

# R数据对象I



课堂测试时间

- 1、数据分析包括几个过程？
- 2、简单描述R语言优点。
- 3、填写下表：

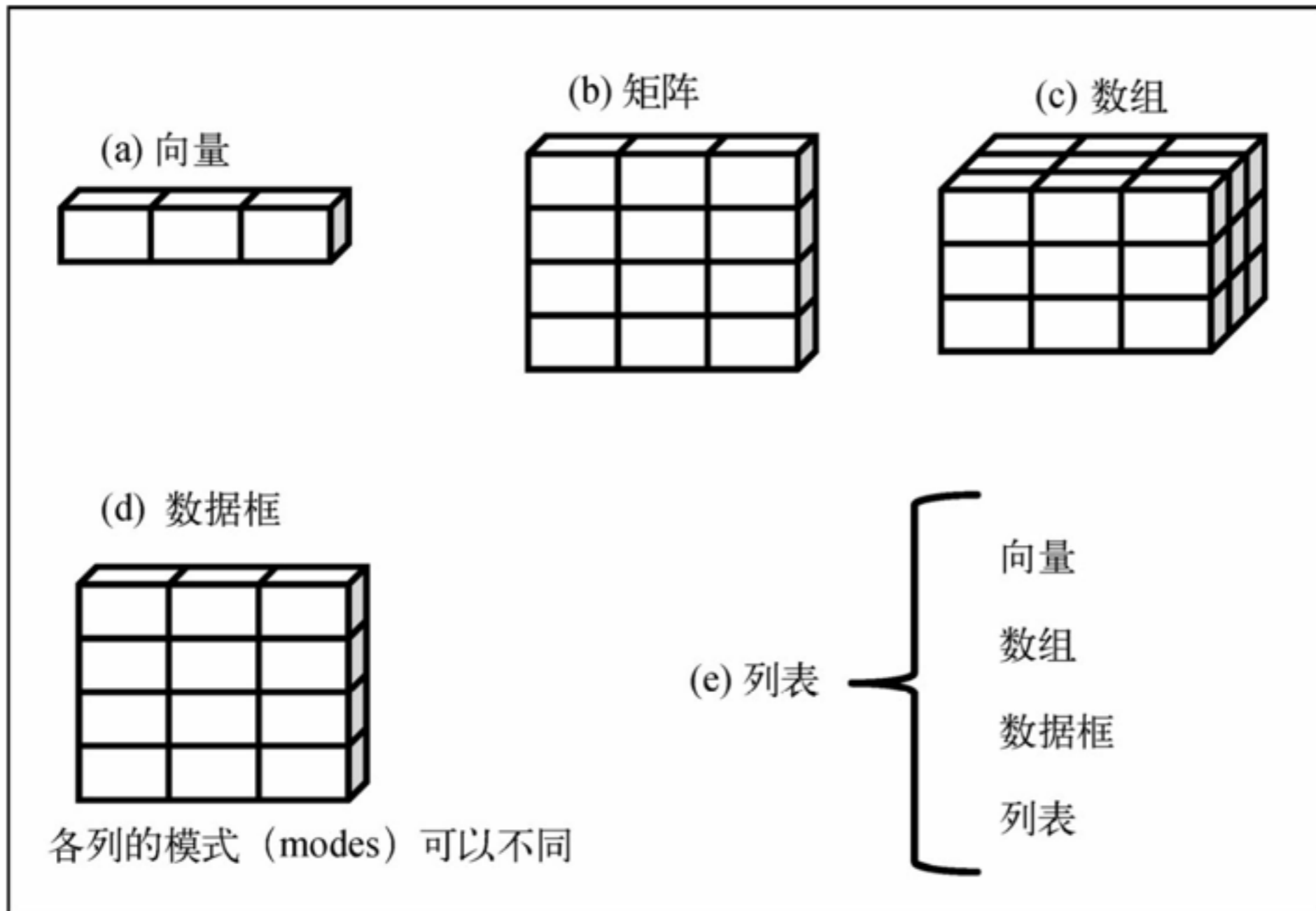
功能	函数	功能	函数
安装vcd包		使用maps包	
显示当前工作目录			demo(image)
帮助		查看函数mean()的示例	
	source()		sink()
	?c		dev.off()

- 4、写出命令，实现如下功能：创建一个从1到9的向量并赋给x，将1付给y，并写出x+y的结果。
- 5、RStudio主界面一般有几个功区？.Rprofile、.Rdata、.Rhistory文件分别有什么作用？

表2-1 病例数据

病人编号 (PatientID)	入院时间 (AdmDate)	年龄 (Age)	糖尿病类型 (Diabetes)	病情 (Status)
1	10/15/2009	25	Type1	Poor
2	11/01/2009	34	Type2	Improved
3	10/21/2009	28	Type1	Excellent
4	10/28/2009	52	Type1	Poor

- 数据集通常是由数据构成的一个矩形数组，行表示观测，列表示变量。
- R中有许多用于存储数据的结构，包括标量、向量、数组、数据框和列表。
- R可以处理的数据类型（模式）包括数值型、字符型、逻辑型（TRUE / FALSE）、复数型（虚数）和原生型（字节）。
- 在R中，对象（object）是指可以赋值给变量的任何事物，包括常量、数据结构、函数，甚至图形。



- 定义
- 提取
- 操作

图2-1 R中的数据结构

- 向量是用于存储数值型、字符型或逻辑型数据的一维数组。执行组合功能的函数`c()` 可用来创建向量。
- 标量是只含一个元素的向量，例如`f<-3`、`g <- "US"` 和`h <- TRUE`。它们用于保存常量。
- 通过在方括号中给定元素所处位置的数值，我们可以访问向量中的元素。
- 使用冒号(:)用于生成一个数值序列。

- `a <- c(1, 2, 5, 3, 6, -2, 4)` 数值类型
- `b <- c("one", "two", "three")` 字符类型
- `c <- c(TRUE, FALSE, TRUE, FALSE)` 逻辑类型

定义

- 
- `a <- c(1, 2, 5, 3, 6, -2, 4)`
  - `a[3]`
  - `a[c(1, 3, 5)]`
  - `a[2:6]`

提取

`c()`可以将不同的向量合并成一个更长的向量

```
> y <- c(1,5,2)
> z <- c(x,0,y)
> z
[1] 8.2 3.7 4.5 5.6 7.3 0.0 1.0 5.0 2.0
```

: 可以生成步长为1的等差数列 (向量)

```
> 1:5
[1] 1 2 3 4 5
> 5:1
[1] 5 4 3 2 1
> 1.2:5
[1] 1.2 2.2 3.2 4.2
> 5.2:1
[1] 5.2 4.2 3.2 2.2 1.2
```

: 运算优先于四则运算

```
> 1:9-1
[1] 0 1 2 3 4 5 6 7 8
> 1:(9-1)
[1] 1 2 3 4 5 6 7 8
```




- `seq()`函数用来生成等距间隔的数列。
- 基本形式是: `seq(from=value1, to=value2, by=value3)`, 表示从value1开始, 到value2结束, 中间间隔为value3;
- 另一个使用形式为: `seq(length=value2, from=value1, by=value3)`。

```
> seq(-2, 2, 0.5)
[1] -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0
> seq(length=9, from=-2, by=0.5)
[1] -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0
```

- `rep()`函数将一个向量重复若干次再放入新的变量。
- 使用形式为: `rep(x, times=n)`, 表示将x重复n次后构成的向量

```
> rep(2, 3)
[1] 2 2 2
> x <- 1:3
> rep(x, 3)
[1] 1 2 3 1 2 3 1 2 3
```

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



<code>+</code>	加
<code>-</code>	减
<code>*</code>	乘
<code>/</code>	除
<code>^, **</code>	求幂
<code>x %% Y</code>	求余
<code>x %/% Y</code>	整除

```
> x <- c(1,3,5)
> y <- c(2,4,6)
> x+y
[1] 3 7 11
> x-y
[1] -1 -1 -1
> x*y
[1] 2 12 30
> x/y
[1] 0.5000000 0.7500000 0.8333333
> x^2
[1] 1 9 25
> x**2
[1] 1 9 25
> x%/%y
[1] 0 0 0
> x%%y
[1] 1 3 5
> x%**y
      [,1]
[1,] 44
```

<code>&gt;,&lt;</code>	大于, 小于
<code>&lt;=, &gt;=</code>	大于等于, 小于等于
<code>!=, ==</code>	不等于, 等于
<code>!x</code>	非x
<code>x   y</code>	x或者y
<code>x &amp; y</code>	x和y
<code>isTRUE(x)</code>	x是否为TRUE

```
> x <- 1:7
> x
[1] 1 2 3 4 5 6 7
> l <- x>3
> l
[1] FALSE FALSE FALSE TRUE TRUE TRUE TRUE
> z <- c(TRUE, FALSE, F, T)
> all(c(1,2,3,4,5,6,7)>3)
[1] FALSE
> any(c(1,2,3,4,5,6,7)>3)
[1] TRUE
```

<i>length(object)</i>	显示对象中元素 / 成分的数量
<i>dim(object), str</i>	显示某个对象的维度 / 结构
<i>class(object)</i>	显示某个对象的类和类型
<i>name(object)</i>	显示某个对象中各成分的名字
<i>head(object), tail</i>	★ 列出某个对象的开始部分 / 最后部分
<i>cbind(object, object, ...), rbind</i>	按列 / 行合并对象
<i>object, ls(), rm(object,object,...), c(object,object,...)</i>	



- `length()`: 对象长度。
- `max()`: 向量中的最大值。
- `min()`: 向量中的最小值。
- `range()`: 边界
- `sum()`: 求和。
- `prod()`: 内积。

```
> x <- c(1,5,7,3)
> length(x)
[1] 4
> min(x)
[1] 1
> max(x)
[1] 7
> range(x)
[1] 1 7
> sum(x)
[1] 16
> prod(x)
[1] 105
```



- 向量排序函数包括：  
`order()`, `sort()`, `sort.list()` 等。
- `sort()` 输出排序后的向量，`order()` 和 `sort.list()` 返回下标排列。
- `which.max()` 和 `which.min()` 返回下标值。

```
> x <- c(10,6,4,7,8)
> order(x)
[1] 3 2 4 5 1
> sort(x)
[1] 4 6 7 8 10
> sort.list(x)
[1] 3 2 4 5 1
> which.min(x)
[1] 3
> which.max(x)
[1] 1
```

- 矩阵是一个二维数组，只是每个元素都拥有相同的模式（数值型、字符型或逻辑型）。可通过函数matrix 创建矩阵。

matrix()

```
myymatrix <- matrix(vector, nrow=number_of_rows, ncol=number_of_columns,  
                    byrow=logical_value, dimnames=list(  
                      char_vector_rownames, char_vector_colnames))
```

---

- 其中vector 包含了矩阵的元素，nrow 和ncol 用以指定行和列的维数，dimnames 包含了可选的以字符型向量表示的行名和列名。
- 选项byrow 则表明矩阵应当按行填充（byrow=TRUE）还是按列填充（byrow=FALSE），默认情况下按列填充。
- 以使用下标和方括号来选择矩阵中的行、列或元素。X[i,] 指矩阵X 中的第i 行，X[,j]指第j 列，X[i,j] 指第i 行第j 个元素。选择多行或多列时，下标i和j可为数值型向量。

- `y <- matrix(1:20, nrow = 5, ncol = 4)`
  - `y`
- 
- `cells <- c(1, 26, 24, 68)`
  - `rnames <- c("R1", "R2")`
  - `cnames <- c("C1", "C2")`
  - `mymatrix <- matrix(cells, nrow = 2, ncol = 2, byrow = TRUE, dimnames = list(rnames, cnames))`
  - `mymatrix`
- 
- `mymatrix <- matrix(cells, nrow = 2, ncol = 2, byrow = FALSE, dimnames = list(rnames, cnames))`
  - `mymatrix`

- `x <- matrix(1:10, nrow = 2)`

- `x`

定义

---

- `x[2, ]`

- `x[, 2]`

- `x[1, 4]`

- `x[1, c(4, 5)]`

提取

- 数组 (array) 与矩阵类似, 但是维度可以大于2。数组可通过array 函数创建。

```
myymatrix <- matrix(vector, nrow=number_of_rows, ncol=number_of_columns,  
                    byrow=logical_value, dimnames=list(  
                    char_vector_rownames, char_vector_colnames))
```

- 其中vector 包含了数组中的数据, dimensions 是一个数值型向量, 给出了各个维度下标的最大值, 而dimnames 是可选的、各维度名称标签的列表。

- `dim1 <- c("A1", "A2")`
- `dim2 <- c("B1", "B2", "B3")`
- `dim3 <- c("C1", "C2", "C3", "C4")`
- `z <- array(1:24, c(2, 3, 4), dimnames = list(dim1, dim2, dim3))`
- `z`

- 由于不同的列可以包含不同模式（数值型、字符型等）的数据，数据框的概念较矩阵来说更为一般。数据框将是你在R中最常处理的数据结构。
- 病例数据集包含了数值型和字符型数据。由于数据有多种模式，无法将此数据集放入一个矩阵。在这种情况下，使用数据框是最佳选择。
- 数据框可通过函数`data.frame()`创建  

```
mydata <- data.frame(col1, col2, col3,...)
```
- 其中的列向量`col1, col2, col3,...`可为任何类型（如字符型、数值型或逻辑型）。每一列的名称可由函数`names`指定。

- `patientID <- c(1, 2, 3, 4)`
- `age <- c(25, 34, 28, 52)`
- `diabetes <- c("Type1", "Type2", "Type1", "Type1")`
- `status <- c("Poor", "Improved", "Excellent", "Poor")`
- `patientdata <- data.frame(patientID, age, diabetes, status)`
- `patientdata`

定义

提取

- 
- `patientdata[1:2]`
  - `patientdata[c("diabetes", "status")]`
  - `patientdata$age`



```
attach(mtcars)
  summary(mpg)
  plot(mpg, disp)
  plot(mpg, wt)
detach(mtcars)
```

```
with(mtcars, {
  summary(mpg, disp, wt)
  plot(mpg, disp)
  plot(mpg, wt)
})
```

可以暂时不看  
后边需要的时候再看

```
> with(mtcars, {
  nokeepstats <- summary(mpg)
  keepstats <<- summary(mpg)
})
> nokeepstats
Error: object 'nokeepstats' not found
> keepstats
```

- 变量可归结为名义型、有序型或连续型变量
  - ✱ 名义型变量是没有顺序之分的类别变量
  - ✱ 有序型变量表示一种顺序关系，而非数量关系
  - ✱ 连续型变量可以呈现为某个范围内的任意值，并同时表示了顺序和数量
- 类别（名义型）变量和有序类别（有序型）变量在R中称为因子（factor）
- 函数factor()以一个整数向量的形式存储类别值，整数的取值范围是[1...k]（其中k是名义型变量中唯一值的个数）
- 要表示有序型变量，需要为函数factor()指定参数ordered=TRUE
- 对于字符型向量，因子的水平默认依字母顺序创建，你可以通过指定levels选项来覆盖默认排序

- `patientID <- c(1, 2, 3, 4)`
- `age <- c(25, 34, 28, 52)`
- `diabetes <- c("Type1", "Type2", "Type1", "Type1")`
- `status <- c("Poor", "Improved", "Excellent", "Poor")`
- `diabetes <- factor(diabetes)`
- `status <- factor(status, order = TRUE)`
- `patientdata <- data.frame(patientID, age, diabetes, status)`
- `str(patientdata)`                      显示对象结构
- `summary(patientdata)`                显示对象统计概要

可以  
暂时  
不看

- 列表 (list) 是R的数据类型中最为复杂的一种。一般来说, 列表就是一些对象 (或成分, component) 的有序集合。列表允许你整合若干 (可能无关的) 对象到单个对象名下
- 

- `g <- "My First List"`
- `h <- c(25, 26, 18, 39)`
- `j <- matrix(1:10, nrow = 5)`
- `k <- c("one", "two", "three")`
- `mylist <- list(title = g, ages = h, j, k)`
- `mylist`
- `mylist[[2]]`

- 对象名称中的句点 (.) 没有特殊意义。但美元符号 (\$) 却有着和其他语言中的句点类似的含义，即指定一个对象中的某些部分。例如，A\$x是指数据框A中的变量x。
- 将一个值赋给某个向量、矩阵、数组或列表中一个不存在的元素时，R将自动扩展这个数据结构以容纳新值。

```
> x <- c(8, 6, 4)
> x[7] <- 10
> x
[1] 8 6 4 NA NA NA 10
```

- R中没有标量。标量以单元素向量的形式现。
- R中的下标不从0开始，而从1开始。在上述向量中，x[1]的值为8。
- 变量无法被声明。它们在首次被赋值时生成。

向量	<i>c(..., recursive=FALSE),</i>
矩阵	<i>matrix(vector, nrow, ncol, byrow=FALSE, dimnames)</i>
数组	<i>array(vector, dimensions, dimnames)</i>
数据框	<i>data.frame(col1, col2,col3,...), \$</i>
列表	<i>list(object1, object2,...)</i>
类型判断设置 ★	<i>is.numeric(), is.integer(), is.logical(), is.character()</i> <i>as.numeric(), as.integer(), as.logical(), as.character()</i>
<i>attach(), detach(),with(), factor(), rep(x,n), mode()</i>	



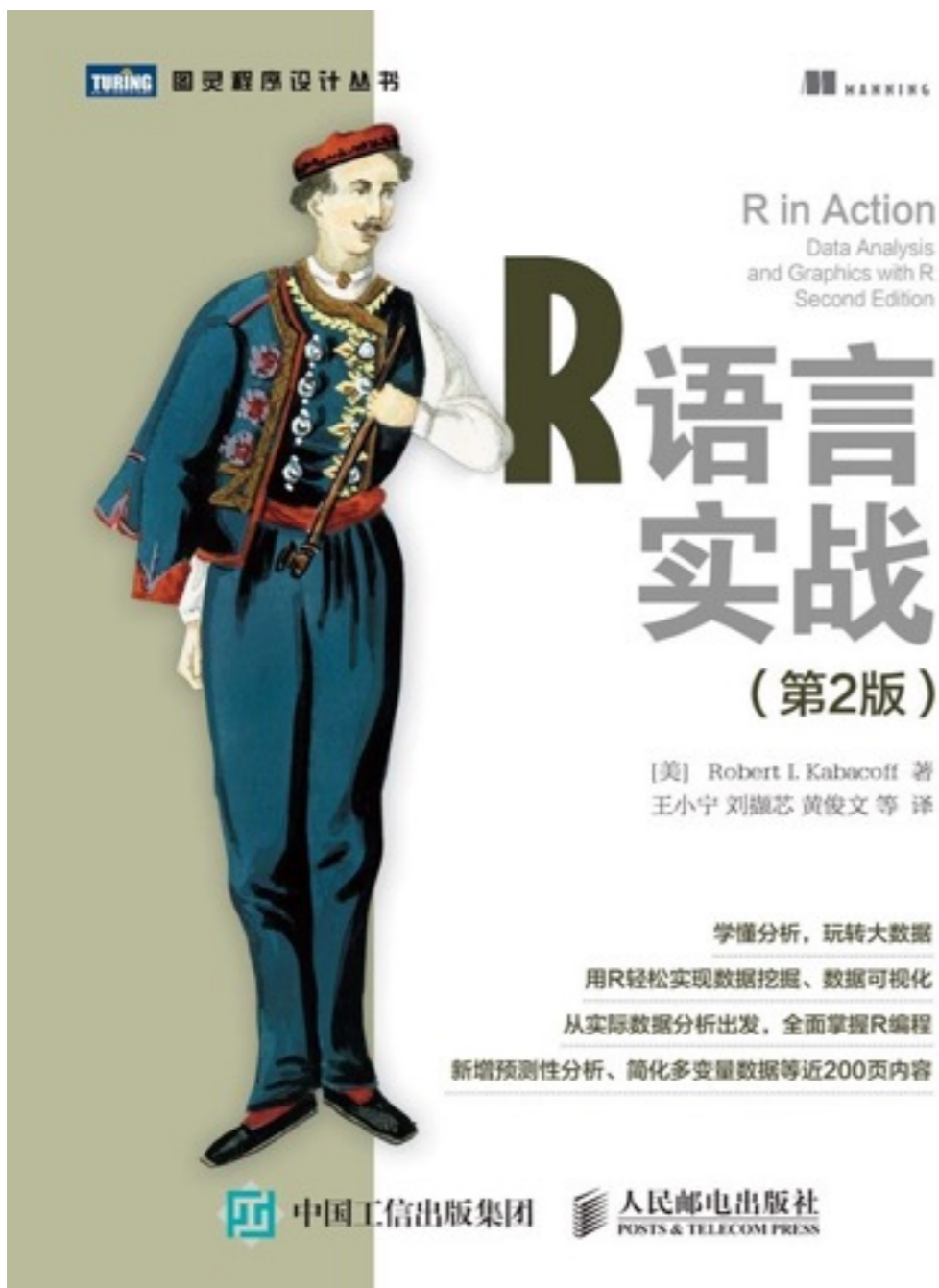
# 提问时间!

孙惠平

[sunhp@ss.pku.edu.cn](mailto:sunhp@ss.pku.edu.cn)

练习





第二章



第一章到第四章

The screenshot shows the DataCamp website interface. At the top, there is a navigation bar with the DataCamp logo and the tagline 'We're hiring!'. The navigation menu includes 'Dashboard', 'Courses', 'Pricing', 'Business', 'Community', a notification bell, and a user profile 'sunhp@ss.pk...'. Below the navigation bar is a search bar with a magnifying glass icon and a right arrow. To the right of the search bar are two dropdown menus: 'All Technologies' and 'All Topics'. The main content area displays a grid of course cards. The first card, 'Introduction to R', is highlighted with a red border. It features a blue header, a description: 'Master the basics of data analysis by manipulating common data structures such as vectors, matrices and data frames.', a circular profile picture of Jonathan Cornelissen, and his name and title: 'JONATHAN CORNELISSEN, Co-founder and CEO of DataCamp'. Below the name is a small circular icon. The other cards in the grid include 'Intro to Python for Data Science' by Filip Schouwenaars, 'Intermediate R' by Filip Schouwenaars, 'Intermediate Python for Data Science', 'Data Visualization with ggplot2 (Part 1)', and 'Introduction to Machine Learning'.

- 注册一个账号
- 完成Introduction to R课程
- 微信提交一个PDF，包括自己账户页面和课程页面

The screenshot shows the progress of two chapters in the 'Intro to basics' course. Both chapters are marked as 100% complete. The first chapter, 'Intro to basics', includes a description of learning to use the console and assign variables. The second chapter, 'Vectors', describes learning to analyze gambling results using vectors. Each chapter entry has a 'Continue Chapter' button and a link to 'Hide Chapter Details' or 'View Chapter Details'.

**1 Intro to basics** 100%

In this chapter, you will take your first steps with R. You will learn how to use the console as a calculator and how to assign variables. You will also get to know the basic data types in R. Let's get started!

[Hide Chapter Details](#) [Continue Chapter](#)

**2 Vectors** 100%

In this free R course, we'll take you on a trip to Vegas, where you will learn how to analyze your gambling results using vectors in R! After completing this chapter, you will be able to create vectors in R, name them, select elements from them and compare different vectors.

[View Chapter Details](#) [Continue Chapter](#)

The screenshot shows the user profile for 'sunhp'. It features a profile picture, the username 'sunhp', and statistics: 3100 XP Earned, 0 Courses Completed, and 32 Exercises Aced. Below this, there is a section for 'sunhp's Skills' with progress bars for 'Programming' (3000/48450 XP) and 'Importing & Cleaning Data' (100/34750 XP).

**sunhp**

3100 XP Earned 0 Courses Completed 32 Exercises Aced

**sunhp's Skills**

**Programming**  
3000/48450 XP

**Importing & Cleaning Data**  
100/34750 XP

# R数据对象II

- 数据结构定义: `c()`; `matrix()`; `array()`; `data.frame()`; `factor()`; `list()`;
- 数据结构访问: 下标; 下标向量; 逻辑向量; 负下标;
- 向量: `:`; `seq()`; `rep()`;
- 算术运算符: `+`; `-`; `*`; `/`; `**`; `^`; `%%`; `%/`;
- 逻辑运算: `>`; `<`; `>=`; `<=`; `==`; `!=`; `!`; `|`; `&`; `isTRUE()`; `identical()`; `any()`; `all()`;
- 属性函数: `length()`; `dim()`; `class()`; `names()`; `head()`; `tail()`;
- 排序函数: `order()`; `sort()`; `sort.list()`; `which()`; `which.max()`; `which.min()`;
- 运算函数: `max()`; `min()`; `range()`; `sum()`; `prod()`; `sqrt()`; `abs()`;
- 类型函数: `is.numeric()`; `is.integer()`; `is.logical()`; `is.character()`; `as.xxxx()`;
- 其余函数: `attach()`; `detach()`; `with()`; `$`; `t()`; `diag()`; `solve()`; `eigen()`;

- 矩阵运算
- 缺失值处理
- 类型转换
- 数据集合并
- 字符处理
- 日期和时间
- apply函数
- 统计函数

<code>t()</code>	矩阵转置
<code>det()</code>	求方阵行列式的值
<code>crossprod(x,y)</code>	x和y的内积( <code>%*%</code> )
<code>tcrossprod(x,y)</code>	x和y的外积( <code>%o%</code> ), <code>outer()</code>
<code>diag()</code>	生成对角阵和矩阵取对角运算
<code>solve()</code>	解线性方程组, 求矩阵的逆
<code>eigen()</code>	求矩阵的特征值和特征向量

```
> A <- matrix(1:6, nrow = 2)
```

```
> A
```

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

```
> t(A)
```

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

```
> x <- 1:5
```

```
> y <- 2*1:5
```

```
> x %*% y
```

```
      [,1]
[1,]  110
```

```
> crossprod(x,y)
```

```
      [,1]
[1,]  110
```

```
> x %o% y
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,]    2    4    6    8   10
[2,]    4    8   12   16   20
[3,]    6   12   18   24   30
[4,]    8   16   24   32   40
[5,]   10   20   30   40   50
```

```
> tcrossprod(x,y)
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,]    2    4    6    8   10
[2,]    4    8   12   16   20
[3,]    6   12   18   24   30
[4,]    8   16   24   32   40
[5,]   10   20   30   40   50
```

```
> outer(x,y,FUN = "*")
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,]    2    4    6    8   10
[2,]    4    8   12   16   20
[3,]    6   12   18   24   30
[4,]    8   16   24   32   40
[5,]   10   20   30   40   50
```

```
> det(matrix(1:4,ncol = 2))
```

```
[1] -2
```



```
> A <- array(1:9,dim=c(3,3))
> B <- array(1:9,dim=c(3,3))
> C <- A * B
> C
```

```
      [,1] [,2] [,3]
[1,]    1   16   49
[2,]    4   25   64
[3,]    9   36   81
```

```
> D <- A %*% B
> D
```

```
      [,1] [,2] [,3]
[1,]   30   66  102
[2,]   36   81  126
[3,]   42   96  150
```

```
> M <- array(1:9, dim=c(3,3))
> M
```

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

```
> diag(M)
[1] 1 5 9
```

```
> A <- t(array(c(1:8,10), dim = c(3,3)))
```

```
> b <- c(1,1,1)
> x <- solve(A,b)
```

```
> x
[1] -1.000000e+00  1.000000e+00  3.330669e-16
```

```
> B <- solve(A)
```

```
> B
      [,1]      [,2] [,3]
[1,] -0.6666667 -1.333333  1
[2,] -0.6666667  3.666667 -2
[3,]  1.0000000 -2.000000  1
```

```
> Sm <- crossprod(A,A)
```

```
> ev <- eigen(Sm)
```

```
> ev
```

```
$values
```

```
[1] 303.19533618  0.76590739  0.03875643
```

```
$vectors
```

```
      [,1]      [,2]      [,3]
[1,] -0.4646675  0.833286355  0.2995295
[2,] -0.5537546 -0.009499485 -0.8326258
[3,] -0.6909703 -0.552759994  0.4658502
```

```
> A
```

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

表4-1 领导行为的性别差异

经理人	日期	国籍	性别	年龄	q1	q2	q3	q4	q5
1	10/24/08	US	M	32	5	4	5	5	5
2	10/28/08	US	F	45	3	5	2	5	5
3	10/01/08	UK	F	25	3	5	5	5	2
4	10/12/08	UK	M	39	3	3	4		
5	05/01/09	UK	F	99	2	2	1	2	1

- 例子见教材74页
- **NA**
- **is.na()**
- **na.rm = TRUE**
- **na.omit()**

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

```
> sum(1:5, NA)
[1] NA
> sum(1:5, NA, na.rm = TRUE)
[1] 15
```

看：例子4-3和4-4

表4-5 类型转换函数

判 断	转 换
<code>is.numeric()</code>	<code>as.numeric()</code>
<code>is.character()</code>	<code>as.character()</code>
<code>is.vector()</code>	<code>as.vector()</code>
<code>is.matrix()</code>	<code>as.matrix()</code>
<code>is.data.frame()</code>	<code>as.data.frame()</code>
<code>is.factor()</code>	<code>as.factor()</code>
<code>is.logical()</code>	<code>as.logical()</code>

- 见教材78页

看：例子4-5

```
> x1 <- rbind(c(1,2),c(3,4))
> x1
      [,1] [,2]
[1,]    1    2
[2,]    3    4
> x2 <- 10 + x1
> x3 <- cbind(x1, x2)
> x3
      [,1] [,2] [,3] [,4]
[1,]    1    2   11   12
[2,]    3    4   13   14
> x4 <- rbind(x1,x2)
> x4
      [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]   11   12
[4,]   13   14
> cbind(1, x1)
      [,1] [,2] [,3]
[1,]    1    1    2
[2,]    1    3    4
```

- 见教材93页

<code>nchar()</code>	计算x中的字符数
<code>substr(s,start,stop)</code>	提取或替换一个字符向量中的子串
<code>strsplit(x,split)</code>	在split处分割字符向量x中的元素
<code>toupper(x), tolower()</code>	大小写转换
<code>paste(..., sep="")</code>	连接字符串
<code>grep(pattern,x,ignore.case=FALSE,fixed=FLASE)</code>	搜索 
<code>sub(pattern,replacement,x,ignore.case=FALSE,fixed=FLASE)</code>	搜索替换 

```
> paste("My", "Job")
[1] "My Job"
>
> labs <- paste("X", 1:6, sep="")
> labs
[1] "X1" "X2" "X3" "X4" "X5" "X6"
>
> paste("Today is", date())
[1] "Today is Wed Mar  2 12:41:21 2016"
>
> paste(c("a", "b"), collapse=".")
[1] "a.b"
```

- 见教材76页

日期函数	<i>as.Date(x, "input_format")</i>
<i>%d</i>	数字表示的日期 (0-31)
<i>%a, %A</i>	星期名 (缩写, 非缩写)
<i>%m</i>	月份 (0-12)
<i>%b, %B</i>	月份 (缩写, 非缩写)
<i>%y, %Y</i>	年份 (两位, 四位)
<i>Sys.Date(), date(), difftime(), format()</i>	

```
> mydates <- as.Date(c("2007-06-22"))
> mydates
[1] "2007-06-22"
> mydates <- as.Date(c("2007-06-22"))
>
> strDates <- c("01/05/1965")
> dates <- as.Date(strDates, "%m/%d/%Y")
>
> Sys.Date()
[1] "2016-03-02"
> date()
[1] "Wed Mar  2 12:48:52 2016"
```



```
> today <- Sys.Date()
> format(today, format = "%B %d %Y")
[1] "March 02 2016"
> format(today, format = "%A")
[1] "Wednesday"
>
> startdate <- as.Date("2004-02-13")
> enddate <- as.Date("2009-06-22")
> days <- enddate - startdate
>
> today <- Sys.Date()
> format(today, format = "%B %d %Y")
[1] "March 02 2016"
> dob <- as.Date("1956-10-10")
> format(dob, format = "%A")
[1] "Wednesday"
> difftime(today, dob, units="weeks")
Time difference of 3099 weeks
```

apply(x, MARGIN, FUN, ...)

```
> a <- matrix(1:6,nrow=2)
> a
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> apply(a, 1,sum)
[1]  9 12
> apply(a,2,sum)
[1]  3  7 11
```

- 见教材95页

看：例子5-5和5-6

```
> mydata <- matrix(rnorm(30), nrow=6)
> mydata
      [,1]      [,2]      [,3]      [,4]      [,5]
[1,]  1.1039131 -0.6779796  0.09072753  0.6943354 -0.68360455
[2,] -1.2876154  0.1540778  1.41431948 -0.9622685  2.07486216
[3,] -0.4221483  0.3073955  0.36975022 -0.4124088  0.54614267
[4,] -0.5283792 -0.7510899 -1.58224514  1.1124982  0.24044145
[5,] -1.1322217  1.0616374  0.37744029  0.1879165 -0.03192165
[6,]  0.7633084  0.6153539  0.58158158  0.3485943  0.17747101
> apply(mydata, 1, mean)
[1]  0.10547839  0.27867512  0.07774626 -0.30175492  0.09257018  0.49726184
> apply(mydata, 2, mean)
[1] -0.2505238  0.1182325  0.2085957  0.1614445  0.3872318
> apply(mydata, 2, mean, trim=.4)
[1] -0.4752638  0.2307367  0.3735953  0.2682554  0.2089562
```

- 见教材87页

<i>mean(x), median(x)</i>		平均数, 中位数
<i>sd(x), var(x)</i>		标准差, 方差
<i>max(x), min(x)</i>		最大值, 最小值
<i>range(x), sum(x)</i>	★	值域, 求和
<i>quantile(x, prob)</i>	★	求分位数
<i>diff(x, lag=n)</i>	★	滞后差分
<i>scale(x, center=TRUE, scale=TRUE)</i>	★	为数据对象x按列进行中心化和标准化
<i>str(), summary()</i>		

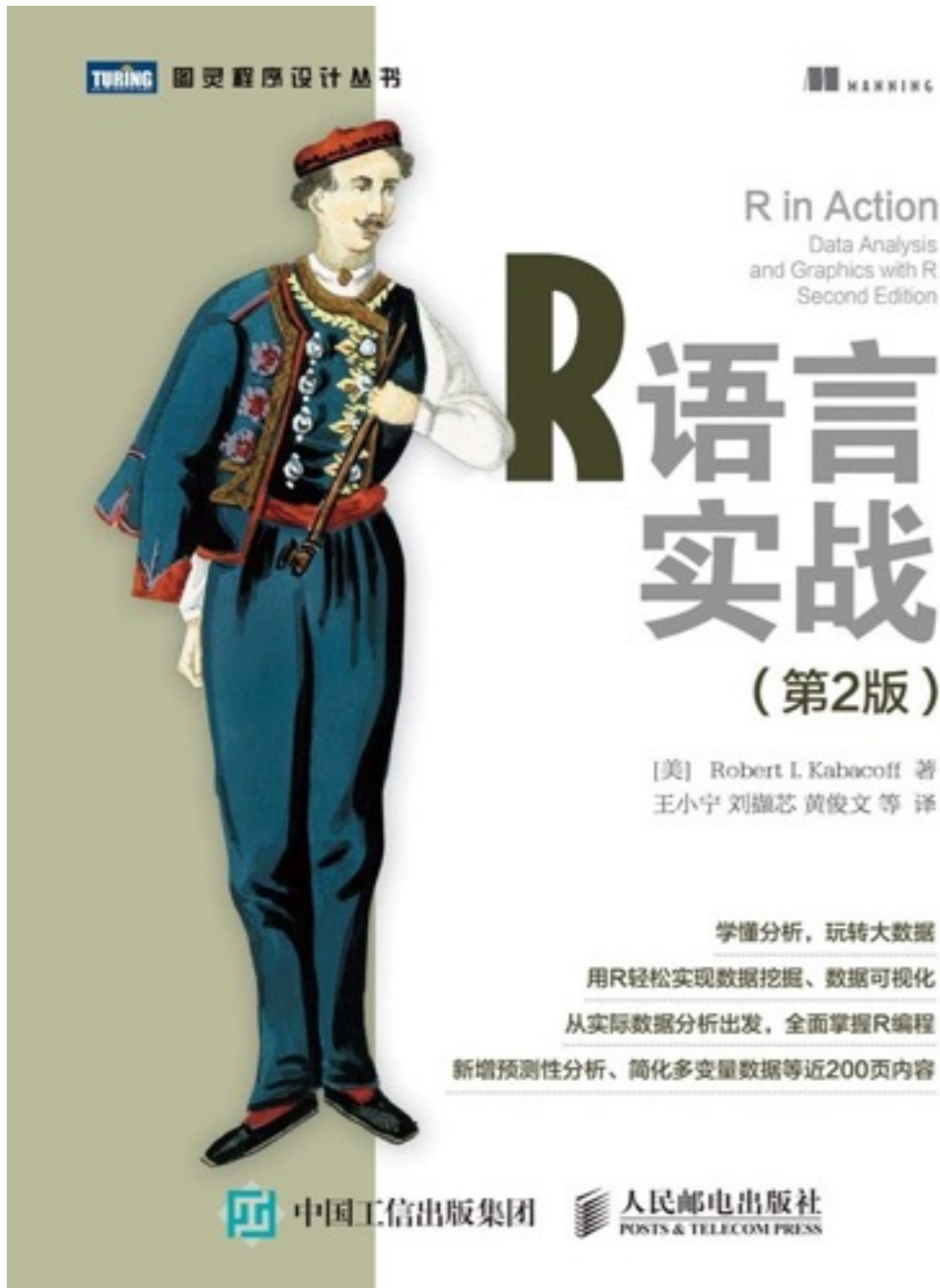
看: 例子5-1

# 提问时间!

孙惠平

[sunhp@ss.pku.edu.cn](mailto:sunhp@ss.pku.edu.cn)

练习



第四章和第五章



第五章、第六章、第七章

谢谢!

孙惠平

[sunhp@ss.pku.edu.cn](mailto:sunhp@ss.pku.edu.cn)