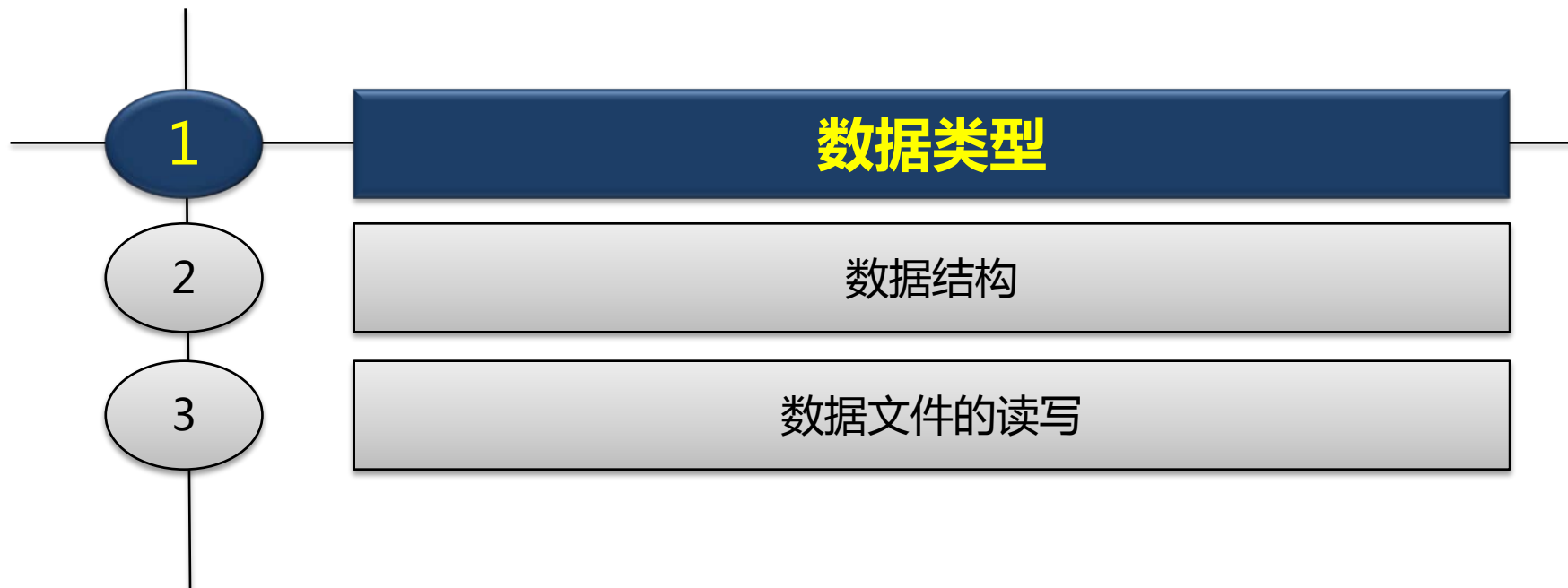


# 信息分析与预测实验



## 第2章 数据对象与数据读写

经济与管理学院 孙蕾



A vertical line on the left side of the page contains three circles. The top circle is dark blue with a yellow number '1'. The middle and bottom circles are light gray with black numbers '2' and '3' respectively. To the right of these circles are three horizontal rectangular boxes. The top box is dark blue with yellow text '数据类型'. The middle and bottom boxes are light gray with black text '数据结构' and '数据文件的读写' respectively. A horizontal line passes through the middle of the top circle and the top box.

1	数据类型
2	数据结构
3	数据文件的读写

# 数据对象类型

## ➤ 基本赋值语句

- `x <- 8`
- `a <- 'city'`



## ➤ C函数

C(常量或向量名列表)

- `W<- c(1, 2, 3)`
- `u <- c("red", "yellow", "blue")`

# 数据对象类型

- R 语言的对象常见的数据类型有：字符型、数值型、逻辑型、复数型。此外，也可能是缺省值(NA)。

## 判别和转换数据对象类型的函数

类型	辨别	转换
character	is.character()	as.character()
complex	is.complex()	as.complex()
integer	is.integer()	as.integer()
logical	is.logical()	as.logical()
NA	is.na()	as.na()
numeric	is.numeric()	as.numeric()

# 日期类型

- 在R中，字符型的日期值无法进行日期变量的计算，因此可通过日期值处理函数，将字符型的日期值转换成日期变量。

## 日期变量常用函数

函数	功能
Sys.Date()	返回系统当前的日期
Sys.time()	返回系统当前的日期和时间
date()	返回系统当前的日期和时间（返回的值为字符串）
as.Date()	将字符串形式的日期值转换为日期变量
as.POSIXlt	将字符串转化为包含时间及时区的日期变量
strptime()	将字符型变量转化为包含时间的日期变量
strftime()	将日期变量转换成指定格式的字符型变量
format()	将日期变量转换成指定格式的字符串

# 日期类型

- `as.Date()` ——将字符串形式的日期值转换为日期变量，以数值形式存储；
- 使用格式：`as.Date(x, format = "", ...)`
- 其中x是要转换的对象，为字符型数据，format则给出了用于读入日期的适当格式。

读入日期的格式

符号	含义	示例	符号	含义	示例
%d	数字表示的日期（00~31）	01~31	%Y	四位数的年份	2016
%a	缩写的星期名	Mon	%H	24小时制小时	00-23
%A	非缩写的星期名	Monday	%I	12小时制小时	01-12
%w	数字表示的星期天数	0-6，周日为0	%p	AM/PM指示	AM/PM
%m	数字表示的月份（00~12）	00~12	%M	十进制的分钟	00-60
%b	缩写的月份	Jan	%S	十进制的秒	00-60
%B	非缩写的月份	January	%y	二位数的年份	16

# 日期类型

as.POSIXlt() ——将字符串形式的日期时间值转换为指定的格式的时间变量

- 使用格式： `as.POSIXlt(x, tz = "", format)`
- 其中x为想要转换的字符串型日期时间值；tz指定转换后的时区，""为当前时区，"GMT"为UTC时区；format指定要转换的日期值的格式。

- 实例：将字符串型日期时间值转换为时间变量

```
x <- c("2016-02-08 10:07:52", "2016-08-07 19:33:02")
is.character(x)
y=as.POSIXlt(x, tz = "", "%Y-%m-%d %H:%M:%S")
y
typeof(y)
z=as.Date(x)
z
typeof(z)
```

# 日期类型

`strptime()` ——将字符型的日期时间值转换为时间变量

- 使用格式： `strptime(x, format, tz = "")`
- 其中x是字符型数据，format指定要转换的日期值的格式, tz指定时区，"" 时为当前时区，“GMT” 为UTC时区。

```
(date2 <- strptime(x, "%Y- %m- %d %H: %M: %S"))
```

`strftime()`——将时间变量按指定的格式转换为字符型日期值。

- 使用格式： `strftime(x, format = "")`
- 其中x是时间变量，format为想要转化成的字符型日期值的输出格式。

```
(date <- strftime(z, "%Y/%m/%d %H: %M: %S"))
```



# 日期类型

---

`format()` ——将对象转化按指定格式转化成字符串

- 使用格式： `format(x,...)`
- 其中x为要转换为字符串的对象，...指定要转换成的字符串的格式。
- 实例：将时间变量转化为字符串日期值

```
(date3<- format(z, "%d/%m/%Y"))  
[1] "08/02/2016" "07/08/2016"
```

```
typeof(date3)  
[1] "character"
```

# 查看对象的类型

对于未知类型的对象，在R中有3个函数可以查看对象的类型: `class()`、`mode()`、`typeof()`。

➤ 使用格式： `class(x)`

➤ 其中x为需要查看类型的对象，`mode()`、`typeof()`函数使用格式与`class()`函数相同。

➤ 实例：创建3个不同类型的数据，展示3个辨别函数的区别。

```
> df <- data.frame(c1 = letters[1:3], c2 = 1:3, c3 = c(1, -1, 3.0), stringsAsFactors = F)
> sapply(df, mode)
```

c1	c2	c3
"character"	"numeric"	"numeric"

```
> sapply(df, class)
```

c1	c2	c3
"character"	"integer"	"numeric"

```
> sapply(df, typeof)
```

c1	c2	c3
"character"	"integer"	"double"

➤ 在展现数据的细节上，`mode()`<`class()`<`typeof()`。`mode()`函数只查看数据的大类，`class()`函数查看数据的类，`typeof()`函数则更加细化，查看数据的细类。





# 数据结构

---

- R拥有许多用于存储数据的对象类型，包括向量、矩阵、数组、数据框和列表。
- 数据框(data frame)是R中用于存储数据的一种结构：列表示变量，行表示观测。在同一个数据框中可以存储不同类型(如数值型、字符型)的变量。数据框将是你用来存储数据集的主要数据结构。

# 向量

- 向量是以一维数组的方法管理数据的一种对象类型。可以说向量是R语言中最基本的数据类型，很多算法函数都是以向量的形式输入的；
- 字符型、逻辑值型(T、F)、数值型和复数型；
- 一个向量的所有元素都必须属于相同的类型。如果不是，R将强制执行类型转换。
- `length()`测长度；
- `c()`函数创建：
  - `> w <- c(1, 3, 4, 5, 6, 7)`  
• `> length(w)`  
[1] 6  
• `> mode(w)`  
[1] "numeric"  
• `w=c(1:4)`
  - `> w2 <- c(T, F, T)`  
• `> length(w2)`  
[1] 3  
• `> mode(w2)`  
[1] "logical"
  - `> w1 <- c('张三', '李四', '王五')`  
• `> length(w1)`  
[1] 3  
• `> mode(w1)`  
[1] "character"  
• `> x2 <- c("a", "b", "c", "d")`
- 判断数据对象是否为向量：`is.vector (数据对象名)`

# 向量

➤ 一个向量的所有元素都必须属于相同的类型。如果不是，R将强制执行类型转换。

- `> w4 <- c(w, w1)`

- `> w4`

```
[1] "1"      "3"      "4"      "5"      "6"      "7"      "张三" "李四" "王五"
```

- `> mode(w4)`

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

- `> w5 <- c(w1, w2)`

- `> w5`

```
[1] "张三" "李四" "王五" "TRUE" "FALSE" "TRUE"
```

- `> mode(w5)`

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



# 向量化

➤ R语言最强大的方面之一就是函数的向量化。

➤ 向量的算术运算

- `> (w <- seq(1:10))`

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

- `> (x <- sqrt(w))`

- `[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427  
3.000000 3.162278`

➤ 如果两个向量的长度不同？

- `> (w1 <- c(2, 3, 2, 3))`

- `[1] 2 3 2 3`

- `> (w2 <- c(3, 1, 4, 2, 5, 3))`

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

- `> (w <- w1 + w2)`

- `[1] 5 4 6 5 7 6`

- Warning message:

- In `w1 + w2` :

- longer object length is not a multiple of shorter object length

# 等差序列的创建

seq( )产生等距间隔的数列，其基本形式为:

- seq(from = 1, to = 1, by = ((to - from) / (length.out - 1)), length.out = NULL, along.with = NULL, ...)

参数	描述
from	等差数列的首项数据，默认为1
to	等差数列的尾项数据，默认为1
by	等差的数值
length.out	产生向量的长度

Seq(from=起始值, to=终止值, by=步长)  
Seq(from=起始值, to=终止值, length=个数)



# 等差序列的创建

---

seq( )产生等距间隔的数列

- `> (seq(1, -9))`

```
[1] 1 0 -1 -2 -3 -4 -5 -6 -7 -8 -9
```

- `> (seq(1, -9, length = 5))`

```
[1] 1.0 -1.5 -4.0 -6.5 -9.0
```

- `> (seq(1, -9, by = -2))`

```
[1] 1 -1 -3 -5 -7 -9
```

- `> (seq(1, 10, 2))`

```
[1] 1 3 5 7 9
```



## 重复序列的创建

`rep()` 是重复函数。其基本形式是 `rep(x,n)`。其中 `x` 是预重复的序列，`n` 是重复的次数。

- `> rep(1:4, 2)`

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

- `> rep(1:4, each=2)`

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

- `> rep(1:4, c(2, 2, 2, 2))`

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

- `> rep(1:4, c(2, 1, 2, 1))`

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

- `> rep(1:4, each=2, len=4)`

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

- `> rep(1:4, each=2, times=3)`

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

`rep` (起始值: 终止值, `each`=重复次数)

每个值依次重复`each`指定的次数

`rep` (起始值: 终止值, `times`=重复次数)

取值范围重复`times`指定的次数

# 索引向量

- 通常，我们只要访问向量中的部分或个别元素。这就是所谓的索引，它用方括号[]来实现。（有人也称之为子集、下标或切片，这些术语所指相同。）

**vector:**

a	t	c	b	f	g	h
---	---	---	---	---	---	---

**i n d e x:**

1	2	3	4	5	6	7
---	---	---	---	---	---	---

# 索引向量

---

R语言中，提供如下多种索引方法。

- 给向量传入正数，它会返回此位置上的向量元素切片。它的第一个位置是 1（而不像其他某些语言一样是0）。
- 给向量传入负数，它会返回一个向量切片，它将包含除了这些位置以外的所有元素。
- 给向量传入一个逻辑向量，它会返回一个向量切片，里面只包含索引为 TRUE 的元素。
- 对于已命名的向量，给向量传入命名的字符向量，将会返回向量中包含这些名字的元素切片。

# 向量索引

---

- `x[n]` 第n个元素
- `x[-n]` 除了第n个元素的x
- `x[1:n]` 前n个元素
- `x[-(1:n)]` 第n+1至最后的元素
- `x[c(1, 4, 2)]` 指定元素
- `x["name"]` 名为"name"的元素
- `x[x > 3]` 所有大于3的元素
- `x[x>3 & x<5]` 区间(3, 5)的元素
- `x[x %in% c("a", "and", "the")]` 给定组中的元素

# 索引向量

➤ 以下三个索引方法都将返回相同的值：

- `> x <- c(2, 4, 6, 8, 1)`  
`> x[c(1, 3, 5)]`  
[1] 2 6 1
- `> x[c(-2, -4)]`  
[1] 2 6 1
- `> x[c(TRUE, FALSE, TRUE, FALSE, TRUE)]`  
[1] 2 6 1
- `> x[-c(5:9,10)]`  
访问除第5至第9以及第10个元素以外的元素

➤ 混合使用正负值是不允许的，会抛出一个错误：

- `> x[c(1,-1)]`
- Error in x[c(1, -1)] : only 0's may be mixed with negative subscripts

向量名[-位置常量]

向量名[-c(位置常量列表)]

向量名[-(位置常量1: 位置常量2)]

向量名[-位置向量名]

# 索引向量

---

➤ which 函数将返回逻辑向量中为TRUE 的位置。

- > a <- c(2,4,6,8,1)

- > which(a>3)

[1] 2 3 4

➤ which.min 和which.max 分别是which(min(x)) 和which(max(x)) 的简写：

- > which.min(a)

[1] 5

- > which.max(a)

[1] 4

# 向量编辑

- R语言可以对已经创建好的向量直接进行元素扩展及删除等编辑操作。
- 向量中元素的删除通过减号加元素下标的形式实现。
- 实例：向量元素的扩展及删除

```
# 向量编辑
```

```
> y <- c(1, 2, 3, 4)
```

```
# 向量扩展
```

```
> (y <- c(y, c(5, 6, 7)))
```

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

```
# 单个元素的删除
```

```
> (y <- y[-1])
```

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

```
# 多个元素的删除
```

```
> (x <- x[c(3:5)])
```

```
[1] 4 5 6
```



# 向量排序

- `y <- c(2, 5, 4, 7, 1, 0, 9)`

`(sort(y))`

`[1] 0 1 2 4 5 7 9`

- `u <- c("red", "yellow", "blue")`

`sort(u, decreasing=true)`

`[1] "yellow" "red" "blue"`

## sort()函数常用参数

常用参数	参数描述	选项
<b>x</b>	排序的对象	排序的对象为数值型，也可以是字符型。
<b>decreasing</b>	排序的顺序	默认设置为FALSE，即升序排序。设置为TRUE时，为降序排序。
<b>na.last</b>	是否将缺失值放到序列的最末尾。	默认设置为FALSE，设置为TRUE时将向量中的NA值放到序列的最末尾。

# 矩阵和数组

---

- **向量vector**用于描述一维数据，是R语言中最基础的数据结构形式。
- 利用**矩阵matrix**可以描述二维数据，和向量相似，其内部元素可以是实数、复数、字符、逻辑型数据。  
矩阵matrix使用两个下标来访问元素， $A[i,j]$ 表示矩阵A第i行、第j列的元素。
- **多维数组array**可以描述多维数据。array有一个特征属性叫维数向量（dim属性），它的长度是多维数组的维数，dim内的元素则是对应维度的长度。
- 矩阵是数组的特殊情况，它具有两个维度。

# 矩阵创建

- `matrix()`函数，以向量形式输入矩阵中的全部元素，使用`ncol`和`nrow`设置矩阵的行和列数。
- 注意向量1~10是按列填充的，如果想要以行为单位填充，则可以将参数`byrow`设置为`TRUE`。

- `> (v <- seq(1:10))`

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

- `> (a <- matrix(v,nrow = 5,ncol = 2))`

```
      [,1] [,2]
```

```
[1,]    1    6
```

```
[2,]    2    7
```

```
[3,]    3    8
```

```
[4,]    4    9
```

```
[5,]    5   10
```

- `> (a <- matrix(v,nrow = 5,ncol = 2,byrow = T))`

```
      [,1] [,2]
```

```
[1,]    1    2
```

```
[2,]    3    4
```

```
[3,]    5    6
```

```
[4,]    7    8
```

```
[5,]    9   10
```

# 矩阵创建

- 在创建矩阵时，也可以使用dimnames参数设置行和列的名称。
- Matrix(向量名，nrow=行数，ncol=列数，byrow=TRUE/FALSE, dimnames=list(行名称，列名称))

- (a <- matrix(v,nrow = 5,ncol = 2,byrow = T,dimnames = list(paste0('r',1:5),paste0('n',1:2))))

```
n1 n2  
r1 1 2  
r2 3 4  
r3 5 6  
r4 7 8  
r5 9 10
```

	C1	C2	C3	C4	C5	C6
R1	1	6	11	16	21	26
R2	2	7	12	17	22	27
R3	3	8	13	18	23	28
R4	4	9	14	19	24	29
R5	5	10	15	20	25	30

- a<-(1:30)
- dim1<-c( "R1" ," R2" ," R3" ," R4" ," R5" )
- dim2<-c( "C1" ," C2" ," C3" ," C4" ," C5" ," C6" )
- a<-matrix(a, nrow=5,ncol=6,byrow=FALSE,dinames=list(dim1,dim2))



# 矩阵的合并

- 函数`cbind()` 将其自变量横向拼成一个大矩阵，`rbind()` 将其自变量纵向拼成一个大矩阵。
- `cbind()` 的自变量是矩阵或看作列向量的向量时，自变量的高度（行数）应该相等。`rbind()` 类似。如果参与合并的自变量比其变量短，则循环不足后合并。

- `> (x1 <- cbind(c(1,2),c(3,4)))`

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

- `> (x1 <- rbind(c(1,2),c(3,4)))`

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

- `> cbind(x1, 1)`

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

# 矩阵的拉直

---

➤ 设A是一个矩阵，则函数`as.vector(A)`可以将矩阵转化为向量。如：

- `> (A <- matrix(1:6,2,3))`

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

```
[1,]  1   3   5
```

```
[2,]  2   4   6
```

- `> as.vector(A)`

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

# 矩阵的行或列计算的函数

- `colSums()` 对矩阵各列求和
- `colMeans()` 求矩阵各列的均值
- `rowSums()` 对矩阵各行求和
- `rowMeans()` 求矩阵各行的均值
- `dim(矩阵名)` 显示矩阵的行列数
- `str(矩阵名)` 显示对象结构
  
- `> a_matrix <- matrix(1:10,nrow = 5,ncol = 2)`
- `> dim(a_matrix)`  
[1] 5 2
- `> nrow(a_matrix)`  
[1] 5
- `> ncol(a_matrix)`  
[1] 2

- `> (A <- matrix(1:16,4,4))`  
[1,] [2,] [3,] [4,]  
[1,] 1 5 9 13  
[2,] 2 6 10 14  
[3,] 3 7 11 15  
[4,] 4 8 12 16
- `> colSums(A)`  
[1] 10 26 42 58
- `> colMeans(A)`  
[1] 2.5 6.5 10.5 14.5
- `> rowSums(A)`  
[1] 28 32 36 40
- `> rowMeans(A)`  
[1] 7 8 9 10

# 矩阵的运算

➤ R语言中有丰富的矩阵运算的函数，包括四则运算、对矩阵各行列的求和、对矩阵各行列的求均值、转置等。下表列出了R语言中一部分常用的用于矩阵运算的函数。

函数	功能
<code>+ - * /</code>	四则运算，要求矩阵的维数相同，对对应位置的各元素进行运算
<code>colSums()</code>	对矩阵的各列求和
<code>rowSums()</code>	对矩阵的各行求和
<code>colMeans()</code>	对矩阵的各列求均值
<code>rowMeans()</code>	对矩阵的各行求均值
<code>t()</code>	对矩阵的行列进行转置
<code>det()</code>	求解方阵的行列式
<code>outer()</code>	求解矩阵的外积（叉积）
<code>%*%</code>	矩阵乘法，要求第一个矩阵的列数与第二个矩阵的行数相同
<code>diag()</code>	对矩阵取对角元素，若对象为向量，则生成以向量为对角元素的对角矩阵
<code>solve()</code>	对矩阵求解逆矩阵，要求矩阵可逆



# 矩阵索引

---

- `x[i, j]` 下标为(i,j)的元素
- `x[i, ]` 第i行
- `x[, j]` 第j列
- `x[, c(1, 3)]` 第 1和3列
- `x["name", ]` 名为"name"的行

矩阵名[行位置常量, 列位置常量]

矩阵名[行位置常量1: 行位置常量2, ,列位置常量1: 列位置常量2]

矩阵名[c(行位置常量列表, c(列位置常量列表))]

# 访问矩阵中的元素

- Claimedata

	Holderage	vehicleage	claimamt	claims
1	22	1	2312	8
2	22	2	2256	8
3	23	3	1064	4
4	23	4	1280	1

Claimedata[2,3]      2256

Claimedata[1:2,1:3]

	Holderage	vehicleage	claimamt
1	22	1	2312
2	22	2	2256

a<-(1:2)

Claimdata[a,c(1,3)]

	Holderage	claimamt
1	22	2312
2	22	2256

# 访问矩阵中的元素

- Claimdata

	Hol derage	vehi cl eage	cl ai mamt	ncl ai ms
1	22	1	2312	8
2	22	2	2256	8
3	23	3	1064	4
4	23	4	1280	1

Claimdata[2,] #访问第2行上的所有元素

Claimdata[c(1,3),] #访问第1,3行上的所有元素

a<-c(TRUE,FALSE,TRUE)

Claimdata[a,] #访问第1,3行上的所有元素以及第4行默认显示（因未指定其逻辑值）

Claimdata[,1:3] #访问第1,3列上的所有元素

访问指定位置除外的元素，需在位置参数前添加负号“-”。

# 练习

---

- 建立下图所示的矩阵：

	v1	v2	v3	v4
u1	1	2	7	10
u2	3	4	8	11
u3	5	6	9	12

- 对第2行、第3列进行索引

# 数组创建

数组是矩阵的扩展，它把数据的维度扩展到两个以上。可以通过函数`array()`方便地创建数组。

➤ 上面表示建立一个三维数据的数组，其维度是 $2 \times 5 \times 3$ 。在结果中会依次展示3个2行5列的矩阵。

```
> (w_array <- array(1:30,dim = c(2,5,3)))
```

```
,, 1
```

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

```
[1,]  1   3   5   7   9
```

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

```
,, 2
```

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

```
[1,] 11  13  15  17  19
```

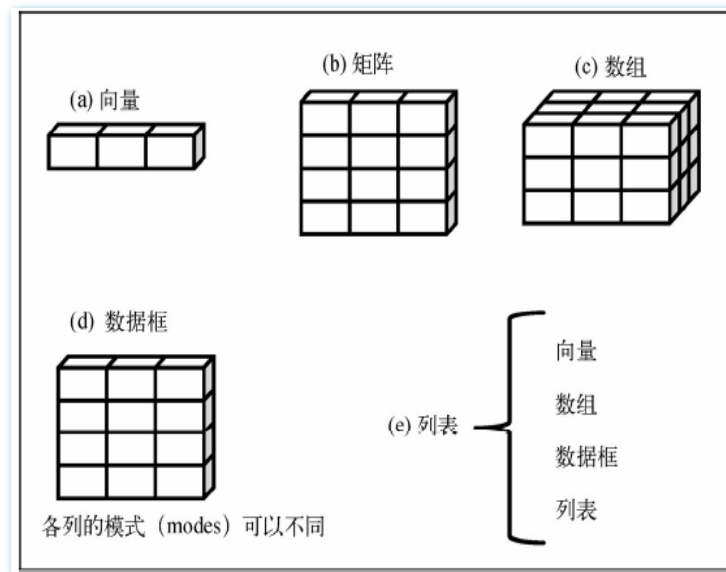
```
[2,] 12  14  16  18  20
```

```
,, 3
```

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

```
[1,] 21  23  25  27  29
```

```
[2,] 22  24  26  28  30
```



# 数组行、列和维度

---

➤ 对于矩阵和数组，dim 函数将返回其维度的整数值向量：

- `> w_array <- array(1:30,dim = c(2,5,3))`
- `> dim(w_array)`  
[1] 2 5 3



# 数据框创建

- 数据框是仅次于向量的最重要的数据对象类型。
- 在实际操作中，通常会用数据框的一列代表某一变量属性的所有取值，用一行代表某一样本数据。
- `data.frame ( )` 函数可以直接把多个向量建立一个数据框，并为列设置名称。

- ```
> (my.datasheet <- data.frame(site = c('A','B','A','A','B'),season =  
c('winter','summer','summer','spring','fall'), pH = c(7.4,6.3,8.6,7.2,8.9)))
```

|   | site | season | pH  |
|---|------|--------|-----|
| 1 | A    | winter | 7.4 |
| 2 | B    | summer | 6.3 |
| 3 | A    | summer | 8.6 |
| 4 | A    | spring | 7.2 |
| 5 | B    | fall   | 8.9 |

`data.frame(域名1=向量名1, 域名2=向量名2,...)`

- 可以通过`names(<数据框>)`来读取并编辑列名称。

- ```
> names(my.datasheet)  
[1] "site" "season" "pH"
```
- ```
> names(my.datasheet)[1] <- 'type'
```
- ```
> names(my.datasheet)  
[1] "type" "season" "pH"
```



# 数据框索引

## ➤ 索引列

列名称索引	列下标索引
数据框对象\$列名称 数据框对像[["域名"]] 数据框对像[[域名编号]]	数据框对象[[列下标]] 数据框对象[,列下标] 数据框对象[,列 下标向量]

返回向量

	site	season	pH
1	A	winter	7.4
2	B	summer	6.3
3	A	summer	8.6
4	A	spring	7.2
5	B	fall	8.9

```
> my.datasheet$site #访问site域
[1] A B A A B
Levels: A B
> my.datasheet[[3]] #访问域3
[1] 7.4 6.3 8.6 7.2 8.9
> my.datasheet[["pH" ]]
[1] 7.4 6.3 8.6 7.2 8.9
```

## ➤ 索引行

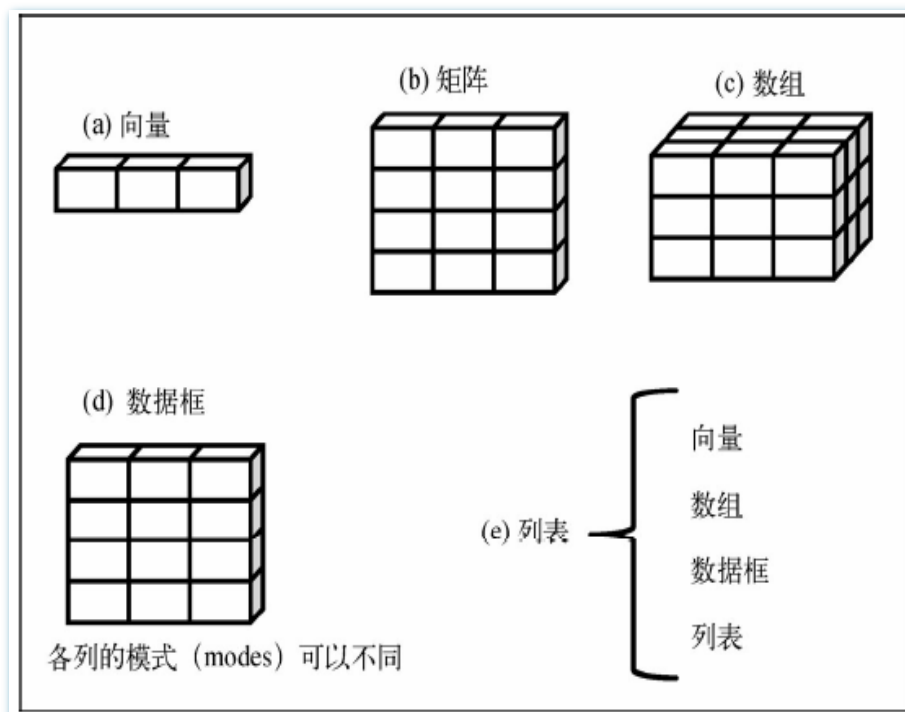
通过<数据框对象>[行下标, ], 可以直接获取相应行的所有元素，并以数据框对象形式返回。例如：

```
> my.datasheet[1:2,]
  site season pH
1  A winter 7.4
2  B summer 6.3
```



# 列表

- 一般地，在使用R语言进行数据分析和挖掘的过程中，向量和数据框的使用频率是最高的，list则在存储较复杂的数据时作为数据对象类型。



# 列表创建

➤ list ( ) 可以用于创建列表对象。列表可包含向量、矩阵、数组、数据框等。

- > (my.list <- list(stud.id = 34453, stud.name = '张三', stud.marks = c(14.3,12,15,19)))

```
$stud.id  
[1] 34453
```

```
$stud.name  
[1] "张三"
```

```
$stud.marks  
[1] 14.3 12.0 15.0 19.0
```

List (成分名1=对象名1, 成分名2=对象名2,...)

➤ 对象my.list由三个成分组成：第一个是名称为stud.id的数值，第二个是名称为stud.name的字符串，第三个是名称为stud.marks的数值向量。

# 列表创建

---

➤ 可以使用函数length ( ) 来检查列表成分的个数:

- > length(my.list)  
[1] 3

➤ 可以通过函数unlist ( ) 把列表中的所有元素转换为向量元素，转换后的向量元素的个数和列表中所有数据对象的个数相同。

- > unlist(my.list)  
stud.id stud.name stud.marks1 stud.marks2 stud.marks3 stud.marks4  
"34453" "张三" "14.3" "12" "15" "19 "

# 列表索引

---

- `x[n]` 列表显示元素n
- `x[[n]]` 列表的第n个元素
- `x[["name"]]` 名为" name"的元素
- `x$name` 同上.

# 因子

- 因子是一种特殊形式的向量，由于一个向量可视为一个变量，如果该变量的计量类型为分类型，则将对应的向量转换为因子，以利于后续的数据分析。

顺序分类型变量                  名义分类型变量

- 为了便于区分，R要求将具有k个类别的分类型变量所对应的向量转换成因子。
- 因子的储存类型为整数型。
- 通常情况下，在创建数据框变量时，R隐式把数据类型为字符的列创建为因子，这是因为R会把文本类型默认为类别数据，并自动转换为因子。

```
heights <- data.frame(height_cm=c(156,182,170),gender=c('f','m','f'))
```

```
class(heights$gender)
```

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

# 创建因子

➤ 通过factor()函数创建因子

➤ 使用格式：

factor(x = character(), levels=c(类别值列表), labels = c(类别值列表), ordered = TRUE/FALSE)

↓  
向量名

参数	描述
x	表示需要创建为因子的数据，是一个向量
levels	表示所创建的因子数据的水平，如果不指定的话，就是x中不重复的所有值
labels	用来标识这一水平的名称，与水平一一对应，方便用户识别
ordered	一个逻辑值，若为TRUE,表示有序因子，为FALSE则表示无序因子

```
p<-c("poor","improved","excellent","poor")
```

```
(b<-factor(p,ordered=FALSE,levels=c("poor","improved","excellent")))
```

 #指定类别值和水平值的对应关系

```
[1] poor    improved excellent poor
```

```
Levels: poor improved excellent
```

```
(b<-factor(p,ordered=TRUE,levels=c("poor","improved","excellent")))
```

```
[1] poor    improved excellent poor
```

```
Levels: poor<improved<excellent
```

# 因子水平

- factor( )将p向量的元素进行分类，创建因子b。
- 因子水平规定了因子取值的范围，每一个因子，都包含因子水平的信息。

```
> p<-c("poor","improved","excellent","poor")  
> (b<-factor(p,levels=c("poor","improved","excellent"),labels=c("C","B","A")))
```

```
[1] C B A C
```

```
Levels: C B A
```

```
> nlevels(b) #水平的级数，相当于level的长度，可以由nlevels函数查询
```

```
[1] 3
```


```
> sex <- factor(c('f','m','f','f','m'),levels=c('f','m'))
```

```
> sex
```

```
[1] f m f f m
```

```
Levels: f m
```

为每个因子水平添加标签



```
> (sex=factor(c('f','f','f','f','m'),levels=c('f','m'),labels=c('female','male'),ordered=TRUE))
```

```
[1] female male  female female male
```

```
Levels: female < male
```

```
> levels(sex) #按因子水平的升序显示它们对应的类别值
```

```
[1] "female" "male"
```

# 有序因子

---

- 因子的顺序，实际上是指因子水平的顺序。
- 通常情况下，因子中先出现的水平小于后出现的水平。
- 按照指定的levels，转换成有序因子。

```
>(sex=factor(c('m','f','f','f','m'),levels=c('f','m'),labels=c('male','female'),ordered=TRUE))  
[1] female male  male  male  female  
Levels: male < female
```

```
> (sex=factor(c('f','f','f','f','m'),levels=c('f','m'),labels=c('female','male'),ordered=TRUE))  
[1] female male  female female male  
Levels: female < male
```



# 向量和矩阵之间的互换

`as.matrix(向量名)`      `as.vector(矩阵名)`

- 向量转换成矩阵时，矩阵默认只有1列，其行数等于原向量包含的元素的个数。
- 矩阵转换成向量时，默认以列为单位依次从左至右读取矩阵数据到向量中。

```
(a<-c(1:6))
```

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

```
(b<-matrix(a,nrow=3,ncol=2,byrow=TRUE))
```

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

```
(a<-as.matrix(a)) #转换成6行一列的矩阵
```

```
(b<-as.vector(b))
```

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

# 向量转换成因子

```
as.factor(向量名 )
```

```
> (a<-c( "poor" , " improved" , " excellent" , " poor" ))
```

```
[1] "poor" "improved" " excellent" "poor"
```

```
> b<-as.factor(a) )
```

```
Levels: excellent improved poor
```

```
>is.factor(b) TRUE
```

```
>levels(b)
```

```
[1] "excellent " "improved" "poor" #按字母升序显示对应的类别值
```

```
>nlevels(b) [1] 3
```

```
>typeof(b) [1] "integer"
```

缺点：

- 水平值和类别值的对应关系是按照类别值的字母升序对应的，字母顺序较小的对应较小的水平值。
- as.factor()函数得到的因子，总是名义分类变量，无法体现顺序分类型变量。

# 因子转换成向量

---

```
as.vector(因子名 )
```

```
(a<-c( "A" , "C" , "B" , "C" ))
```

```
(b<-as.factor(a)
```

```
[1] A C B C
```

```
Levels: A B C
```

```
< b [5]<-" D" # 不允许直接在因子b中添加一个水平（对应 "D" 类别值），所以需要因子转换成向量。
```

```
c<-as.vector(b)
```

```
typeof(c)
```

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

```
c[5]<-" D"
```

```
(b<-as.factor(c)      [1] A C B C D  levels: A B C D
```



# 因子型数据的储存形式

➤ 通过下面的例子，可以加深理解因子的储存形式。

```
> chr <- c('R','Python','R','Ruby','Lisp','R')
```

```
> class(chr)
```

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

```
> (f <- as.factor(chr))
```

```
[1] R    Python R    Ruby  Lisp  R
```

```
Levels: Lisp Python R Ruby
```

```
> class(f)
```

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

```
> storage.mode(f)
```

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

```
> as.numeric(f)
```

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

```
> levels(f)
```

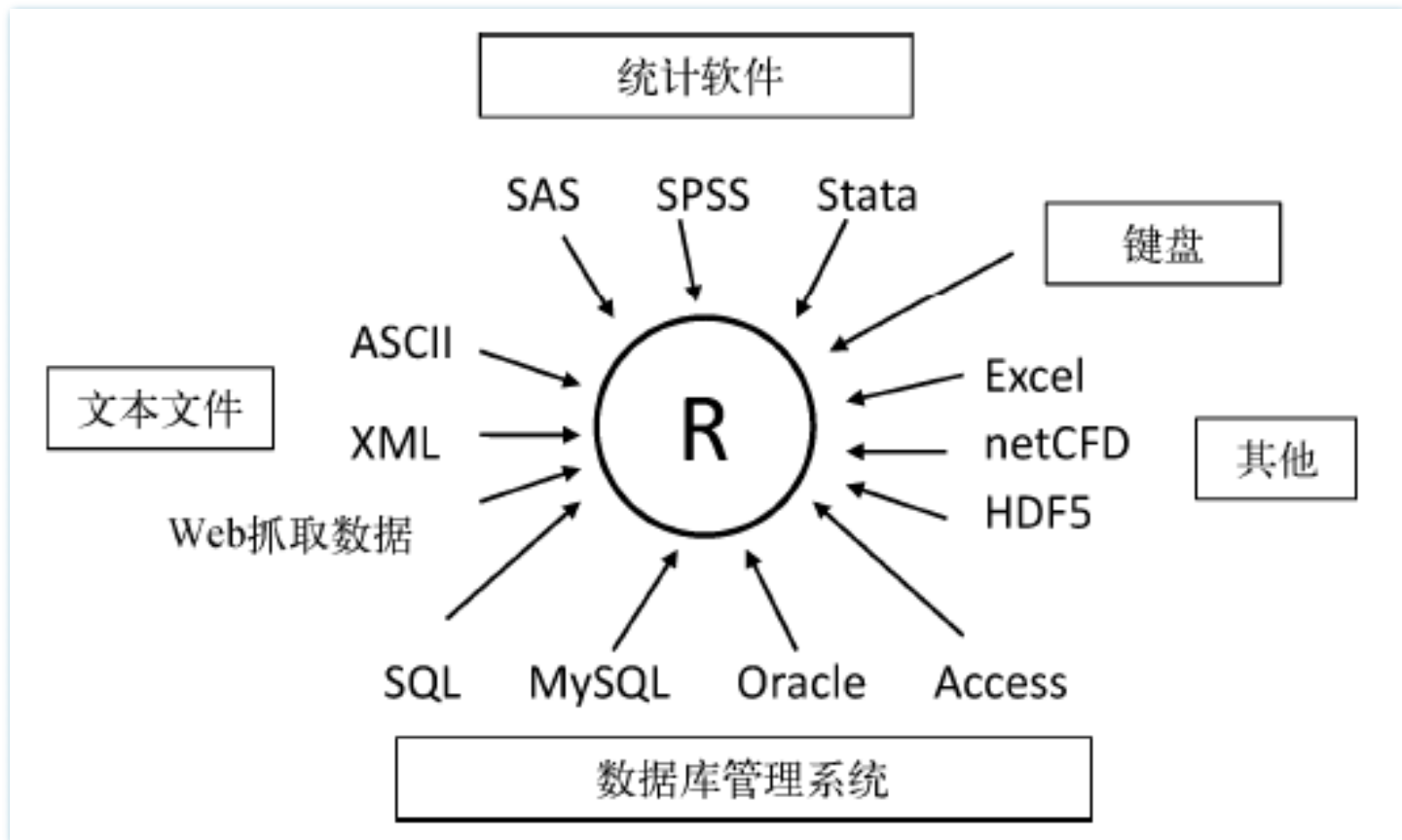
```
[1] "Lisp" "Python" "R"    "Ruby"
```

通过这个例子，可以知道f的类型是整数形式的，并且1对应的是“Lisp”，2对应的是“Python”，3对应的是“R”，4对应的是“Ruby”，这些是按照字母顺序排序的。



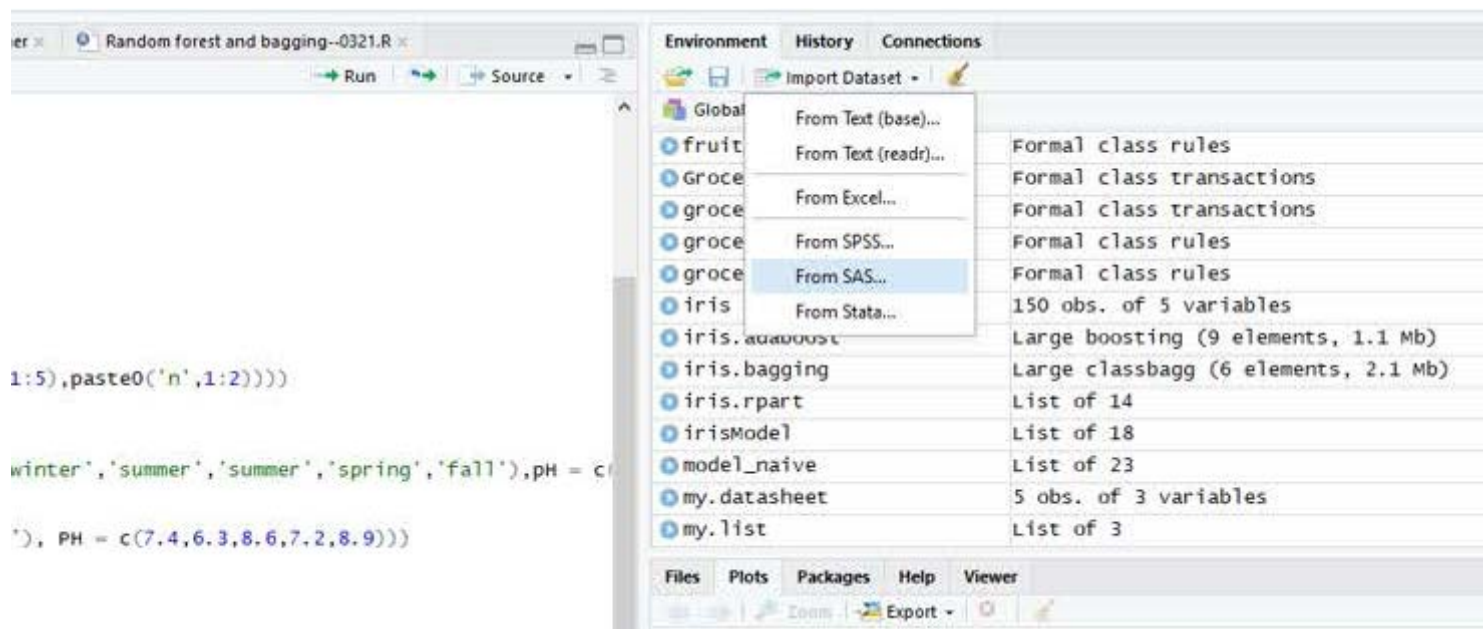
## 可供R导入的数据源

- R可以从键盘、文本文件、Microsoft Excel和Access、流行的统计软件、特殊格式的文件，以及多种关系型数据库中导入的数据。



# 数据的导入

- 使用Rstudio编辑器提供的简单的数据导入功能：



- 使用它可以从txt、csv等文本格式的文件或者从网络中获取数据。



## 读取文本文件和csv文件到数据框

---

- 读取文本文件：**read.table**

`read.table ( file= "文件名" , header=TRUE/FALSE, sep= "数据分隔符" )`

```
file1<-read.table(file="E:/teaching/Data Mining/R codes/redwine.txt",header=TRUE,sep=" ")
file1
str(file1)
```

- 读取CSV文件：**read.csv** 分隔符默认设置为逗号，并假设数据有标题行。

`read.table ( file= "文件名" , header=TRUE/FALSE, sep= "数据分隔符" )`

```
file2 <- read.csv("E:/teaching/Data Mining/ExperimentRcodes/decisiontree/Telephone.csv")
file2
```



## 读取文本文件和csv文件到数据框

<code>file</code>	文件名（包在""内，或使用一个字符型变量），可能需要全路径（注意即使是在Windows下，符号\也不允许包含在内，必须用/替换），或者一个URL链接（http://...）（用URL对文件远程访问）
<code>header</code>	一个逻辑值(FALSE or TRUE)，用来反映这个文件的第一行是否包含变量名
<code>sep</code>	文件中的字段分离符，例如对用制表符分隔的文件使用 <code>sep="\t"</code>
<code>quote</code>	指定用于包围字符型数据的字符
<code>dec</code>	用来表示小数点的字符
<code>row.names</code>	保存着行名的向量,或文件中一个变量的序号或名字,缺省时行号取为1, 2, 3, ...
<code>col.names</code>	指定列名的字符型向量(缺省值是: V1, V2, V3, ...)
<code>as.is</code>	控制是否将字符型变量转化为因子型变量(如果值为FALSE)，或者仍将其保留为字符型(TRUE)。as.is可以是逻辑型，数值型或者字符型向量，用来判断变量是否被保留为字符。
<code>na.strings</code>	代表缺失数据的值(转化为NA)
<code>colClasses</code>	指定各列的数据类型的一个字符型向量
<code>nrows</code>	可以读取的最大行数(忽略负值)
<code>skip</code>	在读取数据前跳过的行数
<code>check.names</code>	如果为TRUE，则检查变量名是否在R中有效
<code>fill</code>	如果为TRUE且非所有的行中变量数目相同，则用空白填补
<code>strip.white</code>	在sep已指定的情况下，如果为TRUE，则删除字符型变量前后多余的空格
<code>blank.lines.skip</code>	如果为TRUE，忽略空白行
<code>comment.char</code>	一个字符用来在数据文件中写注释，以这个字符开头的行将被忽略（要禁用这个参数，可使用 <code>comment.char = ""</code> ）

# 写入文件

---

- 与此相反的任务是写入文件，write.table 和write.csv 分别对应着read.table 和read.csv 的读操作。

```
write.table(x, file = ".....")
```

```
write.csv(x, file = ".....")
```

```
write.csv(file2,file="E:/teaching/file2.csv")
```

# 按行读取文本文件

---

- 如果文件的结构松散，更简单的做法是：先读入文件中的所有文本行，再对其内容进行分析或操作。

`readLines`（注意大写字母L）就提供了这种方法。它接受一个文件路径（或文件连接）和一个可选的最大行数作为参数来读取文件。`readline()` 每次返回一行，字符串变量。`readLines()`列表形式返回全文，每行作为一个字符串作为列表元素。

```
a<-readline("请输入a参数的值：")
```

```
is.character(a)
```

```
a<-as.numeric(readline("请输入a参数的值："))
```

```
file3<-readLines("E:/teaching/Data mining/陋室铭.txt",n=2)
```

```
file3
```



# 导入Excel数据

---

- 将Excel文件转换成CSV文件，再读入。
- 使用read\_excel()读取

```
library(readxl)  
file4<- read_excel ("E: /teachi ng/车险数据. xl sx", 1)  
file4
```

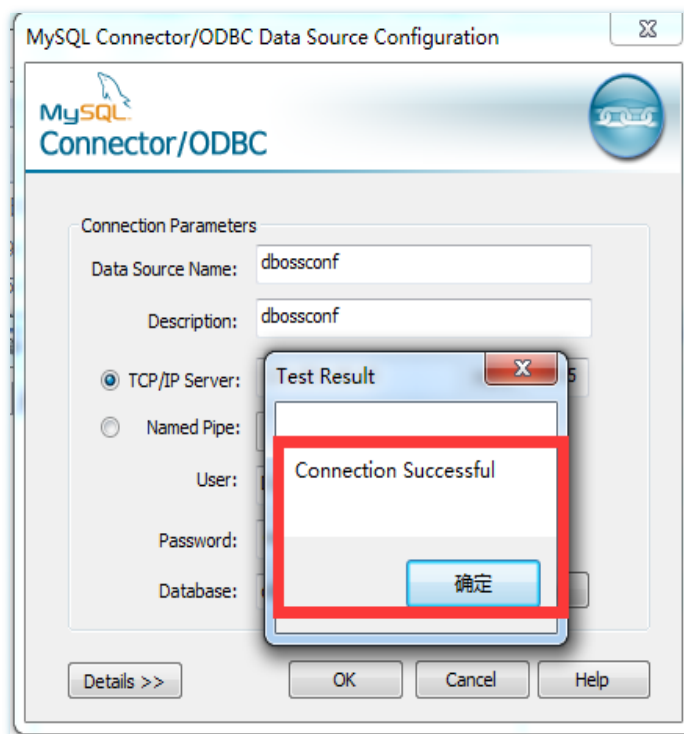
## 其他文件读取

- 由于某些原因，可能需要从其他格式的文件中读入数据，比如SAS的数据文件、SPSS的数据文件等。下表列出了foreign包中读取外部数据的函数。

函数	描述
<code>read.arff</code>	从ARFF文件中读取文件，著名的数据挖掘开源软件weka的数据就是这种格式
<code>read.dbf</code>	读取DBF文件，DBF文件就是数据库文件
<code>read.dta</code>	读取Stata中的数据
<code>read.epiinfo</code>	读取Epi Info的数据
<code>read.mtp</code>	读取Minitab中的数据
<code>read.octave</code>	读取Octave的文本数据
<code>read.spss</code>	读取SPSS的数据文件
<code>read.ssd</code>	读取SAS的永久数据集
<code>read.systat</code>	读取Systat格式的数据

# 访问数据库管理系统

- R中有多种面向关系型数据库管理系统(DBMS)的接口，包括SQL Server、Access、MySQL、Oracle、DB2等。其中一些包通过原生的数据库驱动来提供访问功能，另一些则是通过ODBC或JDBC来实现访问的。
- 通过RODBC包访问一个数据库。这种方式允许R连接到任意一种拥有ODBC驱动的数据库，其实几乎就是市面上的所有数据库。
- 针对选择的数据库安装并配置好驱动：



# RODBC包中的主要函数

RODBC包是扩展包，可以通过命令install.packages(“RODBC”)来安装它。

函数	描述
<code>odbcConnect(dsn,uid="",pwd="")</code>	建立一个到ODBC数据库的连接
<code>sqlFetch (channel,sqltable)</code>	读取ODBC数据库中的某个表到一个数据框中
<code>sqlQuery(channel,query)</code>	想ODBC数据库提交一个查询并返回结果
<code>sqlSave(channel, mydf, tablename = sqltable, append = FALSE)</code>	将数据框写入或更新(append=TRUE)到ODBC数据库的某个表中
<code>sqlDrop(channel,sqltable)</code>	删除ODBC数据库中的某个表
<code>close(channel)</code>	关闭连接

## 案例演示：连接mysql数据库

- 首先加载RODBC包，并通过一个已经配置好的数据库(ids\_user\_action)和用户名(Daniel.xie)以及密码(xie@iedlan)打开一个数据库的连接。

```
> library(RODBC)
> channel <- odbcConnect("ids_user_action", "Daniel.xie", "xie@iedlan")
>
> odbcGetInfo(channel)
          DBMS_Name          DBMS_Ver
          "MySQL"          "5.5.27-log"
Driver_ODBC_Ver  Data_Source_Name
          "03.80"          "ids_user_action"
      Driver_Name      Driver_Ver
      "myodbc5w.dll"      "05.03.0004"
          ODBC_Ver      Server_Name
          "03.80.0000" "121.201.10.15 via TCP/IP"

> |
```



## 案例演示：读取mysql中的某个表

- 利用sqlFetch命令读取Mysql数据库中的channels\_categories表

```
> channels_categories<-sqlFetch(channel,"channels_categories")
> str(channels_categories)
'data.frame':   4681 obs. of  3 variables:
 $ id          : int  6511 2 3 4 5 6 790 8 9 10 ...
 $ channel_id  : int  10000 10001 10002 10003 10004 10005 10006 10008 10009 10$
 $ category_id: int   2 2 2 2 2 2 1 2 2 2 ...
> head(channels_categories)
   id channel_id category_id
1 6511      10000           2
2    2      10001           2
3    3      10002           2
4    4      10003           2
5    5      10004           2
6    6      10005           2
> |
```

## 案例演示：向mysql数据库提交一个查询并返回结果

- 利用sqlQuery命令向mysql数据库提交一个查询并返回结果

```
> mydata <- sqlQuery(channel,"select * from channels_categories limit 10")
> mydata
```

	id	channel_id	category_id
1	6511	10000	2
2	2	10001	2
3	3	10002	2
4	4	10003	2
5	5	10004	2
6	6	10005	2
7	790	10006	1
8	8	10008	2
9	9	10009	2
10	10	10010	2

# 读取网络数据

---

网络数据正在逐渐增多。R中有若干用于抓取网络数据的包。

## quantmod包

- quantmod包是R平台用于金融建模的扩展包主要功能有：从多个数据源获取历史数据、绘制金融数据图表、在金融数据图表中添加技术指标、计算不同时间尺度的收益率、金融时间序列分析、金融模型拟合与计算等等。

## XML包

- XML包包含了一些抓取网络数据的常用函数。对于网络数据，最简单的形式是网络上的表格数据，这种数据通过复制黏贴可以直接粘贴到Excel中。在R中我们也可以很容易将其直接抓取成数据框。

## RCurl包

- RCurl提供了由R到libcurl库的接口，从而实现HTTP的一些功能。例如，从服务器下载文件、保持连接、上传文件、采用二进制格式读取、句柄重定向、密码认证等等。