

大数据挖掘软件系列课之： R语言

讲义

肖韬




2019 Tao Xiao

Sep 2019

Contents

1	课程信息和规章、R语言介绍	5
1.1	课程信息和规章	5
1.2	R语言简介	6
1.3	下载和安装R	7
1.4	下载和安装RStudio	8
1.5	快速上手R	10
1.6	随堂练习	11
2	向量和矩阵	13
2.1	向量 (vector)	13
2.1.1	向量简介	13
2.1.2	赋值	14
2.1.3	合并向量	15
2.1.4	字符型向量	15
2.1.5	向量的粘连	15
2.1.6	逻辑型向量	16
2.1.7	带名字的向量	16
2.2	矩阵 (matrix)	16
2.3	数组 (array)	17
3	数据分拆	19
3.1	分拆数据 (subsetting) 的技巧	19
3.1.1	分拆向量	19
3.1.2	分拆矩阵	20

3.2	分拆数据的规则总结	21
3.3	R Markdown的帮助文件	21
4	列表和数据框	23
4.1	列表 (list)	23
4.2	数据框 (data frames)	23
4.2.1	数据框和数组(arrays)的区别	24
4.2.2	如何创建一个数据框	24
4.2.3	数据框 (data frame) 是什么?	24
4.2.4	查询数据框的列名和行名	25
4.2.5	查询数据框的行数和列数	26
4.2.6	用\$操作符来分拆列表和数据框	26
4.2.7	从硬盘上读入另外一个数据表Forbes2000	26
5	因子、R的工作空间、导入导出数据	29
5.1	因子	29
5.1.1	因子的介绍	29
5.1.2	如何创建因子	29
5.1.3	如何更改因子的级别	30
5.2	R的工作空间 (workspace) 和工作路径 (work directory)	31
5.2.1	工作空间 (workspace)	31
5.2.2	工作路径 (work directory)	31
5.2.3	.RData文件类型	31
5.2.4	载入之前保存的工作空间文件	32
5.3	从硬盘导入数据、导出数据到硬盘	32
5.3.1	从硬盘导入数据: read.table()和read.csv()函数	32
5.3.2	导出数据到硬盘: write.table()和write.csv()函数	32
6	常见运算符及其向量型计算	33
6.1	用常见的运算符和操作符进行向量型计算	33
6.2	向量循环和重复	34
6.3	在数据框中进行向量计算	35
6.3.1	例子: 均值中心化	35
6.3.2	例子: 计算加速度	36
7	逻辑操作符的运用, 非正常值	39
7.1	逻辑操作符的向量化运用	39
7.2	用逻辑型向量来提取数据	40
7.3	三种非正常值: 缺失值, 不定值, 无穷值	40
7.3.1	缺失值: NA (not available)	40
7.3.2	不定值: NaN (not a number)	41
7.3.3	无穷值: Inf 和 -Inf	41



1. 课程信息和规章、R语言介绍

1.1 课程信息和规章

- **上课时间地点**：星期一，9-10节，文科楼H6-104。
- **随堂实验**：基本每堂课都会留出时间，让同学们做一个随堂实验，巩固消化本堂课所学内容，随堂实验期间老师会解答同学们在实验过程中碰到的问题。随堂实验的解答请在该次课下课前在Blackboard中提交，提交截止时间是每次课后的当天晚上9点整，提交了即给满分，提交截止时间后不能再提交。答案会在提交截止时间之后公布。
- **课后作业**：课后作业会放在Blackboard中，为随机选择题的形式。课后作业在截止日期后，不能再在Blackboard中提交。答案会在提交截止日期之后公布，答案公布之后一律不再接收迟交的作业。在答案公布前迟交的作业只能获得所得分值的70%。
- **期末考试**：闭卷上机考，在答卷上作答。
- **成绩评定方式**：随堂实验：占15%；课后作业：占15%；期末考试占70%。
- **接访时间(office hours)**：星期一上午,9:30AM-11:30AM,科技楼313。
- **老师联系方式**：taoxiao@szu.edu.cn
- **参考资料**：老师的课件材料和所发的其他材料。

1.2 R语言简介

首先让我们来看看什么是R。广义上来说，R是一门计算机编程语言。人们可以使用这门语言编写算法或者重复使用前人用这门语言编写的算法。这一点上，R和其他任何计算机编程语言没什么区别。我们先来看看R到底可以做什么(实际上，我们可以用R做你可以想像的任何事情)。你可以用R写函数，进行计算，你可以使用R来调用绝大多数已存在的数据分析方法来探索数据，包括对数据进行各种分析和挖掘和绘制各种简单或者复杂的图形来进行数据可视化。R可以帮助我们做这么多事情，而且它是可以免费使用的！在CRAN (<http://cran.r-project.org/packages>) 上，还有超过10000个人们分享的开源的R扩展包(package)供我们免费安装使用。你甚至可以自己创建一个R扩展包，把它分享给R社区的其他用户使用。人们常说，没有什么工作比创造和分享更让人身心愉悦了。R就是如此让人“免费”的身心愉悦。越来越多的研究机构、公司和大学正在将R这种强大而免费的编程工具来处理和分析数据。

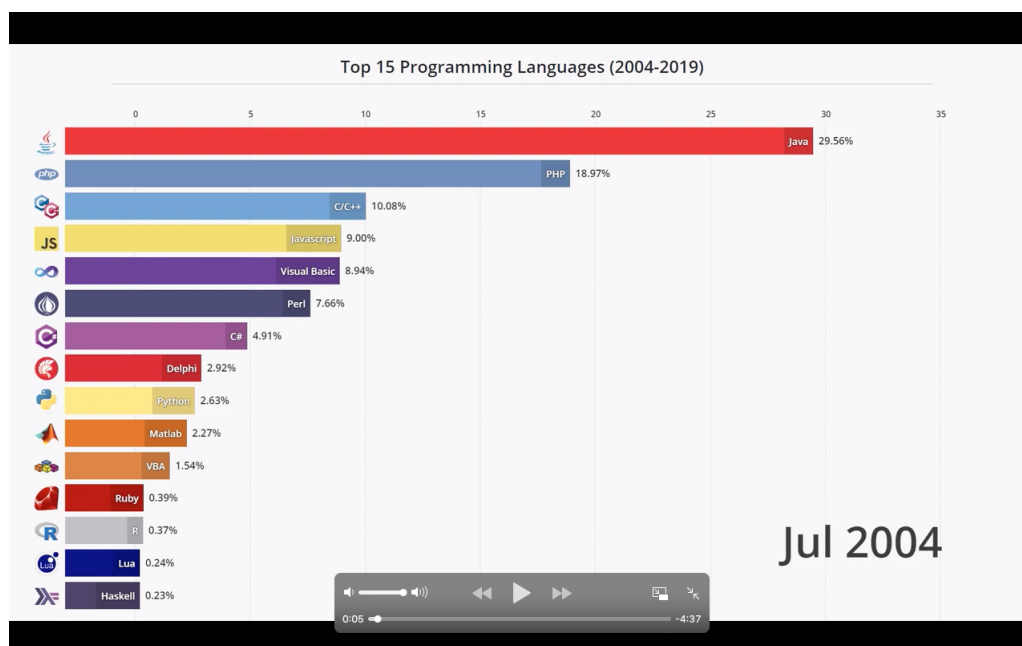


Figure 1.1: 2004年7月R的使用占比

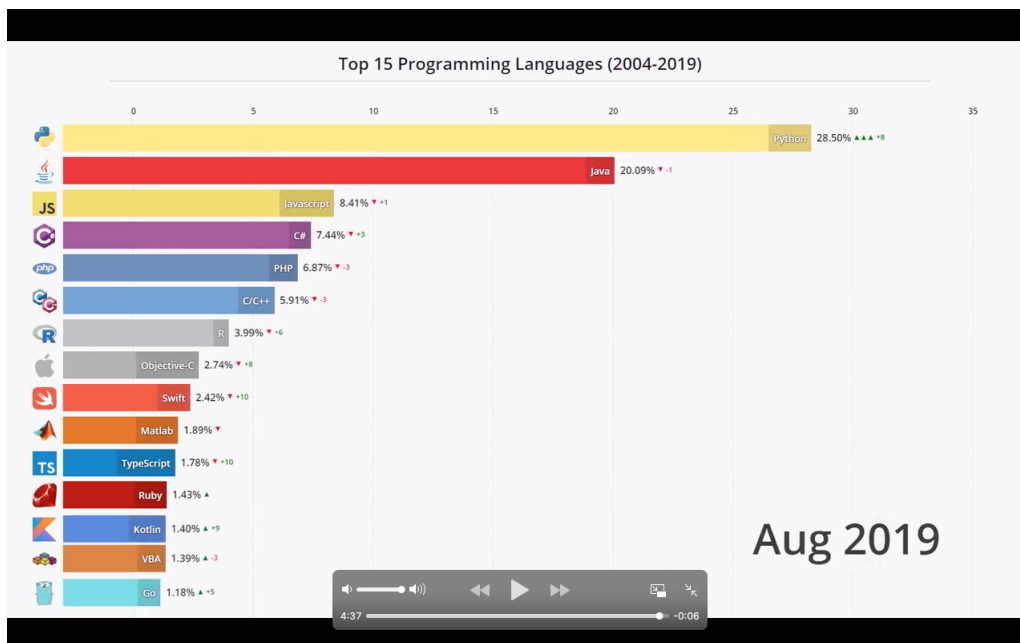


Figure 1.2: 2019年8月R的使用占比

图片来源网址

1.3 下载和安装R

现在我们谈谈如何得到R。如果您已经安装了R在您的电脑上，则您可以跳过这一章。首先，请在浏览器中打开以下这个网站: <https://www.r-project.org/>. 在左侧的“Download”目录下的“CRAN” (是Comprehensive R Archive Network, R综合存档网络, 的缩写)链接中, 选择一个下载镜像网站, 比如我们可以选择“China”目录下的

<https://mirrors.tuna.tsinghua.edu.cn/CRAN/>

即进入下载页面, 在下载页面中, 根据自己电脑的操作系统选择要下载的版本; 可供选择的版本有Linux, MacOS X和Windows。下面我们以Windows版本为例进行说明。点击“Download R for Windows”。我们暂时只装最基本的R软件, 所以我们点选“base”链接, 然后点击里面的软件下载链接, 将软件下载下来之后按默认设置进行安装即可。R是一个快速发展的软件, 经常会有更新版本的R软件在以上下载页推出, 很多人们分享的R扩展包也会要求使用更新版本的R软件。一个R语言的用户经常需要在电脑中下载更新版本的R软件使用。在一台电脑中是可以安装多种版本的R软件的, 这些不同版本的R软件默认会被安装在同一个目录下的不同子目录下, 不会互相影响。

以Windows操作系统为例。如果以默认设置方式安装了R, 会在桌面上有一个启动R软件的快捷键, 这是一个蓝色的大写的“R”的图标。双击这个图标, 即可打开R软件。或者, 在“开始”按钮下的程序列表中打开R软件。打开R软件后, 出现如图1.5所示的R命令行窗口。

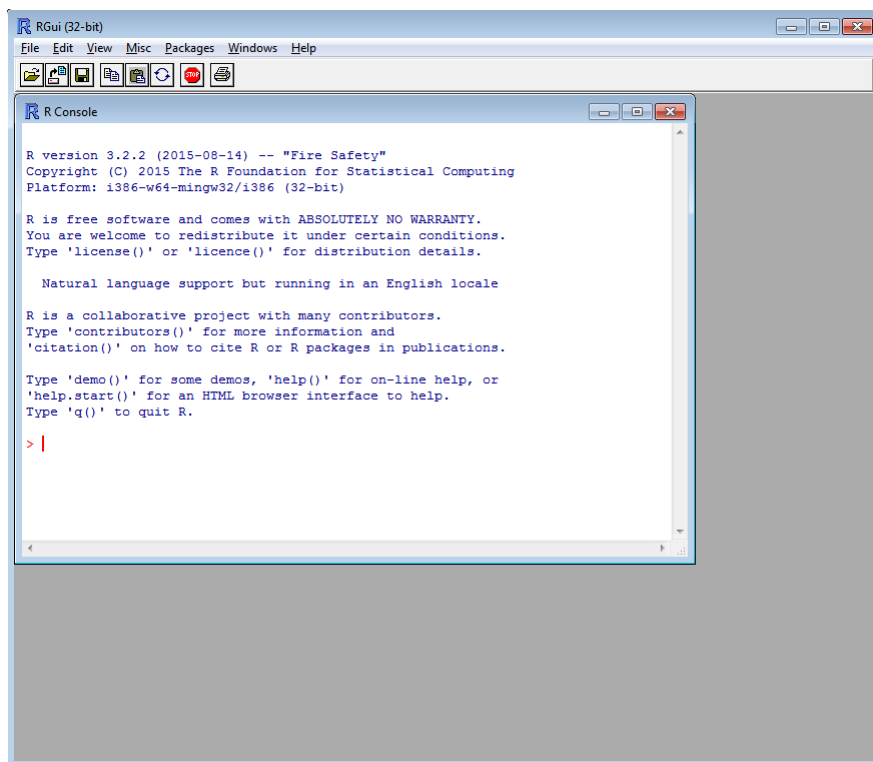


Figure 1.3: R软件开始界面；也叫R的命令行窗口。

1.4 下载和安装RStudio

RStudio是R语言的一个IDE，它帮助用户高效的完成数据分析、图像绘制、生成报表、开发R的packages等工作。在安装好最新版本的R后，可以安装RStudio：

1. 打开RStudio官网：
<https://www.rstudio.com>
2. 找到最左边RStudio下面的Download,即如下网址：
<https://www.rstudio.com/products/rstudio/download/>
3. 在第一列的FREE中选择DOWNLOAD，会直接显示需要下载的版本
4. 选择自己的计算机操作系统即可下载
5. 安装方式为普通软件安装方式, 安装过程中使用默认安装方式。

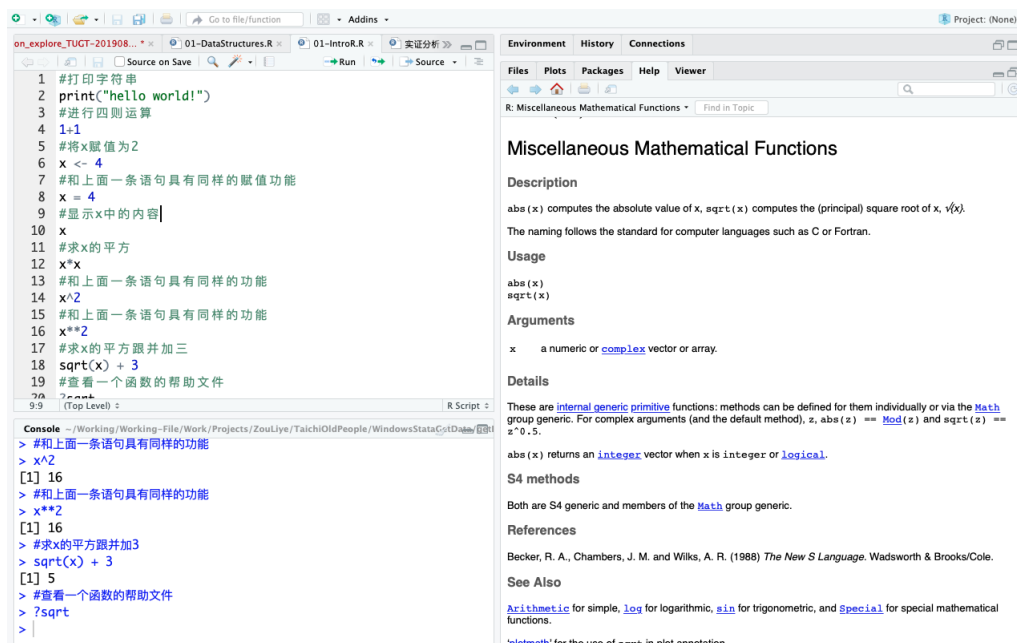


Figure 1.4: RStudio界面

一个题外话：2019年7月31日，在国际统计学年会（JSM）上，统计学会会长委员会（Committee of Presidents of Statistical Societies，简称COPSS）将当年的考普斯会长奖（COPSS Presidents' Award）颁发给了RStudio公司的首席科学家Hadley Wickham，以表彰他在统计应用领域做出的卓越贡献。这个奖项是统计学领域的最高奖项，被誉为是统计届的诺贝尔奖。



Figure 1.5: 2019年考普斯会长奖获得者：RStudio公司的首席科学家Hadley Wickham

Hadley创建并发布了很流行的R包，近一年（2018.8.1-2019.8.1）下载量超过1.3亿次，近两年下载量超过2亿次。下图列出了近一年来下载量超过250万次的他开发的24个包的6年走势。

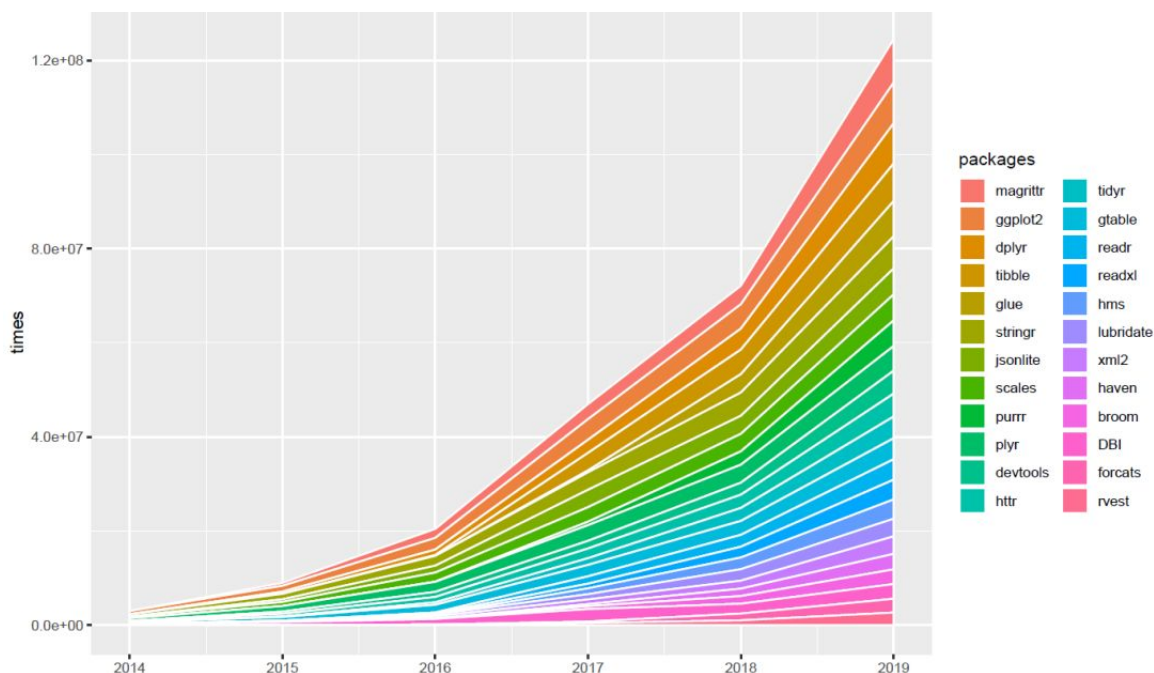


Figure 1.6: 近一年来下载量超过 250 万次的Hadley Wickham开发的24 个包的 6 年走势（图片来源网址）

1.5 快速上手R

如果R检测到一个命令没有写完（比如没有匹配的括号或者引号），那么R会在控制台（Console）显示：+。如果R语言的一行中出现“#”，那么这一行中余下的部分都不会被执行；我们利用这个特性来对我们的代码进行注释。我们现在快速展示一下在R或者RStudio软件做一些有用的事情，让同学们能够快速上手使用R。以下所有的命令可以在控制台（Console）窗口中的>符号后依次敲入然后按回车键，然后控制台会紧接着显示出结果。也可以把这些命令复制到R脚本文件中执行。

打印字符串

```
#
print("hello world!") 进行四则运算
#
1+1 将
#赋值为x2
x <- 4和上面一条语句具有同样的赋值功能
#
x = 4 显示
#中的内容x
x求
#的平方x
x*x和上面一条语句具有同样的功能
#
x^2和上面一条语句具有同样的功能
#
x**2 求
#的平方跟并加三x
sqrt(x) + 3 查看一个函数的帮助文件
#
?sqr和上面一条语句具有同样的功能
#
help(sqrt) 查看线性回归模型函数的帮助文件
```

```
#  
?lm和上面一条语句具有同样的功能  
#  
help(lm)
```

1.6 随堂练习

1. 打开实验室电脑上的RStudio (最好也打开R软件看一下), 熟悉里面的各个模块的功能。
2. 到Blackboard课程网站上的“R Codes”目录中下载1.5小节里的程序“01-IntroR.R”, 在RStudio中或者R软件中打开这个程序, 逐行运行其中的命令, 熟悉在脚本文件中以及在控制台中运行R命令的方法。



2. 向量和矩阵

2.1 向量 (vector)

2.1.1 向量简介

向量 (vector) 是R语言中最基本的数据存储单位。我们这门课要学以下几种类型的向量：

- 逻辑型 (logical)
- 整数型 (integer)
- 数字型 (numeric, 可带小数)
- 字符型 (character)

可以使用以下命令在R中调出关于向量的帮助文件参考：

```
?vector
```

接下来我们来看看使用冒号“:”来创建向量的例子。如果我们在命令行中输入：

```
1:10
```

结果为：

```
[1] 1 2 3 4 5 6 7 8 9 10
```

如果输入：

```
(1:10)^2
```

则结果为：

```
[1] 1 4 9 16 25 36 49 64 81 100
```

另一个例子：

```
sum(1:100)
```

```
[1] 5050
```

又一个例子：

```
letters
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p"
     "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

又一个例子:

```
month.name
```

```
[1] "January" "February" "March" "April" "May"
     "June"
[7] "July" "August" "September" "October" "November"
     "December"
```

我们也可以使用“length()”函数看看month.name这个向量的长度:

```
length(month.name)
```

```
[1] 12
```

向量对象构成了以下将要学到的复杂对象的基础:

- 矩阵 (matrix)
- 数组 (array)
- 列表 (list)
- 数据框 (data frame)

2.1.2 赋值

现在来看看如何给R中的对象赋值。我们使用“<-”符号来给对象赋值。比如以下语句把向量“1:10”赋值给了对象“x”:

```
x <- 1:10
```

然后，直接在命令行中输入“x”并回车，我们可以察看x的值:

```
x
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

接下来，我们可以对“x”使用各种用于向量型对象的函数。比如我们就可以用“sum()”函数来对“x”的各元素进行求和:

```
sum(x)
```

```
[1] 55
```

也可以:

```
sum(x^2)
```

```
[1] 385
```

我们也可以使用“class()”函数来检验“x”的类型:

```
class(x)
```

```
[1] "integer"
```

可看出，“x”是一个整数型的向量。

2.1.3 合并向量

现在来看看如何合并向量。我们可以用“c()”函数(代表combine, 合并, 的意思)来合并向量。比如我们下面把6个整数合并为一个向量:

```
x <- c(1, 1, 2, 3, 5, 8)
```

我们来看一下x的值:

```
x  
[1] 1 1 2 3 5 8
```

同样, 我们也可以用“sum()”函数来对“x”的各元素进行求和:

```
sum(x)  
[1] 20
```

我们也可以用“c()”函数将对象“x”和其他的数字合并:

```
y <- c(x, 13, 21)  
y  
[1] 1 1 2 3 5 8 13 21
```

2.1.4 字符型向量

可以使用类似的方法创建字符型向量:

```
authors <- c("Ross", "Robert")  
authors  
[1] "Ross" "Robert"
```

使用“length()”函数检验字符型向量的长度:

```
length(authors)  
[1] 2
```

2.1.5 向量的粘连

我们可以使用“paste()”函数来进行向量的字符串粘连。我们使用例子来说明:

```
paste("row", 1:5, sep = "_")  
[1] "row_1" "row_2" "row_3" "row_4" "row_5"
```

如果省略“sep”选项, 则会用缺省的sep = " "来间隔粘连的字符串。

```
paste("col", 1:5)  
[1] "col 1" "col 2" "col 3" "col 4" "col 5"
```

我们也可使用“paste0()”函数来粘连, 但是“paste0()”函数在粘连时缺省不用任何间隔:

```
paste0("col", 1:5)  
[1] "col1" "col2" "col3" "col4" "col5"
```

如果粘连的两个对象都为向量, 在两个向量长度一致的情况下, “paste()”函数会按对象的顺序粘连;以我们以上已经创建的字符型向量“authors”为例:

```
paste(authors, c("Ihaka", "Gentleman"))  
[1] "Ross Ihaka" "Robert Gentleman"
```


2.1.6 逻辑型向量

逻辑型向量的元素只有两种值: TRUE或者FALSE。以下用已经创建的长度为2的字符型向量“authors”和“==”号返回一个长度为2的逻辑型向量:

```
authors == "Ross"
```

```
[1] TRUE FALSE
```

另一个例子: 用长度为10的整数型向量和“>”号返回一个长度为10的逻辑型向量:

```
1:10 > 7
```

```
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE
```

逻辑型向量的两种值可以被当作数字“1”和“0”使用:

```
sum(1:10 > 7)
```

```
[1] 3
```

2.1.7 带名字的向量

向量中的每个元素都可以有一个对应的名字:

```
x <- 1:5
names(x) <- letters[1:5]
x
```

```
a b c d e
1 2 3 4 5
```

```
x["c"]
```

```
c
3
```

```
x[c("c", "d")]
```

```
c d
3 4
```

2.2 矩阵 (matrix)

在R中, 我们用 `matrix()` 函数来创建矩阵:

```
matrix(1:9, nrow = 3, ncol = 3)
```

```
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```
mat <- matrix(1:25, nrow = 5)
mat
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
```

2.3 数组 (array)

数组在R中也是一个对象。之前学的矩阵(matrix)是一个二维的数组(array)。而数组这个对象的维数可以有超过2。我们使用array()函数来创建数组。以下创建一个二维的数组(注意这个array()函数所使用的参数与matrix()函数的区别)：

```
array.one <- array(c(1:12), dim = c(3, 4))
array.one
```

```
      [,1] [,2] [,3] [,4]
[1,]     1     4     7    10
[2,]     2     5     8    11
[3,]     3     6     9    12
```

以下创建一个三维的数组：


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

```
, , 1
```

```
      [,1] [,2] [,3] [,4]
[1,]     1     4     7    10
[2,]     2     5     8    11
[3,]     3     6     9    12
```

```
, , 2
```

```
      [,1] [,2] [,3] [,4]
[1,]    13    16    19    22
[2,]    14    17    20    23
[3,]    15    18    21    24
```

3. 数据分拆

3.1 分拆数据 (subsetting) 的技巧

我们用[]符号来分拆向量和矩阵。

3.1.1 分拆向量

我们继续拿letters向量来做例子:

```
letters
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p"  
[17] "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
letters[5]
```

```
[1] "e"
```

```
letters[1:5]
```

```
[1] "a" "b" "c" "d" "e"
```

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

```
[1] "a" "c" "e"
```

```
letters[-(1:10)]
```

```
[1] "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
head(letters, 5)
```

```
[1] "a" "b" "c" "d" "e"
```

```
tail(letters, 5)
```

```
[1] "v" "w" "x" "y" "z"
```

3.1.2 分拆矩阵

我们用之前创建的mat矩阵来作例子:

```
mat
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	6	11	16	21
[2,]	2	7	12	17	22
[3,]	3	8	13	18	23
[4,]	4	9	14	19	24
[5,]	5	10	15	20	25

我们可以使用负数来删除矩阵中对应脚标中的元素。注意，这里的负号不是表示“倒数”，而是表示“删除”。以下例子删除了mat矩阵的第3行和第4列:

```
mat[-3, -4]
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	6	11	21
[2,]	2	7	12	22
[3,]	4	9	14	24
[4,]	5	10	15	25

我们也可以很方便的利用向量来分拆矩阵。以下例子提取mat矩阵中的第2至4行和第3至5列:

```
mat[2:4, 3:5]
```

	[,1]	[,2]	[,3]
[1,]	12	17	22
[2,]	13	18	23
[3,]	14	19	24

对矩阵用rownames()分配行名,用colnames()分配列名 (回忆: 对向量我们用names()函数给向量的元素起名字)。再以已经创建好的mat矩阵为例:

```
mat
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	6	11	16	21
[2,]	2	7	12	17	22
[3,]	3	8	13	18	23
[4,]	4	9	14	19	24
[5,]	5	10	15	20	25

分配行名和列名:

```
colnames(mat) <- paste("col", 1:5, sep = "")
rownames(mat) <- paste("row", 1:5, sep = "")
mat
```

	col1	col2	col3	col4	col5
row1	1	6	11	16	21
row2	2	7	12	17	22
row3	3	8	13	18	23
row4	4	9	14	19	24
row5	5	10	15	20	25

行和列都命了名的矩阵也和命了名的向量类似，我们可以用行名和列名来分拆数据。我们看看一个R中内置的例子数据表mtcars:

```
head(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

用矩阵mtcars的行名和列名来分拆这个矩阵:

```
mtcars[c("Hornet 4 Drive", "Datsun 710"), c("mpg", "cyl")]
```

	mpg	cyl
Hornet 4 Drive	21.4	6
Datsun 710	22.8	4

3.2 分拆数据的规则总结

分拆数据的规则总结如下表所示:

语法	规则	效果
[]	空	返回整个向量
[1:3]	正整数	返回对应位置的元素
["a"]	字符	返回命名了的元素
[-1]	负整数	删除对应位置的元素
[NULL]	NULL	一个空向量

3.3 R Markdown的帮助文件

对R Markdown的学习感兴趣的同学可以到RStudio的以下菜单中找到更多信息:

- Help -> Cheatsheets -> R Markdown Cheet Sheet
- Help -> Markdown Quick Reference

4. 列表和数据框

4.1 列表 (list)

列表(lists)是R语言中非常重要的类，这个类构成了数据框的基础。我们可以用list()列表函数将不同类的向量组合到一起。如下例：

```
x1 <- list(num = 1:10, char = letters[1:15], log = rep(c(TRUE, FALSE), 10))
x1
```

```
$num
[1] 1 2 3 4 5 6 7 8 9 10

$char
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"

$log
[1] TRUE FALSE TRUE FALSE TRUE FALSE TRUE FALSE TRUE FALSE
     TRUE FALSE
[13] TRUE FALSE TRUE FALSE TRUE FALSE TRUE FALSE
```

我们用[]来分拆一个列表：

```
x1[1:2]

$num
[1] 1 2 3 4 5 6 7 8 9 10

$char
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"
```

4.2 数据框 (data frames)

数据框是一种被R中大多数建模函数使用到的基本数据存储方式。

- 从技术上说，数据框是一种特殊形式的列表，只不过这个列表中的每一个元素有相同的长度。
- 数据框的形状是矩形的，有行也有列。
- 我们可以把数据框想成和一个数据库中的一个数据表类似的东西，或者和一个电子制表软件（比如Excel）里的工作表(worksheet)类似的东西。

R软件中内置的数据表mtcars就是一个数据框：

```
head(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

```
class(mtcars)
```

```
[1] "data.frame"
```

4.2.1 数据框和数组(arrays)的区别

- 数组强制其中的数据元素为同一类型类型。
 - 比如数组中所有的元素都为数字型，或者都为字符型。
- 数据框的不同的列可以是不同的
 - 记录列表数据的时候很方便。
 - 数据框和数据库中的数据表非常类似：可以记录多种类型的数据。

数据框的属性：

- 是一种有相同的长度元素的列表。
- 和列表一样，每一个元素可以来自不同的类
- 行和列都可以有名字。

4.2.2 如何创建一个数据框

我们可以用data.frame()函数轻易的将不同类型的向量组合到一起。请看下例：

```
dat <- data.frame(char = letters[1:5], num = 1:5, log = c(TRUE,
  FALSE, TRUE, FALSE, TRUE))
head(dat)
```

```
  char num  log
1    a   1 TRUE
2    b   2 FALSE
3    c   3 TRUE
4    d   4 FALSE
5    e   5 TRUE
```

用以下语句察看关于数据框（data frame）的帮助文件：

```
help(data.frame)
```

4.2.3 数据框（data frame）是什么？

我们先来介绍R中的两个以R对象为参数的函数。第一个是class()函数；这个函数返回对象的类名，从这个类名可以判断这个R对象继承了哪些属性。第二个函数是mode()函数，这个函数返回一个对象的内部存储类型。我们现在分别把这两个函数用于一个数据框对象，也是R中内置的一个数据表mtcars。

```
class(mtcars)
```

```
[1] "data.frame"
```

可看出，mtcars对象是属于一个类，这个类的类名就是“data.frame”。

```
mode(mtcars)
```

```
[1] "list"
```

可看出，mtcars对象的内部存储类型其实是一个“list”。

4.2.4 查询数据框的列名和行名

我们可以用结构查询函数str()来查看mtcars数据框的结构:

```
str(mtcars)
```

```
'data.frame': 32 obs. of 11 variables:
 $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num 6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num 160 160 108 258 360 ...
 $ hp : num 110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt : num 2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num 16.5 17 18.6 19.4 17 ...
 $ vs : num 0 0 1 1 0 1 0 1 1 1 ...
 $ am : num 1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num 4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num 4 4 1 1 2 1 4 2 2 4 ...
```

我们也可以用names()函数查看mtcars数据框“元素”的名字（数据框的“元素”是“列”而不是“行”），即查看列名：

```
names(mtcars)
```

```
[1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "
gear"
[11] "carb"
```

以下使用colnames()函数得到同样的结果

```
colnames(mtcars)
```

```
[1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "
gear"
[11] "carb"
```

使用rownames()函数查看数据框的行名。

```
rownames(mtcars)
```

```
[1] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710"
[4] "Hornet 4 Drive" "Hornet Sportabout" "Valiant"
[7] "Duster 360" "Merc 240D" "Merc 230"
[10] "Merc 280" "Merc 280C" "Merc 450SE"
[13] "Merc 450SL" "Merc 450SLC" "Cadillac Fleetwood"
"
[16] "Lincoln Continental" "Chrysler Imperial" "Fiat 128"
[19] "Honda Civic" "Toyota Corolla" "Toyota Corona"
```

```
[22] "Dodge Challenger"      "AMC Javelin"          "Camaro Z28"
[25] "Pontiac Firebird"      "Fiat X1-9"            "Porsche 914-2"
[28] "Lotus Europa"          "Ford Pantera L"       "Ferrari Dino"
[31] "Maserati Bora"         "Volvo 142E"
```

4.2.5 查询数据框的行数和列数

我们用`nrow()`和`ncol()`函数返回数据框的行数和列数:

```
nrow(mtcars)
```

```
[1] 32
```

```
ncol(mtcars)
```

```
[1] 11
```

我们也可以用`dim()`函数可以同时依次返回行数和列数:

```
dim(mtcars)
```

```
[1] 32 11
```

4.2.6 用\$操作符来分拆列表和数据框

对于列表和数据框来说, 我们有另外一个用来分拆的选项, 这就是\$操作符。\$操作符可以用来根据列表中元素的名称指向列表中的这个元素。在以下例子中, 我们用\$操作符将内置数据表`mtcars`中的“mpg”向量元素分拆出来然后提取这个向量的第一个值:

```
mtcars$mpg[1]
```

```
[1] 21
```

4.2.7 从硬盘上读入另外一个数据表Forbes2000

双击打开Datasets文件夹中的Forbes2000.RData文件。也可以使用以下两种方式之一打开, 我们下堂课会详细讲R中的数据输入和输出操作。

```
load("Forbes2000.RData")
```

或者

```
Forbes2000 = read.csv("Forbes2000.csv", as.is=c("name"))
```

用`str()`函数查看Forbes2000数据框的结构

```
str(Forbes2000)
```

```
'data.frame': 2000 obs. of 8 variables:
 $ rank      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ name      : chr  "Citigroup" "General Electric" "American Intl
   Group" "ExxonMobil" ...
 $ country   : Factor w/ 61 levels "Africa","Australia",...: 60 60
   60 60 56 60 56 28 60 60 ...
 $ category  : Factor w/ 27 levels "Aerospace & defense",...: 2 6 16
   19 19 2 2 8 9 20 ...
 $ sales     : num  94.7 134.2 76.7 222.9 232.6 ...
 $ profits   : num  17.85 15.59 6.46 20.96 10.27 ...
 $ assets    : num  1264 627 648 167 178 ...
 $ marketvalue: num  255 329 195 277 174 ...
```

可以对Forbes2000数据框进行建模或者画图的操作，比如对其中的两个变量画一个散点图（我们将在以后专门的章节介绍R语言中的画图）

```
plot(log(marketvalue) ~ log(sales), data = Forbes2000, pch = ".")
```

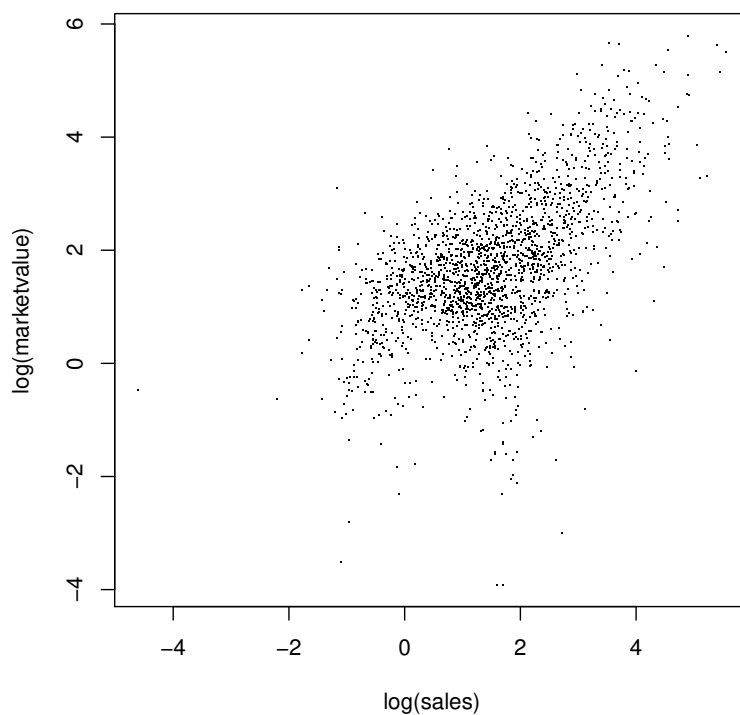


Figure 4.1: 利用Forbes2000数据框画的散点图

查看Forbes2000数据框中的几个典型的变量的类型：字符型变量：

```
class(Forbes2000$name)
```

```
[1] "character"
```

数字型变量

```
class(Forbes2000$sales)
```

```
[1] "numeric"
```

因子型(factor)变量

```
class(Forbes2000$country)
```

```
[1] "factor"
```

我们将在下一章介绍因子型变量

5. 因子、R的工作空间、导入导出数据

5.1 因子

5.1.1 因子的介绍

因子(factor)是R中的一种变量的类，其包含有限个数的不同值;这种变量通常被称为属性型变量或者离散型变量（比如“性别”、“国家”、“籍贯”等）。因子最重要的用途之一是用于统计建模;由于属性型变量与连续型变量在统计模型中经常是被区别对待的（比如线性回归模型会区别对待属性型自变量和连续型自变量），因此将属性型变量存储为因子可确保建模函数正确处理该类变量。R中的因子被存储的信息包括一个包含整数值的向量，以及在显示因子所包含的内容时要使用的相应字符值(即级别)的集合。factor()函数用于创建因子。factor()函数的唯一必需参数是一个向量，函数会将这个向量返回成因子。数字和字符型都可以作为因子，但因子的级别始终是字符值。您可以通过调用levels()函数来查看因子的所有可能级别;nlevels()函数将返回一个因子的级别总数。缺省的情况下，原向量中的值被用作显示因子里面的值的级别，但我们也可以在创建因子时通过在factor()函数中使用labels =参数来更改这些级别的名称。

5.1.2 如何创建因子

将data向量变成一个因子fdata

```
data = c(1,2,2,3,1,2,3,3,1,2,3,3,1)
fdata = factor(data)
```

查看因子fdata的内容：

```
fdata
```

```
[1] 1 2 2 3 1 2 3 3 1 2 3 3 1
Levels: 1 2 3
```

查看因子fdata的级别：

```
levels(fdata)
```

```
[1] "1" "2" "3"
```


5.1.3 如何更改因子的级别

以上我们从整数型向量data创建了一个级别缺省为整数值的因子fdata，以下我们从整数型向量data创建一个级别为罗马数字的因子rdata：

```
rdata = factor(data, labels=c("I", "II", "III"))
rdata
```

```
[1] I   II  II  III I   II  III III I
[10] II  III III I
Levels: I II III
```

```
levels(rdata)
```

```
[1] "I" "II" "III"
```

也可更改级别，比如以下我们在创建rdata对象时利用labels=参数中将以上罗马数字的级别倒序设置：

```
rdata = factor(data, labels=c("III", "II", "I"))
rdata
```

```
[1] III II  II  I   III II  I   I   III
[10] II  I   I   III
Levels: III II I
```

```
levels(rdata)
```

```
[1] "III" "II" "I"
```

我们可以使用进行数字型类型转换的函数as.numeric()函数查看因子型变量中实际存储的数字，比如：

```
as.numeric(rdata)
```

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

注意：也存在其他相似的进行类型转换的函数，比如：as.character(), as.factor()等等

对于已经创建好的因子，我们可以利用levels()函数更改因子的级别。比如以上创建的fdata对象，本来其创建的缺省级别为数字：

```
levels(fdata)
```

```
[1] "1" "2" "3"
```

利用levels()函数更改fdata的级别：

```
levels(fdata) = c("I", "II", "III")
fdata
```

```
[1] I   II  II  III I   II  III III I
[10] II  III III I
Levels: I II III
```

```
levels(fdata)
```

```
[1] "I" "II" "III"
```

5.2 R的工作空间 (workspace) 和工作路径 (work directory)

5.2.1 工作空间 (workspace)

R的工作空间 (workspace) 就是当前R的工作环境，其储存着所有用户定义的对象（比如：向量、矩阵、数据框、列表，还有比如以后会要学习的我们可以自己定义的函数对象）。

- 数据对象通过R的工作空间来存留在电脑的内存中。
- 你可以用`objects()`或`ls()`函数来显示当前在工作空间的对象。

```
objects()
ls()
```

5.2.2 工作路径 (work directory)

- 如果你不显式的指定工作路径，R会缺省的认为你想要从当前的工作路径加载对象或者将对象存取到当前工作路径。
- 用`getwd()`来显示当前工作路径：

```
getwd()
```

- 用`setwd()`来改变当前工作路径；假设在C盘下存在一个叫做“R-Learning”的文件夹，我们可以用以下命令将工作路径设置为这个文件夹：

```
setwd("C:/R-Learning")
```

- 在Linux操作系统中，使用/来在路径中进行分隔；而在Windows操作系统中，使用/或者\\来在路径中进行分隔。
- 在R Markdown (.Rmd) 文件中运行R程序时，工作路径总是会被设置成这个.Rmd文件所在的文件夹。即使你用`setwd()`函数在一个notebook chunk（即.Rmd文件中的一个“{r}”程序块中的程序）中使用`setwd()`函数将工作路径设置到其他文件夹，在这个notebook chunk的程序运行完后，工作路径还是会被重新设置成这个.Rmd文件所在的文件夹。

5.2.3 .RData文件类型

我们可以使用`save()`函数来将一个R的对象存储成.RData的文件类型。比如以下我们灵活运用几个新的函数将cars数据框存储到我们在工作路径中创建的“work”子文件夹的一个.RData文件中。以下的`file.exists()`函数是一个检验一个文件或者文件夹(这里是“work”子文件夹)是否存在的函数，如果存在，这个函数就会返回TRUE,如果不存在，这个函数就会返回FALSE;用!符号对逻辑结果作“取反”操作，即如果这个“work”文件夹不存在，我们就使用`dir.create()`函数在工作路径中创建这个“work”文件夹。最后，使用`save()`函数将cars.mod对象存储到“work”文件夹的cars_mod.RData文件中。`file.path()`函数类似于之前学的`paste()`函数，它会根据操作系统的不同，用合适的分隔符来连接作为其参数的字符型的文件夹名，这个函数产生的结果是一个当前操作系统下的字符型路径。

```
cars.mod <- cars
outdir = "work"
if(!file.exists(outdir)) dir.create(outdir)
save(cars.mod, file = file.path(outdir, "cars_mod.RData"))
```

另外，我们也可以使用`save.image()`函数将R工作空间的所有对象都写入一个.RData型的文件进行保存：

```
save.image(file.path(outdir, "workspace.RData"))
```

在一个R会话(session)结束时（关闭R或者RStudio时，或者执行quit()函数时），你可以选择将当前工作空间保存到一个镜像中（系统会自动保存到Home目录中的一个叫做“.RData”的文件中），并在下次启动R时自动载入它。我们可以通过以下两个语句查看电脑的Home目录在哪里（先将当前工作路径设置成Home目录"，然后查看当前工作路径）：

```
setwd("~/")
getwd()
```

5.2.4 载入之前保存的工作空间文件

load()函数用来将之前保存的.RData文件导入R的工作空间（在此之前，先使用remove()函数从R的工作空间中删除cars.mod数据框，以验证load()函数可以成功从存储在硬盘上的cars_mod.RData文件中将cars.mod数据框导入到R的工作空间）：

```
remove(cars.mod)
load(file.path(outdir, "cars_mod.RData"))
```

5.3 从硬盘导入数据、导出数据到硬盘

5.3.1 从硬盘导入数据：read.table()和read.csv()函数

- read.csv()函数允许你直接读取“逗号分隔型”格式的数据，甚至可以从一个web URL链接直接进行读取：

```
url <- "data/google_stock_data.csv"
GOOG.data <- read.csv(url)
str(GOOG.data)
```

```
'data.frame': 923 obs. of 7 variables:
 $ Date      : Factor w/ 923 levels
   "2007-07-09","2007-07-10",...: 923 922 921 920 919 918 917
   916 915 914 ...
 $ Open      : num 608 606 600 618 610 ...
 $ High      : num 609 611 606 619 616 ...
 $ Low       : num 600 605 595 599 608 ...
 $ Close     : num 601 610 601 601 613 ...
 .....
```

- read.table()函数和read.csv()函数的使用方法类似，但是参数的缺省值不太一样。比如head参数在read.csv()中的缺省值为TRUE,表示read.csv()函数缺省把外部文件的第一行当作列名，而在read.table()中的缺省值却为FALSE,表示read.table()函数缺省把外部文件的第一行当作数据的第一行。又比如，sep参数在read.csv()中的缺省值为",",表示read.csv()函数缺省把","当作外部文件列与列之间的分隔符，而在read.table()中的缺省值却为" ",表示read.table()函数缺省把空格当作外部文件列与列之间的分隔符。注意，如果sep参数被设置成"\t",那么表示读取外部数据时使用TAB作为外部文件列与列之间的分隔符。

5.3.2 导出数据到硬盘：write.table()和write.csv()函数

- 用write.table()或者write.csv()函数来导出矩阵或数据框。write.table()函数导出数据时缺省使用空格作为外部数据列与列之间的分隔符，而write.csv()函数导出数据时缺省使用逗号作为外部数据列与列之间的分隔符。

```
write.table(GOOG.data, file = "google_data.txt")
write.csv(GOOG.data, file = "google_data.csv")
```

6. 常见运算符及其向量型计算

我们之前已经在第2章中接触过一点R中的向量型计算的例子，其计算时会自动的对向量中的元素进行“智能”的循环操作，而无需像传统的计算机语言比如C语言中那样需要编写显式的循环来完成这一操作。向量型计算是一种在R中对向量进行大规模计算时容易且有效率的方式（不管从程序编写的角度还是从程序运行的角度）。向量型计算的程序代码易于编写，易于阅读，易于调试，运行起来也更快。

6.1 用常见的运算符和操作符进行向量型计算

以下再举一些在R中用一些常见的运算符和操作符进行向量型计算的例子：

```
1:5 + 6:10
```

```
[1] 7 9 11 13 15
```

又比如：

```
c(1, 3, 6, 10, 15) + c(0, 1, 3, 6, 10)
```

```
[1] 1 4 9 16 25
```

不只是加号（+），其他所有算术运算符都是向量化的。下例将演示减法、乘法、幂运算、两种除法及余数。

```
c(2, 3, 5, 7, 11, 13) - 2 # 减法
```

```
[1] 0 1 3 5 9 11
```

```
-2:2 * -2:2 # 乘法
```

```
[1] 4 1 0 1 4
```

```
2 ^ 3 # 求幂，^ 或都可以，但用**^ 更加普遍  
2 ** 3
```

```
[1] 8
```

```
1:10 / 3 # 浮点数除法
```

```
[1] 0.3333333 0.6666667 1.0000000
[4] 1.3333333 1.6666667 2.0000000
[7] 2.3333333 2.6666667 3.0000000
[10] 3.3333333
```

```
1:10 %/% 3 # 整数除法
```

```
[1] 0 0 1 1 1 2 2 2 3 3
```

```
1:10 %% 3 # 求余数
```

```
[1] 1 2 0 1 2 0 1 2 0 1
```

R 还包含了多种数学函数。比如有三角函数（sin、cos、tan，以及相反的asin、acos和atan）、对数和指数（log和exp）等等。所有的函数都作用于向量，而不仅仅是单个值。

6.2 向量循环和重复

到目前为止，我们学过的向量型运算包含两种情况：

1. 一个向量和一个单独的元素进行运算，比如：

```
1:5 + 1
```

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

```
paste("row", 1:5, sep = "_")
```

```
[1] "row_1" "row_2" "row_3" "row_4" "row_5"
```

2. 两个相同长度的向量进行运算，比如：

```
1:5 + 1:5
```

```
[1] 2 4 6 8 10
```

```
paste(c("Ross", "Robert"), c("Ihaka", "Gentleman"))
```

```
[1] "Ross Ihaka" "Robert Gentleman"
```

当对不同长度的长度都不为1的向量做以上类似的运算，结果会如何呢？R 将会循环较短向量中的元素以配合较长的那个：

```
1:5 + 1:15
```

```
[1] 2 4 6 8 10 7 9 11 13 15 12 14 16 18 20
```

如果长向量的长度不是短向量长度的倍数，将出现一个警告：

```
1:5 + 1:7
```

```
[1] 2 4 6 8 10 7 9
Warning message:
In 1:5 + 1:7 :
  longer object length is not a multiple of shorter object length
```

必须强调的是，虽然我们可以在不同长度的向量之间做运算，但这并不意味着应该这样做。为向量添加一个标量值没有问题，但我们会在其他方面把自己搞晕。更好的做法是明确地创建同等长度的向量，然后再对它们进行操作。`rep`函数非常适合此类填充向量的任务，它允许我们重复使用元素来创建向量：

```
rep(1:5, 3)
```

```
[1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
```

```
rep(1:5, each = 3)
```

```
[1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
```

```
rep(1:5, times = 1:5)
```

```
[1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5
```

```
rep(1:5, length.out = 7)
```

```
[1] 1 2 3 4 5 1 2
```

6.3 在数据框中进行向量计算

6.3.1 例子：均值中心化

目标：创建一个新变量，这个新变量是对mtcars数据表中的hp变量进行均值中心化处理后的结果。

模仿传统语言中的处理方法：使用循环

注意：以下在R中使用for循环的方法目前还没讲授，之后会在将流程控制的章节讲授。

$$\bar{X} = \frac{1}{N} \sum_1^N X$$

```
hp.sum <- 0
hp.n <- nrow(mtcars)
for (i in 1:length(mtcars$hp)) {
  hp.sum <- hp.sum + mtcars$hp[i]
}
hp.mean <- hp.sum/hp.n
mtcars$hp.cent <- NA
for (i in 1:length(mtcars$hp)) {
  mtcars$hp.cent[i] <- mtcars$hp[i] - hp.mean
}
```


R的处理方法: 向量型计算

```
mtcars$hp.cent2 <- mtcars$hp - mean(mtcars$hp)
head(mtcars[c("hp", "hp.cent", "hp.cent2")])
```

```
      hp  hp.cent hp.cent2
Mazda RX4      110 -36.6875 -36.6875
Mazda RX4 Wag  110 -36.6875 -36.6875
Datsun 710       93 -53.6875 -53.6875
Hornet 4 Drive  110 -36.6875 -36.6875
Hornet Sportabout 175  28.3125  28.3125
Valiant        105 -41.6875 -41.6875
```

6.3.2 例子：计算加速度

比如我们也可以对cars数据表的各列进行向量型计算。这个数据框有两个列：

1. speed numeric Speed (mph)
2. dist numeric Stopping distance (ft)

我们来算一算每辆汽车停车时乘客所经历的加速度（acceleration）。加速度的计算公式为：

$$a = \frac{-v^2}{2d}$$

其中，s为速度（speed），d为位移（distance）。

我们先计算分母：

```
distance <- cars$dist
```

现在这个distance是一个向量，其中的每个元素对应每辆车的停车位移。我们将整个向量乘以2，就得到了将其中每个元素乘以2的效果。不需要一个一个的乘：

```
denominator <- 2 * distance
denominator
```

```
[1] 4 20 8 44 32 20 36 52 68 34 56 28 40 48 56
[16] 52 68 68 92 52 72 120 160 40 52 108 64 80 64 80
[31] 100 84 112 152 168 72 92 136 64 96 104 112 128 132 108
[46] 140 184 186 240 170
```

然后计算分子。使用同样的技巧：

```
numerator <- -(cars$speed^2)
numerator
```

```
[1] -16 -16 -49 -49 -64 -81 -100 -100 -100 -121 -121 -144
[13] -144 -144 -144 -169 -169 -169 -169 -196 -196 -196 -196 -225
[25] -225 -225 -256 -256 -289 -289 -289 -324 -324 -324 -324 -361
[37] -361 -361 -400 -400 -400 -400 -400 -484 -529 -576 -576 -576
[49] -576 -625
```

我们试着把计算出的分子和分母都添加到cars数据表中，作为这个数据表的两个新的列：

```
cars$numerator <- numerator
cars$denominator <- denominator
```

然后将这两列按照加速度的公式组合成这个数据表中的一个新的列，这一列就是我们想要的结果：

```
cars$acceleration <- cars$numerator/cars$denominator
```


我们也可以用一行语句完成以上所有的计算步骤:

```
cars$acceleration <- -(cars$speed^2)/(2 * cars$dist)
```

注意: 我们可以使用赋值为NULL的方式来删除数据框中那些不需要的列; 比如, 我们可以删除cars数据表中的两列不再需要的数据:

```
cars$numerator <- NULL  
cars$denominator <- NULL
```

通常, 我们可以用赋值为NULL的方式来删除列表元素中的元素 (数据框也是一种列表)。

7. 逻辑操作符的运用，非正常值

7.1 逻辑操作符的向量化运用

我们之前学了如何将+、-、*、^、以及/等运算符用在向量上。以下这些逻辑比较符的使用也是可以向量化的：等于（==），不等于（!=），大于（>），小于（<），大于或等于（>=），小于或等于（<=）。以下举几个例子：

```
c(3, 4 - 1, 1 + 1 + 1) == 3
```

```
[1] TRUE TRUE TRUE
```

```
exp(1:5) < 100
```

```
[1] TRUE TRUE TRUE TRUE FALSE
```

```
(1:5) ^ 2 >= 16
```

```
[1] FALSE FALSE FALSE TRUE TRUE
```

我们也可以使用==来比较字符串。在这种情况下，比较会区分大小写，所以字符串必须完全匹配

```
c(
  "Can", "you", "can", "a", "can", "as",
  "a", "canner", "can", "can", "a", "can?"
) == "can"
```

```
[1] FALSE FALSE TRUE FALSE TRUE
[6] FALSE FALSE FALSE TRUE TRUE
[11] FALSE FALSE
```

如果我们想要识别mtcars数据框中哪些汽车是低油耗的（即高mpg）：

```
mtcars$mpg > median(mtcars$mpg)
}
```

```
[1] TRUE TRUE TRUE TRUE FALSE FALSE FALSE TRUE
[9] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[17] FALSE TRUE TRUE TRUE TRUE FALSE FALSE FALSE
[25] FALSE TRUE TRUE TRUE FALSE TRUE FALSE TRUE
```

产生了一个只包含了TRUE/FALSE值的逻辑型向量。

7.2 用逻辑型向量来提取数据

我们可以使用逻辑操作符（和向量）来识别cars数据表中的哪些车的停车加速度小于-3：

```
cars$acceleration < -3
```

```
[1] TRUE FALSE TRUE FALSE FALSE TRUE FALSE FALSE
[9] FALSE TRUE FALSE TRUE TRUE FALSE FALSE TRUE
[17] FALSE FALSE FALSE TRUE FALSE FALSE FALSE TRUE
[25] TRUE FALSE TRUE TRUE TRUE TRUE FALSE TRUE
[33] FALSE FALSE FALSE TRUE TRUE FALSE TRUE TRUE
[41] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[49] FALSE TRUE
```

然后我们就可以用产生的这个包含逻辑型向量的操作符来选取或者更改数据中的子集：

```
length(cars$acceleration[cars$acceleration < -3])
```

```
[1] 28
```

我们也可以用逻辑型向量对一个向量的子集进行赋值。我们现在已经找出了所有停车加速度小于-3的汽车。

如果我们认为这些小于-3的停车加速度的值都是不正常的，我们想要将所有这些小于-3的值替换成-3,该如何做？

```
cars$acceleration[cars$acceleration < -3] <- -3
```

7.3 三种非正常值：缺失值，不定值，无穷值

7.3.1 缺失值：NA (not available)

- 缺失值是什么？
- 用is.na()函数来检测是否为缺失值。
- 初始化数据时，可以用来当占位符（placeholder）。

```
x <- 1:3
x[4]
```

```
[1] NA
```

使用缺失值NA

我们可以将NA直接插到对象中

```
x <- c(1, 2, 3, NA, 4, 5)
```

is.na(x)返回一个逻辑型向量。当x中的元素为NA时，在这个逻辑型向量中对应的元素就为TRUE。

```
is.na(x)

[1] FALSE FALSE FALSE  TRUE FALSE FALSE

x[is.na(x)] <- 3.5
x

[1] 1.0 2.0 3.0 3.5 4.0 5.0
```

7.3.2 不定值：NaN (not a number)

- 0/0 是NaN
- 用is.nan()函数来检测是否为不定值
- 注意：is.na(NaN)的结果为TRUE。

```
0/0

[1] NaN
```

7.3.3 无穷值：Inf 和 -Inf

- 任意一个非0的数除以0的结果。
- R知道如何处理Inf,并不会因为碰到Inf而报错。

```
1/0

[1] Inf
```


The background image shows a modern, multi-story building with a curved facade and many windows, situated behind a line of trees. In the foreground, there is a large, circular, metallic structure, possibly a dome or a large clock face, with a tall, thin pole rising from its center. The sky is blue with scattered white clouds.

Bibliography