]),
    "  SE = ", sd(btr$t[,2]), "\n")
```

```
## v0 slope: bias =  -0.003612172   SE =  0.05450733
```

boot() 函数的第一个自变量是 data，为原始数据集，第二个自变量是 statistic，是统计估计函数，该函数第一个自变量为原始数据，第二个自变量是每次重抽样的下标向量，R 是重复抽样次数，函数输出要考察的参数估计。

boot() 的结果（上面程序中的 btr）是一个列表，列表元素 t0 是对原始数据集的参数估计，而 t 是对每一个重抽样数据集的参数估计，每个重抽样的数据集占据矩阵 t 的一行，每个估计量占据一列。

在这个例子中，bootstrap 对斜率项的参数估计的标准误差估计与参数模型结果相近，但对截距项的标准误差估计与参数模型结果相差较大。还利用 bootstrap 方法估计了偏度，但是注意线性回归模型的最小二乘估计在理论上是无偏估计。

利用得到的 bootstrap 样本计算 β_0 和 β_1 的置信区间：

```
ci_b0 <- quantile(btr$t[,1], c(0.025, 0.975))
ci_b1 <- quantile(btr$t[,2], c(0.025, 0.975))
cat("95% CI for beta0: (",
    ci_b0[1], ", ", ci_b0[2], ")\n")
```

```
## 95% CI for beta0: ( -5.944525 ,  12.44923 )
```

```
cat("95% CI for beta1: (",
    ci_b1[1], ", ", ci_b1[2], ")\n")
```

```
## 95% CI for beta1: ( 0.277491 ,  0.4920218 )
```

40.4.2 rsample 扩展包

boot 扩展包的功能比较固定，另一个 rsample 扩展包则与函数式编程风格比较接近，可以比较灵活地进行各种重抽样，包括 bootstrap、交叉验证、重要性抽样等。

bootstraps 函数输入一个数据集，输出多个 bootstrap 数据集并将其存放在一个 tibble 数据框的列表类型的列 splits 中，每个元素是 rsplit 类型，可以用 as.data.frame() 或者 analysis() 函数将其转换成数据框。

利用 tibble 的列表列，我们可以实现管道形式的 bootstrap 分析。如：

```
library(rsample)
fb <- function(dd){
  lm1 <- lm(v1 ~ v0, data=dd)
  as_tibble(rbind(coef(lm1)))
}

btr2 <- d.cancer |>
  bootstraps(times=1000) |>
  pull(splits) |>
  map_dfr( \(df) fb(analysis(df)) )

cat("Intercept: bias = ", coef(lm0)[1] - mean(btr2[[1]]),
    "  SE = ", sd(btr2[[1]]), "\n")
```

```
## Intercept: bias =  0.1611144   SE =  4.905335
```

```
cat("v0 slope: bias = ", coef(lm0)[2] - mean(btr2[[2]]),
    "  SE = ", sd(btr2[[2]]), "\n")
```

```
## v0 slope: bias =  -0.001650511   SE =  0.05649049
```

```
ci_b0 <- quantile(btr2[[1]], c(0.025, 0.975))
ci_b1 <- quantile(btr2[[2]], c(0.025, 0.975))
cat("95% CI for beta0: (",
    ci_b0[1], ", ", ci_b0[2], ")\n")
```

```
## 95% CI for beta0: ( -6.250138 ,  12.52947 )
```

```
cat("95% CI for beta1: (",
    ci_b1[1], ", ", ci_b1[2], ")\n")
```

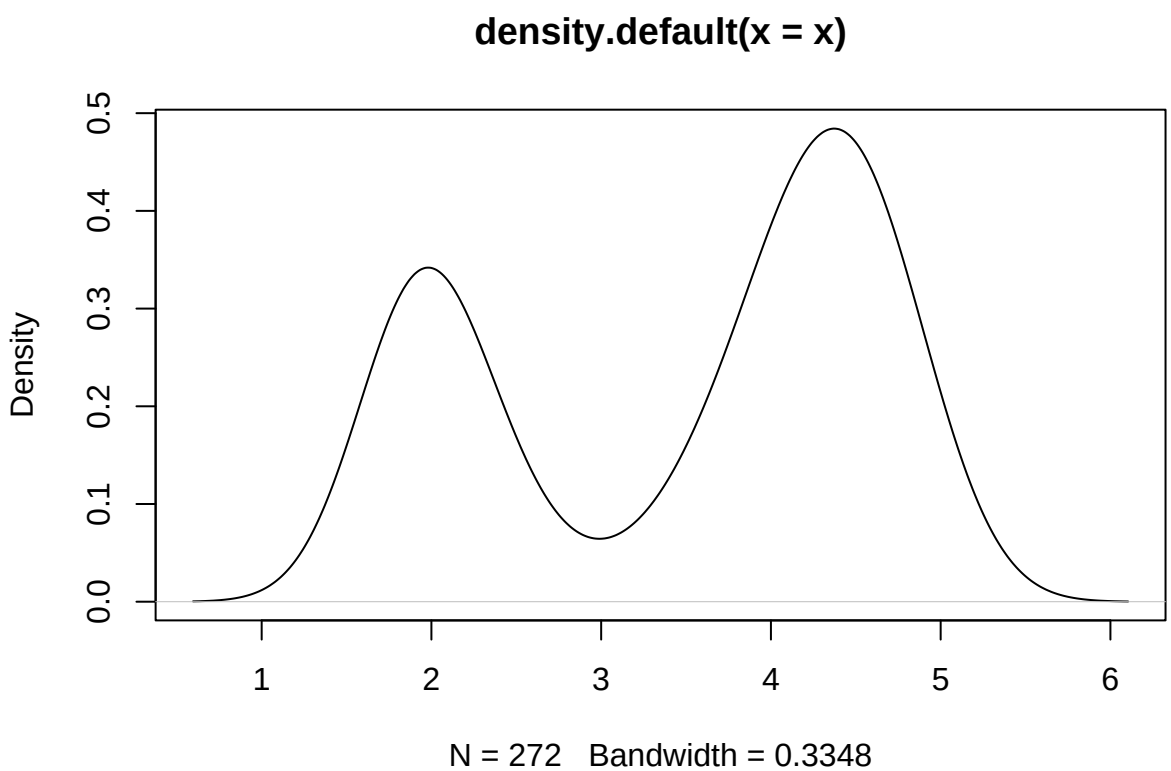
```
## 95% CI for beta1: ( 0.2667938 ,  0.4824034 )
```

结果的 btr2 是一个两列的数据框，每一行包含对一个重抽样数据集的两个回归系数估计。

40.4.3 例：核密度的 bootstrap 置信区间

R 自带的数据框 faithful 内保存了美国黄石国家公园 Faithful 火山的 272 次爆发持续时间和间歇时间。为估计爆发持续时间的密度，可以用核密度估计方法，R 函数 `density` 可以执行此估计，返回 N 个格子点上的密度曲线坐标:

```
x <- faithful$eruptions
est0 <- density(x)
plot(est0)
```



这个密度估计明显呈现出双峰形态。

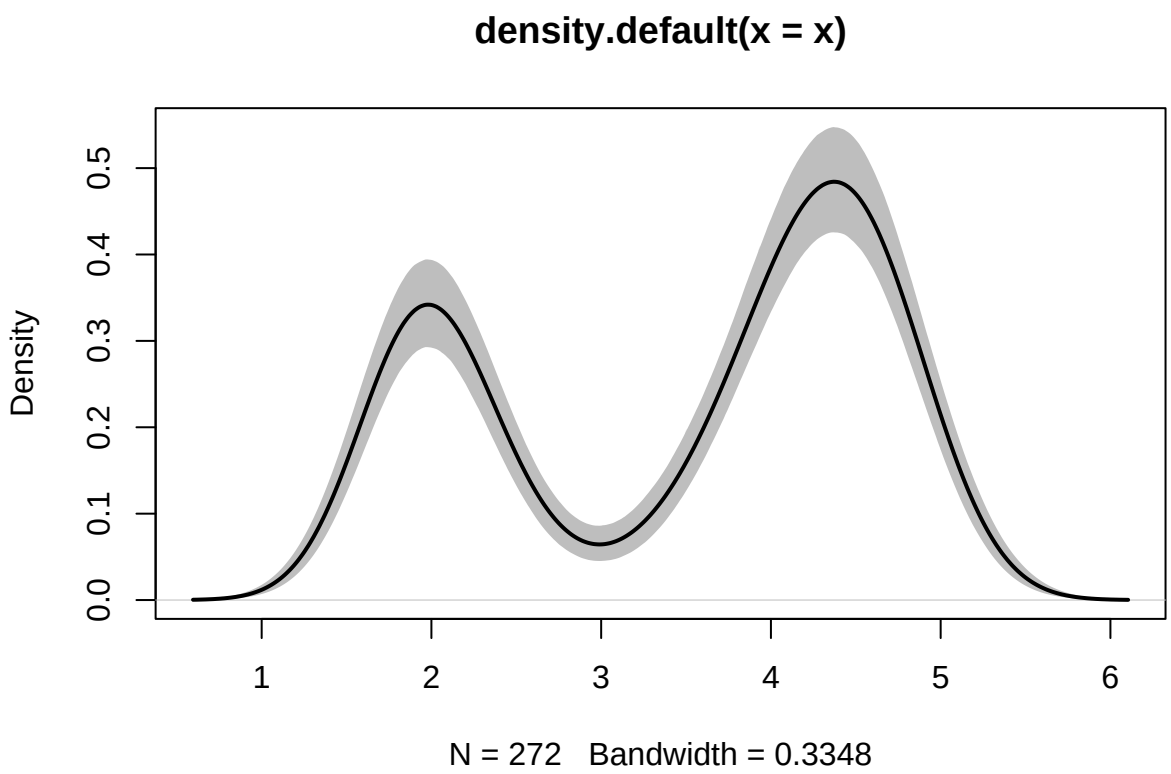
核密度估计是统计估计，为了得到其置信区间（给定每个 x 坐标，真实密度 $f(x)$ 的单点的置信区间），采用如下非参数 bootstrap 方法：

重复 $B = 10000$ 次，每次从原始样本中有重复地抽取与原来大小相同的一组样本，对这组样本计算核密度估计，结果为 $(x_i, y_i^{(j)}), i = 1, 2, \ldots, N, j = 1, 2, \ldots, B$，每组样本估计 N 个格子点的密度曲线坐标, 横坐标不随样本改变。

对每个横坐标 x_i，取 bootstrap 得到的 B 个 $y_i^{(j)}, j = 1, 2, \ldots, B$ 的 0.025 和 0.975 样本分位数，作为真实密度 $f(x_i)$ 的 bootstrap 置信区间。

在 R 中利用 `replicate()` 函数实现：

```
set.seed(1)
resm <- replicate(10000, {
    x1 <- sample(x, replace=TRUE)
    density(x1, from=min(est0$x),
            to=max(est0$x))$y
})
CI <- apply(resm, 1, quantile, c(0.025, 0.975))
plot(est0, ylim=range(CI), type='n')
polygon(c(est0$x, rev(est0$x)),
        c(CI[1,], rev(CI[2,])),
        col='grey', border=FALSE)
lines(est0, lwd=2)
```



程序中用 `set.seed(1)` 保证每次运行得到的结果是不变的，`replicate()` 函数第一参数是重复模拟次数，第二参数是复合语句，这些语句是每次模拟要执行的计算。在每次模拟中，用带有 `replace=TRUE` 选项的 `sample()` 函数从样本中有放回地抽样得到一组 bootstrap 样本，每次模拟的结果是在指定格子点上计算的核密度估计的纵坐标。`replicate()` 的结果为一个矩阵，每一列是一次模拟得到的纵坐标集合。对每个横坐标格子点，用 `quantile()` 函数计算 B 个 bootstrap 样本的 2.5% 和 97.5% 分位数，循环用 `apply()` 函数表示。`polygon()` 函数指定一个多边形的顺序的顶点坐标用 `col=` 指定的颜色填充，本程序中实现了置信下限与置信上限两条曲线之间的颜色填

充。lines() 函数绘制了与原始样本对应的核密度估计曲线。

40.5 附录：完全试验方案设计函数

最简单的模拟试验设计，是将考虑的所有因素的所有水平组合都执行，比如，分别有 2、3、4 个水平的三个因素要进行完全试验，需要 $2 \times 3 \times 4 = 24$ 个方法。下面的函数可以用来产生这样的试验方案，输出为一个数据框，数据框的每一行是一次试验的方法。

tidyr 包的 expand_grid() 函数可以实现这样的功能，基本 R 的 expand.grid() 函数有类似功能。这两个函数输入因子，产生完全试验组合。参见23.17。

下面的程序直接实现了略强一些的功能，支持以数据框为输入。

```
## 生成完全试验设计方案的函数。
## 输入:
##   varlist: 列表，元素可以是正整数、向量或数据框，
##     每个元素代表一个单纯的或者组合的因素。
##     元素取正整数时，表示仅用编号表示的因素的水平数。
##     元素取数值型或字符型向量时，表示因素的各个水平;
##     元素取多列的数据框时，每一行是一种因素组合，
##     数据框的每一列仍看作一个因素，但只允许数据框的各行出现的因素组合。
##   numbering: 真值表示输出中增加一列 expNumber 表示互不相同的试验方案的编号，
##     假值则不增加这一列。
##   size: 每种试验方案重复多少行。
## 输出: 一个 R 数据框，每一行代表一次试验，每一列代表一个因素取值。
complete.design <- function(
  varlist,
  numbering=TRUE,  # 是否为试验方案编号
  size=1){         # 每个方案的重复次数
  nf <- length(varlist)
  vl <- vector(mode="list", length=nf)
  tf <- character(nf)
  for(i in seq_along(varlist)){
    if(length(varlist[[i]])==1){
      if(varlist[[i]]==as.integer(varlist[[i]]) && varlist[[i]] > 0){
        # 用一个正整数表示的因子，就只需要用编码表示各个水平
        vl[[i]] <- factor(seq(varlist[[i]]))
        tf[i] <- "factor"
      } else {
        stop(" 非法输入! ")
```

```
    }
  } else if(is.data.frame(varlist[[i]])){
    # 数据框，可以有一到多列，但是不可拆分，看作固定的组合
    vl[[i]] <- seq(nrow(varlist[[i]]))
    tf[i] <- "df"
  } else {
    # 普通向量，元素为因素水平
    stopifnot(!any(duplicated(varlist[[i]])))
    vl[[i]] <- factor(varlist[[i]])
    tf[i] <- "factor"
  }
}

## 每个因素的水平数
nlev <- vapply(vl, length, 1)
## 总试验次数
ne <- prod(nlev) * size
## 每个因子的重复数
nr <- rev(cumprod(rev(c(nlev, size))))[-1]
## 每个因子的循环次数
nc <- ne / nr

## 产生作为数据集草稿的列表
dl <- list()
for(i in seq_along(vl)){
  idl <- as.integer(gl(nlev[i], nr[i], ne))
  if(tf[i]=="factor"){
    ## 单个因素
    dl[[i]] <- data.frame(
      factor(idl, labels=levels(vl[[i]])))
    names(dl[[i]]) <- names(varlist)[[i]]
  } else {
    ## 数据框，要恢复数据框
    dl[[i]] <- varlist[[i]][idl,]
  }
}

## 合并数据框得到最终结果
d <- dl[[1]]
```

```
  if(nf>1){
    for(i in seq(2, nf)){
      d <- cbind(d, dl[[i]])
    }
  }

  if(numbering){
    d <- cbind(d, data.frame(
      expNumber = rep(seq(ne/size), each=size)))
  }

  d
}


## 测试程序
test1 <- function(){
  complete.design(list(f1=3, f2=c("a", "b"), f3=4))
}
print(test1())

test2 <- function(){
  complete.design(list(f1=3, f2=c("a", "b"), f3=4), size=2)
}
print(test2())

test3 <- function(){
  complete.design(list(f1=3, f2=c("a", "b"), f3=data.frame(
    fa=c(0.01, 0.05, 0.1), fb = c("x", "x", "y"))), size=2)
}
print(test3())
```

Part VIII

特殊应用

Chapter 41

R 语言的文本处理

41.1 介绍

在信息爆炸性增长的今天，大量的信息是文本型的，如互联网上的大多数资源。R 具有基本的文本数据处理能力，而且因为 R 的向量语言特点和强大的统计计算和图形功能，用 R 处理文本数据是可行的。

41.2 字符型常量与字符型向量

字符串常量写在两个双撇号或者两个单撇号中间，建议仅使用双撇号，因为这是大多数常见程序语言的做法。如果内容中有单撇号或者双撇号，可以在前面加反斜杠\。为了在字符串中写一个反斜杠，需要写成两个，比如路径 `C:\work` 写成 R 字符串，要写成`"C:\\work"`。注意，这些规定都是针对程序中的字符串常量，数据中的文本类型数据是不需要遵照这些规定的。

在用 `print()` 显示字符串变量时，也会按照上述的办法显示，比如字符串内的双撇号会被自动加上前导反斜杠，但保存的实际内容中并没有反斜杠。

字符串中可以有一些特殊字符，如`"\n"` 表示换行符，`"\t"` 表示制表符，`"\r"` 表示回车符，等等。

R 的字符型向量每个元素是一个字符串，如：

```
s <- c("123", "abc", " 张三李四", "@#$%^&")
s
```

```
## [1] "123"      "abc"      "张三李四" "@#$%^&"
```

R 中处理文本型数据的函数有文件访问函数以及 `readLines`，`nchar`，`paste`，`sprintf`，`format`，`formatC`，`substring` 等函数。

R 支持正则表达式，函数 `grep`, `grepl`, `sub`, `gsub`, `regexpr`, `gregexpr`, `strsplit` 与正则表达式有关。

字符型函数一般都是向量化的，对输入的一个字符型向量的每个元素操作。

R 扩展包 stringr 和 stringi 提供了更方便、功能更强的字符串功能，包括正则表达式功能。其中 stringr 是常用功能，stringi 是更基本、更灵活的功能，一般使用 stringr 就足够了。stringr 包的函数名大多都以 `str_` 开头。

下面先介绍常用的较简单的字符串函数，包括 stringr 包的函数与基本 R 函数。

```
library(stringr)
```

41.3 字符串连接、重复

`stringr::str_c()` 用来把多个输入自变量按照元素对应组合为一个字符型向量，用 `sep` 指定分隔符，默认为不分隔。类似于 R 中向量间运算的一般规则，各自变量长度不同时短的自动循环使用。非字符串类型自动转换为字符型。如

```
str_c(c("x", "y"), c("a", "b"), sep="*")
```

```
## [1] "x*a" "y*b"
```

```
str_c("data", 1:3, ".txt")
```

```
## [1] "data1.txt" "data2.txt" "data3.txt"
```

字符型缺失值参与连接时，结果变成缺失值；可以用 `str_replace_na()` 函数将待连接的字符型向量中的缺失值转换成字符串`"NA"` 再连接。

用 `collapse` 选项要求将连接后的字符型向量的所有元素连接在一起，`collapse` 的值为将多个元素合并时的分隔符。如

```
str_c(c("a", "bc", "def"), collapse="---")
```

```
## [1] "a---bc---def"
```

在使用了 `collapse` 时如果有多个要连接的部分，`str_c()` 函数先将各部分连接成为一个字符型向量，然后再把结果的各个向量元素连接起来。如

```
str_c("data", 1:3, ".txt", sep="", collapse=";")
```

```
## [1] "data1.txt;data2.txt;data3.txt"
```

`stringr::str_flatten()` 类似于 `stringr::str_c()` 仅有 `collapse` 参数作用一样，仅将一个字符型向量的各个元素按照 `collapse` 参数指定的分隔符连接成一个长字符串，`collapse` 默认值是空字符串，如：

```
str_flatten(c("a", "bc", "def"), collapse="---")
```

```
## [1] "a---bc---def"
```

```
str_flatten(c("a", "bc", "def"))
```

```
## [1] "abcdef"
```

基本 R 的 paste() 函数与 stringr::str_c() 函数有类似的用法，但是参数 sep 的默认值是空格。基本 R 的 paste0() 函数相当于 stringr::str_c() 函数固定 sep 参数为空字符串。如：

```
paste(c("x", "y"), c("a", "b"), sep="*")
```

```
## [1] "x*a" "y*b"
```

```
paste("data", 1:3, ".txt", sep="")
```

```
## [1] "data1.txt" "data2.txt" "data3.txt"
```

```
paste0("data", 1:3, ".txt")
```

```
## [1] "data1.txt" "data2.txt" "data3.txt"
```

```
paste(c("a", "bc", "def"), collapse="---")
```

```
## [1] "a---bc---def"
```

```
paste("data", 1:3, ".txt", sep="", collapse=";")
```

```
## [1] "data1.txt;data2.txt;data3.txt"
```

stringr::str_dup(string, times) 类似于 rep() 函数，可以将字符型向量的元素按照 times 指定的次数在同一字符串内重复，如：

```
str_dup(c("abc", " 长江"), 3)
```

```
## [1] "abcabcabc"    "长江长江长江"
```

也可以针对每个元素指定不同重复次数，如

```
str_dup(c("abc", " 长江"), c(3, 2))
```

```
## [1] "abcabcabc" "长江长江"
```

41.4 格式化输出

41.4.1 format() 函数

format() 函数可以将一个数值型向量的各个元素按照统一格式转换为字符型，如：

```
as.character(1.000)
```

```
## [1] "1"
```

```
as.character(1.2)
```

```
## [1] "1.2"
```

```
as.character(1.23)
```

```
## [1] "1.23"
```

```
format(c(1.000, 1.2, 1.23))
```

```
## [1] "1.00" "1.20" "1.23"
```

选项 digits 与 nsmall 共同控制输出的精度，nsmall 控制非科学记数法显示时小数点后的至少要有的位数，digits 控制至少要有的有效位数。这使得输出的宽度是不可控的，如：

```
format(c(pi, pi*10000), digits=8, nsmall=4)
```

```
## [1] "    3.1415927" "31415.9265359"
```

width 参数指定至少要有的输出宽度，不足时默认在左侧用空格填充，如：

```
format(1.000, width=6, nsmall=2)
```

```
## [1] "  1.00"
```

format() 还有许多选项，详见函数的帮助。

41.4.2 sprintf() 函数

format() 函数无法精确控制输出长度和格式。sprintf 是 C 语言中 sprintf 的向量化版本，可以把一个元素或一个向量的各个元素按照 C 语言输出格式转换为字符型向量。第一个自变量是 C 语言格式的输出格式字符串，其中%d 表示输出整数，%f 表示输出实数，%02d 表示输出宽度为 2、不够左填 0 的整数，%6.2f 表示输出宽度为 6、宽度不足时左填空格、含两位小数的实数，等等。

比如，标量转换

```
sprintf("%6.2f", pi)
```

```
## [1] "  3.14"
```

又如，向量转换：

```
sprintf("tour%03d.jpg", c(1, 5, 10, 15, 100))
```

```
## [1] "tour001.jpg" "tour005.jpg" "tour010.jpg" "tour015.jpg" "tour100.jpg"
```

还可以支持多个向量同时转换，如：

```
sprintf("%1dx%1d=%2d", 1:5, 5:1, (1:5)*(5:1))
```

```
## [1] "1x5= 5" "2x4= 8" "3x3= 9" "4x2= 8" "5x1= 5"
```

41.4.3 字符串插值函数

许多脚本型程序设计语言都有在字符串的内容中插入变量值的功能，R 本身不具有这样的功能，sprintf() 函数有类似作用但只是一个不方便使用的副作用。

stringr::str_glue() 和 stringr::str_glue_data() 提供了字符串插值的功能。只要在字符串内用大括号写变量名，则函数可以将字符串内容中的变量名替换成变量值，如：

```
name <- " 李明"
tele <- "13512345678"
str_glue(" 姓名: {name}\n电话号码: {tele}\n")
```

```
## 姓名: 李明
## 电话号码: 13512345678
```

上面的例子直接用了换行符"\n" 来分开不同内容。也可以输入多个字符串作为自变量，内容自动连接在一起，可以用参数.sep 指定分隔符：

```
name <- " 李明"
tele <- "13512345678"
str_glue(" 姓名: {name}, ", " 电话号码: {tele}")
```

```
## 姓名: 李明, 电话号码: 13512345678
```

```
str_glue(" 姓名: {name}", " 电话号码: {tele}", .sep="; ")
```

```
## 姓名: 李明; 电话号码: 13512345678
```

也可以直接在 str_glue() 中指定变量值，如：

```
str_glue(" 姓名: {name}", " 电话号码: {tele}", .sep="; ",
         name = " 张三", tele = "13588888888")
```

```
## 姓名: 张三; 电话号码: 13588888888
```

stringr::str_glue_data() 则以一个包含变量定义的对象.x 为第一自变量，类型可以是环境、列表、数据框等。如：

```
str_glue_data(list(name = " 王五", tele = "13500000000"),
              " 姓名: {name}", " 电话号码: {tele}", .sep="; ")
```

```
## 姓名: 王五; 电话号码: 13500000000
```

需要插入原样的大括号时，可以双写，如："{{...}}"。

41.5 字符串长度

stringr::str_length(string) 求字符型向量 string 每个元素的长度。一个汉字长度为 1。

```
str_length(c("a", "bc", "def", " 北京"))
```

```
## [1] 1 2 3 2
```

函数 nchar(text) 计算字符串长度，默认按照字符个数计算而不是按字节数计算，如

```
nchar(c("a", "bc", "def", " 北京"))
```

```
## [1] 1 2 3 2
```

注意函数对输入的字符型向量每个元素计算长度。

nchar() 加选项 type="bytes" 可用按字符串占用的字节数计算，这时一个汉字占用多个字节（具体占用多少与编码有关）。如

```
nchar(c("a", "bc", "def", " 北京"), type="bytes")
```

```
## [1] 1 2 3 6
```

41.6 取子串

stringr::str_sub(string, start, end) 字符串字串，用开始字符位置 start 和结束字符位置 end 设定字串位置。用负数表示倒数位置。默认开始位置为 1，默认结束位置为最后一个字符。

如：

```
str_sub("term2017", 5, 8)
```

```
## [1] "2017"
```

```
str_sub(c("term2017", "term2018"), 5, 8)
```

```
## [1] "2017" "2018"
```

```
str_sub("term2017", 5)
```

```
## [1] "2017"
```

```
str_sub("term2017", -4, -1)
```

```
## [1] "2017"
```

```
str_sub("term2017", end=4)
```

```
## [1] "term"
```

取子串时，一般按照字符个数计算位置，如

```
str_sub(" 北京市海淀区颐和园路 5 号", 4, 6)
```

```
## [1] "海淀区"
```

当起始位置超过总长度或结束位置超过第一个字符时返回空字符串；当起始位置超过结束位置是返回空字符串。如：

```
str_sub("term2017", 9)
```

```
## [1] ""
```

```
str_sub("term2017", 1, -9)
```

```
## [1] ""
```

```
str_sub("term2017", 8, 5)
```

```
## [1] ""
```

可以对 str_sub() 结果赋值，表示修改子串内容，如：

```
s <- "term2017"
str_sub(s, 5, 8) <- "18"
s
```

```
## [1] "term18"
```

字符串替换一般还是应该使用专用的替换函数如 stringr::str_replace_all()，gsub()。

基本 R 的 substring(text, first, last) 函数与 stringr::str_sub() 功能相同，但 first 和 last 参数不允许用负数, last 的默认值是一个很大的数，所以省略 last 时会取到字符串末尾。substring() 对三个参数 text, first, last 都是向量化的，长度不一致时按照一般的不等长向量间运算规则处理。如：

```
substring(c("term2017", "term2018"), first=c(1, 5), last=c(4, 8))
```

```
## [1] "term" "2018"
```

```
substring("term2017", first=c(1, 5), last=c(4, 8))
```

```
## [1] "term" "2017"
```

substring() 也允许修改某个字符串的指定子串的内容，如

```
s <- "123456789"
substring(s, 3, 5) <- "abc"
s
```

```
## [1] "12abc6789"
```

R 的 substr(x, start, stop) 作用类似，但是仅支持 x 为字符型向量，start 和 stop 是标量。

41.7 字符串变换

41.7.1 大小写

stringr::str_to_upper(string) 将字符型向量 string 中的英文字母都转换为大写。类似函数有 stringr::str_to_lower 转换为小写,stringr::str_to_title(string) 转换为标题需要的大小写,stringr::str_to_scentence(string) 转换为句子需要的大小写。这都是针对英文的，选项 locale 用来选语言，locale="en" 为默认值。

基本 R 的 toupper() 将字符型向量的每个元素中的小写字母转换为大写, tolower() 转小写。

41.7.2 字符变换表

基本 R 的 chartr(old, new, x) 函数指定一个字符对应关系，旧字符在 old 中，新字符在 new 中，x 是一个要进行替换的字符型向量。比如，下面的例子把所有! 替换成.，把所有; 替换成,:

```
chartr("!;", ".,", c("Hi; boy!", "How do you do!"))
```

```
## [1] "Hi, boy."       "How do you do."
```

```
chartr("。,; 县", ".,; 区", " 昌平县，大兴县；固安县。")
```

```
## [1] "昌平区,大兴区;固安区."
```

第二个例子中被替换的标点是中文标点，替换成了相应的英文标点。

41.7.3 空白处理

stringr::str_trim(string, side) 返回删去字符型向量 string 每个元素的首尾空格的结果，可以用 side 指定删除首尾空格（"both"）、开头空格（"left"）、末尾空格（"right"）。如:

```
str_trim(c("  李明", " 李明  ", "  李明  ", " 李  明"))
```

```
## [1] "李明"   "李明"   "李明"   "李  明"
```

```
str_trim(c("  李明", " 李明  ", "  李明  ", " 李  明"), side="left")
```

```
## [1] "李明"   "李明  " "李明  " "李  明"
```

```
str_trim(c("  李明", " 李明  ", "  李明  ", " 李  明"), side="right")
```

```
## [1] "  李明" "李明"   "  李明" "李  明"
```

stringr::str_squish(string) 对字符型向量 string 每个元素，删去首尾空格，将重复空格变成单个，返回变换后的结果。如：

```
str_squish(c("  李明", " 李明  ", "  李明  ", " 李  明"))
```

```
## [1] "李明" "李明" "李明" "李 明"
```

基本 R 函数 trimws(x, which) 与 str_trim() 作用类似，选项 which="left" 可以仅删去开头的空格，选项 which="right" 可以仅删去结尾的空格。

```
trimws(c("  李明", " 李明  ", "  李明  ", " 李  明"))
```

```
## [1] "李明"   "李明"   "李明"   "李  明"
```

```
trimws(c("  李明", " 李明  ", "  李明  ", " 李  明"), which="left")
```

```
## [1] "李明"   "李明  " "李明  " "李  明"
```

```
trimws(c("  李明", " 李明  ", "  李明  ", " 李  明"), which="right")
```

```
## [1] "  李明" "李明"   "  李明" "李  明"
```

为了去掉输入字符串中所有空格，可以用 gsub() 替换功能，如：

```
gsub(" ", "", c("  李明", " 李明  ", "  李明  ", " 李  明"), fixed=TRUE)
```

```
## [1] "李明" "李明" "李明" "李明"
```

stringr::str_pad(string, width) 可以将字符型向量 string 的每个元素加长到 width 个字符，不足时左补空格，已经达到或超过 width 的则不变，如：

```
str_pad(c("12", "1234"), 3)
```

```
## [1] " 12"  "1234"
```

可以用选项 `side` 选择在哪里填补空格，默认为`"left"`，还可选`"right"`，`"both"`。

41.7.4 截短或分行

`stringr::str_trunc(x, width)` 可以截短字符串。

`stringr::str_wrap(x, width)` 可以将作为字符型向量的长字符串拆分成近似等长的行，行之间用换行符分隔。

41.7.5 排序

基本 R 函数 `sort()` 可以用来对字符型向量的各个元素按照字典序排序，但是字符的先后顺序是按照操作系统的当前编码值次序，见关于 `locales` 的帮助。

`str_sort(x)` 对字符型向量 `x` 排序。可以用 `locale` 选项指定所依据的 locale，不同的 locale 下次序不同。默认为`"en"` 即英语，中国大陆的 GB 编码 (包括 GBK 和 GB18030) 对应的 locale 是`"zh"`。

`str_order(x)` 返回将 `x` 的各个元素从小到大排序的下标序列。

41.8 简单匹配与查找

41.8.1 开头和结尾匹配

基本 R 的 `startsWith(x, prefix)` 可以判断字符型向量 `x` 的每个元素是否以 `prefix` 开头，结果为一个与 `x` 长度相同的逻辑型向量。如

```
startsWith(c("xyz123", "tu004"), "tu")
```

```
## [1] FALSE  TRUE
```

`endsWith(x, suffix)` 可以判断字符型向量 `x` 的每个元素是否以 `suffix` 结尾，如

```
endsWith(c("xyz123", "tu004"), "123")
```

```
## [1]  TRUE FALSE
```

stringr 包的 str_starts(string, pattern) 判断 string 的每个元素是否以模式 pattern 开头，加选项 negate=TRUE 表示输出反面结果。pattern 是正则表达式，如果需要用非正则表达式，可以用 fixed() 或者 coll() 保护，如：

```
str_starts(c("xyz123", "tu004"), fixed("tu"))
```

```
## [1] FALSE  TRUE
```

```
str_starts(c("xyz123", "tu004"), coll("tu"))
```

```
## [1] FALSE  TRUE
```

stringr 包的 str_ends(string, pattern) 判断是否以给定模式结尾。

41.8.2 中间匹配

函数 grep(), grepl() 等可以用于查找子字符串，位置不限于开头和结尾，详见 §42.1。

在 grepl() 函数中加 fixed=TRUE 选项表示查找一般文本内容（非正则表达式）。比如，查找字符串中是否含有 our:

```
grepl("our", c("flavor", "tournament"), fixed=TRUE)
```

```
## [1] FALSE  TRUE
```

41.9 字符串替换

stringr 包的 str_replace_all(string, fixed(pattern), replacement) 在字符型向量 string 的每个元素中查找子串 pattern，并将所有匹配按照 replacement 进行替换。

```
str_replace_all(c("New theme", "Old times", "In the present theme"),
  fixed("the"), "**")
```

```
## [1] "New **me"           "Old times"          "In ** present **me"
```

用 gsub(pattern, replacement, x, fixed=TRUE) 把字符型向量 x 中每个元素中出现的子串 pattern 都替换为 replacement(注意与 str_replace_all 的自变量次序区别)。如

```
gsub("the", "**",
     c("New theme", "Old times", "In the present theme"),
     fixed=TRUE)
```

```
## [1] "New **me"           "Old times"          "In ** present **me"
```

设有些应用程序的输入要求使用逗号“,”分隔，但是用户可能输入了中文逗号“，”，就可以用 gsub() 来替换：

```
x <- c("15.34,14.11", "13.25，16.92")
x <- gsub("，", ",", x, fixed=TRUE); x
```

```
## [1] "15.34,14.11" "13.25,16.92"
```

例子中 x 的第二个元素中的逗号是中文逗号。

函数 sub() 与 gsub() 类似，但是仅替换第一次出现的 pattern。

41.10 字符串拆分

stringr::str_split(string, pattern) 对字符型向量 string 的每一个元素按分隔符 pattern 进行拆分，每个元素拆分为一个字符型向量，结果是一个列表，列表元素为字符型向量。其中 pattern 是正则表达式，为了按照固定模式拆分，用 fixed() 进行保护。如

```
x <- c("11,12", "21,22,23", "31,32,33,34")
res1 <- str_split(x, fixed(","))
res1
```

```
## [[1]]
## [1] "11" "12"
##
## [[2]]
## [1] "21" "22" "23"
##
## [[3]]
## [1] "31" "32" "33" "34"
```

str_split() 可以用选项 n 指定仅拆分出成几项，最后一项合并不拆分，如：

```
x <- c("11,12", "21,22,23", "31,32,33,34")
res2 <- str_split(x, fixed(","), n=2)
res2
```

```
## [[1]]
## [1] "11" "12"
##
## [[2]]
## [1] "21"    "22,23"
##
## [[3]]
## [1] "31"       "32,33,34"
```

拆分的结果可以用 lapply(), sapply()，vapply() 等函数处理。例如，将每个元素的拆分结果转换成数值型：

```
lapply(res1, as.numeric)
```

```
## [[1]]
## [1] 11 12
##
## [2]]
## [1] 21 22 23
##
## [[3]]
## [1] 31 32 33 34
```

可以用 unlist() 函数将列表中的各个向量连接成一个长向量，如：

```
unlist(res1)
```

```
## [1] "11" "12" "21" "22" "23" "31" "32" "33" "34"
```

注意，即使输入只有一个字符串，str_split() 的结果也是列表，所以输入只有一个字符串时我们应该取出结果列表的第一个元素，如

```
strsplit("31,32,33,34", split=",", fixed=TRUE)[[1]]
```

```
## [1] "31" "32" "33" "34"
```

如果确知每个字符串拆分出来的字符串个数都相同，可以用 stringr::str_split_fixed()，用参数 n 指定拆出来的项数，这时结果为一个字符型矩阵，原来的每个元素变成结果中的一行：

```
x <- c("11,12", "21,22", "31,32")
res3 <- str_split_fixed(x, fixed(","), n=2)
res3
```

```
##      [,1] [,2]
## [1,] "11" "12"
## [2,] "21" "22"
## [3,] "31" "32"
```

基本 R 的 `strsplit(x,split,fixed=TRUE)` 可以把字符型向量 `x` 的每一个元素按分隔符 `split` 拆分为一个字符型向量，`strsplit` 的结果为一个列表，每个列表元素对应于 `x` 的每个元素。

如

```
x <- c("11,12", "21,22,23", "31,32,33,34")
res4 <- strsplit(x, split=",", fixed=TRUE)
res4
```

```
## [[1]]
## [1] "11" "12"
##
## [[2]]
## [1] "21" "22" "23"
##
## [[3]]
## [1] "31" "32" "33" "34"
```

41.11 文本文件读写

文本文件是内容为普通文字、用换行分隔成多行的文件，与二进制文件有区别，二进制文件中换行符没有特殊含义，而且二进制文件的内容往往也不是文字内容。二进制文件的代表有图片、声音，以及各种专用软件的的私有格式文件，如 Word 文件、Excel 文件。

对于文本文件，可以用 `readLines()` 函数将其各行的内容读入为一个字符型数组，字符型数组的每一个元素对应于文件中的一行，读入的字符型数组元素不包含分隔行用的换行符。

最简单的用法是读入一个本地的文本文件，一次性读入所有内容，用如

```
lines <- readLines("filename.ext")
```

其中 `filename.ext` 是文件名，也可以用全路径名或相对路径名。

当文本文件很大的时候，整体读入有时存不下，即使能存下处理速度也很慢，可以一次读入部分行，逐批读入并且逐批处理，这样程序效率更高。这样的程序要复杂一些，例如

```
infcon <- file("filename.ext", open="rt")
batch <- 1000
repeat{
  lines <- readLines(infcon, n=batch)
  if(length(lines)==0) break
  ## 处理读入的这些行
}
close(infcon)
```

以上程序先打开一个文件，inffcon 是打开的文件的读写入口（称为一个 “连接对象”）。每次读入指定的行并处理读入的行，直到读入了 0 行为止，最后关闭 infcon 连接。

对文本文件的典型处理是读入后作一些修改，另外保存。函数 writeLines(lines, con="outfilename.txt") 可以将字符型向量 lines 的各个元素变成输出文件的各行保存起来，自动添加分隔行的换行符。如果是分批读入分批处理的，则写入也需要分批写入，以上的分批处理程序变成：

```
infcon <- file("filename.ext", open="rt")
outfcon <- file("outfilename.txt", open="wt")
batch <- 1000
while(TRUE){
  lines <- readLines(infcon, n=batch)
  if(length(lines)==0) break
  ## 处理读入的这些行, 变换成 outlines
  writeLines(outlines, con=outfcon)
}
close(outfcon)
close(infcon)
```

readLines() 也可以直接读取网站的网页文件，如

```
lines <- readLines(url("https://www.r-project.org/"))
length(lines)
## [1] 116
head(lines)
## [1] "<!DOCTYPE html>"
## [2] "<html lang=\"en\">"
## [3] "  <head>"
## [4] "    <meta charset=\"utf-8\">"
```

```
## [5] "    <meta http-equiv=\"X-UA-Compatible\" content=\"IE=edge\">"
## [6] "    <meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">"
```

readr 包的 `read_lines()` 和 `write_lines()` 函数起到与基本 R 中 `readLines()` 和 `writeLines()` 类似的作用，`read_file()` 和 `read_file_raw()` 可以将整个文件读入为一个字符串。

关于读写文件时的编码问题，详见15.6。

Chapter 42

正则表达式

42.1 正则表达式详解

在对字符串进行查找或替换时，有时要查找替换的不是固定的子串而是某种模式。比如，要查找或替换连续的三个数字，正文中的电子邮件地址，网址，电话号码，等等。正则表达式 (regular expressions) 用于表示各种复杂模式。基本 R 中的正则表达式规则可以用 POSIX 1003.2 标准或者 Perl 规则。建议使用 perl 语言的正则表达式，在基本 R 的有关函数中规定参数 `perl=TRUE`。

stringr 包提供了更方便的正则表达式功能，其正则表达式规则是 ICU 正则表达式规则，针对 UTF-8 编码的文本数据，基本与 perl 规则兼容。

在正则表达式的模式 (pattern) 中，

```
.*+?{}\[]^$()
```

等字符是特殊字符，有特殊的解释。除了\之外的其它 12 个都称为“**元字符**”（meta characters）。

在 R 语言中使用正则表达式时，需要注意 R 字符型常量中一个\要写成两个。为了避免\的转义，可以使用 R 的原始字符串功能，将模式写成 `r"{...}"` 这样的形式，其中`...` 是实际的正则表达式内容，不需要将\写成两个。

42.1.1 字面匹配与匹配显示

如果模式中不含特殊字符，匹配为原样的子串。也叫做字面 (literal) 匹配。stringr 包提供了定义正则表达式、匹配正则表达式、按正则表达式替换、抽取匹配结果、用富文本显示匹配结果等强大功能，其中 `str_view()` 函数可以在 RStudio 软件中高亮显示匹配部分，并在匹配部分两边用 `<>` 界定。在所有输出中都用 `<>` 界定匹配部分。

下面的程序在字符型向量 `x` 的三个字符串元素中查找子字符串`"the"` 并加亮显示（一般输出中用 `<>` 界定匹配部分）：

```
x <- c("New theme", "Old times", "In the present theme")
str_view(x, "the")
```

```
## [1] | New <the>me
## [3] | In <the> present <the>me
```

因为某些希望原样匹配的字符串中可能有元字符（对正则表达式需要特殊解释的字符），所以字面匹配的模式应该用 fixed() 函数保护。例如，在文件名中找到以.txt 结尾的：

```
x <- c("myprog.Rmd", "data.txt", "source.R")
str_view(x, fixed(".txt"))
```

```
## [2] | data<.txt>
```

42.1.2 不区分大小写匹配和 regex 函数

str_view(string, pattern) 中的 pattern 应该为正则表达式类型,如果输入了字符串,会自动被函数 regex() 转换成正则表达式类型。正则表达式的模式一般是区分大小写的，如：

```
str_view(c("Dr. Wang", "DR. WANG", "dR. W.R."), "Dr")
```

```
## [1] | <Dr>. Wang
```

通过在 regex() 函数中加选项 ignore_case=TRUE 可以进行不区分大小写的匹配：

```
str_view(c("Dr. Wang", "DR. WANG", "dR. W.R."),
             regex("Dr", ignore_case=TRUE))
```

```
## [1] | <Dr>. Wang
## [2] | <DR>. WANG
## [3] | <dR>. W.R.
```

fixed() 函数也允许使用 ignore_case=TRUE，如：

```
str_view(c("Dr. Wang", "DR. WANG", "dR. W.R."),
             fixed("Dr", ignore_case=TRUE))
```

```
## [1] | <Dr>. Wang
## [2] | <DR>. WANG
## [3] | <dR>. W.R.
```

在模式前面附加 `(?i)` 前缀式选项也可以实现不区分大小写匹配：

```
str_view(c("Dr. Wang", "DR. WANG", "dR. W.R."),
  "(?i)Dr")
```

```
## [1] | <Dr>. Wang
## [2] | <DR>. WANG
## [3] | <dR>. W.R.
```

也可以提前将输入的源字符串统一转换为小写，这时模式中也仅使用小写，也可以达到不区分大小写匹配的目的。还可以将模式中大写字母和小写字母用字符类表示（见 §42.1.4）。

42.1.3 用句点匹配单个字符

在模式中用 “.” 匹配任意一个字符（除了换行符`"\n"`，能否匹配此字符与选项有关）。如

```
s <- c("abc", "cabs", "lab")
str_view(s, "ab.")
```

```
## [1] | <abc>
## [2] | c<abs>
```

像句点这样的字符称为**元字符**（meta characters），在正则表达式中有特殊作用。如果需要匹配句点本身，用 “`[.]`” 或者 “`\.`” 表示。比如，要匹配 `a.txt` 这个文件名，如下做法有错误：

```
str_view(c("a.txt", "a0txt"), "a.txt")
```

```
## [1] | <a.txt>
## [2] | <a0txt>
```

结果连 `a0txt` 也匹配了。用 “[.]” 表示句点则将句点不做特殊解释：

```
str_view(c("a.txt", "a0txt"), "a[.]txt")
```

```
## [1] | <a.txt>
```

```
str_view(c("a.txt", "a0txt"), "a\\.txt")
```

```
## [1] | <a.txt>
```

注意在 R 语言字符型常量中一个\需要写成两个。

如果仅需按照原样进行查找，也可以将 pattern 的字符串用 fixed() 函数保护，如：如

```
str_view(c("a.txt", "a0txt"), fixed("a.txt"))
```

```
## [1] | <a.txt>
```

42.1.4 匹配一组字符中的某一个

模式中使用方括号给定一个字符类，单个字符与字符类中任何一个字符相同都算是匹配成功。比如，模式“[ns]a.[.]xls”表示匹配的第一个字符是 n 或 s，第二个字符是 a，第三个字符任意，第四个字符是句点，然后是 xls。测试：

```
str_view(c("sa1.xls", "dna2.xlss", "nat.xls"), "[ns]a.[.]xls")
```

```
## [1] | <sa1.xls>
## [2] | d<na2.xls>s
## [3] | <nat.xls>
```

注意匹配并不需要从开头匹配到结尾，中间匹配是允许的，类似于搜索符合某种规律的子串。在上例中第二个元素是从第二个字符开始匹配的，也没有匹配到末尾。

例：模式 [Rr]eg[Ee]x 可以匹配 RegEx 或 Regex 或 regex 或 regEx。

在“[]”中有一些特殊字符：

- 允许用-表示一个范围，如 [a-z] 匹配小写英文字母, [A-Z] 匹配大写英文字母, [a-zA-Z] 匹配大小写的英文字母, [a-zA-Z0-9] 匹配大小写的英文字母和数字。为了匹配一个 16 进制数字，可以用 [0-9A-Fa-f]。
- 在开头用“^”表示余集，比如"[^0-9]" 表示匹配非数字。
- 如果要在 [] 包含原样的 ^ 或-，可以用前导转义符\。

例：下面的模式“[ns]a[0-9][.]xls”要求匹配的第三个字符为数字。

```
str_view(c("sa1.xls", "dna2.xlss", "nat.xls"),
  "[ns]a[0-9][.]xls")
```

```
## [1] | <sa1.xls>
## [2] | d<na2.xls>s
```

例：模式 [ns]a[^0-9][.]xls 要求匹配的第三个字符不能为数字:

```
str_view(c("sa1.xls", "dna2.xlss", "nat.xls"),
  "[ns]a[^0-9][.]xls")
```

```
## [3] | <nat.xls>
```

要求第三个字符为 ^ 或-的例子：

```
str_view(c("5 + 3", "5 - 3", "5 ^ 3"),
  r"{5[ ][\^\-][ ]3}")
```

```
## [2] | <5 - 3>
## [3] | <5 ^ 3>
```

42.1.5 原样匹配元字符

元字符 (meta characters) 是在正则表达式中有特殊含义的字符。比如句点可以匹配任意一个字符, 左方括号代表字符集合的开始。所以元字符不能直接匹配自身，将元字符写在方括号内可以原样匹配这些字符，比如可以用 “[.]” 匹配一个句点，用 [$] 匹配 “$” 符号，用 [|] 匹配 “|” 符号，等等。。

还可以用前导转义字符 “\” 来表示某个元字符需要原样匹配，比如，为匹配左方括号，在前面加上转义字符\变成\[，但是在 R 字符串中一个\必须用\\表示，所以模式 “\[” 在 R 中写成字符串常量，必须写成"\\["。其它的元字符如果要原样匹配也可以在前面加上转义字符\，比如匹配\本身可以用\\，但是在 R 字符型常量中需要写成"\\\\"。用 R 的原始字符串格式可以不用将\写成\\，原始字符串是指字符串前面加上 r 字母作为前缀，用原始字符串作为模式字符串时，模式需要用括号包裹起来。

例，匹配 x[5]，因为 [是元字符，需要写成：

```
str_view(c("int x;", "int x[5]"), "int x\\[5\\]")
```

```
## [2] | <int x[5]>
```

可以用原始字符串格式写成：

```
str_view(c("int x;", "int x[5]"), r"{int x\[5\]}")
```

```
## [2] | <int x[5]>
```

当然，因为模式中所有字符都不想作为特殊字符，也可以用 fixed() 说明：

```
str_view(c("int x;", "int x[5]"), fixed("int x[5]"))
```

```
## [2] | <int x[5]>
```

注意其中的方括号不需要用反斜杠保护。

42.1.6 匹配某类字符

42.1.6.1 匹配空白

表示空白的元字符有:

\f 换页符
\n 换行符
\r 回车符
\t 制表符
\v 垂直制表符

不同操作系统的文本文件的行分隔符不同，为了匹配 Windows 格式的文本文件中的空行，用 "\r\n\r\n"；为了匹配 Unix 格式的文本文件中的空行则用 "\r\r"。写成 R 的字符型常量时，这些表示本身也是 R 的相应字符的表示，所以在 R 字符型常量中这些字符不需要用两个\表示一个\。

匹配任意一个空白字符用 "\s"，这等价于 "[\f\n\r\t\v]"，但是其中的\在 R 字符串中要写成\\，所以\s 写成\\s。大写的 "\S" 则匹配任意一个非空白的字符。

42.1.6.2 匹配数字

用\d 匹配一个数字，相当于 [0-9]。用\D 匹配一个非数字。如

```
str_view(c("n1.xls", "na.xls"), r"{n\d[.]xls}")
```

```
## [1] | <n1.xls>
```

42.1.6.3 匹配字母、数字、下划线

匹配字母、数字、下划线字符用\w（小写），等价于 [a-zA-Z0-9_]。\W（大写）匹配这些字符以外的字符。如

```
str_view(c("file-s1.xls", "s#.xls"),
  r"{s\w[.]}")
```

```
## [1] | file-<s1.>xls
```

可以看出，模式匹配了 s1. 而没有匹配 s#.。

42.1.6.4 十六进制和八进制数

在模式中可以用十六进制数和八进制数表示特殊的字符。十六进制数用`\X` 引入，比如`\X0A` 对应`\n` 字符。八进制数用`\0` 引入，比如`\011` 表示`\t` 字符。

例如，`str_view("abc\nefg\n", r"{\x0A}")` 可以匹配两个换行符。

42.1.6.5 POSIX 字符类

`\d`, `\w` 这样的字符类不方便用在方括号中组成字符集合，而且也不容易记忆和认读，按照 R 的双反斜杠规则写在字符串中就更难认读。

在模式中方括号内可以用 `[:alpha:]` 表示任意一个字母。比如，`[[:alpha:]]` 匹配任意一个字母（外层的方括号表示字符集合，内层的方括号是 POSIX 字符类的固有界定符）。

这样的 POSIX 字符类有：

- `[:alpha:]` 表示任意一个字母；
- `[:lower:]` 为小写字母；
- `[:upper:]` 为大写字母；
- `[:digit:]` 为数字；
- `[:xdigit:]` 为十六进制数字。
- `[:alnum:]` 为字母数字 (不包括下划线)；
- `[:blank:]` 为空格或制表符；
- `[:space:]` 为任何一种空白字符，包括空格、制表符、换页符、换行符、回车符；
- `[:print:]` 为可打印字符；
- `[:graph:]` 和 `[:print:]` 一样但不包括空格；
- `[:punct:]` 为 `[:print:]` 中除 `[:alnum:]` 和空白以外的所有字符；

例如：

```
str_view(c("x1", "_x", ".x", ".1"),
  "[[:alpha:]_.][[:alnum:]_.]")
```

```
## [1] | <x1>
## [2] | <_x>
## [3] | <.x>
## [4] | <.1>
```

模式匹配长度为 2 的字符串，第一个字符是字母、下划线或者小数点，第二个字符是字母、数字、下划线或者小数点。这个模式试图匹配由两个字符组成的合法 R 变量名，但是最后一个非变量名`.1` 也被匹配了。解决这样的问题可以采用后面讲到的 `|` 备择模式。

42.1.7 备择模式

如果有两种模式都算正确匹配，则用 | 连接这两个模式表示两者都可以。例如，某个人的名字用 James 和 Jim 都可以，表示为 James|Jim, 如

```
str_view(c("James, Bond", "Jim boy"), "James|Jim")
```

```
## [1] | <James>, Bond
## [2] | <Jim> boy
```

两个字符的合法 R 变量名的匹配：

```
str_view(c("x1", "_x", ".x", ".1"),
  "[[:alpha:]_][[:alnum:]_.]|[.][[:alpha:]_]")
```

```
## [1] | <x1>
## [2] | <_x>
## [3] | <.x>
```

42.1.8 匹配开头和末尾

模式匹配相当于在字符串内部搜索某种模式，如果必须从字符串开头匹配，在模式中取第一个模式规定为 ^ 或\A。如果模式中最后一个字符是 $ 或\Z，则需要匹配到字符串末尾。用\Z 匹配字符串末尾时如果末尾有一个换行符则匹配到换行符之前。

如

```
str_view(c(
  "n1.xls", "na.xls", "cn1.xls", "n1.xlsx"),
  r"{^n\d[.]xls$}")
```

```
## [1] | <n1.xls>
```

只匹配了第一个输入字符串。

用 “\A” 和 “\Z” 的写法：

```
str_view(c(
  "n1.xls", "na.xls", "cn1.xls", "n1.xlsx"),
  r"{\An\\d[.]xls\Z}")
```

有时候源文本的每个字符串保存了一个文本文件内容，各行用\n 分隔，后面将给出匹配每行的行首与行尾的方法。

42.1.9 单词边界

用\b 匹配单词边界，这样可以查找作为单词而不是单词的一部分存在的内容。\B 匹配非单词边界。如

```
str_view(c("a cat meaos", "the category"),
  r"{\bcat\b}")
```

```
## [1] | a <cat> meaos
```

又如，找到单独的函数 sum 而不是子串 sum：

```
x <- c("summary(x)", "summarize(df)",
  "rowsum(x)", "sum(x)")
str_view(x, r"{\bsum\b}")
```

```
## [4] | <sum>(x)
```

42.1.10 重复匹配

42.1.10.1 加号重复匹配

模式中在一个字符或字符集合后加后缀 + 表示一个或多个前一字符。比如

```
str_view(c("sa1", "dsa123"), "sa[[:digit:]]+")
```

```
## [1] | <sa1>
## [2] | d<sa123>
```

例如，匹配电子邮件地址：

```
str_view("abc123@efg.com",
             "^[[:alnum:]_]+@[[:alnum:]_]+[.][[:alnum:]_]+$")
```

```
## [1] | <abc123@efg.com>
```

匹配的电子邮件地址在 @ 前面可以使用任意多个字母、数字、下划线，在 @ 后面由小数点分成两段，每段可以使用任意多个字母、数字、下划线。这里用了 ^ 和 $ 表示全字符串匹配。

42.1.10.2 星号和问号重复匹配

在一个字符或字符集合后加后缀 `*` 表示零个或多个前一字符，后缀`?` 表示零个或一个前一字符。

比如，`^https?://[[:alnum:]./]+$` 可以匹配 http 或 https 开始的网址。如

```
str_view(c("http://www.163.net", "https://123.456."),
             "^https?://[[:alnum:]_./]+$")
```

```
## [1] | <http://www.163.net>
## [2] | <https://123.456.>
```

(注意第二个字符串不是合法网址但是按这个正则表达式也能匹配)

`x[[:digit:]]*` 能匹配 “x”, “x1”, “x123” 这样的变量名，如：

```
str_view(c("x", "x1", "x123"), "x[[:digit:]]*")
```

```
## [1] | <x>
## [2] | <x1>
## [3] | <x123>
```

```
str_view(c("x", "x1", "x123"), "x\\d*")
```

```
## [1] | <x>
## [2] | <x1>
## [3] | <x123>
```

从上面的例子可以看出，在指定某一字符类（或集合）重复时，不需要严格重复前一字符，而是只要保持同一类即可。

42.1.10.3 计数重复

问号可以表示零个或一个，而加号、星号重复不能控制重复次数。在后缀大括号中写一个整数表示精确的重复次数。如

```
str_view(c("1", "12", "123", "1234"),
  "[[:digit:]]{3}")
```

```
## [3] | <123>
## [4] | <123>4
```

模式匹配的是三位的数字。因为没有要求从开头一直匹配到末尾，所以三位以上数字也能匹配其中开始的三位。

可以在后缀大括号中指定重复的最小和最大次数，中间用逗号分隔。最小重复数允许指定为 0。重复数的逗号后面空置表示重复数没有上限。例如，后缀 `{3,}` 表示前一模式必须至少重复 3 次。

例：月日年的日期格式可以用

```
[[:digit:]]{1,2}[-/][[:digit:]]{1,2}[-/][[:digit:]]{2,4}
```

来匹配。

比较长的正则表达式会很难认读，我们可以用 `paste0()` 函数将其拆分为几个部分，并对每一部分添加适当的注释。如 (注意这个模式还会匹配非日期)

```
pat <- paste0(
  "[[:digit:]]{1,2}[-/]", # month
  "[[:digit:]]{1,2}[-/]", # day
  "[[:digit:]]{2,4}")     # year
str_view(c("2/4/1998", "13/15/198"), pat)
```

```
## [1] | <2/4/1998>
## [2] | <13/15/198>
```

也可以用 `regex()` 函数中加选项 `comment=TRUE` 加注释，这时一个正则表达式可以写成多行，在行尾添加注释，所有原样空格必须转义或写在字符集中，如：

```
pat <- regex(
  r"{[[:digit:]]{1,2}[-/]   # month
  [[:digit:]]{1,2}[-/]      # day
  [[:digit:]]{2,4}}",       # year
  comment = TRUE)
str_view(c("2/4/1998", "13/15/198"), pat)
```

```
## [1] | <2/4/1998>
## [2] | <13/15/198>
```

42.1.11 贪婪匹配和懒惰匹配

无上限的重复匹配如 `*`, `+`, `{3,}` 等缺省是贪婪型的，重复直到文本中能匹配的最长范围。比如我们希望找出圆括号这样的结构，很容易想到用`\(.+\)` 这样的模式（注意圆括号是元字符，需要用反斜杠保护），但是这不会恰好匹配一次，模式会一直搜索到最后一个）为止。

例如：

```
str_view("(1st) other (2nd)", r"{\(.+\)}")
```

```
## [1] | <(1st) other (2nd)>
```

我们本来期望的是提取两个 “(1st)” 和 “(2nd)” 组合，不料整个地提取了 “`(1st) other (2nd)`”。这就是因为`.+`的贪婪匹配。

如果要求尽可能短的匹配，使用 `*?`, `+?`, `{3,}?` 等 “懒惰型” 重复模式。在无上限重复标志后面加问号表示懒惰性重复。

比如，上例中模式修改后得到了期望的结果：

```
str_view("(1st) other (2nd)", r"{\(.+?\)}")
```

```
## [1] | <(1st)> other <(2nd)>
```

懒惰匹配会造成搜索效率降低，应仅在需要的时候使用。

42.1.12 允许句点匹配换行

句点通配符一般不能匹配换行，如

```
str_view("(1,\n2)", r"{\(.+?\)}")
```

跨行匹配失败。

一种办法是预先用 `str_replace_all()` 或 `gsub()` 把所有换行符替换为空格。但是这只能解决部分问题。

解决方法是在将模式用 `regex()` 保护并加选项 `dotall=TRUE`，使得句点通配符可以匹配换行符，称为句点全匹配模式。如

```
str_view("(1,\n2)",
  regex(r"{\(.+?\)}", dotall=TRUE))
```

```
## [1] | <(1,
##     | 2)>
```

也可以在在 Perl 格式的正则表达式开头添加 `(?s)` 选项，如：

```
str_view("(1,\n2)", r"{(?s)\(.+?\)}")
```

```
## [1] | <(1,
##     | 2)>
```

42.1.13 多行模式

在 regex() 函数中加选项 multiline=TRUE，或者在正则表达式开头用 (?m) 表示把整个输入字符串看成用换行符分开的多行。这时 ^ 和 $ 匹配每行的开头和结尾，“每行”是指字符串中用换行符分开的各个字符子串。(?s) 与 (?m) 可以同时使用，写成 (?sm)。

例：

```
str_view("(1,2)\n(3,4)\n", r"{^\(.+?\)$}")
```

元数据中包含两行内容，结果没有能够匹配，这是因为模式要求从整个字符串开头一直匹配到末尾。

增加 multiline=TRUE 则可以对每行分别匹配，结果找到两处匹配：

```
str_view("(1,2)\n(3,4)\n",
  regex(r"{^\(.+?\)$}", multiline=TRUE))
```

```
## [1] | <(1,2)>
##     | <(3,4)>
##     |
```

使用 (?m) 选项：

```
str_view("(1,2)\n(3,4)\n",
  r"{(?m)^\(.+?\)$}")
```

```
## [1] | <(1,2)>
##     | <(3,4)>
##     |
```

42.1.14 逐行处理

虽然正则表达式有多行和跨行选项，但是当源数据很长时，匹配效率会很低。

R 的 readLines() 函数可以把一整个文本文件读成一个字符型向量，每个元素为一行，元素中不包含换行符。R 的字符型函数可以对这样的字符型向量每个元素同时处理，也就实现了逐行处理。

如果字符串 x 中包含了一整个文本文件内容，其中以\n 分隔各行，为了实现逐行处理，可以先用 str_split() 函数拆分成不同行：

```
cl <- strs_plit(x, "\r?\n")[[1]]
```

结果将是一个字符型向量，每个元素是原来的一行，最后一个元素是空字符串。如

```
x <- c("This is first line.\nThis is second line.\n")
cl <- str_split(x, "\r?\n")[[1]]
cl
```

```
## [1] "This is first line."  "This is second line." ""
```

42.1.15 替换

stringr 包的 str_replace_all(string, pattern, replacement) 在字符型向量 string 的每个元素中查找模式 pattern，并将所有匹配按照 replacement 进行替换。在 replacement 可以用\1, \2 中表示模式中的捕获，除此之外元字符没有特殊作用。

基本 R 中 gsub() 有类似功能。

如：

```
str_replace_all(c("123,456", "011"), ",", "")
```

```
## [1] "123456" "011"
```

又如：

```
str_replace_all(c("123,456", "011"),
                "([[:digit:]]+),([[:digit:]]+)", "\\2,\\1")
```

```
## [1] "456,123" "011"
```

注意源数据中第二个元素因为不能匹配所以就原样返回了，没有进行替换。

str_replace() 则仅对输入字符型向量的每个元素中模式的第一次出现进行替换，不如 str_replace_all() 常用。

函数 str_remove_all() 相当于在 str_replace_all() 中指定替换内容为空字符串。如：

```
str_remove_all(c("123,456", "011"), ",")
```

```
## [1] "123456" "011"
```

42.1.16 分组与捕获

在正则表达式中用圆括号来分出组，作用是

- 确定优先规则;
- 组成一个整体;
- 拆分出模式中的部分内容（称为捕获）;
- 定义一段供后续引用或者替换。

圆括号中的模式称为**子模式**，或者**捕获**。

42.1.16.1 确定优先级

在使用备择模式时，`James|Jim` 是在单词 James 和 Jim 之间选择。如果希望选择的是中间的 `mes` 和 `Ji` 怎么办？可以将备择模式保护起来，如 `Ja(mes|Ji)m`，就可以确定备择模式的作用范围。

正则表达式有比较复杂的优先级规则，所以在不确定那些模式组合优先采纳时，应该使用圆括号分组，确定运算优先级。

42.1.16.2 组成整体

元字符问号、加号、星号、大括号等表示重复，前面的例子中都是重复一个字符或者字符类。如果需要重复由多个字符组成的模式，如 `x[[:digit:]]{2}` 怎么办？只要将该模式写在括号中，如：

```
str_view(c("x01x02", "_x11x9"),
  "(x[[:digit:]]{2})+")
```

```
## [1] | <x01x02>
## [2] | _<x11>x9
```

上例的元数据中，第一个元素重复了两次括号中的模式，第二个元素仅有一次括号中的模式。

注意：用表示重复的元字符 (`+`, `*`) 重复某个模式时，从第二次开始，并不是要重复前面的字符或者子串，而是重复前面的模式。比如上例中 `x01x02`。

42.1.16.3 捕获与向后引用

有时一个模式中部分内容仅用于定位，而实际有用的内容是其中的一部分，就可以将这部分有用的内容包在圆括号中作为一个捕获。

分组是自动编号的，以左开括号的序号为准（除了作为选项、有名捕获等开扩号以外）。在替换或者向后引用时，可以用`\1`，`\2` 等表示匹配中第一个左括号对应的分组，第二个左扩号对应的分组，……。

比如，如果想严格重复前面的某个子字符串怎么办？在模式中可以用`\1`, `\2` 等表示严格重复前面捕获的子字符串，这称为 “向后引用”(back reference)。

例如，`([a-z]{3})\1` 这样的模式可以匹配如 `abcabc`, `uxzuxz` 这样的三字母重复：

```
str_view(c("abcabc", "aabbcc"),
  r"{([a-z]{3})\1}")
```

```
## [1] | <abcabc>
```

又例如，下面的程序找出了年（后两位）、月、日数字相同的日期：

```
str_view(c("2008-08-08", "2017-01-18"),
  r"{\d{2}(\d{2})-\1-\1}")
```

```
## [1] | <2008-08-08>
```

42.1.16.4 捕获与替换

利用分组捕获可以对字符串进行修改。

例：希望把带有前导零的数字的前导零删除，可以用如

```
x <- c("1204", "01204", "001204B")
pat <- r"{\b0+([1-9][0-9]*)\b}"
repl <- r"{\1}"
str_view(x, pat)
```

```
## [2] | <01204>
```

```
str_replace_all(x, pat, repl)
```

```
## [1] "1204"    "1204"    "001204B"
```

上例的模式中的\b 表示单词边界，所以中间的 0 不会被当作前导零，不是整个数字的也不会被修改。

上例中的 `str_view()` 仅用于调试目的，在进行替换时不是必要步骤。

例：为了交换横纵坐标，可以用如下替换

```
x <- "1st: (5,3.6), 2nd: (2.5, 1.1)"
pat <- paste0(
  "[(]([[:digit:].]+),",
  "[[:space:]]*([[:digit:].]+)[)]", sep="")
repl <- "(\\2, \\1)"
str_view(x, pat)
```

```
## [1] | 1st: <(5,3.6)>, 2nd: <(2.5, 1.1)>
```

```
str_replace_all(x, pat, repl)
```

```
## [1] "1st: (3.6, 5), 2nd: (1.1, 2.5)"
```

例: 要匹配 yyyy-mm-dd 这样的日期，并将其改写为 mm/dd/yyyy，就可以用这样的替换模式：

```
x <- c("1998-05-31", "2017-01-14")
pat <- "([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})"
repl <- r"{\2/\3/\1}"
str_view(x, pat)
```

```
## [1] | <1998-05-31>
## [2] | <2017-01-14>
```

```
str_replace_all(x, pat, repl)
```

```
## [1] "05/31/1998" "01/14/2017"
```

42.1.16.5 命名捕获

可以用 (?<name>...) 这样的格式对捕获命名，而不是仅根据开括号的序号区分。例如：

```
x <- c("1998-05-31", "2017-01-14")
pat <- r"{(?<year>[0-9]{4})-(?<mon>[0-9]{1,2})-(?<day>[0-9]{1,2})}"
str_view(x, pat)
```

```
## [1] | <1998-05-31>
## [2] | <2017-01-14>
```

但 str_replace_all() 还不支持使用命名捕获作为替换。

42.1.16.6 非捕获分组

如果某个分组仅想起到分组作用但是不会提取具体的匹配内容也不会用该组内容做替换，可以将该组变成 “非捕获分组”，办法是把表示分组开始左圆括号变成 (?: 三个字符。这在用分组表示优先级时比较有用，如"Jam(es|Ji)m" 可以写成"Jam(?:es|Ji)m"。非捕获分组在向后引用和替换时不计入\1、\2 这样的排列中。

比如，把 1921-2020 之间的世纪号删去，可以用

```
x <- c("1978", "2017", "2035")
pat <- r"{\A(?:19|20)([0-9]{2})\Z}"
repl <- r"{\1}"
str_view(x, pat)
```

```
## [1] | <1978>
## [2] | <2017>
## [3] | <2035>
```

```
str_replace_all(x, pat, repl)
```

```
## [1] "78" "17" "35"
```

其中用了非捕获分组使得备择模式 19|20 优先匹配。注意模式并没有能保证日期在 1921-2020 之间。更周密的程序可以写成：

```
x <- c("1978", "2017", "2035")
pat1 <- r"{\A19(2[1-9]|[3-9][0-9])\Z}"
pat2 <- r"{\A20([01][0-9]|20)\Z}"
repl <- r"{\1}"
str_view(x, pat1)
```

```
## [1] | <1978>
```

```
str_view(x, pat2)
```

```
## [2] | <2017>
```

```
x |>
  str_replace_all(pat1, repl) |>
  str_replace_all(pat2, repl)
```

```
## [1] "78"   "17"   "2035"
```

42.1.16.7 提取捕获内容

为了找到字符型向量中哪些元素能匹配模式，可以用 `str_subset()`, `str_which()`, `str_detect()`。

为了提取字符型向量中匹配模式的部分子串，可以用 `str_extract()` 和 `str_extract_all()`。

为了提取字符型向量中匹配模式的部分以及捕获的部分，可以用 `str_match()` 和 `str_match_all()`。

参见42.2。

42.1.16.8 非匹配向前向后偷窥

如果要提取的一个模式以某个后缀作为标志，可以将这个后缀作为模式的一部分，然后仅捕获该后缀之前的部分。比如，对 http://baidu.com, https://baidu.com, ftp://baidu.com 等网址，希望提取 http, https, ftp 等协议，可以匹配到冒号为止：

```
addrs <- c(
  "http://baidu.com",
  "https://baidu.com",
  "ftp://baidu.com")
pat1 <- r"{^(.+):}"
str_view(addrs, pat1)
```

```
## [1] | <http:>//baidu.com
## [2] | <https:>//baidu.com
## [3] | <ftp:>//baidu.com
```

这样的模式多匹配了一个“:”号。可以用 str_match() 函数提取其中的捕获：

```
str_match(addrs, pat1)
```

```
##      [,1]     [,2]
## [1,] "http:"  "http"
## [2,] "https:" "https"
## [3,] "ftp:"   "ftp"
```

结果是字符型矩阵，第二列是所需的内容。但这样还是在整个匹配中包含了不需要的内容。

可以用 (?=) 包含要提取的模式的标志性后缀，但匹配结果不包含这个后缀，如：

```
pat2 <- r"{^(.+)(?=:)}"
str_view(addrs, pat2)
```

```
## [1] | <http>://baidu.com
## [2] | <https>://baidu.com
## [3] | <ftp>://baidu.com
```

可见这次没有匹配“:”部分。提取只要用 str_extract()：

```
str_extract(addrs, pat2)
```

```
## [1] "http"  "https" "ftp"
```

类似地，如果需要提取的部分以某个前缀为标志，可以用 `(?<=)` 包含这个前缀，最后匹配的模式不不含这个前缀。例如，设输入的字符串向量 `addrs` 中都是一些网址，希望仅匹配//后面的主要部分，可以用：

```
pat3 <- r"{(?<=//).+$}"
str_view(addrs, pat3)
```

```
## [1] | http://<baidu.com>
## [2] | https://<baidu.com>
## [3] | ftp://<baidu.com>
```

可见仅匹配了//后面的内容。提取：

```
str_extract(addrs, pat3)
```

```
## [1] "baidu.com" "baidu.com" "baidu.com"
```

还可以用 `(?!)` 包含不允许的后缀，用 `(?<!)` 包含不允许的前缀。

42.1.17 在数据框中拆分列

`tidyr::separate_wider_regex()` 可以对一个字符型列用指定的模式拆分出其中的内容。比如，有如下的数据：

smix
Alice-F:13
Duke-M:14
Thomas-M:11

可以在 `tidyr::separate_wider_regex()` 用有名向量形式输入一个分组的模式 `patterns`，将有名部分拆分为新的数据框列，如：

```
d.swd |>
  tidyr::separate_wider_regex(smix,
    patterns = c(
      name = "[[:alpha:]]+", "-",
      sex = "F|M", ":",
      age = "[[:digit:]]+"
    ))
```

```
## # A tibble: 3 x 3
##   name   sex   age
##   <chr>  <chr> <chr>
## 1 Alice  F     13
## 2 Duke   M     14
## 3 Thomas M     11
```

如果字符串中有固定的分隔符，可以用 `tidyr::separate_wider_delim()`；如果字符串有固定的列位置分配，可以用 `tidyr::separate_wider_position()`。

字符串内容中捕获的更一般的提取，可以用 `stringr::str_match_all()`。

42.2 stringr 包的正则表达式函数

前面主要使用了 `str_view()`, `str_replace_all()` 函数。这里对 stringr 包的正则表达式函数进行介绍。

42.2.1 str_view() 函数

`str_view(string, pattern)` 在 RStudio 中加亮显示 `pattern` 给出的正则表达式模式在 `string` 中的所有匹配，并将每个匹配部分用 `<>` 界定。在其它输出中不加亮显示，仅使用 `<>` 界定。`string` 是输入的源字符型向量。

如果要匹配的是固定字符串，写成 `str_view(string, fixed(pattern))`。

如果要匹配的是单词等的边界,模式用 `boundary()` 函数表示,如 `str_view("a brown fox", boundary("word"))` 将匹配首个单词。

42.2.2 regex() 函数

stringr 包用到正则表达式模式的地方，实际上应该写成 `regex(pattern)`，只写模式本身是一种简写。`regex()` 函数可以指定 `ignore_case=TRUE` 要求不区分大小写，指定 `multi_line=TRUE` 使得 `^` 和 `$` 匹配用换行符分开的每行的开头和结尾，`dotall=TRUE` 使得 `.` 能够匹配换行符。`comment=TRUE` 使得模式可以写成多行，行尾的井号后面表示注释，这时空格不再原样匹配，为了匹配空格需要写在方括号内或者用反斜杠开头。

与 `regex()` 类似的表示模式的函数有 `fixed()`，`boundary()`，`coll()`。

42.2.3 构造复杂的模式

可以用 `paste0()` 将多个字符串连接在一起构成一个模式。可以在 `regex()` 中用 `comment = TRUE` 将模式写成多行带有行尾注释的形式。例如，年月日格式的日期：

```
pat_ymd <- regex(r"{
  (19|20)[[:digit:]]{2} # 年份
  ([-/])
  (0?[1-9]|1[012])      # 月
  \2
  (1[0-9]|2[0-9]|3[01]|0?[1-9]) # 日
  }", comments=TRUE)
str_view(c(
  "1978-1-03", "1911-07-31",
  "2023-10-18", "3000-1-1",
  "2023-14-01"), pat_ymd)
```

```
## [1] | <1978-1-03>
## [2] | <1911-07-31>
## [3] | <2023-10-18>
```

可以用 str_flatten(subpats, sep) 将保存在向量 subpats 中的元素用 sep 分隔符连接起来，组成一个长模式。为防止输入的内容有元字符，可以用 str_escape() 进行转义。str_escape() 将输入内容中的正则表达式元字符转义为原样匹配，如：

```
str_escape("x[1] = myfile.txt")
```

```
## [1] "x\\[1\\] = myfile\\.txt"
```

例：要匹配保存在字符型向量中的任何一个颜色名，将这些颜色名用 | 连接成一个长的正则表达式后匹配。

```
subpats <- c("red", "blue", "green")
x <- c("A red fox", "The white house", "Green sleeves")
pat <- regex(str_flatten(str_escape(subpats), "|"),
  ignore_case=TRUE)
str_view(x, pat)
```

```
## [1] | A <red> fox
## [3] | <Green> sleeves
```

42.2.4 检查哪些元素能够匹配

str_detect(string, pattern) 返回字符型向量 string 的每个元素是否匹配 pattern 中的模式的逻辑型结果。与基本 R 的 grepl() 作用类似。如

```
x <- c("New theme", "Old times", "In the present theme")
str_view(x, "the")
```

```
## [1] | New <the>me
## [3] | In <the> present <the>me
```

```
str_detect(x, "the")
```

```
## [1]  TRUE FALSE  TRUE
```

上例中的 `str_view()` 仅用作调试目的。

当要查找的内容是 tibble 的一列时，用 `filter()` 与 `str_detct()` 配合，可以进行行子集选择。比如，在数据框的人名中查找中间有空格的名字：

```
tibble(name=c(" 马思聪", " 李  明")) |>
  filter(str_detect(name,
    "[[:alpha:]]+[[:space:]]+[[:alpha:]]+"))
```

```
## # A tibble: 1 x 1
##   name
##   <chr>
## 1 李  明
```

可以在 `dplyr::summarize()` 中用 `sum(str_detect(x, pat))` 计算符合模式的观测个数，用 `mean` 计算比例。

因为 `str_detect()` 返回与输入源字符串向量等长的逻辑型向量，所以特别适用于用逻辑运算来满足复杂的条件。例如，在姓名中找到同时有 `a` 和 `y` 的：

```
x <- c("Alice", "Becka", "Gail",
  "Karen", "Kathy", "Mary", "Sandy")
x[str_detect(x, fixed("a")) &
    str_detect(x, fixed("y"))]
```

```
## [1] "Kathy" "Mary"  "Sandy"
```

有 `a` 或有 `y` 的：

```
x <- c("Alice", "Becka", "Gail",
  "Karen", "Kathy", "Mary", "Sandy")
x[str_detect(x, fixed("a")) |
    str_detect(x, fixed("y"))]
```

```
## [1] "Becka" "Gail"  "Karen" "Kathy" "Mary"  "Sandy"
```

用这种方法可以将复杂的匹配条件拆分为多个条件，并用与、或、非组合起来。

42.2.5 计数模式匹配次数

str_count() 则返回模式在每个元素中匹配的次数。如

```
str_count(c("123,456", "011", "I,II,III"), "[[:digit:]]")
```

```
## [1] 6 3 0
```

正则表达式在匹配时，是否允许同一源字符串中的两次匹配有重叠部分？这是不允许的。例如：

```
str_view("1212121", "121")
```

```
## [1] | <121>2<121>
```

```
str_count("1212121", "121")
```

```
## [1] 2
```

如果允许重叠，就可以有 3 次 121 模式。

42.2.6 返回匹配的元素或元素序号

str_subset(string, pattern) 返回字符型向量中能匹配 pattern 的那些元素组成的子集，与基本 R 函数 grep(pattern, string, value=TRUE) 效果相同。注意，返回的是整个元素而不是匹配的子串。

比如，查找人名中间有空格的：

```
str_view(c("[马思聪]", "[李  明]"),
             "[[:alpha:]]+[[:space:]]+[[:alpha:]]+")
```

```
## [2] | [<李  明>]
```

```
str_subset(c("[马思聪]", "[李  明]"),
           "[[:alpha:]]+[[:space:]]+[[:alpha:]]+")
```

```
## [1] "[李  明]"
```

注意上例中仅返回了有匹配的元素，而且是匹配元素的整个字符串而不是匹配的部分。

str_which(string, pattern) 返回字符型向量 string 的元素当中能匹配 pattern 中的模式的元素序号。与基本 R 的 grep() 作用类似。如

```
x <- c("New theme", "Old times", "In the present theme")
str_which(x, "the")
```

```
## [1] 1 3
```

42.2.7 提取匹配内容

str_subset() 返回的是有匹配的源字符串，而不是匹配的部分子字符串。用 str_extract(string, pattern) 从源字符串中取出**首次**匹配的子串，没有匹配则不返回结果。结果是一个与 str_subset() 长度相同的字符串向量，仅匹配元素的首个匹配子串被返回。因为输出长度不一定等于输入长度，所以这个函数应该慎用。

如

```
str_view(c("A falling ball", "case"), "all")
```

```
## [1] | A f<all>ing b<all>
```

```
str_extract("A falling ball", "all")
```

```
## [1] "all"
```

注意第二个输入字符串不匹配所有返回结果中不包含该元素的对应输出。这会造成结果的对应困难，所以应仅在确保所有输入都能匹配时才使用 str_extract() 这个函数，一般应使用 str_extract_all()。

str_extract_all(string, pattern) 取出所有匹配子串，结果是一个列表，长度与输入的字符型向量 string 的元素个数相等，列表的每个元素对应于 string 的每个元素。结果列表的每个元素是一个字符型数组，存放所有匹配的子字符串，没有匹配的输入字符串对应的输出元素是空向量（长度为 0 的向量）。如：

```
x <- c("A falling ball", "Phone call.", "case")
str_view(x, "all")
```

```
## [1] | A f<all>ing b<all>
## [2] | Phone c<all>.
```

```
str_extract_all(x, "all")
```

```
## [[1]]
## [1] "all" "all"
##
## [[2]]
## [1] "all"
##
## [[3]]
## character(0)
```

str_extract_all() 可以加选项 simplyfy=TRUE，使得返回结果变成一个字符型矩阵，每行是原来一个元素中取出的各个子串，列数等于最大匹配次数，没有那么多匹配次数的填以空字符串。如果正常匹配结果不会出现空字符就可以用这种方法简化结果的保存和访问。如

```
str_extract_all(x, "all", simplify=TRUE)
```

```
##      [,1]  [,2]
## [1,] "all" "all"
## [2,] "all" ""
## [3,] ""    ""
```

42.2.8 提取分组捕获内容

str_subset() 提取的是能匹配模式的元素子集,而不是匹配的模式或者捕获;str_extract() 和 str_extract_all() 提取的是每个元素的首次或者所有匹配的子字符串，而不是其中的捕获。

str_match(string, pattern) 提取每个元素的首次匹配内容以及其中各个捕获分组内容，结果是一个矩阵，行数等于输入的字符型向量 string 的元素个数，矩阵每行对应于向量 string 中的一个元素，每行第一个元素是匹配内容，其它元素是各个捕获，没有则为字符型缺失值（不是空字符串）。

比如，希望匹配中间有空格的人名并捕获空格前后部分：

```
str_match(c(" 马思聪", " 李  明"),
  r"{([[:alpha:]]+)[[:space:]]+([[:alpha:]]+)}")
```

```
##      [,1]     [,2] [,3]
## [1,] NA       NA   NA
## [2,] "李  明" "李" "明"
```

上例中源数据第一个元素没有匹配，所以结果都是缺失值 NA，第二个元素的结果在第二行，首先是整个匹配的子字符串，然后是捕获的两个部分。

`stringr::str_match_all(string, pattern)` 匹配每个字符串中所有出现位置，结果是一个列表，每个列表元素对应于输入的字符型向量 `string` 的每个元素，结果中每个列表元素是一个字符型矩阵，用来保存所有各个匹配以及匹配中的捕获，每行是一个匹配的结果，首先是匹配结果，其次是各个捕获。结果列表中每个作为列表元素的矩阵大小不一定相同。当某个元素完全没有匹配时，结果列表中对应元素是行数为 0 的矩阵。

比如，模式为 19xx 或者 20xx 的年份，并将其分为前两位和后两位：

```
x <- c("1978-2000", "2011-2020-2099", "2100-2199")
pat <- r"{\b(19|20)([0-9]{2})\b}"
str_view(x, pat)
```

```
## [1] | <1978>-<2000>
## [2] | <2011>-<2020>-<2099>
```

```
mlist <- str_match_all(x,  pat); mlist
```

```
## [[1]]
##      [,1]   [,2] [,3]
## [1,] "1978" "19" "78"
## [2,] "2000" "20" "00"
##
## [[2]]
##      [,1]   [,2] [,3]
## [1,] "2011" "20" "11"
## [2,] "2020" "20" "20"
## [3,] "2099" "20" "99"
##
## [[3]]
##      [,1] [,2] [,3]
```

下面的程序合并上面提取的年份后两位为一个字符型向量：

```
library(purrr)
mlist |>
  keep(\(ele) nrow(ele) > 0) |>
  map(\(ele) ele[,3]) |>
  reduce(c)
```

```
## [1] "78" "00" "11" "20" "99"
```

用基本 R 功能：

```
ml <- Filter(function(m) nrow(m)>0, mlist)
ml <- Map(function(m) m[,3], ml)
ml <- Reduce(c, ml); ml
```

```
## [1] "78" "00" "11" "20" "99"
```

42.2.9 定位匹配位置

str_locate(string, pattern) 对输入字符型向量 string 的每个元素返回首次匹配 pattern 的开始和结束位置。输出结果是一个两列的矩阵，每行对应于输入的一个元素，每行的两个元素分别是首次匹配的开始和结束字符序号（按字符计算）。如

```
x <- c("A falling ball", "Phone call.")
str_view(x, "all")
```

```
## [1] | A f<all>ing b<all>
## [2] | Phone c<all>.
```

```
str_locate(x, "all")
```

```
##      start end
## [1,]     4   6
## [2,]     8  10
```

str_locate_all(string, pattern) 则可以返回每个元素中所有匹配的开始和结束位置，结果是一个列表，每个列表元素对应于输入字符型向量的每个元素，结果中每个列表元素是一个两列的数值型矩阵，每行为一个匹配的开始和结束字符序号。如

```
x <- c("A falling ball", "Phone call.")
str_view(x, "all")
```

```
## [1] | A f<all>ing b<all>
## [2] | Phone c<all>.
```

```
str_locate_all(x, "all")
```

```
## [[1]]
##      start end
## [1,]     4   6
```

```
## [2,]    12  14
##
## [[2]]
##      start end
## [1,]     8  10
```

注意如果需要取出匹配的元素可以用 `str_subset()`,要取出匹配的子串可以用 `str_extract()` 和 `str_extract_all()`，取出匹配的子串以及分组捕获可以用 `str_match()` 和 `str_match_all()`。

42.3 附录：利用基本 R 函数进行正则表达式处理

基本 R 函数 `grep`, `sub`, `gsub`, `regexpr`, `gregexpr`, `regexec` 中的 `pattern` 参数可以是正则表达式,这时应设参数 `fixed=FALSE`。`strsplit` 函数中的参数 `split` 也可以是正则表达式。`regmatches` 函数从 `regexpr`, `gregexpr`, `regexec` 的结果中提取匹配的字符串。

以原样匹配为例。

```r
x <- c("New theme", "Old times", "In the present theme")
regexpr("the", x, perl=TRUE)
```

```
## [1]  5 -1  4
## attr(,"match.length")
## [1]  3 -1  3
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

这里使用了 `regexpr` 函数。`regexpr` 函数的一般用法为：

```r
x <- c("New theme", "Old times", "In the present theme")
regexpr(pattern, text, ignore.case = FALSE, perl = FALSE,
        fixed = FALSE, useBytes = FALSE)
```

自变量为：

- pattern 是一个正则表达式，如果用了 `fixed=TRUE` 选项，则当作普通原样文本来匹配；
- text 是源字符串向量，要从其每个元素中查找 pattern 模式出现的位置；
- ignore.case：是否要忽略大小写匹配；
- perl 选择是否采用 perl 格式，如果不把 pattern 当作普通原样文本，应该选 `perl=TRUE`，perl 语言的正则表达式是事实上的标准，所以这样兼容性更好；

- fixed 当 fixed=TRUE 时 pattern 作为普通原样文本解释；
- useBytes 为 TRUE 时逐字节进行匹配，否则逐字符进行匹配。之所以有这样的区别，是因为有些编码中一个字符由多个字节构成，BGK 编码的汉字由两个字节组成，UTF-8 编码的汉字也是由两个字节构成。

regexpr() 函数返回一个整数值的向量，长度与 text 向量长度相同，结果的每个元素是在 text 的对应元素中 pattern 的首次匹配位置；没有匹配时结果元素取-1。结果会有一个 match.length 属性，表示每个匹配的长度，无匹配时取-1。

如果仅关心源字符串向量 text 中哪些元素能匹配 pattern，可以用 grep 函数，如

```
x <- c("New theme", "Old times", "In the present theme")
grep("the", x, perl=TRUE)
```

```
## [1] 1 3
```

结果说明源字符串向量的三个元素中仅有第 1、第 3 号元素能匹配。如果都不匹配，返回 integer(0)。

grep 可以使用与 regexpr 相同的自变量，另外还可以加选项 invert=TRUE，这时返回的是不匹配的元素的下标。

grep() 如果添加选项 value=TRUE，则结果不是返回有匹配的元素的下标而是返回有匹配的元素本身（不是匹配的子串），如

```
x <- c("New theme", "Old times", "In the present theme")
grep("the", x, perl=TRUE, value=TRUE)
```

```
## [1] "New theme"            "In the present theme"
```

grepl 的作用与 grep 类似，但是其返回值是一个长度与源字符串向量 text 等长的逻辑型向量，每个元素的真假对应于源字符串向量中对应元素的匹配与否。如

```
x <- c("New theme", "Old times", "In the present theme")
grepl("the", x, perl=TRUE)
```

```
## [1]  TRUE FALSE  TRUE
```

就像 grep() 与 grepl() 本质上给出相同的结果，只是结果的表示方式不同，regexec() 与 regexpr() 也给出仅在表示方式上有区别的结果。regexpr() 主要的结果是每个元素的匹配位置，用一个统一的属性返回各个匹配长度；regexec() 则返回一个与源字符串向量等长的列表，列表的每个元素为匹配的位置，并且列表的每个元素有匹配长度作为属性。所以，这两个函数只需要用其中一个就可以，下面仅使用 regexpr()。regexec() 的使用效果如

```
x <- c("New theme", "Old times", "In the present theme")
regexec("the", x, perl=TRUE)
```

```
## [[1]]
## [1] 5
## attr(,"match.length")
## [1] 3
## attr(,"useBytes")
## [1] TRUE
## attr(,"index.type")
## [1] "chars"
##
## [[2]]
## [1] -1
## attr(,"match.length")
## [1] -1
## attr(,"useBytes")
## [1] TRUE
## attr(,"index.type")
## [1] "chars"
##
## [[3]]
## [1] 4
## attr(,"match.length")
## [1] 3
## attr(,"useBytes")
## [1] TRUE
## attr(,"index.type")
## [1] "chars"
```

grep(), grepl(), regexpr(), regexec() 都只能找到源字符串向量的每个元素中模式的首次匹配，不能找到所有匹配。gregexpr() 函数可以找到所有匹配。如

```
x <- c("New theme", "Old times", "In the present theme")
gregexpr("the", x, perl=TRUE)
```

```
## [[1]]
## [1] 5
## attr(,"match.length")
## [1] 3
```

```
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[2]]
## [1] -1
## attr(,"match.length")
## [1] -1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[3]]
## [1]  4 16
## attr(,"match.length")
## [1] 3 3
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

其结果是一个与源字符串向量等长的列表，格式与 regexec() 的结果格式类似，列表的每个元素对应于源字符串向量的相应元素，列表元素值为匹配的位置，并有属性 match.length 保存了匹配长度。匹配位置和匹配长度包含了所有的匹配，见上面例子中第三个元素的匹配结果。

函数 grep, grepl 结果仅给出每个元素能否匹配。regexpr(), regexec(), gregexpr() 则包含了匹配位置与匹配长度，这时，可以用 regmatches() 函数取出具体的匹配字符串。regmatches() 一般格式为

```
regmatches(x, m, invert = FALSE)
```

其中 x 是源字符串向量，m 是 regexpr()、regexec() 或 gregexpr() 的匹配结果。如

```
x <- c("New theme", "Old times", "In the present theme")
m <- regexpr("the", x, perl=TRUE)
regmatches(x, m)
```

```
## [1] "the" "the"
```

可以看出，regmatches() 仅取出有匹配时的匹配内容，无匹配的内容被忽略。

取出多处匹配的例子如：

```
x <- c("New theme", "Old times", "In the present theme")
m <- gregexpr("the", x, perl=TRUE)
regmatches(x, m)
```

```
## [[1]]
## [1] "the"
##
## [[2]]
## character(0)
##
## [[3]]
## [1] "the" "the"
```

当 regmatches() 第二个自变量是 gregexpr() 的结果时，其输出结果变成一个列表，并且不再忽略无匹配的元素，无匹配元素对应的列表元素为 character(0)，即长度为零的字符型向量。对有匹配的元素，对应的列表元素为所有的匹配字符串组成的字符型向量。

实际上，如果 pattern 中没有正则表达式，grep(), grepl(), regexpr(), gregexpr() 中都可以用 fixed=TRUE 参数取代 perl=TRUE 参数，这时匹配总是解释为原样匹配，即使 pattern 中包含特殊字符也是进行原样匹配。

42.3.1 不区分大小写匹配

在基本 R 中，为了不区分大小写匹配，可以在 grep 等函数调用时加选项 ignore.case=TRUE；如

```
grep("Dr", c("Dr. Wang", "DR. WANG", "dR. W.R."))
```

```
## [1] 1
```

```
grep("dr", c("Dr. Wang", "DR. WANG", "dR. W.R."), ignore.case=TRUE)
```

```
## [1] 1 2 3
```

```
grep("(?i)dr", c("Dr. Wang", "DR. WANG", "dR. W.R."))
```

```
## [1] 1 2 3
```

42.3.2 匹配单个字符

在模式中用 “.” 匹配任意一个字符（除了换行符"\n"，能否匹配此字符与选项有关）。如

```
s <- c("abc", "cabs", "lab")
mres <- regexpr("ab.", s, perl=TRUE); mres
```

```
## [1]  1  2 -1
## attr(,"match.length")
## [1]  3  3 -1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

regexpr 仅给出每个元素中模式的首次匹配位置而不是给出匹配的内容。regmatches 函数以原始字符型向量和匹配结果为输入，结果返回每个元素中匹配的各个子字符串 (不是整个元素)，如:

```
regmatches(s, mres)
```

```
## [1] "abc" "abs"
```

注意返回结果和输入字符型向量元素不是一一对应的，仅返回有匹配的结果。

像句点这样的字符称为元字符（meta characters），在正则表达式中有特殊函数。如果需要匹配句点本身，用 “[.]” 或者 “\.” 表示。比如，要匹配 a.txt 这个文件名，如下做法有错误：

```
grep("a.txt", c("a.txt", "a0txt"), perl=TRUE)
```

```
## [1] 1 2
```

结果连 a0txt 也匹配了。用 “[.]” 表示句点则将句点不做特殊解释：

```
grep("a[.]txt", c("a.txt", "a0txt"), perl=TRUE)
```

```
## [1] 1
```

```
grep("a\\.txt", c("a.txt", "a0txt"), perl=TRUE)
```

```
## [1] 1
```

注意在 R 语言字符型常量中一个\需要写成两个。

如果仅需按照原样进行查找，也可以在 grep(), grepl(), regexpr(), gregexpr() 等函数中加选项 fixed=TRUE，这时不要再用 perl=TRUE 选项。如

```
grep("a.txt", c("a.txt", "a0txt"), fixed=TRUE)
```

```
## [1] 1
```

42.3.3 匹配一组字符中的某一个

模式“[ns]a.[.]xls”表示匹配的第一个字符是 n 或 s，第二个字符是 a，第三个字符任意，第四个字符是句点，然后是 xls。例：

```
regexpr("[ns]a.[.]xls", c("sa1.xls", "dna2.xlss", "na3.xls"), perl=T)
```

```
## [1] 1 2 1
## attr(,"match.length")
## [1] 7 7 7
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

42.3.4 原样匹配元字符

例：

```
grep("int x\\[5\\]", c("int x;", "int x[5]"), perl=TRUE)
```

```
## [1] 2
```

也可以用“[[]”表示“[”，用“[]]”表示“]”，如

```
grep("int x[[]5[]]", c("int x;", "int x[5]"), perl=TRUE)
```

```
## [1] 2
```

42.3.5 匹配数字

例：

```
grep("n\\d[.]xls", c("n1.xls", "na.xls"), perl=TRUE)
```

```
## [1] 1
```

42.3.6 匹配开头和末尾

例:

```
grep("^n\\d[.]xls$", c("n1.xls", "na.xls", "cn1.xls", "n1.xlsx"), perl=TRUE)
```

```
## [1] 1
```

```
grep("\\An\\d[.]xls\\Z", c("n1.xls", "na.xls", "cn1.xls", "n1.xlsx"), perl=TRUE)
```

```
## [1] 1
```

只匹配了第一个输入字符串。

42.3.7 匹配字母、数字、下划线

例：

```
m <- regexpr("s\\w[.]", c("file-s1.xls", "s#.xls"), perl=TRUE)
regmatches(c("file-s1.xls", "s#.xls"), m)
```

```
## [1] "s1."
```

可以看出，模式匹配了 `s1.` 而没有匹配 `s#.`。

42.3.8 十六进制和八进制数

例如

```
gregexpr("\\x0A", "abc\nefg\n")[[1]]
```

```
## [1] 4 8
## attr(,"match.length")
## [1] 1 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

匹配了两个换行符。

42.3.9 POSIX 字符类

例如：

```
grep("[[:alpha:]_.][[:alnum:]_.]", c("x1", "_x", ".x", ".1"))
```

```
## [1] 1 2 3 4
```

42.3.10 加号重复匹配

例

```
s <- c("sa1", "dsa123")
mres <- regexpr("sa[[:digit:]]+", s, perl=TRUE)
regmatches(s, mres)
```

```
## [1] "sa1"   "sa123"
```

例如：

```
p <- "^[[:alnum:]_]+@[[:alnum:]_]+[.][[:alnum:]_]+$"
x <- "abc123@efg.com"
m <- regexpr(p, x, perl=TRUE)
regmatches(x, m)
```

```
## [1] "abc123@efg.com"
```

匹配的电子邮件地址在 `@` 前面可以使用任意多个字母、数字、下划线，在 `@` 后面由小数点分成两段，每段可以使用任意多个字母、数字、下划线。这里用了 `^` 和 `$` 表示全字符串匹配。

42.3.11 星号和问号重复匹配

`^https?://[[:alnum:]./]+$` 可以匹配 http 或 https 开始的网址。如

```
s <- c("http://www.163.net", "https://123.456.")
grep("^https?://[[:alnum:]_./]+$", s, perl=TRUE)
```

```
## [1] 1 2
```

(注意第二个字符串不是合法网址但是按这个正则表达式也能匹配)

42.3.12 计数重复

例：

```
grep("[[:digit:]]{3}", c("1", "12", "123", "1234"))
```

```
## [1] 3 4
```

模式匹配的是三位的数字。

日期匹配例：

```
pat <- paste(
  c("[[:digit:]]{1,2}[-/]",
    "[[:digit:]]{1,2}[-/]",
    "[[:digit:]]{2,4}"), collapse="")
grep(pat, c("2/4/1998", "13/15/198"))
```

```
## [1] 1 2
```

42.3.13 贪婪匹配和懒惰匹配

例如：

```
s <- "<B>1st</B> other <B>2nd</B>"
p1 <- "<[Bb]>.*</[Bb]>"
m1 <- regexpr(p1, s, perl=TRUE)
regmatches(s, m1)[[1]]
```

```
## [1] "<B>1st</B> other <B>2nd</B>"
```

我们本来期望的是提取第一个“<B>……</B>”组合，不料提取了两个“<B>……</B>”组合以及中间的部分。

比如，上例中模式修改后得到了期望的结果：

```
s <- "<B>1st</B> other <B>2nd</B>"
p2 <- "<[Bb]>.*?</[Bb]>"
m2 <- regexpr(p2, s, perl=TRUE)
regmatches(s, m2)[[1]]
```

```
## [1] "<B>1st</B>"
```

42.3.14 单词边界

例：

```
grep("\\bcat\\b", c("a cat meaos", "the category"))
```

```
## [1] 1
```

42.3.15 句点全匹配与多行模式

句点通配符一般不能匹配换行，如

```
s <- "<B>1st\n</B>\n"
grep("<[Bb]>.*?</[Bb]>", s, perl=TRUE)
```

```
## integer(0)
```

跨行匹配失败。而在 HTML 的规范中换行是正常的。一种办法是预先用 `gsub` 把所有换行符替换为空格。但是这只能解决部分问题。

另一方法是在 Perl 正则表达式开头添加 `(?s)` 选项，这个选项使得句点通配符可以匹配换行符。如

```
s <- "<B>1st\n</B>\n"
mres <- regexpr("(?s)<[Bb]>.*?</[Bb]>", s, perl=TRUE)
regmatches(s, mres)
```

```
## [1] "<B>1st\n</B>"
```

多行模式例：

```
s <- "<B>1st\n</B>\n"
mres1 <- gregexpr("^<.+?>", s, perl=TRUE)
mres2 <- gregexpr("(?m)^<.+?>", s, perl=TRUE)
regmatches(s, mres1)[[1]]
```

```
## [1] "<B>"
```

```
regmatches(s, mres2)[[1]]
```

```
## [1] "<B>"  "</B>"
```

字符串 s 包含两行内容，中间用\n 分隔。mres1 的匹配模式没有打开多行选项，所以模式中的 ^ 只能匹配 s 中整个字符串开头。mres2 的匹配模式打开了多行选项，所以模式中的 ^ 可以匹配 s 中每行的开头。

42.3.16 备择模式

例如，某个人的名字用 James 和 Jim 都可以，表示为 James|Jim, 如

```
s <- c("James, Bond", "Jim boy")
pat <- "James|Jim"
mres <- gregexpr(pat, s, perl=TRUE)
regmatches(s, mres)
```

```
## [[1]]
## [1] "James"
##
## [[2]]
## [1] "Jim"
```

42.3.17 分组与捕获

例：希望把 “<B>……</B” 两边的 “<B>” 和 “</B>” 删除，可以用如下的替换方法：

```
x <- "<B>1st</B> other <B>2nd</B>"
pat <- "(?s)<[Bb]>(.+?)</[Bb]>"
repl <- "\\1"
gsub(pat, repl, x, perl=TRUE)
```

```
## [1] "1st other 2nd"
```

替换模式中的\1(写成 R 字符型常量时\要写成\\) 表示第一个圆括号匹配的内容，但是表示选项的圆括号（(?s)）不算在内。

例：希望把带有前导零的数字的前导零删除，可以用如

```
x <- c("123", "0123", "00123")
pat <- "\\b0+([1-9][0-9]*)\\b"
repl <- "\\1"
gsub(pat, repl, x, perl=TRUE)
```

```
## [1] "123" "123" "123"
```

其中的\b 模式表示单词边界，这可以排除在一个没有用空格或标点分隔的字符串内部拆分出数字的情况。

例：为了交换横纵坐标，可以用如下替换

```
s <- "1st: (5,3.6), 2nd: (2.5, 1.1)"
pat <- paste0(
  "[(]([[:digit:].]+),",
  "[[:space:]]*([[:digit:].]+)[)]")
repl <- "(\\2, \\1)"
gsub(pat, repl, s, perl=TRUE)
```

```
## [1] "1st: (3.6, 5), 2nd: (1.1, 2.5)"
```

例如，要匹配 yyyy-mm-dd 这样的日期，并将其改写为 mm/dd/yyyy，就可以用这样的替换模式：

```
pat <- "([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})"
repl <- "\\2/\\3/\\1"
gsub(pat, repl, c("1998-05-31", "2017-01-14"))
```

```
## [1] "05/31/1998" "01/14/2017"
```

分组除了可以做替换外，还可以用来表示模式中的重复出现内容。例如，([a-z]{3})\1 这样的模式可以匹配如 abcabc, uxzuxz 这样的三字母重复。如

```
grep("([a-z]{3})\\1", c("abcabc", "aabbcc"))
```

```
## [1] 1
```

又例如，下面的程序找出了年（后两位）、月、日数字相同的日期：

```
x <- c("2008-08-08", "2017-01-18")
m <- regexpr("\\d\\d(\\d\\d)-\\1-\\1", x)
regmatches(x, m)
```

```
## [1] "2008-08-08"
```

下面是一个非捕获分组示例。设需要把 1921-2020 之间的世纪号删去，可以用

```
pat <- "\\A(?:19|20)([0-9]{2})\\Z"
repl <- "\\1"
x <- c("1978", "2017", "2035")
gsub(pat, repl, x, perl=TRUE)
```

```
## [1] "78" "17" "35"
```

其中用了非捕获分组使得备择模式 19|20 优先匹配。注意模式并没有能保证日期在 1921-2020 之间。更周密的程序可以写成：

```
x <- c("1978", "2017", "2035")
p1 <- "\\A19(2[1-9]|[3-9][0-9])\\Z"
r1 <- "\\1"
p2 <- "\\A20([01][0-9]|20)\\Z"
x <- gsub(p1, r1, x, perl=TRUE)
x <- gsub(p2, r1, x, perl=TRUE)
x
```

```
## [1] "78"   "17"   "2035"
```

Chapter 43

文本应用

43.1 正则表达式应用例子

43.1.1 数据预处理

在原始数据中，经常需要审核数据是否合法，已经把一些常见错误输入自动更正。这都可以用正则表达式实现。

43.1.1.1 除去字符串开头和结尾的空格

函数 `stringr::str_trim()` 和 `base::trimws()` 可以除去字符串开头与结尾的空格，也可以仅除去开头或仅除去结尾的空格。

这个任务如果用正则表达式字符串替换函数来编写，可以写成：

```
### 把字符串向量 x 的元素去除首尾的空白。
strip <- function(x){
  x <- str_replace_all(x, "^[[:space:]]+", "")
  x <- str_replace_all(x, "[[:space:]]+$", "")
  x
}
```

或者

```
### 把字符串向量 x 的元素去除首尾的空白。
strip <- function(x){
  x <- gsub("^[[:space:]]+", "", x, perl=TRUE)
  x <- gsub("[[:space:]]+$", "", x, perl=TRUE)
  x
}
```

这个版本可以除去包括空格在内的所有首尾空白字符。

43.1.1.2 除去字符串向量每个元素中所有空格

```
compress <- function(x){
  str_replace_all(x, " ", "")
}
```

或者

```
compress <- function(x){
  gsub(" ", "", x, fixed=TRUE)
}
```

这可以解决" 李明" 与" 李　明" 不相等这样的问题。类似的程序也可以用来把中文的标点替换成英文的标点。

43.1.1.3 判断日期是否合法

设日期必须为 yyyy-mm-dd 格式，年的数字可以是两位、三位、四位，程序为:

```
is_yyyymmdd <- function(x){
  pyear <- "([0-9]{2}|[1-9][0-9]{2}|[1-2][0-9]{3})"
  pmon <- "([1-9]|0[1-9]|1[0-2])"
  pday <- "([1-9]|0[1-9]|1[0-9]|2[0-9]|3[01])"
  pat <- paste0("\\A", pyear, "-", pmon, "-", pday, "\\Z")
  str_detect(x, pat)
}
```

或

```
is.yyyymmdd <- function(x){
  pyear <- "([0-9]{2}|[1-9][0-9]{2}|[1-2][0-9]{3})"
  pmon <- "([1-9]|0[1-9]|1[0-2])"
  pday <- "([1-9]|0[1-9]|1[0-9]|2[0-9]|3[01])"
  pat <- paste("\\A", pyear, "-", pmon, "-", pday, "\\Z", sep="")
  grepl(pat, x, perl=TRUE)
}
```

这样的规则还没有排除诸如 9 月 31 号、2 月 30 号这样的错误。

例:

```
x <- c("49-10-1", "1949-10-01", "532-3-15", "2015-6-1",
       "2017-02-30", "2017-13-11", "2017-1-32")
is_yyyymmdd(x)
```

```
## [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
```

注意错误的 2 月 30 号没有识别出来。

43.1.1.4 把字符型日期变成 yyyy-mm-dd 格式。

```
make_date <- function(x){
  x |>
    str_trim() |>
    str_replace_all("[[:space:]]+", "-") |>
    str_replace_all("/", "-") |>
    str_replace_all("[.]", "-") |>
    str_replace_all(
      "^([0-9]{2})(-[0-9]{1,2}-[0-9]{1,2})$",
      "20\\1\\2") |>
    str_replace_all(
      "^([0-9]{4})-([0-9])-([0-9]{1,2})$",
      "\\1-0\\2-\\3") |>
    str_replace_all(
      "^([0-9]{4}-[0-9]{2})-([0-9])$",
      "\\1-0\\2")
}
```

或者

```
make.date <- function(x){
  x <- trimws(x)
  x <- gsub("[[:space:]]+", "-", x)
  x <- gsub("/", "-", x)
  x <- gsub("[.]", "-", x)
  x <- gsub("^([0-9]{2})(-[0-9]{1,2}-[0-9]{1,2})$",
            "20\\1\\2", x)
  x <- gsub("^([0-9]{4})-([0-9])-([0-9]{1,2})$",
            "\\1-0\\2-\\3", x)
  x <- gsub("^([0-9]{4}-[0-9]{2})-([0-9])$",
```

```
            "\\1-0\\2", x)

  x
}
```

另一办法是用 `strsplit()` 拆分出三个部分，转换为整数，再转换回字符型。

测试:

```
x <- c("49/10/1", "1949.10.01", "532 3 15", "2015/6.1",
       "20170230", "2017.13/11", "2017 1 32")
make_date(x)
```

```
## [1] "2049-10-01" "1949-10-01" "532-3-15"   "2015-06-01" "20170230"
## [6] "2017-13-11" "2017-01-32"
```

目前的函数还不能处理没有分隔符的情况，也不能验证日期的合法性。

43.1.1.5 合并段落为一行

在某些纯文本格式中，各段之间用空行分隔，没有用空行分隔的各行看成同一段。如下的函数把其中的不表示分段的换行删去从而合并这些段落。函数以一个文件名作为输入，合并段落后存回原文件。注意，这样修改文件的函数在调试时，应该注意先备份文件，等程序没有任何错误以后才可以忽略备份。

```
combine_paragraph <- function(fname, encoding="UTF-8"){
  lines <- readr::read_lines(
    fname, locale=locale(encoding=encoding))

  s <- str_flatten(lines, collapse="\n")
  s <- s |>
    # 含有空格的空行去掉空格
    str_replace_all("^[[:space:]]+\n", "\n") |>
    # 有内容行的换行替换成空格
    str_replace_all("([^\n]+)\n", "\\1 ") |>
    # 段落换行增加一个空行
    str_replace_all("([^\n]+)\n", "\\1\n\n") |>
    # 多个空行换成一个空行
    str_replace_all("\n{3,}", "\n\n")

  write_lines(str_split(s, "\n")[[1]],
```

```
     file=fname)
}
```

下面的版本不使用 stringr 和 readr 包：

```
combine.paragraph <- function(
    fname, encoding="UTF-8"){
  lines <- readLines(fname, encoding = encoding)
  s <- paste(lines, collapse="\n") |>
    gsub("^[[:space:]]+\n", "\n", x = _, perl=TRUE) |>
    gsub("([^\n]+)\n", "\\1 ", x = _, perl=TRUE) |>
    gsub("([^\n]+)\n", "\\1\n\n", x = _, perl=TRUE) |>
    gsub("\n{3,}", "\n\n", x = _, perl=TRUE)
  lines <- strsplit(s, "\n", fixed=TRUE)[[1]]
  writeLines(lines, con=fname)
}
```

43.1.2 不规则 Excel 文件处理

- 作为字符型数据处理示例，考察如下的一个 Excel 表格数据。

假设一个中学把所有课外小组的信息汇总到了 Excel 表的一个工作簿中。每个课外小组占一块区域，各小组上下排列，但不能作为一个数据框读取。下图为这样的文件的一个简化样例：

实际数据可能有很多个小组，而且数据是随时更新的，所以复制粘贴另存的方法不太可行，需要用一个通用的程序处理。Excel 文件 (.xls 后缀或.xlsx 后缀) 不是文本型数据。在 Excel 中，用 “另存为” 把文件保存为 CSV 格式，内容如下：

```
XXX中学兴趣组情况总表,,,
,,,
组名：,物理,指导教师：,刘一心
姓名,性别,班级,
伊家宝,男,初二(3),
闻月,女,初二(5),
刘阳,男,初三(1),
宋佰霖,男,初三(2),
洪晓梅,女,初三(1),
,,,
组名：,生物,指导教师：,赵晓辉
姓名,性别,班级,
```

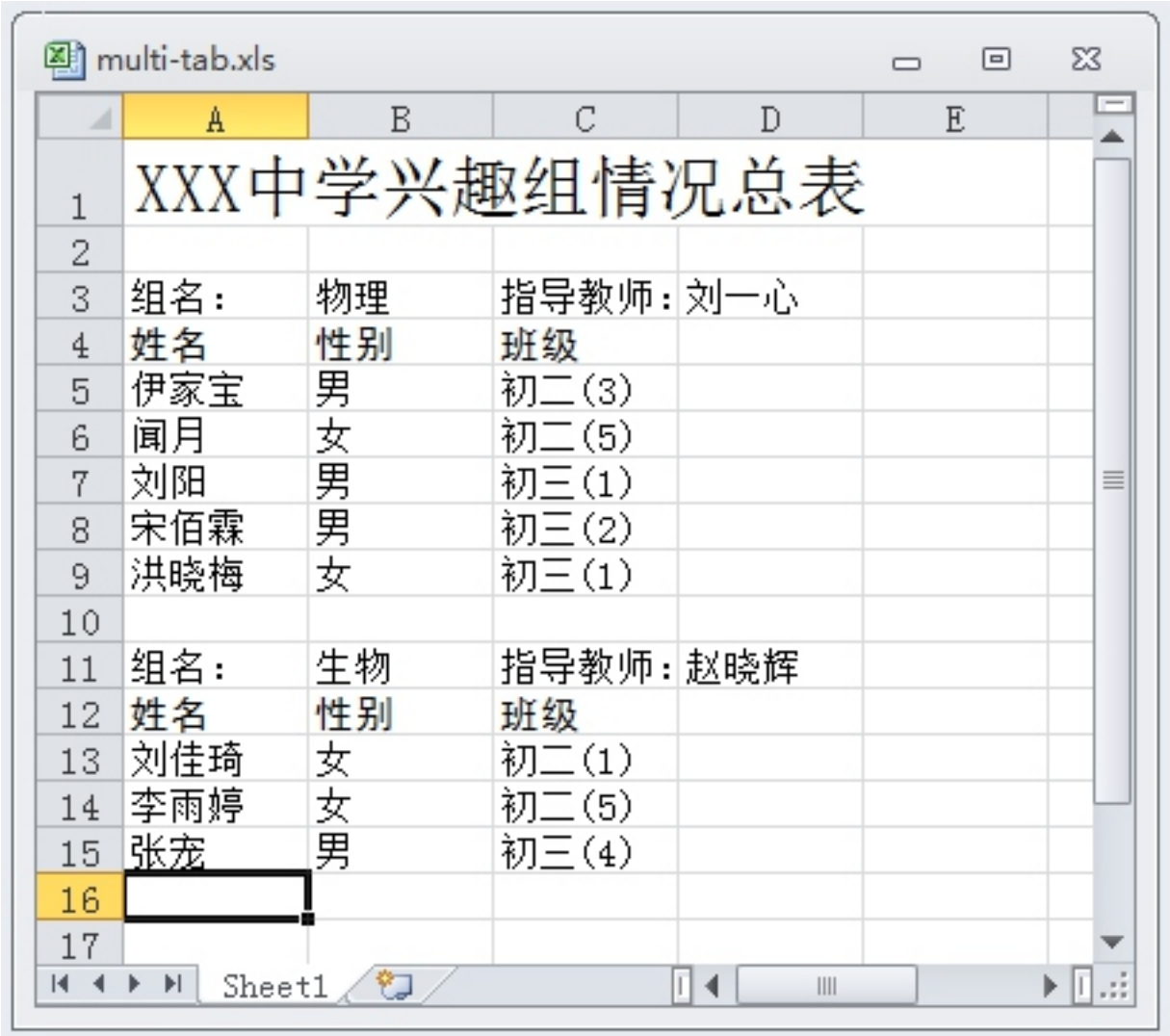


图 43.1: 不规则 Excel 文件样例图形

刘佳琦,女,初二(1),
李雨婷,女,初二(5),
张宠,男,初三(4),

生成测试用的数据文件：

```
demo.multitab.data <- function(){
s <- "
XXX 中学兴趣组情况总表,,,
,,,
组名：，物理，指导教师：，刘一心
姓名，性别，班级，
伊家宝，男，初二（3），
闻月，女，初二（5），
刘阳，男，初三（1），
宋佰霖，男，初三（2），
洪晓梅，女，初三（1），
,,,
组名：，生物，指导教师：，赵晓辉
姓名，性别，班级，
刘佳琦，女，初二（1），
李雨婷，女，初二（5），
张宠，男，初三（4），
"
writeLines(s, "data/multitab.csv")
}
demo.multitab.data()
```

读入测试用的数据，转换为一整个数据框:

```
demo_multitab <- function(){
  ## 读入所有行
  lines <- readr::read_lines("data/multitab.csv")

  ## 去掉首尾空格
  lines <- str_trim(lines)

  ## 删去所有空行和只有逗号的行
  empty <- str_detect(lines, "^[[:space:],]*$")
  if(length(empty)>0){
```

```
    lines <- lines[!empty]
  }

  ## 找到所有包含 “组名:” 的行对应的行号
  heads <- str_which(lines, " 组名: ")

  ## 定位每个表的开始行和结束行 (不包括组名和表头所在的行)
  start <- heads + 2
  end <- c(heads[-1]-1, length(lines))
  ngroups <- length(heads)

  for(ii in seq(ngroups)){
    ## 组名和指导教师所在行:
    line <- lines[heads[ii]]
    v <- str_split(line, ",")[[1]]
    ## 组名: v[2] 指导教师: v[4]

    ## 将表格内容各行合并成一个用换行符分隔的长字符串,
    ## 然后变成可读取的文件
    s <- str_flatten(lines[start[ii]:end[ii]], collapse="\n")
    da1 <- readr::read_csv(
      s, col_names = c(" 姓名", " 性别", " 班级", "xxx"),
      col_types = cols(
        .default=col_character(),
        xxx = col_skip()) )
    da1 <- bind_cols(tibble(" 组名"=v[2], " 指导教师"=v[4]), da1)

    if(ii==1) {
      da <- da1
    } else {
      da <- bind_rows(da, da1)
    }
  }

  da
}
da <- demo_multitab()
knitr::kable(da)
```

组名	指导教师	姓名	性别	班级
物理	刘一心	伊家宝	男	初二 (3)
物理	刘一心	闻月	女	初二 (5)
物理	刘一心	刘阳	男	初三 (1)
物理	刘一心	宋佰霖	男	初三 (2)
物理	刘一心	洪晓梅	女	初三 (1)
生物	赵晓辉	刘佳琦	女	初二 (1)
生物	赵晓辉	李雨婷	女	初二 (5)
生物	赵晓辉	张宠	男	初三 (4)

不使用 stringr 和 readr 的版本：

```
demo.multitab <- function(){
  ## 读入所有行
  lines <- readLines("data/multitab.csv")

  ## 去掉首尾空格
  lines <- trimws(lines)

  ## 删去所有空行和只有逗号的行
  ## (1) 不用正则表达式做法
  #empty <- which(lines == "" | substring(lines, 1, 3)==",,,")
  ## (2) 用正则表达式做法:
  empty <- grep("^[[:space:],]*$", lines)
  if(length(empty)>0){
    lines <- lines[-empty]
  }

  ## 找到所有包含 “组名:” 的行对应的行号
  heads <- grep(" 组名: ", lines, fixed=TRUE)

  ## 定位每个表的开始行和结束行 (不包括组名和表头所在的行)
  start <- heads + 2
  end <- c(heads[-1]-1, length(lines))
  ngroups <- length(heads)

  for(ii in seq(ngroups)){
    ## 组名和指导教师所在行:
    line <- lines[heads[ii]]
    v <- strsplit(line, ",")[[1]]
```

```
    ## 组名: v[2] 指导教师: v[4]

    ## 将表格内容各行合并成一个用换行符分隔的长字符串,
    ## 然后变成可读取的文件
    s <- paste(lines[start[ii]:end[ii]], collapse="\n")
    con <- textConnection(s, "rt")
    da1 <- read.csv(
      con, header=FALSE,
      colClasses=c("character", "character", "character", "NULL"))
    close(con)
    names(da1) <- c(" 姓名", " 性别", " 班级")
    da1 <- cbind(" 组名"=v[2], " 指导教师"=v[4], da1)

    if(ii==1) {
      da <- da1
    } else {
      da <- rbind(da, da1)
    }
  }

  da
}
da <- demo.multitab()
da
```

在程序中，用 readLines 函数读取文本文件各行到一个字符型向量。用 grep 可以找到每个小组开头的行（有“组名：”的行）。然后可以找出每个小组学生名单的开始行号和结束行号。各小组循环处理，读入后每个小组并入结果数据框中。用 strsplit 函数拆分用逗号分开的数据项。用 textConnection 函数可以把一个字符串当作文件读取，这样 read.csv 函数可以从一个字符串读入数据。

43.1.3 字频统计

正则表达式中的字符类 [:alpha:] 指的是当前系统中的字母，所以在中文环境中的中文字也是字母，但中文标点不算。下面是《红楼梦》中“秋窗风雨夕”的文本：

秋花惨淡秋草黄，耿耿秋灯秋夜长。
已觉秋窗秋不尽，那堪风雨助凄凉！
助秋风雨来何速！惊破秋窗秋梦绿。
抱得秋情不忍眠，自向秋屏移泪烛。

泪烛摇摇爇短檠，牵愁照恨动离情。
谁家秋院无风入？何处秋窗无雨声？
罗衾不奈秋风力，残漏声催秋雨急。
连宵脉脉复飕飕，灯前似伴离人泣。
寒烟小院转萧条，疏竹虚窗时滴沥。
不知风雨几时休，已教泪洒窗纱湿。

希望统计每个字的出现次数，并显示频数前十的字。设变量 poem_autumnwindow 中包含了上述诗词的文本。

首先用 str_extract_all() 提取每个中文字，组成一个字符型向量：

```
words_vec <- str_extract_all(poem_autumnwindow, "[[:alpha:]]")[[1]]
head(words_vec)
```

```
## [1] "秋" "花" "惨" "淡" "秋" "草"
```

用 tidyverse 方法：

用 table() 函数计算频数，并按频数排序，输出前 10 结果：

```
tibble(words = words_vec) |>
  count(words) |>
  arrange(desc(n)) |>
  slice_head(n=10)
```

```
## # A tibble: 10 x 2
##    words     n
##    <chr> <int>
##  1 秋       15
##  2 窗        5
##  3 雨        5
##  4 风        5
##  5 不        4
##  6 泪        3
##  7 何        2
##  8 助        2
##  9 声        2
## 10 已        2
```

或者，用 table() 函数计算频数：

```
words_freq <- sort(table(words_vec), decreasing=TRUE)
knitr::kable(head(words_freq, 10))
```

words_vec	Freq
秋	15
窗	5
风	5
雨	5
不	4
泪	3
灯	2
耿	2
何	2
离	2

43.1.4 数字验证

43.1.4.1 整数

字符串完全为十进制正整数的模式，写成 R 字符型常量：

```
"\\A[0-9]+\\Z"
```

这个模式也允许正整数以 0 开始，如果不允许以零开始，可以写成

```
"\\A[1-9][0-9]*\\Z"
```

对于一般的整数，字符串完全为十进制整数，但是允许前后有空格，正负号与数字之间允许有空格，模式可以写成：

```
"\\A[ ]*[+-]?[ ]*[1-9][0-9]*\\Z"
```

43.1.4.2 十六进制数字

字符串仅有十六进制数字，模式写成 R 字符型常量为

```
"\\A[0-9A-Fa-f]+\\Z"
```

在文中匹配带有 `0x` 前缀的十六进制数字，模式为

```
"\\b0x[0-9A-Fa-f]+\\b"
```

43.1.4.3 二进制数字

为了在文中匹配一个以 B 或 b 结尾的二进制非负整数，可以用

```
"\\b[01]+[Bb]\\b"
```

43.1.4.4 有范围的整数

1-12:

```
"\\b1[012]|[1-9]\\b"
```

1-24:

```
"\\b2[0-4]|1[0-9]|[1-9]\\b"
```

1-31:

```
"\\b3[01]|[12][0-9]|[1-9]\\b"
```

1900-2099:

```
"\\b(?:19|20)[0-9]{2}\\b"
```

这里的分组仅用于在 19 和 20 之间选择，不需要捕获，所以用了 (?: 的非捕获分组格式。

43.1.4.5 判断字符型向量每个元素是否数值

如下的 R 函数用了多种数字的正则表达式来判断字符型向量每个元素是否合法数值。

```
all_numbers <- function(x){
  x <- x |>
    str_replace_all("\\A[ ]+", "") |>
    str_replace_all("[ ]+\\Z", "")

  pint <- "\\A[+-]?[0-9]+\\Z"  # 整数，允许有前导零
  ## 浮点数 1，整数部分必须，小数部分可选，指数部分可选
  pf1 <- "\\A[+-]?[0-9]+([.][0-9]*)?([Ee][+-]?[0-9]+)?\\Z"
```

```
  ## 浮点数 2，整数部分省略，小数部分必须，指数部分可选
  pf2 <- "\\A[+-]?[.][0-9]+([Ee][+-]?[0-9]+)?\\Z"
  pat <- str_c(pint, pf1, pf2, sep="|")

  str_detect(x, pat)
}
```

不使用 stringr 的版本：

```
all.numbers <- function(x){
  x <- gsub("\\A[ ]+", "", x, perl=TRUE)
  x <- gsub("[ ]+\\Z", "", x, perl=TRUE)

  pint <- "\\A[+-]?[0-9]+\\Z"  # 整数，允许有前导零
  ## 浮点数 1，整数部分必须，小数部分可选，指数部分可选
  pf1 <- "\\A[+-]?[0-9]+([.][0-9]*)?([Ee][+-]?[0-9]+)?\\Z"
  ## 浮点数 2，整数部分省略，小数部分必须，指数部分可选
  pf2 <- "\\A[+-]?[.][0-9]+([Ee][+-]?[0-9]+)?\\Z"
  pat <- paste(pint, pf1, pf2, sep="|")
  grepl(pat, x, perl=TRUE)
}
```

测试：

```
all_numbers(c("1", "12", "-12", "12.", "-12.",
              "123.45", "-123.45", ".45", "-.45",
              "1E3", "-12E-10", "1.1E3", "-1.1E-3",
              "1.0.0", "1.0-0.5"))
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [13]  TRUE FALSE FALSE
```

43.1.5 文件名中的数字提取

设有如下的一些文件名：

```
s <- c("10-0.16-1700.0-42.csv", "12-0.22-1799.1.csv")
```

希望提取出每个文件名中用减号分隔开的数字，如第一个文件名中的 10, 0.16, 1700.0, 42, 第二个文件名中的 12, 0.22, 1799.1，数字的个数不需要相同。

先定义数字的模式，注意长备择模式中长的模式要写在前面，否则会被短的模式优先匹配：

```
pat <- "[0-9]+[.][0-9]+|[0-9]+"
```

用 stringr::str_match_all() 提取其中的匹配数字：

```
s1 <- str_match_all(s, pat); s1
```

```
## [[1]]
##      [,1]
## [1,] "10"
## [2,] "0.16"
## [3,] "1700.0"
## [4,] "42"
##
## [[2]]
##      [,1]
## [1,] "12"
## [2,] "0.22"
## [3,] "1799.1"
```

每个列表元素是一个矩阵，其中第一列的各行是对模式的多次匹配，取出这些匹配为一个字符型向量：

```
s2 <- s |>
  str_match_all(pat) |>
  map(function(x) x[,1])
s2
```

```
## [[1]]
## [1] "10"     "0.16"   "1700.0" "42"
##
## [[2]]
## [1] "12"     "0.22"   "1799.1"
```

如果需要，也可以将拆分出的字符型的数字结果转换成数值型：

```
s3 <- s |>
  str_match_all(pat) |>
  map(function(x) x[,1]) |>
  map(parse_double)
s3
```

```
## [[1]]
## [1]   10.00    0.16 1700.00   42.00
##
## [[2]]
## [1]   12.00    0.22 1799.10
```

这个问题也可以用 `strsplit()` 或者 `stringr::str_split()` 解决。如：

```r
s <- c("10-0.16-1700.0-42.csv", "12-0.22-1799.1.csv")
s1 <- substring(s, 1, nchar(s)-4) # 去掉".csv"
s2 <- strsplit(s1, "[-]")  # 按减号分成几个部分，结果为列表
s3 <- lapply(s2, as.numeric)  # 转换为数值型
print(s3)
```

```
## [[1]]
## [1]   10.00    0.16 1700.00   42.00
##
## [[2]]
## [1]   12.00    0.22 1799.10
```

43.2 网站数据获取

很多网站定期频繁发布数据，所以传统的手工复制粘贴整理是不现实的。有些网站提供了下载功能，有些则仅能显示。

如果网页不是依赖 JavaScript 来展示的话，可以读取网页然后通过字符型数据处理方法获得数据。一般地，可以用 `download.file()` 下载网页文件，这是特殊的文本文件，其中有许多标记（tags），如 `<html>`, `</html>`, `<head>`, `</head>`, `<body>`, `</body>`, `<p>`, `</p>`，`<table>`, `</table>`, `<tr>`, `</tr>`, `<th>`, `</th>`, `<td>`, `</td>`，等等。用 `str_replace_all()` 或者 `gsub()` 去掉不需要的成分，用 `str_which()` 或者 `grep` 查找关键行。

许多数据网页有固定模式并且含有数据，R 扩展包 rvest 可以对网页按照其中的网页构成节点路径 (xpath) 提取数据，转换为 R 数据框。

以上海证券交易所的上证综指成份股列表为例。使用 Google Chrome 浏览器打开如下的页面：

http://www.sse.com.cn/market/sseindex/indexlist/s/i000001/const_list.shtml

将显示上证综指成份股的名称与编码的列表页面。利用 Chrome 浏览器的功能先获取表格所在页面部分的 xpath，办法是鼠标右键单击表格开头部分，选择“检查”（inspect），这时会在浏览器右边打开一个 html 源代码窗口，当前加亮显示部分是表格开头内容的源代码，将鼠标单击到上层的 `<table class="tablestyle">` 处，右键单击选择“Copy-Copy XPath”，得到如下的 xpath 地址：`'//*[@id="content_ab"]/div[1]/table'`。

然后，用 rvest 的 `read_html()` 将 HTML 页面内容下载为 R 变量，用 `html_nodes()` 函数提取页面中用 xpath 指定的成分，用 `html_table()` 函数将 HTML 表格转换为数据框，结果是一个数据框列表，因为仅有一个，所以取列表第一项即可。程序如下：

```
library(rvest)

## 网页地址
urlb <- paste0("http://www.sse.com.cn/",
  "market/sseindex/indexlist/s/i000001/const_list.shtml")
## 网页中数据表的 xpath
xpath <- '//*[@id="content_ab"]/div[1]/table'

## 读入网页
html_var <- read_html(urlb)

## 提取读入的页面内容中的表格节点
nodes <- html_nodes(html_var, xpath=xpath)

## 从表格节点转换为表格列表
tables <- html_table(nodes)
restab <- tables[[1]]
head(restab)
##                                  X1                                  X2
## 1 浦发银行\r\n            (600000) 白云机场\r\n            (600004)
## 2 中国国贸\r\n            (600007) 首创股份\r\n            (600008)
##                                 X3
## 1 东风汽车\r\n            (600006)
## 2 上海机场\r\n            (600009)
```

可见每一行有三个股票，我们将数据中的\r\n 和空格去掉，然后转换成名称与代码分开的格式：

```
library(tidyverse)

pat1 <- "^(.*?)\\((.*?)\\)"
tab1 <- restab |>
  ## 将三列纵向合并为一列，结果为字符型向量
  reduce(c) %>%
  ## 去掉空格和换行符，结果为字符型向量
  str_replace_all("[[:space:]]", "") %>%
  ## 提取公司简称和代码到一个矩阵行，结果为字符型矩阵
```

```
  str_match(pat1)
tab <- tibble(
  name = tab1[,2],
  code = tab1[,3])
head(tab)
## # A tibble: 6 x 2
##   name     code
##   <chr>    <chr>
## 1 浦发银行 600000
## 2 中国国贸 600007
## 3 包钢股份 600010
## 4 华夏银行 600015
## 5 上港集团 600018
## 6 上海电力 600021
```

```
str(tab)
## Classes 'tbl_df', 'tbl' and 'data.frame':    1551 obs. of  2 variables:
## $ name: chr  " 浦发银行" " 中国国贸" " 包钢股份" " 华夏银行" ...
##  $ code: chr  "600000" "600007" "600010" "600015" ...
```

有些网页是依靠 JavaScript 来显示数据的，比如新浪财经的环球股指汇总网页：

http://finance.sina.com.cn/money/globalindex/

在查看其页面源代码时，仅能看到一堆 JavaScrip 代码而看不到实际数据，这样的网页很难用程序提取数据。

关于 rvest 从 html 中提取数据的许多函数，可参见https://r4ds.hadley.nz/webscraping.html。

43.3 中文分词与词云

为了对中文文章进行分析，需要将文章内容拆分为一个个单词。R 扩展包 jiebaR 可以进行中文分词。用 `w <- worker()` 创建一个分词器，用 `segment(txt, w)` 对字符串 txt 中的中文内容进行分词，得到字符型向量，每个元素是一个词。

也可以调用 `segment(fname, w)`，其中 `fname` 是输入文本文件名，可以自动侦测其中的中文编码，分词结果会自动保存为文件开头和文件扩展名与 `fname` 相同的一个文件，词之间以空格分隔。

分词后，R 可以很容易地进行词频统计，如 `table()` 函数。

例如，对金庸的《侠客行》分词：

```
library(jiebaR)
```

```
## Loading required package: jiebaRD
```

```
wk <- worker()
txt <- readr::read_file("xkx.txt", locale=locale(encoding="GB18030"))
words <- segment(txt, wk)
tab <- table(words)
tab <- sort(tab, decreasing = TRUE)
## 去掉单个字的词语
tab2 <- tab[stringr::str_length(names(tab)) > 1]
knitr::kable(as.data.frame(head(tab2, 20)))
```

words	Freq
石破天	1800
什么	697
说道	602
自己	570
雪山	451
白万剑	443
丁当	443
一个	421
帮主	387
武功	381
石清	372
丁不四	345
谢烟客	340
一声	321
不是	319
二人	319
不知	308
咱们	304
史婆婆	291
夫妇	284

词频可以用“词云”数据可视化方式表现。在词云图形中，词频大的词显示为较大的字体。R 扩展包 wordcloud2 可以输入词频统计表，输出图形格式的词云显示，以 HTML5 格式显示。函数 `wordcloud2()` 可以输入 `table()` 的结果，或者有词和词频构成的两列的数据框。

```
library(wordcloud2)
wordcloud2(data = head(tab2, 20))
```

注意，这个库支持图形在 HTML 结果中显示，且具有一定交互性，但不直接支持 LaTeX 转换的 PDF 输出，所以需要进行设置。可以在 Rmd 源文件开头运行命令：

```
is_html <- knitr::opts_knit$get("rmarkdown.pandoc.to") == "html"
```

这可以定义一个变量 `is_html`，仅在输出格式为 HTML 时才为 `TRUE`，然后在包含特殊 HTML 显示的代码段选项中，加选项 `eval = is_html`。

ggwordcloud 包也提供了词云作图功能，没有提供交互功能。

Chapter 44

数据库访问

44.1 介绍

对于大型的数据，或者保存在多个表中的复杂数据，经常会保存在一个数据库中。数据库可以存在于专用的数据库服务器硬件上，也可以是本机中的一个系统程序，或者 R 直接管理的一个文件。

比较通用的数据库是关系数据库，这样的数据库已经有很标准的设计理念和管理方法，从用户使用的角度来看，都可以使用一种专用的 SQL 语言来访问和管理。

常用的数据库架构包括客户服务器式的，如 Oracle, SQL Server，比较适用于同一企业（机构）；基于云服务的，如 Snowflake，可扩展性更好；在一台电脑上仅适用于同一用户的，如 sqlite。

R 通过扩展包可以访问许多种常用的关系数据库系统，这些扩展包大多按照 DBI 扩展包规定的接口规范为用户提供了方便的访问功能。R 的 dbplyr 包使用与 tidyverse 风格相同的方法访问数据库，不需要使用 SQL 语言。

44.2 SQLite 数据库访问

SQLite 是一个开源的、轻量级的数据库软件，其数据库可以保存在本机的一个文件中，R 的 RSQLite 扩展包直接提供了 SQLite 数据库功能。如果自己的研究数据规模很大，比如有几个 GB，不利于整体读入到计算机内存当中，而每次使用时只需要其中的一个子集，就可以采用保存数据到 SQLite 数据库文件的方法。

学会了在 R 中使用 SQLite 数据库，其它的数据库也可以类似地使用。

44.2.1 NHANES 数据

我们以 NHANES 扩展包的 NHANES 数据框为例演示在 R 中访问关系数据库的方法。在数据库中一个数据框叫做一个表（table）。NHANES 表来自美国国家健康统计中心的调查数据，该调查项目从 1960 年代开始对美国非住院非军人住户进行健康与营养方面的抽样调查，从 1999 年开始大约 5000 各个年龄的受调查者每年在自己家接受调查员面访，完成调查中的健康检查部分，健康检查是在流动检查中心进行的。抽样调查有复杂的抽样设计，并

不是简单随机抽样。R 扩展包中 NHANES 数据框经过了重抽样使其尽可能与目标总体的各种比例一致，但数据应仅用作教学演示目的。NHANES 中有 10000 个观测，是在 2009-2010 和 2011-2012 两次得到的，有 75 个测试变量。部分变量为：

- SurveyYr：用来区分两次考察。
- ID：受试者编码。
- Gender: 性别，male 或 female。
- Age: 年龄，80 岁以上录入为 80。
- Race1: 种族，可取 Mexican, Hispanic, White, Black, 或 Other。
- Education：20 岁及以上受试者的受教育级别，可取 8thGrade, 9-11thGrade, HighSchool, SomeCollege, 或 CollegeGrad。
- MaritalStatus：婚姻状态，可取 Married, Widowed, Divorced, Separated, NeverMarried, 或 LivePartner（与伴侣生活）。
- Weight: 体重（千克）。
- Height: 身高（厘米），对 2 岁以上。

44.2.2 初始化新 SQLite 数据库

用使用 RSQLite，先载入 RSQLite 扩展包，然后指定一个 SQLite 数据库文件（不需要是已经存在的），用 `dbConnect()` 打开该数据库建立连接（如果没有就新建）：

```
library(RSQLite)
f_sqlite <- "_tmp/db2020.SQLITE"
con <- dbConnect(drv=SQLite(), dbname=f_sqlite)
```

可以用 `dbWriteTable()` 函数将 NHANES 数据框写入到打开的数据库中：

```
data(NHANES, package="NHANES")
dbWriteTable(conn=con, name="nh", value=NHANES)
```

写入后，数据库中的表名为 nh。

44.2.3 查看数据库中的表

假设连接 `con` 还保持打开状态，可以用 `dbListTables()` 查看数据库中有哪些表：

```
dbListTables(con)
## [1] "nh"
```

还可以用 `dbExistsTable("tablename")` 查看指定的表是否存在。

可以用 `dbListFields()` 查看某个表有哪些列，数据库中称为域（fields）：

```
dbListFields(con, "nh")
##  [1] "ID"                "SurveyYr"          "Gender"
##  [4] "Age"               "AgeDecade"         "AgeMonths"
##  ................
```

44.2.4 读入数据库中的表

为了从数据库读取整个数据表（数据框），建立连接后用 `dbReadTable()` 函数读取，如：

```
d1 <- dbReadTable(
  conn=con, name="nh")
d1 |>
  count(SurveyYr)
## # A tibble: 2 x 2
##   SurveyYr     n
##   <chr>    <int>
## 1 2009_10   5000
## 2 2011_12   5000
```

两次调查各有 5000 个观测。

44.2.5 用 SQL 命令访问数据

还可以用 `dbGetQuery()` 执行 SQL 查询并以数据框格式返回查询结果。比如，仅返回 SurveyYr 和 ID 两列，且仅选择男性：

```
d2 <- dbGetQuery(
  conn=con,
  statement=paste(
    "SELECT SurveyYr, ID",
    "  FROM nh",
    "  WHERE Gender='male'"))
d2 |>
  count(SurveyYr)
## # A tibble: 2 x 2
##   SurveyYr     n
##   <chr>    <int>
## 1 2009_10   2475
## 2 2011_12   2505
```

44.2.6 分批读入数据库中的表

对于特别大的表，可能超出了计算机的内存，无法整体读入到 R 当中。如果只需要一个子集，可以用上面的 dbGetQuery() 执行 SQL 命令在数据库中提取子集并仅将需要的子集读入到 R 中。如果需要读入的部分还是超过了内存，或者全部读入会使得处理速度很慢，可以分批读入，分批处理。

先用 dbSendQuery() 发送 SQL 查询命令：

```
qry <- dbSendQuery(
  conn=con,
  statement=paste(
    "SELECT Weight ",
    "  FROM nh",
    "  WHERE Gender='male'"))
```

然后，用 dbHasCompleted() 检查是否已经将结果取完，如果没有取完就用 dbFetch() 取出一定行数，分段处理：

```
s <- 0
n <- 0
while(!dbHasCompleted(qry)){
  chunk = dbFetch(qry, n=1000)
  n <- n + sum(!is.na(chunk[["Weight"]]))
  s <- s + sum(chunk[["Weight"]], na.rm=TRUE)
}
cat("Average Weight = ", s/n, "\n")
```

需要释放查询对应的资源：

```
dbClearResult(qry)
```

这段程序只是举例说明如何分段读入、分段处理，如果只是要计算 nh 表中男性的 Weight 变量平均值，可以读入整个 nh 表或者 nh 表中男性的 Weight 变量的所有值然后计算，或者用 SQL 命令计算。

44.2.7 关闭数据库连接

在数据库使用完毕以后，要记得关闭数据库连接：

```
dbDisconnect(con)
```

44.2.8 其它数据库操作

- 删除表：`dbRemoveTable(conn, name)`
- 检查某个表是否存在：`dbExistsTable(conn, name)`
- 向一个表中插入保存在数据框中的一些行：`dbAppendTable(conn, name, value)`
- 执行 SQL 命令并返回受到影响的行数：`dbExecute(con, statement)`

44.3 duckdb 数据库访问

duckdb 与 SQLite 数据库类似，也是作用于本地电脑的数据库，比较适用于数据科学的大数据量快速访问需求。

生成一个临时的空数据库并建立连接如：

```
library(duckdb)
drv <- duckdb()
con <- dbConnect(drv)
```

为了生成的空数据库能够保存使用，方法如：

```
library(duckdb)
drv <- duckdb(dbdir = "duckdb")
con <- dbConnect(drv)
```

这种方法也用于打开一个已有的 duckdb 数据库。

为了关闭打开的数据库连接和数据库服务器，方法如：

```
library(duckdb)
dbDisconnect(don, shutdown = TRUE)
```

44.4 用 dbplyr 包访问数据库

在数据库建立连接后，dbplyr 包的函数在添加一个一个 `con`(连接) 参数后，可以用与 dplyr 类似的方法访问并处理数据库中的表。

可以用 `tbl` 函数建立一个表的访问连接，这个访问连接在 R 中使用时和一个数据框相同，但仅在需要访问其中数据时，才从数据库中将数据下载到 R 中，所以类似于表的 “视图”。如 (设 `con` 为已建立的连接，其中 nh 表是 NHANES 数据集)：

```
dv.nh <- tbl(con, "nh")
dim(dv.nh)
```

因为没有实际将数据下载，所以现在显示的行数还是缺失值。

提取一个数据子集（也不会下载数据）：

```
dv.nh2 <- dv.nh |>
  select(ID, Gender, Age) |>
  filter(SurveyYr == "2011_12")
```

计算男女 2011 年观测频数：

```
dv.nh |>
  filter(SurveyYr == "2011_12") |>
  count(Gender)
```

```
# Source:   SQL [2 x 2]
# Database: DuckDB 0.6.0 [Lenovo@Windows 10 x64:R 4.2.1/:memory:]
  Gender     n
  <fct>  <dbl>
1 male    2505
2 female  2495
```

这些查询一般有对应的 SQL 命令，可以用 `show_query()` 函数显示对应的 SQL 命令，如：

```
dv.nh |>
  filter(SurveyYr == "2011_12") |>
  count(Gender) |>
  show_query()
```

```
<SQL>
SELECT Gender, COUNT(*) AS n
FROM nh
WHERE (SurveyYr = '2011_12')
GROUP BY Gender
```

为了实际将 dblyr 访问结果数据载入到 R 中，应该用 `collect()` 函数，如：

```
d.nh2_loc <- dv.nh |>
  select(ID, Gender, Age) |>
  filter(SurveyYr == "2011_12") |>
  collect()
```

许多简单的计数、总和等计算可以在数据库中完成，不需要下载数据。实际使用 R 进行各种计算时，就用 `collect()` 把数据下载下来。

44.5 SQL 命令简介

SQL 是关系数据库查询和管理的专用语言，关系数据库都支持 SQL 语言，但彼此之间可能有一些技术性的差别。

用 d.class 数据框演示，有 19 个学生的姓名、性别、年龄、身高、体重。

```
str(d.class)
```

```
## spc_tbl_ [19 x 5] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ name  : chr [1:19] "Alice" "Becka" "Gail" "Karen" ...
##  $ sex   : chr [1:19] "F" "F" "F" "F" ...
##  $ age   : num [1:19] 13 13 14 12 12 15 11 15 14 14 ...
##  $ height: num [1:19] 56.5 65.3 64.3 56.3 59.8 66.5 51.3 62.5 62.8 69 ...
##  $ weight: num [1:19] 84 98 90 77 84.5 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   name = col_character(),
##   ..   sex = col_character(),
##   ..   age = col_double(),
##   ..   height = col_double(),
##   ..   weight = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

下面建立一个 duckdb 数据库，将 d.class 保存在数据库中:

```
library(duckdb)
drv <- duckdb()
con <- dbConnect(drv)
dbWriteTable(con, name="class", value=d.class)
```

44.5.1 取行子集

取出所有行的命令：

```
dtmp <- dbGetQuery(
  conn=con,
  statement=paste(
    "SELECT *",
    "  FROM class"
  ))
dim(dtmp)
## [1] 19  5
```

为了取出满足条件的某些行，在 SQL 命令中加上 WHERE 子句：

```
dtmp <- dbGetQuery(
  conn=con,
  statement=paste(
    "SELECT *",
    "  FROM class",
    "  WHERE sex='F' AND age <= 12"  ))
dtmp
##    name sex age height weight
## 1 Karen   F  12   56.3   77.0
## 2 Kathy   F  12   59.8   84.5
## 3 Sandy   F  11   51.3   50.5
```

44.5.2 取行列子集

可以在 SELECT 子句中指定要取出的列，列名之间用逗号分隔，如：

```
dtmp <- dbGetQuery(
  conn=con,
  statement=paste(
    "SELECT name, age, height, weight",
    "  FROM class",
    "  WHERE sex='F' AND age <= 12"))
dtmp
##    name age height weight
## 1 Karen  12   56.3   77.0
```

```
## 2 Kathy  12   59.8   84.5
## 3 Sandy  11   51.3   50.5
```

在 WHERE 子句中可以使用如下的比较和逻辑运算：

- = ^= > < >= <= IN
- 或 EQ NE GT LT GE LE IN
- IS NULL 表示“是空值”，与缺失值类似
- BETWEEN a AND b, 表示属于闭区间 $[a, b]$
- AND 逻辑与，OR 逻辑或，NOT 逻辑非。

44.5.3 取某列的所有不同取值

在列名前加 DISTINCT 修饰，可以取出该列的所有不重复的值：

```
dtmp <- dbGetQuery(
  conn=con,
  statement=paste(
    "SELECT DISTINCT sex",
    "  FROM class"))
dtmp
##   sex
## 1   F
## 2   M
```

多个变量的不重复的组合：

```
dtmp <- dbGetQuery(
  conn=con,
  statement=paste(
    "SELECT DISTINCT sex, age",
    "  FROM class",
    "  WHERE age >= 15"))
dtmp
##   sex age
## 1   F  15
## 2   M  15
## 3   M  16
```

44.5.4 查询结果排序

用 ORDER BY 子句排序：

```
dtmp <- dbGetQuery(
  conn=con,
  statement=paste(
    "SELECT name, height",
    "  FROM class",
    "  WHERE name IN ('Alice', 'Becka', 'Gail')",
    "  ORDER BY height DESC"))
dtmp
##    name height
## 1 Becka   65.3
## 2  Gail   64.3
## 3 Alice   56.5
```

在变量名后面加后缀 DESC 表示降序。

44.5.5 查询时计算新变量

在 SELECT 中用“表达式 AS 变量名”的格式计算新变量。例如，class 表中的身高以英寸为单位，体重以磅为单位，分别转换为厘米和千克单位：

```
dtmp <- dbGetQuery(
  conn=con,
  statement=paste(
    "SELECT name, round(height*2.54) AS hcm,",
    "    round(weight*0.4535924) as wkg",
    "  FROM class"))
head(dtmp, 3)
##    name hcm wkg
## 1 Alice 144  38
## 2 Becka 166  44
## 3  Gail 163  41
```

44.5.6 分组汇总

用 GROUP BY 子句对观测分组，用 SUM, AVG 等统计函数计算汇总统计量。

统计函数有：

- COUNT(*) 表示行数；
- SUM(x) 求和；
- AVG(x) 求平均；
- MAX(x), MIN(x) 求最大值和最小值。

例如：

```
dtmp <- dbGetQuery(
  conn=con,
  statement=paste(
    "SELECT COUNT(*) AS n, AVG(height) AS avgh",
    "  FROM class"))
dtmp
##    n     avgh
## 1 19 62.33684
```

分组计算如：

```
dtmp <- dbGetQuery(
  conn=con,
  statement=paste(
    "SELECT sex, COUNT(*) AS n, AVG(height) AS avgh",
    "  FROM class",
    "  GROUP BY sex"))
dtmp
##   sex  n     avgh
## 1   F  9 60.58889
## 2   M 10 63.91000
```

在用了 GROUP BY 以后，对统计结果的筛选条件要写在 HAVING 子句中而不是用 WHERE 子句，如：

```
dtmp <- dbGetQuery(
  conn=con,
  statement=paste(
    "SELECT age, COUNT(*) AS n",
    "  FROM class",
    "  GROUP BY age",
    "  HAVING COUNT(*)=1"))
dtmp
##   age n
## 1  16 1
```

44.5.7 将查询结果保存为新表

可以将查询结果保存在数据库中，而不是返回到 R 中。这时，需要用 `dbExecute()` 函数，返回值是受到影响的行数：

```
for(tab in c("class1", "class2")){
  if(dbExistsTable(con, tab)){
    dbRemoveTable(con, tab)
  }
}

dbExecute(
  conn=con,
  statement=paste(
    "CREATE TABLE class1 AS",
    "  SELECT name, sex",
    "    FROM class"
  ))
## [1] 0

dbExecute(
  conn=con,
  statement=paste(
    "CREATE TABLE class2 AS",
    "  SELECT name, age",
    "    FROM class"
  ))
## [1] 0

dbListTables(con)
## [1] "class"  "class1" "class2"
```

在执行比较复杂的查询时，可以用这种方法生成一些中间结果，最后要删除这些作为中间结果的表。

44.5.8 从多个表查询

使用数据库的好处除了可以管理大型数据，还有许多其它好处，比如可以保证数据被修改时不会因断电、网络故障等出错，可以并发读取或修改，可以备份、恢复，等等。

关系数据库经常需要将有关的信息保存在多张表中，而不是使用一张大表，这与数据库的设计理念有关，可以减少数据冗余，增强数据的一致性。但是，在使用这些信息时，就需要从多张表进行查询，称为表的连接查询。

最常见的连接查询是所谓内连接（INNER JOIN），按照某一个或者某几个关键列将数据行对齐进行查询。最容易理解的一对一的查询，比如，设学生的姓名、性别保存在 class1 表中，姓名、年龄保存在 class2 表中，没有重名，则姓名可以作为关键列。如果要查询女生年龄大于大于 15 的人，就需要使用两张表：

```
dtmp <- dbGetQuery(
  conn=con,
  statement=paste(
    "SELECT a.name, sex, age",
    "  FROM class1 AS a, class2 AS b",
    "  WHERE a.name = b.name AND sex='F' AND age >= 15"))
dtmp
##     name sex age
## 1   Mary   F  15
## 2 Sharon   F  15
```

上面的程序中 WHERE 子句中的 `a.name = b.name` 就是内连接。在使用多张表时，在 `FROM` 子句的多张表之间用逗号分隔，一般在表名后用 `AS` 关键字引入一个临时的别名，对两张表中共同的变量名如 `name` 需要用 `a.name` 和 `b.name` 区分。连接两个表一般使用两列来匹配，

内连接也支持一对多的连接。比如，下面的表将 `F` 映射到女, `M` 映射到男，可以按 `sex` 连接：

```
dclass.sexm <- data.frame(
  sex = c("F", "F"),
  sexc = c(" 女", " 男"))
if(dbExistsTable(con, "sexm")){
  dbRemoveTable(con, "sexm")
}
dbWriteTable(con, name="sexm", value=dclass.sexm)

dtmp <- dbGetQuery(
  conn=con,
  statement=paste(
    "SELECT a.name, a.sex, sexc",
    "  FROM class1 AS a, sexm AS b",
    "  WHERE a.sex = b.sex AND name IN ('Alice', 'Alfred')"))
dtmp
##    name sex sexc
## 1 Alice   F   女
## 2 Alice   F   男
```

在内连接时如果关键列的值匹配后形成多对多的连接，将会做两两搭配组合。

内连接仅保留关键列能匹配的行。如果希望保留不能匹配的行，就要使用外连接，分为：

- 左外连接，保留左表的所有行，右表仅保留匹配的行；
- 右外连接，保留右表的所有行，左表仅保留匹配的行；
- 全外连接，保留所有匹配和不匹配的行。

左外连接的程序示例：

```
d1 <- data.frame(
  id = c("a", "b"),
  x = c(11, 12))
d2 <- data.frame(
  id = c("a", "c"),
  y = c(21, 22))
```

```
knitr::kable(d1)
```

id	x
a	11
b	12

```
knitr::kable(d2)
```

id	y
a	21
c	22

```
dbWriteTable(con, name="table1", value=d1)
dbWriteTable(con, name="table2", value=d2)

dtmp <- dbGetQuery(
  conn=con,
  statement=paste(
    "SELECT a.id, x, y",
    "  FROM table1 AS a LEFT JOIN table2 AS b",
    "  ON A.id=b.id"))
dtmp
## id  x  y
## 1  a 11 21
## 2  b 12 NA
```

右外连接用 RIGHT JOIN 关键字，全外连接用 FULL OUTER JOIN 关键字。其它的数据库一般是可以支持的。

44.6 用 dm 包管理表

数据库中常常用多个表来存储某一系统的数据，这些表通过键来连接。dm 包可以获取数据库中表的连接关系，使得从多个表访问信息变得容易；也可以将 R 中的多个数据框建立联系，然后上传到数据库中。

设表 A 和表 B 连接，按 A 中的 IDA 变量与 B 中的 IDB 变量匹配连接，则 IDA 列称为表 A 的“主键”（primary key），而 IDB 列称为表 A 关于参考表 B 的一个“外键”（foreign key）。有时键需要用多列的组合。

一个 dm 对象可以包含多个表，这些表的主键和外键一般在从数据库查询时就已经确定，如果不然，可以在程序中指定每个表的主键和外键。在用主键和外键规定了各个表的连接方式后，可以作图显示这些连接。

指定了连接关系后，可以自动化地产生连接后的大表用来做统计分析。

dm 支持 dplyr 方式的访问和处理。

44.7 访问 Oracle 数据库

注：这部分内容编写较早，有可能已经过时。

Oracle 是最著名的数据库服务器软件。要访问的数据库，可以是安装在本机上的，也可以是安装在网络上某个服务器中的。如果是远程访问，需要在本机安装 Oracle 的客户端软件。

假设已经在本机安装了 Oracle 服务器软件，并设置 orcl 为本机安装的 Oracle 数据库软件或客户端软件定义的本地或远程 Oracle 数据库的标识，test 和 oracle 是此数据库的用户名和密码，testtab 是此数据库中的一个表。

为了在 R 中访问 Oracle 数据库服务器中的数据库，在 R 中需要安装 ROracle 包。这是一个源代码扩展包，需要用户自己编译安装。在 MS Windows 环境下，需要安装 R 软件和 RTools 软件包（在 CRAN 网站的 Windows 版本软件下载栏目中）。在 MS Windows 命令行窗口，用如下命令编译 R 的 ROracle 扩展包：

```
set OCI_LIB32=D:\oracle\product\10.2.0\db_1\bin
set OCI_INC=D:\oracle\product\10.2.0\db_1\oci\include
set PATH=D:\oracle\product\10.2.0\db_1\bin;C:\Rtools\bin;C:\Rtools\gcc-4.6.3\bin;"%PATH%"
C:\R\R-3.2.0\bin\i386\rcmd INSTALL ROracle_1.2-1.tar.gz
```

其中的前三个 set 命令设置了 Oracle 数据库程序或客户端程序链接库、头文件和可执行程序的位置，第三个 set 命令还设置了 RTools 编译器的路径。这些路径需要根据实际情况修改。这里的设置是在本机运行的 Oracle 10g 服务器软件的情况。最后一个命令编译 ROracle 扩展包，相应的 rcmd 程序路径需要改成自己的安装路径。

如果服务器在远程服务器上，设远程服务器的数据库标识名为 ORCL，本机需要安装客户端 Oracle instant client 软件，此客户端软件需要与服务器同版本号，如 `instantclient-basic-win32-10.2.0.5.zip`, 这个软件不需要安装，只需要解压到一个目录如 `C:\instantclient_10_2` 中。在本机（以 MS Windows 操作系统为例）中，双击系统，选择高级–环境变量，增加如下三个环境变量：

```
NLS_LANG = SIMPLIFIED CHINESE_CHINA.ZHS16GBK
```

```
ORACLE_HOME = C:\instantclient_10_2
TNS_ADMIN = C:\instantclient_10_2
```

并在环境变量 PATH 的值的末尾增加 Oracle 客户端软件所在的目录 `verb|C:\instantclient_10_2`, 并与前面内容用分号分开。

然后，在 client 所在的目录 `C:\instantclient_10_2` 中增加如下内容的 tnsnames.ora‘文件

```
  orcl =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.102 )
     (PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = orcl)
    )
  )
```

其中 HOST 的值是安装 Oracle 服务器的服务器的 IP 地址，orcl 是一个服务器实例名，能够在服务器端的 tnsnames.ora 文件中查到，等号前面的 orcl 是对数据库给出的客户端别名，这里就干脆用了和服务器端的数据库标识名相同的名字 orcl。

不论是在本机的数据框服务器还是在本机安装设置好客户端后，在 R 中用如下的程序可以读入数据库中的表：

```
libraryROracle)
drv <- dbDriver("Oracle")

conn <- dbConnect(drv, username="test",
                  password="oracle",  dbname="orcl")

rs <- dbSendQuery(conn, "select * from testtab")
d <- fetch(rs)
```

可以用 `dbGetTable()` 取出一个表并存入 R 数据框中。用 `dbSendQuery()` 发出一个 SQL 命令，用 `fetch()` 可以一次性取回或者分批取回，在表行数很多时这种方法更适用。

44.8 MySQL 数据库访问

MySQL 是高效、免费的数据库服务器软件，在很多行业尤其是互联网行业占有很大的市场。为了在 R 中访问 MySQL 数据库，只要安装 RMySQL 扩展包（有二进制版本）。现在性能更好的一个连接 MySQL 的扩展包是 RMariaDB。

假设服务器地址在 192.168.1.111, 可访问的数据库名为 world, 用户为 test, 密码为 mysql。设 world 库中有表 country。

在 R 中要访问 MySQL 数据框，首先要建立与数据库服务器的连接:

```
library(RMySQL)
con <- dbConnect(RMySQL::MySQL(),
    dbname="world",
    username="test", password="mysql",
    host="192.168.1.111")
```

下列代码列出 world 库中的所有表，然后列出其中的 country 表的所有变量:

```
dbListTables(con)
dbListFields(con, "country")
```

下列代码取出 country 表并存入 R 数据框 d.country 中:

```
d.country <- dbReadTable(con, "country")
```

下列代码把 R 中的示例数据框 USArrests 写入 MySQL 库 world 的表 arrests 中:

```
data(USArrests)
dbWriteTable(con, "arrests", USArrests,
    overwrite=TRUE)
```

当然，这需要用户对该库有写权限。

可以用 `dbGetQuery()` 执行一个 SQL 查询并返回结果，如

```
dbGetQuery(con, "select count(*) from arrests")
```

当表很大时，可以用 `dbSendQuery()` 发送一个 SQL 命令，返回一个查询结果指针对象，用 `dbFetch()` 从指针对象位置读取指定行数，用 `dbHasCompleted()` 判断是否已读取结束。如

```
res <- dbSendQuery(con, "SELECT * FROM country")
while(!dbHasCompleted(res)){
  chunk <- dbFetch(res, n = 5)
  print(chunk[,1:2])
}
dbClearResult(res)
```

数据库使用完毕时，需要关闭用 `dbConnect()` 打开的连接:

```
dbDisconnect(con)
```

44.9 利用 RODBC 访问 Access 数据库

扩展包 RODBC 在 MS Windows 操作系统中可以访问 Excel、Access、dBase、FoxPro 等微机数据库软件的数据库，也可以在安装了需要的 ODBC 客户端后访问 Oracle 等数据库。用 odbc 包访问远程数据库速度更快，而且遵从 DBI 的接口规范，RODBC 不使用 DBI 接口规范。

假设有 Access 数据库在文件 `c:/Friends/birthdays.mdb` 中，内有两个表 Men 和 Women，每个表包含域 Year, Month, Day, First Name, Last Name, Death。域名应尽量避免用空格。

下面的程序把女性记录的表读入为 R 数据框:

```
library(RODBC)
con <- odbcConnectAccess("c:/Friends/birthdays.mdb")
women <- sqlFetch(con, sqtable="Women")
close(con)
```

RODBC 还有许多与数据库访问有关的函数，比如，`sqlQuery()` 函数可以向打开的数据库提交任意符合标准的 SQL 查询，`sqlSave()` 可以将 R 数据框保存到数据库中。

44.10 arrow 包

parquet 格式是一种大型数据高效存储、交换的格式。Apache 的 arrow 包可以高效地访问和分析大型数据，包括 parquet 格式数据，在 R 中可以用 arrow 扩展包调用 Apache arrow 的功能，实现快速的数据存储、交换，并可以实现对超过内存容量的大型数据的分析能力。arrow 扩展包提供了 dplyr 形式的工作方式。

```
library(tidyverse)
library(arrow)
```

```
library(NHANES)
data(NHANES, package="NHANES")
```

将 NHANES 数据集保存为 parquet 格式：

```
write_parquet(NHANES, "NHANES.parquet")
```

将 parquet 格式文件读入为一个 Dataset 类型，这样的类型不会将数据全部载入内存，而是在需要时从外部存储访问：

```
pq.nh <- read_parquet("NHANES.parquet")
```

随后可以用 dplyr 访问，如：

```
pq.nh |>
  glimpse()
```

```
Rows: 10,000
Columns: 76
$ ID               <int> 51624, 51624, 51624, 51625,…
$ SurveyYr         <fct> 2009_10, 2009_10, 2009_10, …
$ Gender           <fct> male, male, male, male, fem…
$ Age              <int> 34, 34, 34, 4, 49, 9, 8, 45…
$ AgeDecade        <fct>  30-39,  30-39,  30-39,  0-…
.........
```

```
pq.nh |>
  group_by(Gender) |>
  summarize(
    max_age = max(Age),
    median_age = median(Age)  )
```

```
# A tibble: 2 × 3
  Gender max_age median_age
  <fct>    <int>      <dbl>
1 female      80         37
2 male        80         36
```

当数据很大时，仅记录要进行的分析和变换，需要增加 `collect()` 步才真正执行并将结果转换为 R 的 tibble。

将 Dataset 格式转换为 CSV 格式如：

```
write_dataset(
  pq.nh, "./",
  basename_template="NHANES{i}.csv",
  format="csv")
```

结果将在当前目录生成单个 `NHANES0.csv` 文件。因为 parquet 格式主要用于大型数据，所以通常用一个或多个分组变量将所有数据分组，然后每组保存一个文件，保存在分组变量形成的目录层次中。在使用这样的大型数据时，经常只需要使用其中的子集，而分多个文件存储，就可以仅访问涉及到的文件，而且访问多个规模较小的文件也比较容易。parquet 格式是按列存储的，所以比较适用于统计分析。

读入单个 CSV 文件如：

```
pq.nh2 <- open_dataset(
  "NHANES0.csv",
  partition = NULL,
  format = "csv",
  convert_options = CsvConvertOptions$create(
    col_types = schema(
      Race3 = string(),
      BMICatUnder20yrs = string(),
      Testosterone = float64(),
      TVHrsDay = string(),
      CompHrsDay = string())))
pq.nh2 |>
  glimpse()
```

```
FileSystemDataset with 1 csv file
10,000 rows x 76 columns
$ ID                <int64> 51624, 51624, 51624, 51625…
$ SurveyYr         <string> "2009_10", "2009_10", "200…
$ Gender           <string> "male", "male", "male", "m…
$ Age               <int64> 34, 34, 34, 4, 49, 9, 8, 4…
$ AgeDecade        <string> " 30-39", " 30-39", " 30-3…
$ AgeMonths         <int64> 409, 409, 409, 49, 596, 11…
$ Race1            <string> "White", "White", "White",…
$ Race3            <string> "", "", "", "", "", "", ""…
........
```

用 `open_dataset` 读入一个 CSV 文件，并不会将所有观测读入内存，而是保留在外部存储（硬盘）上，先读入几千行用来判断每列的类型。仅当需要时才从外部存储访问数据，这使得对超过内存大小的数据也可以进行分析。这种自动判断列的类型对因子类型可能会出错，所以可以用上面的方法根据需要指定某些列的类型。没有发现读入更多行以更好地判断列类型的选项。

44.11 层次数据

数据框是统计分析最常见、最实用的数据格式，实际当中还有许多数据是层次的（树状的、嵌套的），比如 Quarto 的 YAML 设置。

用来表示层次数据的格式也有很多，比如，JSON，XML，YAML 等。

在 R 中可以用列表表示层次数据，因为列表的元素也可以是列表，这样可以逐层嵌套。在 R 中用 jsonlite 包访问 JSON 格式的数据，可以用 repurrrsive 包辅助处理层次格式的数据。可以用 purrr 包的 `map` 类函数对保存为嵌套列表的层次数据进行处理。可以用 yaml 包读写 YAML 数据。

需要时，应能够将层次数据转换为整洁格式的数据框。

```
library(tidyverse)
library(jsonlite)
```

```
##
## Attaching package: 'jsonlite'

## The following object is masked from 'package:purrr':
##
##     flatten
```

```
library(repurrrsive)
```

```
## Warning: package 'repurrrsive' was built under R version 4.2.2
```

44.11.1 嵌套列表

某个 Quarto 文件的 YAML 元数据如下：

```
title: " 期末报告"
author: " 张三"
output:
  html:
    toc: true
    number-sections: true
  pdf:
    number-sections: true
```

这就是嵌套数据，可以用嵌套列表表示为：

```
lis_qmd <- list(
  "title" = " 期末报告",
  "author" = " 张三",
  "output" = list(
    "html" = list(
      "toc" = TRUE,
```

```
      "number-sections" = TRUE
    ),
    "pdf" = list(
      "number-sections" = TRUE
    )
  )
)
```

列表显示比较冗长，可以用 `utils::str()` 获得较简单的显示：

```
str(lis_qmd)
```

```
## List of 3
##  $ title : chr "期末报告"
##  $ author: chr "张三"
##  $ output:List of 2
##   ..$ html:List of 2
##   .. ..$ toc            : logi TRUE
##   .. ..$ number-sections: logi TRUE
##   ..$ pdf :List of 1
##   .. ..$ number-sections: logi TRUE
```

在 RStudio 中，可以在 “Environment” 窗格点击查看嵌套列表内容，或者用 “`View(变量名)`” 的方法查看。

44.11.2 tibble 中的列表

列表可以保存在 tibble 的列中，参见 §25.4。

多个类似的内容可以作为 tibble 表的多行保存，每行都可以是嵌套数据。例如：

```
tib_qmd <- tibble(
  id = c(1,2,3),
  title = c(" 作业", " 期中报告", " 期末报告"),
  output = list(
    list(
      "html" = list("toc" = TRUE)
    ),
    list(
      "html" = list("toc" = TRUE),
      "pdf" = list("number-sections" = TRUE)
```

```
    ),
    list(
      "html" = list("toc" = TRUE),
      "pdf" = list(
        "toc" = TRUE,
        "number-sections" = TRUE)
    )
  )
)
knitr::kable(tib_qmd)
```

id	title	output
1	作业	TRUE
2	期中报告	TRUE, TRUE
3	期末报告	TRUE, TRUE, TRUE

可以用 `tidyr::unnest_wider()` 函数将列表列展开为多列，如：

```
tib_qmd |>
  unnest_wider(output) |>
  unnest_wider(col = c(html, pdf), names_sep="_")
```

```
## # A tibble: 3 x 5
##      id title    html_toc `pdf_number-sections` pdf_toc
##   <dbl> <chr>    <lgl>    <lgl>                 <lgl>
## 1     1 作业     TRUE     NA                    NA
## 2     2 期中报告 TRUE     TRUE                  NA
## 3     3 期末报告 TRUE     TRUE                  TRUE
```

可以用 `unnest_longer()` 将列表列展开并堆叠起来。

可以用 `hoist()` 函数从列表列中仅提取需要的元素并保存为新列。

44.11.3 展开示例

44.11.3.1 github 数据

repurrrsive 包的 `gh_repos` 是嵌套列表，有 6 个列表元素，每个列表元素代表一个开发者，而每个开发者的数据又是列表，其列表元素至多有 30 个，每个元素代表一个 github 项目，这些项目也是列表但有元素名，格式统一。

如下的程序先将数据包装到 tible 列表列中，然后用 `unnest_longer()` 将所有开发者的所有项目堆叠在一起，形成了 170 行，每行为一个项目，然后用 `unnest_wider()` 将每个项目内容从有名列表转换为数据框的多列。如：

```
tibble(json = gh_repos) |> # 将 6 位开发者数据转换为列表列的 6 行
  unnest_longer(json) |>   # 将每位开发者的项目放到不同行
  unnest_wider(json) |>    # 提取项目数据到不同列
  select(id, full_name, owner, description)
```

```
## # A tibble: 176 x 4
##          id full_name                owner              description
##       <int> <chr>                    <list>             <chr>
##  1 61160198 gaborcsardi/after        <named list [17]> Run Code in the Background
##  2 40500181 gaborcsardi/argufy       <named list [17]> Declarative function argu~
##  3 36442442 gaborcsardi/ask          <named list [17]> Friendly CLI interaction ~
##  4 34924886 gaborcsardi/baseimports  <named list [17]> Do we get warnings for un~
##  5 61620661 gaborcsardi/citest       <named list [17]> Test R package and repo f~
##  6 33907457 gaborcsardi/clisymbols   <named list [17]> Unicode symbols for CLI a~
##  7 37236467 gaborcsardi/cmaker       <named list [17]> port of cmake to r
##  8 67959624 gaborcsardi/cmark        <named list [17]> CommonMark parsing and re~
##  9 63152619 gaborcsardi/conditions   <named list [17]> <NA>
## 10 24343686 gaborcsardi/crayon       <named list [17]> R package for colored ter~
## # ... with 166 more rows
```

结果中的 owner 还是嵌套列表，可以再多执行一轮 |> unnest_wider(owner, names_sep="_")。

44.11.3.2 角色数据

repurrrsive 包的 got_chars 变量是一个列表，有 30 个元素，每个元素代表了冰火之歌中的一个视角角色信息，是有名列表，本身就是比较像数据框结构的。

```
tibble(json = got_chars) |>  # 变成有 30 行的列表列
  unnest_wider(json) |>      # 每行展开成多列
  select(id, name, gender, titles)
```

```
## # A tibble: 30 x 4
##       id name               gender titles
##    <int> <chr>              <chr>  <list>
##  1  1022 Theon Greyjoy      Male   <chr [2]>
##  2  1052 Tyrion Lannister   Male   <chr [2]>
##  3  1074 Victarion Greyjoy  Male   <chr [2]>
##  4  1109 Will               Male   <chr [1]>
##  5  1166 Areo Hotah         Male   <chr [1]>
```

```
## 6 1267 Chett              Male   <chr [1]>
## 7 1295 Cressen            Male   <chr [1]>
## 8  130 Arianne Martell    Female <chr [1]>
## 9 1303 Daenerys Targaryen Female <chr [5]>
## 10 1319 Davos Seaworth     Male   <chr [4]>
## # ... with 20 more rows
```

还有许多列是列表列。将其中的 titles 堆叠展开：

```
tibble(json = got_chars) |>
  unnest_wider(json) |>
  select(id, name, gender, titles) |>
  unnest_longer(titles)
```

```
## # A tibble: 59 x 4
##       id name              gender titles
##    <int> <chr>             <chr>  <chr>
##  1 1022 Theon Greyjoy     Male   "Prince of Winterfell"
##  2 1022 Theon Greyjoy     Male   "Lord of the Iron Islands (by law of the gree~
##  3 1052 Tyrion Lannister  Male   "Acting Hand of the King (former)"
##  4 1052 Tyrion Lannister  Male   "Master of Coin (former)"
##  5 1074 Victarion Greyjoy Male   "Lord Captain of the Iron Fleet"
##  6 1074 Victarion Greyjoy Male   "Master of the Iron Victory"
##  7 1109 Will              Male   ""
##  8 1166 Areo Hotah        Male   "Captain of the Guard at Sunspear"
##  9 1267 Chett             Male   ""
## 10 1295 Cressen           Male   "Maester"
## # ... with 49 more rows
```

44.11.4 JSON 格式

JSON(Javascript object notation) 是许多网络接口的默认数据数据格式。其基本数据类型与 R 相近但不完全相同。

JSON 基本数据类型有：

- `null`，类似 R 的缺失值；
- `string`，字符串，必须使用双撇号而不能使用单撇号；
- `number`，包括 R 的整型与双精度型；
- `boolean`，可取 `true`，`false` 值。

JSON 中的标量就是标量，不像 R 那样将标量看成是长度为 1 的向量。为了在 JSON 中保存向量，可以用 `array` 数据格式，将内容写在方括号内，如 `["red", "blue", "yellow"]`，对应于 R 的列表。还可以用 `object` 数据格式，这类似 R 的列表，写在大括号内，元素名与元素值用冒号分隔，如 `{"x": 1, "y": 2}`，或 `{"Name": "John", "age": 32}`。

jsonlite 包的 `read_json()` 函数将一个 JSON 格式的文本文件读入到 R 中，变成嵌套列表格式。而函数 `parse_json()` 则直接输入一个 JSON 格式字符串，将其转换为 R 格式，如：

```
str(parse_json('1'))
```

```
##  int 1
```

```
str(parse_json('[1,2,3]'))
```

```
## List of 3
##  $ : int 1
##  $ : int 2
##  $ : int 3
```

```
str(parse_json('{"x": 1, "y": 2}'))
```

```
## List of 2
##  $ x: int 1
##  $ y: int 2
```

```
str(parse_json('
  {"Name": "John",
  "age": 32,
  "scores": [90, 85, 88]}'))
```

```
## List of 3
##  $ Name  : chr "John"
##  $ age   : int 32
##  $ scores:List of 3
##   ..$ : int 90
##   ..$ : int 85
##   ..$ : int 88
```

jsonlite 的 `fromJSON` 在从 JSON 字符串转换时可以进行一些简化，但最好还是显式地用 `unnest_wider()`、`unnest_longer()` 等转换。如：

```
str(fromJSON('1'))
```

```
##  int 1
```

```
str(fromJSON('[1,2,3]'))
```

```
##  int [1:3] 1 2 3
```

```
str(fromJSON('{"x": 1, "y": 2}'))
```

```
## List of 2
##  $ x: int 1
##  $ y: int 2
```

```
str(fromJSON('
  {"Name": "John",
  "age": 32,
  "scores": [90, 85, 88]}'))
```

```
## List of 3
##  $ Name  : chr "John"
##  $ age   : int 32
##  $ scores: int [1:3] 90 85 88
```

将前面的 `lis_qmd` 用 JSON 字符串表示，可写成：

```
json_qmd <- '{
  "title": " 期末报告",
  "author": " 张三",
  "output": {
    "html": {
      "toc": true,
      "number-sections": true
      },
    "pdf": {
      "number-sections": true
      }
    }
  }
'
```

转换：

```
json_qmd |>
  parse_json() |>
  str()
```

```
## List of 3
##  $ title : chr "期末报告"
##  $ author: chr "张三"
##  $ output:List of 2
##   ..$ html:List of 2
##   .. ..$ toc            : logi TRUE
##   .. ..$ number-sections: logi TRUE
##   ..$ pdf :List of 1
##   .. ..$ number-sections: logi TRUE
```

对应于前面 `tib_qmd` 的数据，用 JSON 格式可以写成：

```
json_qmd2 <- '[
  {"id": 1, "title": " 作业",
  "html": {"toc": true},
  "pdf": {"number-sections": true}},
  {"id": 2, "title": " 期中报告",
  "html": {"toc": true},
  "pdf": {"number-sections": true}},
  {"id": 2, "title": " 期中报告",
  "html": {"toc": true},
  "pdf": {"toc": true, "number-sections": true}}
]
'
```

转换为 R 的嵌套列表，然后再转换为 tibble：

```
json_qmd2 |>
  parse_json() |>
  tibble(json = _) |>
  unnest_wider(json) |>
  unnest_wider(c(html, pdf), names_sep="_")
```

```
## # A tibble: 3 x 5
```

```
##      id title    html_toc `pdf_number-sections` pdf_toc
##   <int> <chr>    <lgl>    <lgl>                 <lgl>
## 1     1 作业     TRUE     TRUE                  NA
## 2     2 期中报告 TRUE     TRUE                  NA
## 3     2 期中报告 TRUE     TRUE                  TRUE
```

JSON 格式没有单独的日期和日期时间格式，会保存成字符串，可以随后用 `readr::parse_date()` 和 `readr::parse_datetime()` 转换。有些数值会保存成字符串，可以随后用 `readr::parse_double()` 转换。

Part IX

用 Rcpp 访问 C++ 代码

Chapter 45

Rcpp 介绍

为了提高 R 程序的运行效率，可以尽量使用向量化编程，减少循环，尽量使用内建函数。对于效率的瓶颈，尤其是设计迭代算法时，可以采用编译代码，而 Rcpp 扩展包可以很容易地将 C++ 代码连接到 R 程序中，并且支持在 C++ 中使用类似于 R 的数据类型。

没有学过 C++ 语言的读者，如果需要编写比较独立的不太依赖于 R 的已有功能的算法，可以考虑学习使用 Julia 语言编写。Julia 语言是最近几年才发明的一种比 R 更现代、理念更先进的程序语言，其运行效率一般比 R 高得多，经常接近编译代码的效率。

Rcpp 可以很容易地把 C++ 代码与 R 程序连接在一起，可以从 R 中直接调用 C++ 代码而不需要用户关心那些繁琐的编译、链接、接口问题。可以在 R 数据类型和 C++ 数据类型之间容易地转换。

因为涉及到编译，所以 Rcpp 比一般的扩展包有更多的安装要求：除了要安装 Rcpp 包之外，MS Windows 用户还需要安装 RTools 包，这是用于 C, C++, Fortran 程序编译链接的开发工具包，是自由软件。用户的应用程序路径 (PATH) 中必须有 RTools 包可执行程序的路径 (安装 RTools 可以自动设置)。如果 Rcpp 不能找到编译器，可以把编译器安装到 Rcpp 默认的位置。Mac 系统的用户可以从应用商店安装 Xcode 软件，Linux 操作系统可以在操作系统命令行用如下命令安装编译软件：

```
sudo apt-get install r-base-dev
```

在 MS Windows 系统中，从CRAN网站下载 RTools 工具包，并将其安装到默认位置 `C:\RTools` 中，否则 RStudio 中使用 Rcpp 可能会出错。注意，RStudio 自己的自动安装 RTools 的功能可能有问题，安装后不能正常编译含有 Rcpp 的 Rmd 文件。按照 RTools 4.0 的要求，还要在自己的 “我的文档” 文件夹中生成名为.Renviron 的如下文本文件：

```
PATH="${RTOOLS40_HOME}\usr\bin;${PATH}"
```

Rcpp 支持把 C++ 代码写在 R 源程序文件内，执行时自动编译连接调用；也支持把 C++ 代码保存在单独的源文件中，执行 R 程序时自动编译连接调用；对较复杂的问题，应制作 R 扩展包，利用构建 R 扩展包的方法实现 C++ 代码的编译连接，这时接口部分也可以借助 Rcpp 属性功能或模块功能完成。

45.1 Rcpp 的用途

- 把已经用 R 代码完成的程序中运行速度瓶颈部分改写成 C++ 代码，提高运行效率。
- 对于 C++ 或 C 程序源代码或二进制代码提供的函数库，可以用 Rcpp 编写 C++ 界面程序进行 R 与 C++ 程序的输入、输出的传送，并在 C++ 界面程序中调用外来的函数库。
- 注意，用 Rcpp 编写 C++ 程序，不利于把程序脱离 R 运行或被其他的 C++ 程序调用。当然，可以只把 Rcpp 作为界面，主要的算法引擎完全不用 Rpp 的数据类型。
- RInside 扩展包支持把 R 嵌入到 C++ 主程序中。

45.2 Rcpp 入门样例

45.2.1 用 evalCpp() 转换单一计算表达式

如下的程序可以用来检测 Rcpp 以及所需要的编译器环境是否正确安装设置：

```
library(Rcpp)
evalCpp("1 + 2")
## [1] 3
```

evalCpp() 函数输入包含一个 C++ 表达式的字符串，自动进行编译、连接、接口生成、在 R 中运行。

45.2.2 用 cppFunction() 转换简单的 C++ 函数—Fibnacci 例子

考虑用 C++ 程序计算 Fibonacci 数的问题。Fibonacci 数满足 $f_0 = 0, f_1 = 1, f_t = f_{t-1} + f_{t-2}$。

可以使用如下 R 代码，其中有一部分 C++ 代码，用 cppFunction 转换成了 R 可以调用的同名 R 函数。

```
library(Rcpp)
cppFunction(code='
  int fibonacci(const int x){
    if(x < 2) return x;
    if(x > 40) return -1;
    // 太大的 x 值会占用过多计算资源
    return ( fibonacci(x-1) + fibonacci(x-2) );
  }
')
print(fibonacci(5))
## [1] 5
```

编译、链接、导入是在后台由 Rcpp 控制自动进行的，不需要用户去设置编译环境，也不需要用户执行编译、链接、导入 R 的工作。在没有修改 C++ 程序时, 同一 R 会话期间重新运行不必重新编译。

上面的 Fibonacci 函数仅接受标量数值作为输入，不允许向量输入。这个例子也说明 RCpp 变异的 C++ 函数允许输入标量 (int, double, bool)，输出标量 (int, double, bool)。

从算法角度评价，这个计算 Fibonacci 序列的算法是极其低效的，其算法规模是 $O(2^n)$，n 是自变量值。

45.2.3 用 sourceCpp() 转换 C++ 程序—正负交替迭代例子

设 $x_t, t = 1, 2, \ldots, n$ 保存在 R 向量 x 中，令

$$\begin{aligned} y_1 =& x_1 \\ y_t =& (-1)^t y_{t-1} + x_t,\ t = 2, \ldots, n \end{aligned}$$

希望用 C++ 函数对输入序列 x 计算输出 y，并用 R 调用这样的函数。

下面的程序用 R 函数 `sourceCpp()` 把保存在 R 字符串中的 C++ 代码编译并转换为同名的 R 函数。

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
  NumericVector iters(NumericVector x){
    int n = x.size();
    NumericVector y(n);

    y[0] = x[0];
    double sign=-1;
    for(int t=1; t<n; t++){
      sign *= -1;
      y[t] = sign*y[t-1] + x[t];
    }

    return y;
  }
')
print(iters(1:5))
## [1] 1 3 0 4 1
```

这个例子说明 C++ 程序可以直接写在 R 程序文件内 (保存为 R 多行字符串)，用 `sourceCpp()` 函数编译。

Rcpp 包为 C++ 提供了一个 `NumericVector` 数据类型，用来存储数值型向量。用成员函数 `size()` 访问其大小，用方括号下标访问其元素，但按照 C 和 C++ 的惯例，下标从 0 开始计数。

C 程序中定义的函数可以返回 `NumericVector` 数据类型，将自动转换为 R 的数值型向量。

特殊的注释`//[[Rcpp::export]]` 用来指定哪些 C++ 函数是要转换为 R 函数的。这叫做 Rcpp 属性（attributes）功能。

45.2.4 用 sourceCpp() 转换 C++ 源文件中的程序—正负交替迭代例子

直接把 C++ 代码写在 R 源程序内部的好处是不用管理多个源文件，缺点是当 C++ 代码较长时，不能利用专用 C++ 编辑环境和调试环境, 出错时显示的错误行号不好定位，而且把代码保存在 R 字符串内，C++ 代码中用到字符时需要特殊语法。所以，稍复杂的 C++ 代码应该放在单独的 C++ 源文件内。

假设上面的 iters 函数的 C++ 代码单独存入了一个`iters.cpp`源文件中，内容如下：

```
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericVector iters(NumericVector x){
  int n = x.size();
  NumericVector y(n);

  y[0] = x[0];
  double sign=-1;
  for(int t=1; t<n; t++){
    sign *= -1;
    y[t] = sign*y[t-1] + x[t];
  }

  return y;
}
```

用如下的 `sourceCpp()` 函数把 C++ 源文件中代码编译并转换为 R 可访问的同名函数，测试调用:

```
sourceCpp(file='iters.cpp')
print(iters(1:5))
## [1] 1 3 0 4 1
```

45.2.5 用 sourceCpp() 转换 C++ 源程序文件—卷积例子

考虑向量 $x=(x_0,x_1,\dots,x_{n-1})$, $y=(y_0,y_1,\dots,y_{m-1})$, 定义 x 与 y 的离散卷积 $z=(z_0,z_1,\dots,z_{n+m-2})$ 为

$$z_k=\sum_{(i,j):\,i+j=k} x_iy_j=\sum_{i=\max(0,k-m+1)}^{\min(k,n)} x_iy_{k-i}.$$

假设如下的 C++ 程序保存到了源文件 conv.cpp 中。

```
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericVector convolveCpp(
    const NumericVector a,
    const NumericVector b){
  int na = a.size(), nb = b.size();
  int nab = na + nb - 1;
  NumericVector xab(nab);

  for(int i=0; i < na; i++)
    for(int j=0; j < nb; j++)
      xab[i+j] += a[i] * b[j];

  return xab;
}
```

如下的代码用 sourceCpp() 函数把上面的 C++ 源文件 conv.cpp 自动编译并转换为同名的 R 函数，进行测试:

```
sourceCpp(file='conv.cpp')
print(convolveCpp(1:5, 1:3))
## [1]  1  4 10 16 22 22 15
```

45.2.6 随机数例子

Rcpp 的属性 (attributes) 功能使得在 C++ 中可以访问 R 的随机数生成功能，并且可以自动维护随机数种子状态。例如，生成一个标准正态随机数的例子：

```
set.seed(101)
evalCpp('
R::rnorm(0,1)
```

```
')
## [1] -0.3260365
rnorm(1,0,1)
## [1] 0.5524619
```

```
set.seed(101)
rnorm(2,0,1)
## [1] -0.3260365  0.5524619
```

可以看出 C++ 中 `R::rnorm()` 生成了一个正态随机数并且更新了 R 中的随机数种子状态。

在 C++ 中为了生成多个正态随机数，应该使用 `Rcpp::rnorm(n, mean, sd)`。如：

```
set.seed(101)
evalCpp('
Rcpp::rnorm(2, 0, 1)
')
## [1] -0.3260365  0.5524619
```

对标准正态随机数，也可以简写成 `Rcpp::rnorm(n)`。

45.2.7 bootstrap 例子

考虑单总体 X 的样本 $x_1, \dots, x_n$ 用 bootstrap 方法近似 $\bar{X}$ 和 S(样本标准差) 抽样分布的问题。bootstrap 方法从 $x_1, \dots, x_n$ 中有放回抽取 n 个值 $x_1^{(1)}, \dots, x_n^{(1)}$，从中计算 $\bar{x}^{(1)}$ 和 $S^{(1)}$，并重复 B 次（如 B=200），得到 $\bar{x}^{(j)}$, $j = 1, \dots, n$，作为 $\bar{x}$ 的抽样分布的近似样本，称为 bootstrap 样本；得到 $S^{(j)}$, $j = 1, \dots, n$，作为 S 的抽样分布的近似样本。

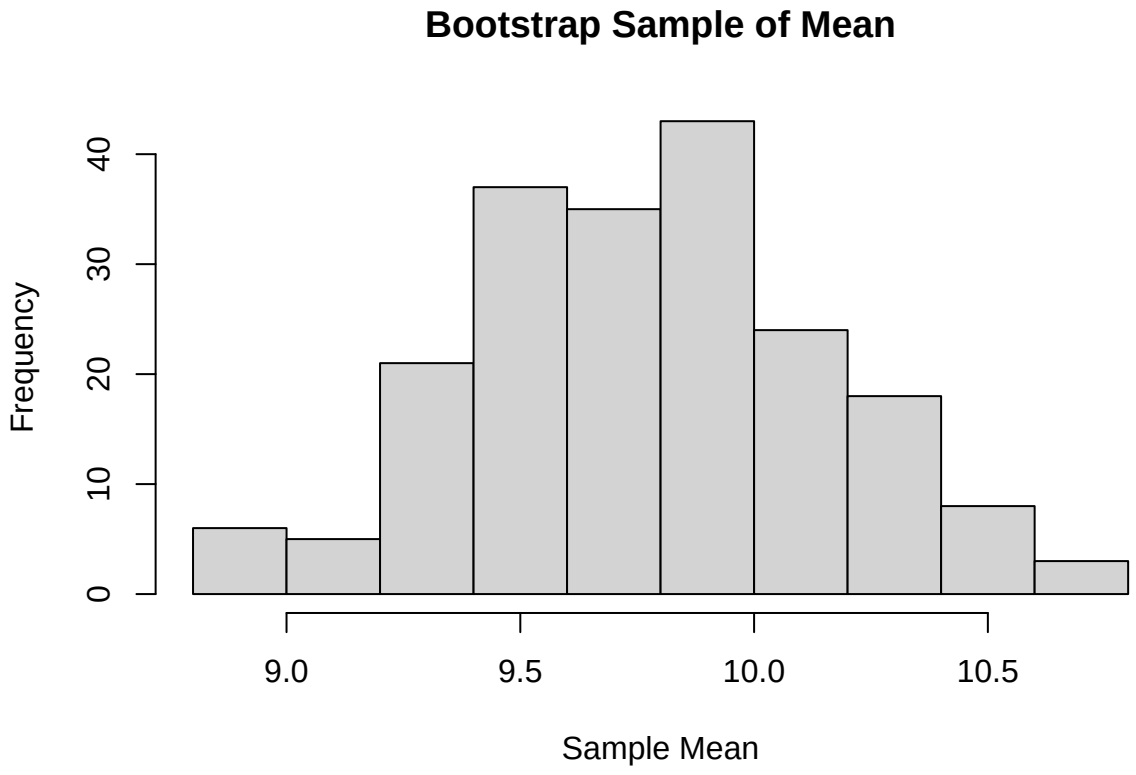
R 的 boot 包可以执行这样的计算，见 §40.4。如果直接用 R 程序编程计算，也比较容易，如：

```
boot_mean_sd <- function(x, B=200){
  out_stat <- matrix(0, B, 2)
  n <- length(x)
  for(j in 1:B){
    x_boot <- x[sample(1:n, size=n, replace=TRUE)]
    out_stat[j,1] <- mean(x_boot)
    out_stat[j,2] <- sd(x_boot)
  }

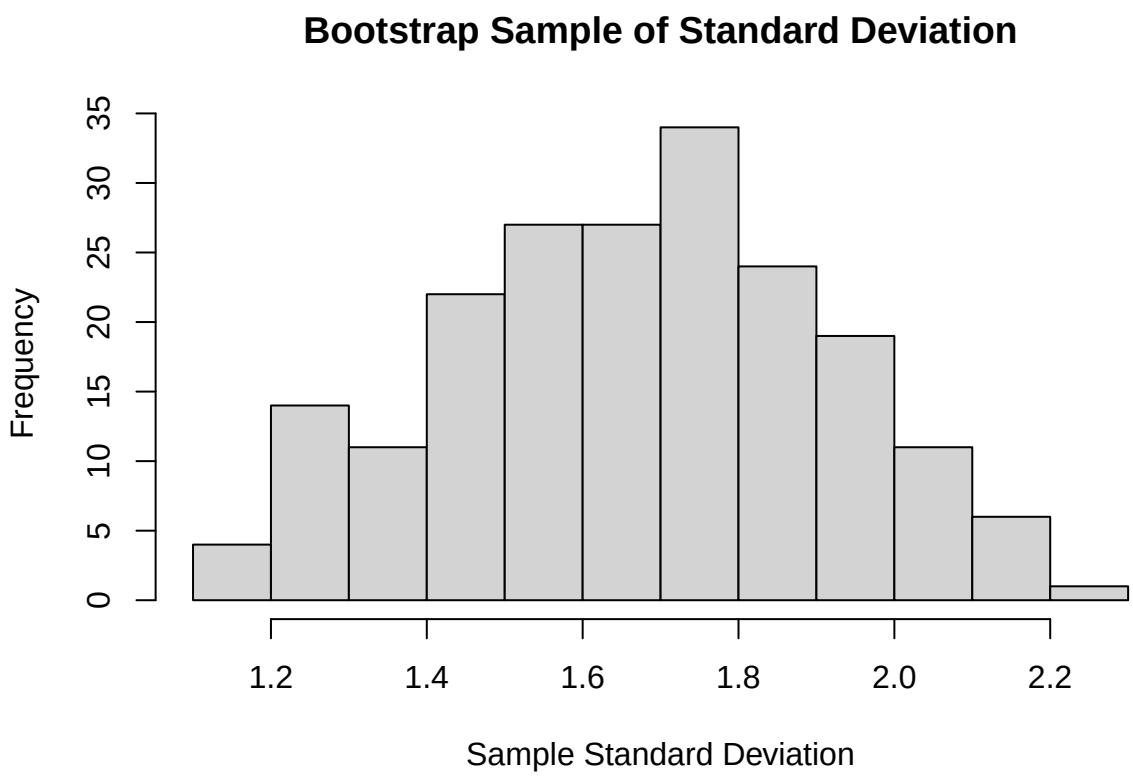
  out_stat
}
```

测试运行：

```
set.seed(101)
xsamp <- rnorm(20, mean=10, sd=2)
set.seed(111)
stats <- boot_mean_sd(xsamp)
hist(stats[,1],
  main="Bootstrap Sample of Mean",
  xlab = "Sample Mean")
```




```
hist(stats[,2],
  main="Bootstrap Sample of Standard Deviation",
  xlab = "Sample Standard Deviation")
```



注意受到原始样本的限制，bootstrap 样本的分布，与理论抽样分布还是有偏离的。只有在对多组实际样本进行 bootstrap 抽样的意义下，bootstrap 方法近似的统计量抽样分布才是近似于理论抽样分布的。

改用 C++ 实现，设如下 C++ 源程序保存在源文件 `boot-mean-sd.cpp` 中：

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericMatrix boot_mean_sd_cpp(
  NumericVector x,
  int B = 200){
  int n = x.size();
  NumericMatrix out_stat(B, 2);
  NumericVector x_boot(n);

  for(int j=0; j<B; j++){
    x_boot = x[floor(runif(n, 0, n))];
    out_stat(j,0) = mean(x_boot);
    out_stat(j,1) = sd(x_boot);
```

```
  }

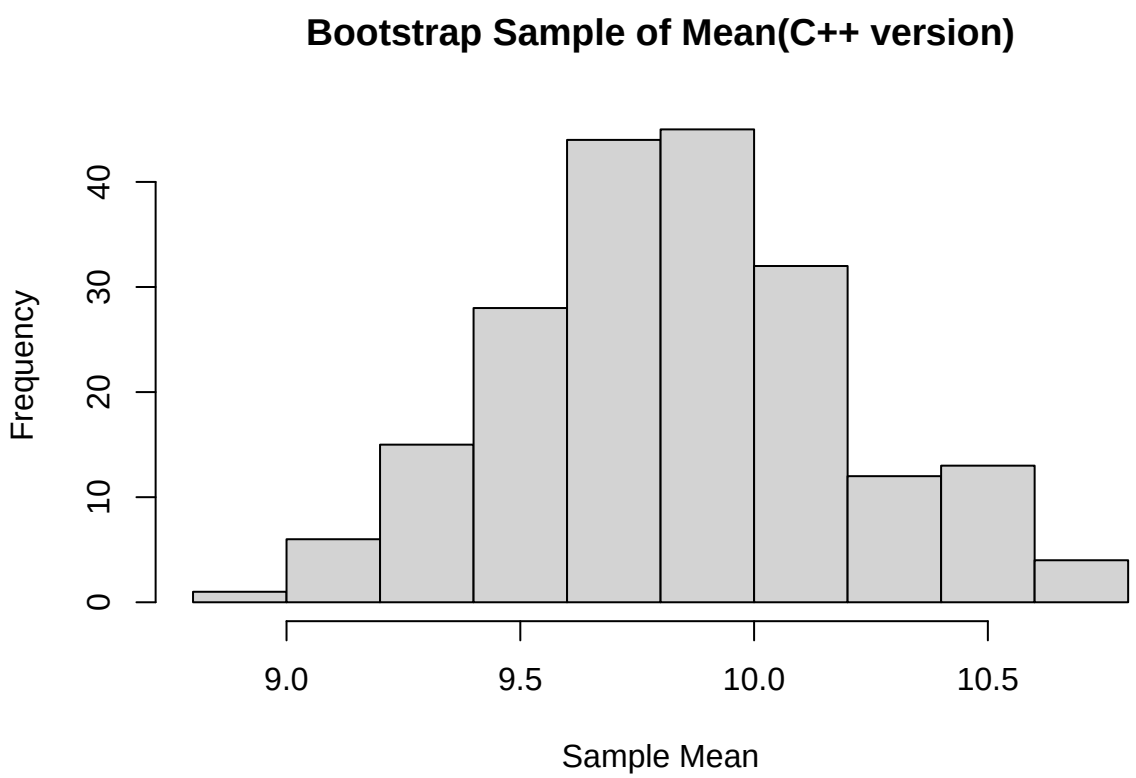
  return out_stat;
}
```

导入为 R 函数：

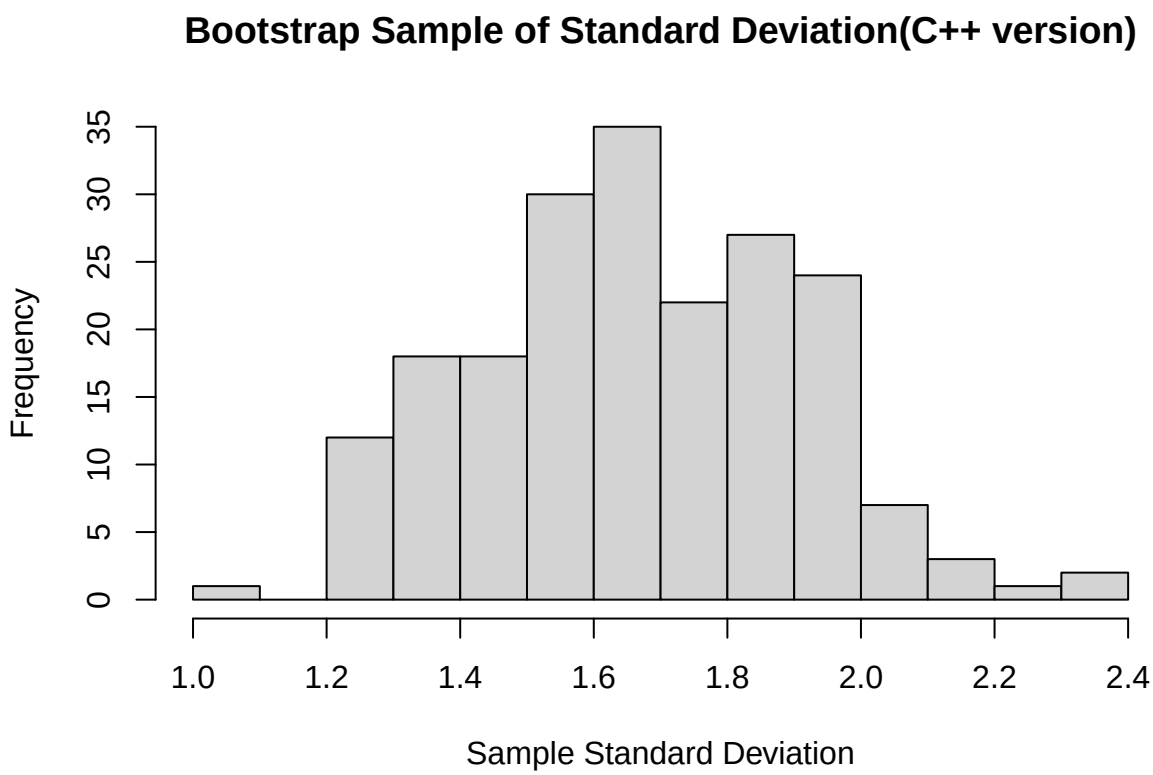
```
library(Rcpp)
sourceCpp(file="boot-mean-sd.cpp")
```

运行测试：

```
set.seed(101)
xsamp <- rnorm(20, mean=10, sd=2)
set.seed(111)
stats_cpp <- boot_mean_sd_cpp(xsamp)
hist(stats_cpp[,1],
  main="Bootstrap Sample of Mean(C++ version)",
  xlab = "Sample Mean")
```



```
hist(stats_cpp[,2],
  main="Bootstrap Sample of Standard Deviation(C++ version)",
  xlab = "Sample Standard Deviation")
```



在 C++ 中抽样用了 `runif()` 函数，而 R 版本用的是 `sample()`，所以两个版本的结果不完全相同。

注意 C 和 C++ 的数组下标是从 0 开始计数的，所以上面 `out_stat` 第 $j+1$ 行第 1 列是 `out_stat(j,0)`，另外要注意矩阵用的是圆括号而不是方括号，因为 C 和 C++ 中的矩阵下标格式是 `A[i][j]`。

45.2.8 在 Rmd 文件中使用 C++ 源程序文件

在 Rmd 文件中，可以使用特殊的 `Rcpp` 代码块，其中包含 C++ 源代码，可以直接起到对其调用 `sourceCpp()` 的作用。

在 Rmd 文件中输入如下的代码块：

```
```{Rcpp}
#include <Rcpp.h>
using namespace Rcpp;
```
```

```
//[[Rcpp::export]]
NumericVector convolveCpp(
    NumericVector a, NumericVector b){
  int na = a.size(), nb = b.size();
  int nab = na + nb - 1;
  NumericVector xab(nab);

  for(int i=0; i < na; i++)
    for(int j=0; j < nb; j++)
      xab[i+j] += a[i] * b[j];

  return xab;
}
```

可以使得 C++ 函数 `convolveCpp()` 被编译并转换成 R 可以调用的同名函数，调用如：

```
convolveCpp(1:5, 1:3)
## [1]  1  4 10 16 22 22 15
```

Rcpp 代码块也可以加 `cache=TRUE` 选项，使得 C++ 程序在没有修改时不必每次重新编译。

在 MS Windows 系统下如果出错,可以将 RTools 安装在默认路径,即 `C:\RTools\bin` 与 `C:\RTools\mingw_64\bin` 中，并可能需要将这两个路径人为地添加到系统的 PATH 变量中，办法是 “Windows 程序–Windows 系统–控制面板–系统和安全–系统–高级系统设置–高级–环境变量–系统变量”，其中的 PATH 变量是用分号分开的可执行程序搜索路径，将其中添加 RTools 包的 `C:\RTools\bin` 与 `C:\RTools\mingw_64\bin`。

Chapter 46

R 与 C++ 的类型转换

R 程序与由 Rcpp 支持的 C++ 程序之间需要传递数据，就需要将 R 的数据类型经过转换后传递给 C++ 函数，将 C++ 函数的结果经过转换后传递给 R。

46.1 用 wrap() 把 C++ 变量返回到 R 中

在 R API 中用.Call() 函数调用 C 程序库函数时，R 对象的数据类型一般是 SEXP。Rcpp 提供了模板化的 `wrap()` 函数把 C++ 的函数返回值转换成 R 的 SEXP 数据类型。此函数的声明为

```
template <typename T> SEXP wrap(const T& object);
```

`wrap()` 能转换的类型包括：

- 把 int, double, bool 等基本类型转换为 R 的原子向量类型（所有元素数据类型相同的向量）;
- 把 `std::string` 转换为 R 的字符型向量;
- 把 STL 容器如 `std::vector<T>` 或 `std::map<T>` 转换成基本类型为 T 的向量，条件是 T 能够转换；
- 把 STL 的映射 `std::map<std::string, T>` 转换为基本类型为 T 的有名向量，条件是 T 能够转换；
- 可以转换定义了 `operator SEXP()` 的 C++ 类的对象；
- 可以转换专门化过 wrap 模板的 C++ 对象。

是否可用 wrap() 转换是在编译时确定的。

46.2 用 as() 函数把 R 变量转换为 C++ 类型

Rcpp 提供了模板化的 as() 用来把 SEXP 类型转换成适当的 C++ 类型。as() 函数的声明为:

```
template <typename T> T as(SEXP x);
```

as() 可以把 R 对象转换为基础的类型如 int, double, bool, std::string 等，可以转换到元素为基础类型的 STL 向量如 std::vector 等。如果 C++ 类定义了以 SEXP 为输入的构造函数也可以利用 as() 来转换。as() 可以针对用户自定义类作专门化。

46.3 as() 和 wrap() 的隐含调用

当 C++ 中赋值运算的右侧表达式是一个 R 对象或 R 对象的部分内容时，可以隐含地调用 as() 将其转换成左侧的 C++ 类型。

当 C++ 中赋值运算的左侧表达式是一个 R 对象或其部分内容时，可以隐含地调用 wrap() 将右侧的 C++ 类型转换成 R 类型。

在用 Rcpp 属性 (Rcpp 属性见下一节) 声明的 C++ 函数中，可以直接以 IntegerVector, NumericVector, CharacterVector, Function 等作为自变量类型或返回值，可以与 R 中相应的类型直接对应。

能自动转换到 R 中的缺省值类型还包括：

- 用双撇号界定的字符串常量；
- 十进数值如 10, 4.5;
- 预定义的常数如 true, false, R_NilValue, NA_STRING, NA_INTEGER, NA_REAL, NA_LOGICAL；

Chapter 47

Rcpp 属性

47.1 Rcpp 属性介绍

Rcpp 属性 (attributes) 用来简化把 C++ 函数变成 R 函数的过程，这可以方便在交互使用中将 C++ 和 C 代码载入到 R 中，也有利于扩展包中 C++ 和 C 代码的使用。做法是在 C++ 源程序中加入一些特殊注释，利用其指示自动生成 C++ 与 R 的接口程序。属性是 C++11 标准的内容，现在的编译器支持还不多，所以在 Rcpp 支持的 C++ 程序中写成了特殊格式的注释。

Rcpp 属性有如下优点：

- 降低了同时使用 R 与 C++ 的学习难度；
- 取消了很多繁复的接口代码；
- 可以在 R 会话中很简单地调用 C++ 代码，不需要用户自己考虑编译、连接、接口问题；
- 可以先交互地调用 C++，成熟后改编为 R 扩展包而不需要修改界面代码。

Rcpp 属性的主要组成部分如下：

- 在 C++ 中，提供 `Rcpp::export` 标注要输出到 R 中的 C++ 函数。
- 在 R 中，提供 `sourceCpp()`，用来自动编译连接保存在文件或 R 字符串中的 C++ 代码，并自动生成界面程序把 C++ 函数转换为 R 函数。
- 在 R 中，提供 `cppFunction()` 函数，用来把保存在 R 字符串中的 C++ 函数自动编译连接并转换成 R 函数。提供 `evalCpp()` 函数，用来把保存在 R 字符串中的 C++ 代码片段自动编译连接并执行。
- 在 C++ 中，提供 `Rcpp::depends` 标注，为了 `sourceCpp()` 说明编译连接 C++ 代码时需要的外部头文件和库的位置。
- 在构建 R 扩展包时，提供 `compileAttributes()` R 函数，自动给 C++ 函数生成相应的 `extern C` 声明和`.Call` 接口代码。

47.2 在 C++ 源程序中指定要导出的 C++ 函数

47.2.1 特殊注释//[[Rcpp::export]]

用特殊注释//[[Rcpp::export]] 说明某 C++ 函数需要在编译成动态链接库时，把这个函数导出到链接库的对外可见部分。

例如

```
//[[Rcpp::export]]
NumericVector convolveCpp(
  NumericVector a, NumericVector b){
  ......
}
```

具体程序参见45.2.5(用 sourceCpp() 转换 C++ 源程序文件—卷积例子)。假设此 C++ 源程序保存到了当前工作目录的 conv.cpp 源文件中，为了在 R 中调用此 C++ 程序，只要用如：

```
sourceCpp(file='conv.cpp')
convolveCpp(1:3, 1:5)
```

注意 sourceCpp() 把 C++ 源程序自动进行了编译链接，并把用//[[Rcpp::export]] 说明的 C++ 函数 convolveCpp 转换成了同名的 R 函数。在同一 R 会话内，如果源程序和其依赖资源没有变化（根据文件更新时间判断），就不重新编译 C++ 源代码。

47.2.2 修改导出的函数名

在用特殊注释说明要导出的 C++ 函数时，可以用特殊的 name= 参数指定函数导出到 R 中的 R 函数名。如果不指定，R 函数名和 C++ 函数名是相同的。

例如

```
//[[Rcpp::export(name="conv")]]
NumericVector convolveCpp(
  NumericVector a, NumericVector b){
  ......
}
```

则 C++ 函数 convolveCpp 导入到 R 中后，改名为 “conv”。

47.2.3 可导出的函数

对于要导出的 C++ 函数有如下要求：

- 必须在全局名字空间中定义，而不能在某个 C++ 名字空间声明内定义。
- 自变量必须能够用 `Rcpp::as()` 转换成 C++ 类型；
- 返回值必须是空值或者能够用 `Rcpp::wrap()` 转换成 R 类型。
- 在自变量和返回值类型说明中，必须使用完整的类型，比如 `std::string` 不能简写成 `string`。Rcpp 提供的类型如 `NumericVector` 可以不必用 `Rcpp::` 修饰。

47.3 在 R 中编译链接 C++ 代码

47.3.1 `sourceCpp()` 函数中直接包含 C++ 源程序字符串

`sourceCpp()` 函数可以用 code= 指定一个 R 字符串，字符串的内容是 C++ 源程序，其中还是用特殊注释`//[[Rcpp::export]]` 标识要导出的 C++ 函数。如

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericVector convolveCpp(
    NumericVector a, NumericVector b){
  .........
}
')
convolveCpp(1:3, 1:5)
```

47.3.2 `cppFunction()` 函数中直接包含 C++ 函数源程序字符串

对于比较简单的单个 C++ 函数，可以用 `cppFunction()` 函数的 code= 指定一个 R 字符串，字符串的内容是一个 C++ 函数定义，转换为一个 R 函数。例如

```
cppFunction(code='
  int fibonacci(const int x){
    if(x < 2) return x;
    else
      return ( fibonacci(x-1) + fibonacci(x-2) );
```

```
  }
')
print(fibonacci(5))
```

47.3.3 evalCpp() 函数中直接包含 C++ 源程序表达式字符串

为了在 R 中计算一个简单的 C++ 表达式，可以用 `evalCpp('C++ 表达式内容')`，如

```
evalCpp('std::numeric_limits<double>::max()')
```

函数将返回该 C++ 表达式的值。

47.3.4 用 depends 指定要链接的库

在 `cppFunction()` 和 `evalCpp()` 中，可以用 `depends` 参数指定要链接的其它库，如

```
sourceCpp(depends='RcppArmadillo', code='......')
```

在编译代码时与 RcppArmadillo 的动态连接库连接。

也可以把这样的链接依赖关系写在特殊的 C++ 注释中，如

```
//[[Rcpp::depends(RcppArmadillo)]]

#include <RcppArmadillo.h>
using namespace Rcpp;

// [[Rcpp::export]]
List fastLm(NumericVector yr, NumericMatrix Xr) { ... }
```

可以依赖多个库，如 `Rcpp::depends(Matrix, RcppArmadillo)`。

这样的注释仅对 `sourceCpp()` 和 `cppFunction()` 有效，如果是在编译 R 扩展包，仍需要把依赖的包列在 DESCRIPTION 文件的 Imports 中，把要链接的包列在 LinkingTo 中。

47.4 Rcpp 属性的其它功能

47.4.1 自变量有缺省值的函数

借助于 Rcpp 属性, 自变量有缺省值的 C++ 函数可以自动转换成自变量有缺省值的 R 函数。定义时要符合 C++ 语法，比如带缺省值的自变量都要在不带缺省值的自变量的后面，缺省值不能有变量。

例如

```
DataFrame readData(
  CharacterVector file,
  CharacterVector colNames = CharacterVector::create(),
  std::string comment = "#",
  bool header = true){ ... }
```

转换到 R 中，相当于

```
function(
  file,
  colNames=character(),
  comment="#",
  header=TRUE)
```

47.4.2 异常传递

Rcpp 支持的 C++ 源程序不能直接调用 R 的 `stop`，可以使用 C++ 的异常，如：

```
if (unexpectedCondition)
  throw Rcpp::exception(
  "Unexpected condition occurred");
```

可简写成：

```
if (unexpectedCondition)
  Rcpp::stop(
  "Unexpected condition occurred");
```

C++ 抛出的异常先被 Rcpp 捕获，然后传递给 R，变成调用 `stop()`。

也可以将警告信息发送给 R：

```
if (unexpectedCondition)
  Rcpp::warning(
  "Unexpected condition occurred");
```

47.4.3 允许用户中断

在 C++ 代码中进行长时间的计算时，应该允许用户可以中断计算。Rcpp 的办法是在 C++ 计算过程中每隔若干步循环就插入一个 `Rcpp::checkUserInterrupt();` 语句。如：

```
for (int i=0; i<1000000; i++) {
  // check for interrupt every 1000 iterations
  if (i % 1000 == 0)
    Rcpp::checkUserInterrupt();
  // ...do some expensive work...
}
```

可以每一两秒检查一次用户中断。R 用户在用 Esc 键或者 RStudio 的停止按钮引发用户中断时，Rcpp 捕捉到中断信号并终止运行，退回到 R 命令行。

47.4.4 把 R 代码写在 C++ 源文件中

正常情况下，应该把 R 代码和 C++ 代码写在分别的源程序中，当 C++ 代码比较短时，也可以把 C++ 代码写在 R 源程序中作为一个字符串。

Rcpp 允许把 C++ 代码和 R 代码都写在一个 C++ 源文件中，R 代码作为特殊的注释，以`/*** R` 行开头，以正常的 `*/`结束。在 R 中用 `sourceCpp()` 调用这个 C++ 源文件，就可以编译 C++ 后执行其中特殊注释内的 R 代码。这样的特殊注释可以有多个。

例如，下述内容保存在文件 `fibo.cpp` 中:

```
//[[Rcpp::export]]
int fibonacci(const int x){
    if(x < 2) return x;
    else
      return ( fibonacci(x-1) + fibonacci(x-2) );
}

/*** R
  # 调用 C++ 中的 fibonacci() 函数
  print(fibonacci(10))
*/
```

只要在 R 中运行

```
sourceCpp(file='fibo.cpp')
```

就可以编译连接此 C++ 文件，把其中用`//[[Rcpp::export]]` 标识的函数转换为 R 函数，并在 R 中执行源文件内特殊注释中的 R 代码。

47.4.5 用 invisible 要求函数结果不自动显示

R 中用 `invisible()` 函数说明函数的返回值，则在命令行调用该函数时不自动显示函数的返回值。对 Rcpp 编译的函数，只有不返回值（返回值用 `void` 声明）时才是不显示的，如果要增加不显示返回值的要求，可以在 `Rcpp::export` 特殊注释中加 `invisible = true` 选项，如：

```
// [[Rcpp::export(invisible = true)]]
NumericVector convolveCpp(
  NumericVector a,
  NumericVector b){ ... }
```

47.4.6 在 C++ 中调用 R 的随机数发生器

在 C 或 C++ 中调用 R 的随机数发生器，需要能够同步地更新随机数发生器状态。如果利用 Rcpp 属性编译 C++ 源程序，则 Rcpp 属性会自动添加一个 RNGScope 实例进行随机数发生器状态的同步。

Chapter 48

Rcpp 提供的 C++ 数据类型

48.1 RObject 类

Rcpp 包为 C++ 定义了 NumericVector, IntegerVector, CharacterVector, Matrix 等新数据类型，可以直接与 R 的 numeric, charactor, matrix 对应。

Rcpp 最基础的 R 数据类型是 RObject, 这是 NumericVector, IntegerVector 等的基类, 通常不直接使用。RObject 包裹了原来 R 的 C API 的 SEXP 数据结构，并且提供了自动的内存管理，不再需要用户自己处理建立内存和消除内存的工作。RObject 存储数据完全利用 R C API 的 SEXP 数据结构，不进行额外的复制。

因为 RObject 类是基类，所以其成员函数也适用于 NumericVector 等类。`isNULL`, `isObject`, `isS4` 可以查询是否 NULL, 是否对象,是否 S4 对象。`inherits` 可以查询是否继承自某个特定类。用 `attributeNames`, `hasAttribute`, `attr` 可以访问对象的属性。用 `hasSlot`, `slot` 可以访问 S4 对象的插口（slot）。

RObject 有如下导出类:

- IntegerVector: 整数向量；
- NumercicVector: 数值向量；
- LogicalVector: 逻辑向量；
- CharacterVector: 字符型向量；
- GenericVector: 列表；
- ExpressionVector: 表达式向量；
- RawVector: 元素为 raw 类型的向量。
- IntergerMatrix, NumericMatrix: 整数值或数值矩阵。

在 R 向量中，如果其元素都是同一类型（如整数、双精度数、逻辑、字符型），则称为原子向量。Rcpp 提供了 IntegerVector, NumericVector, LogicalVector, CharacterVector 等数据类型与 R 的原子向量类型对应。在 C++ 中可以用 `[]` 运算符存取向量元素，也可以用 STL 的迭代器。用`.begin()`, `.end()` 等界定范围，用循环或或者 accumulate 等 STL 算法处理整个向量。

48.2 IntegerVector 类

在 R 中通常不严格区分整数与浮点实数，但是在与 C++ 交互时，C++ 对整数与实数严格区分，所以 RCpp 中整数向量与数值向量是区分的。

在 R 中，如果定义了一个仅有整数的向量，其类型是整数 (integer) 的，否则是数值型 (numeric) 的，如：

```
x <- 1:5
class(x)
## [1] "integer"
y <- c(0, 0.5, 1)
class(y)
## [1] "numeric"
```

用 `as.integer()` 和 `as.numeric()` 函数可以显式地确保其自变量转为需要的整数型或数值型。

RCpp 可以把 R 的整数向量传递到 C++ 的 IntegerVector 中，也可以把 C++ 的 IntegerVector 函数结果传递回 R 中变成一个整数向量。也可以在 C++ 中生成 IntegerVector 向量，填入整数值。

48.2.1 IntegerVector 示例 1：返回完全数

如果一个正整数等于它所有的除本身以外的因子的和，称这个数为完全数。如

$$6 = 1 + 2 + 3$$
$$28 = 1 + 2 + 4 + 7 + 14$$

是完全数。

任务：用 C++ 程序输入前 4 个完全数偶数，返回到 R 中。这 4 个数为 6, 28, 496, 8182。

程序：

```
library(Rcpp)
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;
```

```
//[[Rcpp::export]]
IntegerVector wholeNumberA(){
    IntegerVector epn(4);
    epn[0] = 6;    epn[1] = 28;
    epn[2] = 496;  epn[3] = 8182;

    return epn;
}
')
print(wholeNumberA())
## [1]    6   28  496 8182
```

对 `IntegerVector` 类型的 `x`，访问单个下标 `i` 格式为 `x[i]`，下标从 0 开始计算，与 R 向量下标从 1 开始的规则不同。

从例子看出，可以在 C++ 中建立一个 IntegerVector, 需指定大小。可以逐个填入数值。直接返回 IntegerVector 到 R 即可，不需用 `wrap()` 显式地转换，函数返回值的声明可以用 `IntegerVector` 声明。

48.2.2 IntegerVector 示例 2：输入整数向量

任务：用 C++ 编写函数，从 R 中输入整数向量，计算其元素乘积（与 R 的 `prod()` 函数功能类似）。程序如下：

```
library(Rcpp)
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
IntegerVector prod1(IntegerVector x){
    int prod = 1;
    for(int i=0; i < x.size(); i++){
        prod *= x[i];
    }
    return wrap(prod);
}
')
print(prod1(1:5))
```

为了获得向量的元素个数，用 `x.size()` 这种方法。

从程序看出，用 IntegerVector 从 R 中接受一个整数值向量时，不需要显式地转换。把一个 C++ 整数值返回给 R 时，可以用 IntegerVector 返回，因为返回值原来是一个 C++ 的 `int` 类型，所以需要用 `Rcpp::wrap()` 转换一下。在 `sourceCpp` 中可以省略 `Rcpp::` 部分。

也可以返回 int, 返回值类型用 int 声明：

```
library(Rcpp)
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
int prod2(IntegerVector x){
    int prod = 1;
    for(int i=0; i < x.size(); i++){
        prod *= x[i];
    }
    return prod;
}
')
print(prod2(1:5))
```

还可以用 C++ STL 的算法库进行这样的累计乘积计算，`std::accumlate()` 可以对指定范围进行遍历累计运算。前两个参数是一个范围，用迭代器 (iterators) 表示开始和结束，第三个参数是初值，第四个参数是对每个元素累计的计算。程序如下：

```
require(Rcpp)
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
int prod3(IntegerVector x){
    int prod = std::accumulate(
        x.begin(), x.end(),
        1, std::multiplies<int>());
    return prod;
}
')
print(prod3(1:5))
```

在以上的输入 IntegerVector 的 C++ 程序中，如果从 R 中输入了实数型的向量，则元素被转换成整数型。比如 `prod2(seq(1,1.9,by=0.1))` 结果将等于 1。如果输入了无法转换为整数型向量的内容，比如 `prod2(c('a','b','c'))`, 程序会报错。

48.3 NumericVector 类

NumericVector 类在 C++ 中保存双精度型一维数组，可以与 R 的实数型向量 (class 为 numeric) 相互转换。这是自己用 C++ 程序与 R 交互时最常用到的数据类型。

`x.size()` 返回元素个数。

48.3.1 示例 1：计算元素 p 次方的和

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
double ssp(
    NumericVector vec, double p){
  double sum = 0.0;
  for(int i=0; i < vec.size(); i++){
    sum += pow(vec[i], p);
  }
  return sum;
}
')
ssp(1:4, 2)
## [1] 30
ssp((1:4)/10, 2.2)
## [1] 0.2392496
sum( ((1:4)/10)^2.2 )
## [1] 0.2392496
```

从程序看出，用 Rcpp 属性编译时，C++ 函数的输入与返回值类型转换有如下规则：R 中数值型向量在 C++ 中可以用 NumericVector 接收；R 中单个的实数在 C++ 中可以用 double 来接收；为了返回单个的实数，可以直接返回 double，也可以返回 NumericVector 类型, 或者从一个 double 型用 `wrap()` 转换。C++ 中如果返回 NumericVector, 在 R 中转换为数值型向量。

48.3.2 示例 2：clone 函数

在自定义 R 函数时，输入的自变量的值不会被改变，相当于自变量都是局部变量。如果在自定义函数中修改了自变量的值，实际上只能修改自变量的一个副本的值。如

```
x <- 100
f <- function(x){
  print(x); x <- 99; print(x)
}
c(f(x), x)
## [1] 100
## [1] 99
## [1]  99 100
```

但是，在用 Rcpp 编写 R 函数时，因为 RObject 传递的是指针，并不自动复制自变量值，所以修改自变量值会真的修改原始的自变量变量值。如：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericVector f2(NumericVector x){
  x[0] = 99.0;
  return(x);
}
')
x <- 100
c(f2(x), x)
[1] 99 99
```

可见自变量的值被修改了。当然，对这个问题而言，因为输入的是一个标量，只要函数自变量不是 NumericVector 类型而是用 double 类型，则自变量值会被复制，达到值传递的效果，自变量值也就不会被真的修改。如

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericVector f3(double x){
  x = 99.0;
```

```
  return(wrap(x));
}
')
x <- 100
c(f3(x), x)
## [1]  99 100
```

下面的程序把输入向量每个元素平方后返回，为了不修改输入自变量的值而是返回一个修改后的副本，使用了 Rcpp 的 clone 函数：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericVector square(NumericVector x){
  NumericVector y = clone(x);
  for(int i=0; i < x.size(); i++){
    y[i] = x[i]*x[i];
  }
  return(y);
}
')
x <- c(2, 7)
cbind(square(x), x)
##         x
## [1,]  4 2
## [2,] 49 7
```

x.sort() 会对 x 从小到大排序，同时返回排序结果，x 被修改了。如：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericVector test_sort(NumericVector x){
  return x.sort();
}
')
```

```
x <- c(3, 5, 1)
test_sort(x)
## [1] 1 3 5
x
## [1] 1 3 5
```

用 clone 制作副本，避免修改 x:

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericVector test_sort(NumericVector x){
  NumericVector y = clone(x);
  return y.sort();
}
')
x <- c(3, 5, 1)
test_sort(x)
## [1] 1 3 5
x
## [1] 3 5 1
```

48.3.3 示例 3：向量子集

C++ 中 Rcpp 支持的向量和列表，仍可以用正整数向量作为下标。例如，下面的 C++ 函数求指定的元素子集的和：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
double sumsub(
  NumericVector x, IntegerVector i){
  NumericVector y = x[i-1];
  double s = 0.0;
  int n = y.size();
  for(int j=0; j<n; j++)
```

```
    s += y[j];

  return s;
}
')
x <- 1:10
sumsub(x, c(1, 3, 5))
## [1] 9
```

注意在程序中将输入的下标向量减了 1，以适应从 R 规则到 C++ 规则的变化。Rcpp 在输入和输出时会自动在整型和双精度型之间转换，比如上面例子中输入的 x 是整型的，要求输入 NumericVector 是双精度型的，Rcpp 会自动转换。

Rcpp 也支持逻辑下标，比如，下面在 C++ 定义的函数可以取出向量的正值的子集：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericVector get_positive(NumericVector x){
  NumericVector y = x[x > 0];

  return y;
}
')
x <- c(1, -1, 2, -2)
get_positive(x)
## [1] 1 2
```

Rcpp 也支持按元素名访问向量元素或向量子集。

48.4 NumericMatrix 类

`NumericMatrix x(n,m)` 产生一个未初始化的 $n \times m$ 矩阵，元素为 `double` 类型。`x.nrow()` 返回行数，`x.ncol()` 返回列数，`x.size()` 返回元素个数。

下标从 0 开始计算。为了访问 x_{ij}，用 `x(i,j)` 的格式：注意，**不是** `x[i,j]` 也不是 `x[i][j]`。

为了访问 `x` 的第 `i` 行，用 `x.row(i)` 或 `x(i,_)`；为了访问 `x` 的第 `j` 列，用 `x.column(j)` 或 `x(_,j)`。

`transpose(x)` 返回 `x` 的转置。

`NumericMatrix::zeros(n)` 返回 `n` 阶元素为 0 的方阵。`NumericMatrix::eye(n)` 返回 `n` 阶单位阵。

48.4.1 示例 1：计算矩阵各列模的最大值

输入一个矩阵，计算各列的向量模，并返回模的最大值。

```
library(Rcpp)
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
double testfun(NumericMatrix x){
  int nr = x.nrow();
  int nc = x.ncol();
  double y, y1;

  y = 0.0;
  for(int j=0; j<nc; j++){
    y1 = 0.0;
    for(int i=0; i<nr; i++){
      y1 += pow(x(i,j), 2);
    }
    if(y1 > y) y = y1;
  }
  y = sqrt(y);

  return y;
}
')
x <- matrix(c(1, 2, 4, 9), 2, 2)
print(x)
print(testfun(x))
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    9
## [1] 9.848858
```

其中对每列计算模的部分，也可以改写成：

```
  y = 0.0;
  NumericVector ycol(nr);
  for(int j=0; j<nc; j++){
    ycol = x.column(j);
    y1 = sum(ycol * ycol);
    if(y1 > y) y = y1;
  }
  y = sqrt(y);
```

这里用了 `x.column(j)` 函数提取矩阵的一列，用了 `ycol * ycol` 作向量元素之间的四则运算，用了 R 的 `sum` 函数求和。

48.4.2 示例 2：把输入矩阵制作副本计算元素平方根

下面的例子输入一个 R 矩阵，输出其元素的平方根，用了 clone 函数来避免对输入的直接修改。没有按行列循环而是将矩阵看作一个按列拉直的向量进行遍历，矩阵实际上也是按向量存储的。

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericMatrix matSqrt(NumericMatrix x){
  NumericMatrix y = clone(x);
  std::transform(y.begin(), y.end(),
    y.begin(), ::sqrt);
  return(y);
}
')
x <- rbind(c(1,2), c(3,4))
cbind(matSqrt(x), x)
##          [,1]     [,2] [,3] [,4]
## [1,] 1.000000 1.414214    1    2
## [2,] 1.732051 2.000000    3    4
```

在上面的 C++ 程序中，NumericMatrix 看成了一维数组，用 STL 的 iterater 遍历，用 STL 的 transform 对每个元素计算变换。

48.4.3 示例 3：访问列子集

可以用 `x(_,j)` 这样的格式将矩阵 `x` 的第 `j` 列取出为普通向量，并可以赋值修改。也可以用 `x.column(j)`，用法相同。`x(i,_)` 取出第 `i` 行为一个普通向量。比如，下面的 C++ 函数将输入的矩阵的每一行计算累加和：

```
library(Rcpp)
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericMatrix testfun(NumericMatrix x){
  NumericMatrix y = clone(x);
  for(int j=1; j<x.ncol(); j++){
    y(_,j) = y(_,j-1) + x(_,j);
  }

  return y;
}
')
x <- matrix(c(1, 2, 4, 9), 2, 2)
print(x)
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    9
print(testfun(x))
##      [,1] [,2]
## [1,]    1    5
## [2,]    2   11
```

48.5 Rcpp 的其它向量类

48.5.1 Rcpp 的 LogicalVector 类

LogicalVector 类可以存储 C++ 值 true, false, 还可以保存缺失值 NA_REAL, R_NaN, R_PosInf, 但是这些不同的缺失值转换到 R 中都变成 NA。

如:

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
LogicalVector f4(){
  LogicalVector x(5);
  x[0] = false; x[1] = true;
  x[2] = NA_REAL;
  x[3] = R_NaN; x[4] = R_PosInf;
  return(x);
}
')
f()
## [1] FALSE  TRUE    NA    NA    NA
identical(f(), c(FALSE, TRUE, rep(NA,3)))
## [1] TRUE
```

48.5.2 Rcpp 的 CharacterVector 类型

CharacterVector 类型可以与 R 的字符型向量相互交换信息，在 C++ 中其元素为字符串。字符型缺失值在 C++ 中为 R_NAString。R 的字符型向量也可以转换为 std::vector。

Rcpp 定义了 `String` 用来表示单个字符串。

如：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
CharacterVector f5(){
  CharacterVector x(3);
  x[0] = "This is a string";
  x[1] = "Test";
  x[2] = R_NaString;
  return(x);
}
')
```

```
f5()
## [1] "This is a string" "Test"             NA
```

48.6 Rcpp 提供的其它数据类型

48.6.1 Named 类型

R 中的向量、矩阵、数据框可以有元素名、列名、行名。这些名字可以借助 Named 类添加。

例如：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericVector f6(){
  NumericVector x = NumericVector::create(
    Named("math") = 82,
    Named("chinese") = 95,
    Named("English") = 60);
  return(x);
}
')
f6()
##   math chinese English
##     82      95      60
```

“Named(元素名)” 可以简写成 “_元素名”。如：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericVector f6b(){
  NumericVector x = NumericVector::create(
    _["math"] = 82,
    _["chinese"] = 95,
    _["English"] = 60);
```

```
  return(x);
}
')
```

48.6.2 List 类型

Rcpp 提供的 List 类型对应于 R 的 list(列表) 类型，在 C++ 中也可以写成 GenericVector 类型。其元素可以不是同一类型，在 C++ 中可以用方括号和字符串下标的格式访问其元素。

例如，下面的函数输入一个列表，列表元素 vec 是数值型向量，列表元素 multiplier 是数值型标量，返回一个列表，列表元素 sum 为 vec 元素和，列表元素 dsum 为 vec 元素和乘以 multiplier 的结果：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
List f7(List x){
  NumericVector vec = as<NumericVector>(x["vec"]);
  double multiplier = as<double>(x["multiplier"]);
  double y = 0.0, y2;
  for(int i=0; i<vec.length(); i++){
    y += vec[i];
  }
  y2 = y*multiplier;
  return(List::create(Named("sum")=y,
    Named("dsum")=y2));
}
')
f7(list(vec=1:5, multiplier=10))
## $sum
## [1] 15
##
## $dsum
## [1] 150
```

上面的程序用了 `Rcpp::List::create()` 当场生成 List 类型，因为用 Rcpp 属性功能编译所以可以略写 `Rcpp::`。也可以在程序中预先生成指定大小的列表，然后再给每个元素赋值，元素值可以是任意能够转化为 SEXP 的类型，如：

```
.........
List gv(2);
gv[0] = "abc";
gv[1] = 123;
```

可以用 List 的 reserve 函数为列表指定元素个数。

48.6.3 Rcpp 的 DataFrame 类

Rcpp 的 DataFrame 类用来与 R 的 data.frame 交换信息。

示例如：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
DataFrame f8(){
  IntegerVector vec =
    IntegerVector::create(7,8,9);
  std::vector<std::string> s(3);
  s[0] = "abc"; s[1] = "ABC"; s[2] = "123";
  return(DataFrame::create(
    Named("x") = vec,
    Named("s") = s));
}
')
f8()
##   x   s
## 1 7 abc
## 2 8 ABC
## 3 9 123
```

48.6.4 Rcpp 的 Function 类

Rcpp 的 Function 类用来接收一个 R 函数，并且可以在 C++ 中调用这样的函数。

示例如:

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
SEXP dsort(Function sortFun, SEXP x){
  return sortFun(x, Named("decreasing", true));
}
')
dsort(sort, c('bb', 'ab', 'ca'))
## [1] "ca" "bb" "ab"
## dsort(sort, c(2,1,3))
## [1] 3 2 1
```

程序用 Function 对象 sortFun 接收从 R 中传递过来的排序函数，实际调用时传递过来的是 R 的 sort 函数。

在 C++ 中调用 R 函数时，有名的自变量用 “Named(自变量字符串, 自变量值)” 的格式给出。

程序中的待排序的向量与排序后的向量都用了 SEXP 来说明，即直接传送原始 R API 指针，这样可以不管原来类型是什么，在 C++ 中完全不进行类型转换或复制。

从运行例子看出数值和字符串都正确地按照降序排序后输出了。

R 函数可以不作为 C++ 的函数自变量传递进来，而是直接调用 R 的函数。R 的许多函数都已经在 Rcpp 名字空间中输出了，所以可以直接调用 R 的各个向量化的函数，包括随机数函数。例如，下面的 C++ 函数返回一个各列为随机数的数值型矩阵：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericMatrix rngCpp(const int N) {
  NumericMatrix X(N, 4);
  X(_, 0) = runif(N);
  X(_, 1) = rnorm(N);
  X(_, 2) = rt(N, 5);
  X(_, 3) = rbeta(N, 1, 1);
  return X;
}')
```

使用了 Rcpp 属性时，生成的界面程序会自动生成一个 RNGScope 的实例，用来保存当前的随机数发生器状态，在

解构时将自动更新随机数发生器状态。这是 R 调用编译代码时对编译代码的要求，如果不使用 Rcpp 和属性，就需要用户自己添加命令来维护随机数种子。

测试上面的随机数调用：

```
set.seed(101)
xm <- rngCpp(3)
xm
##            [,1]       [,2]      [,3]       [,4]
## [1,] 0.37219838  0.4061679  0.107946 0.78664800
## [2,] 0.04382482 -0.5242428  1.539425 0.07668112
## [3,] 0.70968402 -0.4303593 -2.259442 0.92878745
set.seed(101)
xm2 <- cbind(
  runif(3),
  rnorm(3),
  rt(3, 5),
  rbeta(3,1,1)
)
all.equal(xm, xm2)
## [1] TRUE
```

还可以使用名字空间 R 中的标量随机数发生器，如：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector rngCppScalar() {
  NumericVector x(4);
  x[0] = R::runif(0,1);
  x[1] = R::rnorm(0,1);
  x[2] = R::rt(5);
  x[3] = R::rbeta(1,1);
  return(x);
}')
```

48.6.5 Rcpp 的 Environment 类

R 的环境是分层的，可以逐层查找变量名对应的内容。Rcpp 的 Environment 类用来与 R 环境对应。可以利用 Environment 来定位 R 的扩展包中的函数或数据，例如下面的程序片段在 C++ 中定位了 stats 扩展包中的 rnorm 函数并进行了调用:

```
Environment stats("package:stats");
Function rnorm = stats["rnorm"];
return rnorm(3, Named("sd", 100.0));
```

当然，也可以用 Function 对象直接从各环境中搜索 rnorm 函数名，但是这样指定环境更可靠。

下面的例子访问了 R 全局环境，取出了全局变量 x 的值存入 C++ 的双精度型 STL 向量中，并把一个 C++ 的 STL map 型数据转换成有名字符型向量存到了全局变量 y 中。

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
void f9(){
  Environment global = Environment::global_env();
  std::vector<double> vx = global["x"];
  std::map<std::string, std::string> ma;
  ma["foo"] = "abc";
  ma["bar"] = "123";
  global["y"] = ma;
  return;
}
')
x <- c(1,5)
f9()
y
##   bar   foo
## "123" "abc"
```

Chapter 49

Rcpp 糖

在 C++ 中，向量和矩阵的运算通常需要逐个元素进行，或者调用相应的函数。Rcpp 通过 C++ 的表达式模板 (expression template) 功能和懒惰求值 (lazy evaluation) 功能，可以在 C++ 中写出像 R 中对向量和矩阵运算那样的向量化表达式。这称为 Rcpp 糖 (sugar)。

R 中的很多函数如 `sin` 等是向量化的，Rcpp 糖也提供了这样的功能。Rcpp 糖提供了一些向量化的函数如 `ifelse`, `sapply` 等。

比如，两个向量相加可以直接写成 `x + y` 而不是用循环或迭代器 (iterator) 逐元素计算; 若 `x` 是一个 NumericVector, 用 `sin(x)` 可以返回由 `x` 每个元素的正弦值组成的 NumericVector。

Rcpp 糖不仅简化了程序，还提高了运行效率。

49.1 简单示例

比如，函数

$$f(x,y)=\begin{cases} x^2 & x<y, \\ -y^2 & x\geq y \end{cases}$$

如下的程序可以在 C++ 中定义一个向量化的版本：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericVector f11(NumericVector x, NumericVector y){
  return ifelse(x < y, x*x, -(y*y));
}
```

```
')
f11(c(1, 3), c(4,2))
## [1]  1 -4
```

上面简单例子中，`x < y` 是向量化的，`x * x`, `y * y`, `-(y * y)` 都是向量化的。`ifelse` 也是向量化的。

49.2 向量化的运算符

49.2.1 向量化的四则运算

Rcpp 糖向量化了 `+`, `-`, `*`, `/`。设 `x`, `y` 都是 NumericVector, 则 `x + y`, `x - y`, `x * y`, `x * y` 将返回对应元素进行四则运算后的 NumericVector 变量。

向量与标量运算，如 `x + 2.0`, `2.0 - x`, `x * 2.0`, `2.0 / x` 将返回标量与向量每个元素进行四则运算后的 NumericVector 变量。

还可以进行混合四则运算，如：

```
NumericVector res = x * y + y / 2.0;
NumericVector res = x * (y - 2.0);
NumericVector res = x / (y * y);
```

参加四则运算的或者都是同基本类型的向量而且长度相等，或者一边是向量，一边是同类型的标量。

注意：对向量整体赋一个相同的值，不能简单地写成如 `x=0;` 这样的赋值，需要用循环或 STL 的 fill 算法，如 `std::fill(x.begin(), x.end(), 0);`。

49.2.2 向量化的赋值运算

可以将糖表达式的值赋值给一个已分配存储空间的等长向量，如：

```
NumericVector x = NumericVector::create(1.0, 2.0);
NumericVector y(2);
y = x / 2.0;
```

49.2.3 向量化的二元逻辑运算

Rcpp 糖扩展了两个元素的比较到两个等长向量的比较，以及一个向量与一个同类型标量的比较，结果是一个同长度的 LogicalVector。比较运算符包括 `<`, `>`, `<=`, `>=`, `==`, `!=`。

也可以使用嵌套的表达式，比如，设 x, y 是两个 NumericVector，可以写:

```
LogicalVector res = (x + y) < (x * x);
```

49.2.4 向量化的一元运算符

对数值型向量或返回数值型向量的表达式前面加负号，可以返回每个元素取相反数的结果。比如，设 x 是 NumericVector, 可以写:

```
NumericVector res = -x;
NumericVector res = -x * (x + 2.0);
```

对逻辑型向量或返回逻辑型向量的表达式前面加叹号，可以返回每个元素取反的结果，如：

```
NumericVector y, z;
NumericVector res = ! ( y < z );
```

49.3 用 Rcpp 访问数学函数

在 C 和 C++ 代码中可以调用 R 中的函数，但是格式比较复杂，而且原来不支持向量化。Rcpp 糖则使得从 C++ 中调用 R 函数变得和在 R 调用函数格式类似。

R 源程序中提供了许多数学和统计相关的函数，这些函数可以编译成一个独立的函数库，供其它程序链接使用，函数内容在 R 的 Rmath.h 头文件中有详细列表。

Rcpp 糖把 R 中的数学函数在 C++ 中向量化了，输入一个向量，结果是对应元素为函数值的向量。自变量类型可以是数值型或整数型。这些数学函数包括 abs, exp, floor, ceil, pow 等。在 Rcpp 中支持的 C++ 源程序中调用这些函数, 最简单的一种方法是直接在 Rcpp 名字空间中使用和 R 中相同的函数名和调用方法，类似 sqrt 这样的函数允许输入一个向量，对向量的每个元素计算。

49.4 用 Rcpp 访问统计分布类函数

R 中提供了许多分布的分布密度（或概率质量函数）、分布函数、分位数函数，分布密度函数和概率质量函数命名类似 dxxxx, 分布函数命名类似 pxxxx, 分位数函数命名类似 qxxxx。在 Rcpp 糖支持下，这些函数可以直接用类似 R 中的格式调用。比如，下面的程序输入一个向量，计算相应的标准正态分布函数值：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
```

```
NumericVector f10(NumericVector x){
  NumericVector y(x.size());
  y = pnorm(x);
  return y;
}
')
f10(c(0, 1.96, 2.58))
## [1] 0.5000000 0.9750021 0.9950600
```

如果需要对单个的数值计算，可以使用 Rmath.h 中定义的带有 `Rf_` 的版本，如:

```
y = ::Rf_pnorm5(x, 0.0, 1.0, 1, 0);
```

这里用`::` 使用了缺省的名字空间。注意所有自变量都不能省略。

Rcpp 还提供了一个 `R` 名字空间，可以用不带 `Rf_` 前缀的函数，但是自变量也不能省略。如

```
y = R::pnorm(x, 0.0, 1.0, 1, 0);
```

在 Rcpp 的 `R` 名字空间中有许多的数学和统计相关的函数，各函数的自变量参见 Rcpp 的 Rmath.h 文件。

更多分布类函数使用如：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
List f14(){
  NumericVector x =
    NumericVector::create(0, 1.96, 2.58);
  NumericVector p =
    NumericVector::create(0.95, 0.975, 0.995);
  NumericVector y1 = dnorm(x, 0.0, 1.0);
  NumericVector y2 = pnorm(x, 0.0, 1.0);
  NumericVector y3 = qnorm(p, 0.0, 1.0);
  return List::create(Named("y1")=y1,
    Named("y2")=y2, Named("y3")=y3);
}
')
f14()
```

```
## $y1
## [1] 0.39894228 0.05844094 0.01430511
##
## $y2
## [1] 0.5000000 0.9750021 0.9950600
##
## $y3
## [1] 1.644854 1.959964 2.575829
##
```

C++ 中这些分布类函数的分布参数选择可能与 R 中有所不同，比如 R 中 `dexp()` 的参数是 `rate`，而 Rcpp 中是 `scale`。

49.5 在 Rcpp 中产生随机数

R 中的 rxxxx 类的函数可以产生各种分布的随机数向量，随机数向量与当前种子有关。这些函数在 Rcpp 中也可以使用，Rcpp 属性会自动地维护随机数发生器的状态使其与 R 同步。如

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericVector f15(){
  NumericVector x = rnorm(10, 0.0, 1.0);
  return x;
}
')
round(f15(), 2)
## [1]  0.62 -0.06 -0.16 -1.47 -0.48
## [6]  0.42  1.36 -0.10  0.39 -0.05
```

49.6 返回单一逻辑值的函数

在 R 中，`any()` 和 `all()` 对一个逻辑向量分别判断是否有任何真值，以及所有元素为真值。Rcpp 糖在 C++ 中也提供了这样的 `any()` 和 `all()` 函数。如

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
List f12(){
  IntegerVector x = seq_len(1000);
  LogicalVector res1 = any( x*x < 3 );
  LogicalVector res2 = all( x*x < 3 );
  return List::create(Named("any")=res1,
    Named("all")=res2);
}
')
f12()
## $any
## [1] TRUE
##
## $all
## [1] FALSE
```

`any()` 和 `all()` 不是直接返回逻辑值，而是返回一个类对象，该类定义了 `is_true`, `is_false`, `is_na` 方法与向 SEXP 转换的运算符。此种结果可以保存到 LogicalVector 中，但不能赋值到 bool 类型，因为可能有缺失值。

逻辑型糖表达式结果可以用 `is_true`, `is_false`, `is_na` 来判断结果。之所以这样，而不是将表达式结果直接转换为布尔型，是因为 R 的这种结果可能是缺失值，而布尔型并不包括缺失值。如

```
bool res1 = is_true( any( x < y ) );
bool res2 = is_na( all( x < y ) );
```

在求 `any()` 结果时，一旦遇到一个真值结果就为真值，即使后面有缺失值也没有关系。在求 `all()` 结果时，一旦遇到一个假值结果就为假值，即使后面有缺失值也没有关系。

49.7 返回糖表达式的函数

49.7.1 `is_na`

`is_na` 以任何糖表达式为输入，输出一个逻辑类型的元素个数相同的糖表达式。结果每个元素当输入中对应元素缺失时为 TRUE, 否则为 FALSE。如

```
IntegerVector x = IntegerVector::create(0, 1, NA_INTEGER, 3);
LogicalVector res1 is_na(x);
bool res2 = all( is_na(x) );
if( is_true( any( ! is_na(x) ) ) ) ...
```

49.7.2 seq_along

seq_along 输入一个向量，输出一个元素为该向量的各个下标值的糖表达式。如

```
IntegerVector x = IntegerVector::create(0, 1, NA_INTEGER, 3);
IntegerVector ind1 = seq_along(x);
IntegerVector ind2 = seq_along(x*x*x*x*x);
```

注意上述程序不会计算 x*x*x*x*x 的值而只利用其结果的元素个数。所以两次调用的计算量是一样的。

49.7.3 seq_len

seq_len 自变量为个数，返回一个元素值为正数的元素个数等于自变量值的糖表达式，第 i 个元素等于 i。常可与 sapply, lapply 配合使用。如

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
List f13(){
  IntegerVector x =seq_len(3);
  List y = lapply( seq_len(3),  seq_len);
  return List::create(Named("x")=x,
    Named("y")=y);
}
')
f13()
## $x
## [1] 1 2 3
##
## $y
## $y[[1]]
## [1] 1
```

```
##
## $y[[2]]
## [1] 1 2
##
## $y[[3]]
## [1] 1 2 3
##
```

49.7.4 pmin 和 pmax

pmin 和 pmax 自变量为两个等长的向量或糖表达式，返回长度相同的结果，结果元素等于对应元素的最小值或最大值。

自变量也可以取一个向量一个标量，向量的每个元素与标量比较得到最小值或最大值。如

```
IntegerVector x = seq_len(10);
IntegerVector y = pmin(x, x*x);
IntegerVector z = pmin(x*x, 2);
```

49.7.5 ifelse

ifelse 函数有三个自变量，第一自变量是一个逻辑型向量值的糖表达式，第二和第三自变量或者是两个同类型并与第一自变量等长的糖表达式，或者其中一个是同类型标量，结果仍为向量型糖表达式，第 i 元素当第一自变量第 i 元素为真时取第一自变量的第 i 元素，当第一自变量第 i 元素为假时取第二自变量的第 i 元素，当第一自变量第 i 元素为缺失值时取缺失值。如

```
IntegerVector x, y;
IntegerVector res1 = ifelse( x < y, x, (x+y)*y );
IntegerVector res2 = ifelse( x > y, x, 2 );
```

49.7.6 sapply 和 lapply

sapply 函数第一自变量是一个向量或列表，第二自变量是一个函数。返回值类型在编译时从函数的结果类型导出。第二自变量可以是任意的 C++ 函数，比如，可以是如下的重载的模板化函数：

```
template <typename T>
T square( const T& x ){
  return x * x;
}
sapply( seq_len(4), square<int> );
```

下面的例子中 `sapply` 第二自变量使用了 functor，这是一种能够产生函数的函数：

```
template <typename T>
struct square : std::unaray_function<T,T> {
  T operator() (const T& x) {
    return x * x;
  }
}
sapply( seq_len(4), square<int>() );
```

`lapply` 函数与 `sapply` 函数基本相同，只不过 `lapply` 函数总是返回列表, 列表在 Rcpp 中为 List 或 GenericVector, 在 R API 中类型为 VECSXP。

49.7.7 sign

`sign` 函数输入一个数值型或整型表达式，返回各元素值在 $\{1, 0, -1, \mathrm{NA}\}$ 中取值的糖表达式，可以保存到 IntegerVector 中。结果各元素的取值表示输入中对应元素为正、零、负和缺失。如

```
IntegerVector x;
IntegerVector res1 = sign( x );
IntegerVector res2 = sign( x*x );
```

49.7.8 diff

`diff` 函数自变量为数值型或整数型向量或有这样结果的糖表达式，输出后一元素减去前一元素的结果，结果长度比输入长度少一。如

```
IntegerVector res = diff( seq_len(5) );
```

49.8 R 与 Rcpp 不同语法示例

RCpp 糖通过一些现代的 C++ 技术，支持部分的向量化运算，比如，NumericVector 与标量相加，两个等长 NumericVector 相加，对 NumericVector 计算如绝对值这样的数学函数值等。但是，C++ 毕竟和 R 有很大差别，要区分 Rcpp 能做的和不能做的。

例如，NumericVector 为了把向量所有元素都改成同一值，不能直接用等于赋值。可以用 `std::fill(y.begin(), y.end(), 99.99)` 这样的做法。

49.9 用 RcppArmadillo 执行矩阵运算

虽然 Rcpp 糖提供了许多与 R 类似的向量化功能，但是 Rcpp 仍未支持像 R 那样用简单表达式表示矩阵运算。RcppArmadillo 包提供了与 C++ 库 Armadillo 的接口，Armadillo 库利用 C++ 的模板功能实现了矩阵运算表达式功能，这样，借助于 Rcpp 和 RcppArmadillo，可以将一些涉及到矩阵的算法完全用 C++ 完成，而 Rcpp 可以轻易地将 R 输入转换为 C++ 类型，将 C++ 代码的输出转换为 R 类型。

本节例子来自 Rcpp 文档。

49.9.1 生成多元正态分布随机数

设随机向量 X 服从多元正态分布 $\mathrm{N}(\mu, \Sigma)$，为了生成 X 的随机样本，可以从标准多元正态分布 $Z \sim \mathrm{N}(0, I)$ 的样本进行线性变换得到 X 的样本。有两种方法，一种是对 Σ 作 Cholesky 分解 $\Sigma = AA^T$，其中 A 是下三角阵，然后令 $X = \mu + AZ$。另一种方法是利用 Σ 的特征值分解。Cholesky 分解方法效率较高，而特征值分解方法较稳健但效率较低。

在 C++ 源程序中，要引入 `RcppArmadillo.h` 头文件，并且用 `Rcpp::depends` 说明链接时对 `RcppArmadillo` 库的依赖。设 C++ 源程序保存在 `rmvnorm.cpp` 文件中，内容如下：

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]

// Sample N x P observations from a Standard
// Multivariate Normal given N observations, a
// vector of P means, and a P x P cov matrix

// [[Rcpp::export]]
arma::mat rmvnorm(
    int n,
    const arma::vec& mu,
    const arma::mat& Sigma) {
  unsigned int p = Sigma.n_cols;

  // First draw N x P values from a N(0,1)
  Rcpp::NumericVector draw = Rcpp::rnorm(n*p);

  // Instantiate an Armadillo matrix with the
  // drawn values using advanced constructor
  // to reuse allocated memory
  arma::mat Z = arma::mat(
    draw.begin(), n, p, false, true);
```

```
  // Simpler, less performant alternative
  // arma::mat Z = Rcpp::as<arma::mat>(draw);

  // Generate a sample from the Transformed
  // Multivariate Normal
  arma::mat Y = arma::repmat(mu, 1, n).t() +
    Z * arma::chol(Sigma);
  return Y;
}
```

注意 C++ 源程序中仅引入了 RcppArmadillo.h 头文件，该头文件会引入 Rcpp.h 头文件，用户不能自己引用 Rcpp.h 头文件。

在 R 中测试：

```
library(Rcpp)
sourceCpp(file = "rmvnorm.cpp")
set.seed(101)
Y <- rmvnorm(50, c(100, 200),
  rbind(c(2,1), c(1,4)))
colMeans(Y)
## [1]  99.82468 200.00511
var(Y)
##          [,1]     [,2]
## [1,] 1.737795 1.061398
## [2,] 1.061398 3.701758
```

49.9.2 快速计算线性回归

R 中的 lm() 函数利用了数值稳健方法进行计算，效率较低，多次重复调用也会影响速度。

这里用 RcppArmadillo 进行较简单的计算，仅返回回归系数、误差项标准差和回归系数标准误差估计。RcppArmadillo 已经提供了 fastLm() 函数，这里给出源程序：

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]

// Compute coefficients and their standard error
// during multiple linear regression given a
// design matrix X containing N observations with
// P regressors and a vector y containing of
```

```
// N responses

// [[Rcpp::export]]
Rcpp::List fastLm(
    const arma::mat& X,
    const arma::colvec& y) {

  // Dimension information
  int n = X.n_rows, p = X.n_cols;

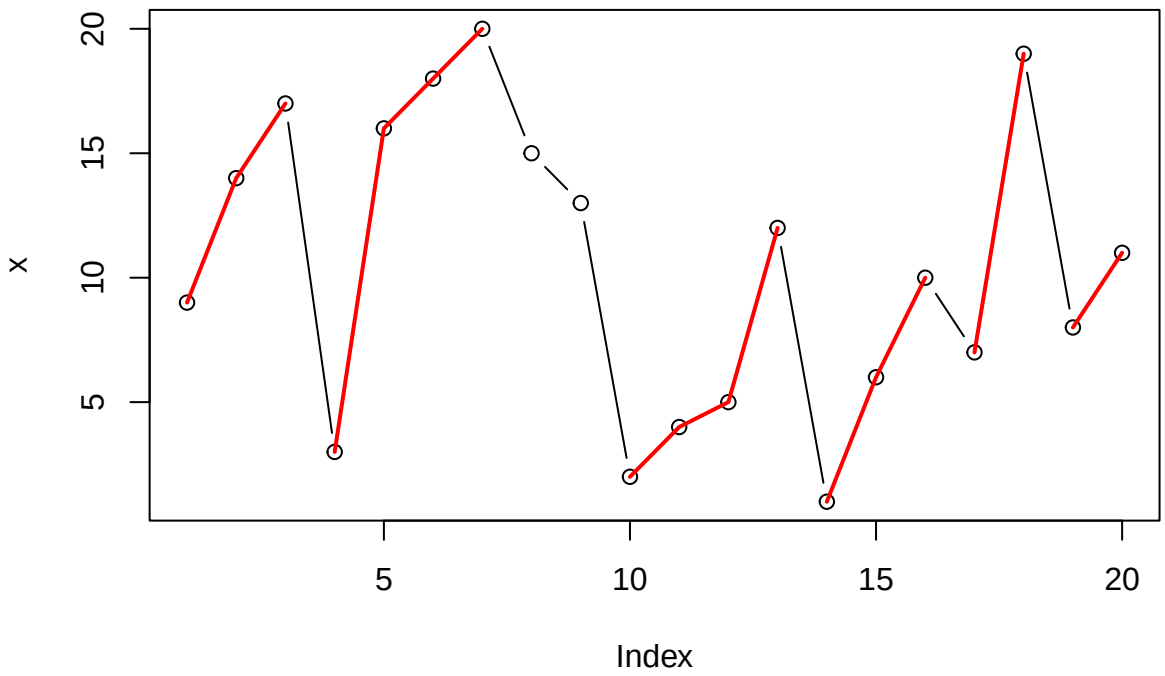
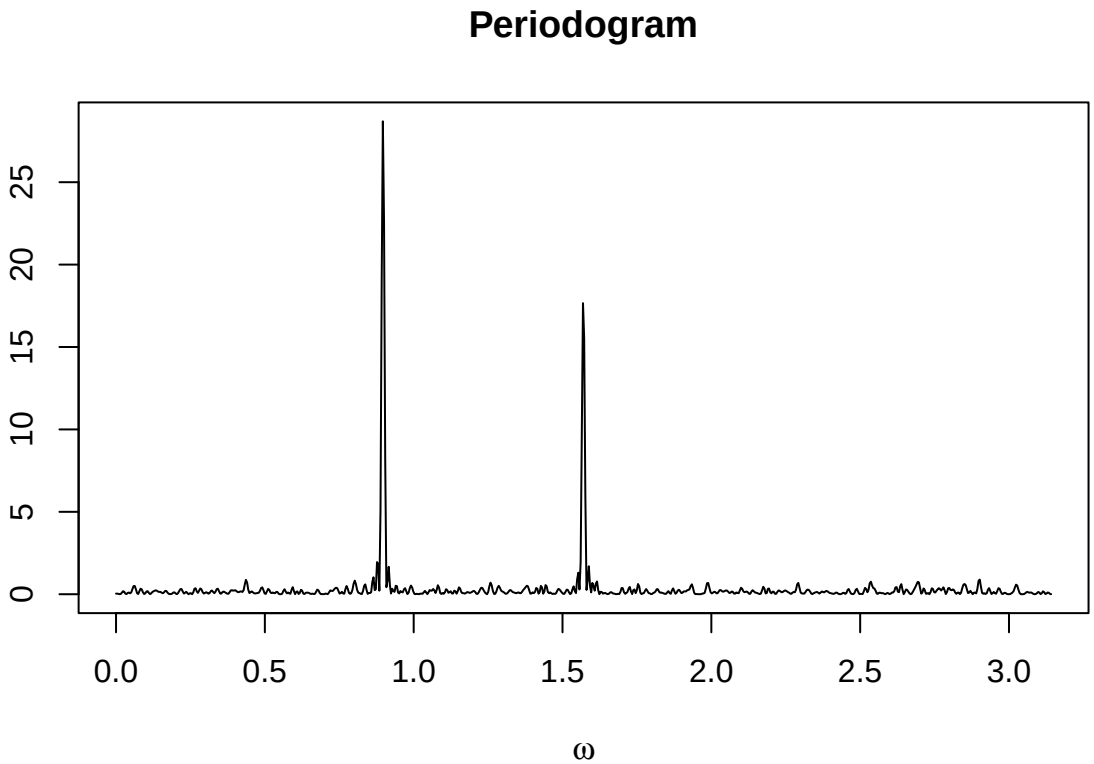
  // Fit model y ~ X
  arma::colvec coef = arma::solve(X, y);

  // Compute the residuals
  arma::colvec res = y - X*coef;

  // Estimated variance of the random error
  double s2 = std::inner_product(
    res.begin(), res.end(),
    res.begin(), 0.0) / (n - p);

  // Standard error matrix of coefficients
  arma::colvec std_err = arma::sqrt(
    s2 * arma::diagvec(arma::pinv(X.t()*X)));

  // Create named list with the above quantities
  return Rcpp::List::create(
    Rcpp::Named("coefficients") = coef,
    Rcpp::Named("stderr") = std_err,
    Rcpp::Named("df.residual") = n - p );
}
```

在 R 中测试（不自行编译，直接调用 RcppArmadillo 中已有函数）：

```
library(RcppArmadillo)
test_fastlm <- function(n=30){
  x <- sample(1:10, size=2*n, replace=TRUE) |>
    matrix(nrow=n, ncol=2)
  x <- cbind(1, x)
  y <- x %*% c(100, 2, 3) + rnorm(n, 0, 0.5)
```

```
  fastLm(x, y)
}
set.seed(101)
res <- test_fastlm()
res$coefficients
## [1] 100.268414   1.984362   2.972958
res$stderr
##             [,1]
## [1,] 0.25189799
## [2,] 0.02958671
## [3,] 0.02621272
res$df.residual
## [1] 27
```

Chapter 50

用 Rcpp 帮助制作 R 扩展包

R 扩展包是把解决某种问题的可复用代码、文档整合在一起的最好的方法。写成 R 扩展包后，可以自己用，也可以利用 CRAN 分发。扩展包用户一般不用自己编译。

使用扩展包来组织程序，多个源程序、头文件之间的依赖关系可以自动得到处理。

扩展包提供了测试、文档和一致性检查的统一框架。

扩展包中代码可以仅有 R 程序，也可以包括 C 程序、C++ 程序、Fortran 程序。如果仅有 R 代码，就不需要借助于 Rcpp，可以使用 `package.skeleton()` 函数生成一个扩展包框架。如果有 C++ 代码，就可以用 Rcpp 作为接口，并用 Rcpp 提供的 `Rcpp.package.skeleton()` 函数制作扩展包框架。

Rcpp 属性的 exports 注释仍可在制作扩展包时指定如何输出 C++ 中定义的函数使其在 R 中可调用。

50.1 不用扩展包共享 C++ 代码的方法

Rcpp 属性的 `sourceCpp()` 通常只适用于写在 R 程序内部的简短 C++ 代码，或者写在一个单独 C++ 文件中，不依赖于其它 C++ 程序的单独代码。如果有多个 C++ 源程序、头文件，彼此有依赖关系，最好使用扩展包。

在多个单独的 C++ 文件共享某些简单的代码，彼此不互相依赖时，可以用 C++ 的预处理 include 命令共享这些代码。

比如，有多个 C++ 源程序都用到如下的代码：

```
#ifndef __UTILITIES__
#define __UTILITIES__
inline double timesTwo(double x) {
  return x * 2;
}
#endif // __UTILITIES__
```

假设这段代码保存到当前子目录的 “utilities.hpp” 文件中。则在每个需要用到这段代码的 C++ 源程序中，插入如:

```
#include "utilities.hpp"
//[[Rcpp::export]]
double transformValue(double x){
  return timesTwo(x) * 10;
}
```

50.2 生成扩展包

50.2.1 利用已有基于 Rcpp 属性的源程序制作扩展包

假设在当前目录中有了若干个 C++ 文件，其中需要转换到 R 中的 C++ 函数已经用 `Rcpp::export` 声明过。其中一个是 conv1.cpp。

从当前目录启动 R，运行

```
Rcpp.package.skeleton("testpack",
  example_code=FALSE,
  attributes=TRUE,
  cpp_files=c("conv1.cpp"))
```

运行显示:

```
Creating directories ...
Creating DESCRIPTION ...
Creating NAMESPACE ...
Creating Read-and-delete-me ...
Saving functions and data ...
Making help files ...
Done.
Further steps are described in './testpack/Read-and-delete-me'.

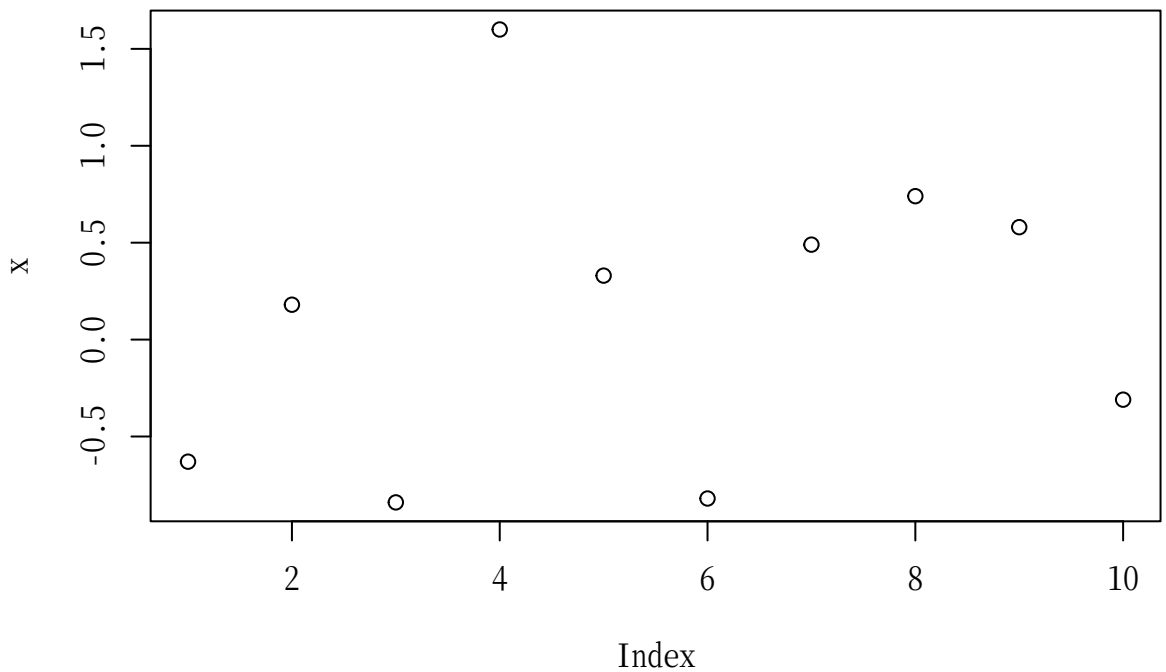
Adding Rcpp settings
 >> added Imports: Rcpp
 >> added LinkingTo: Rcpp
 >> added useDynLib directive to NAMESPACE
 >> added importFrom(Rcpp, evalCpp) directive to NAMESPACE
 >> copied conv1.cpp to src directory
```

运行完后，在当前目录生成了一个 testpack 子目录，这是要制作的扩展包的名字。在 testpack 子目录中，有文件 DESCRIPTION, NAMESPACE, Read-and-delete-me, 有子目录 src, R, man。

子目录 src 中为 C++ 和 C 源程序、头文件。子目录 R 中为 Rcpp 从 C++ 程序转换过来的 R 接口程序，用户自己的 R 程序也可以放在这里。子目录 man 是特殊格式的文档，其格式类似 LaTeX。

50.2.2 DESCRIPTION 文件

在 DESCRIPTION 文件中，有扩展包名称、版本、日期、作者姓名、维护者姓名和联系方式、简单描述、授权，还有 Imports 和 LinkingTo 两项。除此之外，还有许多可选的域，如 Depends。

Imports 给出本软件包要调用的其它扩展包, 但是这些扩展包并不随本扩展包一起调入，仅是会调入其名字空间。这里的值为

```
Imports: Rcpp (>= 0.12.3)
```

LinkingTo 指定在编译本软件包的 C、C++、Fortran 等源程序时，会用到哪些其它扩展包的头文件。这里的值为

```
LinkingTo: Rcpp
```

指定的这些扩展包一般是编译时才有用的，所以一般不会出现在 Depends 和 Imports 域中。

LinkingTo 只解决了头文件的问题，要链接除了 Rcpp 之外的二进制库文件，还需要手工编辑 src/Makevars 和 src/Makevars.win 文件。

和 Imports 有些相像的的 DESCRIPTION 域是 Depends，指定调入本扩展包时必须预先调入的软件包。这里“调入”是指用 `libraray()` 或 `require()` 调入扩展包。多个扩展包名用逗号分开，可以在扩展包名字后面加圆括号，在圆括号内写上 >= 某个版本号, 如“`MASS(>=3.1-20)`”。

Depends 也可以指定依赖于某个 R 版本之后，如“`R(>=2.14.0)`”。

DESCRIPTION 文件中的 Suggests 与和 Depends 域类似，但不是本扩展包必须的，比如仅用在某个例子中或测试中、仅用来编译 vignettes。

50.2.3 NAMESPACE 文件

示例 NAMESPACE 文件如下：

```
useDynLib(testpack)
exportPattern("^[[:alpha:]]+")
importFrom(Rcpp, evalCpp)
```

其中第一行指定调用本软件包时，需要调入的本扩展包的动态链接库。第二行指定扩展包需要对外部可见的 R 函数是所有函数名字以字母开头的 R 函数。用户可以自己指定其它的模式或者指定固定的若干个函数。第三行说明了需要从 Rcpp 包导入 evalCpp 函数。

50.3 重新编译

修改了扩展包中的 C++ 源程序后，需要重新编译。只要在 R 中把工作目录设为软件包的子目录内，运行

```
compileAttributes()
```

这会自动生成两个文件，一个是 `src/RcppExports.cpp`，是 C++ 程序的接口函数。另一个是 `R/RcpExports.R`，用.Call 来调用 C++ 接口函数，转换成 R 函数。这两个文件不要自己修改。

50.4 建立 C++ 用的接口界面

利用了 Rcpp 属性可以指定要输出到 R 中的函数。

在 C++ 源程序中加入特殊注释

```
//[[Rcpp::interfaces(r, cpp)]]
```

则软件包在编译时也会生成该源程序文件中函数的外部可访问的接口，这些接口的界面会在安装后的包的 include 子目录中出现，在开发时出现在 inst/include 子目录中。

设要生成的扩展包名为 testpack, 则界面文件包括 `include` 子目录中的 `testpack_RcppExports.h` 文件和 `testpack.h` 文件，`testpack.h` 文件仅用来包含入 `testpack_RcppExports.h` 文件。

如果需要添加自己的一些界面程序，可以修改 `testpack.h` 文件，这时需要去掉文件开始的自动生成标记，并且保留对 `testpack_RcppExports.h` 文件的包含。

导出的 C++ 界面都在与制作的扩展包同名的名字空间中，比如，如果制作的软件包名为 testpack, 其中导出的一个 C++ 函数为 `convolveCpp`, 则在别的包的 C++ 源程序中调用时，应该包含 `testpack.h` 文件，并用 `testpack::convolveCpp()` 格式调用。

如果自己本扩展包需要在编译时包含这些头文件，需要自己编辑 src 子目录中的 Makevars 文件和 Makevars.win 文件，添加行:

```
PKG_CPPFLAGS += -I../inst/include/
```

Part X

其它

Chapter 51

R 编程例子

因为这些例子要用作学生习题，所以这里只有问题，没有解答。

51.1 R 语言

51.1.1 用向量作逆变换

设向量 `x` 长度为 `n`, 其中保存了 1 到 `n` 的正整数的一个排列。把 `x` 看成是在集合 $\{1, 2, ..., n\}$ 上的一个一一变换，求向量 `y` 使得 `y` 能够表示上述变换的逆变换。即任给长度为 n 的向量 `z`, `z[x]` 表示按照 `x` 的次序重新排列 `z` 的元素，而 `z[x][y]` 则应该恢复为 `z`。

51.1.2 斐波那契数列计算

设数列 $x_0 = 0, x_1 = 1$，后续值按如下公式递推计算：

$$x_n = x_{n-2} + x_{n-1},\ n = 2, 3, ...$$

这样的数列叫做斐波那契数列。希望编写 R 函数，输入 n，返回计算的 x_n 的值。

51.1.3 穷举所有排列

设向量 `x` 的各个元素为某个集合的元素。想要列出 `x` 的元素的所有不同排列。比如，如果 `x = 1:3`, 所有排列为

```
1 2 3
1 3 2
2 1 3
2 3 1
```

```
3 1 2
3 2 1
```

共 $3! = 6$ 种不同的排列。

51.1.4 可重复分组方式穷举

设有 n 个编号卡片，分别有号码 $1, 2, \ldots, n$。从中有放回地抽取 m 个并记录每次的号码，穷举 m 个号码中多少个 1，多少个 2，..., 多少个 n 这样的结果。

例如，有 3 个编号卡片，随机有放回第抽取 2 次。用 (x_1, x_2, x_3) 表示每一种个数组合，x_1 表示 2 次抽取中号码 1 的个数，x_2 表示 2 次抽取中号码 2 的个数，x_3 表示 2 次抽取中号码 3 的个数。问题就是列出 (x_1, x_2, x_3) 的所有不同值。

51.1.5 升降连计数

51.1.5.1 问题 1

考虑如下的数据：

```
x <- c(
 9, 14, 17,      # (1)
 3, 16, 18, 20, # (2)
15, 13,          # 降连
 2,  4,  5, 12, # (3)
 1,  6, 10,      # (4)
 7, 19,          # (5)
 8, 11)          # (6)
```

x 的序列中连续上升的一段 (至少两个) 称为一个升连，共 6 个升连。这可以用绘图更形象地表现：

```
plot_runs_prob1 <- function(x){
  plot(x, type="b", lwd=1)
  segs <- matrix(c(
    1,3,  4,7,  10,13,  14,16,  17,18,  19,20),
    ncol=2, byrow=TRUE)
  for(i in seq(nrow(segs))) {
    lines(seq(segs[i,1], segs[i,2]),
          x[seq(segs[i,1], segs[i,2])],
          col="red", lwd=2)
  }
```

```
}
plot_runs_prob1(x)
```



如何用编程方式找到段数？

要求编写 R 函数，输入一个序列，设序列中的数值没有重复数值，输出升连的个数。

最后的程序需要能解决如下问题：

```
n <- 10000
set.seed(102)
x <- sample(n)
```

升连个数 =?

51.1.5.2 问题 2

如果输入的序列中有重复数值，将一个升连定义为：序列中连续的一段，长度至少有两个数，后一个数大于等于前一个数，两个升连之间是下降的。

例如：

```
x <- c(
  9, 14, 17, # (1)
  3, 3, 9,   # (2)
  3, 3,      # (3)
  2, 20,     # (4)
  17, 17,    # (5)
  12, 10,   # 下降
  1, 5, 5, 16, # (6)
  10, 11)    # (7)
```

编写函数，输入这样的序列，输出一个两列的矩阵，每一行是一个升连的开始序号和结束序号。比如，上述数据的输出应为：

```
     start end
[1,]     1   3
[2,]     4   6
[3,]     7   8
[4,]     9  10
[5,]    11  12
[6,]    15  18
[7,]    19  20
```

更多测试数据生成的程序例子：

```
set.seed(101)
x <- sample(1:5, size = 50, replace=TRUE)
```

51.1.6 高斯八皇后问题

国际象棋的棋盘是 8×8 的格子。国际象棋手马克斯·贝瑟尔于 1848 年提出如下问题：在 8×8 格的国际象棋上摆放八个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，问有多少种摆法。

高斯认为有 76 种方案。1854 年在柏林的象棋杂志上不同的作者发表了 40 种不同的解，后来有人用图论的方法解出 92 种结果。

任务：

- 编程穷举所有的不同摆法，显示结果。
- 一个摆法通过旋转 90 度、180 度、270 度，左右对调、上下对调以及这些变换的组合，可以转换为另一种摆法。按是否可以互相转换将所有摆法分成不同的组，显示这些组。
- 用图形显示每一种摆法，并分组显示。
- 将问题推广到 9x9 的格子摆 9 个皇后的问题。

51.1.7 最小能量路径

动态规划是最优化问题的一种算法。最优化涉及最优决策问题，此决策问题经常可以分解为若干步骤，要解决许多子问题，而这些子问题往往是重复的，记住已经解决的子问题，会使得决策问题大大简化，这是动态规划问题的主要思想。

下面的例子来自MIT 计算思想公开课课件。

考虑一个 $m \times n$ 的矩阵，如：

$$M = \begin{pmatrix} 7 & 7 & 9 & 9 & 7 & 1 \\ 1 & 3 & 7 & 3 & 3 & 2 \\ 3 & 6 & 5 & 6 & 1 & 8 \\ 5 & 1 & 1 & 7 & 6 & 6 \\ 9 & 2 & 5 & 3 & 9 & 6 \end{pmatrix}$$

考虑从第一行某个元素出发到最后一行某个元素的路径，每条路径从第一行的某个元素出发，然后进入下一行的正下方或正下方的左侧元素或右侧元素，直到进入最后一行的元素为止，每条路径有 m 个元素，计算路径上元素总和。将 M 的 (i, j) 元素看作是该位置的“能量”，试图找到从顶层（第一行）到底层（最后一行）的“最小能量”路径。比如，上面的例子从第一行出发，依次经过的列号为

$$6, 5, 4, 3, 2,$$

最小能量值为 13。

51.2 概率

51.2.1 智者千虑必有一失

成语说：“智者千虑，必有一失；愚者千虑，必有一得”。设智者作判断的准确率为 $p_1 = 0.99$, 愚者作判断的准确率为 $p_2 = 0.01$，计算智者做 1000 次独立的判断至少犯一次错误的概率，与愚者做 1000 次独立判断至少对一次的概率。

51.2.2 圆桌夫妇座位问题

在一张圆桌上用餐时，n 对夫妇随机入座。计算没有任何一位妻子和她的丈夫相邻的概率。通过推导可得此概率为

$$p_n = 1 + \sum_{k=1}^{n} (-1)^k C_n^k \frac{(2n-k-1)!2^k}{(2n-1)!}. \qquad (1)$$

例如 $p_2 = 1/3$, $p_3 = 4/15 = 0.2667$, $p_4 = 0.2952$, $p_5 = 0.3101$。

分别用上面的理论公式以及直接穷举验证的方法，对 $n = 2, 3, 4, 5$ 的情形进行验证。

51.3 科学计算

51.3.1 城市间最短路径

假设有 n 个城市，编号为 $1, 2, \ldots, n$。已知其中的部分城市之间有高速公路连通，每对连通城市记为 (F_i, T_i), 其中 $F_i, T_i \in \{1, 2, \ldots, n\}$ 且 $F_i < T_i$, $i = 1, 2, \ldots, m$。除了这些直接连通的城市以外，其它的任意两个城市只能途经别的城市连通，或者根本不能靠高速公路连通。用一个 $m \times 2$ 的 R 矩阵 M 可以输入这些连通情况，矩阵的每行是一对 (F_i, T_i) 值。

要求编写一个 R 函数，输入直接连通情况 M 后，输出一个 $n \times n$ 矩阵 R, R[i,i]=0, R[i,j]=1 表示直接相连，R[i,j]=k ($k \geq 2$) 表示城市 i 与城市 j 至少需要经过 k 段高速公路连通，R[i,j]=Inf 表示城市 i 与城市 j 不能靠高速公路连通。R 的元素值仅考虑途经的高速公路段数而不考虑具体里程。如果从一个城市通过高速公路移动到直接相连城市叫做移动一步，R 的 (i, j) 元素是从第 i 城市通过高速公路到第 j 城市需要移动的步数。

这个问题也可以作为 “相识” 问题的模型。设 n 个人中有些人是直接相识的，如果两个不相识的人想认识，假设必须经过相识的人引荐，问最少需要多少个引荐人。

本问题必须使用循环，很难向量化，属于 R 比较不擅长的问题。可以考虑使用 Rcpp 把程序用 C++ 语言实现，可以大大加速。当然，如果这个程序仅需要执行不多的次数，用 R 就足够了。

51.3.2 Daubechies 小波函数计算

这个例子主要展示了不用每次计算函数值而是尽可能从已经计算并储存的函数值中查找的技巧。程序中用了比较多的循环，如果有需要，可考虑用 Rcpp 转换成 C++ 代码以提高效率。

小波是重要数学工具, 在图像处理、信号处理等方面有广泛应用。小波中一个重要的函数叫做尺度函数 (scale function), 它满足所谓双尺度方程：

$$\phi(x) = \sqrt{2} \sum_k h_k \phi(2x - k)$$

一种特殊的尺度函数是只在有限区间上非零的, 叫做紧支集的。紧支集尺度函数可以在给定 $\{h_k\}$ 后用以下迭代公式生成：

$$\begin{aligned} \eta_0(x) =& I_{[-0.5, 0.5]}(x) \\ \eta_{n+1}(x) =& \sqrt{2} \sum_{k=0}^{2N-1} h_k \eta_n(2x - k) \end{aligned}$$

其中 N 是正整数, N=2 时 h_0=0.482962913145, h_1=0.836516303738, h_2=0.224143868042, $h_3 = -0.129409522551$。已知 $\phi(x)$ 的支集 (不为零的区间) 为 $[0, 2N-1]$, $\eta_n(x)$ 的支集包含于 $[-0.5, 2N-1]$ 中。

任务：编写计算 $\phi(x)$ 的 R 程序，通过 20 次迭代计算，输出 $\phi(x)$ 在 $[0, 2N-1]$ 区间的 256 个等间隔点上的函数值并作图。在迭代过程中，应不重新结算函数格子点的值，仅计算新加入的格子点的值。不要利用递归函数，递归函数需要每次重新计算已经计算过的函数值。

51.3.3 房间加热温度变化

某个房屋带有天花板，天花板与屋顶之间有一定的空间。用壁炉保持房间温度。为了研究房间内与天花板上方的温度变化，建立了如下的微分方程组：

$$\begin{aligned}\frac{dx_1}{dt} =&0.35\left(-9.7\sin\frac{(t+3)\pi}{12}+8.3-x_1(t)\right)+0.46(x_2(t)-x_1(t))+11.1\\ \frac{dx_2}{dt} =&0.28\left(-9.7\sin\frac{(t+3)\pi}{12}+8.3-x_2(t)\right)+0.46(x_1(t)-x_2(t))\end{aligned}$$

其中 $x_1(t)$ 是房间在 t 时刻的温度（单位：摄氏度），$x_2(t)$ 是天花板上方在 t 时刻的温度，t 是单位为小时的时间。设 $t=0$ 时 $x_1(t)=x_2(t)=4$。

（1）用每一秒钟重新计算的方法计算 24 小时内的房间温度与天花板上温度，绘制求解出的这两个温度的时间序列曲线图。

（2）计算 7 天的温度，作图，查看周期性。当温度循环变化差距小于 0.1 度时认为开始周期变化了。

51.4 统计计算

51.4.1 线性回归实例

1. 用 readr 包的 `read_csv` 函数读入 class.csv 为 tibble 类型的数据框，命名为 `da`。
2. 将其中的 weight 列取出，保存为变量 `y`。
3. 将数据中的 age 和 height 两列取出，并在左侧添加一列全是 1 的列，构成一个 3 列的矩阵，保存为变量 `X`。
4. 用如下的公式计算以体重为因变量，以年龄和身高为自变量的线性回归模型系数：

   $$\hat{\beta}=(X^TX)^{-1}X^Ty$$

   其中 y 是包含了因变量观测值的列向量。
5. 用如下公式计算残差的方差估计：

   $$e=y-X\hat{\beta},\quad \hat{\sigma}^2=\frac{1}{n-3}\sum_{i=1}^{n}\hat{e}_i^2.$$

   其中 n 是样本量，e_i 是向量 e 的第 i 个分量。
6. 显示回归系数和 $\hat{\sigma}^2$ 的值。
7. 运行如下的 R 回归程序，与自己计算的结果比较：

```
lm(weight ~ age + height, data=da) |> summary()
```

51.4.2 核回归与核密度估计

考虑核回归问题。核回归是非参数回归的一种, 假设变量 Y 与变量 X 之间的关系为：

$$Y=f(X)+\varepsilon$$

其中函数 f 未知。观测到 X 和 Y 的一组样本 X_i, Y_i, i=1,...,n 后, 对 f 的一种估计为：

$$\hat{f}(x) = \frac{\sum_{i=1}^{n} K\left(\frac{x-X_i}{h}\right) Y_i}{\sum_{i=1}^{n} K\left(\frac{x-X_i}{h}\right)}$$

其中 $h > 0$ 称为窗宽，窗宽越大，得到的密度估计越平滑。K 叫做核函数, 一般是一个非负的偶函数, 原点处的函数值最大, 在两侧迅速趋于零。例如正态密度函数, 或所谓双三次函数核：

$$K(x) = \begin{cases} \left(1-|x|^3\right)^3 & |x| \le 1 \\ 0 & \text{其它} \end{cases}$$

与核回归类似，可以用核平滑方法估计总体分布密度。设样本为 $Y_1, Y_2, \ldots, Y_n$, 密度估计公式为

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h} K\left(\frac{x-Y_i}{h}\right)$$

其中 $K(x)$ 满足 $\int_{-\infty}^{\infty} K(x)\,dx = 1$。$h$ 是窗宽，窗宽越大，估计的曲线越光滑。h 的一种建议公式为 $h = 1.06Sn^{-1/5}$, S 为样本标准差。

对以上两个问题进行编程，其中窗宽 h 由用户输入。

51.4.3 二维随机模拟积分

设二元函数 $f(x,y)$ 定义如下

$$\begin{aligned} & f(x,y) \\ =& \exp\left\{-45(x+0.4)^2 - 60(y-0.5)^2\right\} \\ & + 0.5\exp\left\{-90(x-0.5)^2 - 45(y+0.1)^4\right\} \end{aligned}$$

(注意其中有一个 4 次方。) 求如下二重定积分

$$I = \int_{-1}^{1}\int_{-1}^{1} f(x,y)\,dx\,dy$$

$f(x,y)$ 有两个分别以 $(-0.4, 0.5)$ 和 $(0.5, -0.1)$ 为中心的峰，对积分有贡献的区域主要集中在 $(-0.4, 0.5)$ 和 $(0.5, -0.1)$ 附近，在其他地方函数值很小，对积分贡献很小。可以用随机模拟方法估计 I 的值。

第一种估计方法是平均值法。设 $X_i, i = 1, 2, \ldots, N$ 是均匀分布 $U(-1,1)$ 的随机数，$Y_i, i = 1, 2, \ldots, N$ 也是均匀分布 $U(-1,1)$ 的随机数，两者独立，可估计 I 为

$$\hat{I}_1 = \frac{4}{N}\sum_{i=1}^{N} f(X_i, Y_i).$$

第二种估计方法是重要抽样法。设正态分布 $\mathrm{N}(\mu, \sigma^2)$ 的密度记为 $p(x; \mu, \sigma^2)$，令

$$\begin{aligned} & g(x,y) \\ =& 0.5358984 p(x; -0.4, 90^{-1}) p(y; 0.5, 120^{-1}) \\ & + 0.4641016 p(x; 0.5, 180^{-1}) p(y; -0.1, 20^{-1}), \\ & -\infty < x < \infty, -\infty < y < \infty, \end{aligned}$$

这是一个二元随机向量的密度，是两个二元正态密度的混合分布。设 $K_i, i=1,2,\dots,N$ 是取值于 $\{1,2\}$ 的随机数，$P(K_i=1)=0.5358984$, $P(K_i=2)=1-P(K_i=1)$。当 $K_i=1$ 时，取 X_i 为 $\mathrm{N}(-0.4,90^{-1})$ 随机数，Y_i 为 $\mathrm{N}(0.5,120^{-1})$ 随机数；当 $K_i=2$ 时，取 X_i 为 $\mathrm{N}(0.5,180^{-1})$ 随机数，Y_i 为 $\mathrm{N}(-0.1,20^{-1})$ 随机数，这样得到的 $(X_i,Y_i), i=1,2,\dots,N$ 是 $g(x,y)$ 的随机数。令

$$\hat{I}_2=\frac{1}{N}\sum_{i=1}^{n}\frac{f(X_i,Y_i)}{g(X_i,Y_i)},$$

称为 I 的重要抽样法估计。

51.4.3.1 编程任务

1. 编写 R 函数估计计算 $\hat{I}_1$。重复模拟 $B=100$ 次，得到 $\hat{I}_1$ 的 B 个值，计算这些值的平均值和标准差。

2. 编写 R 函数估计计算 $\hat{I}_2$。重复模拟 $B=100$ 次，得到 $\hat{I}_2$ 的 B 个值，计算这些值的平均值和标准差。

3. 比较模拟得到的平均值和标准差，验证两者是否基本一致，通过标准差大小比较两种方法的精度。

注意尽量用向量化编程。

51.4.4 潜周期估计

设时间序列 $\{y_t\}$ 有如下模型:

$$y_t=\sum_{k=1}^{m}A_k\cos(\lambda_k t+\phi_k)+x_t,\ t=1,2,\dots$$

其中 x_t 为线性平稳时间序列，$\lambda_k\in(0,\pi)$, $k=1,2,\dots,m$。这样的模型称为潜周期模型。如果有 $\{y_t\}$ 的一组样本 $y_1,y_2,\dots,y_n$, 可以定义周期图函数

$$P(\omega)=\frac{1}{2\pi n}\left|\sum_{t=1}^{n}y_t e^{-it\omega}\right|^2,\ \omega\in[0,\pi].$$

这里 ω 是角频率。对于潜周期数据，在 λ_j 的对应位置 $P(\omega)$ 会有尖峰，而且当 $n\to\infty$ 时尖峰高度趋于无穷。下面是一个样例图形。



如下算法可以在 n 较大时估计 m 和 $\{\lambda_k\}$：首先，对 $\omega_j = \pi j/n$, $j = 1, 2, \dots, n$ 计算 $h_j = P(\omega_j)$，求 $\{h_j, j = 1, 2, \dots, n\}$ 的 3/4 分位数记为 q。令 $C = qn^{0.25}$，以 C 作为分界线，设 $\{h_j\}$ 中大于 C 的下标 j 的集合为 J, 当 J 非空时，把 J 中相邻点分入一组，但是当两个下标的差大于等于 $n^{0.6}$ 时就把后一个点归入新的一组。在每组中，以该组的 h_j 的最大值点对应的角频率 $j\pi/n$ 作为潜频率 $\{\lambda_k\}$ 中的一个的估计。

用如下 R 程序可以模拟生成一组 $\{y_t\}$ 的观测数据:

```
set.seed(1); n <- 500; tt <- seq(n)
m <- 2; lam <- 2*pi/c(4, 7); A <- c(1, 1.2)
y <- A[1]*cos(lam[1]*tt) + A[2]*cos(lam[2]*tt) + rnorm(n)
```

编写 R 程序:

(1) 编写计算 $P(\omega)$ 的函数，输入 $y = (y_1, y_2, \dots, y_n)^T$ 和 $\omega = (\omega_1, \omega_2, \dots, \omega_s)^T$, 输出 $(P(\omega_1), P(\omega_2), \dots, P(\omega_s))$。

(2) 对输入的时间序列样本 $y_1, y_2, \dots, y_n$，编写函数用以上描述的算法估计 m 和 $\{\lambda_j, j = 1, 2, \dots, m\}$。

(3) 用上述模拟数据测试编写的算法程序。

(4) 进一步地，用 R 函数 `fft()` 计算 $h_j = P(\pi j/n), j = 1, 2, \dots, n$。

(5) 把整个算法用 Rcpp 和 C++ 程序实现。

51.4.5 ARMA(1,1) 模型估计

考虑如下的零均值高斯 ARMA(1,1) 模型：

$$X_t = aX_{t-1} + e_t + be_{t-1}, t = 1, 2, \dots, T$$

其中 $0 < |a| < 1$, $0 < |b| < 1$，$-a \neq b$, $\{e_t\}$ 独立同 $\mathrm{N}(0, \sigma_e^2)$ 分布。

(1) 给定观测 $x_1, x_2, \dots, x_T$，写出在 $x_0 = 0$ 且 $e_0 = 0$ 条件下的条件对数似然函数。
(2) 编写 R 程序，用条件最大似然估计方法估计参数 a, b, σ_e^2。可使用数值优化程序如 `nlm()` 或 `optim()`。
(3) 对 $a = 0.5$, $b = 0.7$, $\sigma_e = 1$，$T = 100$, 模拟生成 x_t 的样本并重复模拟 $B = 1000$ 次，据此评估 $\hat{a}, \hat{b}, \hat{\sigma}_e$ 的估计精度。模拟可使用 R 的 `arima.sim()` 函数。

若 x_0, e_0 已知，从 $x_1, \dots, x_T$ 可递推地计算

$$e_t = x_t - ax_{t-1} - be_{t-1},\ t = 1, 2, \dots, T$$

记 $\theta = (a, b, \sigma_e, x_0, e_0)^T$，有 $X_t | x_{t-1}, \dots, x_1, \theta$ 服从 $\mathrm{N}(ax_{t-1} + be_{t-1}, \sigma_e^2)$ 条件分布。于是用联合密度的乘积公式可得

$$\begin{aligned}
f(x_1, \dots, x_T | \theta) =& f(x_1|\theta) f(x_2|x_1, \theta) \dots f(x_T | X_{T-1}, \dots, x_1, \theta) \\
=& \prod_{t=1}^{T} \mathrm{dnorm}(x_t, ax_{t-1} + be_{t-1}, \sigma_e) \\
=& \prod_{t=1}^{T} (2\pi)^{-1/2} \sigma_e^{-1} \exp\left\{ -\frac{1}{2} \frac{1}{\sigma_e^2} (x_t - ax_{t-1} - be_{t-1})^2 \right\} \\
=& (2\pi)^{-T/2} \sigma_e^{-T} \exp\left\{ -\frac{1}{2} \frac{1}{\sigma_e^2} \sum_{t=1}^{T} e_t^2 \right\}
\end{aligned}$$

其中 $\mathrm{dnorm}(x, \mu, \sigma)$ 表示 $\mathrm{N}(\mu, \sigma^2)$ 的分布密度。

取 $x_0 = 0, e_0 = 0$，给定 a, b 后递推计算 $\{e_t\}$ 序列，对数似然函数为

$$l(a, b, \sigma_e) = -T \ln \sigma_e - \frac{1}{2} \frac{1}{\sigma_e^2} \sum_{t=1}^{T} e_t^2$$

注意其中的 $\{e_t\}$ 也与待估参数 a, b 有关。

可以用数值优化算法估计求如上的问题的最大值点。显然 a, b 的最大值点不依赖于 σ_e 的取值，所以可以先求 $\sum_{t=1}^{T} e_t^2$ 的最小值点。这基本是一个最小二乘估计问题，但是 e_{t-1} 也依赖于 a, b 的值所以不能直接用线性最小二乘求解。

51.4.6 VAR 模型平稳性

称 k 元时间序列 r_t 服从一个 VAR(p) 模型，如果

$$r_t = \phi_0 + \Phi_1 r_{t-1} + \dots + \Phi_p r_{t-p} + a_t \tag{51.1}$$

其中 ϕ_0 和 $\{a_t\}$ 同 VAR(1) 的规定，Φ_j 是 k 阶方阵（$k=1,2,\dots,p$）。利用向后推移算子（滞后算子）B 可以将模型写成

$$(I-\Phi_1 B-\cdots-\Phi_p B^p)r_t=\phi_0+a_t$$

记

$$P(z)=I-\Phi_1 z-\cdots-\Phi_p z^p \tag{51.2}$$

这是一个从复数 z 到 k 阶方阵 $P(z)$ 的变换，$P(z)$ 的每个元素是关于 z 的阶数不超过 p 的多项式。称一元多项式函数 $\det(P(z))$（或记为 $|P(z)|$）为模型的（逆序）特征多项式。

如果 $|P(z)|\neq 0,\ \forall|z|<1$（$z$ 为复数），则模型(51.1)是平稳的。

编写 R 函数，输入三维（三个下标）的系数矩阵数组 `arrcoef`, 返回特征多项式的所有复根。`arrcoef[,,1]` 保存 Φ_1, `arrcoef[,,2]` 保存 Φ_2, 等等。

作为例子，考虑如下的三个模型，对每个计算特征多项式的复根并判断是否平稳：

$$\begin{pmatrix} r_{1t} \\ r_{2t} \end{pmatrix}=\begin{pmatrix} 0.2 & 0.3 \\ -0.6 & 1.1 \end{pmatrix}\begin{pmatrix} r_{1,t-1} \\ r_{2,t-1} \end{pmatrix}+\begin{pmatrix} a_{1t} \\ a_{2t} \end{pmatrix}$$

$$\begin{pmatrix} x_{1t} \\ x_{2t} \end{pmatrix}=\begin{pmatrix} 0.5 & -1.0 \\ -0.25 & 0.5 \end{pmatrix}\begin{pmatrix} x_{1,t-1} \\ x_{2,t-1} \end{pmatrix}+\begin{pmatrix} a_{1t} \\ a_{2t} \end{pmatrix}$$

$$r_t=\begin{pmatrix} 0.39 & 0.10 & 0.05 \\ 0.35 & 0.34 & 0.47 \\ 0.49 & 0.24 & 0.24 \end{pmatrix} r_{t-1}+\begin{pmatrix} 0.06 & 0.11 & 0.02 \\ -0.19 & -0.18 & -0.01 \\ -0.31 & -0.13 & 0.09 \end{pmatrix} r_{t-2}+a_t$$

51.4.7 贮存可靠性评估

设某种设备的贮存寿命为随机变量 X, 服从指数分布 $\mathrm{Exp}(\theta)$, $EX=\theta>0$。假设有 n 台此种设备分别在时间 $t_1,t_2,\dots,t_n$ 进行了试验，第 i 台试验成功用 $\delta_i=1$ 表示，试验失效用 $\delta_i=0$ 表示。把 n 次试验的结果写成

$$Z_n=\begin{pmatrix} t_1 & \delta_1 \\ t_2 & \delta_2 \\ \vdots & \vdots \\ t_n & \delta_n \end{pmatrix} \tag{1}$$

这里 $\delta_1,\delta_2,\dots,\delta_n$ 认为是随机变量，$t_1,t_2,\dots,t_n$ 是固定的时间。δ_i 服从两点分布 $\mathrm{B}(1,\exp(-t_i/\theta))$。$Z_n$ 取某个特定组合的概率为

$$P(Z_n)=\prod_{i=1}^{n}\left\{[\exp(-t_i/\theta)]^{\delta_i}\,[1-\exp(-t_i/\theta)]^{1-\delta_i}\right\}. \tag{2}$$

要利用观测数据 Z_n 估计 θ 的置信度为 $1-\alpha$ 的置信下限，可以用陈家鼎《生存分析与可靠性》中的样本空间排序法。注意到 Z_n 中每个 δ_i 可以取 1 或 0，所以 Z_n 的所有不同取值共有 2^n 个，记这些所有不同取值为 $\mathcal{D}$，在 $\mathcal{D}$ 中定义如下的序：设 Z_n' 与 Z_n 都是 $\mathcal{D}$ 中的试验结果，称 Z_n' 不次于 Z_n, 并记作 $Z_n'\gtrsim Z_n$，如果如下两个条件之一成立：

(1) $\sum_{i=1}^n \delta_i' > \sum_{i=1}^n \delta_i$;
(2) $\sum_{i=1}^n \delta_i' = \sum_{i=1}^n \delta_i$, 但 $\sum_{i=1}^n \delta_i' t_i \geq \sum_{i=1}^n \delta_i t_i$.

记

$$G(\theta) = \sum_{Z_n' \gtrsim Z_n} P_\theta(Z_n') = \sum_{Z_n' \gtrsim Z_n} \prod_{i=1}^n \left\{ [\exp(-t_i/\theta)]^{\delta_i'} [1-\exp(-t_i/\theta)]^{1-\delta_i'} \right\}. \tag{3}$$

这是 θ 的严格单调增函数，求解如下的方程

$$G(\theta) - \alpha = 0 \tag{4}$$

得到解 $\underline{\theta}$ 是 θ 的置信度为 $1-\alpha$ 的置信下限。

当所有 n 次试验都失效时，恒有 $G(\theta) = 1$, 方程 (4) 无解，取 $\underline{\theta} = 0$。

当 n 次试验都没有失效，即失效数 $n - \sum_{i=1}^n \delta_i = 0$ 时, 方程为

$$\exp\left(-\frac{1}{\theta}\sum_{i=1}^n t_i\right) - \alpha = 0,$$

解得

$$\underline{\theta} = \frac{\sum_{i=1}^n t_i}{\ln \frac{1}{\alpha}}.$$

当失效数为 1 时，最多仅有 n 个结果不次于 Z_n，很容易可以计算 $G(\theta)$ 的值。在 n 较大而且 Z_n 中失效数较多时，$G(\theta)$ 的求和 (3) 中项数很多，计算量很大。当 $n = 10$ 时，$\mathcal{D}$ 约有一千项，当 $n = 20$ 时，$\mathcal{D}$ 约有一百万项，还在可以穷举计算的范围内。但是，当 $n \geq 30$ 时，$\mathcal{D}$ 就有 10 亿项，每计算一次 $G(\theta)$ 都需要很长时间。

针对 n 不太大的情形，可以用精确的公式 (3) 计算 $G(\theta)$ 并用 (4) 求解 $\underline{\theta}$，编写这个问题的纯 R 程序版本，输入一组 Z_n 值后（包括试验时间和试验结果），输出用 (3) 和 (4) 求解 $\underline{\theta}$ 得到的置信下限 $\underline{\theta}$ 的值。在文件`store-reliab-data.csv`中已经生成了 10 组模拟数据，对这 10 组模拟数据计算相应的 $\underline{\theta}$ 的值。在此数据文件中，testid 相同的行属于同一次试验的不同设备的时间和结果。

这个程序中用到比较多的循环，考虑把主要计算部分用 Rcpp 包和 C++ 代码实现，看能够提高效率多少倍。

另一种计算 $G(\theta)$ 的方法是用随机模拟方法估计 $G(\theta)$ 的值然后求解 (4)。随机模拟方法计算简单而且不受失效个数多少的影响，但是有随机误差，在求解 (4) 要考虑到随机误差的影响。随机模拟方法如下。取模拟次数 N, 对给定 θ，为了计算 $G(\theta)$，模拟生成 N 组独立的寿命 $(X_1^{(i)}, X_2^{(i)}, \ldots, X_n^{(i)})$, $i = 1, 2, \ldots, N$，其中每个 $X_j^{(i)}$ 服从 $\mathrm{Exp}(\theta)$ 分布，各分量相互独立。计算 $\delta_j^{(i)} = I_{\{X_j^{(i)} > t_j\}}$, 记

$$Z_n^{(i)} = \begin{pmatrix} t_1 & \delta_1^{(i)} \\ t_2 & \delta_2^{(i)} \\ \vdots & \vdots \\ t_n & \delta_n^{(i)} \end{pmatrix}$$

用

$$\frac{1}{N}\sum_{i=1}^N I_{\{Z_n^{(i)} \gtrsim Z_n\}}$$

来估计 $G(\theta)$。

51.5 数据处理

51.5.1 小题分题型分数汇总

考虑中学某科的一次考试，考卷各小题小题情况汇总在如下的用逗号符分隔的文本文件subscore-subtype.csv中:

```
序号,题型
1,选择题
2,选择题
3,选择题
4,选择题
5,选择题
6,选择题
7,选择题
8,选择题
9,选择题
10,选择题
11,简答题
12,简答题
13,填空题
14,简答题
15,简答题
16,简答题
17,简答题
18,写作
```

文件的中文编码为 GB18030。读入此数据为 R 数据框 dm。两列都读为字符型。

结果显示如下:

```
knitr::kable(dm)
```

序号	题型
1	选择题
2	选择题
3	选择题
4	选择题
5	选择题
6	选择题
7	选择题
8	选择题
9	选择题
10	选择题
11	简答题
12	简答题
13	填空题
14	简答题
15	简答题
16	简答题
17	简答题
18	写作

设部分学生的小题分录入到如下的用逗号分隔的文本文件 `subscore-subscore.csv`中:

```
学号,Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8,Y9,Y10,Y11,Y12,Y13,Y14,Y15,Y16,Y17,Y18
1138010104,3,3,3,3,0,3,7.5,2.5,3.5,6,5,4,2.5,5.5,0,4,2,45.5
1138010108,3,0,3,0,3,0,6,3,4,4,2,5,2,5,6,4,2,48.5
1138010114,3,3,3,3,0,3,5.5,3.5,4,6,2,5,2,5.5,6,1,2,44.5
1138010128,3,3,3,0,0,0,8.5,3.5,2.5,4,5,5,0,4.5,6,4,2,45
1138010126,3,3,3,3,3,3,7,0,4,6,5,5,2.5,4.5,6,4.5,2,44
```

文件的中文编码为 GB18030。读入小题分数据为 R 数据框 ds，分数读为数值型，学号读为字符型。

结果显示如下:

```
knitr::kable(ds)
```

学号	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11	Y12	Y13	Y14	Y15	Y16	Y17	Y
1138010104	3	3	3	3	0	3	7.5	2.5	3.5	6	5	4	2.5	5.5	0	4.0	2	45
1138010108	3	0	3	0	3	0	6.0	3.0	4.0	4	2	5	2.0	5.0	6	4.0	2	48
1138010114	3	3	3	3	0	3	5.5	3.5	4.0	6	2	5	2.0	5.5	6	1.0	2	44
1138010128	3	3	3	0	0	0	8.5	3.5	2.5	4	5	5	0.0	4.5	6	4.0	2	45
1138010126	3	3	3	3	3	3	7.0	0.0	4.0	6	5	5	2.5	4.5	6	4.5	2	44

在数据框 dm 中有每个小题的题型信息，在数据框 ds 中有每个学生的每个小题的分数。从这两个数据框，汇总计算每个学生的题型分，即每个学生选择题共考多少分，简答题共考多少分，等等。

要注意的是，最终的程序应该写成一个函数，其中的计算不依赖于具体的题型名称、小题个数、小题与题型如何对应，只要输入 dm 和 ds 两个数据框就可以进行统计。

如果没有这样的通用性要求，这个问题就可以这样简单解决：

```
resm <- tibble(
  '学号'=ds[['学号']],
  '选择题'= rowSums(ds[, paste('Y', 1:10, sep='')]),
  '简答题'=rowSums(ds[,paste('Y', c(11,12,14:17), sep='' )]),
  '填空题'=ds[['Y13']],
  '作文'=ds[['Y18']]
)
resm |>
  arrange(`学号`) |>
  knitr::kable()
```

学号	选择题	简答题	填空题	作文
1138010104	34.5	20.5	2.5	45.5
1138010108	26.0	24.0	2.0	48.5
1138010114	34.0	21.5	2.0	44.5
1138010126	35.0	27.0	2.5	44.0
1138010128	27.5	26.5	0.0	45.0

但是，我们要求最后的函数在允许修改小题类型、试卷中小题个数、各小题的类型、学生分数数据的条件下也能正常工作。

51.5.2 类别编号重排

设聚类分析对若干个样本点聚成了 3 类，结果如下：

```
dc0 <- tibble::tibble(
  obs = 1:10,
  g = c(3, 1, 3, 1, 2, 2, 1, 3, 3, 1)
)
knitr::kable(dc0)
```

obs	g
1	3
2	1
3	3
4	1
5	2
6	2
7	1
8	3
9	3
10	1

这些类别的次序并没有意义，重新聚类可能会将类别标签打乱重排但类别组成不变。所以，需要人为地指定合适的类别标签。

设对每个类进行概括统计得到了有代表性的统计量，结果如下：

```
dcstat <- tibble::tibble(
  g = 1:3,
  stat=c(2.3, 1.1, 3.0)
)
knitr::kable(dcstat)
```

g	stat
1	2.3
2	1.1
3	3.0

按照统计量从大到小的次序重排 3 个组，得到新的分组：

g	stat	ng
3	3.0	1
1	2.3	2
2	1.1	3

将原始观测添加新的分组标签 ng，并按 ng 排列，结果为：

obs	g	ng
1	3	1
3	3	1
8	3	1
9	3	1
2	1	2
4	1	2
7	1	2
10	1	2
5	2	3
6	2	3

输入 dc0 和 dcstat 数据框，如何用程序解决上述问题？

51.6 文本处理

51.6.1 用 R 语言下载处理《红楼梦》htm 文件

网上许多资源是 html 格式的文本文件。比如，《红楼梦》在许多网站可以浏览，是在浏览器中按章节浏览。我们希望将其下载到本地，并转换为 txt 格式，在手机或者电纸书阅读器中阅读。

这个任务涉及到 R 的文件访问，字符型连接，正则表达式，中文编码问题。

设下载网站主页是 http://www.xiexingcun.net/hongloumeng/index.html，各个章节的文件名是 `01.htm` 到 `99.htm`，`100.htm` 到 `120.htm`。

已下载的文件在如下链接中：hongloumeng.zip

将这些文件转换成 txt 格式，然后合并成一个 txt 文件。

要求：每一段仅用一行，中间不换行；不同段落之间用换行分开；每一回目开始有回目标题，标题前后空行，标题格式为 “第 xxX 回标题内容”。

Chapter 52

使用经验

52.1 文件管理

52.1.1 文件备份

对于正规的研究或者开发项目，一定要有完备的备份方案，最好是自动实时上传到云端。对于不需要保密的项目，可以使用各种免费的网盘服务，比如百度网盘提供了与网盘的同步功能，还可以提供一定程度的版本管理，但是我建议仅从电脑向网盘同步而不要从网盘向电脑同步。

可以每天将自己的项目用复制的方式备份到 U 盘或者移动硬盘。在 MS Windows 系统中，打开 cmd 命令窗口，复制备份的参考命令示例如下：

```
xcopy d:\disk G:\disk /E/D/R/Y
```

其中 `d:\disk` 是电脑硬盘上的项目工作目录，`G:\disk` 是 U 盘上的备份目录并以事先建立好。选项`/E/D/R/Y` 使得备份可以仅针对更新过的文件，会同时备份子目录，覆盖时不询问。

52.1.2 工作空间

R 和 RStudio 软件提供了保存工作空间的功能。我建议不要自动保存工作空间，而是用 `save()` 函数保存重要的数据、变量到硬盘中，需要时用 `load()` 函数载入。

52.2 程序格式

源程序必须按照一定的格式编写。其中比较重要的是：

- 必须有完善的注释。每个源文件和每个函数都需要有完整注释。函数内的算法不是很显然的情况下也应该用注释说明。

- 按照一定规则进行缩进。函数体必须缩进；`if`、`while` 等结构必须缩进，结构嵌套时进行更多的缩进对齐。

未完待续

附录 A

R Markdown 文件格式

附录 A-D 的内容现在已经升级为 Quarto(第22章) 内容，这些内容放在这里备查。

A.1 R Markdown 文件

借助于 R 的 knitr 和 rmarkdown 扩展包的帮助，可以在 Markdown 格式的源文件中插入 R 代码，使得 R 代码的结果能够自动插入到最后生成的研究报告中。这种格式称为 R Markdown 格式，简称为 Rmd 格式，相应的源文件扩展名为.Rmd。输出格式可以是 HTML、docx、pdf、beamer 等。R Markdown 的基础格式是 markdown 格式，严格说来是 Pandoc 软件支持的增强版的 markdown 格式，比如，支持 LaTex 格式的数学公式，支持各种编程语言语法彩色加亮显示，等等。

knitr 的详细文档参见网站knitr 文档。关于 R Markdown 可参考专著 (Xie et al., 2019) 和 (Xie et al., 2020)。RStudio 网站提供了一个 R Markdown 使用摘要下载：rmarkdown-2.0.pdf。Pandoc 的文档见pandoc 网站。

一个 Rmd 文件中包含元数据 (metadata)、正文内容和 R 代码三种成分，比如，下面是一个简单的 Rmd 文件样例：

```
---
title: "R 语言简介"
author: " 李东风"
date: `r Sys.Date()`
output: html_document
---
```

```
这里是正文段落。
段落中仍可以利用一般的 Markdown 语法，
比如在两边用双星号表示 ** 粗体加重 **，
在用两边反单撇号表示代码，如`y <- sin(x)`。
```

```
下面是一个 R 代码段，有文字输出：
```{r}
set.seed(1)
x <- round(rnorm(10), 2)
print(x)
```

下面是一个 R 代码段，有图形输出：

```{r}
plot(x)
```

下面是一个 R 代码段，
有富文本表格输出：

```{r}
knitr::kable(as.data.frame(x))
```

在文字段落内部也可以有代码，
比如，x 的第一个元素值为`r x[1]`。
```

在文件开头用三个减号组成的行包围的内容称为**元数据**，可以用来规定文章标题、作者、日期、输出格式、输出设置等属性。见A.11。

因为中文需要一些特殊的设置，以及在网络条件不好的条件下支持数学公式显示，本书作者提供了一个 Rmarkdown 模板，下载链接为:

- `bookdown-template-v0-6.zip`。

其中的 Rmarkdown 子目录包含了所需模板，其它子目录有一些别的模板，为了在本地支持网页中的数学公式显示还有一个 MathJax 目录。

A.2 R Markdown 文件的编译

RStudio 是一个集成的 R 软件环境，可以用来编辑和执行 R 程序，这个软件也可以用来编辑和编译 R Markdown 格式的文件，使得 R Markdown 格式的文件变得容易使用。在 RStudio 中可以直接用一个快捷图标一次性地把 R 代码结果插入内容中并编译为 HTML 或 MS Word docx 格式，还支持 Markdown 中 LaTeX 格式的数学公式。建议使用 RStudio 软件作为 R Markdown 文件的编辑器。

在 RStudio 软件中，用菜单 “File–New File–R Markdown” 新建一个 R Markdown 文件，扩展名为`.Rmd`。用快捷图标 Knit 可以将文件转换成 HTML 格式、PDF 格式（需要安装 LaTeX 编译软件 TinyTeX）、MS Word 格式。

从 HTML 格式可以转换成 PDF 格式。为此，安装 Google 的 Chrome 浏览器，在 Chrome 中打开 HTML 文件后，然后选择菜单 “打印”，选打印机为 “另存为 PDF”，然后选 “更多设置”，将其中的 “缩放” 改为自定义，比例改为 “90%”，就可以将 HTML 网页转换成 PDF，其中的数学公式、表格、图形都可以比较好地转换。注意，如果不缩小打印，数学公式的编号以及较长的数学公式可能会被裁剪掉。

从 Word 文件也可以转换成 PDF 格式，用 MS Word 软件的 “文件-导出” 或者 “文件-另存为” 功能即可。

如果想将 R Markdown 文件借助于 LaTeX 格式转换为 PDF，需要在系统中安装一个 TeX 编译器。rmarkdown 包从第 1.9 版本开始要求使用 tinytex 扩展包以及配套的TinyTeX软件包，没有提供用本机原有的 LaTex 编译系统的选择，如果不安装 tinytex，编译为 PDF 格式时会出错。关于 TinyTeX 的安装和使用参见A.12。

如果不借助于 RStudio 软件，可以用 R 软件、knitr 包、rmarkdown 包、pandoc 软件来完成 R Markdown 源文件的编译。比如，假设 `myfile.Rmd` 是一个这样的 R Markdown 格式的文件，注意一定要使用 UTF-8 编码保存，可以在 R 或 RStudio 中运行如下命令以生成含有运行结果的 html 文件:

```
rmarkdown::render("myfile.Rmd", output_format = "html_document", encoding="UTF-8")
```

其中 `myfile.Rmd` 是源文件，产生的 HTML 文件带有图形、支持数学公式。

在 R 或 RStudio 中可以用如下命令把.Rmd 文件转化为 MS Word docx 格式:

```
rmarkdown::render("myfile.Rmd", output_format = "word_document", encoding="UTF-8")
```

使用 RStudio 软件使得这些任务可以一键完成，而且有很好的数学公式支持，并且可以在 RStudio 软件中预览 HTML 结果，所以建议使用 RStudio 软件来编辑和转换 R Markdown 文件。

用 RStudio 的 Knit 图标一键编译与用 `rmarkdown::render()` 命令编译有重要差别：

- 用 Knit 图标编译，Rmd 文件中的程序会在一个崭新的会话中执行，当前会话中已经定义的函数、变量、导入的扩展包不会影响到编译结果；
- 用 `rmarkdown::render()` 编译，Rmd 文件中的程序是在当前会话中执行的，会带来一定的兼容性问题，有可能在别人的环境下就不能正确执行或者会给出不同结果。但是，`rmarkdown::render()` 可以通过程序调用，比如，循环地从同一个 Rmd 生成一系列不同的报告。为了不让当前会话环境干扰结果，可以人为地打开一个新会话。作者的经验是用程序而不是用图表编译的做法更稳定可靠。

A.2.1 编译的实际过程

编译 Rmd 文件，一般是调用 `rmarkdown::render()` 函数执行任务。编译的第一步，会调用 knitr 包的 `knit()` 函数将代码段中的程序依次运行，结果保存为一个临时.md 文件中，图形结果暂存在单独的图形文件中，文字结果会直接保存在临时.md 文件中。编译的第二步，调用外部程序 Pandoc 将临时的.md 文件转换为要求的 Word、

HTML 等输出格式；如果要求的输出是 `pdf_document`，则会调用 Pandoc 先将临时的.md 文件转换为一个中间的.tex 文件，然后利用 tinytex 包调用外部的 TinyTex 软件将.tex 文件编译为 PDF 结果。

Pandoc是一个功能强大的文件转换程序，可以在多种不同的文件格式之间转换，具有丰富的选项。对于 markdown 格式，Pandoc 也提供了丰富的扩充功能，最重要的是支持 LaTeX 格式的数学公式。见21.4。

A.3 在 R Markdown 文件中插入 R 代码

插入的 R 代码分为行内代码与代码块。

行内代码的结果插入到一个段落中间，代码以`` `r `` 开头，以`结尾，如`` `r sin(pi/2)` ``在结果中会显示为 1。为了原样显示一个反向单撇号，可以在两边用双反向单撇号界定并用空格隔开内部的内容。

代码块则把结果当作单独的段落，按照 Markdown 格式的规定，代码块的前后需要有空行，但是 R Markdown 实际上放松了这个要求，允许前后不空行。R 代码段以单独的一行开头，此行以三个反单撇号开始，然后是 `{r}`，如```{r}。代码段以占据单独一行的三个反单撇号```结尾。如

```
```{r rmd-block01, cache=TRUE}
set.seed(1)
x <- round(rnorm(10), 2)
print(x)
```
```

结果将变成

```
set.seed(1)
x <- round(rnorm(10), 2)
print(x)
```

```
##  [1] -0.63  0.18 -0.84  1.60  0.33 -0.82  0.49  0.74  0.58 -0.31
```

可以看出，代码段程序会被插入到最终结果中，代码段的文本型输出会插入到程序的后面。

代码块也可以嵌入到引用、列表等环境中。

代码块中作的图将自动插入到当前位置。下面的程序：

```
```{r rmd-block02, cache=TRUE}
plot(x)
```
```

结果将显示为:

```
plot(x)
```



在 RStudio 中，可以用 Insert 快捷图标插入代码段，还可以用 Ctrl+Alt+I 快捷键插入代码段。

A.4 输出表格

knitr 包提供了一个 `kable()` 函数可以用来把数据框或矩阵转化成富文本格式的表格，支持 HTML、docx、LaTeX 等格式。

例如，计算线性回归后，`summary()` 函数的输出中有 coefficients 一项，是一个矩阵，如果直接文本显示比较难看：

```
x <- 1:10; y <- x^2; lmr <- lm(y ~ x)
co <- summary(lmr)$coefficients
print(co)
```

```
##             Estimate Std. Error   t value     Pr(>|t|)
## (Intercept)      -22  5.5497748 -3.964125 4.152962e-03
## x                 11  0.8944272 12.298374 1.777539e-06
```

可以用 knitr 包的 kable 函数来显示:

```
knitr::kable(co)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-22	5.5497748	-3.964125	0.0041530
x	11	0.8944272	12.298374	0.0000018

`kable()` 函数的 `digits=` 选项可以控制小数点后数字位数，`caption=` 选项可以指定表的标题内容。

R 扩展包 xtable 提供了一个 `xtable()` 函数，也可以用来生成 HTML 格式和 LaTeX 格式的表格，但是需要指定要输出的格式。xtable 对比较多的 R 数据类型和输出类型提供了表格式显示功能，包括矩阵、数据框、回归分析结果、方差分析结果、主成分分析结果、若干分析结果的 summary 结果等。例如，上面的回归结果用 `xtable()` 函数显示如:

```
print(xtable::xtable(lmr), type='html')
```

这个代码段用了选项 `results='asis'`，因为 xtable 生成的是直接用来插入到结果中的 html 代码。注意这里指定了输出为 HTML 类型。如果将本文件转化为 docx, xtable 的结果不可用, 所以在代码段选项中加了 `eval=knitr::is_html_output()` 使得代码段仅在针对 HTML 输出时才运行。

R 扩展包 pander 提供了更好的表格能力，也能与 knitr 包很好的合作输出。其 `pander()` 函数可以将多种 R 输出格式转换成 knitr 需要的表格形式。如

```
pander::pander(lmr)
```

表 A.1: Fitting linear model: y ~ x

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-22	5.55	-3.964	0.004153
x	11	0.8944	12.3	1.778e-06

但是，经过试验发现，表中中有中文时 pander 包会出错。

A.5 利用 R 程序插图

Rmd 文件的插图有两种，一种是已经保存为图形文件的，主要是 png 和 pdf 图片；另一种是文中的 R 代码生成的图形。

已经有图形文件的，可以用 markdown 格式原来的插图方法，见markdown 格式介绍。但是，这样做不能给图形自动编号，另外因为制作图书是有网页和 PDF 书两种主要输出格式的，原有的插图方式在这两种输出格式上有细微的不一致。所以，最好是统一使用 Rmd 的插图方法。

Rmd 的插图方法就是写一段 R 代码段来插图，如果是用程序作图，则代码中写作图的代码；如果是已有的图形文件，可以在一个单独的 R 代码段中用类似下面的命令插图：

```
```{r, echo=FALSE}
knitr::include_graphics("figs/myfig01.png")
```
```

其中 `figs` 是存放图形文件的子目录名，`myfig01.png` 是要插入的图形文件名。这样，如果同时还有 `myfig01.pdf` 的话，则 HTML 输出使用 png 图片而 PDF 输出自动选用 pdf 文件。另外，插图的选项在代码段的选项中规定：用代码段的 `fig.with` 和 `fig.height` 选项指定作图的宽和高（英寸），用 `out.width` 和 `out.height` 选项指定在输出中实际显示的宽和高，实际显示的宽和高如果使用如`"90%"` 这样的百分数单位则可以自动适应输出的大小。

由于 PDF 中的中文编码不能自动识别，所以在每个 Rmd 源文件的开头应该加上如下的设置，使得生成 PDF 图时中文能够正确显示：

```
```{r setup-pdf, include=FALSE}
pdf.options(family="GB1")
```
```

其中 `include=FALSE` 表示要不显示代码段的代码，有运行结果也不插入到输出结果中，是否运行视缺省的 `eval=` 的值而定。

A.6 冗余输出控制

在交互运行时，R 的函数调用输出的警告、提示信息是很有必要的。但是，在使用 R Markdown 制作论文、技术报告、图书时，这些信息会分散注意力，使得结果显得不够专业。可以设法抑制这些信息的显示。

函数 `suppressWarnings()` 可以抑制警告信息，`suppressMessages()` 可以抑制提示信息，某些函数本身也提供了一些抑制信息的选择。例如，`library()` 函数在载入扩展包时往往显示许多提示、警告信息，可以用类似如下程序的方法已知冗余的输出信息：

```
library(tidyverse,
  warn.conflicts=FALSE,
  quietly=TRUE,
  verbose=FALSE) |>
  suppressMessages() |>
  suppressWarnings()
```

A.7 代码段选项

独立代码段以`` ```{r} `` 开头，在大括号内还可以写一些选项，选项之间以及与开始的 `r` 之间用逗号分隔，所有选项写在同一行内不要换行。选项都使用“选项名 = 选项值”的格式，选项值除了使用常量外也可以使用全局变量名或表达式。在大括号内开头的 `r` 空格后写一个由英文大小写字母、数字、减号组成的标识符，作为代码段的标签。标签中不要用其它类型的字符，下划线也不要用。如

```
```{r firstCode}
cat('This is 第一段，有标签.\n')
```
```

关于代码段选项，详见https://yihui.name/knitr/options。

A.7.1 代码和文本输出结果格式

R 代码块和 R 代码块的运行结果通常是代码块原样输出，运行结果用井号保护起来，这样有利于从文章中复制粘贴代码。如：

```
```{r}
s <- 0
for(x in 1:5) s <- s + x^x
s
```
```

结果为:

```r
s <- 0
for(x in 1:5) s <- s + x^x
s
```

```
## [1] 3413
```

A.7.1.1 highlight 选项

转化后的 R 代码块缺省显示为彩色加亮形式。用选项 `highlight=FALSE` 关闭彩色加亮功能。

A.7.1.2 prompt 和 comment 选项

如果希望代码用 R 的大于号提示符开始，用选项 `prompt=TRUE`。如果希望结果不用井号保护，使用选项 `comment=''`。例如：

```
```{r prompt=TRUE, comment=''}
sum(1:5)
```
```

结果为:

```
> sum(1:5)
```

```
[1] 15
```

A.7.1.3 echo 选项

如果希望不显示代码，加选项 echo=FALSE。如

```
```{r echo=FALSE}
print(1:5)
```
```

结果为:

```
## [1] 1 2 3 4 5
```

A.7.1.4 tidy 选项

加选项 tidy=TRUE 可以自动重新排列代码段，使得代码段格式更符合规范。例如：

```
```{r tidy=TRUE}
s <- 0
for(x in 1:5) {s <- s + x^x; print(s)}
```
```

结果为:

```
s <- 0
for (x in 1:5) {
    s <- s + x^x
    print(s)
}
```

```
## [1] 1
## [1] 5
## [1] 32
## [1] 288
## [1] 3413
```

A.7.1.5 eval 选项和 include 选项

加选项 `eval=FALSE`, 可以使得代码仅显示而不实际运行。这样的代码段如果有标签，可以在后续代码段中被引用。

加选项 `include=FALSE`，则本代码段仅运行，但是代码和结果都不写入到生成的文档中。

A.7.1.6 child 选项

加选项 `child='文件名.Rmd'`可以调入另一个.Rmd 文件的内容。如果有多个.Rmd 文件依赖于相同的代码，可以用这样的方法。

A.7.1.7 collapse 选项

一个代码块的代码、输出通常被分解为多个原样文本块中，如果一个代码块希望所有的代码、输出都写到同一个原样文本块中，加选项 `collapse=TRUE`。例如，没有这个选项时：

```
```{r}
sin(pi/2)
cos(pi/2)
```
```

结果为：

```r
sin(pi/2)
```

```
## [1] 1
```

```r
cos(pi/2)
```

```
## [1] 6.123032e-17
```

代码和结果被分成了 4 个原样文本块。加上 `collapse=TRUE` 后，结果为：

```r
sin(pi/2)
## [1] 1
cos(pi/2)
## [1] 6.123032e-17
```

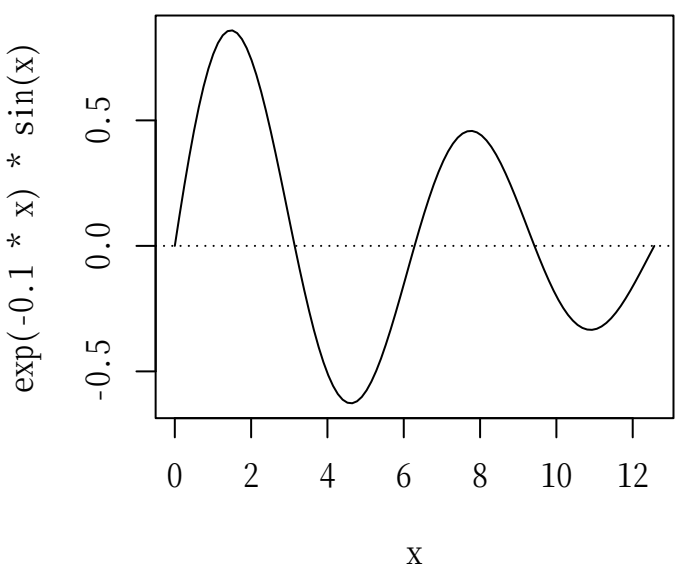
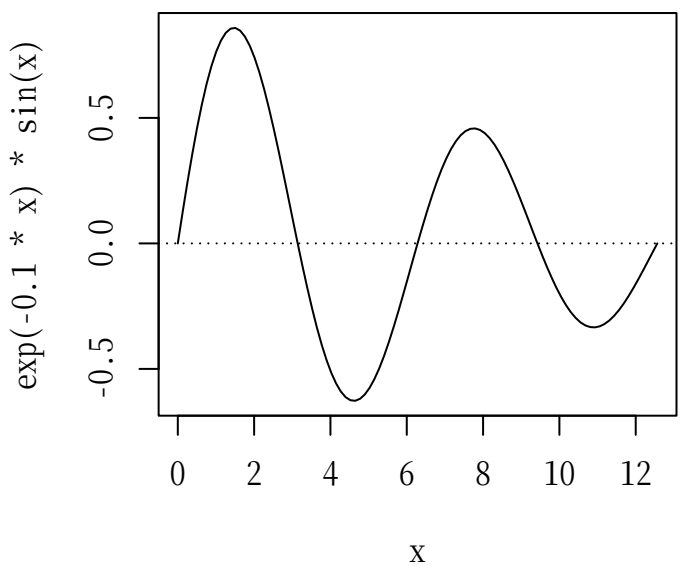
代码和结果都在一个原样文本块中。

A.7.1.8 results 选项

用选项 `results=` 选择文本型结果的类型。取值有：

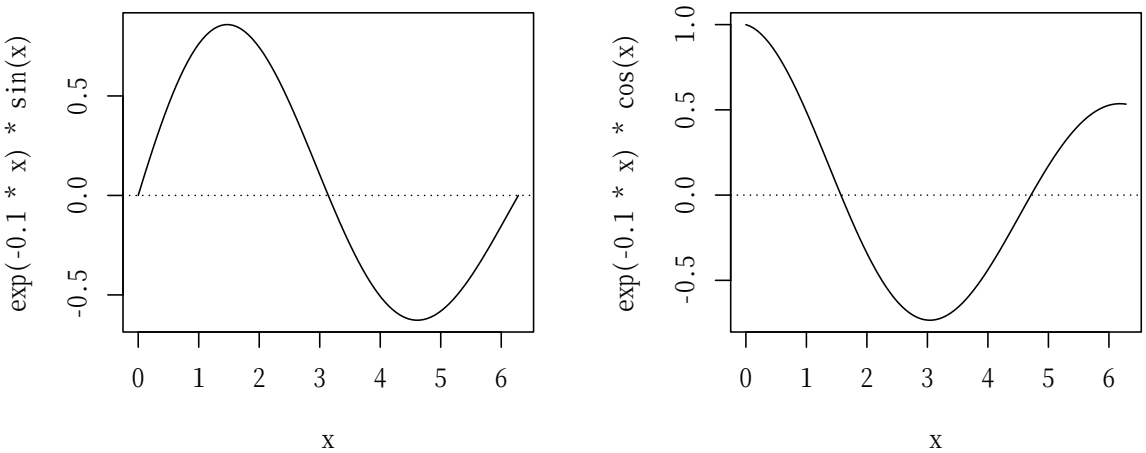
- `markup`, 这是缺省选项，会把文本型结果变成 HTML 的原样文本格式。
- `hide`, 运行了代码后不显示运行结果。
- `hold`, 一个代码块所有的代码都显示完，才显示所有的结果。
- `asis`, 文本型输出直接进入到 HTML 文件中，这需要 R 代码直接生成 HTML 标签，knitr 包的 `kable()` 函数可以把数据框转换为 HTML 代码的表格。

例如：`results='hold'`的示例:

```
```{r collapse=TRUE, results='hold'}
sin(pi/2)
cos(pi/2)
```
```

结果为:

```
sin(pi/2)
cos(pi/2)
## [1] 1
## [1] 6.123032e-17
```

A.7.1.9 错误信息选项

选项 `warning=FALSE` 使得代码段的警告信息不进入编译结果，而是在控制台 (console) 中显示。有一些扩展包的载入警告可以用这种办法屏蔽。

选项 `error=FALSE` 可以使得错误信息不进入编译结果，而是出错停止并将错误信息在控制台中显示。

选项 `message=FALSE` 可以使得 message 级别的信息不进入编译结果，而是在控制台中显示。

A.7.1.10 控制普通文字结果宽度

Rmd 代码段选项中没有直接提供控制输出的普通文字结果行宽的功能，但可以用代码行 `options(width=80)` 控制，这里 `80` 可以替换成自己选择的每行字符个数。如：

```
options(width=80)
print(123456789001:123456789020)
##  [1] 123456789001 123456789002 123456789003 123456789004 123456789005
##  [6] 123456789006 123456789007 123456789008 123456789009 123456789010
```

```
## [11] 123456789011 123456789012 123456789013 123456789014 123456789015
## [16] 123456789016 123456789017 123456789018 123456789019 123456789020
```

用 `options()` 可以设置行宽，PDF 输出的编码，这样的调用一般放在 Rmd 文件的开头，YAML 的后面，用 `setup` 作代码段标签，加 `include=FALSE` 选项表示仅运行但不使用运行得到的文字和图形输出结果。

A.7.2 图形选项

A.7.2.1 图形大小

用 `fig.width=` 指定生成的图形的宽度，用 `fig.height=` 指定生成的图形的高度，单位是英寸（1 英寸等于 2.54 厘米）。

下面给出一个长宽都是 10 厘米的图例。

```
```{r fig.width=10/2.54, fig.height=10/2.54}
curve(exp(-0.1*x)*sin(x), 0, 4*pi)
abline(h=0, lty=3)
```
```

结果为:

```
curve(exp(-0.1*x)*sin(x), 0, 4*pi)
abline(h=0, lty=3)
```



`fig.width=` 和 `fig.height=` 规定的是生成的图形大小，实际生成图形的显示大小会受到 dpi（分辨率）影响，默认 dpi 是 72（每英寸 72 个点），也可以用 `dpi=` 选择分辨率。转化后的 HTML 文件显示时不一定按原始大小显示。用 `out.width=` 和 `out.height=` 可以指定显示大小，但是必须是最终文件格式承认的单位，比如 HTML 的图形大小单位是点（平常说屏幕分辨率的单位）。或者写成如 `90%` 这样的百分比格式。例如在上面的例子中加上输出度为页面宽度 80% 的选项：

```
```{r fig.width=10/2.54, fig.height=10/2.54, out.width="80%"}
curve(exp(-0.1*x)*sin(x), 0, 4*pi)
abline(h=0, lty=3)
```
```

注意所有选项都要写在一行中，不能换行。结果为:

```
curve(exp(-0.1*x)*sin(x), 0, 4*pi)
abline(h=0, lty=3)
```



当输出结果为 HTML 结果或者 LaTeX 转换的 PDF 时，只要两个图形的总宽度小于页面宽度，两个连续的图形就可以左右并列放置。Word 不支持这种功能。例如：

```
```{r, echo=FALSE, fig.width=10/2.54, fig.height=10/2.54, out.width="45%"}
curve(exp(-0.1*x)*sin(x), 0, 2*pi)
abline(h=0, lty=3)
curve(exp(-0.1*x)*cos(x), 0, 2*pi)
```
```

```
abline(h=0, lty=3)
```

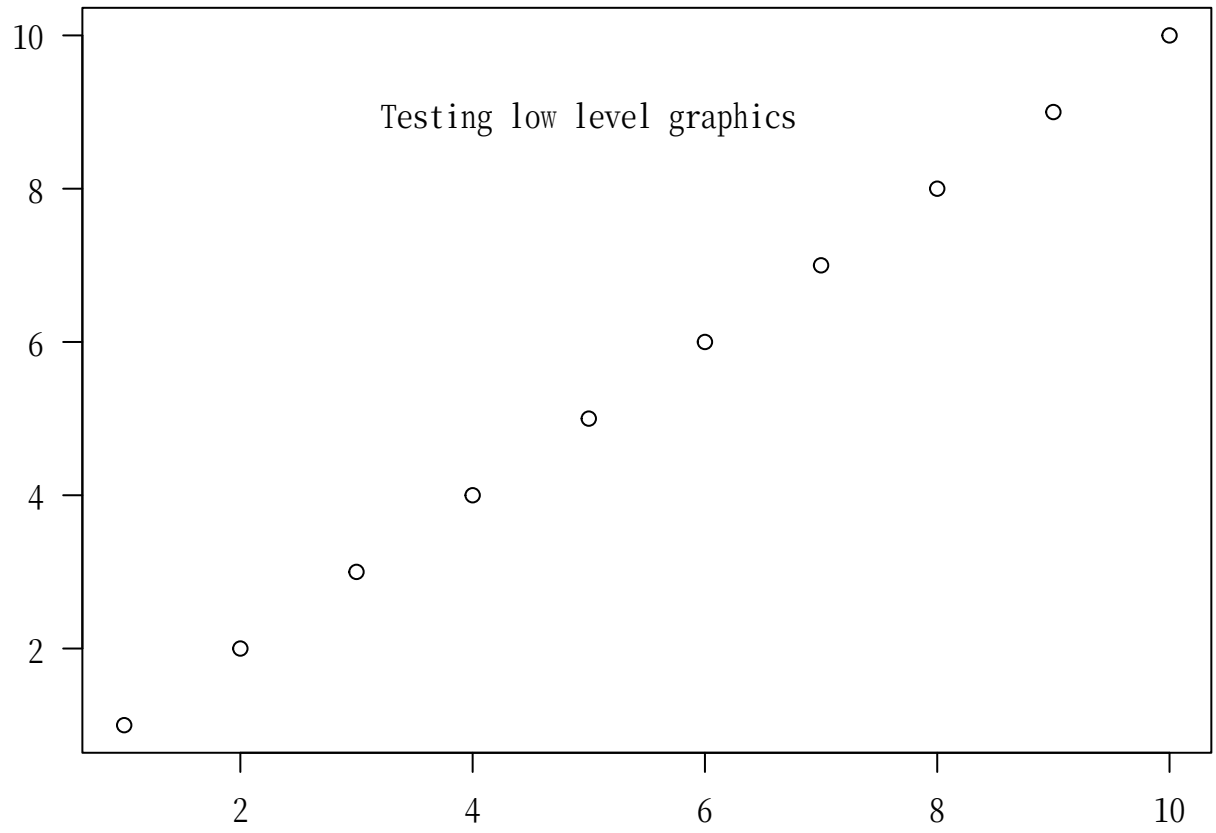
结果为



A.7.2.2 图形结果选择

用 `fig.keep=` 选项可以选择保留哪些 R 代码生成的图。缺省是 `fig.keep="high"`, 即保留每个高级图形函数的结果图形，低级图形函数对高级图形函数的更改不单独保存而汇总到高级图形函数结果中。如

```
par(mar = c(3, 3, 0.1, 0.1))
plot(1:10, ann = FALSE, las = 1)
text(5, 9, "Testing low level graphics")
```



其中 `text()` 函数的结果与高级图形函数 `plot()` 的结果一起显示。

`fig.keep` 还可以取：`"all"`, 会把低级图形函数修改后的结果单独保存；`"last"`, 仅保留最后一个图形；`"first"`, 仅保留第一个图；`"none"`, 所有图都不显示出来。

A.7.2.3 图形中的中文编码

如果编译输出目标是 HTML，程序产生的图形是 PNG 格式，属于点阵图，图中的任何中文内容不会有乱码问题。但是，如果输出目标是 PDF，程序将产生 PDF 格式的图形，这种图形属于矢量图，需要正确设置文字编码才能保证图中的中文不会乱码。设置的办法是用 `pdf.options()` 函数作如下指定：

```
pdf.options(height=10/2.54, width=10/2.54, family="GB1")
```

其中的 `family` 选项指定了支持中文，而 `height` 和 `width` 则是指定生成的每个图片的物理大小，以英寸为单位。这样的设定一般都放在 Rmd 文件开头，YAML 后面的位置，用 `setup` 作代码段标签，加 `include=FALSE` 代码段选项表示仅运行但不使用运行得到的文字和图形输出结果。

A.7.3 缓存 (cache) 选项

当 R Markdown 文章比较长，包含的 R 代码比较多，或者代码段运行需要比较长时间时，反复编译整篇文章会造成不必要的计算，因为有些代码段并没有修改，依赖的数据也没有改变。knitr 提供了缓存功能，代码段选项

cache=TRUE 对代码段打开缓存，允许暂存上次运行的结果（包括产生的变量、函数、文本结果和图形）而不需要重复运行代码段。当代码段被修改时，缓存被放弃，编译时重新运行代码段。

缓存这种功能需要慎重使用，免得错误地使用了旧的结果。另外，缓存功能现在还不够完善，如果更新了 R 软件版本，有可能缓存还依赖于老版本中的某些文件，而这些文件已经被删除，就会出错。如果出现这种错误，可以删除所有缓存文件，重新编译整个项目。

当后面的代码段需要使用前面代码段结果时，如果前面结果改了，后面的代码段就不能使用缓存的结果而必须重新计算。为此，在后面的代码段中应该加上 dependson= 选项，比如 dependson=c("codeA", "codeB")，其中 codeA 和 codeB 是前面的缓存了的代码段的标签，其结果会用在本代码段中。也可以使用代码段选项 autodep=TRUE，knitr 试图自动确定前后代码段之间的依赖关系，每当前面的代码段改变时，后面的用到其结果代码段也自动重新计算而不使用缓存的旧结果。

建议仅对计算一次需要较长时间的代码段使用缓存功能，后面依赖于其结果的代码一定要加上 dependson= 选项。建议代码段尽可能有标签，这样在编译失败时能马上看出失败的地点。

对于用 options()、par() 等改变全局设置的代码段，应不缓存，缓存功能不能缓存这些全局设置的修改。

为了保险起见，可以尽量不使用缓存功能，而是用 save() 功能将需要长时间结算的结果保存下来，用 load() 载入然后输出到结果文档中。

A.8 章节目录链接问题

对于英文文件，R Markdown 基于的 pandoc 软件可以自动从标题生成适当的链接标签，将 HTML 输出或者 PDF 输出设置属性 toc: true 后可以生成可点击的章节目录。但是，Pandoc 对于中文文件的支持差一些，所以中文文件要自动生成可点击的目录，需要在每个标题行的末尾，空格后添加 {#label}，其中 label 是自己指定的标签内容，但是建议仅使用英文字母、数字、减号。pandoc 软件也在改进，这样的做法更为稳妥可靠。如

```
### 第三章第一节标题 {#c3-s1-int}
```

一个常见的问题是标题末尾的标签忘记了大括号前面的逗号或者大括号开始的井号。

为了引用某个章节，方法如见 [第三章第一节](#c3-s1-int)，与 markdown 格式的链接类似，但是地址采用以井号开头的标签格式。

如果希望某个标题不参与自动编号，可以在标题末尾空格之后加上 {-} 标记。但是，这种办法使得该标题没有合适的链接，所以另一种做法是在大括号内写.unnumbered 属性和标签，如：

```
## 摘要 {.unnumbered #abstract}
```

还可以用.unlisted 属性表示不要将标题加入到目录中。

bookdown 包支持的部分（PART）、附录（APPENDIX）也使用 {-} 来标记，也存在没有合适的链接的问题，也可以试验用.unnumbered 属性。

A.9 其它编程语言引擎

Rmd 文件中除了 R 代码段以外，还可以插入 Rcpp、Python、Julia、SQL 等许多编程语言的代码段，常用编程语言还可以与 R 代码段进行信息交换。

如果安装了多个 Julia 语言的版本，需要在操作系统 PATH 环境变量中正确设置需要使用的版本的路径。

A.10 交互内容

在 Rmd 生成 HTML 结果时，可以在结果中包含允许读者交互操作的内容，比如，用户修改模型参数，使得模型报表、图形交互地改变。有两种方法可以实现交互：

- HTML 小程序 (widgets)，由 R 扩展包htmlwidgets实现，利用 JavaScript 函数库进行交互。在这一功能基础上有一些派生的扩展包，如 DT，leaflet, dygraph。
- 利用 shiny 框架，这也是一个 R 扩展包，交互功能由 R 的一个会话驱动。Rmd 文件的 YAML 元数据中需要设置 `runtime: shiny`。

A.11 属性设置

A.11.1 YAML 元数据

可以在 RStudio 软件中编辑 markdown 文件与 Rmarkdown 文件，RStudio 的 markdown 支持来自 knitr、rmarkdown、bookdown 扩展包和 pandoc 软件，.Rmd 文件支持一些特殊的文件设置，这些设置写在 markdown 文件和.Rmd 文件的开头位置，用三个减号组成的行作为开始标记与结束标记，称为 YAML 元数据块 (YAML meta data block)，块后面必须用空行分隔。元数据块中可以用 “设置属性名: 设置属性值” 的办法设置属性，用缩进表示属性内容，用上下对齐的行表示多项列表。常用属性有 `title`(标题)、`author`(作者)、`date`(日期)、`output_format`(输出格式) 等，如

```
---
title: " 临时测试"
author: " 李东风"
date: "2016 年 7 月 6 日"
---
```

这三个设置会出现在转换后结果的标题部分。RStudio 的中文支持还是有时出现问题，如果出现涉及到 YAML 的错误，先将中文内容替换为英文试一试。

因为冒号：在属性设置中有特殊意义，属性设置值如果含有冒号，需要把整个属性值两边用单撇号界定。

属性值可以是列表或多项，例如：

```
---
title:  'This is the title: it contains a colon'
author:
  - name: Author One
    affiliation: University of Somewhere
  - name: Author Two
    affiliation: University of Nowhere
tags: [nothing, nothingness]
abstract: |
  This is the abstract.

  It consists of two paragraphs.
---
```

此例中有 title、author、tags、abstract 四个属性。author 属性包含两个作者，每个作者又有 name 和 affiliation 两个属性。tags 属性是一个两项的列表。abstract 属性用管道符号 `|` 表示开头，不需要结束标志，内容缩进。

A.11.2 输出格式

.Rmd 文件开头的 YAML 元数据中，属性 `output` 选择输出的格式，如

- `html_document` 是 HTML 输出;
- `pdf_document` 是 PDF 输出，通过系统中另外安装 LaTeX 编译系统转换;
- `word_document` 是 docx 格式的 Word 文件输出，等等。

输出格式指定如：

```
output:
  html_document: default
  word_document: default
  pdf_document: default
```

其中 default 表示该输出格式完全使用默认设置。在输出格式没有定制设置时必须有 default 指定。

除了上述三种常见格式以外，在 RStudio 中使用时，还支持一种 `html_notebook` 输出格式，这种输出格式完全在当前的 R 会话中运行代码块，其 Knit 按钮会变成 Preview 按钮，可以显示当前已有的结果到 HTML 窗口，不显示未运行部分的结果。这可以交互地人为控制代码块的运行，因为不需要重新运行代码块，预览的速度也比较快。其它输出格式在用 Knit 编译时都要在一个新的 R 会话中逐一运行文中的代码块。

有时需要根据编译输出目标采用不同的设置，比如，输出为 html 时可以使用 HTML 插件 (widgets)，而输出到 PDF 则不可以。函数 `knitr::is_html_output()` 可以判断当前编译的源文件是否正在被编译为 HTML，`knitr::is_latex_output()` 可以判断是否正在编译为 LaTeX 然后再转换为 PDF。

另外，也可以用 `knitr::opts_knit$get("rmarkdown.pandoc.to")` 感知当前编译过程的目标，对 HTML 类，结果为`"html"`，对 PDF，结果为`"latex"`。

A.11.3 输出格式设置

在每种输出格式后面还可以继续添加该输出特有属性。如：

```
output:
  html_document:
    toc: true
    number_sections: true
  word_document:
    toc: true
  pdf_document:
    toc: true
```

A.11.4 目录设置

输出格式的 `toc: true` 选项指定自动生成目录，HTML 输出、PDF 输出、Word 输出都支持此功能。如：

```
output:
  html_document:
    toc: true
    toc_depth: 2
```

上例中，`html_document` 是 `output` 的属性，隶属关系用缩进表示；`html_document` 又有自己的属性 `toc` 和 `toc_depth`，隶属关系用缩进表示。属性 `toc: true` 表示要自动生成可点击的目录。输出格式的属性 `toc_depth` 是目录包含的章节层级数。

对 `html_document` 输出，属性 `toc_float: true` 使得生成的文档在左侧显示一个目录导览窗格。而 `toc_float` 属性又可以指定一些属性，隶属关系用缩进表示。`toc_float` 的属性 `collapse: true` 表示仅展开章节第一层级，`smooth_scroll: true` 使得通过导览窗格跳跃时页面有平滑滚动过程显示。用如：

```
output:
  html_document:
    toc: true
    toc_float:
      collapse: true
      smooth_scroll: true
```

A.11.5 章节自动编号

输出格式的属性 `number_sections` 为 `true` 可以自动对章节编号。HTML 输出与 PDF 输出都支持此功能。

在章节自动编号时，如果不是利用 bookdown 包而是基于基本的 R Markdown，文中的分节最高层级最好用一个井号，对应于节，如果用两个井号，编号会变成 `0.1`，`0.2` 这样。

A.11.6 Word 输出章节自动编号及模板功能

Word 文件不能自动编号，解决办法是生成适当的模板文件，如 wordstyle.docx, 然后在 Rmd 文件的元数据部分为 `word_document` 输出增加 `reference_docx` 选项，如:

```
output:
  html_document:
    toc: true
    number_sections: true
  word_document:
    toc: true
    reference_docx: wordstyle.docx
```

这样，新编译的 Rmd 文件就可以采用 `wordstyle.docx` 的格式设置，比如章节自动编号等。目前目录标题还是英文，需要在结果中人为地修改为中文。一个样例模板下载如下：wordstyle.docx

A.11.7 HTML 特有输出格式设置

对于 `html_document` 输出类型，可以用 `theme` 属性设置一个主题，取值如 default, cerulean, journal, flatly, darkly, readable, spacelab, united, cosmo, lumen, paper, sandstone, simplex, yeti。

用 `highlight` 属性设置程序语言语法高亮样式，可取值有 default, tango, pygments, kate, monochrome, espresso, zenburn, haddock, and textmate。用 `null` 表示取消语法高亮。

可以用 `css` 属性指定一个自定义的 CSS 样式表文件。如果希望完全用自己的样式代替原有样式，可以设置 `theme: null`。

`code_folding` 属性设置 HTML 结果中代码折叠选项，取值 `hide` 可以隐藏代码，但读者通过选项可见。取值 `show` 可以显示代码，但读者通过选项可隐藏代码显示。

某些显示功能仅适用于 HTML 输出。在使用 rmarkdown、bookdown 创作文章和书籍时，因为同一源文件需要能同时支持 HTML、LaTeX 转换 PDF，可以在 Rmd 源文件开头运行命令：

```
is_html <- knitr::opts_knit$get("rmarkdown.pandoc.to") == "html"
```

这可以定义一个变量 `is_html`，仅在输出格式为 HTML 时才为 `TRUE`，然后在包含特殊 HTML 显示的代码段选项中，加选项 `eval = is_html`。

A.11.8 关于数学公式支持的设置

Rmd 格式支持数学公式，在生成的 HTML、Word、PDF 中都可以正常显示数学公式，RStudio 软件内部的浏览器也可以直接显示数学公式。

以 HTML 为输出时，会使用 MathJax 库显示数学公式。这是一个用于在浏览器中显示数学公式的 Javascript 程序库，显示效果很好，还支持多种显示实现方式，支持包括 LaTeX 在内的多种数学公式输入方法。R Markdown 原来使用 MathJax 版本 2，现在 MathJax 已经发布了版本 3，有很大的改动，我们这里应仍使用版本 2。

MathJax 库很大，所以一般是从其网站按需远程调用的，但是远程调用 MathJax 库在网络不畅通时会使得公式显示极为缓慢甚至无法显示，所以 Rmd 允许将 MathJax 本地化。在.Rmd 文件头部 YAML 元数据的某种 HTML 输出格式的属性中，设置属性 `mathjax: local` 和就可以使得 MathJax 库被放在生成的 HTML 的一个下层目录中。设置如

```
output:
  html_document:
    toc: true
    self_contained: false
    mathjax: local
```

注意设置 `mathjax: local` 同时必须设置 `self_contained: false`。在不使用本地 MathJax 副本时，可以设置 `self_contained: true`，这使得图形文件、JavaScript 代码等依赖项也打包在生成的单一 HTML 文件中，如果设为 `false`，这些依赖文件会存放在单独文件中。可以用 `lib_dir` 属性指定一个子目录用来存放这些依赖文件，多个 Rmd 文件可以共用同一个 `lib_dir` 值。

将 MathJax 用 `mathjax: local` 属性本地化有明显的缺点。这时，每个 Rmd 项目都需要一个 MathJax 库副本，使得文件备份变得很麻烦。下面的设置方法可以令多个 Rmd 项目共享一个共同的本地 MathJax 库。

A.11.9 输出设置文件

如果一个项目包含多个.Rmd 文件，每个文件单独用 YAML 元数据进行设置比较繁琐，修改时也很麻烦。可以在项目目录中增加一个 `_output.yml` 文件，其中包含所有.Rmd 文件共用的 `output` 属性的设置，各个.Rmd 文件还可以有自己单独的 output 属性，.Rmd 文件 YAML 元数据中的属性优先级高于共同设置的属性。

例如，`_output.yml` 文件内容如下：

```
html_document:
  toc: yes
  number_sections: yes
  mathjax: "../../MathJax/MathJax.js"
  includes:
    in_header: "_header.html"
word_document:
```

```
  toc: yes
  reference_docx: wordstyle.docx
pdf_document:
  includes:
    in_header: preamble.tex
  latex_engine: xelatex
```

上面设置了使用本地硬盘的 MathJax 库，位于项目目录的上两层目录中的 MathJax 子目录中。这样可以使得多个项目共享一个 MathJax 库。将生成的结果当作网站发布时，只要将 MathJax 库也安装到项目在网站的目录的上两层的 MathJax 子目录中就可以。其中 `_header.html` 内容如下，是关于 MathJax 具体的一些设置：

```
<script type="text/x-mathjax-config">
MathJax.Hub.Config({
  jax: ["input/TeX","output/SVG"],
  extensions: ["tex2jax.js","MathMenu.js","MathZoom.js"],
  TeX: {
    extensions: ["AMSmath.js","AMSsymbols.js","noErrors.js","noUndefined.js"]
  }
});
</script>
```

注意，`_output.yml` 文件都是输出设置，所以这个文件中不能再写 `output` 属性，下面的 `_output.yml` 写法是不正确的：

```
output:
  html_document:
    toc: yes
```

A.12 LaTeX 和 PDF 输出

Rmd 文件的 `pdf_document` 输出类型是先转换为 LaTeX 文件，再借助 tinytex 扩展包和 LaTeX 编译程序包 TinyTeX 转换为 PDF。rmarkdown 从第 1.9 版本开始不再支持使用本机系统中已有的 LaTeX 编译系统，只允许使用TinyTeX。TinyTeX 优点是直接用 R 命令就可以安装，更新也由 R 自动进行，不需要用户干预。

A.12.1 TinyTex 的安装使用

为了安装 R 的 tinytex 扩展包和单独的 TinyTeX 编译软件，如果已经安装了其它的 LaTeX 编译系统需要先卸载，否则可能有冲突，然后在 R 中运行：

```
install.packages('tinytex')
tinytex::tlmgr_repo('http://mirrors.tuna.tsinghua.edu.cn/CTAN/')
tinytex::install_tinytex()
```

其中上面第一行命令安装 R 的 tinytex 扩展包，第二行将下载 LaTeX 编译程序的服务器设置为清华大学 tuna 镜像站，第三行安装 LaTeX 编译程序。

如果安装成功,TinyTeX 软件包在 MS Windows 系统中一般会安装在 `C:\Users\用户名\AppData\Roaming\TinyTeX` 目录中，其中“用户名”应替换成系统当前用户名。如果需要删除 TinyTeX 软件包，只要直接删除那个子目录就可以。

为了判断 TinyTeX 是否安装成功，在 RStudio 中运行

```
tinytex::is_tinytex()
```

结果应为 `TRUE`, 出错或者结果为 `FALSE` 都说明安装不成功。

当用户使用 RMarkdown 和 tinytex 包转换 latex 并编译为 PDF 时，如果缺少某些 latex 宏包，tinytex 会自动安装缺少的宏包。

因为 TinyTex 与已有的其它 Latex 编译软件会发生冲突，所以安装了 TinyTex 后用户不在 Rmd 中使用 Latex 也需要使用 TinyTex 这个软件。在安装 TinyTex 时已经设置好了程序运行路径 (PATH)，所以在 Texworks 等软件中可以直接调用 xelatex 等编译程序，但是可能会遇到某些 latex 宏包找不到的问题，并且不能自动安装下载这些宏包，需要用户用命令查找并下载安装。

一种安装缺失文件的方式是在 R 中用 tinytex 包搜索编译产生的.log 文件，自动安装缺少的宏包。比如，编译文件为 `C:\user\rep.log` 中，可以用如下命令查找并安装缺失的宏包：

```
library(tinytex)
tlmgr_repo('http://mirrors.tuna.tsinghua.edu.cn/CTAN/')
parse_install("C:/user/rep.log")
```

另一种办法是在 R 中用 tinytex 扩展包直接搜索所需的扩展包然后安装。比如，latex 编译错误信息提示缺少了一个 `dsfont.sty` 宏包，可以在 R 中运行如下命令搜索：

```
library(tinytex)
tlmgr_repo('http://mirrors.tuna.tsinghua.edu.cn/CTAN/')
tlmgr_search("dsfont.sty")
## tlmgr.pl: package repository
##   http://mirrors.tuna.tsinghua.edu.cn/CTAN/systems/texlive/tlnet (verified)
## doublestroke:
##  texmf-dist/tex/latex/doublestroke/dsfont.stytlmgr search --file --global "dsfont.sty"
```

结果中显示了当前使用的下载镜像网站的网址，以及要安装 dsfont.sty，需要安装的是 doublestroke 包。所以：

```
library(tinytex)
tlmgr_repo('http://mirrors.tuna.tsinghua.edu.cn/CTAN/')
tlmgr_install("doublestroke")
## tlmgr install doublestroke
## tlmgr.pl: package repository
##   http://mirrors.tuna.tsinghua.edu.cn/CTAN/systems/texlive/tlnet (verified)
## [1/1, ??:??/??:??] install: doublestroke [66k]
## running mktexlsr ...
## done running mktexlsr.
## running updmap-sys ...
## done running updmap-sys.
## tlmgr.pl: package log updated:
##   C:/Users/user/AppData/Roaming/TinyTeX/texmf-var/web2c/tlmgr.log
```

这样就安装了 dsfont.sty 宏包。

如下的 R 命令可以更新已安装的 latex 宏包：

```
library(tinytex)
tlmgr_repo('http://mirrors.tuna.tsinghua.edu.cn/CTAN/')
tlmgr_update()
```

如果 latex 系统有大幅改版，可能会导致已安装的 tinytex 出错而且不能通过更新安装包纠正，这时可以删除已安装的 tinytex 软件，重新安装。在 MS Windows 系统中 tinytex 一般安装在 C:\Users\用户名\AppData\Roaming\TinyTeX 目录中，其中“用户名”应替换成系统当前用户名。如果需要删除 TinyTeX 软件包，只要直接删除那个子目录就可以。

tinytex 包的 tlmgr_search()、tlmgr_install()、tlmgr_update() 都是 TexLive 软件包中的 tlmgr 命令在 R 中的方便调用方法，可以在命令行直接调用 tlmgr 命令，参考https://www.tug.org/texlive/doc/tlmgr.html。

A.12.2 Rmd 中 Latex 设置

许多 LaTeX 相关的 YAML 元数据不是设置在输出属性中，而是设置为文件的顶级元数据。如

```
---
title: "R 语言简介"
author: " 李东风"
date: "2023-04-26"
output: pdf_document
```

```
fontsize: "12pt"
papersize: "a5"
geometry: "margin=1in"
---
```

其中属性 `fontsize`, `papersize` 和 `geometry` 都是 LaTeX 选项，但是没有作为 `output--pdf_document` 的属性，而是作为 Rmd 文件的顶级元数据属性。使用较小的纸张大小 (如 a5) 和较大的字体 (12pt)，可以使得制作的 PDF 结果容易放大显示，这对于将 PDF 文件传送到屏幕较小的电子书阅读器上或者手机上阅读时比较有用。

用属性 `documentclass` 指定 LaTeX 文档类型，比如 article 是论文格式，book 是图书格式。ctexart 和 ctexbook 是适用于中文的格式，但兼容性不好。

`fontsize` 是正文字体大小，可取 `10pt`, `11pt`, `12pt`，其中 `10pt` 是缺省大小。

`geometry` 用来指定 geometry 包的参数。

默认的 LaTeX 编译引擎是 pdfLaTeX。可以在 `pdf_document` 输出的属性中设置 `latex_engine: xelatex` 将引擎替换成 xeLaTeX。xeLaTeX 的优点是支持 UTF8 编码，可以使用系统中现有字体，而我们在写中文的 R Markdown 文档时必须使用 UTF-8 编码。

Rmd 文件中的文献引用可以由 pandoc 软件执行，但是在用 LaTeX 转换 PDF 时，可以指定用一个 LaTeX 引用包，如：

```
---
output:
  pdf_document:
    latex_engine: xelatex
    citation_package: natbib
---
```

为了自动生成目录并自动为章节编号，仍在 `pdf_document` 的属性中指定 `toc: true` 和 `number_section: true`。

在使用 bookdown 管理 Rmd 文件时，文献引用、目录、章节编号方式有另外的指定办法以及默认规则。

为了能够检查转换过程中的错误，可以用 `pdf_document` 的属性 `keep_tex: true`，保留作为中间结果的.tex 文件，人工检查其中的错误。

可以在 `pdf_document` 输出的属性中指定若干个.tex 文件插入到 LaTeX 文档的导言、正文最前面、正文最后面，如

```
---
output:
  pdf_document:
    latex_engine: xelatex
    keep_tex: true
```

```
    citation_package: natbib
    toc: true
    number_sections: true
    includes:
      in_header: preamble.tex
      before_body: doc-prefix.tex
      after_body: doc-suffix.tex
---
```

preamble.tex 主要用来载入额外的 LaTeX 包以及定义新的命令，一个 preamble.tex 的例子如下

```
\usepackage{ctex}     % 支持中文的标点和章节模式

%\usepackage{xltxtra} % XeLaTeX的一些额外符号

\usepackage{amsthm,mathrsfs} % 基本的ams数学公式
\usepackage{booktabs}        % 增强的表格功能
\usepackage{longtable}       % 支持超过一页的表格

% 下面是自定义的定理环境排版方法
\makeatletter
\def\thm@space@setup{%
  \thm@preskip=8pt plus 2pt minus 4pt
  \thm@postskip=\thm@preskip
}
\makeatother
```

A.13 生成期刊文章

节A.12的办法可以用来生成 LaTeX 然后转换为 PDF，可以用来撰写学位论文和期刊论文。但是学位论文和期刊论文都有比较严格的格式要求，用节A.12的一般方法需要对 LaTeX 比较熟悉才能达到格式要求。

用 R 扩展包 rticles 可以比较容易地将 Rmd 文件转换成期刊可接受的 LaTeX 和 PDF 格式。见https://bookdown.org/yihui/rmarkdown/journals.html。扩展包中对一些常见期刊提供了模板。

首先要安装 rticles 扩展包，命令为

```
install.packages("rticles")
```

为了生成新的论文源文件，在 RStudio 中，用 “File – New File – R Markdown – from Template” 菜单，从列出的文章模板中选一个。

杂志投稿的 Rmd 模板是用许多元数据来规定格式的。用户可以修改其中的标题、作者、作者单位等信息。

因为 rticlels 的输出只有 PDF，所以必要时可以在 Rmd 源文件中直接使用 LaTeX 命令，但如果 Rmd 本身可以完成时尽可能用 Rmd 的功能。

因为定理、图表自动标号和索引功能是由 bookdown 包提供的，而 rticles 的输出类型实际是 `rmarkdown::pdf_docment`，所以可以联合使用 bookdown 与 rticles，如

```
output:
  bookdown::pdf_book
    base_format: rticles::peerj_article
```

其中 `peerj_article` 可以替换成 rticles 提供的其它格式。

如果自己需要的模板 rticles 没有提供，可以自己设法定义模板，参见https://bookdown.org/yihui/rmarkdown/document-templates.html。

A.14 附录：经验与问题

A.14.1 Word 模板制作

如果有合适的 Word 格式样例，可以在 YAML 元数据的 `word_document` 的属性中，设置 `reference_docs` 属性指向该模板文件。

为了制作 Word 模板，不应该从头开始，而是在没有模板的情况下从 Rmd 文件转换生成一个 Word 文件，将其作为模板文件的基础进行修改。在修改过程中，应该制作过程的备份，以便修改错误时能退回上一个版本。

如果文章的大标题样式不是“标题”，将样式改为“标题”以免在后续的自动编号过程中参与编号。修改的方法是选中大标题，然后在 Word 软件主菜单的“开始–样式”选择框中选中“标题”，就可以设置其样式。

Word 的目录是与“标题 1”同级的，这样会在自动编号时参与编号；为此，选中“Contents”，然后在主菜单的“开始–样式”选择框中，找到突出显示的“TOC 标题”，右键单击“修改”，在弹出的对话框中将“样式基准”从“标题 1”改为“标题”，并在同一对话框中将适当调整样式，如字体类型、大小、颜色、是否居中。改完后保存并保存一个副本。

如果需要修改“标题”，“作者”，“日期”，“标题 1”等等的字体、大小、颜色、对齐方式，可以选中该部分，并在“样式”中右键单击相应的样式，选择“修改”进行修改，保存样式文件并试验 Rmd 文件是否正确地利用了修改的样式。

为了实现自动编号，先在正文中选中一个一级标题（不能是目录标题），从 Word 的主菜单栏找到“开始——段落——多级列表——定义新的多级列表”，在打开的对话框中点击“更多”选项，逐一地选择“单击要修改的级别”，将级别 1、2、3 等分别修改，主要是修改“将级别链接到样式”使得”级别 1“与“标题 1”对应，“级别 2”与“标题 2”对应，等等。注意这个修改要一次性完成，不能保存了文件后再次定义新的多级列表。

修改模板文件的页边距也会使得利用其为模板的 Rmd 文件转换结果被修改。为了修改模板文件的页边距，在主菜单“布局——页边距——自定义页边距”弹出的对话框中修改。

生成的目录的标题内容是 “Table of Contents”，目前只能对 Rmd 输出的 docx 文件直接修改，修改模板文件的标题内容并不能使得 Rmd 输出被修改。

参考：Richard Layton 的网站文章 https://rmarkdown.rstudio.com/articles_docx.html。

A.14.2 数学公式设置补充

如果不使用 RStudio, R 扩展包 rmarkdown 的 `markdownToHTML()` 函数可以把含有数学公式的内容转换成可显示公式的 HTML 文件，R 扩展包 knitr 的 `knit2html()` 也可以实现此功能。用 rmarkdown 扩展包的 `markdownToHTML()` 函数生成的含有数学公式的内容。

单独使用 pandoc 软件时，也可以用普通文本编辑器在 Markdown 文件中输入 LaTeX 格式的数学公式，然后用 pandoc 软件转化为带有数学公式的 docx 文件，不需要额外选项。但是，为了能够在转换的 HTML 文件中正常显示数学公式，需要在运行 Pandoc 时增加运行选项`--maxjax`，如

```
pandoc --mathjax -s -o test.html test.md
```

经试验，这样转化的 HTML 文件可以在 Firefox 以及 Microsoft 的 Edge 浏览器和 Internet Explorer 浏览器中正常显示数学公式。如果公式中的中文显示不正常，对显示的公式右键单击弹出选项菜单，选择 “Math Settings–Math Renderer–SVG” 或者 “HTML-CSS”。

附录 B

用 bookdown 制作图书

B.1 介绍

R 的 bookdown 扩展包 (https://github.com/rstudio/bookdown) 是继 knitr 和 rmarkdown 扩展包之后，另一个增强 markdown 格式的扩展，使得 Rmd 格式可以支持公式、定理、图表自动编号和引用、链接，文献引用和链接等适用于编写书籍的功能。在 bookdown 的管理下一本书的内容可以分解成多个 Rmd 文件，其中可以有可执行的 R 代码，R 代码生成的文字结果、表格、图形可以自动插入到生成的内容中，表格和图形可以是浮动排版的。输出格式主要支持 gitbook 格式的网页图书，这种图书在左侧显示目录，右侧显示内容，并可以自动链接到上一章和下一章；通过单独安装的 LaTeX 编译器支持将书籍转换为一个 PDF 文件，支持中文；可以生成 ePub 等格式的电子书。

主要用于编写有多个章节的书籍，也可以用来生成单一文件的研究报告。

建议使用 RStudio 集成环境制作这样的图书，该软件内建了一键编译整本书的功能。需要安装 bookdown 扩展包的最新版本。bookdown 扩展包现在还比较新，还有一些 BUG，所以尽可能使用最新版的 bookdown 扩展包并且及时更新 RStudio 软件。查看编译的网站建议使用 Google Chrome 浏览器，此浏览器对 gitbook 的支持较好。

为了新写一本书或者从已有的书转换，最简单的做法是从 bookdown 的网站下载 bookdown 配套的例书的 zip 文件 (见https://github.com/rstudio/bookdown-demo)，将其解压到本地硬盘某个子目录，然后修改其中的内容适应自己的书的需要。

因为中文需要一些特殊的设置，以及在网络条件不好的条件下支持数学公式显示，本书作者提供了一个粗浅的中文书 bookdown 模板，下载链接为:

- `bookdown-template-v0-6.zip`。

其中的 CBook 子目录包含了所需的中文书模板，CArticle 子目录包含了论文格式模板，其它子目录有一些别的模板，为了在本地支持网页中的数学公式显示还有一个 MathJax 目录。参见其中的 readme.txt 说明文件。

B.2 一本书的设置

一本用 bookdown 管理的书，一般放置在某个子目录下，并作为一个 RStudio 项目 (project) 用 RStudio 管理。

也可以自己新建一个目录，然后编辑生成必要的文件。注意，所有的文本文件都要使用 UTF-8 编码。一本 bookdown 书，一般都需要有一个 `index.Rmd` 文件，这是最后生成的网站的主页的原始文件，可以在这个文件中写一些书的说明，并在开头的 YAML 元数据部分进行有关设置，如标题、作者、日期等。`index.Rmd` 的一个例子如下：

```
---
title: " 统计计算"
author: " 李东风"
date: `r Sys.Date()`
site: bookdown::bookdown_site
output: bookdown::gitbook
documentclass: book
bibliography: [myrefs.bib, ../docs/refs.bib]
biblio-style: apa
link-citations: true
description: " 本科生《统计计算》教材。采用 R 的 bookdown 制作，输出格式为 bookdown::gitbook."
---
```

```
# 前言 {-}

统计计算研究如何将统计学的问题用计算机正确、高效地实现。
```

其中在三个减号组成的两行之间的内容叫做 YAML 元数据，是一本书的设置，上例中有书的标题、作者名、日期（用 R 程序自动生成）、描述。其中的 `site` 选项很重要，一定要有这个选项，`site: bookdown::bookdown_site` 使得 RStudio 软件能辨认这是一个 bookdown 图书项目，从而为其提供一键编译快捷方式。元数据中 `output` 项指定默认的输出格式。`documentclass` 项为借助 LaTeX 编译 PDF 格式指定 LaTeX 的模板，现在还不能支持 ctexbook 模板所以使用了 book 模板。`bibliography` 项指定一个或者几个.bib 格式的文献数据库。

一个 bookdown 图书项目除了 `index.Rmd` 文件之外，一般还应该有一个 `_bookdown.yml` 文件存放与整本书有关的 YAML 元数据。例如

```
new_session: true
book_filename: 'statcompc'
language:
  label:
    thm: '定理'
    def: '定义'
    exm: '例'
    proof: '证明: '
```

```
    solution: '解: '
    fig: '图'
    tab: '表'
  ui:
    chapter_name: ''
delete_merged_file: true
```

其中 `new_session: true` 设置很重要，这使得每一个 Rmd 文件中的 R 程序都在一个单独的 R 会话中独立地运行，避免了不同 Rmd 文件之间同名变量和同名标签的互相干扰。`book_filename` 是最终生成的 LaTeX PDF 图书或者 ePub 电子书的主文件名。`language` 下可以定制一些与章节名、定理名等有关的名称。

另外一个需要的设置文件是 `_output.yml` 文件，用于输出格式的设置。这部分内容也可以包含在 `index.Rmd` 的元数据中 `output` 条目下面。内容如

```
bookdown::gitbook:
  includes:
    in_header: mathjax-local.html
  config:
    toc:
      before: |
        <li><a href="http://www.math.pku.edu.cn/teachers/lidf/course/statcomp/_book/index.html"> 统计
      after: |
        <li><a href="http://www.math.pku.edu.cn/teachers/lidf/" target="blank"> 编著: 李东风 </a></li>
    download: ["pdf"]
bookdown::pdf_book:
  includes:
    in_header: preamble.tex
  latex_engine: xelatex
  citation_package: biblatex
  keep_tex: yes
bookdown::epub_book: default
```

其中的 `style.css` 是自定义的 CSS 显示格式，可以去掉这一行，使用默认的 CSS 格式。这个例子文件分为三部分，gitbook、pdf_book 和 epub_book 三种输出格式分别设置了一些输出选项。在 gitbook 部分，设置了目录上方显示的书的主页的链接（`before` 项）和目录下方显示的作者信息。在 `in_header` 部分插入了一部分个性化的 HTML 代码，这部分代码是使用本地的数学公式显示支持以免外网不通时数学公式不能显示，插入的内容将出现在每个生成的 HTML 文件的 head 部分。

在 `pdf_book` 部分，设置了通过 LaTeX 编译整本书为 PDF 的一些选项。指定了 `latex_engine` 为 `xelatex`，这对中文支持很重要。`in_header` 选项要求在 LaTeX 文件导言部分插入一个 `preamble.tex` 文件，内容如：

```
\usepackage{ctex}
```

```
\usepackage{amsthm,mathrsfs}
\usepackage{booktabs}
\usepackage{longtable}
\makeatletter
\def\thm@space@setup{%
  \thm@preskip=8pt plus 2pt minus 4pt
  \thm@postskip=\thm@preskip
}
\makeatother
```

其中很重要的是使用 ctex 包来支持中文。在 index.Rmd 的元数据中也可以指定一些 LaTeX 选项，比如指定页边距等（这些设置有些是非标准的，一般不需要）：

```yaml
classoption: twoside
fontsize: 12pt
linestretch: 1.5
geometry: "left=4cm, right=3cm, top=2.5cm, bottom=2.5cm"
fontsize: 12pt
linestretch: 1.5
toc-depth: 1
lof: True
lot: True
```

在 bookdown 项目中与 `index.Rmd` 同级的所有.Rmd 文件都自动作为书的一章，除非文件名以下划线开头。这样做的好处是作者可以任意地增删章节，编译整本书时章节编号会自动调整。但是，章节的顺序将按照文件名的字典序排列，所以，所有的包含一章内容的.Rmd 文件，最好命名为类似 `0201-rng.Rmd` 这样的名字，文件名前面人为地加上排序用的序号，使得章节按照自己的次序排列。实际上，也可以设置不自动将每个.Rmd 文件都作为一章，而是在 `_output.yml` 中设置一项 `rmd_files`，列出所有需要作为一章的文件，并以列出次序编译，如

```yaml
rmd_files: ["index.Rmd", "rng.Rmd", "simulation.Rmd", "refs.Rmd"]
```

这时，应该添加一个 `_site.yml` 文件，内容如：

```yaml
site: "bookdown::bookdown_site"
output: bookdown::gitbook
```

B.3 章节结构

除了 `index.Rmd` 文件，项目中每个.Rmd 文件都作为一章。每个.Rmd 文件第一行，应该是以一个井号和空格开头的一级标题，后面再加空格然后有大括号内以井号开头的章标签，如

```
# 随机数 {#rng}
```

这些章标签去掉井号后会作为生成的 HTML 文件的名字，所以一定要有章标签，而且章节标签在全书中都不要重复以免冲突。文件内可以用两个井号和一个空格开始的行表示节标题，最后也应该有大括号内以井号开头的节标签，如

```
# 随机数 {#rng}

## 均匀随机数发生器 {#rng-unif}
```

使用 bookdown 写书，一般每章不要太长，否则编译预览很慢，读者浏览网页格式也慢。

内容相近的章节可以作为一个“部分”。为此，在一个部分的第一个章节文件的章标题前面增加一行，以 `# (PART)` 开头，以 `{-}` 结尾，中间是部分的名称，如

```
# (PART) 随机数和随机模拟 {-}

# 随机数 {#rng}
```

书的最后可以有附录，附录的章节将显示为 `A.1`, `B.1` 这样的格式。为此，在附录章节的第一个文件开头加如下的第一行标题行：

```
# (APPENDIX) 附录 {-}

# 一些定理的证明 {#formula}
```

B.4 书的编译

建议使用 RStudio 软件编辑内容，管理和编译整本书。

在 `index.Rmd` 或者 `_bookdown.yml` 中设置 `site: bookdown::bookdown_site` 后，RStudio 就能识别这个项目是一个 bookdown 项目，这时 RStudio 会有一个 Build 窗格，其中有“Build book”快捷图标，从下拉菜单中选择一个输出格式（包括 gitbook、pdf_book、epub_book），就可以编译整本书。对 gitbook 格式，即 HTML 网页格式，编译完成后会弹出一个预览窗口，其中的“Open in Browser”按钮可以将内容在操作系统默认的网络浏览器中打开。

另一种办法是在命令窗口用如下命令编译（以输出 gitbook 为例），我个人认为这种办法更好用：

```
bookdown::render_book("index.Rmd",
  output_format="bookdown::gitbook")
```

编译结果默认保存在 `_book` 子目录中，可以在 `_bookdown.yml` 中设置 `output_dir` 项改为其它子目录。编译整本书为 pdf_book 格式时，如果成功编译，也会弹出一个 PDF 预览窗口。可以在 `_book` 子目录中找到这个 PDF 文件。

将书编译为 PDF 需要利用 LaTeX 编译器，这需要单独安装 LaTeX 编译软件，LaTeX 编译器对输入要求十分严格，一丁点儿错误都会造成整本书的编译失败，所以对于不熟悉 LaTeX 的用户，不建议使用 bookdown 的 pdf_book 输出格式。当前的 R Markdown 仅支持谢益辉的TinyTeX软件，安装和管理参见A.12.1。

对于较短的书，做了一定修改后都可以重新编译 gitbook 结果和 pdf_book 结果。在书比较长了以后，每次编译都花费很长时间，所以可以仅编译 gitbook 格式的一章，修改满意后再编译整本书。仅编译一章也需要所有的.Rmd 文件都是已经编译过一遍的，新增的 Rmd 文件会使得编译单章出错，每次新增了 Rmd 文件都应该重新编译整本书，但是内容修改后不必要重新编译整本书，可以仅编译单章。

编译单章现在没有快捷图标，只能在 RStudio 控制台（命令行）运行如下命令：

```
bookdown::preview_chapter("chap-name.Rmd",
  output_format="bookdown::gitbook")
```

其中 `chap-name.Rmd` 是要编译的单章的文件名。编译完成后在结果目录（默认是 `_book`）中找到相应的 HTML 文件打开查看，再次编译后仅需在浏览器中重新载入文件。建议使用 Google chrome 浏览器，用 MS IE 或者 Edge 浏览器对 gitbook 的 Javascrpt 支持不够好，使得目录的层级管理、自动滚动、单章编译后的目录更新不正常，而 chrome 则没有问题。

编译单章也不能解决所有的问题，有些问题还是需要编译整本书，而章节很多时整本书编译又太慢。为此，可以在项目中增加一个临时的部分内容子目录，如 `testing` 子目录，在子目录中存放相同的设置文件 `index.Rmd`、`_bookdown.yml`、`_output.yml`，以及图形文件、文献数据库文件，并将要检查的若干章节复制到 `testing` 子目录中，在 `testing` 中新建一个 bookdown 项目，然后编译其中的整本书。这在调试部分章节的 HTML 和 PDF 输出时很有效。解决问题后只要将修改过的章节复制回原始的书的目录中。

有时仅仅想验证某个长数学公式或者表格，用上述的编译单章或者单独一个小规模测试项目的办法也不经济。这时，单独开一个备用的普通 RStudio 项目，不能是 bookdown 项目，在其中的 Rmd 文件中验证数学公式和表格的编排，这样最为方便。

有时需要根据编译输出目标采用不同的设置，比如，输出为 html 时可以使用 HTML 插件 (widgets)，而输出到 PDF 则不可以。可以用 `knitr::opts_knit$get("rmarkdown.pandoc.to")` 感知当前编译过程的目标，对 HTML 类，输出为`"html"`，对 PDF，输出为`"latex"`。

B.5 交叉引用

在写作时，每个一级到三级标题都应该有自定义的标签，格式是在标题行末尾空格后添加 `{#label}`，其中 label 是自己指定的标签，使用英文、数字、减号，不要使用中文，而且整本书不要有重复的标签。为避免不同章节使用了重复标签，可以取 `label` 的前一部分为所在章节的文件名。

如果要引用某一章节，有如下的做法：

- §\@ref(label)，label 是某个标题对应的标签。结果显示为如 §3.1.1 这样的章节号，并可点击，点击时跳跃到相应的章节。
- [链接文本](#label) 其中 label 是某个标题对应的标签。结果产生一个链接，显示为链接文本，点击时跳到 label 对应的章节。

有时需要在数学公式内部、图形的说明中引用定理编号、公式编号等，这需要预先定义单独的文字性引用，然后再使用该文字引用。定义文字引用如：

可以在数学公式内部、图形说明的字符串内用\texttt{(ref:mytextlabel)}的格式调用上面定义的文字引用。目前在数学公式内部使用有BUG。

```
\hypertarget{bookdown-formula}{%
\section{数学公式和公式编号}\label{bookdown-formula}}

通过R的knitr和rmarkdown扩展包以及pandoc软件，
.Rmd格式文件已经支持数学公式，
见\href{rmarkdown.html}{R Markdown说明}。

在用\texttt{\$\$}符号在两端界定的公式后面，
可以用\texttt{\textbackslash{}tag\{标号\}}命令增加人为的公式编号，如

\begin{verbatim}
$$
y = f(x)
\tag{*}
$$
```

结果显示为

$$y = f(x) \tag{*}$$

要注意的是，在 $$ 界定的数学公式内用了 aligned 环境后，仅能在\end{aligned} 之后加\tag{标号} 命令，而不能写在 aligned 环境内。这样，多行的公式将不能为每行编号。

用\tag 命令人为编号比较简单易用，但是在有大量公式需要编号时就很不方便，只要增加了一个公式就需要人为地重新编号并修改相应的引用。bookdown 包支持对公式自动编号，并可以按公式标签引用公式，引用带有超链接。

bookdown 的自动编号对 LaTeX 的 equation 环境、align 环境都可以使用，而且不需要在两端用 `$$` 界定。在公式的末尾或者一行公式的\\换行符之前，写 (\#eq:mylabel)，其中 `mylabel` 是自己给公式的文字标签，文字标签可以使用英文字母、数字、减号、下划线。如

```
\begin{align}
f(x) =& \sum_{k=0}^\infty \frac{1}{k!} x^k \label{eq:efunc-sum} \\
  = e^x \label{eq:efunc-ex}
\end{align}
```

将会对两行公式自动编号。引用公式时，用如\@ref(eq:mylabel)，其中 `mylabel` 是公式的自定义标签，编译后这样的引用会变成带有链接的圆括号内的编号。

公式编号在全书中都不要有冲突（不同的公式定义了相同的编号）。一种办法是，自定义的公式标签的开头以章节文件名开头。

B.6 定理类编号

定理、引理、命题、例题等，使用特殊的 markdown 代码格式，以三个冒号开头，以三个冒号结尾，在开头的三个反单撇号后面空格后写 `{.theorem}` 表示定理。在`.theorem` 后面，可以用空格分隔后写一个定理的自定义标签，标签以 `#` 开头，由字母、数字、减号组成，`#` 号作为标签的开头标志但不作为标签的一部分。可以用 `name="定理名称"` 指定一个显示的定理名。

设某个定理的自定义标签是 `#mythlabel`，则可以用如\@ref(thm:mythlabel) 引用此定理的编号，编号是在每一章内从头编号的。编号有自动生成的链接。

例如：

```
::: {.theorem #norlim-weakconv name="弱收敛"}
$\xi_n$依分布收敛到$\xi$，
当且仅当对任意$\mathbb R$上的一元实值连续函数$f(\cdot)$都有
$$
  E f(\xi_n) \to E f(\xi), \ n \to \infty .
$$
:::
```

当定理或例子内有列表时，一定注意列表前后要空行，否则会导致嵌套错误。这种错误在编译 HTML 时无法发现，但是会造成结果莫名其妙地出错。

bookdown 提供了证明环境，但是不太实用。

对例题，将 `theorem` 替换成 `example`，在引用时将 `thm` 替换成 `exm`。如：

```
::: {.example #noralim-aspr-ce name="依概率收敛不a.s.收敛反例"}
a.s.收敛推出依概率收敛，
但是反之不然。给出反例。
:::
```

定理类的段落包括如下的种类：

表 B.1: 定理类段落

环境名	默认显示名	标签前缀
theorem	Theorem	thm
lemma	Lemma	lem
corollary	Corollary	cor
proposition	Proposition	prp
conjecture	Conjecture	cnj
definition	Definition	def
example	Example	exm
exercise	Exercise	exr

其中的显示名可以在 `_bookdown.yml` 的 `language` 的 `label` 属性中修改，见B.2。

B.7 文献引用

bookdown 使用.bib 格式的文献数据库，关于.bib 格式的文献数据库请参考 LaTeX 的有关说明。在 `index.Rmd` 的 YAML 元数据部分或者 `_bookdown.yml` 中用 `bibliography` 可以设置使用的一个或者多个.bib 格式的文献数据库文件。设某篇文章的.bib 索引键是 `Qin2007:comp`，用 `@Qin2007:comp` 可以引用此文献，用 `[@Qin2007:comp]` 可以生成带有括号的引用。

指定.bib 文件时可以用相对路径，如 "`../docs/mybib.bib`"。有多个文件时，写在方括号 `[]` 中并用逗号分隔，如 `bibliography: [mybib.bib, ../docs/refs.bib]`。

为了使得产生的文献引用包括跳转超链接，可以在 `index.Rmd` 的元数据部分添加 `link-citations: true` 设置。

为了生成某个 R 扩展包的.bib 格式的引用，可以用 R 程序制作，例如：

```
toBibtex(citation("MASS"))
## @Book{,
##   title = {Modern Applied Statistics with S},
##   author = {W. N. Venables and B. D. Ripley},
##   publisher = {Springer},
##   edition = {Fourth},
```

```
##   address = {New York},
##   year = {2002},
##   note = {ISBN 0-387-95457-0},
##   url = {https://www.stats.ox.ac.uk/pub/MASS4/},
## }
```

下面是一个样例.bib 文件的内容（注意要用 UTF-8 编码保存）：

```
% Encoding: UTF-8

@Book{MWP06-HighStat,
  author    = {茆诗松 and 王静龙 and 濮晓龙},
  title     = {高等数理统计},
  year      = {2006},
  edition   = {第二版},
  publisher = {高等教育出版社}
}


@BOOK{Unwin-Visualize06,
  title = {Graphics of Large Datasets Visualizing a Million},
  publisher = {Springer},
  year = {2006},
  author = {Antony Unwin and Martin Theus and Heike Hofmann}
}


@Article{Wichmann1982:RNG,
  author  = {Wichmann, B. A. and Hill, I. D.},
  title   = {Algorithm as 183: An efficient and portable pseudo-random number generator},
  journal = {Applied Statistics},
  year    = {1982},
  volume  = {31},
  pages   = {188 – 190. Remarks: 34, 198 and 35, 89},
}

@Book{QGCZ2011:StochProc,
  author    = {钱敏平 and 龚光鲁 and 陈大岳 and 章复熹},
  title     = {应用随机过程},
  year      = {2011},
```

```
  publisher = {高等教育出版社},
}
```

B.8 插图

bookdown 图书的插图有两种，一种是已经保存为图形文件的，主要是 png、jpg 和 pdf 图片；另一种是文中的 R 代码生成的图形。

已经有图形文件的，可以用 markdown 格式原来的插图方法，见markdown 格式介绍。但是，这样做不能给图形自动编号，另外因为制作图书是有网页和 PDF 书两种主要输出格式的，原有的插图方式在这两种输出格式上有细微的不一致。所以，最好是统一使用 Rmd 的插图方法。

Rmd 的插图方法就是写一段 R 代码段来插图，如果是用程序作图，则代码中写作图的代码；如果是已有的图形文件，可以在一个单独的 R 代码段中用类似下面的命令插图：

```
knitr::include_graphics("figs/myfig01.png")
```

其中 `figs` 是存放图形文件的子目录名，`myfig01.png` 是要插入的图形文件名。这样，如果同时还有 `myfig01.pdf` 的话，则 HTML 输出使用 png 图片而 PDF 输出自动选用 pdf 文件。另外，插图的选项在代码段的选项中规定：用代码段的 `fig.with` 和 `fig.height` 选项指定作图的宽和高（英寸），用 `out.width` 和 `out.height` 选项指定在输出中实际显示的宽和高，实际显示的宽和高如果使用如`"90%"` 这样的百分数单位则可以自动适应输出的大小。

为了使得插图可以自动编号并可以被引用，为代码段指定标签并增加一个 `fig.cap="..."` 选项指定图形标题。代码段的标签变成浮动图形的标签，如 `myfiglabel`，则为了引用这个图只要用`\@ref(fig:myfiglabel)`。注意，在整本书中这些标签都不能重复，否则编译 LaTeX 支持的 PDF 输出会失败。

有些插图会伴随很长的说明文字，这可以用代码段的 `fig.cap=` 选项指定，但是其中的 Markdown 特有的格式在转换 LaTeX 时不一定支持，而且在代码段选项中写太长的文字说明也是的程序难以辨认。所以，可以使用文字引用的方式：在单独的一段中，用如下格式定义一段可引用的文字内容：

```
其中\texttt{mylabel}是自己定义的仅由英文大小写字母、数字和减号组成的引用标志符。在需要使用这段文字的
注意定义和引用都是用的\texttt{(ref:mylabel)}语法。

用R生成PDF格式的图形时，
需要指定中文作为\texttt{family}选项，
所以在每个源文件的开头应该加上如下的设置，
使得生成PDF图时中文能够正确显示：

\begin{verbatim}
```

```{r setup-pdf, include=FALSE}
pdf.options(family="GB1")
```

其中 `include=FALSE` 表示要不显示代码段的代码，有运行结果也不插入到输出结果中，是否允许运行视缺省的 `eval=` 的值而定。

B.9 表格

B.9.1 Markdown 表格

bookdown 书的表格也有两种，一种是原来 markdown 格式的表格，最好仅使用管道表，管道表对中文内容支持最好。为了对这样的表格自动编号，需要在表格的前面或者后面空开一行的位置，写

`Table: \label{tab:mylabel} 表的说明`

其中 `mylabel` 是自定义的表格标签。在引用这个表时用如 `\@ref(tab:mylabel)`。

B.9.2 用 `kable()` 函数制作表格

另一种表格是 R 代码生成的表格，主要使用 `knitr::kable()` 函数。在 `knitr::kable()` 函数中用选项 `caption=` 指定表格的说明文字（标题），这时生成表格的 R 代码段的标签，如 `myfiglab`，就自动构成了表格的引用标签主干，实际引用如 `\@ref(tab:myfiglab)`。

加选项 digits，可以指定小数点后的数字位数。不同列使用不同位数时，可以指定一个包含各列小数点后位数的向量。

为了适应较长的表，可以在 LaTeX 的 `preamble.tex` 中引入 longtable 包，并在 `knitr::kable()` 中加选项 `longtable=TRUE`。

可以设置 `kable()` 显示缺失值的方式，如；

```
options(knitr.kable.NA = "")
```

则缺失值显示成空单元。默认显示是 `NA`。

B.9.3 R 中其它制作表格的包

如果要对表格进行更丰富的定制，可以使用kableExtra 包。

flextable 和 fluxtable 支持较多的输出格式和常见表格功能。

还有许多扩展包，不一一列举。

B.10 数学公式的设置

bookdown 在生成 PDF 时使用 LaTeX 软件，所以 PDF 输出的数学公式的支持很好，但是 LaTeX 编译器也很挑剔，稍微一点错误也造成编译失败。比如，在行内公式内部如果紧邻 `$` 符号有多余的空格，如 `$ y $`，编译 PDF 时会出错。

bookdown 生成的 gitbook 格式的网页书籍，在有数学公式时，使用 MathJax 库在浏览器中显示数学公式。MathJax是用于网络浏览器中显示数学公式的优秀的 Javascript 程序库，可免费使用。但是，当数学公式中含有中文（用`\text{}` 或`\mbox{}` 命令）时，数学公式可能会显示不正常。另外，数学公式默认使用远程服务器上的 MathJax 程序库处理，在网络不通畅时显示很慢或者无法显示。bookdown 中使用 MathJax 的版本 2，但 MathJax 已经升级到版本 3，版本 3 有很大改动，暂时还应该使用版本 2 以避免冲突。

为了避免远程调用 MathJax 程序库的麻烦，改为本地使用。将 MathJax 安装在了书生成的网站主目录的上三层，用`../../../MathJax/mathjax.js` 路径访问。假设下载整个 MathJax 库后解压放在了书的网页文件所在子目录的上三层的位置。

增加设置文件 `_header.html`：

```
<script type="text/x-mathjax-config">
MathJax.Hub.Config({
  jax: ["input/TeX","output/SVG"],
  extensions: ["tex2jax.js","MathMenu.js","MathZoom.js"],
  TeX: {
    extensions: ["AMSmath.js","AMSsymbols.js","noErrors.js","noUndefined.js"]
  }
});
</script>
<script type="text/javascript"
   src="../../../MathJax/MathJax.js">
</script>
```

如果希望用远程服务器上的 MathJax，可以使用如下的 `mathjax-cdnjs.html` 设置文件：

```
<script type="text/x-mathjax-config">
MathJax.Hub.Config({
  jax: ["input/TeX","output/HTML-CSS"],
  extensions: ["tex2jax.js","MathMenu.js","MathZoom.js"],
  TeX: {
    extensions: ["AMSmath.js","AMSsymbols.js","noErrors.js","noUndefined.js"]
  }
});
</script>
```

```
<script type="text/javascript"
   src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.7.9/MathJax.js">
</script>
```

为了调用这样的设置文件，在 `_output.yml` 设置文件中如下设置：

```
bookdown::gitbook:
  css: style.css
  includes:
    in_header: _header.html
```

注意，MathJax 要求先设置再调入。由于浏览器对 MathJax 输出方式有记忆，所以如果不能正常显示中文公式，需要右键单击公式，选择 “Math Settings—Math Renderer” 中的 “SVG” 或者 “HTML-CSS”。

B.11 使用经验

B.11.1 学位论文

Ed Berry 在网上分享了用 bookdown 生成 PDF 学位论文的经验：https://eddjberry.netlify.com/post/writing-your-thesis-with-bookdown/ 。Chester Ismay 提供了一个 R 扩展包 thesisdown，见 https://github.com/ismayc/thesisdown，例子见https://thesisdown.netlify.com/。

B.11.2 LaTeX

有数学公式时，对行内公式边缘处多余的空格十分敏感，所以行内公式边缘不要有空格。

如果已经有用 LaTeX 写的书，要转换为 bookdown 的 Rmd 格式，可以用 RStudio 的支持 RegEx 的替换模式，如

- `\\keyword{([^}]+?)}` 替换成 `**\1**`。
- `\\textbf{([^}]+?)}` 替换成 `**\1**`。
- `\\texttt{([^}]+?)}` 替换成`\1`。

可以写一个函数对 LaTeX 文件进行转换生成 Rmd 文件，再手工修改。函数如

```
latex2rmd <- function(fname="_tmp/tomd.tex", encoding="UTF-8"){
  lines <- readLines(fname, encoding=encoding)
  if(encoding != "UTF-8")
    lines <- iconv(lines, from=encoding, to="UTF-8")
  print(head(lines, 10))
```

```
  lines <- gsub("\\\\keyword[{]([^}]+?)[}]", "**\\1**", lines, perl=TRUE)
  lines <- gsub("\\\\textbf[{]([^}]+?)[}]", "**\\1**", lines, perl=TRUE)
  lines <- gsub("\\\\texttt[{]([^}]+?)[}]", "`\\1`", lines, perl=TRUE)
  lines <- gsub("\\\\label[{](eq:[^}]+?)[}]", "(\\\\#\\1)", lines, perl=TRUE)
  lines <- gsub("\\\\eqref[{]([^}]+?)[}]", "\\\\@ref(\\1)", lines, perl=TRUE)
  lines <- gsub("\\\\bs\\\\", "\\\\boldsymbol\\\\", lines, perl=TRUE)
  lines <- gsub("\\\\bs ", "\\\\boldsymbol ", lines, perl=TRUE)
  lines <- gsub("\\\\Rbb", "\\\\mathbb R", lines, perl=TRUE)
  lines <- gsub("\\\\Zbb", "\\\\mathbb Z", lines, perl=TRUE)
  lines <- gsub("\\\\Var[(]", "\\\\text\\{Var\\}(", lines, perl=TRUE)
  lines <- gsub("\\\\Cov[(]", "\\\\text\\{Cov\\}(", lines, perl=TRUE)
  lines <- gsub("(\\\\S)(\\d)", "§\\2", lines, perl=TRUE)
  lines <- gsub("(\\\\S) ", "§ ", lines, perl=TRUE)
  lines <- gsub("(\\\\defeq)", "\\\\stackrel{\\\\triangle}{=}", lines, perl=TRUE)
  lines <- gsub("\\\\argmin_", "\\\\mathop{\\\\text{argmin}}_", lines, perl=TRUE)
  lines <- gsub("^\\s*\\\\item", "*", lines, perl=TRUE)
  lines <- gsub("\\\\begin[{]frame[}]", "", lines, perl=TRUE)
  lines <- gsub("\\\\end[{]frame[}]", "", lines, perl=TRUE)
  lines <- gsub("\\\\begin[{]itemize[}]\\[<\\+->\\]", "", lines, perl=TRUE)
  lines <- gsub("\\\\end[{]itemize[}]", "", lines, perl=TRUE)
  lines <- gsub("\\\\begin[{]itemize[}]", "", lines, perl=TRUE)
  lines <- gsub("\\\\bm ", "\\\\boldsymbol ", lines, perl=TRUE)
  lines <- gsub("\\\\bm\\\\", "\\\\boldsymbol\\\\", lines, perl=TRUE)
  lines <- gsub("\\\\tr[(]", "\\\\text{tr}(", lines, perl=TRUE)
  lines <- gsub("\\begin{align*}", "$$\\begin{aligned}", lines, fixed=TRUE)
  lines <- gsub("\\end{align*}", "\\end{aligned}$$", lines, fixed=TRUE)

  print(head(lines, 20))
  ##writeLines(lines, "_tmp/fromtex-clean.Rmd")
  con1 <- file("_tmp/fromtex-clean.Rmd", "wt", encoding="UTF-8")
  writeLines(lines, con=con1)
  close(con1)
}
```

B.11.3 算法

Rmd 格式对算法编排的支持不够好。编排算法可以用表格来换行，仍用`\qquad` 和`\qquad` 缩进，不要用空格缩进，因为在 LaTeX 转 PDF 时空格会损失。但是算法内容中有公式含有竖线时还是无法与表格线区分开来。

编排算法也可以用数学公式，写在 `$$` 界定范围内，用 aligned 环境分行，缩进使用`\quad` 和`\qquad`。只是无法对

`if` 这样的关键字用重体排印。

B.11.4 中文乱码

为了能够生成中文的 PDF，不要指定 documentclass 为 ctexbook，而是指定为 book，然后在 `preamble.tex` 中引入 `ctex` 包。

为了 PDF 输出，不要引入太多的数学包，因为从 markdown 到 HTML 不支持复杂的数学。

用 R 作图时如果图形中有汉字，在代码块选项中加上 `dev="png", dpi=300`。否则生成 PDF 时会有中文编码问题。另一办法是在每个.Rmd 文件开头的 setup 源代码段插入

```
pdf.options(family="GB1")
```

这样可以生成支持中文字的 PDF 图形。

在编译出错时，会在主目录留下编译的 tex 源文件及相关文件。但是，此 tex 源文件中使用的 R 生成的图片路径不对，需要将 tex 源文件复制到 `_bookdown_files` 目录，将直接插入的图片也复制到这个目录，然后编译 tex 文件发现问题，逐个修复。建议建立小的测试项目专门调试有问题的文件。

B.11.5 图片格式

Bookdown 在网页和 PDF 输出格式中都支持 PNG，所以这是应尽量采纳的格式。也支持 JPEG 和 PDF 格式，但网页中可能无法显示 PDF 格式。支持 SVG 格式，但仅在网页中可显示。

对于少量的图形文件，可以安装一个图像编辑软件 GIMP，这是一个自由软件，与 PhotoShop 功能类似。比如，要将一个 SVG 文件转换为 PNG，可以在 GIMP 中打开，打开时可以选一个放大比例，如 2 倍，然后调用文件菜单的导出功能，将扩展名改为 ".png" 就可以保存成 PNG。

如果有大批图片格式要转换，可以安装 ImageMagick 软件。可以用命令行命令转换，这样可以编程生成一系列的转换命令进行批量转换。在 Windows 的命令行窗口运行转换命令如：

```
magick test.svg -resize 300% test.png
```

产生批量转换命令如：

```
setwd("d:/tmp")
li = list.files(pattern="[.]svg$")
for (fi in li){
  fn = gsub("(.*)[.]svg$", "\\1.png", fi)
  cmd = paste("magick", fi, "-resize 300%", fn)
  cat(cmd, "\n")
}
```

B.11.6 其它经验

连分数的加号是`\genfrac{}{}{0pt}{}{}{+}`。

B.12 bookdown 的一些使用问题

- 在数学公式中用`\text{\@ref(eq:label)}`引用公式，HTML 成功，LaTeX 版本会有 BUG，重复了`\`。如果使用文内的链接，则 LaTeX 成功，HTML 不成功。
- YAML 部分有的中文文字（如作者名）会出错，代以 ASCII 则不出错。2020-06-05 发现此问题已消失。
- example 的自动编号有问题，不加 label 的 example，其 HTML 结果在不同章之间会编号混淆，避免问题的临时办法是 example 都加上 label。
- 用本地文件格式提供的下载文件不能用中文文件名。
- 这种在一本书末尾生成参考文献列表的办法，可能造成 “file name too long” 错误。2020-06-05 发现此问题已消失。
- BUG: 编译时出现 “file name too long” 错误，可能原因：
  - 章节名称必须有英文、数字、减号的标签。
  - 用行尾标记 `{-}` 的方法取消章节进入目录时，中文标题会出错，改用英文标题。`# (PART)` 部分名 `{-}` 也是如此，需要用英文名。

  2020-06-05 发现此问题已消失。

附录 C

用 R Markdown 制作简易网站

C.1 介绍

为了从多个.Rmd 源文件制作网站，可以使用 blogdown 扩展包或者 bookdown 扩展包。bookdown 扩展包可以生成 gitbook 格式的网站，带有左侧的章节目录和前后页面的导航链接，很适用于一本书的网站。但是，bookdown 使用时需要重新编译整个网站才能产生正确的链接。

也可以仅利用 rmarkdown 包的 `render.site()` 功能制作简易的网站。

C.2 简易网站制作

一种简易的制作网站的办法是仅仅利用 rmarkdown 包的 `render.site()` 功能。为了使得 rmarkdown 和 RStudio 将一个子目录（项目）看成一个网站项目，只要此项目中含有 `index.Rmd` 和 `_site.yml` 文件。所有网页.Rmd 源文件都必须在项目的顶级目录中。只要将编译所得的网站目录上传到任意的网站服务器就可以发布到网上。

C.2.1 网站结构

`index.Rmd` 是主页内容，可以在此处人工加入其它页面链接（参见 markdown 格式说明中链接写法），也可以制作单独的目录页面。内容如

```
---
title: " 概率论"
---
```

```
* [概率分布](dist.html)
* [期望](expect.html)
```

`_site.yml` 是一个 YAML 文件，其中包含站点的设定和输出设定。内容如

```yaml
name: " 概率论"
output_dir: "_probbook"
output:
  html_document:
    toc: true
    mathjax: "../../../MathJax/MathJax.js"
    self_contained: false
    includes:
      in_header: "_header.html"
navbar:
  title: " 概率论"
  left:
    - text: "Home"
      href: index.html
    - text: "About"
      href: about.html
```

这里 `name` 设置站点名称，`output_dir` 给出生成的 html 及其他辅助文件的存放目录，缺省时用 `_site` 子目录。navbar 域设置了网站的菜单，这里包括 Home(主页), Contents(目录), About(关于) 三个页面的导航菜单。output 域设定输出的选项，其中 `html_document` 就是用 `rmarkdown::render_site()` 生成的网站的输出文件格式，其中 `toc: true` 说明每个页面都有自身内容的目录, 关于 `mathjax`, `self_contained`, `includes` 都与数学公式设置有关，这里设置数学公式使用与本网站在同一目录系统或同一网站地址存放的 MathJax 数学公式显示库，该库放在生成的 HTML 文件所在目录向上两层的 MathJax 子目录中。`in_header` 指定一个要插入到生成的每个 HTML 的 head 部分的内容文件，这里内容文件 `_header.html` 的内容是

```html
<script type="text/x-mathjax-config">
MathJax.Hub.Config({
  jax: ["input/TeX","output/SVG"],
  extensions: ["tex2jax.js","MathMenu.js","MathZoom.js"],
  TeX: {
    extensions: ["AMSmath.js","AMSsymbols.js","noErrors.js","noUndefined.js"]
  }
});
</script>
```

是对 MathJax 的一些设置，主要使用 LaTeX 作为输入格式。

C.2.2 编译

具有 index.Rmd 和 _site.yml 文件的项目，在 RStudio 软件中会显示一个 Build 窗格，点击其中的 Build Website 可以自动调用 rmarkdown::render_site() 生成整个站点，存放在 output_dir 指定的输出目录中，默认目录名为 _site。为了将生成的站点发布到网站，只要将输出目录的所有内容全部复制到服务器的某个子目录中就可以了。

rmarkdown::render_site() 生成整个站点时会自动逐个编译.Rmd 文件，并将项目目录中其它的文件和子目录复制到输出目录中，以句点或者下划线开头的文件和子目录不复制，.R, .r, .Rmd 文件不复制。

因为是自动依次编译所有的.Rmd 文件，所以中间某个源文件编译错误就会使得整个编译失败，修正错误后还是需要再次编译所有文件，这一点不太方便。但是，rmarkdown::render_site() 提供了这样一种操作模式：将有错的文件先移动到其它目录中，编译整个站点，这时生成的站点仅包含确认无误的各个页面；然后，再将可能有错的文件移动回到项目目录，逐个地打开每个未成功编译的文件，用 Knit 按钮单独编译，编译成功后结果文件也自动进入输出目录。

RStudio 的 Knit 按钮或者 rmarkdown::render_site() 命令可以人工控制一个一个地编译.Rmd 文件，编译成功一个就存放到输出目录中一个，这是用 rmarkdown::render_site() 制作网站比用 bookdown 扩展包制作网站的一个优点。因为网站的目录是自己编写的链接，所以这样逐个编译不会破坏网站的链接。编译单个文件的命令如

```
rmarkdown::render_site("sec1.Rmd", output_format = "html_document", encoding="UTF-8")
```

其中 sec1.Rmd 是某个网页内容源文件。

可以用 rmarkdown::clean_site() 删除生成的文件，包括程序结果缓存。

如果用 bookdown 包，只有编译所有文件才能生成正确的网站目录，bookdown 也能编译单个文件，但是仅能作为调试，调试成功后还是需要编译所有文件才能使得网站目录正确。

bookdown 站点的优势在于自动生成网站目录而不需要人工管理目录，而且 bookdown 支持图书的公式、定理、图标自动编号和链接，文献列表和文献引用，所以 bookdown 是比 rmarkdown::render_site() 编译调试速度比较慢但是功能更强大的一种制作网站的工具，而且还可以生成 PDF 版图书。

所以，一本书如果不需要很多的公式自动编号、图表编号、文献，在编写阶段可以使用 rmarkdown::render_site() 制作，定稿后再改成 bookdown 图书格式，实际上每个源文件内容部分不需要改变，只需要修改一下头部就可以了。站点的设置需要将 _site.yml 中的 site 域的值改为 bookdown::bookdown_site，并增加 _bookdown.yml 和 _output.yml 两个设置文件。详见B。

C.2.3 内容文件

内容文件只要用普通的 Rmd 文件即可，如某一网页的源文件如

```
---
title: " 期望"
---
```

```
随机变量 ** 数学期望 ** 可以看成是对随机变量取值的加权平均，
某个值的取值概率越大，加权越大。

对离散随机变量，
期望是加权和；
对连续型随机变量，
期望是用密度函数加权对`x`积分。
```

C.2.4 网站设置

在 `_site.yml` 文件中进行网站设置，样例见C.2.1。顶级设置有：

- `name`: 网站名称；
- `output_dir`: 保存编译结果的子目录名，默认为 `_site`，如果设置为`.`，则表示编译结果放在项目的顶级目录中而不是子目录中；
- `includes`: 指定输出的网站中要包含的文件，输出时默认不包含.R，.r, .Rmd，.RData, .rds 文件和文件名以小数点或者下划线开头的文件，所以例外要写在这里，如 `includes: ["data1.R", "data2.R"]`;
- `excludes`: 指定项目中不希望输出的结果网站的文件，可以有通配符，如 `excludes: ["backup.txt", "*.xlsx"]`;
- `output`: 其中的 `html_document` 属性的设置是所有 Rmd 网页源文件的共同设置，参见A.11.3，每一个 Rmd 文件本身也可以有自己的属性设置；
- `navbar`: 用来设置导航菜单，属性 `left` 指定主菜单栏，根据所用的网站主题，可能显示在左侧或者顶部。属性 `right` 指定一个右侧导航栏。

`includes` 和 `excludes` 设置主要是为了改变默认的文件发布规则，所有的.Rmd 文件只要不是文件名以下划线开头就都会被编译并且复制到结果子目录中。

C.3 用 blogdown 制作网站

R 扩展包 blogdown 可以与 Hugo 软件配合制作简单的静态网站。网站的所有文件都存在于一个目录中，只要上传到任意的网站服务器就可以发布，没有任何限制。

网站内容用 R Markdown 格式编写，在 R 或者 RStudio 中编译为网站。这样的做法使得网站内容容易再现，而且也可以将内容转换为 PDF、Word 等格式。blogdown 包的 R Markdown 格式支持 bookdown 包的扩充，如公式、图表编号与引用。

blogdown 包的名称虽然包含了 blog，但是其建站并不限于博客类型，任何类型的静态网站都可以构建，比较适合的是个人网站。

参考：

- (Yihui Xie, 2017)
- (Xie et al., 2019)

C.3.1 生成新网站的框架

在 RStudio 中，菜单 "File – New Project – blogdown site" 可以生成一个网站框架，注意一定要新建目录而不是用已有目录，而且第一次新建网站时还会自动下载 Hugo 静态网站生成程序。

生成的网站框架包括如下文件与子目录：

- index.Rmd：主页面的源文件。
- config.toml: 网站设置文件。
- ./content/: 存放作为网站内容的 md 或者 Rmd 文件。可以在下面再建立子目录。./content/post/用来保存博客类型的网页，但是不做要求。
- ./resources:
- ./static: 保存图片、CSS 等文件，在发布网站时会被复制到./public 子目录中。
- ./themes: 存放多个网站主题。
- ./public: 发布网站时所用到的所有文件。生成的内容被复制到这个目录中。只要将这个目录的内容全部复制到任何一个支持静态内容的网站服务器，就可以发布网站。

这样建站，建成的网站应该在 RStudio 的 Viewer 窗格自动显示，如果没有自动显示网站或者再次打开网站项目，点击图标栏的 `Addins -- Serve site` 可以启动网站服务器发布该网站。或者用用命令

```
blogdown::serve_site()
```

启动网站服务器。每次在 RStudio 或者 R 中打开一个网站项目，都只需要启动一次服务器。blogdown 的服务器会在后台运行，侦测到文件修改时会自动重建，这称为在线重建 (Live Reload)。在线重建功能基于 R 扩展包 servr。所以用户只要发布新的消息或者修改已有消息，不用关心后台的转换问题。

blogdown 提供了一些 RStudio 的 addin 程序，可以为网站功能提供方便，比如 `New Post` 可以生成新消息，`Update Metadata` 可以修改消息的 YAML 元数据，`Insert Image` 可以在消息中插入图片。

使用在线重建功能时，应关闭一些其他功能。在 RStudio 的 "Tools – Project Options – Build Tools" 对话框中，取消选定 "Preview site after building" 和 "Re-knit current preview when supporting files change"。

如果不使用在线重建功能，即不使用 Serve site，可以在 RStudio 的 Build 窗格中用 "Build Book" 功能手工控制网站的重建。

C.3.2 网页内容文件及其设置

网页内容文件可以用 Rmd 文件也可以用 md 文件，但是 Rmd 文件使用 Pandoc 和 bookdown 包支持的文件格式，功能更强，比如支持 LaTeX 数学公式，公式、图表自动标号与交叉索引，等等，所以最好是用 Rmd 文件。

如果有某个 Rmd 源文件暂时不想编译输出，可以将其扩展名临时改为一个不被编译的扩展名如.txt。

blogdown 编译结果的网页是 HTML 格式，其 Rmd 元数据输出类型为 `blogdown::html_page`，此类型基于 `bookdown::html_document2`，又基于 `rmarkdown::html_document`。这样，可以在 Rmd 源文件的元数据部分对该输出类型设置相应的属性，如 `toc: true` 等。例子：

```
---
title: " 一个试验网页"
author: " 李东风"
date: "2019-07-08"
output:
  blogdown::html_page:
    toc: true
    fig_width: 6
    dev: svg
---
```

为了给所有输出设置统一的属性，可以在项目目录中添加 `_output.yml` 文件，在其中设置输出文件的各种属性，如：

```
blogdown::html_page:
  toc: true
  fig_width: 6
  dev: svg
```

C.3.3 初学者的工作流程

Hugo 建站程序比较复杂，如果要挑选自己喜爱的网站主题 (theme)，移植网站也比较麻烦。对于了解网站技术不多的建站者，建议用如下的流程建立新站：

- 在https://themes.gohugo.io/仔细地挑选一个合适的网站主题 (theme)；
- 在 RStudio 中通过生成一个在已有目录的新项目；
- 在新项目的控制台中提交命令 `blogdown::new_site(theme = 'user/repo')`，其中 `user/repo` 是前面所选的主题的用户名与资源 (repository) 名；
- 试验新网站看是否满意。如果不满意可以重复以上步骤，如果满意，就可以修改 `config.toml` 中的设置。如果某个选项不明白，可以在该主题的文档中查找，通常是资源的 README 文件。只需要修改必须需改的选项。

为修改已有网站内容，用如下的步骤：

- 点击 RStudio 的 addin “Serve Site”，可以在修改过程中动态地构建网站。每次打开 RStudio 的网站项目仅需执行一次这个命令。
- 用 “New Post” addin 添加新的网页。
- 在修改某个网页时，用 “Update Metadata” addin 修改网页元数据，主要是网页标签等。

为了发布网站，只要将项目内的 public 目录上传到任一支持静态内容的网站服务器。如果没有合适的服务提供商，可以考虑https://www.netlify.com/，可以用 Github 账户登录并有一定的免费建站服务。如果熟悉 Github，还可以将自己的网站源文件作为一个 Github 资源 (repository)，将 Github 资源发布到 Netlify，就不需将 public 的内容上传到网站，而只要有源文件就可以了，Netlify 支持 Hugo。

C.3.4 网站设置文件

在项目根目录中有一个 `config.toml` 文件，是 Hugo 软件的建站设置文件，可以设置网站标题、描述、菜单等。一个例子如下：

```
baseurl = "/"
languageCode = "en-us"
title = "试验Blogdown网站"
theme = "hugo-lithium"
googleAnalytics = ""
disqusShortname = ""
ignoreFiles = ["\\.Rmd$", "\\.Rmarkdown$", "_files$", "_cache$"]

[permalinks]
    post = "/:year/:month/:day/:slug/"

[[menu.main]]
    name = "About"
    url = "/about/"
[[menu.main]]
    name = "Li Dongfeng Homepage"
    url = "http://www.math.pku.edu.cn/teachers/lidf/"

[params]
    description = "用Hugo和blogdown制作的网站"

    # options for highlight.js (version, additional languages, and theme)
    highlightjsVersion = "9.12.0"
```

```
highlightjsCDN = "//cdnjs.cloudflare.com/ajax/libs"
highlightjsLang = ["r", "yaml"]
highlightjsTheme = "github"

MathJaxCDN = "//cdnjs.cloudflare.com/ajax/libs"
MathJaxVersion = "2.7.5"

# path to the favicon, under "static"
favicon = "favicon.ico"

[params.logo]
url = "logo.png"
width = 50
height = 50
alt = "Logo"
```

在设置文件中，字符串要写成用双撇号界定的形式，布尔值要写成没有双撇号的 true 或者 false。用方括号包围的项（如 `[permalinks]`）会定义一个列表类型的变量，下面缩进后的内容是列表的元素。用双方括号包围的项是列表数组，即每一项都是列表，但是合并在一起由组成一个元素为列表的数组。如上面 `[[menu.main]]` 有若干项，每一项是一个主菜单项，其设置是一个列表。主菜单位置、配色等在其它地方设置，主设置文件中仅设置其文字和对应链接。

不同的网站主题需要不同的 `config.toml` 文件。

下面给出一些选项的解释。

- `baseURL`：这是你的网页发布到网站上时，该网站给你的主页面的地址。一般是需要设置为自己发布以后的 divide。
- `languageCode`: 中文是 `zh-cn`，英文是 `en-us`，可能会影响一些排版规则，但是编码都是 UTF-8。
- `permalinks`: Hugo 和 blogdown 在发布 HTML 与图片等内容时需要不变的链接，默认是 content 中的每个 Rmd 文件变成 `public` 或 `public/post` 下的同名子目录中的 `index.html` 文件。可以用 `permalinks` 指定命名规则。`slug` 是网页元数据可以设置的一项类似于文章标识符的东西，不设置则使用文章标题。
- `publishDir`: 生成的网站存放的文件，默认为 `public` 目录。
- `theme`：在 `themes/`目录中保存所选主题的目录的名字。
- `ignoreFiles`: 项目中那些文件不被复制到生成结果网站中。
- `hasCJKLanguage`: 如果有大量中文、日文、韩文内容，设置为 `true` 可以在网页统计和单词书统计时更准确。
- `[params]`: 主题特有的设置放在这里。

C.3.5 静态文件

项目中在`./static/`下的文件称为静态文件，可以保存自己的图片、Javascript 库等，发布时自动复制到`./public/`中。比如，文件`.static/figs/mypic01.png`，在 Rmd 文件中可以用`![](figs/mypic01.png)`引

用。

每个主题中一般也有一个 static 目录，自己的 static 目录中的同名文件在发布时可以覆盖主题中的同名文件，这样可以修改主题的一些资源或者设置。从 Rmd 编译生成的文件也可能会被 static 中的同名文件覆盖。

还可以将网页编译为 PDF 等格式存放在 static 目录中。`rmarkdown::build_dir()` 命令可以将目录中所有 Rmd 文件按照其元数据的输出格式编译输出。

附录 D

制作幻灯片

D.1 介绍

R Markdown 文件 (.Rmd) 文件支持多种输出，如网页 (`html_document`)、MS Word(`word_document`)、PDF(`pdf_document`, 需要 LaTeX 编译器支持) 等，还支持生成网页格式的幻灯片 (`slidy_presentation`, `ioslides_presentation`)，以及 LaTeX beamer 格式的 PDF 幻灯片 (`beamer_presentation`)，和 Microsoft Office 的 PowerPoint 幻灯片 (`powerpoint_presentation`) 格式。

D.2 Slidy 幻灯片

Rmd 文件选用输出格式 `slidy_presentation` 可以生成网页格式的幻灯片，并具有缩放字体大小、显示幻灯片目录等功能。只要在.Rmd 文件开头的 YAML 元数据部分指定 `output: slidy_presentation`。因为幻灯片的单位是帧 (frame)，与论文的结构有很大区别，所以幻灯片 Rmd 文件很难同时作为论文的源文件。

D.2.1 文件格式

幻灯片分为多个帧，每帧用二级标题作为标志并以其为标题。二级标题就是行首以两个井号和空格开始的行，或者在标题下面画由减号组成的线的行。用一级标题作为单独的分节帧，将单独显示在一帧中。一级标题是行首以一个井号和空格开始的行，或者在标题下面画由等于号组成的线的行。

帧也可以没有标题，比如仅有照片的帧，这时，用三个或三个以上的减号连在一起标识新帧的开始。

一个简单的 slidy_presentation 幻灯片源文件 example-slidy.Rmd，内容如：

```
---
title: "R 幻灯片演示样例"
author: " 李东风"
date: "2017-11-16"
```

```
output: slidy_presentation
---
```

```
# 用 R Markdown 的 slidy 输出作幻灯片

## 幻灯片结构

- 用二级标题标志一个页面开始
- 用一级标题制作单独的分节页面
- 用三个或三个以上减号标志没有标题的页面开始
- 每个页面一般用 markdown 列表显示若干个项目

## 幻灯片编译

- 用 RStudio 编辑
- 用 RStudio 的 Knit 按钮，选`slidy_presentation`作为输出格式

-----

[一个演示画面](figs/demoscreen.png)
```

D.2.2 幻灯片编译

幻灯片用 RStudio 的 knit 按钮编译，选择输出格式为 `slidy_presentation`，结果在浏览器中播放。

除了使用 knit 按钮，还可以用类似如下命令：

```
rmarkdown::render("mydemo.Rmd", output_format = "slidy_presentation", encoding="UTF-8")
```

其中 `mydemo.Rmd` 是源文件。

为了制作幻灯片，最好单独设置一个 RStudio 项目，并且此项目仅生成幻灯片，而不生成普通网页、Word、PDF 等输出，否则可能造成结果混乱。希望 rmarkdown 包的后续版本能取消这个限制。

D.2.3 播放控制

播放时，用如下方式控制：

- 鼠标左键单击、光标右移键、向下翻页键、空格键都可以翻到下一页；
- 光标左移键、向上翻页键回退一页；
- 单击下方的 Contents 或单击 C 键显示幻灯片目录列表，可单击转移到任意页面；

- Home 键回到幻灯片开头；
- 用 A 键切换是否将所有页面合并显示成一个长的网页；
- 用 S 键缩写字体，用 B 键放大字体；
- 通过选择保存为 PDF 的打印机进行打印，可以将幻灯片转换为 PDF，但是打印生成的 PDF 仍是每页仅有原来的一帧，所以转换成一个单页的 HTML 再打印可能更合适。

D.2.4 生成单页 HTML

对 slidy 幻灯片的 Rmd 源文件不加修改，也可以通过命令直接转换为普通的单页 HTML 文件，但是因为在 slidy 幻灯片源文件中二级标题用来分帧，而普通 Rmd 文件中二级标题用来分小节，所以生成的单页 HTML 文件会有许多小节。这样的文件更适合打印以及转换为 PDF 文件。

命令如：

```
rmarkdown::render("mydemo.Rmd", output_file="handout.html", output_format="html_document", encoding="
```

D.2.5 数学公式处理与输出设置文件

讲课用的幻灯片经常会有数学公式，比较关键的问题是数学公式如何处理。网页中的数学公式一般使用一个公开的自由 Javascript 库 MathJax 显示，但是这个库很大，如果使用远程的库，在网络不畅通时显示公式就不正常。更好的办法是使用局部的 MathJax 库或将 MathJax 库安装在临近的网站服务器上。

为了使用局部的 MathJax 库，简单的办法是在 YAML 的 `ioslides_presentation` 项目下面指定 `mathjax: local`，如：

```
---
title: "R 幻灯片演示样例"
output:
  slidy_presentation:
    mathjax: local
---
```

上述办法容易使用，缺点是多个不同的演示项目无法共用一个局部的 MathJax 库，生成的结果包含了许多小的支持文件。

为此，可以将 MathJax 库装在演示项目所在目录的上层（比如上两层的 `MathJax` 目录内），将设置 MathJax 的代码放在 `_header.html` 文件中，`_header.html` 中的内容如：

```
<script type="text/x-mathjax-config">
MathJax.Hub.Config({
  jax: ["input/TeX","output/SVG"],
  extensions: ["tex2jax.js","MathMenu.js","MathZoom.js"],
```

```
  TeX: {
    extensions: ["AMSmath.js","AMSsymbols.js","noErrors.js","noUndefined.js"]
  }
});
</script>
```

在演示项目所在目录中增加一个 `_output.yml` 文件，这使得该项目所有输出的共用输出设置，内容如：

```
slidy_presentation:
    toc: false
    mathjax: "../../MathJax/MathJax.js"
    self_contained: false
    includes:
      in_header: "_header.html"
html_document:
    toc: true
    number_sections: false
    mathjax: "../../MathJax/MathJax.js"
    self_contained: false
    includes:
      in_header: "_header.html"
```

这里关闭了 `self_contained` 选项，设置了 MathJax 库在本地目录，具体是演示项目所在目录上面两层的 MathJax 目录中。

公式中如果有中文，开始时可能显示不正常，这时右键点击公式弹出菜单选择 “Math Settings–Math Renderer”，取为 “HTML-CSS” 或 “SVG” 应可解决问题。

上面的输出设置文件中也设置了输出 `html_document`，这是单页的 HTML 格式，取消了自动章节编号，因为源文件中每帧都是一个小节。

D.2.6 其它选项

用 `slidy_presentation` 的 `font_adjustment: -1` 可以缩小字体一号，类似可以设为 `+1`，`-2` 等。在播放时也可以用 S 和 B 键缩小或放大显示。

可以在播放时在浏览器状态栏显示倒数计时器，用 `slidy_presentation` 的 `duration: 5` 表示每帧显示 5 分钟。这是总的预计时间，适用于演讲有时间限制的情况。

可以用 `footer` 属性指定每帧都显示的状态栏脚注，如：

```
---
output:
  slidy_presentation:
    font_adjustment: -1
    duration: 5
    footer: " 北京大学"
---
```

`slidy_presentation` 输出属性 `incremental: true` 可以使得列表显示需要每点击一次才显示下一项。

D.2.7 slidy 幻灯片激光笔失效问题的修改

slidy 幻灯片翻页是用空格、左右光标、上下翻页键，而一般激光笔翻页是模拟上下光标键。为此，在安装的 R 软件目录的 `library/rmarkdown/rmd/slidy/Slidy2/scripts` 子目录中，找到 slidy.js 文件，用编辑器打开，用编辑器的搜索功能搜索 `key == 37`，将其替换成 `key == 37 || key == 38`，这里 37 是向左光标的编码，替换后就是向左或者向上光标。用编辑器的搜索功能搜索 `key == 39`, 将其替换成 `key == 39 || key == 40`，这里 39 是向右光标的编码，替换后就是向右或者向上光标。修改完毕后保存，然后将 slidy.js 用文件压缩程序（如 7zip）压缩为 slidy.js.gz。这样就可以在用 rmarkdown 制作的 `slidy_presentation` 结果中支持激光笔翻页了。

这样修改后，如果某帧超高，需要滚动显示，就只能通过鼠标滚轮滚动了。

D.3 MS PowerPoint 幻灯片

`powerpoint_presentation` 输出格式可以生成 MS PowerPoint 文件。设置如：

```
---
output:
  powerpoint_presentation:
    slide_level: 2
---
```

可以用 `powerpoint_presentation` 的 `reference_doc` 属性指向一个.pptx 的文件，作为模板，模板中的样式将被输出结果采用。

可以用 Rstudio 的 Knit 快捷图标实现转换（选其中的 knit to PowerPoint），或者用如下命令：

```
rmarkdown::render("slides.Rmd", output_format="powerpoint_presentation", encoding="UTF-8")
```

如果从 bookdown 内容转化成演示幻灯片，需要大量修改原始文件内容，设置成幻灯片常用的逐条播放格式。如果不希望进行这样的修改，可以直接在 MS PowerPoint 软件中直接输入或者复制粘贴已经编译好的网页格式输出的内容。这样复制粘贴有一个缺点，就是基于 MathJax 显示在网页中的公式，无法通过复制粘贴转换到 PowerPoint

中。一种办法是用 rmarkdown 将需要转换的带有公式的 Rmd 文件编译为 Word 格式或者 PowerPoint 格式，再复制进入 PowerPoint 软件的新页面中。编译为 Word 格式的命令如：

```
rmarkdown::render("slides.Rmd", output_format="word_document", encoding="UTF-8")
```

D.4 Bearmer 幻灯片格式

上述 Slidy 格式的幻灯片，也可以通过 LaTeX 编译器转换成 LaTeX beamer 格式的幻灯片，设置如：

```
---
output:
  beamer_presentation:
    includes:
      in_header: preamble.tex
    latex_engine: xelatex
    slide_level: 2
    theme: CambridgeUS
    colortheme: dolphin
    citation_package: biblatex
    keep_tex: yes
---
```

其中 `slide_level` 用来规定几级标题开始新的一帧。`theme` 指定一种主题，`colortheme` 指定一种配色方案，`in_header` 在 LaTeX 导言部分插入 `preamble.tex`，内容如：

```
\usepackage{ctex}
\usepackage{amsthm,mathrsfs}
```

用一个井号分节，用两个井号开始一个新的帧，在两个井号和空格之后输入帧的标题。

编译命令如：

```
rmarkdown::render("mydemo.Rmd",
  output_format = "beamer_presentation", encoding="UTF-8")
```

结果是一个 PDF 文件。目前不支持 bookdown，所以 bookdown 的哪些自动编号、交叉引用功能还无法使用。

D.5 R Presentation 格式

R Studio 软件单独提供了对一种 R Presentation 格式的源文件的支持，以.Rpres 扩展名结尾，是一种特殊的 R Markdown 文件，与 slidy 的源文件也类似。

Rpres 文件编译为 HTML 格式的幻灯片，使用 reveal.js 控制显示。reveal.js 中也有对激光笔支持不好的问题，这是因为 reveal.js 中用向右光标键翻页，对向下光标另有定义，激光笔一般是模拟向下和向上光标键来翻页的。为了支持激光笔，找到 RStudio 的安装目录，在 `resources/presentation/revealjs/js` 中找到 `reveal.js` 文件，在文件编辑器中打开，通过搜索找到 `case 38:`, 将其剪切到 `case 33:` 后面，变成 `case 33: case 38:`。找到 `case 40:`, 将其剪切到 `case 34:` 后面，变成 `case 34: case 40:`。同一目录还有一个 `reveal.min.js` 文件，也进行上述修改。

Bibliography

Andrzej Galecki, T. B. (2013). *Linear mixed-effects models using r*. Springer.

Becker, R. A., & Chambers, J. M. (1984). *S: An interactive environment for data analysis and graphics*. Wadsworth Advanced Books Program, Belmont CA.

Becker, R. A., Chambers, J. M., & Wilks, A. R. (1988). *The new s language*. Chapman; Hall, New York.

Bretz, F., Hothorn, T., & Westfall, P. (2011). *Multiple comparison with r*. CRC Press.

Chambers, J. M., & Hastie, T. (1992). *Statistical models in s*. Chapman; Hall, New York.

Chambers, J. (2016). *Extending r*. Chapman; Hall/CRC, London.

Chambers, J. M. (2008). *Software for data analysis: Programming with r*. Springer.

Cleveland, W. S. (1979). Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, *74*, 829–36.

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd). Routledge. https://doi.org/https://doi.org/10.4324/9780203771587

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning with applications in r*. Springer.

Kabacoff, R. I. (2012). *R 语言实战*. 人民邮电出版社.

Knuth, D. E. (1984). Literate programming. *The Computer Journal*, *27*(2), 97–111.

Roback, P., & Legler, J. (2021). *Beyond multiple linear regression - applied generalized linear models and multilevel models in r*. Chapman; Hall/CRC. https://bookdown.org/roback/bookdown-BeyondMLR/

Trevor Hastie, J. F., Robert Tibshirani. (2009). *The elements of statistical learning* (2nd Ed.). Springer.

Venables, W. N., & Ripley, B. D. (2002). *Modern applied statistics with s* (4th Ed.). Springer.

Wickham, H. (2014). Tidy data. *J Stat Software*, *59*. http://www.jstatsoft.org/v59/i10/

Wickham, H. (2019). *Advanced r* (2nd Edition).

Wickham, H., & Bryan, J. (2022). *R packages*. https://r-pkgs.org/index.html

Wilke, C. O. (2019). *Fundamentals of data visualization: A primer on making informative and compelling figures*. O'Reilly Media. https://serialmentor.com/dataviz/

Xie, Y. (2017). *Bookdown: Authoring books and technical documents with r markdown*.

Xie, Y., Allaire, J. J., & Grolemund, G. (2019). *R markdown: The definitive guide*. CRC Press. https://bookdown.org/yihui/rmarkdown/

Xie, Y., Dervieux, C., & Riederer, E. (2020). *R markdown cookbook*. CRC Press. https://bookdown.org/yihui/rmarkdown-cookbook/

Yihui Xie, A. P. H., Amber Thomas. (2017, October 10). *Blogdown: Creating websites with r markdown.*
陈家鼎、孙山泽、李东风、刘力平. (2015). 数理统计学讲义 (第 3 版). 高等教育出版社.