统计分析与建模

高珍

gaozhen@tongji.edu.cn

R基础语法

- **Objects**
  - Vectors
  - Arrays
  - Matrices
  - Lists
  - data frames
- Data type and operation
  - Data type
  - Special Constants
  - Operation
- Control flow
  - Conditions
  - Loops
- Function
- I/O

https://www.bejson.com/runcode/r/

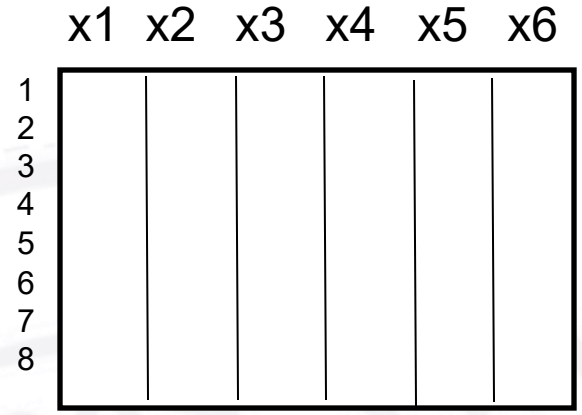
Objects

- Almost all things in R – functions, datasets, results, etc. – are **Objects**.
  - (graphics are written out and are not stored as objects)

- Objects are classified by two criteria:
  - **MODE**(…): how objects are stored in R - character, numeric, logical, factor, list, & function
  - **CLASS**(…): how objects are treated by functions (important to know!) - [vector], matrix, array, data.frame, & hundreds of special classes created by specific functions

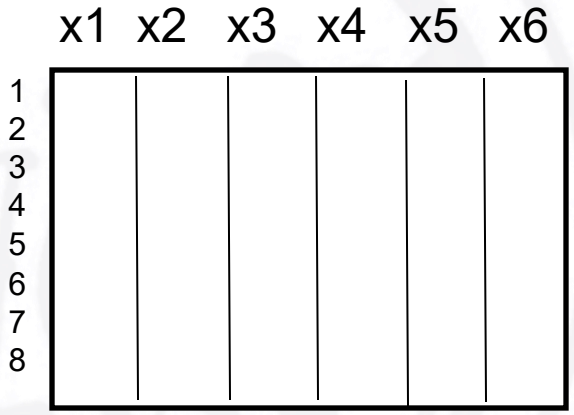
Objects

Z <-



The MODE of Z is determined **automatically** by the types of things stored in Z – numbers, characters, etc. If it is a mix, mode = list.

Z <-



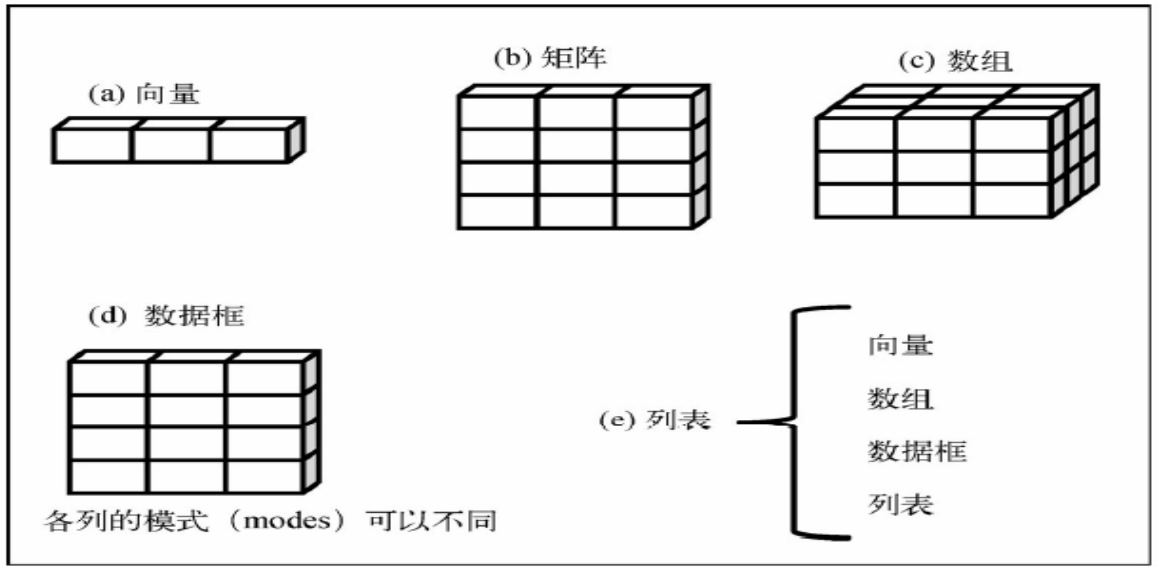
The CLASS of Z is either set by default depending, on how it was created, or is **explicitly set by user**. You can check the objects' class and change it. It determines how functions deal with Z.

DEMO

```
a=c(1,2,3)
class(a)='small_num_list'
(mode(a))
(class(a))
```

Objects in R

- **Vectors**
- Arrays
- Matrices
- Data frames
- Lists



Vectors

- A vector is a single entity consisting of an ordered collection of elements of the **same** type
- To define a vector use the function **c()** (concatenate)
- You can include inputs to a vector of different data type however the resulting vector will make them **all the same type**
- You can transform the vector to any data type as well
  - as.numeric(), as.character()

DEMO

Vector生成

①函数c(...)为自定义量
②from:to产生一个序列
③seq() 产生一个等差向量序列：
seq(from = n, to =m, by = k, len = w)
④rep() 重复一个对象： rep(x, times)
⑤rnorm()随机产生正态分布向量：
rnorm(number, mean, variance)

Vector访问

- You can **access** specific elements of a vector

（1）如何排除下标对应的元素？
（2）满足条件的元素？
which(…), which.min(…), which.max(…)

```
> x[3]              #下标为正数，取出下标对应的元素
> x[-3]             #下标为负数，排除下标对应的元素
> x[3:5]            #取出连续的元素
> x[c(3,5,8)]       #如果一次取出多个元素，需要用向量做下标
> x[-c(3,5,8)]      #如果一次排除多个元素，需要用向量做下标，注意负号
> x[which(x>6)]     #取出满足条件的元素，要使用which()函数
> x[which.max(x)]   #取出最大元素，最小元素小标为which.min
```

Vector更新

- You can **change, add, or delete** parts to a vector

```
13  x <- seq(1, 3, by=0.2)
14  x
15  x[8] <- 34
16  x
17  x[12] <- 3.2
18  x
19  x <- x[c(-8, -12)]
20  x
```

Vector运算

```
27  a <- c(1, 2, 3, 4, 5)
28  a + 1
29  a^2
30  sqrt(a)
31  sum(a)
```

(1) 如果向量长度不同？——v1+v2
(2) 如何获得向量长度？——length(v1)
(3) 如何连接向量？——c(v1,v2)

Objects in R

- Vectors
- **Arrays**
- Matrices
- Data frames
- Lists

Array生成

- Arrays are the R data objects which can store data in **two or more dimensions**

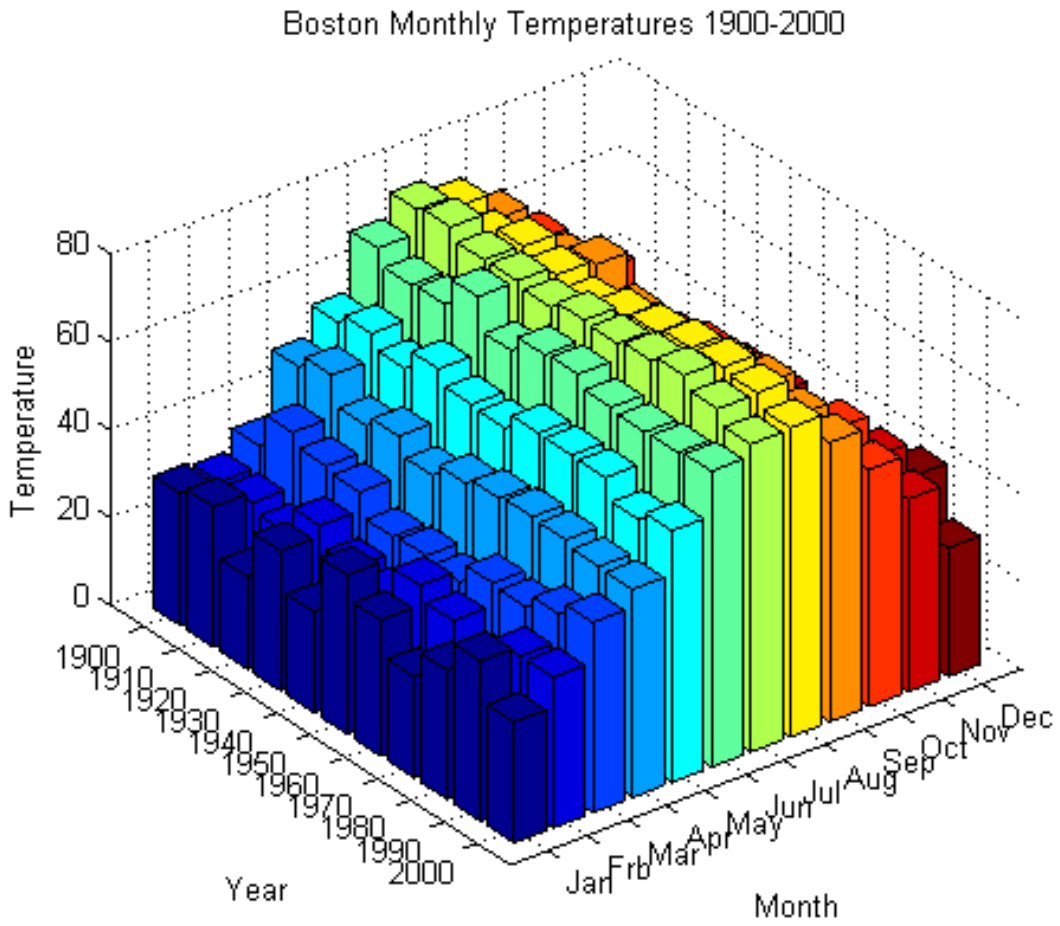
```
> my_array <- array(1:24, dim=c(4,6))
> my_array
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
>
> my_array <- array(c(1, 4, 8, 10), dim=c(2,2))
> my_array
     [,1] [,2]
[1,]    1    8
[2,]    4   10
```

Array排列更新

- You can always **change the dimensions** of an existing array

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
> dim(my_array) <- c(6,4)
> my_array
     [,1] [,2] [,3] [,4]
[1,]    1    7   13   19
[2,]    2    8   14   20
[3,]    3    9   15   21
[4,]    4   10   16   22
[5,]    5   11   17   23
[6,]    6   12   18   24
```

Example of Array Use



- Array访问
- Array更新
- Array运算

单元素访问

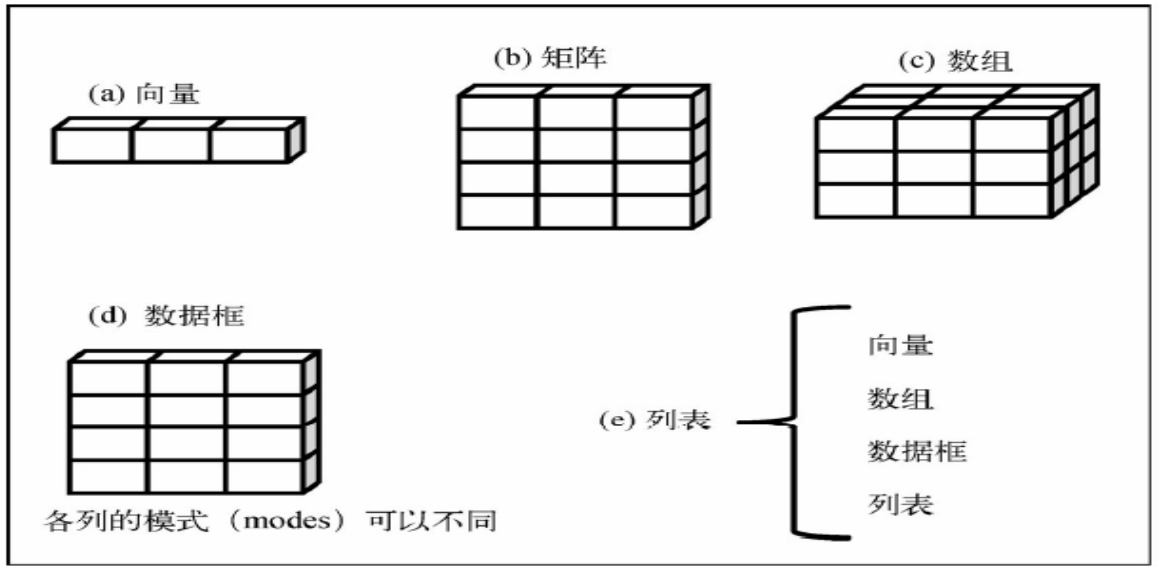
a[i, j, k]

多元素访问

a[c(i1, i2), c(j1, j2), c(k1, k2)]

Objects in R

- Vectors
- Arrays
- **Matrices**
- Data frames
- Lists



Matrice生成

- An array with two dimensions
- You can change the ordering of the matrix to order **by row**

matrix(data,**nrow**=,**ncol**=,**byrow**=T)

```
# 创建一个 3x3 的矩阵
A <- matrix(1:9, nrow = 3)
B <- matrix(9:1, nrow = 3)

# 矩阵乘法
C <- A %*% B
print(C)
```

Matrice访问

- You can **select** specific elements and subsets of a matrix

方法1：使用下标和方括号来选择矩阵中的行、列或元素

- y[i,]：返回矩阵y中的第i行
- y[,j]：返回第j列
- y[i,j]：返回第i行第j列元素
- y[i,-j]：返回第i行,但排除第j列元素
- y[-i,j]：返回第j行,但排除第i行元素

方法2：使用向量和方括号来选择矩阵中的行、列或元素

- y[c(1,3),c(2,4)]：返回第1,3行，第2,4列元素
- y[c(1,3),-c(2,4)]：返回第1,3行，但排除第2,4列元素

Matrice运算

- 横向合并矩阵:cbind();
- 纵向合并矩阵:rbind();
- 转置:t(y);
- 将矩阵转化为向量： as.vector();
- 返回矩阵维度： dim()、nrow() 和ncol()；
- 对矩阵各列求和:colSums();
- 求矩阵各列的均值:colMeans();
- 对矩阵各行求和:rowSums();
- 求矩阵各列的均值:rowMeans();
- 计算行列式:det()；

Objects in R

- Vectors
- Arrays
- Matrices
- **Data frames**
- Lists

Data Frame生成

```
53  Pitchers_Speeds <- data.frame(
54    Pitchers_Name = c("Nolan Ryan","Bob Feller","Aroldis Chapman","Aroldis Chapman","Joel Zumaya"),
55    MPH = c(108.1, 107.6, 106, 105.1, 104.8),
56    stringsAsFactors = FALSE
57  )
58  Pitchers_Speeds
```

```
> Pitchers_Speeds
    Pitchers_Name   MPH
1      Nolan Ryan 108.1
2      Bob Feller 107.6
3 Aroldis Chapman 106.0
4 Aroldis Chapman 105.1
5     Joel Zumaya 104.8
```

Data Frame访问

- d[r,]: rth row of object d
- d[,c]: cth column of object d
- d[r,c]: entry in row r and column c of object d
- d["age"]: extract column "age" from object d
- d$age: extract column "age" from object d

Data Frame函数

- colnames()
- rownames()
- nrow()
- ncol()

Data Frame函数

- **merge()** Two data frames can be merged into one data frame using the function merge.

```
test1 <- read.delim("test1.txt", sep =" ")
test1
    name year   BA   HR
1   Dick 1963 0.12 0.27
2   Gose 1970 0.53 0.74
3   Rolf 1971 0.53 0.28
4 Heleen 1974 0.81 0.29

test2 <- read.delim("test2.txt", sep =" ")
    name year    A   FA
1   Dick 1963 0.42 0.12
2   Gose 1970 0.26 0.57
3   Rolf 1971 0.87 0.37
4 Heleen 1974 0.86 0.15

test.merge <- merge(test1,test2)
test.merge
 name year   BA   HR    A   FA
1   Dick 1963 0.12 0.27 0.42 0.12
2   Gose 1970 0.53 0.74 0.26 0.57
3 Heleen 1974 0.81 0.29 0.86 0.15
4   Rolf 1971 0.53 0.28 0.87 0.37
```

```
quotes = data.frame(date=1:100, quote=runif(100))
testfr = data.frame(date=c(5,7,9, 110), position = c(45,89,14,90))

testfr = merge(quotes,testfr,all.y=TRUE)
testfr
  date     quote position
1    5 0.6488612       45
2    7 0.4995684       89
3    9 0.5242953       14
4  110        NA       90
```

Data Frame函数

- **aggregate**()

The function aggregate is used to aggregate data frames. It splits the data frame into groups and applies a function on each group.

```
gr <- c("A","A","B","B")
x <- c(1,2,3,4)
y <- c(4,3,2,1)
myf <- data.frame(gr, x, y)
aggregate(myf, list(myf$gr), mean)
  Group.1 gr   x   y
1       A NA 1.5 3.5
2       B NA 3.5 1.5
```

```{r}
gr1=c('A','A','B','B')
gr2=c('A1','A2','B1','B1')
x=c(1,2,3,4)
y=c(5,6,7,8)
(f=data.frame(gr1,gr2,x,y))
aggregate(f[,c(3,4)],by=list(gr1,gr2),mean)
```

Description: df [3 × 4]

Group.1 <chr>	Group.2 <chr>	x <dbl>	y <dbl>
A	A1	1.0	5.0
A	A2	2.0	6.0
B	B1	3.5	7.5

3 rows

Description: df [4 × 4]

gr1 <chr>	gr2 <chr>	x <dbl>	y <dbl>
A	A1	1	5
A	A2	2	6
B	B1	3	7
B	B1	4	8

4 rows

Data Frame函数

- **stack**()
  - The function stack can be used to stack columns of a data frame into one column and one grouping column

```
group1 <- rnorm(3)
group2 <- rnorm(3)
group3 <- rnorm(3)
df <- data.frame(group1,group2, group3)
```

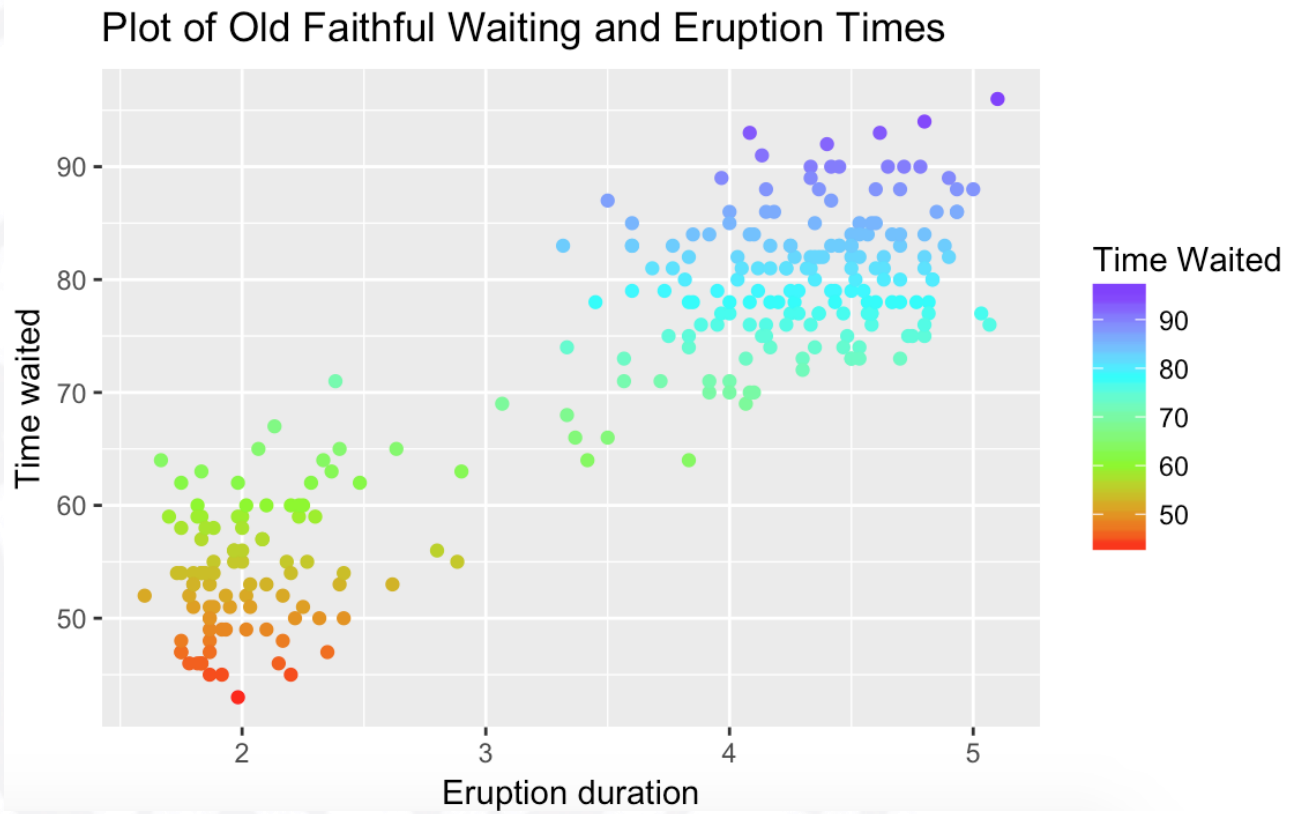
group1 <dbl>	group2 <dbl>	group3 <dbl>
-0.8935259	-1.05613011	0.7447249
-0.6174068	1.92203525	-0.7599139
0.6729324	-0.08939907	-0.4169499

```
stack(df)
```

values <dbl>	ind <fctr>
-0.89352595	group1
-0.61740677	group1
0.67293242	group1
-1.05613011	group2
1.92203525	group2
-0.08939907	group2
0.74472487	group3
-0.75991386	group3
-0.41694994	group3

Data Frame应用

```
21  ggplot(faithful, aes(x = faithful$eruptions, y = faithful$waiting, colour = faithful$waiting)) +
22    geom_point(aes()) +
23    scale_colour_gradientn(name = "Time Waited", colours=rainbow(4)) +
24    labs(x = "Eruption duration", y = "Time waited") +
25    ggtitle("Plot of Old Faithful Waiting and Eruption Times")
```




```
> faithful
    eruptions waiting
1       3.600      79
2       1.800      54
3       3.333      74
4       2.283      62
5       4.533      85
6       2.883      55
7       4.700      88
8       3.600      85
9       1.950      51
10      4.350      85
11      1.833      54
12      3.917      84
13      4.200      78
14      1.750      47
```

Objects in R

- Vectors
- Arrays
- Matrices
- Data frames
- **Lists**

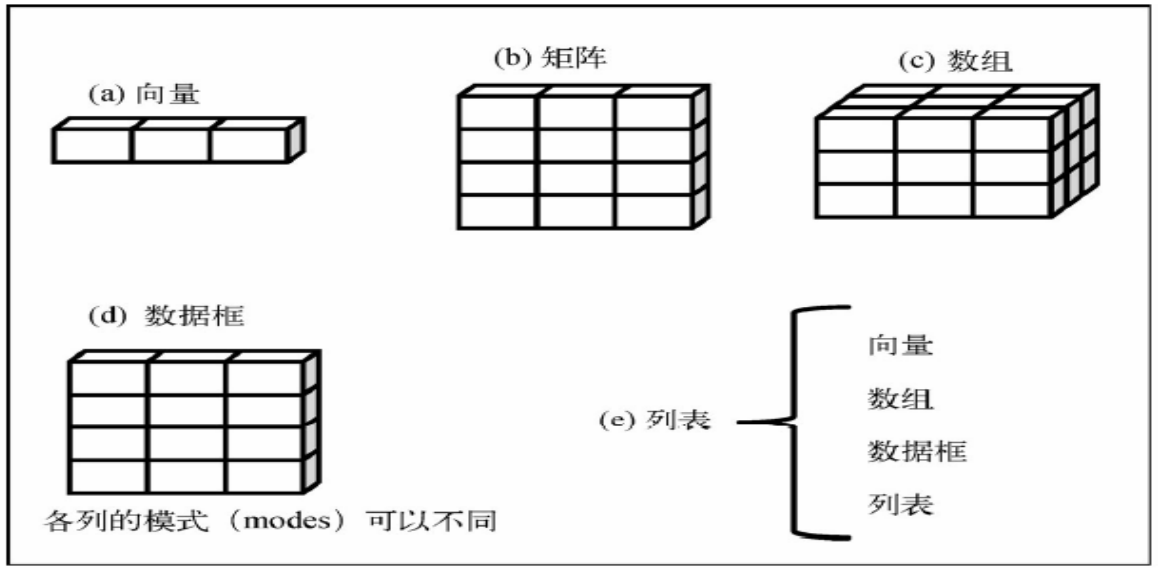
List 生成&访问

- A list is an object containing elements of any type, including other objects

```
mylist <- list(a = 1:3, b = "hello", c = matrix(1:4, 2, 2))
mylist[1]    # 返回包含元素a的列表
mylist[”a”] # 返回包含元素a的列表
mylist$a     # 访问元素 a
mylist[[1]] # 访问元素 a
```

Objects in R

- Vectors
- Arrays
- Matrices
- Data frames
- Lists



扩展阅读：ts

```{r ts}
 myts1 <- ts(data = rnorm(100), start=c(1987), freq = 12)
 tsp(myts1)
 myts2 <- ts(data = matrix(rnorm(100),ncol=2), start=c(1987,4), freq=12)
 tsp(myts2)
```

```
[1] 1987.00 1995.25   12.00
[1] 1987.250 1991.333   12.000
```

Attributes

```{r ts-2}
 attributes(myts1)
```

```
$tsp
[1] 1987.00 1995.25   12.00

$class
[1] "ts"
```

```
              Jan          Feb          Mar          Apr          May
1987  0.596810423  0.673090025  0.221728908 -0.945973650 -0.275669232
1988  0.046128499  0.729604206  1.571093199  0.793378194 -2.124475262
1989  1.136379750  0.354477352 -0.020180716  0.502346831 -2.049895972
1990 -1.605261218 -1.097419428  0.056647473  0.904257794 -1.492322196
1991 -0.997676795  0.732180442 -0.644546109  0.403500684 -0.056973110
1992 -0.713443569  1.015353141 -1.619333396  0.222546252  0.318044949
1993 -0.812307978 -0.776311435  0.116474834  0.634127039 -0.019213579
1994  0.129156883  0.975929852  1.528174631  0.054049106  1.329992724
1995  0.347079293 -1.662569406  1.134133996 -0.708107736
              Jun          Jul          Aug          Sep          Oct
1987 -1.723999355 -1.711261593  0.308002424 -1.811040362  0.529584847
1988 -0.013527532  0.262976206 -1.892582681  0.427367007  0.791023022
1989 -1.399155638 -0.537394067 -3.368797340  1.591608556  1.128888373
1990 -0.983085334  0.448448783 -0.359620799  0.895006618  0.790878123
1991 -0.751655757 -0.637122966 -0.278422051  1.645725778  1.280306695
1992 -1.072479106  1.272251578 -0.276779046  1.189197255  0.229086277
1993  0.091579973 -0.206504772 -0.006067391  1.696747114 -0.892035033
1994 -0.164085694  0.054521574 -0.409036551 -1.308784910  0.264421432
1995
              Nov          Dec
1987  0.752834818 -1.912643758
1988  1.785845775  1.455185240
1989  0.201966987 -0.444604725
1990  0.565319694  0.044472613
1991 -0.139837665  0.361251897
1992  0.007191677  0.751474723
1993 -1.244280470  0.578293815
1994  1.141872976  1.196038968
1995
```

Contents

- Objects
  - Vectors
  - Arrays
  - Matrices
  - Lists
  - data frames
- **Data type and operation**
  - Data type
  - Special Constants
  - Operation
- Control flow
  - Conditions
  - Loops
- Function
- I/O

Contents

- Data type and operation
  - **Data type**
  - Special Constants
  - Operation

Data Types

- Logical – True/False
- Integer – 4, 6, 2
- Numeric – 3.4, 566, 2.34
- Character – "a", "hello"
- Factor – "Female", "Male”
- Date & Times – “2021-09-15”, “ 2021-09-15 13:30:00”

Assigned Variables

- Make sure to take into account data type when doing this

```
> a <- 1
> b <- 2
> a+b
[1] 3
> r <- "Hello"
> t <- "World"
> paste(r,t, sep = " ")
[1] "Hello World"
```

Character

- **nchar**()

```
x <- c("a","b","c")
mychar1 <- "This is a test"
mychar2 <- "This is another test"
charvector <- c("a", "b", "c", "test")
nchar(mychar1)
[1] 15
nchar(charvector)
[1] 1 1 1 4
```

- **substring**()

```
x <- c("Gose", "Longhow", "David")
substring(x,first=2,last=4)
[1] "ose" "ong" "avi"
```

- **paste**()

```
paste("number",1:10, sep=".")
[1] "number.1"  "number.2"  "number.3"  "number.4"
[5] "number.5"  "number.6"  "number.7"  "number.8"
[9] "number.9"  "number.10"
```

grep () Finding patterns in character objects

```
car.names <- row.names(cars)
grep("Volvo", car.names)
[1] 37
```

- **gsub**()

```
mychar <- c("mytest", "abctestabc", "test.po.test")
gsub(pattern="test", replacement="", x=mychar, fixed=TRUE)
[1] "my"     "abcabc" ".po."
```

String Split

- strsplit()

```
strsplit(x = c("Some text", "another string", split = NULL)
[[1]]
[1] "S" "o" "m" "e" " " "t" "e" "x" "t"

[[2]]
 [1] "a" "n" "o" "t" "h" "e" "r" " " "s" "t" "r" "i" "n" "g"

strsplit(
   x = c("Some~text" , "another-string", "Amsterdam is a nice city"),
   split = "[~-]"
)
[[1]]
[1] "Some" "text"

[[2]]
[1] "another" "string"

[[3]]
[1] "Amsterdam is a nice city"
```

Factor(因子)

- 名义型变量是没有顺序之分的类别变量。下表中，糖尿病类型Diabetes（Type1、Type2）是名义型变量的一例。即使在数据中Type1编码为1而Type2编码为2，这也并不意味着二者是有序的。
- 有序型变量表示一种顺序关系，而非数量关系。病情Status（poor, improved, excellent）是顺序型变量的典型示例。我们知道，病情为poor（较差）病人的状态不如improved（病情好转）的病人，但并不知道相差多少。

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

Factor

- 名义型
  - diabetes<-c("type1","type2", "type1", "type1")
- 有序型
  - status<-c("Poor", "Improved", "Excellent", "Poor")
  - status <- factor(status, ordered=TRUE)
  - status<-factor(status, levels=c("Poor","Improved", "Excellent"))

Creating factors from continuous data

Example 1

```
x <- 1:15
breaks <- c(0,5,10,15,20)
cut(x,breaks)
 [1] (0,5]   (0,5]   (0,5]   (0,5]   (0,5]   (5,10]  (5,10]  (5,10]  (5,10]
[10] (5,10]  (10,15] (10,15] (10,15] (10,15] (10,15]
Levels: (0,5] (5,10] (10,15] (15,20]
```

Example 2

```
cut( x, breaks=5)
 [1] (0.986,3.79] (0.986,3.79] (0.986,3.79] (3.79,6.6]   (3.79,6.6]
 [6] (3.79,6.6]   (6.6,9.4]    (6.6,9.4]    (6.6,9.4]    (9.4,12.2]
[11] (9.4,12.2]   (9.4,12.2]   (12.2,15]    (12.2,15]    (12.2,15]
Levels: (0.986,3.79] (3.79,6.6] (6.6,9.4] (9.4,12.2] (12.2,15]
```

Example 3

```
x <- rnorm(15)
cut(x, breaks=3, labels=c("low","medium","high"))
 [1] high   medium medium medium medium high   low    high   low    low
[11] high   low    low    medium high
Levels: low medium high
```

Date & Time

```
temp <- c("12-09-1973", "29-08-1974")
z <- as.Date(temp, "%d-%m-%Y")
z
[1] "1973-09-12" "1974-08-29"
data.class(z)
[1] "Date"
format(z, "%d-%m-%Y")
[1] "12-09-1973" "29-08-1974"
```

```
z + 19
[1] "1973-10-01" "1974-09-17"
```

Unit: day

```
dz = z[2] -z[1]
dz
data.class(dz)
Time difference of 351 days
[1] "difftime"
```

```
t1 <- as.POSIXct("2003-01-23")
t2 <- as.POSIXct("2003-04-23 15:34")
t1
t2
[1] "2003-01-23 W. Europe Standard Time"
[1] "2003-04-23 15:34:00 W. Europe Daylight Time"
```

```
# pasting 4 character dates and 4 character times together
dates <- c("02/27/92", "02/27/92", "01/14/92", "02/28/92")
times <- c("23:03:20", "22:29:56", "01:03:30", "18:21:03")
x <- paste(dates, times)
z <- strptime(x, "%m/%d/%y %H:%M:%S")
zt <- as.POSIXct(z)
zt
[1] "1992-02-27 23:03:20 W. Europe Standard Time"
[2] "1992-02-27 22:29:56 W. Europe Standard Time"
[3] "1992-01-14 01:03:30 W. Europe Standard Time"
[4] "1992-02-28 18:21:03 W. Europe Standard Time"
```

```
zt + 13
[1] "1992-02-27 23:03:33 W. Europe Standard Time"
[2] "1992-02-27 22:30:09 W. Europe Standard Time"
[3] "1992-01-14 01:03:43 W. Europe Standard Time"
[4] "1992-02-28 18:21:16 W. Europe Standard Time"
```

Unit: second

```
t2 <- as.POSIXct("2004-01-23 14:33")
t1 <- as.POSIXct("2003-04-23")
d <- t2-t1
d
Time difference of 275.6479 days
```

Date & Time

- Functions
  - weekdays ()
  - months ()
  - quarters ()
  - Sys.time ()
  - …
- Packages
  - zoo
  - chron
  - tseries
  - Rmetrics
  - …

```
weekdays(zt)
[1] "Thursday" "Thursday" "Tuesday"  "Friday"
```

Contents

- Data type and operation
  - Data type
  - **Special Constants**
  - Operation

Special Constants

- NA
- Inf
- -Inf
- TRUE
- FALSE

Contents

- Data type and operation
  - Data type
  - Special Constants
  - **Operation**

Math Operators

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

Relational Operators

Sign	Meaning
==	Equals
!=	Does not equal
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal to
%in%	Included in
is.na()	Is a missing value

```{r}
a=c(1,NA,3,3)
is.na(a)
a[is.na(a)]
a[!is.na(a)]
```

```
[1] FALSE  TRUE FALSE FALSE
[1] NA
[1] 1 3 3
```

String operation

- nchar()
- paste(str1,str2,sep)
- strsplit(string,sep)
- substr(string,start,stop)
- chartr(old,new,string)

Object operation

函数	用途
length(object)	显示对象中元素/成分的数量
dim(object)	显示对象的维度
str(object)	显示对象的结构
class(object)	显示对象的类别
mode(object)	显示对象的模式
names(object)	显示象中各成分的名称
c(object,object,…)	将对象合并入一个向量
cbind(object,object,…)	按列合并对象
rbind(object,object,…)	按行合并对象

Convert operation

From To	Vector	Matrix	Data frame
Vector	c(x,y)	cbind(x,y) rbind(x,y)	data.frame(x,y)
Matrix	as.vector(mymatrix)		as.data.frame(mymatrix)
Data frame		as.matrix(myframe)	

Apply operation

```{r for_loop}
set.seed(431)
(mat43=replicate(4,sample(3,3)))
(mat43l=rep(0,4))
for (j in 1:4) {mat43l[j]=max(mat43[,j])}
mat43l
```

```
     [,1] [,2] [,3] [,4]
[1,]    3    2    2    3
[2,]    2    1    3    2
[3,]    1    3    1    1
[1] 0 0 0 0
[1] 3 3 3 3
```

```
apply(mat43,2,max)
```

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

```{r apply}
apropos("apply")
```

```
 [1] ".rs.api.applyTheme" ".rs.applyTheme"
 [3] ".rs.applyTransform" "apply"
 [5] "dendrapply"         "eapply"
 [7] "kernapply"          "lapply"
 [9] "mapply"             "rapply"
[11] "sapply"             "tapply"
[13] "vapply"
```

扩展阅读：

《An introduction to R》

6 Efficient calculations

Contents

- Objects
  - Vectors
  - Arrays
  - Matrices
  - Lists
  - data frames
- Data type and operation
  - Data type
  - Special Constants
  - Operation
- **Control flow**
  - Conditions
  - Loops
- Function
- I/O

Conditions

if (cond) {expr1} else {expr2}

```
> w = 3
> if( w < 5 )
   {
   d=2
   }else{
   d=10
   }
> d
2
```

switch

```
switch(object,
    "value1" = {expr1},
    "value2" = {expr2},
    "value3" = {expr3},
               {other expressions}
)
```

Loops

While-loop

while (cond) {expr}

```
while(tmp < 100){
  tmp <- tmp + rbinom(1,10,0.5)
  n <- n +1
}
```

For-loop

for (var in vec) {expr}

```
for(i in 2:length(x))
{
  x[i] <- x[i] + phi*x[i-1]
}
```

Repeat-loop

```
repeat
{
   some expressions
}
```

In the repeat loop some expressions are repeated ‘infinitely’, so repeat loops will have to contain a break statement to escape them.

Contents

- Objects
  - Vectors
  - Arrays
  - Matrices
  - Lists
  - data frames
- Data type and operation
  - Data type
  - Special Constants
  - Operation
- Control flow
  - Conditions
  - Loops
- **Function**
- I/O

Build-in functions

Data creation

- `read.table`: read a table from file. Arguments: `header=TRUE`: read first line as titles of the columns; `sep=","`: numbers are separated by commas; `skip=n`: don't read the first `n` lines.
- `write.table`: write a table to file
- `c`: paste numbers together to create a vector
- `array`: create a vector, Arguments: `dim`: length
- `matrix`: create a matrix, Arguments: `ncol` and/or `nrow`: number of rows/columns
- `data.frame`: create a data frame
- `list`: create a list
- `rbind` and `cbind`: combine vectors into a matrix by row or column

Data processing

- `seq`: create a vector with equal steps between the numbers
- `rnorm`: create a vector with random numbers with normal distribution (other distributions are also available)
- `sort`: sort elements in increasing order
- `t`: transpose a matrix
- `aggregate(x,by=ls(y),FUN="mean")`: split data set `x` into subsets (defined by `y`) and computes means of the subsets. Result: a new list.
- `na.approx`: interpolate (in `zoo` package). Argument: vector with `NA`s. Result: vector without `NA`s.
- `cumsum`: cumulative sum. Result is a vector.
- `rollmean`: moving average (in the `zoo` package)
- `paste`: paste character strings together
- `substr`: extract part of a character string

Function

name=function(arglist){expr}

```
ExpAnd <- function(vec=seq(4,25,3),exponent=2,addto=3)
{
  # Function that takes argument vec to the power
  #  exp, adds add and then outputs the result
  out<-vec^exponent+addto
  return(out)
}
ExpAnd()
```

```
> fun1 = function(arg1, arg2 )
   {
   w = arg1 ^ 2
   return(arg2 + w)
   }
> fun1(arg1 = 3, arg2 = 5)
[1] 14
```

Contents

- Objects
  - Vectors
  - Arrays
  - Matrices
  - Lists
  - data frames
- Data type and operation
  - Data type
  - Special Constants
  - Operation
- Control flow
  - Conditions
  - Loops
- Function
- **I/O**

I/O: txt file

```
file.show("matHap.txt")
```

```
"DYS19" "DXYS156Y" "DYS389m" "DYS389n" "DYS389p"
"H1" 14 12 4 12 3 10 8 10 1 4 15 13 0 1 1
"H3" 15 13 4 13 3 9 8 10 1 4 13 12 0 1 1
"H4" 15 11 5 11 3 10 8 10 1 4 11 14 0 1 1
```

```
haptable=read.table("matHap.txt")
haptable
```

```
##    DYS19 DXYS156Y DYS389m DYS389n DYS389p
## H1    14       12       4      12       3
## H3    15       13       4      13       3
## H4    15       11       5      11       3
## H5    17       13       4      11       3
## H7    13       12       5      12       3
## H8    16       11       5      12       3
```

```
> d = data.frame(a = c(3,4,5),
 b = c(12,43,54))
> d
  a  b
1 3 12
2 4 43
3 5 54
> write.table(d, file="tst0.txt",
 row.names=FALSE)
> d2 = read.table(file="tst0.txt",
 header=TRUE)
> d2
  a  b
1 3 12
2 4 43
3 5 54
```

I/O: csv file

```
> 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: 0x7fbbca2577b0>
<environment: namespace:utils>
```

```
file.show("Haplotype.csv")
```

```
Hap=read.csv("/Users/susan/Dropbox/stat32/Slides_ABCofR/Haplotype.csv")
head(Hap)
```

```
Hapscan<-scan("/Users/susan/Dropbox/stat32/Slides_ABCofR/Haplotype.csv",nlines=5,what="")
Hapscan
```

```
## [1] "Individual,DYS19,DXYS156Y,DYS389m,DYS389n,
## [2] "H1,14,12,4,12,3,10,8,10,1,4,15,13,0,1,1"
## [3] "H3,15,13,4,13,3,9,8,10,1,4,13,12,0,1,1"
## [4] "H4,15,11,5,11,3,10,8,10,1,4,11,14,0,1,1"
## [5] "H5,17,13,4,11,3,10,7,10,1,4,14,12,0,1,1"
```

write.csv(data,file=”data.csv",row.names = FALSE)

I/O: excel file

```
install.packages("readxl")
```

```
library(readxl)
?read_excel
Hapall <- read_excel("/Users/susan/RWork/data/Haplotype.xlsx")
```

Find more at https://cran.r-project.org/doc/manuals/r-release/R-data.html

R基础语法

- Objects
  - Vectors
  - Arrays
  - Matrices
  - Lists
  - data frames
- Data type and operation
  - Data type
  - Special Constants
  - Operation
- Control flow
  - Conditions
  - Loops
- Function
- I/O

Canvas 随堂测试

课后练习

- 分析faithful数据
  - 保存到csv文件(write.csv)
  - 从csv文件读入(read.csv)
  - 查看数据一阶统计特征(summary)
  - 画数据散点图(plot)
  - 画数据分布图(hist)
  - 根据以上分析，回答下面问题
    - 如果游客刚错过一次喷泉，您建议游客多久后回来看下一次喷泉？
    - 你能根据等待的时间来估计下一次喷泉能喷多久吗？

Thanks

高 珍
同济大学
计算机科学与技术学院
（gaozhen@tongji.edu.cn）