第4章 数据结构与数据处理

内容导航

CONTENTS

▸▸▸ 用seq()创建向量

- seq()函数的一般格式

```
seq (from = 1, to = 10, by = ((to - from)/(length.out - 1)),
        length.out = NULL)
> seq (0, 1, length.out = 11)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> seq( from = 0, to = 1, by = 0.1, length.out = 11)
Error in seq.default(from = 0, to = 1, by = 0.1, length.out = 11) :
  太多参数
```

- from和to指定起始和结束数字，by指定步长，length.out指定输出向量的长度
- 同时指定from、to、by和length.out会报错，即使给定参数在数学上是完美的

▸▸▸ 用rep()创建向量

- rep()函数的一般格式

rep (x, times)

```
> x <- rep (3, 3)                #3重复3次
> rep (x, 3)                     #向量x重复3次
[1] 3 3 3 3 3 3 3 3 3
> rep (1:3, each = 2)            #注意是元素重复，不是向量重复
[1] 1 1 2 2 3 3
```

- rep(x, times)，把向量x重复times次组成新的向量。

►►► 使用正数索引访问向量元素

```
> my_vec <- 1:10
> my_vec
 [1]  1  2  3  4  5  6  7  8  9 10
> my_vec[c(1, 5, 7, 19)] #取给定索引的元素，超出范围会导致NA
[1]  1  5  7 NA
> my_vec[3.14]
[1] 3
```

- 使用冒号：构造连续的整数向量
- 索引也可以是向量
- 取超出向量索引的元素会得到NA
- 使用浮点数索引会直接截取整数部分作为索引值

▸▸▸ 使用负数索引删除向量元素

```
> my_vec <- 2:11  #初始化为2到11的连续整数向量
> my_vec[c(-8, -9, -10)]  #不显示索引为8，9，10的元素
[1] 2 3 4 5 6 7 8
> my_vec[-3:-1]  #元素9,10,11被保留，原数组my_vec未被改动
[1]  5  6  7  8  9 10 11
> my_vec[c(1, -2)]  #注意不要正负混用
Error in my_vec[c(1, -2)] :
  only 0's may be mixed with negative subscripts
```

- R中除引用类型外的对象，在修改时都会在内存中拷贝一个完整的对象进行修改，不会影响原对象的值。
- 要修改my_vec向量的元素，可使用赋值的方法。

▸▸▸ 使用逻辑型索引访问向量元素

```
> my_vec <- 1:5; my_vec
[1] 1 2 3 4 5
> my_vec[c(T, T, F, T, F)]
[1] 1 2 4
> my_vec[10] <- 10; my_vec
 [1]  1  2  3  4  5 NA NA NA NA 10
> length(my_vec)
[1] 10
```

- 使用逻辑型索引会提取向量中索引为TRUE的元素组成新的向量。
- 给超出索引范围的元素赋值会使用NA自动填充空余的元素，同时向量长度也会增加

▸▸▸ 往向量中插入元素

```
> my_vec <- 1:6
> my_vec <- c (my_vec[1:3], 3.5, my_vec[4:length(my_vec)])
> my_vec
[1] 1.0 2.0 3.0 3.5 4.0 5.0 6.0
> append (my_vec, 'a', after = 3)
[1] "1"   "2"   "3"   "a"   "3.5" "4"   "5"   "6"
```

- 可使用append函数来完成，也可使用索引和c()函数来手动实现。
- 浮点数插入整型向量会将向量转换成浮点型。
- 字符元素插入浮点型向量将向量转换成字符型。

```
> c (1, 2, 3) + c (1, 2, 3, 4, 5, 6, 7)  #两个向量长度不同，如何相加?
[1] 2 4 6 5 7 9 8
Warning message:
In c(1, 2, 3) + c(1, 2, 3, 4, 5, 6, 7) :
  longer object length is not a multiple of shorter object length
# 上面的操作等价于下面的语句。
> c (1, 2, 3, 1, 2, 3, 1) + c (1, 2, 3, 4, 5, 6, 7)
[1] 2 4 6 5 7 9 8
```

- 若长向量长度是短向量整数倍，就不会得到警告信息。

```
> c(1, 2, 3) == c(1, 3, 2)
[1]  TRUE FALSE FALSE
> c(1, 2, 3) > c(2, 1, 3, 3, 2, 1)
[1] FALSE  TRUE FALSE FALSE FALSE  TRUE
> v1 <- c(1, 2, 3); v2 <- c(1, 3, 2); v3 <- c(1, 2, 3)
> identical(v1, v2); identical(v1, v3)                    #判断全等关系
[1] FALSE
[1] TRUE
```

- R中的比较运算符号会将两向量所有索引相等的元素各作一次比较然后输出结果向量。
- 比较运算同样依照循环补齐原则
- 判断数学意义上的相等关系，需要使用identical()函数

```
> 0.9 + 0.2 == 1.1; 1.1 - 0.2 == 0.9
[1] TRUE
[1] FALSE
> identical(1.1 - 0.2, 0.9)
[1] FALSE
> all.equal(v1, v3); all.equal(v1, v2)
[1] TRUE
[1] "Mean relative difference: 0.4"
```

- 计算机处理double型数据会产生误差，使用identical()函数或==运算符可能会得到意想不到的结果
- all.equal ()函数可避免计算误差导致的异常结果，它还返回比较对象之间差异的描述。

►►► 向量的比较

```
> v1 <- 1:5; v2 <- 3:7
> v1 > 3
[1] FALSE FALSE FALSE TRUE  TRUE
> # v1中是否有大于3的元素；v2中元素是否全小于7
> any(v1 > 3); all(v2 < 7)
[1] TRUE
[1] FALSE
```

- any ()和all ()函数分别指出其参数向量是否至少有一个或全部为TRUE。

▸▸▸ 按条件提取向量元素

```
> v1 <- -3:3; v1
[1] -3 -2 -1  0  1  2  3
> v2 <- v1[v1 * v1 > 5]; v2
[1] -3  3

> v1 * v1 > 5
[1]  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE
```

- 运算v1 * v1不是矩阵运算。R语言中的“*”号将向量中对应位置的元素相乘组成新向量。
- v1 * v1 > 5得到的逻辑型向量作为v1的索引值得到新向量v2。

内容导航

CONTENTS

- 矩阵（matrix）是一种特殊的向量，矩阵包含两个附加的属性：行数和列数。
- 矩阵内的元素必须属于同一种基本的数据类型，所以矩阵也有类型的概念。
- 矩阵是维度限定为2的数组（array）

```
> y <- 1:10
> dim (y) <- c (2, 5); y
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> class (y)
[1] "matrix"
```

- 给向量添加维度属性来创建矩阵。

- 使用matrix()函数创建矩阵

```
matrix (data = NA, nrow = 1, ncol = 1, byrow = FALSE,
        dimnames = NULL)
```

```
> matrix (c(1,2,3, 11,12,13), nrow = 2, byrow = TRUE,
+         dimnames = list (c ("row1", "row2"), c ("C.1", "C.2", "C.3")))
     C.1  C.2 C.3
row1   1    2   3
row2  11   12  13
```

- 指定行或列维度，R自动计算另一个维度。
- byrow默认为FALSE，表示元素按列依次填充。
- dimnames给矩阵指定行和列名。

```
> mat1 <- rbind (A = 1:3, B = 4:6); mat1
  [,1] [,2] [,3]
A    1    2    3
B    4    5    6
> mat2 <- cbind (mat1, cbind(c(11, 12), c(13, 14))); mat2
  [,1] [,2] [,3] [,4] [,5]
A    1    2    3   11   13
B    4    5    6   12   14
```

- **rbind()和cbind()分别可以按行和按列连接两个矩阵。**

```
> mat1 <- matrix(c(1:6), nrow = 3)
> mat2 <- matrix(c(11:16), nrow = 2)
> mat1 %*% mat2
     [,1] [,2] [,3]
[1,]   59   69   79
[2,]   82   96  110
[3,]  105  123  141
> mat1 * mat2
Error in mat1 * mat2 : non-conformable arrays
```

- 矩阵乘法使用%*%，单独使用*符号会得到矩阵对应索引元素依次相乘的结果。

```
> mat1
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> c(1, 2) * mat1
     [,1] [,2]
[1,]    1    8
[2,]    4    5
[3,]    3   12
```

- 向量与矩阵使用符号*进行运算，会将向量按列循环补齐至与矩阵同维度再进行元素乘法运算。

```
> mat3 <- matrix(1:4, 2); mat3
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> solve(mat3) %*% mat3
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

- 矩阵求逆

```
> mat <- matrix(c(1, 3 ,6, -3, -5, -6, 3, 3, 4), nrow = 3)
> mat.eig <- eigen(mat)
> mat %*% mat.eig$vectors[, 1]
          [,1]
[1,] -1.632993
[2,] -1.632993
[3,] -3.265986
> mat.eig$values[1] * mat.eig$vectors[, 1]
[1] -1.632993 -1.632993 -3.265986
```

- 矩阵的特征值与特征向量。
- Ax=λx
- t(mat) #矩阵转置
- det(mat) #求方阵行列式
- svd(mat) # 矩阵奇异值分解

- 基本用法：apply(X, MARGIN, fun)

```
> mat <- matrix(1:9, nrow = 3); mat
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> apply(mat, 1, sum); apply(mat, 2, mean)
[1] 12 15 18
[1] 2 5 8
```

- **apply()会把一个函数同时作用于一个矩阵中的一个维度，然后把返回值存储在一个向量中。**
- MARGIN是维度编号，取值为1表示对每一行应用函数，取值为2则表示对每一列应用函数。
- 对矩阵mat，按行求和；按列求均值

```
> f <- function(x) x – 1
> mat <- matrix(1:6, 3); > f(mat)
     [,1] [,2]
[1,]    0    3
[2,]    1    4
[3,]    2    5
> apply (mat, 1, f)
     [,1] [,2] [,3]
[1,]    0    1    2
[2,]    3    4    5
```

- 自定义apply()函数中的fun参数。
- 注意本例中使用apply ()函数后得到的矩阵与原始矩阵维度不一致。
- 若传入apply中的fun函数的返回值是一个含有n个元素的向量，那么apply ()函数执行的结果就有n行。

- 基本用法：

  array (data = NA, dim = length(data), dimnames = NULL)

```
, , semester one

    Math Chemistry Physics
Tom    1         3       5
Bob    2         4       6

, , semester two

    Math Chemistry Physics
Tom    7         9      11
Bob    8        10      12
```

- 数组结构与矩阵类似，但其维度可以大于2。

内容导航

CONTENTS

- 数据框与矩阵有些相似，但数据框中允许不同的列包含不同类型（数值型、字符型等）的数据。
- 数据框与数据库中的表十分相似，由一系列等长的向量组成。
- 数据框也被称为“数据矩阵”或“数据集”。

```
> names <- c('Tom', 'Ross', 'Jerry')
> ages <- c(19, 18, 20)

> df <- data.frame(names, ages, stringsAsFactors = F);df
  names ages
1   Tom   19
2  Ross   18
3 Jerry   20
```

- 使用字符串向量创建数据框时，会被自动转换成因子。因子将在下一节详述。
- 指定参数**stringsAsFactors为FALSE可阻止自动转换。**

```
> cbind (df, weight = c(70, 73, 60))        #给数据框添加一列
  names ages weight
1   Tom   19     70
2  Ross   18     73
3 Jerry   20     60
> str(rbind(df, list('Bob', '23')))
'data.frame':     4 obs. of  2 variables:
 $ names: chr  "Tom" "Ross" "Jerry" "Bob"
 $ ages : chr  "19" "18" "20" "23"
```

- 与矩阵类似，我们亦可以使用rbind ()和cbind ()函数来合成数据框。
- str()函数查看数据框结构信息。

```
> merge(df, df2)
  names ages height
1   Jerry   20    170
2   Ross    18    180
3   Tom     19    176
```

- merge ()函数能将两数据框进行合并得到新的数据框。

```
> df[2,]
  names ages
2  Ross   18
> df['ages']
  names
1    19
2    18
3    20
> df$ages
[1] 19 18 20
```

- 用索引值或列名来访问数据框中的元素。
- 使用df['ages']得到的结果是数据框。使用mode()查看其类型为list。
- 使用df$ages得到的结果是向量。

▸▸▸ 访问数据框中的元素

```
> # 修改数据框的行名和列名
> row.names(df) <- c('r1', 'r2', 'r3'); df
     names ages
r1     Tom   19
r2    Ross   18
r3   Jerry   20
> colnames(df)
          #显示列名
[1] "names" "ages"
```

- 一般要访问数据框中的行需要用索引，而不能使用df['rowname']和$。

▸▸▸ 使用SQL语句查询数据框

```
> library (sqldf)
> df
   names ages
r1   Tom   19
r2  Ross   18
r3 Jerry   20
> sqldf("select * from df where ages > 18")
  names ages
1   Tom   19
2 Jerry   20
```

- R语言中的数据框与关系型数据库中的表很相似，可用SQL语句对数据框进行一些操作。

内容导航

CONTENTS

```
> data <- c('East', 'West', 'North', 'West', 'South', 'East')
> typeof (data)
[1] "character"
> fac_data <- factor(data); fac_data    #转换成因子型
[1] East  West  North West  South East
Levels: East North South West
> as.numeric(fac_data)  #以数值型来显示
[1] 1 4 2 4 3 1
```

- 使用factor()函数将向量作为输入来创建因子。
- 将因子转换为数值型向量后得到的是向量元素对应的因子水平的编码。这意味着因子中的数据已经重新编码并存储为水平的序号。

```
> #手动改变因子对象的水平
> levels(fac_data) <- c('East', 'South', 'West', 'North'); fac_data
[1] East  North South North West  East
Levels: East South West North
> as.numeric(fac_data)
[1] 1 4 2 4 3 1
```

- 手动改变因子变量的levels后，因子变量的打印输出信息发生变化，但转换成数值型向量后的结果未发生变化。

```
> fac_data[length(fac_data) + 1] <- "southeast"
Warning message:
In `[<-.factor`(`*tmp*`, length(fac_data) + 1, value = "southeast") :
  invalid factor level, NA generated
> fac_data
[1] East  North South North West  East  <NA>
Levels: East South West North
> summary(fac_data)
 East South  West North  NA's
    2     1     1     2     1
```

- 向因子中添加一个不存在于水平中的元素会产生空值。
- summary函数能求出因子各水平出现的频率。

内容导航

CONTENTS

```
> list_data <- list (
+ "It is a string of a list!",
+ num_vec1 = c(1, 2, 1),
+ num_vec2 = c(3, 2, 1),
+ df = data.frame(name = c('zhang_xiao_hong',
'zhang_xiao_huang'), age = c(11, 12)),
+ fun = function(v1, v2) return(v1 + v2))
> list_data$fun(list_data[[2]], list_data$num_vec2)
[1] 4 4 2
```

- 列表是一种更灵活的数据结构，允许其中存在不同类型的对象，甚至是函数对象。
- 通过'$'符号和列表项的名称标签访问列表项目，也可使用索引值。

内容导航

CONTENTS

- 例如，当前工作路径下有一名为data.txt的文本文件，内容如下：

```
names       ages    Gender
Alice       18      Female
Lucy        19      Female
Tim         20      Male
```

```
> stu_info <- read.table ("data.txt", header = TRUE); stu_info
   names  ages   Gender
1    Alice    18    Female
2    Lucy     19    Female
3     Tim     20      Male
```

- read.table()函数将文件中的数据集读入数据框。

```
> read.csv
function (file, header = TRUE, sep = ",", quote = "\"", dec = ".",
    fill = TRUE, comment.char = "", ...)
read.table(file = file, header = header, sep = sep, quote = quote,
    dec = dec, fill = fill, comment.char = comment.char, ...)
<bytecode: 0x0000000003ae90a8>
<environment: namespace:utils>
```

- read.table()函数中的sep参数指定数据文件中分隔符。
- csv格式文件的一个广泛应用是在应用程序之间转移数据。

```
> stu_info
 names  ages   Gender
1     Alice     18    Female
2     Lucy      19    Female
3      Tim      20      Male
> write.table (stu_info, file = "saveData.txt");
"names" "ages" "Gender"
"1" "Alice" 18 "Female"
"2" "Lucy" 19 "Female"
"3" "Tim" 20 "Male"
```

- write.table将数据写入到文件，file参数指定文件名。

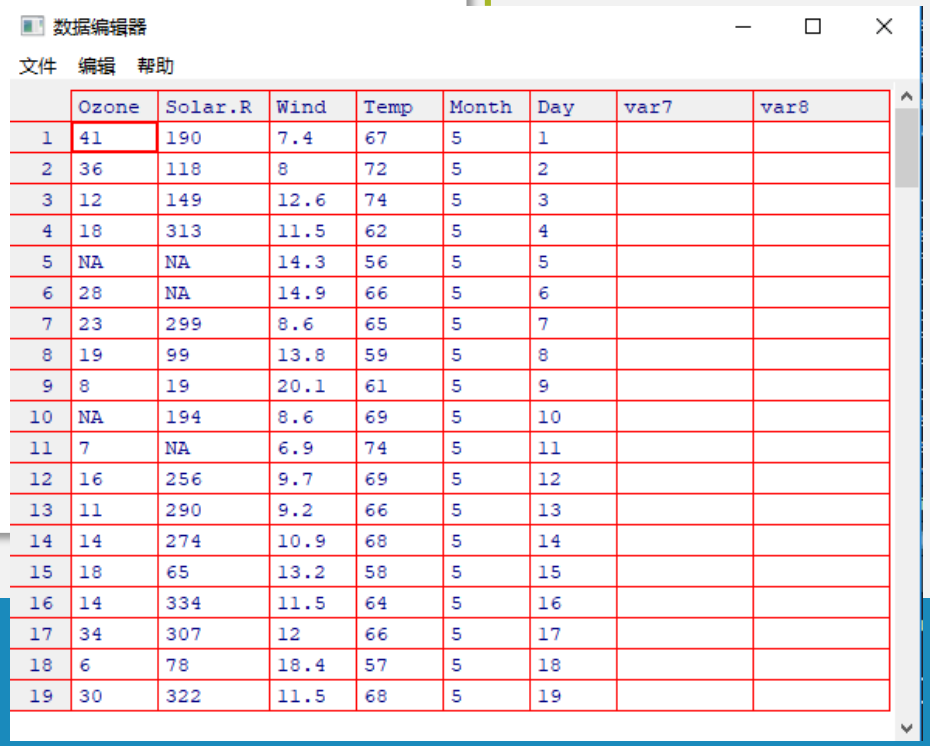
```
> library(rio) # 载入rio包，若未安装则需先安装
> library(datasets) # mtcars数据集在datasets包中
> export (mtcars, "mtcars.csv")
> convert ("mtcars.csv", "mtcars.json")
> import("mtcars.json")
> unlink("mtcars.json") # 删除mtcars.json文件
```

- rio包中的export函数将数据集写入到文件。R对象→文件。
- rio包中的convert函数将csv格式文件转换为json格式文件。文件→文件。
- unlink是base包中函数。

```
> aq <- edit(airquality)
> class (aq)
[1] "data.frame  “
> #若要在原数据框上进行修改并保存，可使用如下语句
> fix(aq) # 等价于aq <- edit(aq)

> # 用数据编辑器创建数据框
> df <- edit(data.frame())
```

- edit()函数并不会修改原数据集。关闭数据集编辑器时，修改后的数据集会赋值给aq。



	Ozone	Solar.R	Wind	Temp	Month	Day	var7	var8
1	41	190	7.4	67	5	1		
2	36	118	8	72	5	2		
3	12	149	12.6	74	5	3		
4	18	313	11.5	62	5	4		
5	NA	NA	14.3	56	5	5		
6	28	NA	14.9	66	5	6		
7	23	299	8.6	65	5	7		
8	19	99	13.8	59	5	8		
9	8	19	20.1	61	5	9		
10	NA	194	8.6	69	5	10		
11	7	NA	6.9	74	5	11		
12	16	256	9.7	69	5	12		
13	11	290	9.2	66	5	13		
14	14	274	10.9	68	5	14		
15	18	65	13.2	58	5	15		
16	14	334	11.5	64	5	16		
17	34	307	12	66	5	17		
18	6	78	18.4	57	5	18		
19	30	322	11.5	68	5	19		

内容导航

CONTENTS

```
> v <- c (2, 6, 4, 1, 5)
> sort (v); order (v)
[1] 1 2 4 5 6
[1] 4 1 3 5 2
> v[order(v)] # 与sort(v)得到的结果一致
[1] 1 2 4 5 6
```

- **sort()函数输出向量元素排序后的结果。默认为升序排列。**
- **order()函数输出排序后的元素在原向量中的索引。**

```
> df
  a b   y
1 5 2  75
2 2 5 435
3 2 4  43
4 2 9 735

> df[order(df$a, -df$b),]
  a b   y
4 2 9 735
2 2 5 435
3 2 4  43
1 5 2  75
```

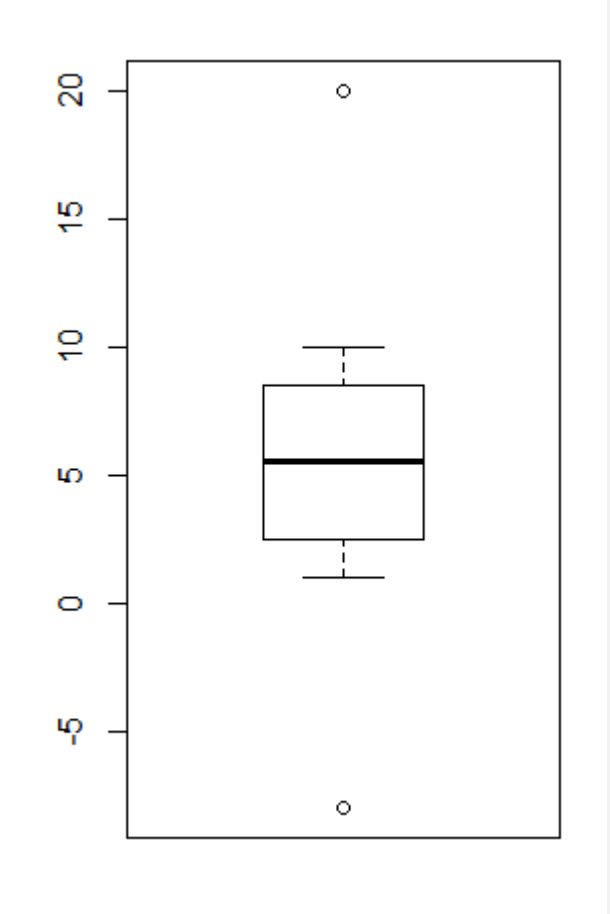
- 在排序属性前添加负号将其指定为降序排列。
- 对数据框中的每一行，依据属性a升序排序，若a相等则按b降序排序。

- 数据分析工作要求数据必须具有满足统计分析所需要的一致性。
  - 记录内的一致性：同一条记录内不能有自相矛盾的信息
  - 记录间的一致性：不同记录的统计属性不能相互冲突
  - 如果用到了多个数据集，可能还会要求数据集之间的一致性。数据需要在所有同一主题的数据集之间保持一致性
- 数据清洗的步骤
  - 数据一致性检测，发现违反规则的数据。比如，年龄不能为负数，GPA不能超过4.0。
  - 挑出造成不一致的变量。
  - 修正错误。

- boxplot.stats()可以得出箱型图统计信息，可查看数据离群点。离群点不一定代表错误，但是发现离群点对于数据分析是很有必要的。

```
> x <- c (1:10,20,-8)  # x是一个1,2...,10,20,-8组成的向量
> boxplot(x)  # 绘制箱型图
> boxplot.stats (x)$out    #显示离群值
[1] 20 -8
```

- 第一和第三四分位数（按升序排列时25%的数据分界点和75%的数据分界点）之间的差值叫做四分位距。比第一四分位数小1.5倍四分位距 以上，或比第三四分位数大1.5倍四分位距以上的数据被当作离群点。



- 缺失值是数据清洗中经常要处理地一种情况，清洗缺失值通常会使用以下几种方法：
  - 删除：删除带有缺失值的变量或样本
  - 替换：用均值、中位值、众数
  - 补全：基于统计模型推断出缺失值
- 对于某些统计量，可以将缺失值直接排除在外即可。例如求某一组数据的均值：

```
> age <- c(25,24,NA,26)
> mean (age)   # 对存在缺失值的向量求均值会得到NA
[1] NA
> mean (age, na.rm = TRUE)   #消除缺失数据对mean函数造成的异常
[1] 25
```

```
> # 构造一个带有缺失值的数据框
> m <- matrix (sample(c(NA, 1:8), 25, replace = TRUE), 5)
> d <- as.data.frame (m); dim(d)
[1] 5 5
> na.omit(d)
  V1 V2 V3 V4 V5
4  6  6  6  3  4
5  4  2  5  1  2
> d[is.na(d)] <- 0 # 用0代替缺失值(仅作演示,实际依需求而定)
```

- 若数据集中含有缺失值，一种常见的做法是使用na.omit()函数删除带有缺失值的行。
- is.na(d)可得到数据框d中为缺失值的索引，随后按需要对其赋值即可完成对缺失值的替换。