

R语言基础

第一讲&第二讲



R语言简介及特点

“R是统计领域广泛使用的诞生于1980年左右的[S语言](#)的一个分支。可以认为R是S语言的一种实现。

而S语言是由AT&T贝尔实验室开发的一种用来进行数据探索、统计分析和作图的[解释型语言](#)。

最初S语言的实现版本主要是[S-PLUS](#)。S-PLUS是一个[商业软件](#)，它基于S语言，并由MathSoft公司的统计科学部进完善。

后来新西兰奥克兰大学的Robert Gentleman和Ross Ihaka及其他志愿人员开发了一个R系统。由“R Core Team”负责开发。

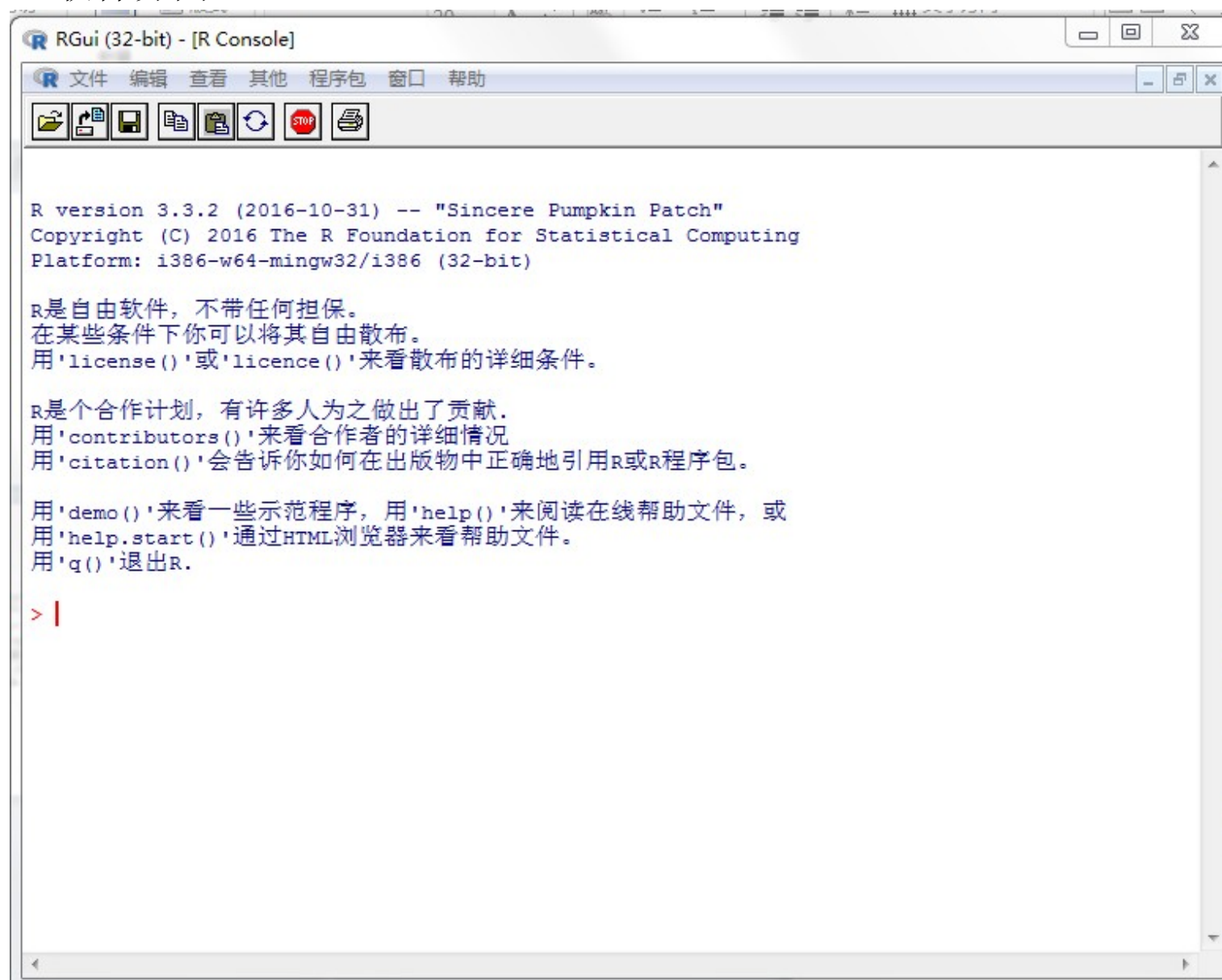
所以, R与S-PLUS在程序语法上可以说是几乎一样的,可能只是在函数方面有细微差别,程序十分容易地就能移植到一程序中。”



R语言简介及特点

- R是自由、免费、源代码开放的数据分析软件，
- 全面的统计研究平台，在统计分析、数据挖掘和生物信息学等领域提供了多样化的数据分析技术
- 具有顶尖水平的制图功能
- R可运行于多种平台之上，包括Windows、Mac OS X及Linux
- R语言和其它编程语言、数据库之间有很好的接口
- R提供了非常丰富的程序包，除了推荐的标准包外还有很多志愿者贡献的贡献包，可以直接利用这些包，大大提高工作效率。
- R的编程思想非常简单

R 软件界面

A screenshot of the RGui (32-bit) - [R Console] window. The window has a title bar with standard Windows controls. Below the title bar is a menu bar with options: 文件 (File), 编辑 (Edit), 查看 (View), 其他 (Other), 程序包 (Packages), 窗口 (Windows), and 帮助 (Help). Below the menu bar is a toolbar with icons for file operations (open, save, print, etc.) and a stop button. The main area is a text console showing the R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch" startup message, copyright information, and a series of instructions in Chinese about the R license and how to use the console. The console text is as follows:

```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R是自由软件，不带任何担保。
在某些条件下你可以将其自由散布。
用 'license()' 或 'licence()' 来看散布的详细条件。

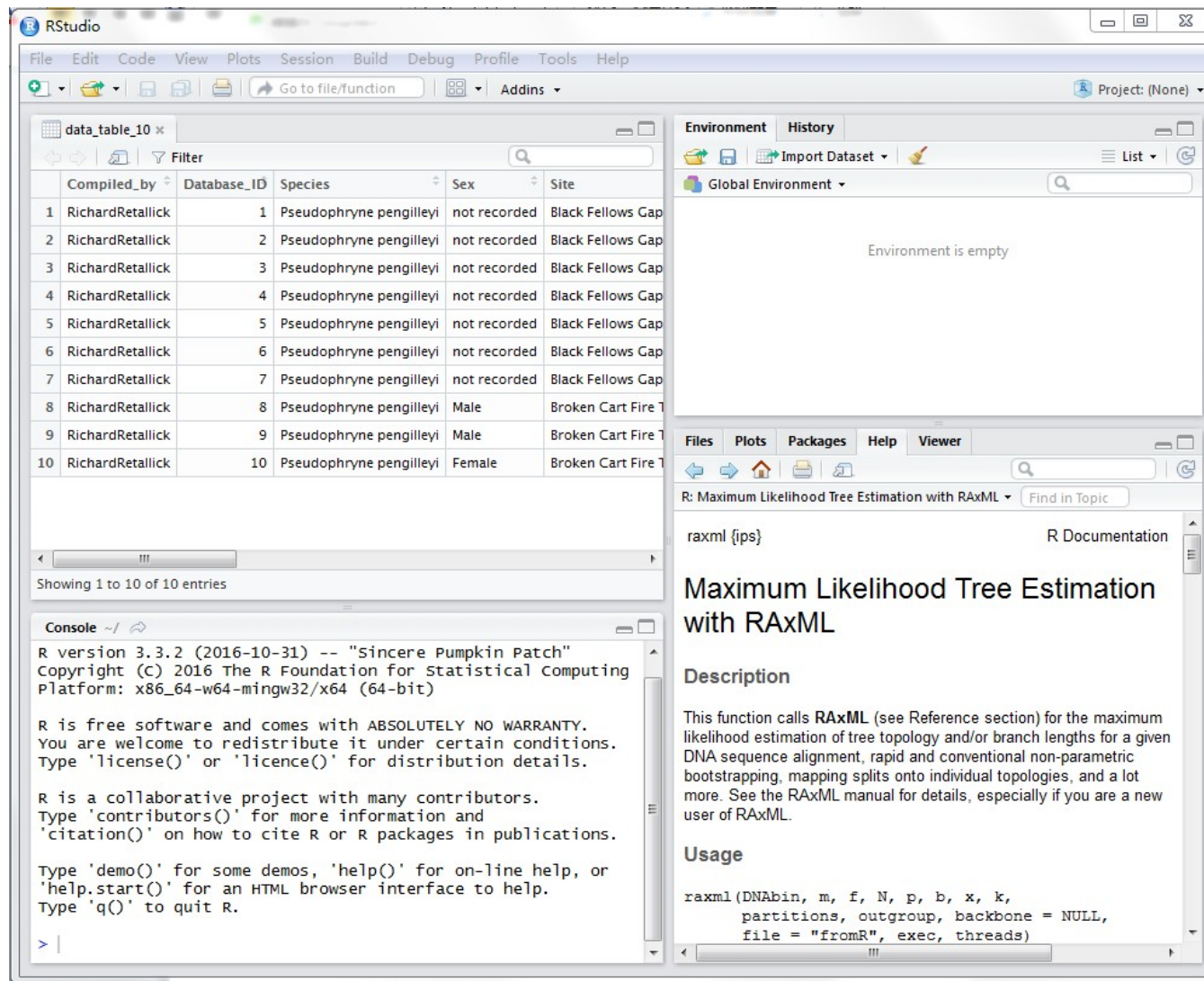
R是个合作计划，有许多人为之做出了贡献。
用 'contributors()' 来看合作者的详细情况
用 'citation()' 会告诉你如何在出版物中正确地引用R或R程序包。

用 'demo()' 来看一些示范程序，用 'help()' 来阅读在线帮助文件，或
用 'help.start()' 通过HTML浏览器来看帮助文件。
用 'q()' 退出R。

> |
```

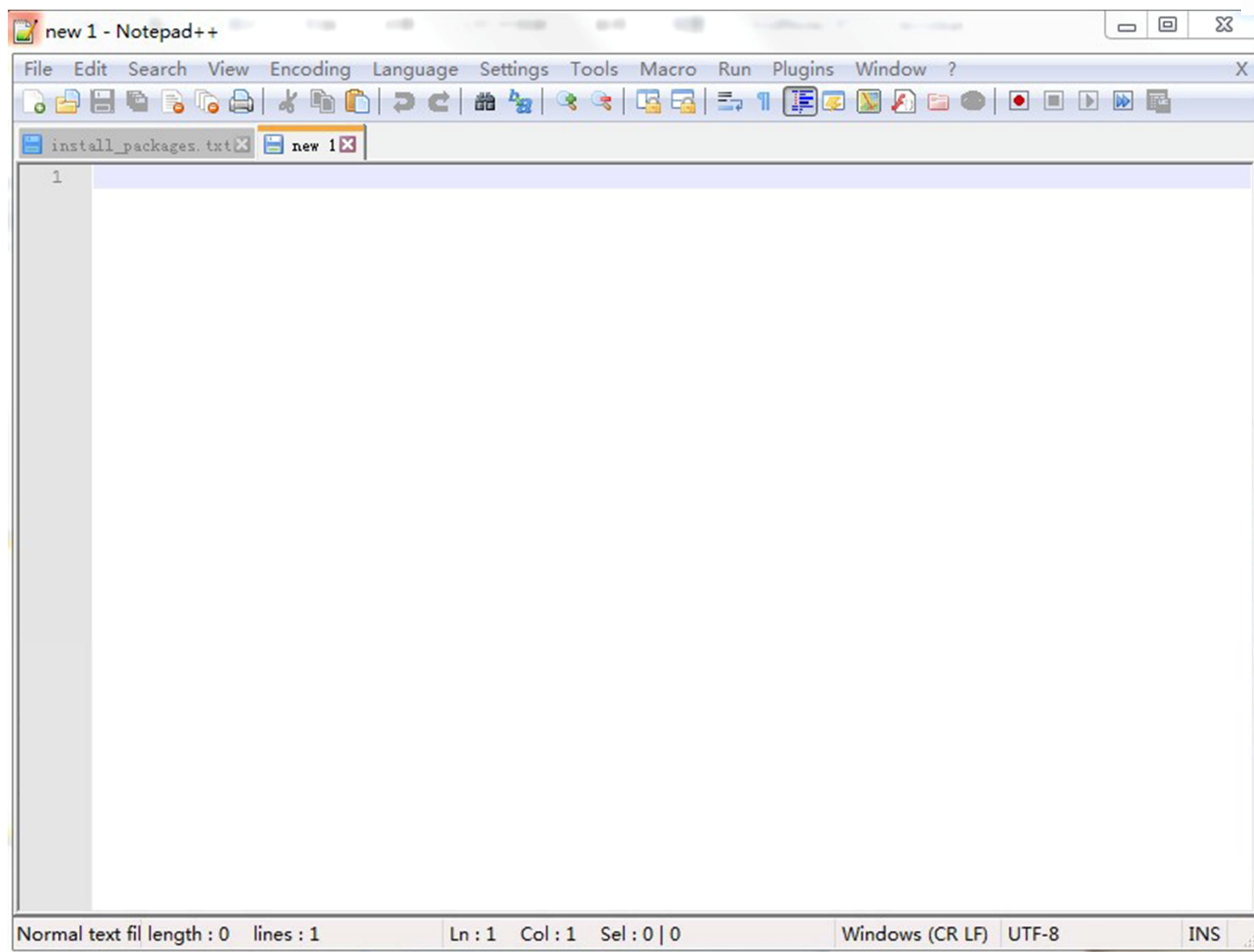


Rstudio-友的好程序界面



因为是友好界面，在Rstudio上运行R更为简易方便

Notepad++ 功能强大的程序编辑器





R语言基本语法

命令提示符“>”，在该提示符后输入命令，如：

```
> 1+2
```

在命令后输入回车键，即得到结果

```
[1] 3
```

练习：输入

```
> 5*7-8
```

回车后看结果是什么

所以，R也能当成一个功能强大的计算器，当然R的作用远不止于些

当表达式不在同一行写完时

```
> (15-3*2)*(4*3
```

```
+ +9)
```

```
[1] 189
```

第二行的“+”号是一个继续的命令提示符，在符号后继续输入余下的表达式即可



程序包及路径

安装及调用程序包

```
install.packages( "MBCluster.Seq" ) #安装程序包  
library(MBCluster.Seq) #调用程序包
```

设置路径

```
> getwd()  
[1] "C:/Users/Angel/Documents"
```

改变工作路径

```
> setwd("D:/data")  
> getwd()  
[1] "D:/other"
```

变回到原来的工作路径

```
> setwd("~")  
> getwd()  
[1] "C: /Users/Angel /Documents"
```




R的帮助文档

用help() 或 ?命令查看帮助文档

例:

```
> ?log
```

```
> help()
```

- 描述(Description): 描述某个命令/函数的功能
- 用法(Usage): 基本用法, 命令的格式。
- 参数(Arguments): 所需的参数, 规定了参数的数据类型、格式等各种详细要求。
- 细节(Details): 其它补充细节
- 取值(Values:) 命令/函数的输出的结果
- 例子(Examples): 一些有用的例子



R的帮助文档

- Help文档不够具体或不够用时，可直接上网查询，用google或必应(www.bing.com)输入适当的英文关键词进行查询



变量和赋值

- 赋值符号 “<-”

赋值符号也可以用 “=”，但 “=”除了赋值之外，还具有传递函数的功能，因此有时会造成歧义，所以建议赋值符号采用 “<-”

```
> x <- 1 # assign value 1 to variable x
> x
[1] 1
> x < -1 # Is x greater than 1 or not? Return
logical value
> [1] FALSE
```

“#”号，非命令起始符，在其后写标注文字等，程序运行时，“#”号及后面的文字不参与程序运行



向量的操作及运算

函数c() - 数值向量函数 (Vector Function)

```
> x <- c(1, 3, 5.8, 7.1, 9)
> x [1] 1.0 3.0 5.8 7.1 9.0
```

函数:- 数列函数

```
x <- 1:9
> x
[1] 1 2 3 4 5 6 7 8 9
```

```
> y <- 5:1
> y
[1] 5 4 3 2 1
```



向量的操作及运算

函数seq() –数列 (sequence) 生成

```
> seq(-2,2)
```

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

```
> seq(2,-2)
```

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

```
> seq(2,-2,by=-.5)
```

```
[1] 2.0 1.5 1.0 0.5 0.0 -0.5 -1.0 -1.5 -2.0
```

```
> seq(from=2,to=-2,length=11)
```

```
[1] 2.0 1.6 1.2 0.8 0.4 0.0 -0.4 -0.8 -1.2 -1.6 -2.0
```

- 函数rep() –用各种方式重复(repeat)一个对象

```
> x <- 1:3
```

```
> rep(x,each=5)
```

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

```
> rep(x,times=5)
```

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



向量的操作及运算

- 一个向量有一个或很多值组成，若想知道向量中某些位置的数值，可以用指标符号“方括号”[]得到. 向量的元素下标取值是以1开始

```
> x <- 6:12
```

```
> x[1]
```

```
[1] 6
```

```
> x[2:3]
```

```
[1] 7 8
```

也可以改变向量某个或某些位置的值

```
> x[3] <- 0
```

```
> x
```

```
[1] 6 7 0 9 10 11 12
```

也可以一次取出几个位置的值

```
> i <- c(2,4)
```

```
> x[i]
```

```
[1] 7 9
```

```
> x[-i]
```

```
[1] 6 0 10 11 12
```



向量的操作及运算

向量元素的添加及合并

```
>v <- c(1,2,3)
> v <- c(v,55) #格式为新向量<-(原向量, 新元素)
> v
[1] 1 2 3 55
> v1 <- c(1, 2, 3)
> v2 <- c(4, 5, 6)
> v3 <- c(v1,v2)
[1] 1 2 3 4 5 6
```



向量的操作及运算

向量排序

`sort()`: 输出排序后的结果; `order()`: 输出排序后的各个分量位置;

`rank()`: 各个分量的排名

```
> a <- c(3,9,0,12,19)
```

```
> sort(a)
```

```
[1] 0 3 9 12 19
```

```
> order(a)
```

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

```
> rank(a)
```

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




数学运算符

常用的数学运算包括

`+, -, *, /, ^, %%, %/%` # 四则、同余

`<, >, <=, >=, ==, !=` # 比较、逻辑

`log(x), log10(), log2(), exp(), expm1(), log1p()` # 对数、指数

`sqrt(), sign(), factorial()` # 开方、符号、阶乘

`union(), append(), intersect()` # 集合

`max(), min(), which.max(), which.min()` # 最大值，最小值

`identical(), all.equal(), setdiff(), setequal()` # 比较

`append(), rbind(), cbind()` # 合并

`sum(), prod(), cumsum(), cumprod()` # 连加、连乘

`length(), range(), round()` # 长度，范围，取整

*** 绿色字的部分为非常常用的运算符



常规数学运算

练习

- 计算 $\log(3)$, $\log_{10}(100)$, $\log_2(8)$
 - 计算 $\sqrt{3}$, 6.5^3
 - $x \leftarrow c(3,8,4,10,12,5)$, 计算 $\max(x)$, $\min(x)$, $\text{which.max}(x)$, $\text{which.min}(x)$
 - $\text{length}(x)$, $\text{range}(x)$
 - $\text{round}(3.1415926, \text{digits}=2)$
 - $\text{sum}(1.1, 2.3, 2.1, 5.6, 8.5)$
- * `> y1 <- c(1,2,3,4)`
`> y2 <- c(2,3,4,5)`
`> y3 <- c(5,6,7,8)`
`> cbind(y1, y2, y3) >`
`> rbind(y1, y2, y3)`



矩阵及基本操作

定义一个矩阵

一个矩阵可以通过改变某个向量的维数来定义，比如一个长度为20 的向量可以变成4×5 的矩阵

```
> x <- 1:20
```

```
> y <- matrix(x,nrow=4,ncol=5)
```

```
> y
```

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



矩阵及基本操作

可见矩阵是按照列将向量的各个元素依次排列。如果想按照行排列，就可以修改参数byrow=TRUE。

```
> z <- matrix(x,4,5,byrow=TRUE)
```

```
> z
```

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



矩阵及基本操作

合并矩阵

`rbind` 和 `cbind` 是两个可以合并矩阵(或向量) 的函数，分别按照行(row) 和列(column), 但合并的时候要保证行数或列数相等。继续上面的例子：

```
> rbind(y,z)
```

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



矩阵及基本操作

```
> cbind(y,z)
```

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



矩阵及基本操作

子矩阵

可以使用方括号提取矩阵的元素, $y[i,j]$ 表示矩阵第 i 行第 j 列的那个元素, $y[i,]$ 表示第 i 行, $y[,j]$ 表示第 j 列。若在 i 或 j 前面加上负号 -, 则表示除去这些行或列剩下的部分

```
> y[2,4]
```

```
[1] 14
```

```
> y[,4]
```

```
[1] 13 14 15 16
```

```
> y[2,]
```

```
[1] 2 6 10 14 18
```

```
> y[-2,]
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	5	9	13	17
[2,]	3	7	11	15	19
[3,]	4	8	12	16	20



矩阵及基本操作

```
> y[,-4]
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	5	9	17
[2,]	2	6	10	18
[3,]	3	7	11	19
[4,]	4	8	12	20

也可以提取一个子矩阵，比如

```
> i <- c(1,3)
```

```
> j <- c(2,3,5)
```

```
> y[i,j]
```

	[,1]	[,2]	[,3]
[1,]	5	9	17
[2,]	7	11	19

```
> y[-i,-j]
```

	[,1]	[,2]
[1,]	2	14
[2,]	4	16



矩阵及基本操作

给矩阵行或列起名

可以用rownames 或colnames 为矩阵的行或列命名，名字应该是一串数字或字符，注意名字的长度必须与行数或列数相等。

```
> rownames(y)=c("apple","orange","peach","grape")  
> colnames(y)=c("yellow","red","blue","green","purple")  
> y
```

	yellow	red	blue	green	purple
apple	1	5	9	13	17
orange	2	6	10	14	18
peach	3	7	11	15	19
grape	4	8	12	16	20



矩阵及基本操作

```
> y["apple",]
```

```
yellow    red    blue  green purple  
      1      5      9     13     17
```

```
> y[,c("red","blue")]
```

```
      red blue  
apple    5    9  
orange    6   10  
peach     7   11  
grape     8   12
```



矩阵及基本操作

矩阵函数

常用的矩阵函数有如下几个

`t()` ## 矩阵转置

`dim()`, `nrow()`, `ncol()` ## 维数、行数、列数

`rowMeans()`, `rowSums`, `colMeans()`, `colSums()` ## 每行/列的平均值/和

以前面的`y`为例，练习

`t(y)`, `dim(y)`, `nrow(y)`, `ncol(y)`,

`rowMeans(y)`, `rowSums(y)`, `colMeans(y)`, `colSums(y)`



非数值型数据

字符串

字符串（`string`）可以用“”标示的一串字符，R 用一个向量存储，比如

```
> s <- c("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday")  
> s[2:3]  
[1] "Monday" "Tuesday"
```



非数值型数据

因子

因子（**factor**）是一个对等长的其他向量元素进行分类（分组）的向量对象。R 同时提供有序（**ordered**）和无序（**unordered**）因子。无序的因子可以由任意字符串或数字（数字被当成字符串处理）组成，并可以用函数**factor()**、**ordered()** 转换为有序的因子

```
> grade <- c("A","1",100,"100","a",1,"*",TRUE,"Z")
```

```
> factor(grade)
```

```
[1] A 1 100 100 a 1 * TRUE Z
```

```
Levels: * 1 100 a A TRUE Z
```

```
> ordered(grade)
```

```
[1] A 1 100 100 a 1 * TRUE Z
```

```
Levels: * < 1 < 100 < a < A < TRUE < Z
```



非数值型数据

无序的字符串没有水平(levels) 之分，而有序的因子存在高低之分

```
> grade <- c("A","1",100,"100","a",1,"*",TRUE,"Z")
```

```
> levels(grade)
```

```
NULL
```

```
> Grade <- ordered(grade)
```

```
> levels(Grade)
```

```
[1] "*" "1" "100" "a" "A" "TRUE" "Z"
```

可以将有序的因子转换为正整数，而无序的字符串的不可以

```
> levels(grade)
```

```
> k <- as.numeric(grade)
```

```
Warning message:
```

```
NAs introduced by coercion
```

```
> k <- as.numeric(Grade)
```

```
> k
```

```
[1] 5 2 3 3 4 2 1 6 7
```



R的列表 (List)

R 的列表 (list) 是一个以对象组成的有序集合构成的对象。列表中包含的对象又称为它的分量 (components)。分量可以是不同的模式或类型，如一个列表可以同时包括数值向量，逻辑向量，矩阵，复向量，字符数组，函数等等。下面的例子演示怎么创建一个列表：

```
> Lst <- list(name="Fred", wife="Mary", no.children=3,child.ages=c(4,7,9))
```

```
> Lst
```

```
$name
```

```
[1] "Fred"
```

```
$wife
```

```
[1] "Mary"
```

```
$no.children
```

```
[1] 3
```

```
$child.ages
```

```
[1] 4 7 9
```



R的列表 (List)

分量常常会被编号的 (numbered)，并且可以利用这种编号来访问分量。如果列表Lst 有四个分量，这些分量则可以用Lst[[1]], Lst[[2]], Lst[[3]] 和Lst[[4]] 独立访问。如果Lst[[4]] 是一个有下标的数组，那么Lst[[4]][1] 就是该数组的第一个元素。这里特别要注意一下Lst[[1]] 和Lst[1] 的差别。前者得到的是列表Lst 中的第一个对象, 后者得到的则是列表Lst 中仅仅由第一个元素构成的子列表。

```
> Lst[4]
$child.ages
[1] 4 7 9
> Lst[[4]]
[1] 4 7 9
> Lst[[4]][2]
[1] 7
```




R的列表 (List)

列表的分量可以被命名，这种情况下可以通过名字访问。此时，可以把字符串形式的分量名字放在列表名后面的双中括号中，或者用一个\$符号连接，比如

```
> Lst$child.ages
```

```
[1] 4 7 9
```

```
> Lst$child.ages[2]
```

```
[1] 7
```



数据框 (Data Frame)

数据框 (data frame) 是一个属于 "data.frame" 类的列表。不过，对于可能属于数据框的列表对象有下面一些限制条件，

- 分量必须是向量(数值, 字符, 逻辑), 因子, 数值矩阵, 列表或者其他数据框;
- 矩阵, 列表和数据框为新的数据框提供了尽可能多的变量, 因为它们各自拥有列, 元素或者变量;
- 数值向量, 逻辑值, 因子保持原有格式, 而字符向量会被强制转换成因子并且它的水平就是向量中出现的独立值;
- 在数据框中以变量形式出现的向量结构必须长度一致, 矩阵结构必须有一样的行数.



数据框 (Data Frame)

比如

```
> city <- c("DC","LA","NY")  
> PI <- c(11.3,10.2,13.0)  
> FS <- c(4.2,4.5,3.5)  
> fin <- data.frame(City=city,Person.Income=PI,Family.Size=FS)  
> fin
```

访问数据框特定的列可以用\$ 符号

```
> fin$City  
[1] DC LA NY  
Levels: DC LA NY
```



数据类型的判断与转换

- 数据类型的判断: `is.numeric()` `is.factor()` `is.character()` `is.data.frame()`
- 数据类型的转换: `as.numeric()` `as.factor()` `as.character()`
- `data.frame`转换成数值

```
data1 <- as.matrix(fin[,2:3])
```

```
class(data1) <- "numeric"
```

```
fin.new <- data.matrix(data1)
```

* 练习: 对以下数据进行判断

```
is.numeric(fin.new)
```

```
is.numeric(fin[,2:3])
```



数据的读写

(i) 读取

需要确定当前R的工作路径(working directory)，通过setwd() 和getwd() 查看或改变默认工作路径

R能读取多种存储结构的数据，但最常用、最好用的是.txt 文件和.csv 文件

假设D:/data 下有两个文件, authors.csv 和book.txt，则可以分别用read.csv() 和read.table() 读取，读取的结果分别存入了两个数据框a 和b 中。



数据的读写

```
a <- read.csv("authors.csv")
```

a

	surname	nationality	deceased
1	Tukey	US	yes
2	Venables	Australia	no
3	Tierney	US	no
4	Ripley	UK	no
5	McNeil	Australia	no

```
b <- read.table("books.txt", header = TRUE, sep="\t")
```

b

	surname	nationality	deceased	x
1	1	Tukey	US	yes
2	2	Venables	Australia	no
3	3	Tierney	US	no
4	4	Ripley	UK	no
5	5	McNeil	Australia	no



数据的读写

对应的，可以用`write.csv()` 或`write.table()` 将数据写入文件。若给定的文件不存在，则创建新的文件；若已经存在，则会覆盖掉，因此应当小心设置文件名。

```
> write.csv(a,"authors-new.csv",row.names=FALSE)  
> write.table(b,"books-new.txt",row.names=FALSE)
```



控制语句

(i) 条件控制

`if(...){...}` 是基本的条件控制语句，即若某个条件满足，就会执行一段命令，否则就不做任何事情。比如下面这段话可以判断a是不是一个正数

```
> a <- 1  
> if(a>0){  
+ print("a is positive")  
+ }  
[1] "a is positive"
```




控制语句

if 语句可以和else 语句联合使用，如果条件满足执行第一段命令，否则执行else 之后的命令。比如, 下面的语句可以找出a 和b 之间的最小值。

```
> a <- 1  
> b <- 2  
> if(a<b){  
+ m <- a  
+ } else{  
+ m <- b  
+ }  
> m  
[1] 1
```

请解读以上程序

控制语句



(ii) 循环控制

`for(... in ...){...}` 是一个循环控制语句，比如下面的语句可以打印出1,2,3 的平方

```
> for(i in 1:3){  
+ print(i^2)  
+ }
```

```
[1] 1
```

```
[1] 4
```

```
[1] 9
```



控制语句

关键字**break** 可以用于结束任何循环，关键字**next** 可以用来结束一次特定的循环，然后直接跳入下一次” 循环。

```
> for(i in 1:10){  
+ if(i<=3) next  
+ if(i>3 & i<6) cat("the square of",i,"is",i^2,"\n")  
+ if(i>=6) break  
+ }  
the square of 4 is 16  
the square of 5 is 25
```

请解读上面这段程序

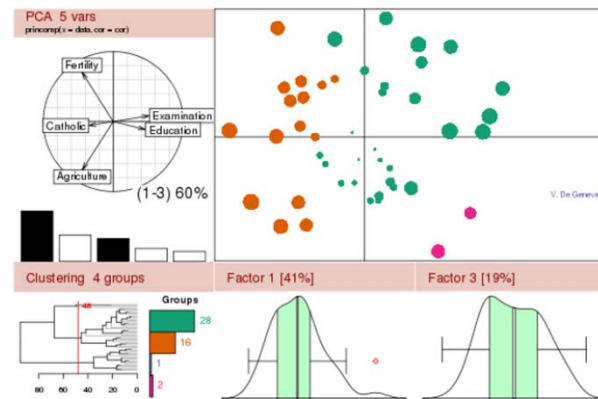
第1&2讲参考资料



西南财经大学课程讲义

R 语言入门

司亚卿



谢 谢