# 附录 B R 入门

R 语言是一个综合的统计研究平台，提供各类数据分析技术的实现。随着数据科学领域的发展，人们不再满足于 R 默认加载的包（package）中所提供的基础操作函数，tidyverse 包应运而生。它是一系列用于数据科学领域的 R 包集合，包含 ggplot2、dplyr、tidyr、readr、purrr、tibble、stringr 和 forcats 等软件包，可以高效、便捷地处理数据，是近年来 R 语言数据分析的主流工具。因此，本附录基于 tidyverse 包，简要地介绍 R 语言的主要操作。本附录的写作主要参考 Wickham et al. (2016) 和 Kabacoff (2015)，读者可进一步阅读以获取更多详情。

附录代码使用一致的书写规则：

- 正文中的斜体字、代码块中的 < > 表示占位，实际使用中应用真实文本替换。
- 为避免不同包中同名函数的冲突，有时在包名与函数名间加双冒号，指定使用来自某包的函数。如：purrr::set_names() 表示使用 purrr 包中的 set_names() 函数。
- 使用 #> 注释代码的输出结果。

## B.1 安装与配置

### R 与 RStudio 的安装

我们可以从官方网站 CRAN，网址为http://cran.r-project.org，免费下载各操作平台的 R 安装包进行安装。

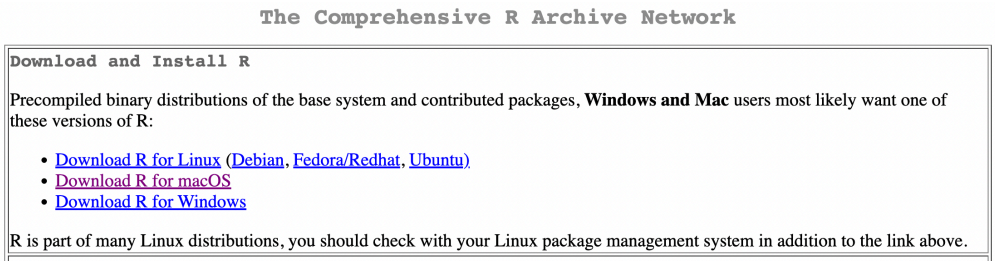


图 B.1: R 的各安装包

由于 R 软件仅提供一个简单的命令行界面，为了使得编程过程更为舒适，我们推荐使用与 R 交互的代码编辑器，RStudio IDE。我们可以从官方网站 https://www.rstudio.com/download 下载和电脑系统匹配的 RStudio Desktop 版并进行安装。

### 包的安装与使用

包是 R 函数、数据、预编译代码组成的集合，存储包的目录称为库（library）。除 R 自带的一系列默认包（如 base、datasets 和 stats 等），其他包需要安装并被加载到会话中才能使用。为避免各种包因为语法、接口的差异而造成的不便，tidyverse 包整合了一系列常用包，各包共享基本语法和数据结构。

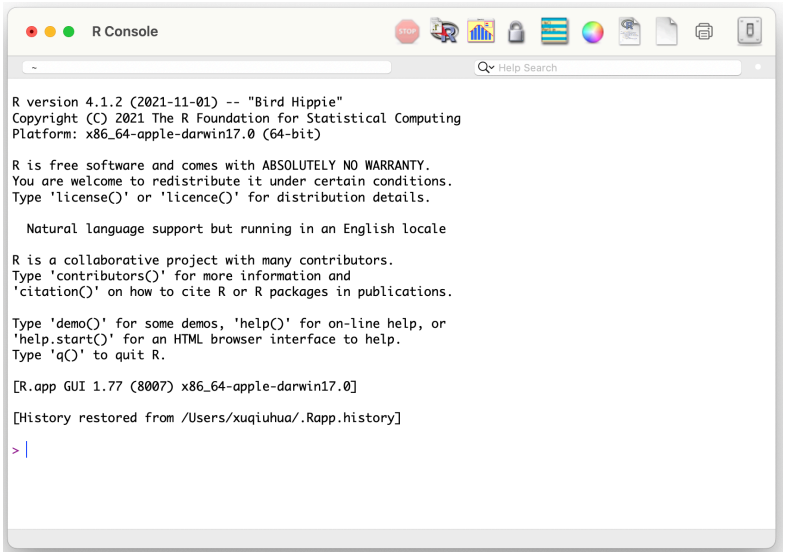


图 B.2: R 的命令行界面

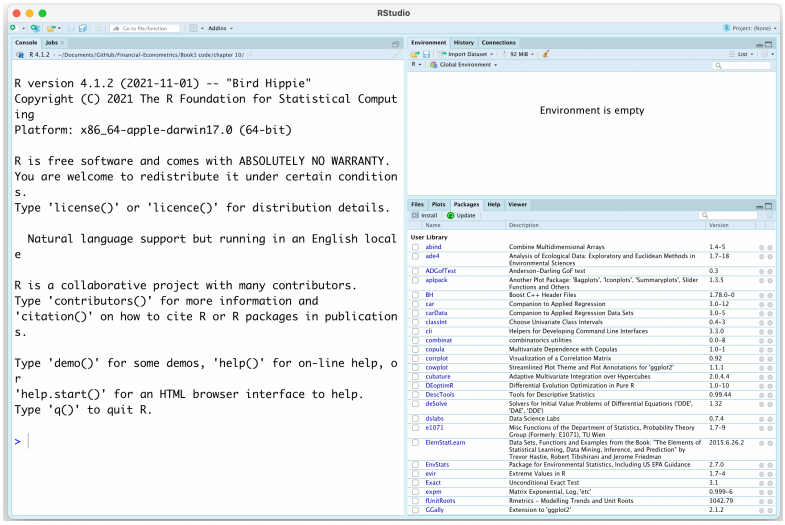


图 B.3: RStudio 的图形用户界面

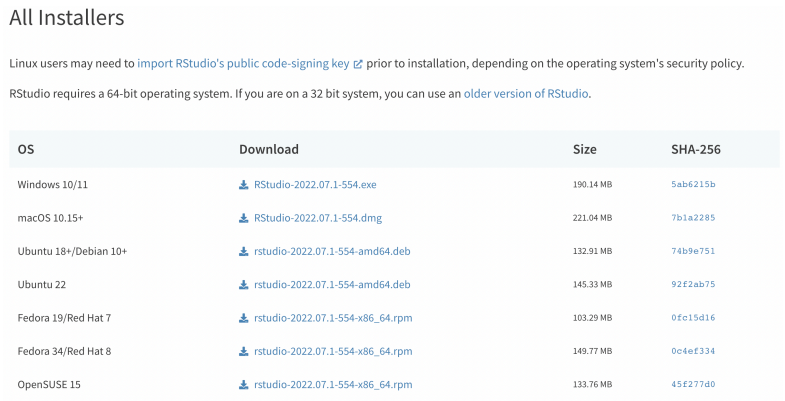


图 B.4: RStudio Desktop 的各安装包

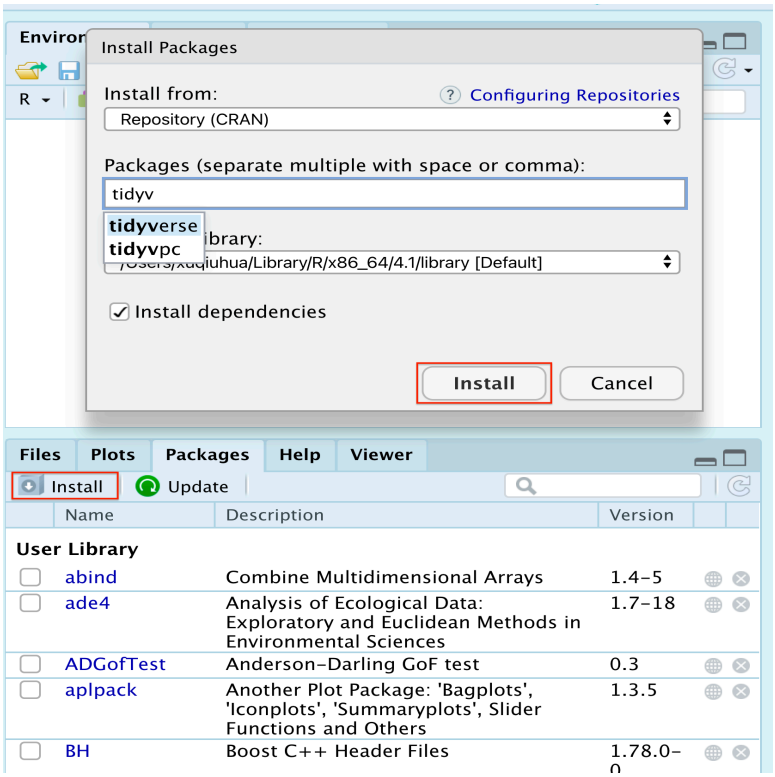


图 B.5: 手动安装 tidyverse 包

我们可以通过 RStudio 手动安装所需包，如图 B.5 所示。我们也可以使用命令 install.packages(*"packagename"*) 安装所需包。命令.libPaths() 可以显示包含 R 包的库所在的位置：

```
install.packages("tidyverse") #安装单个包
.libPaths() #查看库所在位置
```

命令 install.packages() 将列出所有已安装的包的信息：

```
rownames(installed.packages()) #查看当前已安装的包的名称
```

为避免重复安装，我们应在安装前先判断该包是否已安装：

```
pkgs = c("xts","vars","rugarch") #批量安装多个包
lapply(pkgs, function(x) if (!(x %in% installed.packages())) {
  install.packages(x) # 若该包已被安装，则跳过该包
})
```

包安装好后，使用 library( *packagename* ) 命令加载后才能在会话中使用。在同一会话中，一个包只需加载一次。命令 search() 可以查看哪些包当前已加载可被使用：

```
library("tidyverse")
search() #查看当前已加载的包
```

我们可以使用命令 help(package="*packagename*") 快速了解某个包所含的数据集和函数，也可以使用 help(*datafunction*) 或命令?*datafunction* 查看某个数据集或函数的更多细节。此外，RStudio CheatSheets，即 RStudio 备忘录，归纳了常用软件包的主要函数，可供读者查阅，网址为：https://www.rstudio.com/resources/cheatsheets/。

```
help(package="tidyverse")
#查看mutate函数的帮助文档
help(mutate)
?mutate
```

还有一些函数用于 R 包的管理。如 update.packages() 将已安装的包更新至最新版本，remove.packages() 移除已安装的包。

### 工作空间的管理

工作空间（workspace）指 R 当前的工作环境，它存储着用户定义的对象。我们可以使用命令 getwd() 查看当前的工作目录，它是 R 用来读取文件和保存结果的默认路径。如需更改工作目录，我们可以通过命令 setwd("*directory*") 设置。此外，函数 ls() 用于列出当前工作空间的对象，rm() 删除指定对象。

```
getwd() #查看当前工作路径
setwd("~/Documents/GitHub") #设置新工作路径
rm(list = ls(all = TRUE)) #清除当前所有对象
```

注 命令 setwd() 更改的路径应该是事先已经存在的，若路径不存在，可以先使用 dir.create() 创建新路径，再使用 setwd() 将当前工作路径更改为新创建的路径。

## B.2 数据结构

R 包含多种存储数据的对象类型，如字符串、向量、矩阵、数组、数据框和列表，以及基于向量构建的因子、日期和时间等。

我们事先加载好 tidyverse 包，加载后 R 的输出结果表明它已经载入了 ggplot2 等 8 个核心包；冲突信息显示了已加载的包的重名函数，我们需使用 :: 明确指定。

```
library(tidyverse)
#> -- Attaching packages ------------------------- tidyverse 1.3.2 --
#> v ggplot2 3.3.6   v purrr   0.3.4
#> v tibble 3.1.8    v dplyr   1.0.9
#> v tidyr  1.2.0    v stringr 1.4.0
#> v readr  2.1.2    v forcats 0.5.1
#> -- Conflicts ---------------------------- tidyverse_conflicts() --
```

```
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()  masks stats::lag()
```

## 向量、矩阵与数组

### 创建向量

向量是 R 的基础对象，常见类型有数值型、字符型和逻辑型，其中，数值型包括整型与双精度型，R 中默认数值为双精度型，若想创建整型数值，可在数字后加字母 L。我们通常使用函数 c() 创建向量，也可以通过比较运算符构造逻辑向量。

```
a <- c(1,3,7) #创建双精度型向量
b <- c(1L,3L,7L) #创建整型向量
c <- c("adc","mgn","kk")
d <- c(TRUE,TRUE,FALSE,TRUE,FALSE)
e <- 1:10 %% 3==1 #通过比较运算符创建逻辑型向量
```

向量中不能包含不同类型的元素（否则会进行强制转换）。函数 typeof() 用于确定向量类型，函数 length() 用于确定向量长度。

```
#变量d,e接上例
typeof(d)
#> [1] "logical"
length(e)
#> [1] 10
```

向量的每个元素是可以命名的。我们可以在创建向量时进行命名，也可以在向量创建好后，使用 tidyverse 中的 purrr::set_names() 函数进行命名。

```
 c(xc=-1,y=0,z=1)
 #> xc  y  z
 #> -1  0  1
 purrr::set_names(1:3,c("A","B","C"))
 #> A B C
 #> 1 2 3
```

### 向量取子集

取子集是向量的常用操作，方括号 [ ] 是取子集函数，常用的向量取子集方式有：

- 使用数值向量取子集：使用正整数向量取子集时，取出对应位置的元素。而使用负整数向量取子集时，则舍弃对应位置的元素。注意 R 中的索引（index）是从 1 开

始的。

```
X <- c("red", "black", "white", "green", "blue")
X[c(1,5,3,3)]
#> [1] "red"  "blue" "white" "white"
X[c(-2,-4,-5)]
#> [1] "red"  "white"
```

- 使用逻辑向量取子集：此时可以取出 TRUE 值对应的元素。

```
Y <- c(-3, NA, 1, 7, NA)
Y[!is.na(Y)] #取出Y的所有非缺失值
#> [1] -3 1  7
```

- 使用字符向量取子集：对于命名向量，可以直接根据元素名称取出对应元素。

```
Z <- c(ad = -3, eee = 0, W1 = 2)
Z[c("W1", "eee")]
#> W1 eee
#> 2  0
```

### 矩阵与数组

矩阵由一组向量构成，可以通过函数 matrix() 创建：

```
cel <- c(11,6,2,8,-9,22)
row_names <- c("R1", "R2","R3")
col_names <- c("C1", "C2")
mat <- matrix(cel, nrow=3, ncol=2, byrow=TRUE, dimnames=list(row_names,
    col_names)) #byrow=TRUE代表按行填充
mat
#>    C1 C2
#> R1 11 6
#> R2 2  8
#> R3 -9 22
```

我们可以使用索引和方括号来选取矩阵的行、列或元素。$\boldsymbol{X}[i,]$ 指矩阵 $\boldsymbol{X}$ 的第 $i$ 行，$\boldsymbol{X}[,j]$ 指 $\boldsymbol{X}$ 的第 $j$ 列，$\boldsymbol{X}[i,j]$ 指 $\boldsymbol{X}$ 的第 $(i,j)$ 位置上的元素。选择矩阵的多行或多列时，可由多个索引 $i$ 或 $j$ 构成数值型向量。

需注意矩阵代数的运算符与常规运算有所不同:

- %*% 表示矩阵乘法，即内积。

- * 表示矩阵的对应位置元素相乘，即 Hadamard 积。
- %o% 表示矩阵外积。

数组与矩阵类似，但其维度可以大于 2。我们通过函数 array() 创建数组。其余操作与矩阵类似。

### 数据框

数据框是矩阵的拓展形式，它允许不同列包含不同类型的数据（但每一列的数据类型必须唯一）。传统的 R 数据框由基础包中的函数 data.frame(*col1,col2,col3,...*) 生成，其中 *col1,col2,col3* 等为各列向量。tidyverse 中的 tibble 包定义了简单数据框 tibble 代替传统数据框，操作更为简便。

### 创建数据框

我们可以通过函数 tibble() 创建 tibble，该函数能自动重复长度为 1 的输入，并可以使用新创建的变量。与 data frame 不同，tibble 是不创建行名的。

```
library(tidyverse)
tibble(
  x = c(2,-1,3),
  y = 1,      #自动重复该输入
  z = x^2 + y, #使用刚刚创建的变量
  ch = LETTERS[1:3]
  )
#> A tibble: 3 x 4
#>       x     y     z ch
#>   <dbl> <dbl> <dbl> <chr>
#> 1     2     1     5     A
#> 2    -1     1     2     B
#> 3     3     1    10     C
```

由输出结果可知，除列名外，tibble 还会输出各列的类型。相比于 data frame，对数据使用者更为友好。

我们还能使用函数 tribble() 创建 tibble。tribble 代表 transposed tibble，即转置 tibble，可以对数据按行进行填充。其中，列名以 ~ 开头。这种布局方法更加易读。

```
tribble(
  ~stu,~num,~score,
  "Bob","001",89,
  "Lily","024",95
)
```

```
#> A tibble: 2 x 3
#>  stu   num   score
#>   <chr> <chr> <dbl>
#> 1 Bob   001     89
#> 2 Lily  024     95
```

大多数 R 包使用的是传统数据框，我们可以通过函数 as.tibble() 将传统数据框转换为 tibble：

```
as_tibble(rock) #将rock石头形态数据集转换为tibble类型
#> A tibble: 48 x 4
#>    area  peri  shape  perm
#>   <int> <dbl> <dbl> <dbl>
#> 1  4990 2792. 0.0903 6.3
#> 2  7002 3893. 0.149  6.3
#> 3  7558 3931. 0.183  6.3
#> 4  7352 3869. 0.117  6.3
#> 5  7943 3949. 0.122 17.1
#> 6  7979 4010. 0.167 17.1
#> 7  9333 4346. 0.190 17.1
#> 8  8209 4345. 0.164 17.1
#> 9  8393 3682. 0.204 119
#> 10 6425 3099. 0.162 119
#> # ... with 38 more rows
#> # i Use `print(n = ...)' to see more rows
```

注意到，与 data frame 输出所有数据不同，tibble 默认输出前 10 行结果，更适用于大型数据集。当需要显示更多行时，可通过 print() 函数的参数 n（打印行数）来设置。此外，我们也可以使用函数 View() 滚动查看数据，使用函数 dplyr::glimpse() 查看数据摘要。

### 数据框的简单操作

add_row() 函数与 add_column() 函数用于增加 tibble 的某行或某列：

```
tb=tibble(
x = 1:3,
y = 3:1,
)
add_column(tb, z = 0, .after = 1) #插入一列常数向量至倒数第2列
#> A tibble: 3 x 3
```

```
      x     z     y
#>  <int> <dbl> <int>
#>1     1     0     3
#>2     2     0     2
#>3     3     0     1
add_row(tb, x = 7, y = 7, .before = 0) #插入一行数据至第1行
#> A tibble: 4 x 2
#>      x     y
#>  <dbl> <dbl>
#>1     7     7
#>2     1     3
#>3     2     2
#>4     3     1
```

在子集的取用上，操作 `[ ]` 的返回值仍为 tibble，`[[ ]]` 或 `$` 的返回值为向量。dplyr 包中的函数可以更好地筛选 tibble，我们将在后续介绍。

- 取 tibble 某列

```
#变量tb接上例
#按名称取
tb$x # 或 tb[["x"]]
#> [1] 1 2 3
#按位置取
tb[[1]]
#> [1] 1 2 3
tb[,1]
#> A tibble: 3 x 1
#>       x
#>   <int>
#> 1     1
#> 2     2
#> 3     3
```

- 取 tibble 某行，需用逗号作分隔符

```
tb[1,]
#> A tibble: 1 x 2
#>       x     y
#>   <int> <int>
#> 1     1     3
```

### 数据框文件的导入与导出

文件数据通常以数据框的形式读入 R 中。tidyverse 的 readr 包提供了一系列函数实现这一功能。该类函数语法一致，我们以最为常用的 read_csv() 函数为例。该函数用于读取逗号分隔文件，主要参数有：

- file：要读取的文件的路径。
- col_names：默认为 TRUE，使用数据的第一行作为列名。也可以向该参数传递一个字符向量作为列名。
- skip：设定读取数据时跳过前几行。
- na：规定了需将文件中哪些值处理为缺失值。

与 R 基础包中 read.*() 函数族相比，read_*() 函数族读取速度更快，且可直接生成 tibble。

readr 包还提供 write_*() 函数族将数据写回到磁盘中，如函数 write_csv()。如果想将数据导为 Excel 文件，可以使用函数 write_excel_csv()。这类函数的主要参数有：

- x：数据框
- file：保存文件的位置
- na：设定如何写入缺失值
- append：导出内容是否追加到现有文件

tidyverse 也提供了导入其他类型的数据的包，如：haven 包读取 Stata 文件，readxl 包读取 Excel 文件（.xls 和.xlsx）。

### 列表

列表可以包含类型、长度均不同的对象，甚至能够包含其他列表。这种性质使得列表能方便地打包多种对象，并清晰地表示多层结构。

### 创建列表

函数 list() 创建列表，names() 给列表成分命名：

```
set.seed(123) # 设置随机种子
l = list(n=rnorm(1),chr=letters[1:3], lo=rnorm(4)>0)
# rnorm()生成标准正态随机数
l
#> $n
#> [1] -0.5604756
#>
#> $chr
#> [1] "a" "b" "c"
```

```
#>
#>  $lo
#>  [1]  FALSE  TRUE  TRUE  TRUE
names(l)=NULL #移除名称
names(l)=c("N","CHR","LO") #修改列表成分名
```

### 列表取子集

- 使用 [ ] 提取子列表。与向量类似，可以通过表示索引的整数向量、逻辑向量和字符向量提取。

```
str(l[1]) #通过str()函数查看数据结构
#> List of 1
#>  $ N: num -0.56
```

- 使用 [[ ]] 提取列表单个成分。

```
str(l[[2]])
#> chr [1:3] "a" "b" "c"
```

- 使用 $ 通过成分名提取列表单个成分。

```
str(l$LO)
#> logi [1:4] FALSE TRUE TRUE TRUE
```

### 列表的转换

函数 as.list() 可将向量转换为列表：

```
as.list(c(3,1))
#> [[1]]
#> [1] 3

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

unlist() 可将列表转换为一个向量，各元素名称由原列表对应成分名称加对应序号组成：

```
str(unlist(l))
#> Named chr [1:8] "-0.560475646552213" "a" "b" "c" "FALSE" ...
#> - attr(*, "names")= chr [1:8] "N" "CHR1" "CHR2" "CHR3" "LO1" ...
```

purrr 包还提供了若干函数对列表进行操作：

- flatten()：将列表展平至只有一层
- prepend()：在列表顶端添加元素
- keep()：保留满足条件的元素
- discard()：删除满足条件的元素

## 字符串

### 创建字符串

使用一对单引号或双引号便可创建字符串：

```
ch1="What is Econometrics?"
ch1
#> [1] "What is Econometrics?"
```

tidyverse 中的 stringr 包专门用来处理字符串数据，对于字符向量，此包中的函数将自动针对向量中的每个字符串元素进行运算。

- 修改长度
  - str_length()：返回字符串的长度。
  - str_pad()：将字符串扩充至指定长度。
  - str_trunc()：将字符串截断至指定长度。

```
ch2 = c("ab","TTT","Q-R")
str_length(ch2)
#>  [1] 2 3 3
str_pad(ch2, 4, pad="-") #pad参数指定填充符
#>  [1] "--ab" "-TTT" "-Q-R"
x <- "This string is moderately long."
str_trunc(x, 10, ellipsis = "...")
#注：截断长度不能短于省略符参数ellipsis的长度
#>  [1] "This st..."
```

- 合并：str_c()
- 拆分：str_split()
- 排序：str_sort()

```
#注意间隔符参数sep和collapse的作用
str_c(c("x","y","z"),c("A","B","C"),sep=".")
#str_c将对应的字符串两两相连，中间用.分割
#> [1] "x.A" "y.B" "z.C"
str_c(c("x","y","z"),c("A","B","C"), sep=".", collapse="-")
```

```
#str_c先根据sep的设置将对应的字符串两两相连，中间用.分割，
#再根据collapse的设置将所得字符串再次连接，中间用-分割，最终得到一个字符串。
#> [1] "x.A-y.B-z.C"

str_split("q-eeF-WCAR", pattern="-") # pattern为指定的分隔符
#> [[1]]
#> [1] "q"   "eeF" "WCAR"

#变量ch2接上例
str_sort(ch2, decreasing = TRUE, locale = "en") # 降序排列，语言为英文
#> [1] "TTT" "Q-R" "ab"
```

### 字符串取子集

str_sub() 可根据指定位置提取字符串的一部分：

```
ch3 = c("long","short","big","small","thick")
str_sub(ch3, start=2, end=4)
#若字符串长度不足，则取到最后。如"big"，取子集为"ig"
#> [1] "ong" "hor" "ig" "mal" "hic"
```

### 字符串匹配

- str_detect()：检验是否能匹配指定模式
- str_starts()：检验是否以指定模式开头
- str_ends()：检验是否以指定模式结尾
- str_which()：返回匹配索引值
- str_count()：计算匹配个数

```
# 变量ch3接上例
str_detect(ch3, "h")
#> [1] FALSE TRUE FALSE FALSE TRUE
str_which(ch3, "h")
#> [1] 2 5
str_count(ch3, "l")
#> [1] 1 0 0 2 0
str_starts(ch3, "l")
#> [1]  TRUE FALSE FALSE FALSE FALSE
str_ends(ch3, "g")
```

```
#> [1] TRUE FALSE TRUE FALSE FALSE
```

### 日期和时间

R 中表示日期和时间的数据可以分为 3 类：

- 日期：在 tibble 中显示为 <date>
- 时间：一天中的某个时刻。在 tibble 中显示为 <time>
- 日期时间：相当于日期加时间。在 tibble 中显示为 <dttm>

tidyverse 的 lubridate 包可以便捷地创建、管理日期时间数据。该包在加载 tidyverse 时不会自动载入，故需手动加载。

### 创建日期时间

我们通常使用字符串创建日期时间。lubridate 提供了一类辅助函数实现该功能，其函数名由 y、m、d 和 h、m、s 组合，组合顺序按照字符串中年、月、日和小时、分钟、秒的顺序。这类函数总能准确识别字符串所代表的日期时间值。

```
library(lubridate) # 加载lubridate
ymd("2020,08,11")
#> [1] "2020-08-11"
dmy("15th,January,2018")
#> [1] "2018-01-15"
dmy_hm("12-10-2022 21:34")
#> [1] "2022-10-12 21:34:00 UTC"
mdy_hms("12/30/2021 10:00:30")
#> [1] "2021-12-30 10:00:30 UTC"
```

函数 make_date() 和 make_datetime() 则通过各个成分创建日期时间数据：

```
make_date(2022,6,10)
#> [1] "2022-06-10"
make_datetime(2022,6,10,8,30,10)
#> [1] "2022-06-10 08:30:10 UTC"
```

as.date() 将日期时间数据转换为日期数据；as.datetime() 将日期数据转换为日期时间数据。

### 获取日期时间成分

如果想提取出日期中的独立成分，可以使用以下函数：

- year()、month()、hour()、minute()、second()
- wday()、mday()、yday()：分别显示该日期是在一周、一月、一年中的第几天。

**注** R 默认周日为一周中的第一天。

### 因子

因子用作数据分类，只能在固定集合中取值。该固定集合称为因子的水平（level）。

### 创建因子

函数 factor(*x, levels, labels*) 用来创建因子：

- *x* 为创建因子的数据向量；
- *levels* 为因子的水平，默认取 *x* 中的非重复元素，默认顺序为英文字母顺序；
- *labels* 为与 *levels* 一一对应的标签。

```
ch=c("Thur.","Wed.","Thur.","Fri.","Tue.","Mon.")
f=factor(ch)
f
#> [1] Thur. Wed. Thur. Fri. Tue. Mon.
#> Levels: Fri. Mon. Thur. Tue. Wed.
```

函数 levels() 可以访问因子水平：

```
#修改因子水平
levels(f) = c("周五","周一","周四","周二","周三")
f
#> [1] 周四 周三 周四 周五 周二 周一
#> Levels: 周五 周一 周四 周二 周三
```

### 处理因子数据

tidyverse 中的 forcats 包提供了一系列处理因子数据的函数：

- fct_count()：统计数据中各因子水平的频数、占比；
- fct_lump_min()：把频数小于某数的因子水平归为其他类；
- fct_recode()：修改某个因子水平。

```
fct_count(f) #统计因子数据频数
#>  # A tibble: 5 x 2
#>    f        n
#>    <fct> <int>
#>  1 周五      1
#>  2 周一      1
#>  3 周四      2
#>  4 周二      1
```

```
#>  5 周三     1

#将频数小于2的归为“其他”
fct_lump_min(f,2, other_level = "其他")
#> [1] 周四 其他 周四 其他 其他 其他
#> Levels: 周四 其他

fct_recode(f, 礼拜四="周四")
#> [1] 礼拜四 周三 礼拜四 周一  周二  周五
#> Levels: 周一 周五 礼拜四 周二 周三
```

**注** R 以整型向量储存因子数据，每个因子水平对应一个整型数据，在处理时不能将其作为字符型进行操作。

## B.3 基本语法

在正式学习 tidyverse 的数据管理工具前，我们介绍一些基础操作，包括管道、控制流以及自定义函数。

### 管道

管道操作符%>% 是一种强大的工具。它就像生产线上的管道，将前一步的结果传送给下一步的函数。使用管道可以清楚表示由多个操作组成的操作序列，提高了代码的可读性，也避免使用过多的中间变量。

我们举一个简单的例子：

```
library(tidyverse)
sqrt(81) %>% log() %>% cos() %>% exp() %>% round(3)
```

该式等价于

```
round(exp(cos(log(sqrt(81)))),3)
```

支持管道操作是 tidyverse 的核心原则之一。在下一章介绍 dplyr 包处理数据时，我们会频繁使用%>% 。届时，读者将更加体会到它带来的便捷。

使用管道时，需要特别注意数据输入的位置，适时使用 `.` 占位：

- 管道默认将数据传递至下一个函数的第一个参数，此时占位符可省略。

```
c(7,-4,5,NA) %>%
  mean(.,na.rm = TRUE)
# .可省略
```

```
c(7,-4,5,NA) %>%
  mean(na.rm = TRUE)
```

- 若数据输入的位置并非函数的第一个参数，则需占位符。

```
iris %>%
  plot(Sepal.Length~Petal.Length,data = .) #iris数据集传递给第二个参数
```

- 在管道中使用提取操作，通常也需要占位符。

```
set.seed(123)
df <- tibble(
  x = rnorm(4), #生成标准正态分布随机数
  y = runif(4) #生成区间[0,1]内的均匀分布随机数
)
# 提取列 x
df %>% .$x
#> [1] -0.560 -0.230 1.559 0.071
df %>% .[["x"]]
#> [1] -0.560 -0.230 1.559 0.071
```

### 控制流

通常，R 的语句是自上而下执行的。但有时我们希望仅在满足特定条件的情况下执行某些语句，或重复执行某些语句，此时便需要控制流结构。

### if 语句

函数 if() 可以有条件的执行代码。其形式如下：

```
  if(<conditon>){
    <statement1 when condition is TRUE>
  } else {
    <statement2 when condition is FALSE>
  }
```

需注意，*<conditon>* 是一元逻辑向量。常见的 *<condition>* 由比较运算符、逻辑运算符、类型判断函数族 is_*() (purrr 包) 或 is.*() (R 基础包) 构成。

- 比较运算符：`>`、`>=`、`<`、`<=`、`!=`（不等于）、`==`（等于）
- 逻辑运算符：`|`（或）、`&`（且）

```
library(tidyverse)
```

```
x <- 9L
if(!is_integer(x)) {
  print("X is not integer") # 若x不是整型，则输出"x不是整型"
} else {
  print(x^2) # 否则，输出x的平方
}
#> [1] 81
```

当涉及多重条件时，我们可以将多个 if 语句串联起来：

```
if(<con1>){
  <statement1>
} else if (<con2>){
  <statement2>
} else {
  <statement3>
}
```

ifelse() 是 if() 的向量化版本，其语法为：ifelse(*con*, *statement1*, *statement2*)。当条件为多元逻辑值时，需使用 ifelse() 函数。

```
# 对多元p值向量，标记其在0.05水平下的显著性
p <- c(.0014, .1021, .2478, .0003, .8216)
test <- ifelse(p <.05, "Significant", "Not Significant")
test
#> [1] "Significant"  "Not Significant" "Not Significant" "Significant"
#> [5] "Not Significant"
```

### for 结构

for 循环重复地执行一个语句，直到循环变量不再包含在序列中：

```
for (<i in sequence>){ # i为循环变量
  <statement>
}
```

我们以计算 R 自带的 mtcars 数据集每列的均值为例，该数据包含 32 种汽车在性能方面的 11 个指标。

```
# 为输出结果分配足够的空间
mtcars_mean <- vector("double", ncol(mtcars))
```

```
for (i in seq_along(mtcars)) { # 循环序列
  mtcars_mean[[i]] <- mean(mtcars[[i]]) # 循环体
}
```

其中，vector() 用于创建给定长度的空向量，包括向量类型和向量长度两个参数；seq_along() 与我们熟悉的 1:length() 的作用基本相同，但在遇到意外值时（如长度为 0 的向量），会进行提示。

### while 结构

当事先不确定循环序列的长度时，我们应使用 while 循环。while 通过一个逻辑判定准则来控制循环，当条件不再为真时停止循环：

```
while(<conditon>){
  <statement>
}
```

以模拟掷硬币过程为例，我们想求出连续出现 3 次正面向上所需的投掷总次数：

```
set.seed(123)
total <- 0 # 设置投掷总次数的初值
heads <- 0 # 设置连续正面向上的次数的初值
while (heads < 3) { # 正面向上次数小于3为循环条件
  if (sample(c("T", "H"), 1) == "H") {
      #sample()函数模拟随机掷硬币，“H”代表正面向上
    heads <- heads + 1 # 当正面向上，heads计数累加
  } else {
    heads <- 0 #当出现了反面向上，heads计数从零重新开始
  }
  total <- total + 1
} # 每次循环，total计数加1
total
#> [1] 8
```

### purrr 包实现循环

循环结构通常需要花费很长时间。tidyverse 的 purrr 包提供的 map 族函数能够在迭代中节省大量时间。

### 单参数迭代

map_*() 函数族对输入的向量进行循环，将其中的每个元素带入运算，返回新的向量或列表。map_*($x$, $f$, ...) 中的 $x$ 为数据对象，$f$ 为应用到数据的函数，... 为传递给 $f$ 的其他参数。每种类型的输出都对应一个函数：

- map() 返回列表；
- map_lgl() 返回逻辑型向量；
- map_int() 返回整型向量；
- map_dbl() 返回双精度型向量；
- map_chr() 返回字符型向量。

在前文介绍 for 结构中，我们计算了 mtcars 数据集每列的均值。下面我们使用 map_dbl() 实现上述操作：

```
mtcars_mean2 <- mtcars %>%
  map_dbl(mean) # 此处省略了.占位符，等价于map_dbl(.,mean)
```

R 基础包中的 apply 函数族与单参数迭代的 map 函数族类似：

- apply() 函数可以遍历数组的所有维度；
- lapply() 函数与 map_*() 函数功能基本相同；
- sapply() 函数是对 lapply() 简化输出的版本。

以 apply() 为例，同样计算 mtcars 数据集每列的均值：

```
mtcars_mean3 <- apply(mtcars,2,mean)
# 第二个参数MARGIN代表维度
# 对于二维数据框，维度为1表示对行进行运算，为2表示对列进行运算
```

我们更推荐使用 purrr 包的函数，因为它们具有更为一致的参数，且在多个相关输入的同步迭代上具有更强大的功能。

### 多参数迭代

函数 map2() 和 pmap() 针对多个参数需要同步迭代的情况。例如，我们需要生成几组均值、标准差都不同的正态分布随机数：

```
set.seed(123)
mu <- list(0,1,-1)
sigma <- list(1,2,3)
map2(mu, sigma, rnorm, n=4) %>% # 注意参数位置
  str() # str()用来查看数据的内部结构
#> List of 3
#> $ : num [1:4] -0.5605 -0.2302 1.5587 0.0705
#> $ : num [1:4] 1.26 4.43 1.92 -1.53
```

```
#> $ : num [1:4] -3.0606 -2.337 2.6722 0.0794
```

**注** 每次迭代时发生变化的参数（此例为 mu 和 sigma）需放在指定函数（此例为 rnorm）的前面，保持不变的参数（此例为 n）需放在指定函数的后面。

函数 pmap() 可以将一个列表作为参数，对于列表的所有变量，进行同步迭代。仍以生成正态分布随机数为例，此时要求三组分布的均值、方差和样本数量各不相同：

```
# 随机种子、变量mu和sigma接上例
num <- list(4,6,8)
# 为par各变量命名。函数rnorm(n,mean,sd)将依据变量名匹配参数。
par <- list(mean = mu, sd = sigma, n = num)
# 若省略命名步骤，函数rnorm(n,mean,sd)将按照变量位置引入参数。
# 此时，变量num、mu和sigma的位置与函数rnorm()的参数n、mean和sd位置
# 必须对应。
par <- list(num, mu, sigma)
par %>%
  pmap(rnorm) %>%
  str()
#> List of 3
#> $ : num [1:4] 0.401 0.111 -0.556 1.787
#> $ : num [1:6] 1.9957 -2.9332 2.4027 0.0544 -1.1356 ...
#> $ : num [1:8] -4.08 -3.19 -2.88 -6.06 1.51 ...
```

我们也可以将参数以一个 tibble 的形式引入，这样确保每一列都具有名称，在匹配参数时不易出错。

```
#使用tribble()按行填充
par2 <- tribble(
  ~mean, ~sd, ~n,
  0, 1, 4,
  1, 2, 6,
  -1, 3, 8
)
par2 %>%
  pmap(rnorm)
```

### 调用不同函数的迭代

当不仅传给指定函数的参数不同，指定函数本身也不同时，可以使用 invoke_map() 函数。invoke_map(*f*, *x*, ...) 的第一个参数为指定函数组成的列表或包含函数名称的字符向

量，其余参数与 map_*() 类似。

例如，除正态分布外，我们还需生成几组参数不同的学生 $t$ 分布随机数：

```
# 生成一组正态分布随机数及两组参数不同的学生t分布随机数
set.seed(123)
f <- list(rnorm,rt,rt) # 定义函数列表
#也可为函数名称组成的字符向量 f <- c("rnorm","rt","rt")
param <- list(
  list(mean=1, sd = 2),
  list(df = 4),
  list(df = 10)
  )
invoke_map(f, param, n = 4) %>%
  str()
#> List of 3
#> $ : num [1:4] -0.121 0.54 4.117 1.141
#> $ : num [1:4] 0.0878 1.1007 -1.1802 0.2863
#> $ : num [1:4] 0.548 -0.847 0.882 -0.666
```

### 自定义函数

至此，我们涉及到的函数均为现有函数。在实际中，往往需要自行定义函数以实现某些操作。R 中定义函数的结构如下：

```
<name> <- function(<arglist>){
  <statements>
  return(<object>)
}
```

*name* 是自定义的函数名。我们通过函数名对函数进行调用。此外，R 也支持匿名函数，它们通常被用作另一些函数的参数。

```
map(
  c(-3, -0.7, 2.1, 0.5), function(x){
  if(abs(x) <= 1){x^2} else{1}
  } # 此处定义的匿名函数作为map()的第二个参数
)
```

值得注意的是，purrr 包支持创建匿名函数的快捷方式：由 `~` 和含有特殊参数符号的函数表达式组成：

- 单个参数：`.`
- 两个参数：`.x`和`.y`
- 多个参数：`..1`, `..2`, `..3`等

```
map(c(-3, -0.7, 2.1, 0.5),
    ~if(abs(.) <= 1){.^2} else{1} #此处用.指代参数
 )

set.seed(123)
par=list(rnorm(5),runif(5),rt(df=10,5))
pmap(par,~log(abs(..1 + ..2 + ..3))) # 此处用..1和..2等指代参数
```

*arglist* 是函数的参数。可以在定义时直接赋值来设定某参数的默认值，也可以使用特殊参数`...`使得函数能够接受任意长度的输入。

```
# 定义两种数据调整方式，将其中常用的一种设置为默认
adjust <- function(x, type=1){
  if(type==1) {
    new = (x-mean(x))/sd(x)
  } else {
    new = (x-mean(x))/(max(x)-min(x))
  }
  return(new)
}

# 自定义函数，包含任意长度的参数
add_pos <- function(x, ...){
  args = list(...) # 通过list()获得所有参数
  x + unlist(args[args>0])
  # 将x与剩余正值参数相加，其中unlist()函数将列表转换为向量以进行数值运算
}
add_pos(x=3,-1,0,28,-7,3) #调用含有参数...的函数时，最好对其余参数命名
#> [1] 31 6
```

可以通过 return() 制定函数的返回值。若不使用 return() 语句，则默认函数体最后一个语句的结果为返回值。

## B.4 数据处理

tidyverse 中的 dplyr 包是数据处理的有效工具，主要包括以下 6 个核心函数：

- filter()：选取满足条件的某些行；
- arrange()：对行重新排序；
- select()：选取满足条件的某些列；
- mutate()：使用现有变量的函数创建新变量；
- summarize()：进行统计性汇总；
- group_by()：进行分组操作。

前 5 个函数的工作方式一致：第一个参数为输入的数据框，随后的参数使用不带引号的变量名，函数的返回结果为一个新的数据框。group_by() 函数则往往与前 5 个函数一起使用，改变其他函数的作用范围，实现分组操作。本节中，我们会频繁地使用管道%>%以便更简洁地完成复杂的操作。

我们以虚拟的学生体测成绩数据集为例，展示常用的数据操作。该数据包含学号、年龄、性别、短跑成绩、跳绳成绩以及仰卧起坐成绩。

```
library(tidyverse)
fit_test <- read_csv('~/fit_test.csv')
#> Rows: 150 Columns: 6
#> --------------- Column specification ---------------
#> Delimiter: ","
#> chr (1): sex
#> dbl (5): stu_ID, age, sprint, rope_skipping, sit_up
```

注：dplyr 包的函数不修改输入，若想保存函数结果，需另使用赋值语句。

## 行操作

### 删除重复行或有缺失的行

dplyr 包的 distinct() 函数可以删除重复行；tidyr 包的 drop_na() 函数可以删除缺失值所在行。我们可以通过列名限制这些行操作的作用范围：

```
#删除完全重复的行
fit_test %>%
  distinct()

#删除在某些变量上重复的行
fit_test %>%
  distinct(stu_ID, .keep_all = TRUE)
#.keep_all = TRUE 返回所有列,否则返回指定列

#删除所有含缺失值的行
fit_test %>%
```

```
  drop_na()

#删除在某些变量上有缺失值的行
fit_test %>%
  drop_na(stu_ID:sex)# 作用范围由stu_ID到sex
```

### 过滤行

filter() 函数可以筛选满足条件的行：

```
#选出age为19或20，sprint在[8，9]之间，且rope_skipping>190的行
#同时保留缺失值所在行
fit_test %>%
  filter(age %in% c(19,20) & between(sprint, 8, 9) &
           (is.na(rope_skipping) | rope_skipping>190)
         )
#> # A tibble: 4 x 6
#>     stu_ID  age sex    sprint rope_skipping sit_up
#>      <dbl> <dbl> <chr> <dbl>        <dbl>  <dbl>
#> 1 41823064   19 female  8.48           NA     24
#> 2 41815560   19 male    8.69          200     37
#> 3 41820089   19 female  8.81           NA     33
#> 4 41825389   20 male    8.47          199     33
```

### 调整行序

arrange() 函数依据列名或表达式对行进行排序，默认为升序排列：

```
# 先按age升序排列，再在此基础上按sit_up升序排列
fit_test %>%
  arrange(age,sit_up)
#> # A tibble: 150 x 6
#>     stu_ID  age sex    sprint rope_skipping sit_up
#>      <dbl> <dbl> <chr> <dbl>        <dbl>  <dbl>
#> 1 41814922   18 female  8.12          183     18
#> 2 41824001   18 female 10.7           179     18
#> 3 41821331   18 female  8.91          168     20
#> 4 41816995   18 female  8.23          148     24
#> # ... with 146 more rows
```

```
#> # i Use `print(n = ...)` to see more rows
```

使用 desc() 可以实现降序排列，且无论升序降序，缺失值总排在最后。我们可以通过 is.na() 使得缺失值排在最前。

```
# 按sprint降序排列，且缺失值在最前
fit_test %>%
  arrange(desc(is.na(sprint)),desc(sprint)) # 两个desc()语句顺序不可换
#> # A tibble: 150 x 6
#>    stu_ID  age sex    sprint rope_skipping sit_up
#>     <dbl> <dbl> <chr> <dbl>        <dbl> <dbl>
#> 1 41817084   21 male    NA           172    39
#> 2 41824001   18 female  10.7         179    18
#> 3 41823450   21 female  10.6         177    33
#> 4 41819242   20 female  10.6         148    20
#> # ... with 146 more rows
#> # i Use `print(n = ...)` to see more rows
```

## 列操作

### 选择列

我们可以向 select() 函数传入各种参数实现选择列的操作，最简单的是直接使用列名或索引：

```
fit %>%
  select(sex, sit_up) # 或select(3, 6)
```

我们还能在 select() 中嵌套其他函数，实现多样化的选择要求：

- last_col(*p*)：选择倒数第 (p+1) 列；
- starts_with(*"BG"*)：选择列名以*"BG"* 开头的列；
- ends_with(*"ED"*)：选择列名以*"ED"* 结尾的列；
- contains(*"CN"*)：选择列名含有*"CN"* 的列；
- num_range(*"x", 1:3*)：选择列*"x1"*, *"x2"*, *"x3"*；
- where(*fun*)：将函数 *fun* 作用到所有列，选择返回结果为 TRUE 的列。常见的 *fun* 为类型判断函数 is.*()，也可以自定义函数，支持 purrr 快捷写法的匿名函数。

```
#选择唯一值的数量小于5的列
#函数n_distinct()计算变量唯一值的个数
fit_test %>%
  select(where(~n_distinct(.) < 5))
```

```
#> # A tibble: 150 x 1
#> sex
#> <chr>
#> 1 male
#> 2 male
#> 3 female
#> 4 male
#> # ... with 146 more rows
```

### 调整列序

列的顺序是它们被选择的顺序，因此，我们仍然可以使用 select() 调整列序，函数 everything() 将返回未被选择的剩余列，两者结合便能改变列的顺序：

```
#将sex移到第一列
fit_test %>%
  select(sex, everything())
#> # A tibble: 150 x 6
#>   sex     stu_ID  age sprint rope_skipping sit_up
#>   <chr>    <dbl> <dbl> <dbl>        <dbl> <dbl>
#> 1 male  41827587   19   7.76          164    55
#> 2 male  41815464   18   7.18          175    48
#> # ... with 148 more rows
```

以上方法在将某些变量移至开头时显得尤为方便。但对于更为复杂的顺序调整，我们可以使用函数 relocate() 将所选列移动到指定位置，其使用格式为：

```
relocate(.data, ..., .before = NULL, .after = NULL)
# .data为一数据框
# ...为要移动的列
# .before和.after规定了移动的位置，只需设置其中之一
```

例如：

```
#将字符型变量移动至最后
fit_test %>%
  relocate(where(is.character), .after =last_col())
#> # A tibble: 150 x 6
#>    stu_ID  age sprint rope_skipping sit_up sex
#>     <dbl> <dbl> <dbl>        <dbl> <dbl> <chr>
```

```
#> 1 41827587   19   7.76          164     55 male
#> 2 41815464   18   7.18          175     48 male
#> # ... with 148 more rows
```

### 对列重命名

函数 rename(*new* = *old*) 可以用来更改指定列的列名：

```
#将stu_ID更名为学号
fit_test %>%
  rename(学号=stu_ID)
#> # A tibble: 150 x 6
#>        学号   age sex   sprint rope_skipping sit_up
#>       <dbl> <dbl> <chr> <dbl>        <dbl>  <dbl>
#>  1 41827587   19 male    7.76          164     55
#>  2 41815464   18 male    7.18          175     48
#> # ... with 148 more rows
```

当需对所有列重新命名时，可以使用函数 set_names()，此时列名需加引号：

```
#为所有列重新命名
fit_test %>%
  set_names("学号","年龄","性别","短跑","跳绳","仰卧起坐")
#>  # A tibble: 150 x 6
#>       学号  年龄 性别    短跑  跳绳 仰卧起坐
#>      <dbl> <dbl> <chr> <dbl> <dbl>  <dbl>
#> 1 41827587   19 male   7.76   164      55
#> 2 41815464   18 male   7.18   175      48
#> # ... with 148 more rows
```

### 创建新列

我们有时需要通过现有变量，计算出新的变量。函数 mutate() 可以同那些创建新变量的函数一起使用，将新变量并入数据框的最后。如果只想保留新变量，可以使用 transmute() 函数。创建新变量的常用函数有：

- 累计函数：cumsum()、cumprod()、commin()、commax()、cummean() 分别返回序列的累计和、累计积、累计最小值、累计最大值、累计均值。

```
x <- c(1,-6,2,0,5)
cumsum(x) # 累计和
```

```
#> [1]  1 -5 -3 -3  2
cummax(x) # 累计最大值
#> [1] 1 1 2 2 5
```

- 排秩函数：min_rank() 返回单个元素在序列升序排列中的“名次”。缺失值不参与排名，仍返回 NA。

```
#将sprint的名次（男女分开排序）作为新变量rank
fit_test %>%
  group_by(sex) %>% # 男女两组分别排序
    mutate(rank = min_rank(sprint))
#> # A tibble: 150 x 7
#> # Groups:  sex [2]
#>    stu_ID  age sex    sprint rope_skipping sit_up rank
#>     <dbl> <dbl> <chr>  <dbl>        <dbl>  <dbl> <int>
#> 1 41827587   19 male    7.76          164     55    21
#> 2 41815464   18 male    7.18          175     48     7
#> 3 41819710   20 female  9.93          166     39    47
#> 4 41811076   18 male    9.23          155     32    66
#> # ... with 146 more rows
```

## 统计描述

计算数据的描述统计量主要通过 group_by()、summarize() 和各种描述函数组合实现。输出结果由原变量和描述变量构成的新列组成。常用的描述函数有：

- 计数函数：
  - n() 不需任何参数便返回各分组的观测值个数；
  - n_distinct($x$) 返回 $x$ 唯一值的数量；
  - sum(is.na($x$)) 可计算缺失值个数；
  - count($x$) 计算变量 $x$ 各取值的个数。
- 聚合函数：sum()、mean()、median()、sd() 等。当数据含有缺失值时，可设置参数 na.rm=TRUE，除去缺失值。

summarize() 也常与 across() 函数结合，对满足条件的多列同时进行统计。across(*col*, *fun*) 将函数 *fun* 应用到满足条件的所有列上。

```
#根据age,sex分组，并统计除stu_ID外其余变量的平均值与中位数
fit_test %>%
  group_by(age,sex) %>%
    summarize(across(-stu_ID, list(Mean=mean,Median=median), na.rm =
```

```
      TRUE))

#> # A tibble: 10 x 8
#> # Groups:  age [5]
#>     age sex   sprint_Mean sprint_Median
#>   <dbl> <chr>       <dbl>         <dbl>
#> 1    18 female       9.16          9.18
#> 2    18 male         8.35          8.49
#> 3    19 female       9.77          9.86
#> 4    19 male         8.31          8.54
#> 5    20 female       9.58          9.93
#> 6    20 male         8.34          8.49
#> 7    21 female       9.36          9.28
#> 8    21 male         8.22          8.3
#> 9    22 female       9.56          9.80
#>10    22 male         8.11          8
#> # … with 4 more variables:
#> #   rope_skipping_Mean <dbl>,
#> #   rope_skipping_Median <dbl>, sit_up_Mean <dbl>,
#> #   sit_up_Median <dbl>
```

## B.5 数据可视化

利用 R 语言对数据进行可视化是 R 语言的重要用途之一，ggplot2 是功能强大的绘图包，包括 10 个部件：

- **数据 (data)**
- **映射 (mapping)**
- **几何对象 (geom)**
- 标度 (scale)
- 统计变换 (stats)
- 坐标系 (coord)
- 位置 (position)
- 分面 (facet)
- 主题 (theme)
- 存储和输出 (output)

其中，前三个为必要部件，其余部件均有默认配置。其语法结构如下：

```
ggplot(data = <DATA>, mapping = aes(<MAPPING1>)) +
```

```
  <GEOM_FUNCTION>(mapping = aes(<MAPPING2>),
                  stat = <STAT>,
                  position = <POSITION>) +
  <SCALE_FUNCTION> +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <THEME_FUNCTION>

ggsave()
```

需要注意的是 ggplot2 采用分层语法，各函数间通过 `+` 连接，且 `+` 只能放在行尾。ggplot() 层的映射（mapping）可以被其他层使用，此时我们常在其他层省略该共用对象，而其他层的映射只能在该层内使用。我们将在后续示例中进一步说明这一点。

### 数据、映射和几何对象

#### 映射

ggplot2 接收数据框作为绘图对象，映射将数据框中的变量映射为可见的图形属性，映射函数 aes() 常用的图形属性如下：

- x：$x$ 轴；
- y：$y$ 轴；
- color：点、线以及填充区域的边界颜色；
- fill：填充区域的填充色；
- size：大小；
- shape：形状；
- alpha：透明度；
- linetype：线的类型，如实线、虚线、点线等。

注意到，$x$ 轴和 $y$ 轴的设置本身就是图形属性，除 $x, y$ 这两个最基本的图形属性外，创建其余图形属性均会生成对应图例（legend），用以清楚展示变量和图形属性之间的映射关系。我们也可以将坐标轴理解为 $x, y$ 属性的图例，B.5 小节会介绍如何修改图例。

#### 几何对象

建立好变量与图形属性之间的关系后，可以通过几何图形来表示数据关系，这便是几何对象，常见的几何对象函数有：

- geom_point()：散点图；
- geom_line()：折线图；
- geom_smooth()：平滑曲线图；
- geom_bar()：条形图；

- geom_histogram()：直方图；
- geom_boxplot()：箱线图；
- geom_density()：概率密度图。

几何对象可以通过 position 参数调整位置属性，如改变条形图的堆叠方式，散点图的散点位置等。我们也可以为几何对象设置图形属性。区别于映射中的图形属性，几何对象的图形属性只改变图形外观，并不传达任何关于变量的映射信息。故需在映射函数 aes() 的外部进行设置，所赋的值不再为变量名，而是该图形属性的具体值。多个几何对象还可以在同一图纸上依次叠加，遵循分层语法即可。

以 iris 鸢尾花数据集为例，绘制 Sepal.Length（花萼长）与 Sepal.Width（花萼宽）的散点图，将 Species 映射为点的颜色，揭示花的种类，并在散点图的基础上添加平滑曲线。

```
iris %>%
  ggplot(aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_point(aes(color = Species), #使用ggplot()中的x和y映射
             shape = 2, position = "jitter") +
  #在aes()外的shape为几何对象的图形属性，2代表散点为三角形；
  #参数position调整几何对象的位置，
  #jitter为每个数据点添加微小随机扰动，避免重叠
  geom_smooth(aes(color = Species),
              method = lm, #设置拟合方式为线性回归
              se = FALSE, #不显示置信区间
              linetype= 2) #设置线条为虚线
```

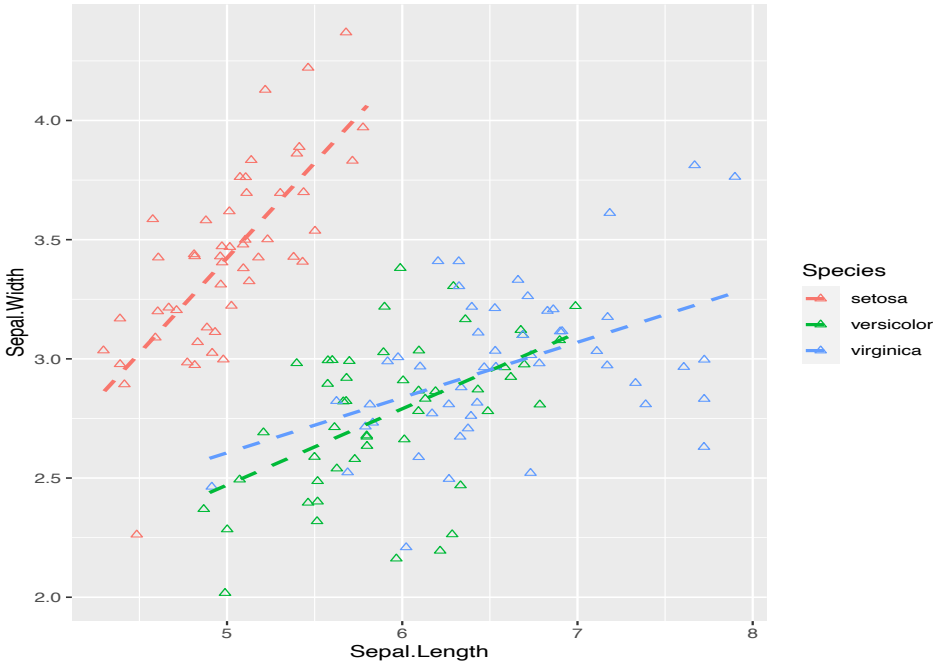


图 B.6: 花萼长与花萼宽的散点图及平滑曲线图

### 标度

针对各类图形属性、离散与连续变量，ggplot2 提供了大量标度函数，用于调整坐标轴及图例，常用的有：

- 坐标轴标度：scale_*_discrete(), scale_*_continuous(), scale_x_date()，其中 * 为 x 或 y；
- color 标度和 fill 标度：scale_*_discrete(), scale_*_gradient(), scale_*_manual()，其中 * 为 color 或 fill；
- size 标度：scale_size()；
- linetype 标度：scale_linetype_discrete()；
- shape 标度：scale_shape()。

每个标度函数内部也有一系列参数，常用参数有：

- name：更改坐标轴或图例的名称，默认为该图形属性对应变量的变量名，name = NULL 不显示名称；
- limits：限制坐标轴或图例的范围；
- breaks：控制显示在坐标轴或图例上的值；
- labels：坐标轴和图例各值的标签，与 breaks 的值一一对应。

我们通过例子展示常见的更改坐标轴或图例的操作。

### 更改坐标轴的刻度、标签和名称

scale_x_date() 可以修改日期刻度。以 ggplot2 自带的 economics 数据集为例，该数据集包含了美国 5 种经济指标 1967-2015 年的月度数据。

```
# date为日期数据，psavert为个人储蓄率，
economics[1:50,] %>%
 ggplot(aes(x=date, y=psavert/100)) +
   geom_line() +
   scale_x_date(name="日期",date_breaks = "8 months", date_labels =
       "%Y-%m") +
# 设置刻度间隔为8个月，日期形式为四位数年份-两位数月份
   scale_y_continuous(name="个人储蓄率",labels = scales::percent)
# 使用scales包的percent函数更改y轴标签，将数据由小数转化为百分数格式
```

函数 labs()、xlab()、ylab() 也可以更改名称。它们需通过 + 与几何对象函数层连接。

### 更改颜色值

根据变量的不同类型，可以采用不同的函数更改图像颜色：

- 离散变量：可以使用 scale_*_manual() 函数自定义数据对应到 color 或 fill 的颜色，其中 * 为 color 或 fill。

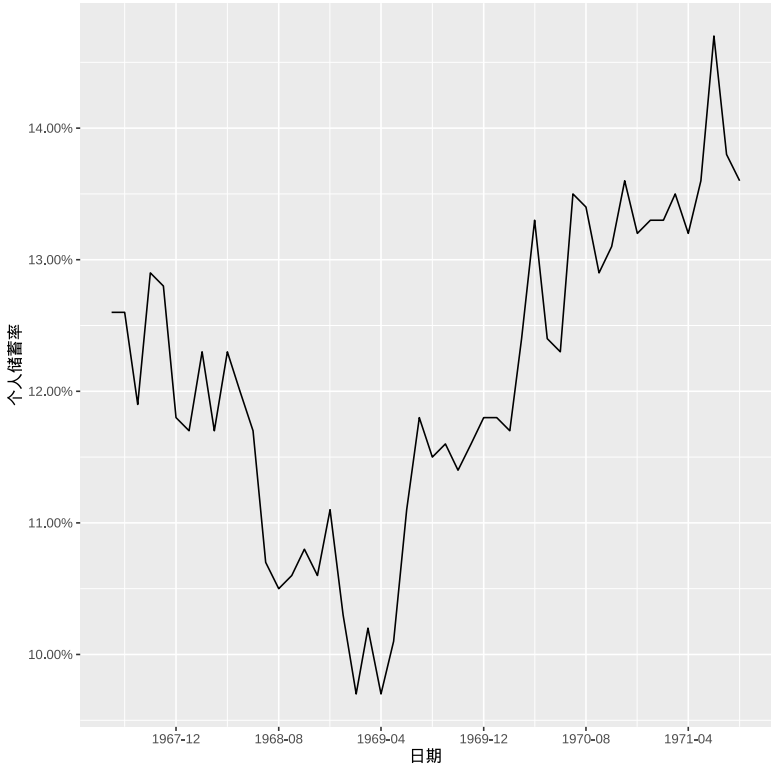


**图 B.7:** 美国个人储蓄率的日期折线图

```
#绘制Sepal.Length的直方图，并将离散变量Species映射到fill属性上，形成分块
iris %>%
  ggplot(aes(x = Sepal.Length, fill = Species)) +
  geom_histogram() +
  scale_fill_manual(values = c("orange", "pink", "lightgreen")) +
                  # 自定义Species对应的颜色
  labs(x="花萼长度", y="数量",fill="品种") # 更改坐标轴和图例名称
```

- 连续变量：可以使用 scale_*_gradient() 函数设置二元渐变色与连续变量对应，其中 * 为 color 或 fill。

```
iris %>%
ggplot(aes(x = Sepal.Width, y=Petal.Width,color = Sepal.Length)) +
geom_point() +
scale_color_gradient(name="花萼长度",low="blue",
    high="yellow",breaks=seq(4,8,by=0.7)) +
scale_x_continuous(name="花萼宽度") +
scale_y_continuous(name="花瓣宽度")
```

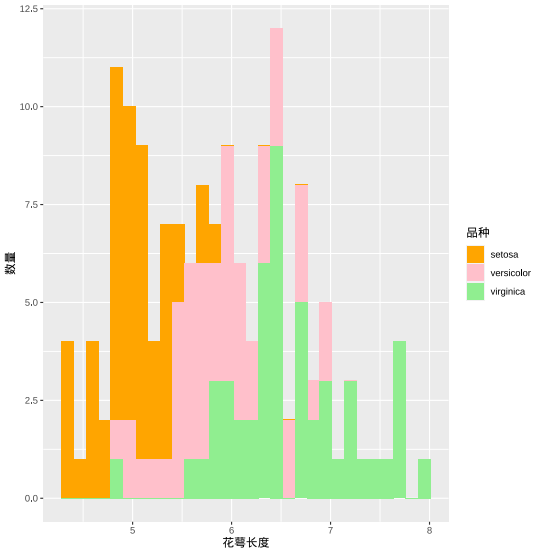


图 B.8: 花萼长度与品种的堆叠直方图

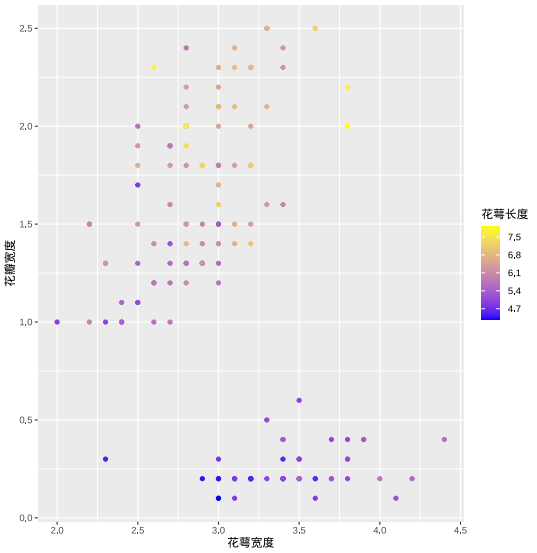


图 B.9: 花萼宽度与花瓣宽度的散点图

## 统计变换和坐标系

### 统计变换

ggplot2 将统计变换融入绘图语法中。例如，geom_boxplot() 计算数据分布的最小值、25% 分位数、中位数、75% 分位数、最大值，并显示为一个箱体。图 B.10 给出了箱体图每个部分含义的说明。

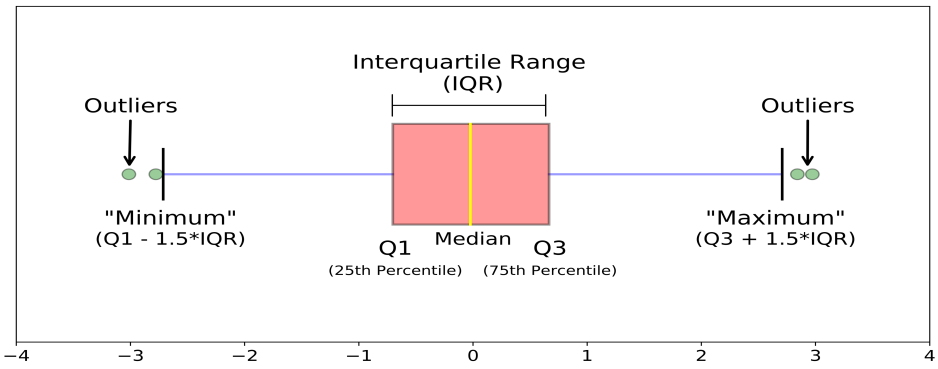


图 B.10: 箱线图组成部分图解，图片来源：https://www.kdnuggets.com/2019/11/understanding-boxplots.html

除几何对象函数外，还有一系列统计函数 stat_*()，可在绘图语法中直接计算统计量。例如，stat_ecdf() 可计算变量的经验累积分布函数。以 carData 包的 Salaries 数据集为例，该数据集包含美国某大学助理教授、副教授和教授 2008 年前 9 个月工资收入的信息。

```
library(carData)
#计算salary变量的经验累积密度分布并绘图
Salaries %>%
  ggplot(aes(salary)) +
  stat_ecdf() +
```

```
  xlab("工资") +  # 使用xlab(), ylab()更改坐标轴名称
  ylab("经验累积分布")
```

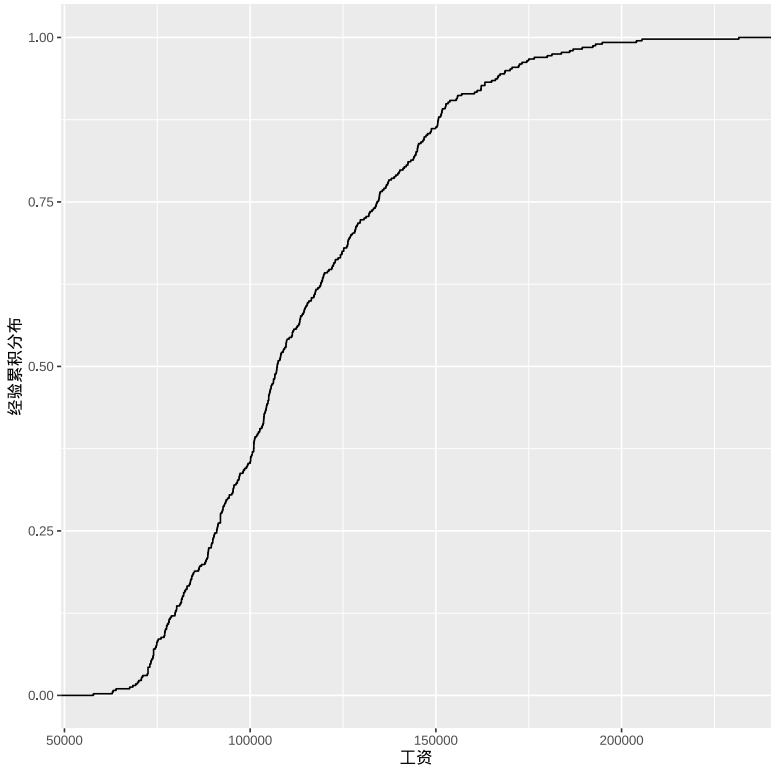


**图 B.11:** 工资的经验累积分布函数

## 坐标系

ggplot2 默认坐标系为直角坐标系，常用的坐标系函数有：

- coord_flip()：互换 $x, y$ 轴；
- coord_polar()：使用极坐标系，此时直角坐标系的条形图将转换为极坐标系下的饼图。

## 分面

数据分组绘图也可以通过分面来实现，我们仍然使用 Salaries 数据集为例进行说明：

- facet_wrap()：通过单个变量分面，参数形式为 *~x1*。

```
#以sex（性别）为分面变量，绘制yrs.service（工龄）与salary（工资）的散点图
Salaries %>%
  ggplot() +
  geom_point(mapping = aes(x = yrs.service, y = salary)) +
  facet_wrap(~sex, ncol = 2)
```

- facet_grid()：通过两个变量分面，参数形式为 *x1~x2*。

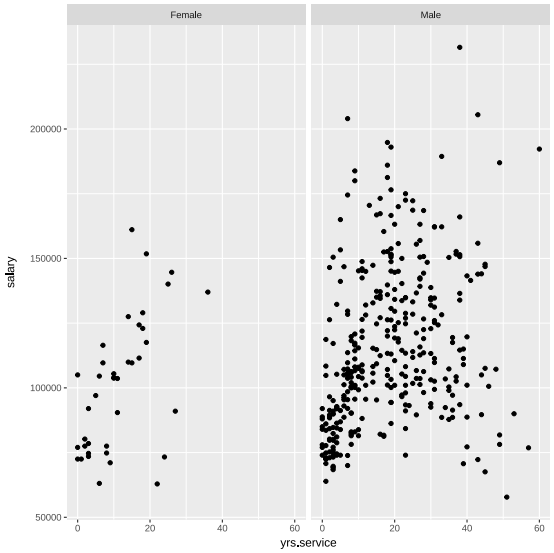


图 B.12: 工龄与工资关于性别的分面散点图

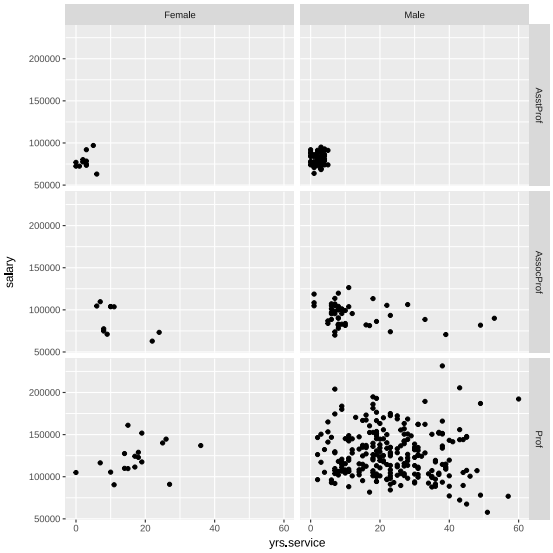


图 B.13: 工龄与工资关于性别和等级的分面散点图

```
#以rank（教授等级）和sex（性别）为分面变量
Salaries %>%
  ggplot() +
  geom_point(mapping = aes(x = yrs.service, y = salary)) +
  facet_grid(rank ~ sex)
```

### 主题、输出

主题函数层 theme() 可以通过设置参数 legend.position 更改图例位置，可选值有"none"、"left"、“right"、"bottom" 和"top"。

ggsave() 函数可以保存当前图形为 pdf 或 png 等格式文件，该函数位于绘图语句的末尾，不需要用 + 与 ggplot 各层连接。

## B.6 线性回归

线性回归模型是计量经济学中最基础的模型，R 基础包提供了拟合线性模型最基础的函数：lm( *formula*, *data* )，其中，*formula* 为需拟合的模型形式，*data* 为用于拟合的数据构成的数据框。

```
#随机模拟多元线性回归
set.seed(123)
m <- tibble(
  x1=2*rnorm(1000), x2=rnorm(1000)/3, u=4*rnorm(1000), y=-1+7*x1+2*x2+u
  ) %>%
```

```
    lm(y~x1+x2,.) #.为占位符传递数据至data参数
```

下列函数可进一步分析、应用拟合的模型：

- summary()：展示拟合模型的详细结果；
- coefficients()、fitted()、residuals()：分别列出拟合模型的估计参数、被解释变量的拟合值以及残差。这些结果也可以使用 $ 直接从拟合模型中提取；
- plot()：绘制拟合模型的诊断图，只需 *fitted_model* 一个参数，R 将自动绘制出残差与拟合值图、标准化残差正态 Q-Q 图、位置尺度图、标准化残差与杠杆图；
- predict(*fitted_model*, *newdata*)：利用拟合模型对新数据集进行预测。

```
summary(m) #查看拟合模型的统计描述

#> Call:
#> lm(formula = y ~ x1 + x2, data = .)

#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -11.3441 -2.5108 -0.1479  2.6153 13.5146

#> Coefficients:
#>             Estimate  Std. Error t value  Pr(>|t|)
#> (Intercept) -1.08374   0.12392   -8.745   < 2e-16 ***
#> x1           6.95702   0.06269    110.980 < 2e-16 ***
#> x2           2.33008   0.36942    6.307   4.26e-10 ***
```

## 致谢

此附录初稿的撰写由厦门大学-中科院“计量建模与经济政策研究中心”在读博士生孙蓓蕾完成，在此表示感谢。

# 参考文献

WICKHAM H, GROLEMUND G, 2016. R for Data Science[M]. Sebastopol: O'Reilly Media.

KABACOFF R I, 2015. R in Action[M]. 2nd ed. New York: Manning Publications.