Contents

高级数据管理

接下来，我们将讲解如何自己编写函数来完成数据处理和分析任务。首先，我们将探索控制程序流程的多种方式，包括循环和条件执行语句。然后，我们将研究用户自编函数的结构，以及在编写完成后如何调用它们。最后，我们将了解数据的整合和概述方法，以及数据集的重塑和重构方法。

各种函数的讲解

```
x <- c(1, 2, 3, 4, 5, 6, 7, 8)
mean(x)
```

```
## [1] 4.5
```

```
sd(x)
```

```
## [1] 2.44949
```

```
n <- length(x)
meanx <- sum(x)/n
css <- sum((x - meanx)**2)
sdx <- sqrt(css / (n-1))
meanx
```

```
## [1] 4.5
```

```
sdx
```

```
## [1] 2.44949
```

```
# Listing 5.2 - Generating pseudo-random numbers from
# a uniform distribution

runif(5)
```

```
## [1] 0.3940060 0.6440463 0.1677900 0.7537042 0.2963267
```

```
runif(5)
```

```
## [1] 0.1884361 0.7930731 0.2780681 0.8548928 0.5307490
```

```
set.seed(1234)
runif(5)
```

```
## [1] 0.1137034 0.6222994 0.6092747 0.6233794 0.8609154
```

```
set.seed(1234)
runif(5)
```

```
## [1] 0.1137034 0.6222994 0.6092747 0.6233794 0.8609154
```

```
# Listing 5.3 - Generating data from a multivariate
# normal distribution

library(MASS)
options(digits=3)
set.seed(1234)

mean <- c(230.7, 146.7, 3.6)
sigma <- matrix( c(15360.8, 6721.2, -47.1,
                   6721.2, 4700.9, -16.5,
                    -47.1,  -16.5,   0.3), nrow=3, ncol=3)

mydata <- mvrnorm(500, mean, sigma)
mydata <- as.data.frame(mydata)
names(mydata) <- c("y", "x1", "x2")

dim(mydata)
```

```
## [1] 500   3
```

```
head(mydata, n=10)
```

```
##        y    x1   x2
## 1   98.8  41.3 3.43
## 2  244.5 205.2 3.80
## 3  375.7 186.7 2.51
## 4  -59.2  11.2 4.71
## 5  313.0 111.0 3.45
## 6  288.8 185.1 2.72
## 7  134.8 165.0 4.39
## 8  171.7  97.4 3.64
## 9  167.2 101.0 3.50
## 10 121.1  94.5 4.10
```

```
# Listing 5.4 - Applying functions to data objects

a <- 5
sqrt(a)
```

```
## [1] 2.24
```

```
b <- c(1.243, 5.654, 2.99)
round(b)
```

```
## [1] 1 6 3
```

```
c <- matrix(runif(12), nrow=3)
c
```

```
##        [,1]  [,2]  [,3]  [,4]
## [1,] 0.9636 0.216 0.289 0.913
## [2,] 0.2068 0.240 0.804 0.353
## [3,] 0.0862 0.197 0.378 0.931
```

```
log(c)
```

```
##         [,1]  [,2]   [,3]    [,4]
## [1,] -0.0371 -1.53 -1.241 -0.0912
```

```
## [2,] -1.5762 -1.43 -0.218 -1.0402
## [3,] -2.4511 -1.62 -0.972 -0.0710
```

```
mean(c)
```

```
## [1] 0.465
```

```
# Listing 5.5 - Applying a function to the rows
# (columns) of a matrix

mydata <- matrix(rnorm(30), nrow=6)
mydata
```

```
##        [,1]   [,2]   [,3]   [,4]   [,5]
## [1,]  0.459  1.203  1.234  0.591 -0.281
## [2,] -1.261  0.769 -1.891 -0.435  0.812
## [3,] -0.527  0.238 -0.223 -0.251 -0.208
## [4,] -0.557 -1.415  0.768 -0.926  1.451
## [5,] -0.374  2.934  0.388  1.087  0.841
## [6,] -0.604  0.935  0.609 -1.944 -0.866
```

```
apply(mydata, 1, mean)
```

```
## [1]  0.641 -0.401 -0.194 -0.136  0.975 -0.374
```

```
apply(mydata, 2, mean)
```

```
## [1] -0.478  0.777  0.148 -0.313  0.292
```

```
apply(mydata, 2, mean, trim=.4)
```

```
## [1] -0.542  0.852  0.499 -0.343  0.302
```

一个实例

一组学生参加了数学、科学和英语考试。为了给所有学生确定一个单一的成绩衡量指标，需要将这些科目的成绩组合起来。另外，你还想将前 20% 的学生评定为 A，接下来 20% 的学生评定为 B，依次类推。最后，你希望按字母顺序对学生排序。观察此数据集，马上可以发现一些明显的障碍。首先，三科考试的成绩是无法比较的。由于它们的均值和标准差相去甚远，所以对它们求平均值是没有意义的。你在组合这些考试成绩之前，必须将其变换为可比较的单元。其次，为了评定等级，你需要一种方法来确定某个学生在前述得分上百分比排名。再次，表示姓名的字段只有一个，这让排序任务复杂化了。为了正确地将其排序，需要将姓和名拆开。

准备数据

```
options(digits=2)

Student <- c("John Davis", "Angela Williams",
    "Bullwinkle Moose", "David Jones",
    "Janice Markhammer", "Cheryl Cushing",
    "Reuven Ytzrhak", "Greg Knox", "Joel England",
    "Mary Rayburn")
Math <- c(502, 600, 412, 358, 495, 512, 410, 625, 573, 522)
Science <- c(95, 99, 80, 82, 75, 85, 80, 95, 89, 86)
English <- c(25, 22, 18, 15, 20, 28, 15, 30, 27, 18)
roster <- data.frame(Student, Math, Science, English,
    stringsAsFactors=FALSE)
#knitr::kable(roster)
```

表格展示数据

Student	Math	Science	English
John Davis	502	95	25
Angela Williams	600	99	22
Bullwinkle Moose	412	80	18
David Jones	358	82	15
Janice Markhammer	495	75	20
Cheryl Cushing	512	85	28
Reuven Ytzrhak	410	80	15
Greg Knox	625	95	30
Joel England	573	89	27
Mary Rayburn	522	86	18

数据标准化

由于数学、科学和英语考试的分值不同（均值和标准差相去甚远），在组合之前需要先让它们变得可以比较。一种方法是将变量进行标准化，这样每科考试的成绩就都是用单位标准差来表示，而不是以原始的尺度来表示了。这个过程可以使用 scale() 函数来实现：

```
z <- scale(roster[,2:4])
z
```

```
##          Math Science English
##  [1,]  0.013   1.078   0.587
##  [2,]  1.143   1.591   0.037
##  [3,] -1.026  -0.847  -0.697
##  [4,] -1.649  -0.590  -1.247
##  [5,] -0.068  -1.489  -0.330
##  [6,]  0.128  -0.205   1.137
##  [7,] -1.049  -0.847  -1.247
##  [8,]  1.432   1.078   1.504
##  [9,]  0.832   0.308   0.954
## [10,]  0.243  -0.077  -0.697
## attr(,"scaled:center")
##    Math Science English
##     501      87      22
## attr(,"scaled:scale")
##    Math Science English
##    86.7     7.8     5.5
```

```
score <- apply(z, 1, mean)
roster <- cbind(roster, score)
knitr::kable(roster)
```

Student	Math	Science	English	score
John Davis	502	95	25	0.56
Angela Williams	600	99	22	0.92
Bullwinkle Moose	412	80	18	-0.86
David Jones	358	82	15	-1.16
Janice Markhammer	495	75	20	-0.63
Cheryl Cushing	512	85	28	0.35
Reuven Ytzrhak	410	80	15	-1.05
Greg Knox	625	95	30	1.34

Student	Math	Science	English	score
Joel England	573	89	27	0.70
Mary Rayburn	522	86	18	-0.18

然后，可以通过函数 mean() 来计算各行的均值以获得综合得分，并使用函数 cbind() 将其添加到花名册中：

划分等级

函数 quantile() 给出了学生综合得分的百分位数。可以看到，成绩为 A 的分界点为 0.74，B 的分界点为 0.44，等等。

```
y <- quantile(score, c(.8,.6,.4,.2))
y
```

```
##   80%   60%   40%   20%
##  0.74  0.44 -0.36 -0.89
```

通过使用逻辑运算符，你可以将学生的百分位数排名重编码为一个新的类别型成绩变量。下面在数据框 roster 中创建了变量 grade：

```
roster$grade[score >= y[1]] <- "A"
roster$grade[score < y[1] & score >= y[2]] <- "B"
roster$grade[score < y[2] & score >= y[3]] <- "C"
roster$grade[score < y[3] & score >= y[4]] <- "D"
roster$grade[score < y[4]] <- "F"
knitr::kable(roster)
```

Student	Math	Science	English	score	grade
John Davis	502	95	25	0.56	B
Angela Williams	600	99	22	0.92	A
Bullwinkle Moose	412	80	18	-0.86	D
David Jones	358	82	15	-1.16	F
Janice Markhammer	495	75	20	-0.63	D
Cheryl Cushing	512	85	28	0.35	C
Reuven Ytzrhak	410	80	15	-1.05	F
Greg Knox	625	95	30	1.34	A
Joel England	573	89	27	0.70	B
Mary Rayburn	522	86	18	-0.18	C

按姓和名排序

使用函数 strsplit() 以空格为界把学生姓名拆分为姓氏和名字。把 strsplit() 应用到一个字符串组成的向量上会返回一个列表：

```
name <- strsplit((roster$Student), " ")
name
```

```
## [[1]]
## [1] "John"  "Davis"
##
## [[2]]
## [1] "Angela"   "Williams"
##
## [[3]]
## [1] "Bullwinkle" "Moose"
##
```

```
## [[4]]
## [1] "David" "Jones"
##
## [[5]]
## [1] "Janice"     "Markhammer"
##
## [[6]]
## [1] "Cheryl"  "Cushing"
##
## [[7]]
## [1] "Reuven"  "Ytzrhak"
##
## [[8]]
## [1] "Greg" "Knox"
##
## [[9]]
## [1] "Joel"    "England"
##
## [[10]]
## [1] "Mary"    "Rayburn"
```

使用函数 sapply() 提取列表中每个成分的第一个元素，放入一个储存名字的向量 Firstname ，并提取每个成分的第二个元素，放入一个储存姓氏的向量 Lastname 。"[" 是一个可以提取某个对象的一部分的函数——在这里它是用来提取列表 name 各成分中的第一个或第二个元素的。你将使用 cbind() 把它们添加到花名册中。由于已经不再需要 student 变量，可以将其丢弃:

```
lastname <- sapply(name, "[", 2)
firstname <- sapply(name, "[", 1)
roster <- cbind(firstname,lastname, roster[,-1])
```

最后，可以使用函数 order() 依姓氏和名字对数据集进行排序：

```
roster <- roster[order(lastname,firstname),]
knitr::kable(roster)
```

	firstname	lastname	Math	Science	English	score	grade
6	Cheryl	Cushing	512	85	28	0.35	C
1	John	Davis	502	95	25	0.56	B
9	Joel	England	573	89	27	0.70	B
4	David	Jones	358	82	15	-1.16	F
8	Greg	Knox	625	95	30	1.34	A
5	Janice	Markhammer	495	75	20	-0.63	D
3	Bullwinkle	Moose	412	80	18	-0.86	D
10	Mary	Rayburn	522	86	18	-0.18	C
2	Angela	Williams	600	99	22	0.92	A
7	Reuven	Ytzrhak	410	80	15	-1.05	F

控制流

条件语句和循环语句

循环

1.for 循环

```
for(i in 1:5)
  print("Hello")
```

```
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
```

2.while 循环

```
i <- 5
while(i > 0){
  print("Hello")
  i <- i - 1
}
```

```
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
## [1] "Hello"
```

条件

1.if-else 结构

```
(x <- runif(1))
```

```
## [1] 0.48
```

```
if(x > 0.5){
  print("yes")
}else{
  print("no")
}
```

```
## [1] "no"
```

2.ifelse 结构

```
(x <- runif(1))
```

```
## [1] 0.11
```

```
(result <- ifelse(x > 0.5, "yes", "no"))
```

```
## [1] "no"
```

3.switch 结构

```
level <- c("poor","good","excelent")
(i <- round(runif(1,1,3)))
```

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

```
(result <- switch(level[i],
                  poor = "you are poor",
                  good = "yes,good",
                  excelent = "very good"))
```

```
## [1] "yes,good"
```

用户自编函数

R 的最大优点之一就是用户可以自行添加函数。事实上，R 中的许多函数都是由已有函数构成的。函数中的对象只在函数内部使用。返回对象的数据类型是任意的，从标量到列表皆可。让我们看一个示例。假设你想编写一个函数，用来计算数据对象的集中趋势和散布情况。此函数应当可以选择性地给出参数统计量（均值和标准差）和非参数统计量（中位数和绝对中位差）。结果应当以一个含名称列表的形式给出。另外，用户应当可以选择是否自动输出结果。除非另外指定，否则此函数的默认行为应当是计算参数统计量并且不输出结果。

```
mystats <- function(x, parametric=TRUE, print=FALSE) {
  if (parametric) {
    center <- mean(x); spread <- sd(x)
  } else {
    center <- median(x); spread <- mad(x)
  }
  if (print & parametric) {
    cat("Mean=", center, "\n", "SD=", spread, "\n")
  } else if (print & !parametric) {
    cat("Median=", center, "\n", "MAD=", spread, "\n")
  }
  result <- list(center=center, spread=spread)
  return(result)
}
# trying it out
set.seed(1234)
x <- rnorm(500)
y <- mystats(x)
y <- mystats(x, parametric=FALSE, print=TRUE)
```

```
## Median= -0.021
##  MAD= 1
```