数据结构

数值型：数值可以用于直接结算，加减乘除
字符串型：可以进行连接、转换、提取等
逻辑型：或真或假
日期型等

向量

数值型：数值可以用于直接结算，加减乘除
字符串型：可以进行连接、转换、提取等
逻辑型：或真或假
日期型等

向量索引

1、 正（负）整数索引

可以使用 length()访问向量的个数，访问向量 x 的第一个值可以使用 x[1]；还可以使用负整数进行索引，表示访问除了这一行的其他行数据，比如不访问一个三行数据的第二行，可以使用 x[-2]

2、逻辑向量索引

```
> y <- c(1:10)
> y
 [1]  1  2  3  4  5  6  7  8  9 10
> y [c(T,F,T,T,F,F,T,T,T,F)]
[1] 1 3 4 7 8 9
>
```

以上代码是只输出对应逻辑值为真的值。
还可以进行逻辑判断，例如：

```
> y[y>5 & y<9]
[1] 6 7 8
> y[y>5]
[1]  6  7  8  9 10
```

对于字符串向量，我们可以使用一些特殊的操作符进行逻辑判断，例如“%in%”

```
> "one" %in% z
[1] TRUE
```

也可以用于索引：

```
> z[z %in% c("one","two" )]
[1] "one" "two"
> z %in% c("one","two" )
[1]  TRUE  TRUE FALSE FALSE
>
```

3、名称索引

我们可以使用 names()函数为向量的每一个元素添加名称：

```
> y
 [1]  1  2  3  4  5  6  7  8  9 10
> names(y)
NULL
> names(y) <- c("one","two","three","four","five","six")
> names(y)
 [1] "one"   "two"   "three" "four"  "five"  "six"   NA      NA      NA      NA
> y
  one   two three  four  five   six  <NA>  <NA>  <NA>  <NA>
    1     2     3     4     5     6     7     8     9    10
```

然后即可以通过每一个元素的 names 来访问它的值：

```
>
> y["two"]
two
  2
```

添加向量：

可以直接通过索引来添加向量；

```
> x <- c(1:100)
> x
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26
 [27]  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52
 [53]  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78
 [79]  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
> x[101] <- 101
> x
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26
 [27]  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52
 [53]  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78
 [79]  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101
>
```

也可以一次性添加多个元素：

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

在向量中插入一个新元素：

可以使用 append()函数，以下代码表示，在 x 这个向量的 5 这个元素后面插入一个 99 的元素：

```
 append(x=v,values = 99,after = 5)
 [1]  1  2  3  4  5 99  6 NA NA NA NA NA NA NA NA NA NA NA NA NA  4
```

如果 after=0 则代表在向量的头部插入数据：

```
 append(x=v,values = 99,after = 0)
 [1] 99  1  2  3  4  5  6 NA NA NA NA NA NA NA NA NA NA NA NA NA  4
```

删除向量：

如果想删除整个向量，可以直接使用 rm()函数，如果想删除某个函数，可以直接采用负整数索引的方式：

```
> y[-c(1:3)]
four five  six <NA> <NA> <NA> <NA>
   4    5    6    7    8    9   10
> y <- y[-c(1:3)]
> y
four five  six <NA> <NA> <NA> <NA>
   4    5    6    7    8    9   10
```

其实就是重新生成一个新的向量，替换掉原来的向量

修改向量值：直接将需要修改的值索引出来，然后给它赋一个新的值就可以了。

```
> v
 [1]  1  2  3  4  5  6 NA NA NA NA NA NA NA NA NA NA NA NA NA  4
> v[2]
[1] 2
> v[2]=15
> v
 [1]  1 15  3  4  5  6 NA NA NA NA NA NA NA NA NA NA NA NA NA  4
```

但是注意这里是数值型的向量，我们不能赋值给字符串，会把整个向量变成字符型向量

```
> v
 [1]  1 15  3  4  5  6 NA NA NA NA NA NA NA NA NA NA NA NA NA  4
> class(v)
[1] "numeric"
> v[2]="two"
> v
 [1] "1"   "two" "3"   "4"   "5"   "6"   NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
[19] NA    "4"
> class(v)
[1] "character"
```

向量运算

向量是 R 中最基本的数据结构。

向量运算是对应位置的元素进行运算，其中长的向量的个数必须是短向量个数的整数倍。

Ceiling()不小于 x 的最小整数，floor()函数不大于 x 的最大整数：

```
> ceiling(c(-2.3,3.1415))
[1] -2  4
> floor(c(-2.3,3.1415))
[1] -3  3
```

Trunc()函数返回整数部分：

```
> trunc(c(-2.3,3.1415))
[1] -2  3
```

Round()函数用于进行四舍五入，digits 用来表示返回的位数

```
> round(c(-2.3,3.1415),digits = 2)
[1] -2.30  3.14
```

Signif()保留小数部分有效数字

```
> signif(c(-2.3,3.1415),digits = 2)
[1] -2.3  3.1
```

Which()函数可以返回索引值，也就是元素所在的位置

```
> t <- c(1,4,2,5,7,9,6)
> t
[1] 1 4 2 5 7 9 6
> which.max(t)
[1] 6
>
```

矩阵与数组

矩阵是一个按照长方阵列排列的复数或实数集合，向量是一维的，矩阵是二维的，需要有行和列。在 R 软件中，矩阵是有维数的向量，这里的矩阵元素可以是数值型，字符性改革或者是逻辑型，但是每个元素必须都拥有相同的模式，这个和向量一致。
我们可以通过 matrix()函数来创建矩阵

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

行数和列数的分配要满足分配的条件，如果只给出一个行或者是一个列，R 会自动进行分配，可以通过 byrow 来指定按行还是按列进行排列：

```
> h <- matrix(1:20,nrow=4,byrow = T)
> h
     [,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
> h <- matrix(1:20,nrow = 4,byrow = F)
> H
Error: object 'H' not found
> h
     [,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
```

Dinames 参数可以通过一个列表，指定矩阵行和列的名字：

```
> rnames <- c("r1",'r2','r3','r4')
> cnames <- c('c1','c2','c3','c4','c5')
> dimnames(h) <- list(rnames,cnames)
> h
   c1 c2 c3 c4 c5
r1  1  5  9 13 17
r2  2  6 10 14 18
r3  3  7 11 15 19
r4  4  8 12 16 20
```

Dim()函数可以显示向量的维数，可以通过 dim()函数来对向量添加维数，从而构建矩阵

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

下面介绍一下数组这个数据结构。R 中的数组其实就是多维的矩阵，我们重新定义一个向量 x：

```
> x <- 1:20
> dim(x) <- c(2,2,5)
> x
, , 1

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

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8

, , 3
```

只要向 dim()函数传入三个参数，就可以构建三位数组，数组还可以使用 array()函数来创建：

```
> 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
, , C1

   B1 B2 B3
A1  1  3  5
A2  2  4  6

, , C2

   B1 B2 B3
A1  7  9 11
A2  8 10 12

, , C3

   B1 B2 B3
A1 13 15 17
A2 14 16 18

, , C4

   B1 B2 B3
A1 19 21 23
A2 20 22 24
```

还可以创建字符型和逻辑型的数组，那么创建了矩阵，要如何访问矩阵的数据呢，下面是矩阵的索引：
首先可以通过矩阵下标进行访问，m[1,2]表示访问第一行第二列的元素

```
> m <- matrix(1:20,4,5)
> m
     [,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
> m[1,2]
[1] 5
```

也可以一次访问多个元素，比如访问第一行的第二、三、四列元素，访问第二、三行的第一列元素。

```
> m[1,c(2,3,4)]
[1]  5  9 13
> m[c(2,3),1]
[1] 2 3
```

输出矩阵的一个子集：

```
     [,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
> m[c(2:4),c(2,3)]
     [,1] [,2]
[1,]    6   10
[2,]    7   11
[3,]    8   12
>
```

如果下标只写一个数字，就是单独访问行或者列：

```
> m
     [,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
> m[c(2:4),c(2,3)]
     [,1] [,2]
[1,]    6   10
[2,]    7   11
[3,]    8   12
> m[2,]
[1]  2  6 10 14 18
> m[,3]
[1]  9 10 11 12
>
```

也可以同样使用负索引进行访问，还可以输入对应的行列名称进行访问：

```
state.x77[,"Population"]
     Alabama          Alaska         Arizona
        3615             365            2212
 Connecticut        Delaware         Florida
        3100             579            8277
    Illinois         Indiana            Iowa
       11197            5313            2861
       Maine        Maryland   Massachusetts
        1058            4122            5814
    Missouri         Montana        Nebraska
        4767             746            1544
  New Mexico        New York  North Carolina
        1144           18076            5441
      Oregon    Pennsylvania    Rhode Island S
        2284           11860             931
       Texas            Utah         Vermont
       12237            1203             472
   Wisconsin         Wyoming
        4589             376
```

矩阵中的四则运算需要行和列一致，与向量一致，可以使用 colsums()、rowsums()、rowmeans()等函数对整个矩阵进行计算也可以进行矩阵的乘法：

```
> m
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> t
     [,1] [,2] [,3]
[1,]    2    5    8
[2,]    3    6    9
[3,]    4    7   10
> m*t
     [,1] [,2] [,3]
[1,]    2   20   56
[2,]    6   30   72
[3,]   12   42   90
> m %*% t
     [,1] [,2] [,3]
[1,]   42   78  114
[2,]   51   96  141
[3,]   60  114  168
>
```

分别是矩阵的内积（对应元素相乘）以及矩阵的外积

列表

State.center 就是一个典型的列表，是美国每个周的经纬度，可以使用 list()函数来创建列表：

```
> a <- c(1:20)
> b <- matrix(1:20,4)
> c <- mtcars
> d <- "this is a test list"
```

```
> mlist <- list(a,b,c,d)
> mlist
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

[[2]]
     [,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

[[3]]
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
```

这样就生成了一个列表，我们也可以为每个变量添加一个名称，例如：

```
> mlist <- list(first=a,second=b,thrid=c,forth=d)
> mlist
$first
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

$second
     [,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

$thrid
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
```

列表中的元素不存在顺序，使用名称就可以访问数据，下面介绍一下数据列表的访问
第一种方法是可以使用索引的方式进行访问

```
> mlist[1]
$first
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

> mlist[c(1,4)]
$first
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

$forth
[1] "this is a test list"
```

第二种方法是可以使用名字来进行访问

```
> state.center[c("x",'y')]
$x
 [1]  -86.7509 -127.2500 -111.6250  -92.2992 -119.7730 -10
[10]  -83.3736 -126.2500 -113.9300  -89.3776  -86.0808  -9
[19]  -68.9801  -76.6459  -71.5800  -84.6870  -94.6043  -8
[28] -116.8510  -71.3924  -74.2336 -105.9420  -75.1449  -7
[37] -120.0680  -77.4500  -71.1244  -80.5056  -99.7238  -8
[46]  -78.2005 -119.7460  -80.6665  -89.9941 -107.2560

$y
 [1] 32.5901 49.2500 34.2192 34.7336 36.5341 38.6777 41.59
[13] 40.0495 40.0495 41.9358 38.4204 37.3915 30.6181 45.62
[25] 38.3347 46.8230 41.3356 39.1063 43.3934 39.9637 34.47
[37] 43.9078 40.9069 41.5928 33.6190 44.3365 35.6767 31.38
[49] 44.5937 43.0504
```

另外，列表比之后的数据框多了一种$的访问方式：

```
> mlist$first
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
>
```

对于列表还有一种双中括号的访问方式：

```
> mlist[1]
$first
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

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

两者的差别在于，一个中括号其实输出的是列表的一个子集，它其实还是一个列表，因为如果一个访问多个元素，这些元素的数据类型又不同，那么输出结果只能是列表，当我们使用两个中括号进行输出，那么就是输出数据本身的类型，可以使用 class()函数来测试一下

```
> class(mlist[1])
[1] "list"
> class(mlist[[1]])
[1] "integer"
>
```

两个中括号每次只能访问一个元素，如果要像列表中添加元素，可以使用双中括号进行添加：

```
> length(mlist)
[1] 4
> mlist[[5]] <- c(1:10)
> mlist[5]
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10
```

如果想删除列表中的元素可以使用负索引的方式，然后再赋值给原来的列表：

```
> mlist <- mlist[-5]
> mlist
```

或者是使用 Null 来赋值

```
> mlist[[3]] <- NULL
> mlist
$first
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18

$second
     [,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

$forth
[1] "this is a test list"
```

数据框

数据框是一种表格式的数据结构，数据框子在模拟数据集，与其他统计软件例如 SAS 或者 SPSS 中的数据集的概念一致，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，不同的行业对于数据集的行和列叫法不同。

数据框实际上是一个列表，列表中的元素是向量，这些向量构成数据框的列，每一列必须具有相同的长度，所以数据框市矩形结构，而且数据框的列必须命名。

矩阵和数据框的不同，矩阵必须是同一数据类型，数据框每一列必须为同一类型，每一行可以不同

数据框可以通过 data.frame()函数进行创建

数据框的访问：

直接使用索引

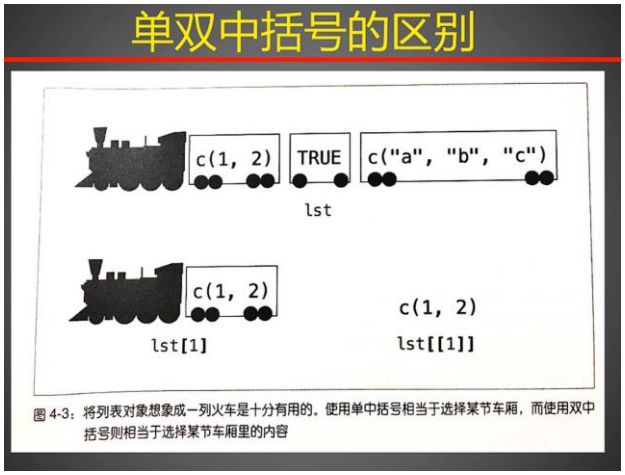
直接使用名称进行索引

可以使用 attach()将数据集存进 R 的内存，这样就便于访问

```
Volvo 142E           21.4    4 121.0 109
> mtcars$mpg
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3
[20] 33.9 21.5 15.5 15.2 13.3 19.2 27.3
> attach(mtcars)
> mpg
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3
[20] 33.9 21.5 15.5 15.2 13.3 19.2 27.3
>
```

使用完之后可以使用 detach()函数取消加载。

数据框也可以使用双中括号的方式进行访问，返回的是向量而不是列表。



因子

在 R 中变量可以分为名义型变量、有序型变量、连续型变量，名义型变量没有顺序的区别，有序型变量介于二者之间，不同值之间有顺序关系。

在 R 中，名义型变量和有序型变量被称为因子，factor，这些分类变量的可能只被称为一个水平，level，例如 good、better、best，都被称为一个 level，由这些水平值构成的向量就称为因子。在很多会吐函数中，输入的数据也必须是因子类型。

例如 mtcars 数据集，cyl 这一列可以作为因子，而 4、6、8 就是这一列因子的水平：

```
> mtcars$cyl
 [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8
> table(mtcars$cyl)

 4  6  8
11  7 14
>
```

那么如何来定义一个因子数据呢？

可以使用 factor 函数：

```
> f <- factor(c("red",'red','green','green','bule','bule'))
> f
[1] red   red   green green bule  bule
Levels: bule green red
```

我们还可以在定义因子的时候人为指定 level 的水平：

```
> week <- factor(c('Mon','Fri','Thu','Wed','Mon','Fri','Sun'),ordered=T,levels =c('Mon','The','Wed','Thu','Fri',"Sat","Sun"))
> week
[1] Mon Fri Thu Wed Mon Fri Sun
Levels: Mon < The < Wed < Thu < Fri < Sat < Sun
>
```

还可以使用 factor()函数将向量转变成因子：

```
> fcyl <- factor(mtcars$cyl)
> fcyl
 [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8
Levels: 4 6 8
```

R 中有一个 cut()函数，可以将连续型变量 x 分割成连续水平的因子：

```
> x <- 1:100
> cut(x,c(seq(0,100,10)))
  [1] (0,10]   (0,10]   (0,10]   (0,10]   (0,10]
 [11] (10,20]  (10,20]  (10,20]  (10,20]  (10,20]
 [21] (20,30]  (20,30]  (20,30]  (20,30]  (20,30]
 [31] (30,40]  (30,40]  (30,40]  (30,40]  (30,40]
 [41] (40,50]  (40,50]  (40,50]  (40,50]  (40,50]
 [51] (50,60]  (50,60]  (50,60]  (50,60]  (50,60]
 [61] (60,70]  (60,70]  (60,70]  (60,70]  (60,70]
 [71] (70,80]  (70,80]  (70,80]  (70,80]  (70,80]
 [81] (80,90]  (80,90]  (80,90]  (80,90]  (80,90]
 [91] (90,100] (90,100] (90,100] (90,100] (90,100
Levels: (0,10] (10,20] (20,30] (30,40] (40,50] (5
```

缺失数据

缺失数据的分类：

缺失数据的分类

统计学家通常将缺失数据分为三类。它们都用概率术语进行描述，但思想都非常直观。我们将用sleep研究中对做梦时长的测量（有12个动物有缺失值）来依次阐述三种类型。

(1) **完全随机缺失** 若某变量的缺失数据与其他任何观测或未观测变量都不相关，则数据为完全随机缺失（MCAR）。若12个动物的做梦时长值缺失不是由于系统原因，那么可认为数据是MCAR。注意，如果每个有缺失值的变量都是MCAR，那么可以将数据完整的实例看做是对更大数据集的一个简单随机抽样。

(2) **随机缺失** 若某变量上的缺失数据与其他观测变量相关，与它自己的未观测值不相关，则数据为随机缺失（MAR）。例如，体重较小的动物更可能有做梦时长的缺失值（可能因为较小的动物较难观察），“缺失”与动物的做梦时长无关，那么该数据就可以认为是MAR。此时，一旦你控制了体重变量，做梦时长数据的缺失与出现将是随机的。

(3) **非随机缺失** 若缺失数据不属于MCAR或MAR，则数据为非随机缺失（NMAR）。例如，做梦时长越短的动物也更可能有做梦数据的缺失（可能由于难以测量时长较短的事件），那么数据可认为是NMAR。

在 R 中，用 NA 代表缺失值，NA 是不可用的意思，用于存储缺失信息，这里缺失值 NA 表示没有，但注意不一定就是 0，NA 是不知道是多少，也可能是 0，也可能是任何值，缺失值和值为零是完全不同的。

但是有时候 NA 值的缺失会导致数据计算出现问题：

```
> a <- c(NA,1:49)
> a
 [1] NA  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
[33] 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
> sum(a)
[1] NA
> mean(a)
[1] NA
>
```

所以此时我们需要剔除 NA 值，可以通过一些函数的 na.rm 选项参数剔除：

```
> sum(a,na.rm = T)
[1] 1225
> mean(a,na.rm = T)
[1] 25
>
```

但是计算平均值时是以 50 个总数来计算还是 49 个总数来计算呢，我们来验证一下这个问题：

```
> mean(1:49)
[1] 25
> mean(1:50)
[1] 25.5
>
```

说明是将 NA 值剔除之后求平均值。

可以在前期数据处理的时候就检查数据集是否存在缺失值，来进行逻辑测试，我们来测试一下 VIM 包中的 sleep 数据集：

```
> library(VIM)
> sleep
    BodyWgt BrainWgt NonD Dream Sleep  Span  Gest Pred Exp Danger
1  6654.000  5712.00   NA    NA   3.3  38.6 645.0    3   5      3
2     1.000     6.60  6.3   2.0   8.3   4.5  42.0    3   1      3
3     3.385    44.50   NA    NA  12.5  14.0  60.0    1   1      1
4     0.920     5.70   NA    NA  16.5    NA  25.0    5   2      3
5  2547.000  4603.00  2.1   1.8   3.9  69.0 624.0    3   5      4
6    10.550   179.50  9.1   0.7   9.8  27.0 180.0    4   4      4
7     0.023     0.30 15.8   3.9  19.7  19.0  35.0    1   1      1
8   160.000   169.00  5.2   1.0   6.2  30.4 392.0    4   5      4
9     3.300    25.60 10.9   3.6  14.5  28.0  63.0    1   2      1
10   52.160   440.00  8.3   1.4   9.7  50.0 230.0    1   1      1
11    0.425     6.40 11.0   1.5  12.5   7.0 112.0    5   4      4
12  465.000   423.00  3.2   0.7   3.9  30.0 281.0    5   5      5
13    0.550     2.40  7.6   2.7  10.3    NA    NA    2   1      2
14  187.100   419.00   NA    NA   3.1  40.0 365.0    5   5      5
```

使用 is.na()函数来测试一下数据集：

```
> is.na(sleep)
      BodyWgt BrainWgt  NonD Dream
 [1,]   FALSE    FALSE  TRUE  TRUE
 [2,]   FALSE    FALSE FALSE FALSE
 [3,]   FALSE    FALSE  TRUE  TRUE
 [4,]   FALSE    FALSE  TRUE  TRUE
 [5,]   FALSE    FALSE FALSE FALSE
 [6,]   FALSE    FALSE FALSE FALSE
 [7,]   FALSE    FALSE FALSE FALSE
```

可以使用 colsums()和 rowsums()来计算每一行的缺失值数目

如果想去除掉数据集中的缺失值，形成一个新的函数，则可以使用 na.omit()函数：

```
> c <- c(NA,1:20,NA,NA)
> c
 [1] NA  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 NA NA
> d <- na.omit(c)
> d
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
attr(,"na.action")
[1]  1 22 23
attr(,"class")
[1] "omit"
>
```

使用 na.omit()函数处理数据框，通常是直接删除缺失的行或者列：

```
> length(rownames(sleep))
[1] 62
> length((rownames(na.omit(sleep))))
[1] 42
>
```

但是这样处理有一个问题，就是当缺失值超过一半的时候，会对分析结果造成很大的影响，所以 R 中有很多处理缺失值的办法：

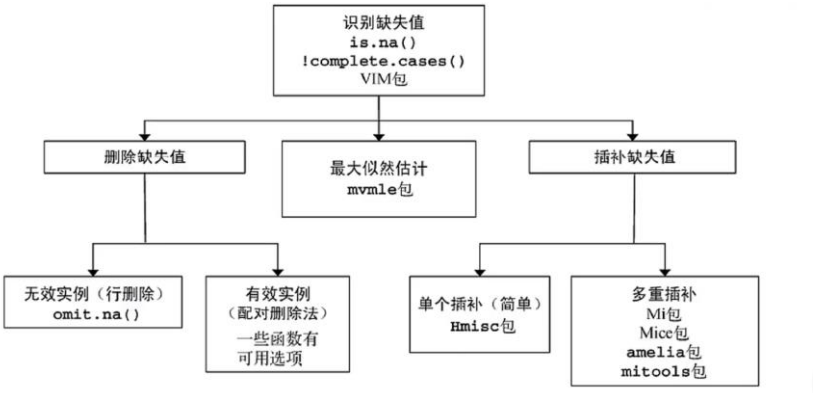


图15-1　处理不完整数据的方法，以及R中相关的包和函数

其他缺失数据：

缺失数据 NaN，代表不可能的值

Inf 表示无穷，分为正无穷 Inf 和负无穷 Inf，代表无穷大或者无穷小。

字符串

nchar()函数可以用来统计字符串的长度：

```
> nchar("hello world")
[1] 11
> month.name
 [1] "January"   "February"  "March"     "April"     "
 [9] "September" "October"   "November"  "December"
> nchar(month.name)
 [1] 7 8 5 5 3 4 4 6 9 7 8 8
>
```

Length()返回向量中元素的个数，而 nchar 返回每个元素字符串的个数：

```
> c <- c(1,"hello world",2,7)
> c
[1] "1"           "hello world" "2"           "7"
> length(c)
[1] 4
> nchar(c)
[1]  1 11  1  1
```

Paste()函数用于粘贴字符串，将多个字符串合并为一个，默认使用空格分割，也可以通过 sep 选项参数来设置分隔符：

```
> paste("i",'love','Arcgis')
[1] "i love Arcgis"
```

```
> paste("i",'love','Arcgis',sep = "-")
[1] "i-love-Arcgis"
```

向量与字符串的连接是向量和字符串分别连接，例如：

```
> paste(names,"loves stats")
[1] "moe loves stats"  "jack loves stats" "lily loves stats"
>
```

Substr()函数用于提取字符串，函数的参数分别是一个原始的字符串，一个起始点和一个结束点，返回值是起始点和结束点之间的字符串。

```
substr(month.name,start = 1,stop = 3)
[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

然后使用 toupper()函数便可以将单词大写，tolower()可以转换为小写。
Grep()函数可以用于查找字符串：

```
> x <- c('b','A+','AC')
> x
[1] "b"  "A+" "AC"
> grep("A+",X,fixed = T)
Error in is.factor(x) : object 'X' not found
> grep("A+",x,fixed = T)
[1] 2
>
```

表示与第二个位置上的字符串匹配上了，如果 fixed 参数为 F，则表示支持正则表达式，那么‘A+’表示匹配一到正无穷个字符 A，那么“AC”也会入选。
Match()函数可以进行字符串匹配
Strsplit()可进行字符串的分割，这个函数需要两个参数，字符串和分割符：

```
> path <- "/user/local/bin/R"
> strsplit(path,"/")
[[1]]
[1] ""      "user"  "local" "bin"   "R"
```

但是这个函数返回的是一个列表，而不是向量。

日期与时间

时间序列分析：
对时间序列的描述
利用前面的结果进行预测

```
>  class(presidents)
[1] "ts"
```

“ts”是 time series 的简称，代表时间序列数据。
在 R 中，日期数据别单独归为一个 date 类，我们可以使用 sys.date()函数查看当前系统的时间：

```
> Sys.Date()
[1] "2021-07-19"
> class(Sys.Date())
[1] "Date"
>
```

在 R 中可以使用 as.date()函数将数据转换为日期数据，使用 format 选项参数决定外观，。比如哪部分作为年，哪部分作为月

```
> t <- as.Date(a,format = "%Y-%m-%d")
> class(t)
[1] "Date"
> t
[1] "2022-01-09"
> a
[1] "2022-01-09"
```

也可以使用 seq()函数创建连续的时间点：

```
> seq(as.Date("2021-01-01"),as.Date("2021-07-05"),by=5)
 [1] "2021-01-01" "2021-01-06" "2021-01-11" "2021-01-16" "2021-01-21" "2021-01-26" "2021-01-31"
 [8] "2021-02-05" "2021-02-10" "2021-02-15" "2021-02-20" "2021-02-25" "2021-03-02" "2021-03-07"
[15] "2021-03-12" "2021-03-17" "2021-03-22" "2021-03-27" "2021-04-01" "2021-04-06" "2021-04-11"
[22] "2021-04-16" "2021-04-21" "2021-04-26" "2021-05-01" "2021-05-06" "2021-05-11" "2021-05-16"
[29] "2021-05-21" "2021-05-26" "2021-05-31" "2021-06-05" "2021-06-10" "2021-06-15" "2021-06-20"
[36] "2021-06-25" "2021-06-30" "2021-07-05"
```

要使用 as.date()系统才会当做时间数据进行处理

使用 ts()函数可以把向量转化为时间序列数据：

```
Time Series:
Start = 2014
End = 2017
Frequency = 1
[1] 75 92 75 75
> ts(sales,start = c(2010,5),end = c(2014,4),frequency = 12)#frequnecy为1代表以年为单位，为12代表以月为单位
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2010                 75  92  75  75  69  71  61  52
2011  63  81  69  82  84  67  76  66  52  90  57  92
2012  90  90  60  99  89  84  51  83  97  77  74  56
2013  97  86  71  78  98  73  76  65  76  90  66  73
2014  97  78  78  51
> ts(sales,start = c(2010,5),end = c(2014,4),frequency = 4)#frequnecy为1代表以年为单位，为12代表以月为单位
     Qtr1 Qtr2 Qtr3 Qtr4
2011   75   92   75   75
2012   69   71   61   52
2013   63   81   69   82
2014   84   67   76   66
```

常见错误

定义向量不要忘记加 c，定义多个数据时记得要使用 c()

使用函数要记得加括号，尽量多使用自动补齐功能

两个等号才是比较是否相等

获取数据

R 获取数据一共有三种途径：

利用键盘来输入数据

通过读取存储在外部文件上的数据

通过访问数据库系统来获取数据

读入文件（一）

R 读取文件

纯文本文件（.csv/.txt）：

通常使用逗号，也可以使用空白分割

Read.table()函数可以读取一个纯文本文件，read.table(file=要读入的文件的名称；sep=指定文件使用的分隔符，默认是空白分割；header=代表在读取数据之后是否将数据的第一行作为变量的名称，而不是当成具体的值来处理，如果是，header=T，不是，header=F；skip=表示读取参数时，跳过部分内容，比如说可以跳过一些介绍性文字；nrows=用于读取文件的行数；na.strings=用于处理缺失值的信息)

```
x <- read.table(file = "input.txt")#需要在工作目录下才可以直接输入名称
```

如果不在工作目录下，可以利用 `setwd()` 函数来更改 R 的工作目录，或者使用文件的全路径：

```
> x <- read.table("D:/工作数据/RData/input.txt")
> head(x)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   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
```

可以使用 head()和 tail()函数读取文件的前几行或者是后几行数据，可以通过函数中选项参数 n 的数值来确定显示的行数

```
> head(x)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   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
> tail(x)
    Ozone Solar.R Wind Temp Month Day
148    14      20 16.6   63     9  25
149    30     193  6.9   70     9  26
150    NA     145 13.2   77     9  27
151    14     191 14.3   75     9  28
152    18     131  8.0   76     9  29
153    20     223 11.5   68     9  30
```

使用 sep 可以指定对数据进行分割，使数据格式变得整洁：

```
> x <- read.table("input.csv")
> head(x)
                V1                                                                                  V2
1                ,"mpg","cyl","disp","hp","drat","wt","qsec","vs","am","gear","carb"
2        Mazda RX4                                                ,21,6,160,110,3.9,2.62,16.46,0,1,4,4
3    Mazda RX4 Wag                                               ,21,6,160,110,3.9,2.875,17.02,0,1,4,4
4       Datsun 710                                              ,22.8,4,108,93,3.85,2.32,18.61,1,1,4,1
5   Hornet 4 Drive                                             ,21.4,6,258,110,3.08,3.215,19.44,1,0,3,1
6 Hornet Sportabout                                             ,18.7,8,360,175,3.15,3.44,17.02,0,0,3,2
> x <- read.table("input.csv",sep=",")
> head(x)
                V1   V2  V3   V4  V5   V6    V7    V8 V9 V10  V11  V12
1                mpg cyl disp  hp drat    wt  qsec vs  am gear carb
2        Mazda RX4  21   6  160 110  3.9 2.620 16.46  0   1    4    4
3    Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0   1    4    4
4       Datsun 710 22.8   4  108  93 3.85 2.320 18.61  1   1    4    1
5   Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1   0    3    1
6 Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0   0    3    2
```

如果提前知道文件格式，也可以直接用 read.csv()进行直接读取：

```
> read.csv("input.csv")
                   X  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
1          Mazda RX4 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
2      Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
3         Datsun 710 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
4     Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
5  Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
6            Valiant 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
7         Duster 360 14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
8          Merc 240D 24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
```

例如需要读取文件前 100 行的数据：

```
> read.table("input 1.txt",header = T,nrows = 100)
    Ozone Solar.R Wind Temp Month Day
1      41     190  7.4   67     5   1
2      36     118  8.0   72     5   2
3      12     149 12.6   74     5   3
```

```
96     78      NA  6.9   86     8   4
97     35      NA  7.4   85     8   5
98     66      NA  4.6   87     8   6
99    122     255  4.0   89     8   7
100    89     229 10.3   90     8   8
```

在此基础上使用 skip 和 nrows 两个参数相结合，就可以读取任意部分的数据，比如读取上述文件的第 10-50 行数据：

```
read.table("input 1.txt",header = T,skip=10,nrows = 50)
```

R 在读取数据时，字符串数据会被默认读取成因子型数据。

读入文件（二）

如果一个纯文本文件并不在本机上，R 可以支持读取网络文件，可以通过一些协议进行读取，只要将 read.table()函数的选项参数 file=文件的网络链接即可。R 会将文件下载到本地。

如何读取非文本文件？

我们可以使用 XML 包进行读取，里面包含一个 readHTMLTable()函数，可以用于读取网页中的数据。

R 中的 foreign 包可以帮助导入其他软件的数据

或者可以直接导入剪切板的内容，直接将 read.table()函数选项参数改为 clipboard 即可：

```
> x <- read.table("clipboard",header = T,sep=",")
```

R 可以直接读取压缩文件，并不需要解压缩：

```
read.table(gzfile("input.txt.gz"))
                   X  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
          Mazda RX4 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
      Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
         Datsun 710 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
     Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
  Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
            Valiant 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
         Duster 360 14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
          Merc 240D 24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
           Merc 230 22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
           Merc 280 19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
          Merc 280C 17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
         Merc 450SE 16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
         Merc 450SL 17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
        Merc 450SLC 15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
 Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
```

如何写入文件

使用 write.table()函数可以将数据写入文件，函数中，write.table(x=写入的数据，file=数据的存储路径及格式”)
如：
x=read.table("input.txt")
write.tale(x,file="H:/Rdata/newfile.txt")
也可以使用 sep 参数确定分隔符
write.table(x,file = "H:/RData/newfile.csv",sep=",")
这样就等同于做了一次数据转换，将文本数据转变成了表格数据
每一次加载数据，R 会自动给每一行数据添加行号，为避免多次打开数据导致行号重复，可使用 row.names 参数
write.table(x,file = "H:/RData/newfile.csv",sep=",",row.names = F)
R 会覆盖同名文件，使用 append 参数可以添加同名参数
R 可以直接写入压缩文件：
write.table(mtcars,gzfile("newfile.txt.gz"))

读写 excel 文件

可以使用两种方式：
将 excel 文件另存为 csv 文件，再使用 read.csv()打开
如:
read.csv("H:/RData/mtcars.csv",header = T)
或者将数据复制到剪贴板，使用 readclipboard()函数将数据导入 R 中，在 R 中打开：
readClipboard()
read.table("clipboard",sep = '\t',header = T)
在 R 中也提供了很多包用于直接读写 excel 数据
比如：xlconnect 包（这个包依赖于 JAVA，需要有 java 的环境），如果该 R 包无法安装，可以使用 openxlsx 包进行读取，如下：
library(openxlsx)
read.xlsx("H:/RData/data.xlsx",sheet = 1)
还可以使用 openxlsx 包的函数创建并写入新的 excel 文件，代码如下：
wb <- createWorkbook()
addWorksheet(wb,sheetName = 1)
x <- mtcars
writeData(wb,sheet = 1,x,startCol = 1,startRow = 1)
write.xlsx(x,"cars.xlsx")
这样就创建了具有 mtcars 数据的 excel 文件。

读写 R 格式文件

R 提供了两种存储的方式，一种时候.Rds 文件，一种是 Rdata 文件。
Rdata 文件类似于工程文件，会存储所有导入的数据集和处理的数据
Rdse 文件是保存数据集的文件，比如 iris 数据
使用 saveRds 命令可以将数据集保存为 Rds 格式
将 readRds 赋值给一个变量 x，可以完成对 Rds 文件的读取。
使用 load()函数可以直接打开 Rdata 文件。

数据转换（一）

首先，使用 openxlsx 包中的 read.xlsx()函数打开 mtcars.xlsx 文件
read.xlsx('mtcars.xlsx',sheet = 1,startRow = 1)

将其赋值为 car32 变量
car32 <- read.xlsx('mtcars.xlsx',sheet = 1,startRow = 1)
可以使用 is.data.frame()判断是否为数据框
is.data.frame(car32)，结果是 ture,说明数据是数据框
is.data.frame()函数还可以将数据转换为数据框格式
unlist()可以用于转化成列表
as.factor、as.vector 可以用来转化成因子和向量。

数据转换（二）

如何对数据取子集？
可以使用索引的方式：
who <- read.csv("WHO.csv",header = T)
取该数据集的前 50 行，10 列
who1 <- who[c(1:50),c(1:10)]
也可以不连续的提取，取该数据集的 1,3,5,8 行，2,14,18 列：
who2 <- who[c(1,3,5,8),c(2,14,18)]
还可以使用逻辑值来进行筛选，比如使用 which 函数取出 who 数据集中 continent 列的值等于 7 的数据集合：
who3 <- who[which(who$Continent==7),]
还可以使用逻辑值的设置范围进行取值：
who4 <- who[which(who$CountryID>50 &who$CountryID<=100)，]
取出 who 数据集中 CountryID 列的值在 50 到 100 之间的数据集合
可以直接使用 subset()函数进行子集的提取
who4 <- subset(who,who$CountryID>50 &who$CountryID<=100)
在 R 中可以使用 sample 函数进行随机抽样
x <- 1:100
（设置一个 x 样本，数据范围在 1 到 100）
sample(x,30)
（随机取 x 中的 30 个样本）
sample(x,30,replace = T)
（随机取 x 中的 30 个样本，但是是有放回的抽样，也就是说样本中可以有重复数字出现）
sample 函数用于数据框时，如下
who[sample(who$CountryID,30,replace =F),]
随机取出了一个子集
如何删除固定行？最简单的就是使用负索引的方式，如下
mtcars[-1:-5,] 删除对应的列，逗号在前，删除对应的行
将列的值赋值给 NULL，相当于清空该列的值
mtcars$mpg <- NULL
数据框如何进行添加与合并？
最简单的方法是使用 data.frame（）直接生成一个新的数据框
data.frame(USArrests,state.division)
如果单纯的想添加一列，可以用 cbind 函数
cbind(USArrests,state.division)
直接在数据后面添加一列 state.division 的数据
但是添加行不容易，因为使用 Rbind()必须两者的列名是一样的，下面进行一个示范：

data1 <- head(USArrests,20) 取出前 20 行数据
data2 <- tail(USArrests,20) 取出后 20 行数据
rbind(data1,data2) 将两个行合并
在使用 cbind 和 rbind 的时候，数据必须有相同的行数和列数
如果数据集中有重复的数据应该如何处理呢？
假设取一个 50 个数据量的数据集的两个子集，容量分别为 30，重复的数据有 10
data1 <- head(USArrests,30)
data2 <- tail(USArrests,30)
使用 rbind 合并
data4 <- rbind(data1,data2)
使用 duplicated(data4)判断哪些是重复值，取出重复值：
data4[duplicated(data4),]
加感叹号取出非重复的部分 （感叹号是取反的意思）
data4[!duplicated(data4),]
可以使用 unique（）函数一步完成非重复部分的提取
unique(data4)

数据转换（三）

数据框的翻转
使用 t()函数可以实现数据框的转置，行转成列。列转成行

```
sractm <- t(mtcars)
```

翻转单独一行
可以使用 rev()函数

```
rev(row.names(women))
women[rev(row.names(women)),]
```

先试用 rev()函数翻转利用 row.names()函数取得的行名，然后再将翻转后的行名作为索引，取出该行就行
如何修改数据框的值
比如将 women 数据框中的 height 列数据的值的单位由英寸转化成厘米

```
women$height*2.54
data.frame(women$height*2.54,women$weight)
```

先取出这一列，让数值全部乘以换算值，之后使用 data.frame()函数重新组合成一个数据框
但这种方法并不高效
可以使用 transform()函数

```
transform(women,height=height*2.54)
```

如果不想更改原来的数据，也可以直接用 transform 增加一列

```
transform(women,cm=height*2.54)
```

	height	weight	cm
1	58	115	147.32
2	59	117	149.86
3	60	120	152.40
4	61	123	154.94
5	62	126	157.48
6	63	129	160.02
7	64	132	162.56
8	65	135	165.10
9	66	139	167.64
10	67	142	170.18
11	68	146	172.72
12	69	150	175.26
13	70	154	177.80
14	71	159	180.34
15	72	164	182.88

如何对数据框进行排序

Sort()对向量进行排序，返回的是排序后的结果向量，rev(sort())则是按照相反的顺序进行排序

Order()也可以对向量进行排序，但返回的是向量所在的位置，也即是向量中 元素的索引

如果想按照某一行的元素的大小顺序进行排序，利用 order 的话可以这样写：

```
order(mtcars$cyl)
mtcars[order(mtcars$cyl),]
```

如果想取相反的顺序，直接在数据前添加一个负号即可

```
mtcars[order(-mtcars$cyl),]
```

还可以进行多重条件下的排序

```
mtcars[order(mtcars$mpg,mtcars$disp),]
```

数据转换（四）

如何对数据框进行数学计算？

有很多种方法，以 wordphones 数据为例：

```
rs <- rowSums(WorldPhones)
cm <- colMeans(WorldPhones)
```

rowsums()和 colmeans()函数可以计算行和还有列的平均数

使用 cbind()函数可以直接在数据后面添加计算后的一列数据

```
total <- cbind(WorldPhones,total=rs)
```

使用 rbind()函数可以直接在数据后面添加计算后的一行数据

```
total2 <- rbind(WorldPhones,meanvalue=cm)
```

在 R 中提供了 apply 系列函数

首先是 apply()函数的用法，对数据框的每一行进行求和

```
apply(WorldPhones,MARGIN =  1,FUN =sum)
```

其中，wordphones 代表要进行求和的数据集，margin 是数据的维数，margin=1 代表按行，margin=2 代表按列，FUN 参数代表使用的函数

对数据框的每一列进行求平均值

```
apply(WorldPhones,MARGIN =  2,FUN =mean)
```

对数据框的每一列进行求 log 值（对数值）

```
apply(WorldPhones,MARGIN =  2,FUN =log)
```

lapply()函数用法与 apply()函数类似，不过返回值是列表，同理，sapply()也是返回值不同，sapply()函数返回的是向量或者矩阵。
以 state.center 数据为例，该数据是列表数据：
统计一下列表中元素的个数：

```
lapply(state.center,FUN = length)
```

因为列表中不存在行和列，所以不需要有 margin 参数，返回的结果值是一个列表
tapply()函数用于处理因子数据，根据因子来分组，然后根据分组来处理
以 state.name 和 state.division 为例，state.division 作为因子数据进行分组：
计算每个类型区中州的数量

```
tapply(state.name,state.division,FUN = length)
```

```
        New England     Middle Atlantic      South Atlantic East
                  6                   3                       8
West North Central          Mountain            Pacific
                  7                   8                       5
```

接下来是数据的中心化与标准化处理（为了消除数据量纲对数据的影响），以 state.x77 数据为例：
数据的中心化，是指数据集中的各项数据减去数据集的均值
数据的标准化，是指在中心化之后再除以数据集的标准差，即数据集汇总的各项数据减去数据集的均值再除以数据集的标准差
在 R 中可以直接使用 scale()函数进行中心化和标准化的处理，当 scale()函数中的两个参数都为 True 时，就是做中心化（center=T）和标准化处理（scale=T）

```
scale(state.x77,center = T,scale = T)
```

经过中心化和标准化处理过后的数据，在绘制 heatmap 时会比较精确，对比性比较强。

数据转换（五）——reshape2 包的使用（详细可参考 R 语言实战）

假设有 x，y 两个数据框数据如下：

```
x <- data.frame(k1=c(NA,NA,3,4,5),k2=c(1,NA,NA,4,5),data=1:5)
y <- data.frame(k1=c(NA,2,NA,4,5),k2=c(NA,NA,3,4,5),data=1:5)
```

使用 merge()函数可以根据一个或多个共有的变量来进行合并，合并两组数据中共有变量列中相同的行
以 k1 作为共有变量合并：

```
merge(x,y,by = "k1")
```

以 k2 作为共有变量合并，并排除有 NA 的情况：

```
merge(x,y,by = "k2",incomparables = NA)
```

reshape2 包是一个重构和整合数据的万能工具，可以把数据转化成任何想要的形式
使用 airquality 数据为例：
首先，为了输入方便，使用 toloower()函数将列名改写成小写

```
names(airquality) <- tolower(names(airquality))
```

然后可以使用 melt()函数融合数据，融合之后，每一行都是唯一的标识符~变量的组合：

```
aql <- melt(airquality)
```

也即是将宽数据变成长数据的过程
Id 参数是用于告诉 melt()函数哪一行或者那一列用作观测，而剩余的数据作为观测值

```
aql <- melt(airquality,id.vars = c('month','day'))
```

数据的重构使用 dcast()或者 acast()函数，处理数据框使用 dcast()函数，acast()函数处理向量、矩阵或者数组：

Formula 参数用于描述数据的形状，“~”在 R 中代表两者相关联，用以下代码即可重构数据

```
aqw <- dcast(aql,month+day ~variable)
```

也可以只重构月数据，并对日数据求平均值，求一个月的平均值，还可以使用 na.rm()去除缺失值：

```
aqw <- dcast(aql,formula = month ~variable,fun.aggregate = mean,
na.rm=TRUE)
```

数据转换（六）——tidyr 包的使用

Tidyr 数据是一种很简洁的数据，数据的每一列代表一个变量，每一行代表一个观测，每一个观测的值在表中为一个单元格，也就是一个观测一个变量确定唯一的一个值。

接下来使用 mtcars 的数据来介绍一下 tidyr 包的函数用法

Mtcars 数据是很典型的一个 tidyr 数据，先取数据的 10 行和 3 列，赋值到一个新的变量上。

```
tdata <- mtcars[1:10,1:3]
```

取数据的行名生成新的一个数据框（方便处理）

```
tdata <- data.frame(names=rownames(tdata),tdata)
```

gather 函数：

gather（data,key,value…..），data 参数是要处理的数据，key 参数是宽数据变为长数据时，存放需要编码的变量的变量名称，需要自己定义，value 参数是需要数据转换的变量的数值，也需要自己定义。

```
gather(tdata,kkey = 'key',value = 'value',cyl,disp,mpg)
```

处理前数据：

```
    names  mpg cyl  disp
Mazda RX4               Mazda RX4 21.0   6 160.0
Mazda RX4 Wag         Mazda RX4 Wag 21.0   6 160.0
Datsun 710              Datsun 710 22.8   4 108.0
Hornet 4 Drive       Hornet 4 Drive 21.4   6 258.0
Hornet Sportabout Hornet Sportabout 18.7   8 360.0
Valiant                   Valiant 18.1   6 225.0
Duster 360              Duster 360 14.3   8 360.0
Merc 240D               Merc 240D 24.4   4 146.7
Merc 230                 Merc 230 22.8   4 140.8
Merc 280                 Merc 280 19.2   6 167.6
```

处理后数据：

```
              names  key value
1         Mazda RX4  cyl   6.0
2     Mazda RX4 Wag  cyl   6.0
3        Datsun 710  cyl   4.0
4    Hornet 4 Drive  cyl   6.0
5 Hornet Sportabout  cyl   8.0
6           Valiant  cyl   6.0
7        Duster 360  cyl   8.0
8         Merc 240D  cyl   4.0
9          Merc 230  cyl   4.0
10         Merc 280  cyl   6.0
```

```
11          Mazda RX4 disp 160.0
```

可以使用冒号，制定多列聚到同一列中，此处是将 cyl 与 mpg 先合并，然后在和 disp 组成一个数据框

```
gather(tdata,key = 'key',value = 'value',cyl:mpg,disp)
```

如果该在列名称前添加一个负号，代表不需要转换该列

```
gather(tdata,key = 'key',value = 'value',cyl,-dis
              names  mpg  disp key value
          Mazda RX4 21.0 160.0 cyl     6
      Mazda RX4 Wag 21.0 160.0 cyl     6
         Datsun 710 22.8 108.0 cyl     4
     Hornet 4 Drive 21.4 258.0 cyl     6
  Hornet Sportabout 18.7 360.0 cyl     8
            Valiant 18.1 225.0 cyl     6
          Duster 360 14.3 360.0 cyl     8
           Merc 240D 24.4 146.7 cyl     4
            Merc 230 22.8 140.8 cyl     4
0           Merc 280 19.2 167.6 cyl     6
```

也可以直接使用列的索引编号进行数据的重新构造

```
gather(tdata,key = 'key',value = 'value',2:4)
```

```
> gather(tdata,key = 'key',value = 'value',2:4)
              names  key value
1          Mazda RX4  mpg  21.0
2      Mazda RX4 Wag  mpg  21.0
3         Datsun 710  mpg  22.8
4     Hornet 4 Drive  mpg  21.4
5  Hornet Sportabout  mpg  18.7
6            Valiant  mpg  18.1
7          Duster 360  mpg  14.3
8           Merc 240D  mpg  24.4
9            Merc 230  mpg  22.8
10           Merc 280  mpg  19.2
11         Mazda RX4  cyl   6.0
12     Mazda RX4 Wag  cyl   6.0
```

Spread 函数则是将合并数据转化成 tidyr 形式的数据

例如，对之前合并的 tdata 函数进行分开：

```
gdata <- gather(tdata,key='key',value = 'value',2:4)
spread(gdata,key = 'key',value="value")
```

就可以还原数据：

```
               names cyl  disp  mpg
1          Datsun 710   4 108.0 22.8
2          Duster 360   8 360.0 14.3
3      Hornet 4 Drive   6 258.0 21.4
4   Hornet Sportabout   8 360.0 18.7
5           Mazda RX4   6 160.0 21.0
6       Mazda RX4 Wag   6 160.0 21.0
7            Merc 230   4 140.8 22.8
8            Merc 240D  4 146.7 24.4
9            Merc 280   6 167.6 19.2
10           Valiant    6 225.0 18.1
```

Separate 函数可以把一列拆成多个列：

例如我们创建一个数据框 df

```
df <- data.frame(x=c(NA,"a.b","a.d","b.c"))
```

将数据按照“.”分成两列：

```
separate(df,col = x,into = c('A','B'))
```

对 x 这列进行分割，分割成 A、B 两列，默认识别分隔符

分割之前：

```
> df
     x
1 <NA>
2  a.b
3  a.d
4  b.c
```

分割之后：

```
     A    B
1 <NA> <NA>
2    a    b
3    a    d
4    b    c
```

当出现多个分隔符而识别出现错误时，例如以下情况：

```
df <- data.frame(x=c(NA,"a.b-c","a-d","b-c"))
```

则可以利用 sep 函数指定分隔符，这样就能顺利分列：

```
separate(df,col = x,into = c('A','B'),sep="-")
```

可以使用 unite 函数将分开的列连接起来

```
unite(x,col ="AB",A,B,sep = "-")
```

x 是需要处理的数据，col 是连接后字段的名字，A,B 是需要连接的字段，sep 是用于连接的连接符

连接之后：

```
     AB
1 NA-NA
2 a.b-c
3   a-d
4   b-c
```

数据处理——dplyr 包的使用

使用 iris 数据对 dplyr 包的用法做介绍，首先是 filter 函数

可以使用 filter 函数用指定条件对数据进行筛选，比如以下代码可以筛选掉花萼长度在 7 以下的数据，保留在 7 以上的数据

```
dplyr::filter(iris,Sepal.Length>7)
```

distinct 函数可以用于去除重复行，相当于 unique 函数的功能

比如，可以使用 rbind 函数合并 iris 数据集的 1-10 行数据与 1-15 行数据，再使用第三条函数去除多余行：

```
dplyr::distinct(rbind(iris[1:10,],iris[1:15,]))
```

以上代码的运行结果只会保留，10-15 行的数据，因为 1-10 行数据是重复的

Slice 函数可以用于切片，可以取出数据的任意行：

```
dplyr::slice(iris,10:15)
```

以上代码可以取出 iris 数据的 10-15 行

Sample_n 函数用于随机取样，例如：

```
dplyr::sample_n(iris,10)
```

代表在 iris 这个数据中随机抽取 10 行

Sample_frac 表示按比例随机选取，比如抽取源数据的 10%的数据，是指百分比：

```
dplyr::sample_frac(iris,0.1)
```

arrange 函数用于排序，比如将 iris 数据按照花萼长度进行排序（正向），可写成：

```
dplyr::arrange(iris,Sepal.Length)
```

如果加上前面加上负号则是进行反向排序（从大到小）

```
dplyr::arrange(iris,-Sepal.Length)
```

关于 dplyr 的统计函数：

可以使用 summarise()函数进行统计，比如统计花萼长度的平均值：

```
summarise(iris,avg=mean(Sepal.Length))
```

统计花萼长度的和：

```
summarise(iris,total=sum(Sepal.Length))
```

R 中一个非常有用的负号：%>% 这个符号是链式操作符，它的功能是用于实现将一个函数的输出传递给下一个函数，作为下一个函数的输入，在键盘上可以用 Ctrl+shift+M 的快捷键打出来，有点类似于“且”的概念。

比如，先使用 head 函数取出数据集的前 20 行，再使用链式操作符，可以将这 20 行数据作为下一个命令的输入，如下一个命令是 tail（10），也就是取出倒数 10 行，那么就会取出这 20 行数据的第 11-20 行：

```
head(mtcars,20)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1

```
head(mtcars,20) %>% tail(10)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1

Group_by 函数可以对数据进行分组，例如根据 species 这列对数据进行分组：

如果用链式操作符，代码可以改为：

```
iris %>% group_by(Species)
```

还可以结合 summarize 统计函数进行进一步的计算，例如可以计算每一种类型鸢尾花品种的花萼宽度的平均值：

```
iris %>% group_by(Species) %>% summarise(avg=mean(Sepal.Width))
```

在此基础上，还可以使用链式操作符，对宽度的平均值进行排序：

```
iris %>% group_by(Species) %>% summarise(avg=mean(Sepal.Width))
 %>% arrange(avg)
```

mutate 函数可以添加新的变量

使用 mutate 函数在 iris 数据中增加一行，该行数据是花萼与花瓣长度的总和：

```
dplyr::mutate(iris,new=Sepal.Length+Petal.Length)
```

下面介绍一下的 dplyr 包对于双表格的操作：

比如将两个表格整合到一起，先创建两个数据框，分别是 a,b

```
a=data.frame(x1=c('A','B','C'),x2=c(1,2,3))
b=data.frame(x1=c('A','B','D'),x2=c(T,F,T))
```

首先是左连接，left_join()，以 x1 列为基础进行连接

```
dplyr::left_join(a,b,by="x1")
```

```
> dplyr::left_join(a,b,by="x1")
  x1 x2.x  x2.y
1  A    1  TRUE
2  B    2 FALSE
3  C    3    NA
```

然后是右连接，right_join()，是以 b 的 x1 列为基础进行连接，

```
> dplyr::right_join(a,b,by="x1")
  x1 x2.x  x2.y
1  A    1  TRUE
2  B    2 FALSE
3  D   NA  TRUE
```

然后是内连接和全连接，内连接是取 x1 的交集，全连接是取 x1 的并集

内连接：

```
dplyr::inner_join(a,b,by="x1")
```

```
  x1 x2.x  x2.y
1  A    1  TRUE
2  B    2 FALSE
```

全连接：

```
dplyr::full_join(a,b,by="x1")
```

```
> dplyr::full_join(a,b,by="x1")
  x1 x2.x  x2.y
1  A    1  TRUE
2  B    2 FALSE
3  C    3    NA
4  D   NA  TRUE
```

半连接相当于根据右侧表的内容对左侧表进行过滤，也就是将两个表的数据的交集取出来：

半连接：

```
dplyr::semi_join(a,b,by="x1")
```

```
  x1 x2
1  A  1
2  B  2
```

反连接是输出两个表的补集部分，也就是取出数据框 a 中 b 不含有的行：

```
dplyr::anti_join(a,b,by="x1")
```

```
  x1 x2
1  C  3
```

以上两个函数是互补的函数，一个是取共有的，一个是取没有的。

下面来看几个数据框的合并操作

首先定义数据 first 变量，变量取 mtcars 数据前 20 行，second 变量取第 10-30 行，由于 slice 函数取出的数据不含行名，需要在取出数据前用 mutate 函数添加一列行名

```
mtcars <- mutate(mtcars,names=row.names(mtcars))
```

```
first <- slice(mtcars,1:20)
second <- slice(mtcars,10:30)
```

然后验证函数

Intersect()取交集：

```
> intersect(first,second)
```

Union_all()取并集：

```
dplyr::union_all(first,second)
```

Union()取非冗余的并集，去除掉重复部分再进行合并：

```
dplyr::union(first,second)
```

setdiff()取数据的补集，也就是取出 frist 数据中 second 数据没有的部分：

```
setdiff(first,second)
```

函数介绍——R 语言中的函数

使用 lm()函数进行回归分析，例如研究 state.x77 这个数据，研究犯罪率与其他指标的关系：

首先将数据转化为数据框，因为 lm()函数只能对数据框进行操作

```
state <- as.data.frame(state.x77[,c('Murder','Population','Illiteracy','Income','Frost')])
```

再使用 lm()函数进行回归分析，研究人口、文盲率、收入以及天气对犯罪率的影响：

```
fit <- lm(Murder ~ Population+Illiteracy+Income+Frost,data=state)
```

使用 summary()得出统计结果

```
summary(fit)
```

```
Call:
lm(formula = Murder ~ Population + Illiteracy + Income + Frost,
    data = state)

Residuals:
    Min      1Q  Median      3Q     Max
-4.7960 -1.6495 -0.0811  1.4815  7.6210

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.235e+00  3.866e+00   0.319   0.7510
Population  2.237e-04  9.052e-05   2.471   0.0173 *
Illiteracy  4.143e+00  8.744e-01   4.738 2.19e-05 ***
Income      6.442e-05  6.837e-04   0.094   0.9253
Frost       5.813e-04  1.005e-02   0.058   0.9541
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.535 on 45 degrees of freedom
Multiple R-squared:  0.567,     Adjusted R-squared:  0.5285
F-statistic: 14.73 on 4 and 45 DF,  p-value: 9.133e-08
```

后面的*代表数据的显著性，数据中，文盲率的回归系数是 4.14，表示其他数据不变时，文盲率上升 1%，则犯罪率上升 4.14%，且回归系数在 p<0.001 的情况下，显著不为 0，也就是显著。而如果数据不显著，则说明两者没有直接关系。

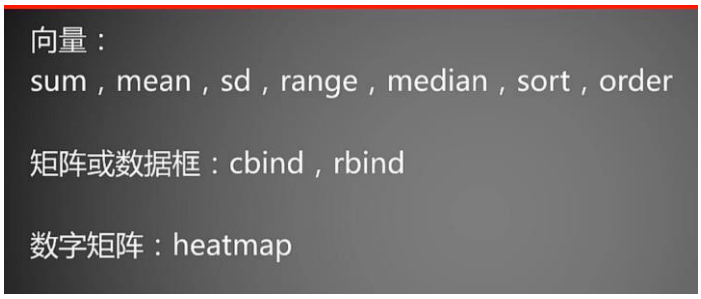
R 函数的返回值：

ls()返回当前环境中的对象，也就是有多少变量

sys.date()返回当前系统时间

rm()删除指定的变量，但是这个函数是直接删除，不会有返回

使用函数要注意输入数据的格式



可以使用 help()函数查看每个函数的帮助文档

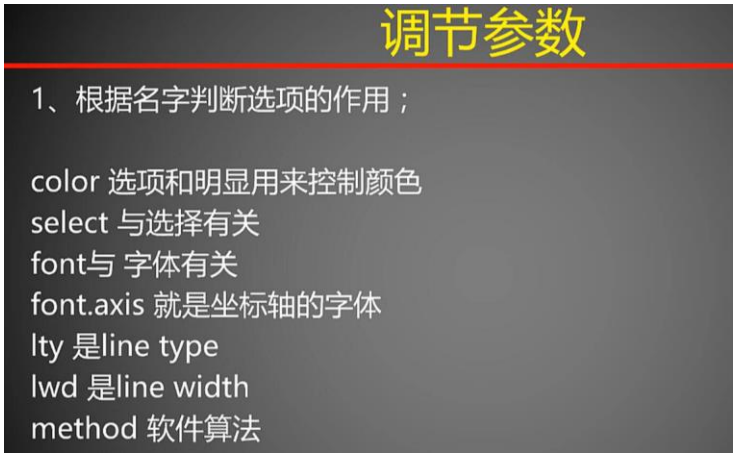
函数介绍——函数的选项参数

一般函数的选项参数可以分成三个部分：输入控制部分；输出控制部分；调节部分

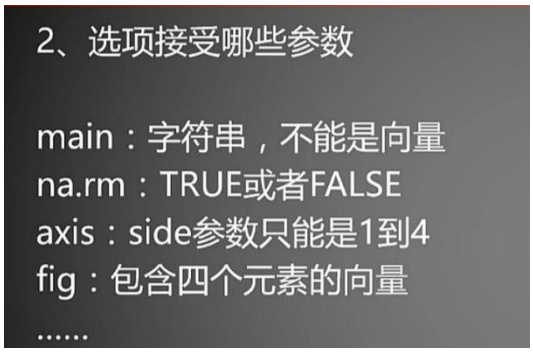
输入控制部分负责告诉用户函数能接受哪种类型的数据，这个选项参数往往出现在函数的第一位，比如说，有些函数的第一位选项参数是“file”说明使用这个函数你需要输入一个文件；如果是“data”则是需要输入一个数据框；“x”代表单独的一个对象，一般都是向量，也可以是矩阵或者列表；“x 和 y”代表函数需要两个输入变量；“x、y、z”函数需要三个输入变量；“formula”输入的是公式；（具体查看每个函数的帮助文档）

输出控制部分

调节参数：



选项接受哪些参数：



函数介绍——数学统计函数

（1）概率函数

概率论是统计学的基础，R 有许多用于处理概率，概率分布以及随机变量的函数，R 对每一个概率分布都有一个简称，这个名称用于识别与分布相联系的函数，这部分涉及到很多统计学基础的理论知识，比如随机试验、样本空间、对立与互斥、随机事件与必然事件、概率密度、概率分布等。

R 中的概率函数（正态分布）：d 前缀—概率密度函数；P 前缀—概率分布函数；q 前缀—分位数函数（分布函数的反函数）；r 前缀—产生相同分布的随机数

Norm 表示正态分布

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

例如生成一组符合正态分布的随机数，这组数据均值为 15，标准差为 2，总共 100 个数据：

```
> rnorm(n=100,mean = 15,sd=2)
```

R 中的概率函数（离散分布）：同样的道理，在这些分布缩写前面加上 d、p、q、r 就变成函数

表1 概率分布

分布名称	缩写	分布名称	缩写
Beta分布	beta	Logistic分布	logis
二项分布	binom	多项分布	multinom
柯西分布	cauchy	负二项分布	nbinom
（非中心）卡方分布	chisq	正态分布	norm
指数分布	exp	泊松分布	pois
F分布	f	Wilcoxon符号秩分布	signrank
Gamma分布	gamma	t分布	t
几何分布	geom	均匀分布	unif
超几何分布	hyper	Weibull分布	weibull
对数正态分布	lnorm	Wilcoxon秩和分布	wilcox

这些分布函数可以帮助我们在 R 中绘制各种分布函数图。

R 中如何生成随机数：

最简单的是 runif()函数，可以生成 0-1 之间的随机数

生成 50 个 0-1 之间的随机数

```
> runif(50)
```

如果想生成 0-1 之外的随机数，可以通过修改选项参数来更改

```
runif(50,min=1,max = 100)
```

这样就能生成 1-100 以内的随机数了

Set.seed()函数可以绑定随机数，当输入 Set.seed()函数时，回到最初的随机数：

```
> set.seed(123)#这个数字是随便起的，都可以
> runif(50,min=1,max=100)
 [1] 29.470174 79.042208 41.488715 88.418723
[12] 45.880081 68.079493 57.690707 11.189544
[23] 64.410175 99.432708 65.914874 71.144516
[34] 79.751274  3.436755 48.301801 76.087494
[45] 16.092030 14.741800 24.070376 47.130283
> runif(50,min=1,max=100)
 [1]  5.537286 44.777807 80.093560 13.068027
[12] 10.389225 39.012994 28.163981 81.649364
[23] 71.308058  1.061853 48.056341 22.791770
[34] 79.031388 11.183600 44.054381 98.510741
[45] 32.716951 19.581421 78.447136 10.265904
> set.seed(123)
> runif(50,min=1,max=100)
 [1] 29.470174 79.042208 41.488715 88.418723
[12] 45.880081 68.079493 57.690707 11.189544
[23] 64.410175 99.432708 65.914874 71.144516
[34] 79.751274  3.436755 48.301801 76.087494
[45] 16.092030 14.741800 24.070376 47.130283
```

每个 Set.seed()号码对应的随机数是相同的，这个功能主要是可以在研究发表时重现随机分组，以保证在不同设备上也能独立获得相同的结果。

函数介绍——描述性统计函数（R 语言实战 p.131）

首先是 summary()函数，运行一次该函数，就可以对数据进行详细的统计。

先自定义一个变量，把 mtcars 的数据赋值给这个变量：

```
myvars <- mtcars[c("mpg",'hp','wt','am')]
```

使用 summary()函数计算变量的数据，结果包括最小值，最大值，四分位数和数值型变量的均值。

```
> summary(myvars)
      mpg              hp              wt              am        
 Min.   :10.40   Min.   : 52.0   Min.   :1.513   Min.   :0.0000  
 1st Qu.:15.43   1st Qu.: 96.5   1st Qu.:2.581   1st Qu.:0.0000  
 Median :19.20   Median :123.0   Median :3.325   Median :0.0000  
 Mean   :20.09   Mean   :146.7   Mean   :3.217   Mean   :0.4062  
 3rd Qu.:22.80   3rd Qu.:180.0   3rd Qu.:3.610   3rd Qu.:1.0000  
 Max.   :33.90   Max.   :335.0   Max.   :5.424   Max.   :1.0000  
```

在回归分析中也会大量用到 summary()函数

Fivenum()函数和 summary()函数类似，但可以返回 5 个基本的统计量，包括最小值，四分位数、中位数、上四位数、最大值。

```
fivenum(myvars$hp)
```

这里使用了$符取出单个数据列进行统计。

Hmisc 包中的 describe()函数也可以计算统计量，可以返回变量和观测的数量、缺失值和唯一值的数目、以及平均值、分位数、已经五个最大的值和五个最小的值

```
describe(myvars)
```

pastecs 包中有一个 stat.desc()函数可以计算种类繁多的描述性统计量，stat.desc()函数的选项参数中，x 是一个数据框或者是时间序列

```
install.packages("pastecs")
library(pastecs)
stat.desc(myvars)
```

```
> stat.desc(myvars)
                     mpg
nbr.val       32.0000000
nbr.null       0.0000000
nbr.na         0.0000000
min           10.4000000
max           33.9000000
range         23.5000000
sum          642.9000000 4
median        19.2000000
mean          20.0906250
```

如果设置 base 选项等于 true，那么就会计算一些基本值，包括全部值的数量、空值以及缺失值的数量、最小值、最大值、值域、还有总和

```
stat.desc(myvars,basic = T)
```

如果设置 desc 选项等于 true，那么就会计算一些描述值，包括中位数、平均数、平均数的标准误、平均置信度为 95%的置信空间、方差、标准差、以及变异系数等

```
> stat.desc(myvars,desc = T)
```

默认情况下，两个选项参数都是 T

如果设置 norm 为 T，那么就会计算一些统计值，包括正态分布统计量、偏度和峰度等

```
stat.desc(myvars,norm = T)
```

psych 包中也有一个 describe()函数，可以计算非缺失值的数量、平均数、标准差、中位数、截尾的均值、最大值、最小值、偏度和峰度等等内容

截尾的均值是去掉两头的数据取均值，就像打分时，去掉一个最低分、去掉一个最高分，然后中间数据求均值。

describe()函数可以通过设置 trim 参数，设置去除比例，如 trim=0.1，则是去除数据中最高和最低的 10%的部分

当两个包的函数名一样时，后面载入的包的函数会覆盖前面载入的包的函数，如果要使用前面一个包的函数，只需要在包后加冒号再使用即可，如：

```
Hmisc::describe()
```

Aggregate()函数可对数据进行分组描述，能够对数据按照指定的分组信息进行统计，将分组信息通过一个列表指定出来即可

例如我们使用 mass 这个包中的 cars93 数据集【93 年许多不同汽车的指标】

```
> library(MASS)
```

以制造商这一列数据为例，

```
aggregate(Cars93[c("Min.Price","Max.Price","MPG.city")],by=lis
t(Manufacturer=Cars93$Manufacturer),mean)
```

这样就是根据汽车制造商来对数据进行分组统计，计算每个汽车制造商产品平均的价格

还可以根据产地：

```
aggregate(Cars93[c("Min.Price","Max.Price","MPG.city")],by=lis
t(origin=Cars93$Origin),mean)
```

```
    origin Min.Price Max.Price MPG.city
1      USA  16.53542  20.62708 20.95833
2 non-USA  17.75556  23.25556 23.86667
```

还可以将 mean 函数替换成其他函数，比如 sd，计算数据的标准差

```
aggregate(Cars93[c("Min.Price","Max.Price","MPG.city")],by=lis
t(origin=Cars93$Origin),sd)
```

也可以一次性使用多个分组条件，只需要在列表中添加即可，例如同时使用产地和制造商来分组：

```
aggregate(Cars93[c("Min.Price","Max.Price","MPG.city")],by=lis
t(origin=Cars93$Origin,Manufacturer=Cars93$Manufacturer),mean)
```

aggregate 函数的缺点是一次只用使用一个统计函数，比如只能返回平均值、方差等，可以使用一些扩展包来进行分组计算并实现返回多种描述性统计量

首先是 doBy 包中的 summaryBy()函数，summary_by(data, formula,id = NULL,FUN = mean..)在波浪线左侧是需要分析的数值型变量，直接写数据框中的列的名字就可以，不需要添加引号；不同变量之间用+号表示；右侧的变量是类别型的分组变量；data 参数指定数据集，fun 参数指定统计函数，也可以是自定义函数：

```
summaryBy(mpg+hp+wt~am,data=myvars,FUN = mean)
```

```
  am mpg.mean  hp.mean  wt.mean
1  0 17.14737 160.2632 3.768895
2  1 24.39231 126.8462 2.411000
```

Fun 参数后面可以接多个统计函数，如

```
summaryBy(mpg+hp+wt~am,data=myvars,FUN = c(mean,sd,sum))
```

```
summaryBy(mpg+hp+wt~am,data=myvars,FUN = c(mean,sd,sum))
```

```
 am mpg.mean   hp.mean   wt.mean    mpg.sd     hp.sd      wt.sd mpg.sum
  0 17.14737 160.2632 3.768895 3.833966 53.90820 0.7774001   325.8
  1 24.39231 126.8462 2.411000 6.166504 84.06232 0.6169816   317.1
```

Psych 包中的 describe.by()函数和 describe()函数能够计算相同的统计量，但是 describe.by()函数可以通过分组来计算，只需要添加一个分组的列表，直接给定一个 list 即可：

```
describe.by(myvars,list(am=mtcars$am))
```

```
 Descriptive statistics by group
am: 0
    vars  n   mean    sd median trimmed   mad   min    max  range
mpg    1 19  17.15  3.83  17.30   17.12  3.11 10.40  24.40  14.00
hp     2 19 160.26 53.91 175.00  161.06 77.10 62.00 245.00 183.00
wt     3 19   3.77  0.78   3.52    3.75  0.45  2.46   5.42   2.96
am     4 19   0.00  0.00   0.00    0.00  0.00  0.00   0.00   0.00
------------------------------------------------------------------
am: 1
    vars  n   mean    sd median trimmed   mad   min    max  range
mpg    1 13  24.39  6.17  22.80   24.38  6.67 15.00  33.90  18.90
hp     2 13 126.85 84.06 109.00  114.73 63.75 52.00 335.00 283.00
wt     3 13   2.41  0.62   2.32    2.39  0.68  1.51   3.57   2.06
am     4 13   1.00  0.00   1.00    1.00  0.00  1.00   1.00   0.00
Warning message:
```

describe.by()适合详细查看每一个分组的统计值，但缺点是给出的统计值是固定不变的，没办法使用自定义的函数

函数介绍——频数统计函数

频数在数据分析中是非常重要的一个概念，因为经常需要进行分组统计，比较不同组之间的差异，这些都要涉及到频数的统计。因子是专门用来进行分组的，有因子才能分组，分组之后才能进行频数统计。

首先介绍一下 R 如何对数据进行分组：

如果一个数据本身就是因子，那么直接就可以进行分组，例如 mtcars 数据集，里面的“cyl”这一列数据直接就可以作为因子，依据气缸数的不同来进行分组。

用 as.factor()函数将这列转换为因子数据：

```
cylfactors <- as.factor(mtcars$cyl)
```

然后可以使用 split()函数对数据进行分组：

```
myvars <- dplyr::mutate(myvars,cylfactors1=cylfactors)
split(myvars,myvars$cylfactors1)
```

还可以使用 cut()函数（如果没有明显的分类），cut 可以对连续的数据进行切割，使用 cut()函数对“mpg”这列进行分割，根据 10-50 切成 10 份，就是以步长为 10，每十下一切：

```
cut(myvars$mpg,c(seq(10,50,10)))
```

```
 [1] (20,30] (20,30] (20,30] (20,30] (10
[15] (10,20] (10,20] (10,20] (30,40] (30
[29] (10,20] (10,20] (10,20] (20,30]
Levels: (10,20] (20,30] (30,40] (40,50]
```

此处 seq()函数的选项参数是：seq(from=，to=，by=组距)

在分组之后，就可以用 table()函数进行频数的统计，table()函数可以计算频数表。

table()函数的使用比较简单：

```
table(myvars$cylfactors1)
```

```
 4  6  8
11  7 14
```

cut()函数的结果也可以使用 table()来统计：

```
table(cut(myvars$mpg,c(seq(10,50,10))))
```

```
(10,20] (20,30] (30,40] (40,50]
     18      10       4       0
```

以上就是频数统计的做法

用频数除以总数就是频率值，R 中可以直接使用 prop.table()函数计算频率值：

```
prop.table(table(myvars$cylfactors1))
```

```
      4       6       8
0.34375 0.21875 0.43750
```

频率值*100 就是百分比的结果。

那么二维的数据框如何进行频率统计呢？

可以使用 table()或者是 as.table()函数，选项参数输入两个因子就可以计算二维数据的频率了。

这里我们以 vcd 包中的风湿病数据集（arthritis）进行示范。

Arthritis 数据集中的“treatment”、“sex”以及“improved”列都可以作为因子数据，因为它们是类型量，我们取两个作统计：

```
table(Arthritis$Treatment,Arthritis$Improved)
```

返回的结果是一个二维的列联表，横向是安慰剂组与治疗组的区分，每一列分别是“没有效果”、“有一些效果”、“很有效果”的区分：

```
          None Some Marked
  Placebo   29    7      7
  Treated   13    7     21
```

如果变量太多，我们也可以先使用 with()或者是 attach()函数先加载数据：

```
with(data = Arthritis,table(Treatment,Improved))
```

这样就不用反复的书写变量名称

处理二维列联表还可以使用 xtabs()函数，这个函数的好处是它的选项参数使用的是 formula 参数，这样就可以根据需要写成多种公式。

同样是“treatment”以及“improved”列，使用 xtabs()函数计算如下：

```
xtabs(~ Treatment+Improved,data = Arthritis)
```

formula 参数是可以省略的，结果和 with()函数的一致

```
          Improved
Treatment None Some Marked
  Placebo   29    7      7
  Treated   13    7     21
```

对于二维列联表我们还可以使用 margin.table()和 prop.table()函数分别计算边际频数与比例（边际频率），边际频数的意思就是单独按照行或者列的数据进行处理。

我们将 xtabs()的结果保存到变量 x 中，再使用 margin.table()函数统计一下 x:

```
x <- xtabs(~ Treatment+Improved,data = Arthritis)
margin.table(x)
```

返回值只有一个，代表返回的是所有的结果，这里需要给定一个边际值，1 或者 2,1 代表行，2 代表列：

```
margin.table(x,1)
```

```
Treatment
Placebo Treated
     43      41
```

使用 prop.table()函数计算比例（边际频率）：

```
prop.table(x,1)
```

```
          Improved
Treatment      None      Some    Marked
  Placebo 0.6744186 0.1627907 0.1627907
  Treated 0.3170732 0.1707317 0.5121951
```

从边际和比例的计算结果可以看出，有治疗组有 51%的比例是效果显著的，比边际频数 41 要大，说明药物是有用的。

再将 margin.table()参数换成 2，则是按列进行统计

```
margin.table(x,2)
```

addmargin()函数可以直接将边际的和添加到频数表中

```
addmargins(x)
```

```
          Improved
Treatment None Some Marked Sum
  Placebo   29    7      7  43
  Treated   13    7     21  41
  Sum       42   14     28  84
```

这个函数也可单独计算行、列

```
addmargins(x,1)
```

```
          Improved
Treatment None Some Marked
  Placebo   29    7      7
  Treated   13    7     21
  Sum       42   14     28
```

```
addmargins(x,2)
```

```
          Improved
Treatment None Some Marked Sum
  Placebo   29    7      7  43
  Treated   13    7     21  41
```

我们也可以计算三维的列联表，加多一个参数就可以：

```
xtabs(~ Treatment+Improved+Sex,data = Arthritis)
```

```
, , Sex = Female

          Improved
Treatment None Some Marked
  Placebo   19    7      6
  Treated    6    5     16

, , Sex = Male

          Improved
Treatment None Some Marked
  Placebo   10    0      1
  Treated    7    2      5
```

结果看起来有点乱，这时候我们可以使用 ftable()函数，它能将结果转换为一个平铺式的列联表

```
y <-  xtabs(~ Treatment+Improved+Sex,data = Arthritis)
ftable(y)
```

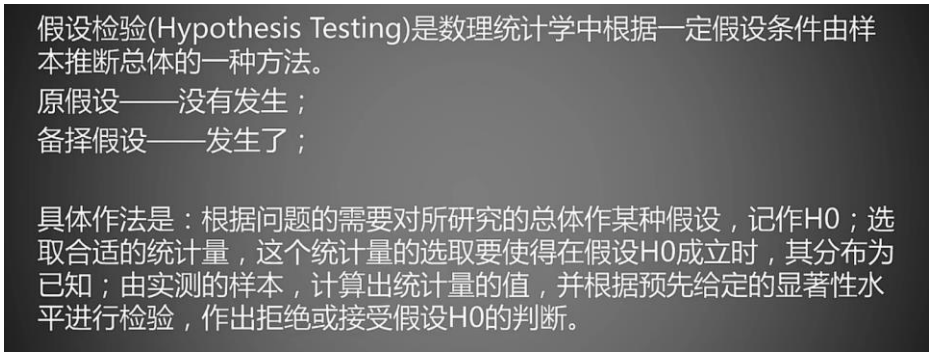
```
                    Sex Female Male
Treatment Improved
Placebo   None             19   10
          Some              7    0
          Marked            6    1
Treated   None              6    7
          Some              5    2
          Marked           16    5
```

函数介绍——独立性检验函数

独立性检验是根据频数信息判断两类因子彼此相关或相互独立的假设性检验，所谓独立性就是指变量之间是独立的，没有关系。

根据分组计算的频数表就可以进行独立性检验。

主要介绍三种检验方法：卡方检验、Fisher 检验、cochan-mantel-haenszel 检验



p-value 就是 probability 的值，它是一个通过计算得到的概率值，也就是在原假设为真实，得到最大的或者抄书做得到的检验统计量值的概率。一般将 p 值定位到 0.05，当 p<0.05 时，拒绝原假设（也就是假设成立），p>0.05 是不拒绝原假设。

还是使用 arthritis 数据集，探究药物治疗有没有成效，检验“treatment”和“improved”是不是相互独立的，如果相互独立，说明二者没有关系，药物治疗没有作用，反之则是有效果。

先使用 table()函数计算两者的频数：

```
mytable <- table(Arthritis$Treatment,Arthritis$Improved)
```

接下来就可以直接使用 chisq.test()函数进行卡方独立检验，直接将结果输入这个函数就行：

```
chisq.test(mytable)
```

结果如下，p 值约等于 0.0014 小于 0.05，说明两者不是独立的，两者有关系，治疗是有效果的：

```
        Pearson's Chi-squared test

data:  mytable
X-squared = 13.055, df = 2, p-value = 0.001463
```

这种待检测的变量之间没有顺序的关系，调整两者的顺序，结果是一样的：

```
mytable <- table(Arthritis$Improved,Arthritis$Treatment)
chisq.test(mytable)
```

```
        Pearson's Chi-squared test

data:  mytable
X-squared = 13.055, df = 2, p-value = 0.001463
```

Fisher 精确检验：

Fisher 精确检验使用的函数是 Fisher.test()，同样是进行独立性检验，但是与卡方检验不同的是，Fisher 精确检验的原假设是：边界固定的列联表中行和列相互独立的。

还是同样的例子，采用 Fisher 精确检验：

```
mytable <- xtabs(~Treatment+Improved,data = Arthritis)
> fisher.test(mytable)
```

结果为 p 值约等于 0.0013，Fisher 检验适合小样本的检验，精度低于卡方检验：

```
        Fisher's Exact Test for Count Data

data:  mytable
p-value = 0.001393
alternative hypothesis: two.sided
```

cochan-mantel-haenszel 检验：

cochan-mantel-haenszel 检验使用的函数是 mantelhaen.test()，该鲜艳的原假设是两个名义变量在第三个变量的每一层中都是条件独立的。这个检验需要三个变量，此处我们来检测一下“treatment”、“sex”以及“improved”之间的关系：

首先计算三个变量的列联表，使用 xtabs()计算：

```
mytable <- xtabs(~Treatment+Improved+Sex,data = Arthritis)
mantelhaen.test(mytable)
```

结果为 p 值约等于 0.0006，小于 0.05，也就说明，药物治疗和改善情况在性别的每一个水平上不独立（因为这里使用了性别做第三层分类量）：

```
        Cochran-Mantel-Haenszel test

data:  mytable
Cochran-Mantel-Haenszel M^2 = 14.632, df = 2, p-value = 0.000664
```

如果调整变量顺序，反映的结果是有差别的：

```
mytable <- xtabs(~Treatment+Sex+Improved,data = Arthritis)
mantelhaen.test(mytable)
```

```
, , Improved = None

         Sex
Treatment Female Male
  Placebo     19   10
  Treated      6    7

, , Improved = Some

         Sex
Treatment Female Male
  Placebo      7    0
  Treated      5    2
```

```
        Mantel-Haenszel chi-squared test with continuity correction

data:  mytable
Mantel-Haenszel X-squared = 2.0863, df = 1, p-value = 0.1486
```

结果为 p 值约等于 0.14，大于 0.05，也就说明，药物治疗和性别在改善情况的每一个水平上独立（因为这里使用了改善情况做第三层分类量）

函数介绍——相关性分析函数

相关性分析是指对两个或者多个具备相关性的变量元素进行分析，从而衡量两个变量因素的相关密切程度。相关性元素之间需要存在一定的联系或者概率才可以进行相关性分析，简单来说就是变量之间是否有关系（也就是说需要先进行独立检测之后才能进行相关分析）。

相关系数的大小表示相关性的大小，相关系数包括：pearson 相关系数、spearman 相关系数、kendall 相关系数、偏相关系数、多分格相关系数和多系列相关系数

与独立性检验不同，相关性分析中每种方法都没有独立的函数，这里面计算相关性系数都使用同一个函数：cor()函数。

cor()函数可以计算三种相关性系数，包括 pearson 相关系数、spearman 相关系数和 kendall 相关系数，具体使用哪种方法可以使用选项参数中的参数 method 来指定（默认是用 pearson 相关系数），；函数中还有一个 use 选项，用于指定如何对待缺失值，是不处理还是删除等。

此处我们使用 state.x77 数据作为实例数据（这是一个矩阵数据）：

```
cor(state.x77)
```

部分结果截图：

	Population	Income	Illiteracy	Life Exp
Population	1.00000000	0.2082276	0.10762237	-0.06805195
Income	0.20822756	1.0000000	-0.43707519	0.34025534
Illiteracy	0.10762237	-0.4370752	1.00000000	-0.58847793
Life Exp	-0.06805195	0.3402553	-0.58847793	1.00000000
Murder	0.34364275	-0.2300776	0.70297520	-0.78084575
HS Grad	-0.09848975	0.6199323	-0.65718861	0.58221620
Frost	-0.33215245	0.2262822	-0.67194697	0.26206801
Area	0.02254384	0.3633154	0.07726113	-0.10733194

一般相关数据都是在[0,1]之间，数值越大越相关，正负号表示是正相关还是负相关

除了 cor()函数之外，还有一个 cov()函数可以用来计算协方差，携房产可以用来衡量两个变量的整体误差

例如我们定义两个变量：x,y

```
x <- state.x77[,c(1,2,3,6)]
y <- state.x77[,c(4,5)]
```

再使用 cor()函数计算两者之间的相关系数：

```
cor(x,y)
```

结果看起来会比较整洁、清爽：

	Life Exp	Murder
Population	-0.06805195	0.3436428
Income	0.34025534	-0.2300776
Illiteracy	-0.58847793	0.7029752
HS Grad	0.58221620	-0.4879710

cor()函数只能计算三种相关系数，其他相关系数的计算可以通过 R 的拓展包来实现：

可以使用“ggm”这个包中的 pcor()函数计算偏相关系数。

偏相关系数是指在控制一个或者多个变量时，剩余其他变量之间的相互关系：

pcor()函数需要输入两个重要的参数，第一个参数是一个数值向量（前两个数值表示要计算相关系数的下标，其余的数值为条件变量的下标），第二个参数是 cov()函数计算出来的协方差结果

比如此处我们想控制收入水平、文盲率和高中毕业率的影响，看人口（第一列）和谋杀率（第

五列）之间的关系（先获取列名，不然容易混）：

```
colnames(state.x77)
```

```
1] "Population" "Income"     "Illiteracy" "Life Exp"   "Murder"     "HS Grad"    "Frost"      "Area"
```

```
pcor(c(1,5,2,3,6),cov(state.x77))
```

偏相关系数的结果：

```
[1] 0.3462724
```

函数介绍——相关性检验函数

在进行相关性分析之后，对相关性进行检验，cor.test()函数可用于相关性的检验，该函数有四个比较重要的选项参数，其中 x 和 y 是需要检测的相关性变量，alternative 是用来指定进行双侧检验还是单侧检验，two.sided 代表分别检测正负相关性，greater 代表正相关，less 代表负相关。Method 选项用于指定用哪种相关系数，可选的有 pearson 相关系数、spearman 相关系数、kendall 相关系数。

我们使用 state.x77 数据检测一下谋杀率与文盲率之间的关系

```
cor.test(state.x77[,3],state.x77[,5])
```

```
        Pearson's product-moment correlation

data:  state.x77[, 3] and state.x77[, 5]
t = 6.8479, df = 48, p-value = 1.258e-08
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.5279280 0.8207295
sample estimates:
      cor
0.7029752
```

置信区间：confidential interval，是指样本统计量所构造的总体参数的估计区间，在统计学中，一个概率样本的置信区间是对这个样本的某个总体参数的区间估计，置信区间展现的是这个参数的真实值有一定的概率落在测量结果周围的程度，置信区间给出的是被测量参数的测量值的可信程度（也就是说，光给出概率还不行，还要给出概率发生的范围）

Cor.test()只能一次性检测一组变量的关系，psych 包中的 corr.test()函数可以一次性进行多个变量的检验，这个函数还可以进行递归操作

```
corr.test(state.x77)
```

函数不仅计算了相关系数，还隔出了检测值：

```
Call:corr.test(x = state.x77)
Correlation matrix
           Population Income Illiteracy Life Exp Murder HS Grad Frost  Area
Population       1.00   0.21       0.11    -0.07   0.34   -0.10 -0.33  0.02
Income           0.21   1.00      -0.44     0.34  -0.23    0.62  0.23  0.36
Illiteracy       0.11  -0.44       1.00    -0.59   0.70   -0.66 -0.67  0.08
Life Exp        -0.07   0.34      -0.59     1.00  -0.78    0.58  0.26 -0.11
Murder           0.34  -0.23       0.70    -0.78   1.00   -0.49 -0.54  0.23
HS Grad         -0.10   0.62      -0.66     0.58  -0.49    1.00  0.37  0.33
Frost           -0.33   0.23      -0.67     0.26  -0.54    0.37  1.00  0.06
Area             0.02   0.36       0.08    -0.11   0.23    0.33  0.06  1.00
Sample Size
[1] 50
Probability values (Entries above the diagonal are adjusted for multiple tests.)
           Population Income Illiteracy Life Exp Murder HS Grad Frost Area
Population       0.00   1.00       1.00     1.00   0.23    1.00  0.25 1.00
Income           0.15   0.00       0.03     0.23   1.00    0.00  1.00 0.16
Illiteracy       0.46   0.00       0.00     0.00   0.00    0.00  0.00 1.00
Life Exp         0.64   0.02       0.00     0.00   0.00    0.00  0.79 1.00
Murder           0.01   0.11       0.00     0.00   0.00    0.01  0.00 1.00
HS Grad          0.50   0.00       0.00     0.00   0.00    0.00  0.16 0.25
Frost            0.02   0.11       0.00     0.07   0.00    0.01  0.00 1.00
Area             0.88   0.01       0.59     0.46   0.11    0.02  0.68 0.00
```

如果想进行偏相关系数的检验，可以使用 ggm 包中的 pcor.test()函数，先计算偏相关系数

```
pcor(c(1,5,2,3,6),cov(state.x77))
```

其中，x 是 pcor()函数计算的偏相关系数，然后是要控制的变量数，最后就是样本数

```
x <- pcor(c(1,5,2,3,6),cov(state.x77))
pcor.test(x,3,50)
```

返回三个值，分别是 t 检验，自由度和 p value

```
$tval
[1] 2.476049

$df
[1] 45

$pvalue
[1] 0.01711252
```

分组数据的相关性检验，这种分组的检验可以使用 t 检验，t 检验使用 t 分布理论，推论差异分布的概率，从而比较两个平均数的差异是否显著。主要用于样本含量较小，一般小于 30 个，总体标准差未知的正态分布数据

这里我们使用 MASS 包中的 UScrime 数据集，它包含了 1960 年美国 47 各州的刑罚制度对犯罪率的影响。

首先使用 t.test()进行独立样本的 t 检验，t.test(y~x)，y 是一个数值型变量，x 是类别型变量，

```
t.test(Prob ~ So,data = UScrime)
```

```
data:  Prob by So
t = -3.8954, df = 24.925, p-value = 0.0006506
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.03852569 -0.01187439
sample estimates:
mean in group 0 mean in group 1
     0.03851265      0.06371269
```

P<0.05，因此可以拒绝南方各州北方各州拥有相同犯罪率的假设。

如果不满足正态分布，就需要用非参数的方法，非参数检验在总体方差未知或者知道甚少的情况下，利用样本数据对总体分布形态等进行推断的方法，由于非参数检验方法在推断过程中不涉及有关总体分布的参数，因而得名为“非参数检验”

参数检验是在总体分布形式已知的情况下，对总体分布的参数如均值、方差等进行推断的方法，也就是数据分布已知，比如满足正态分布

函数介绍——绘图函数

R 语言四大作图系统

基础绘图系统、lattice 包、ggplot2 包、grid 包

可以使用 demo(graphics)命令显示出 R 内置绘图函数可以做的一些图

R 基础绘图系统包括两种绘图，高级绘图是一步到位可以直接绘制出图，而第几回吐，不能单独使用，必须在高级绘图产生图形的基础上，对图形进行调整，比如加一条线，加上标题文字等。

对于绘图函数要注意的是输入数据的格式

Plot()函数

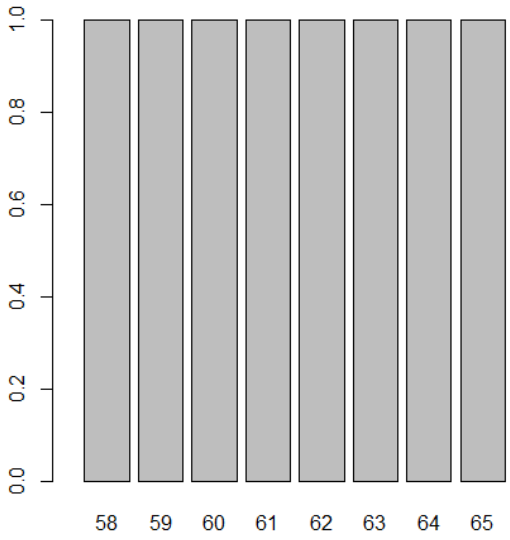
Plot()函数可以接受一个单独的数值向量，例如：

```
plot(women$height)
```

绘制的是散点图

如果输入的数据是因子，绘制出来的则是直方图：

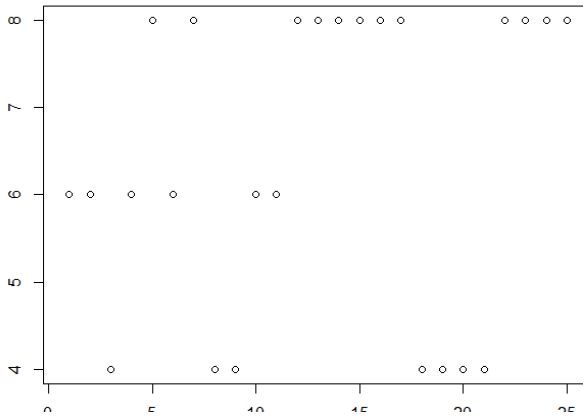
```
plot(as.factor(women$height))
```



再使用 mtcars 数据集看看：

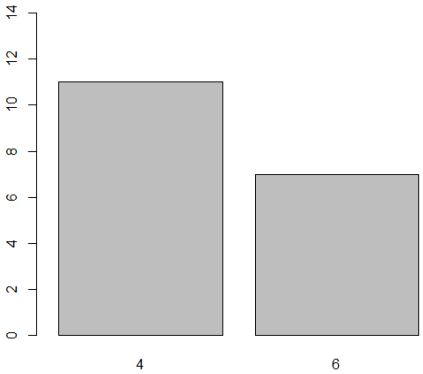
```
plot(mtcars$cyl)
```

直接绘图就是散点图：



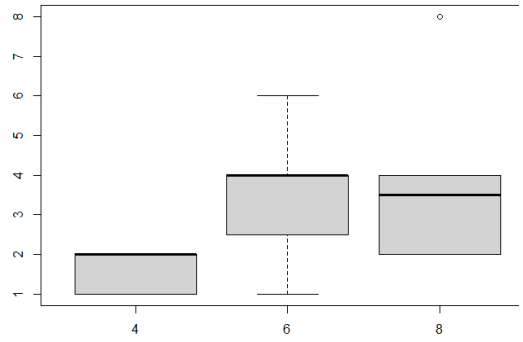
转换为因子数据输入就是直方图：

```
plot(as.factor(mtcars$cyl))
```



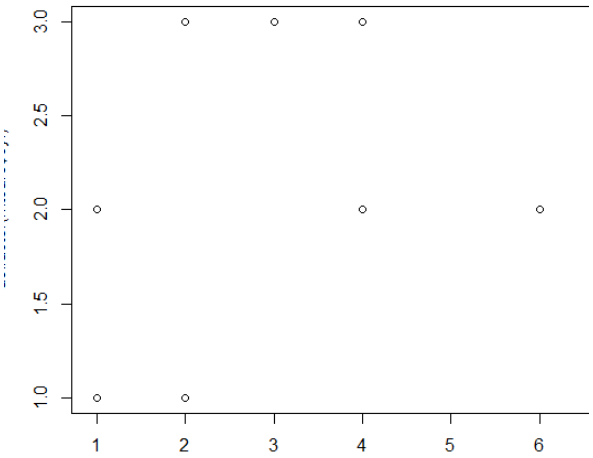
如果输入数据是一个因子数据和一个数值数据，则是箱线图：

```
plot(as.factor(mtcars$cyl),mtcars$carb)
```



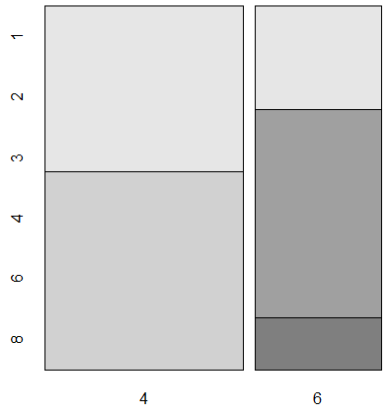
如果输入第一个是数值数据，第二个是因子，输出的是散列图：

```
plot(mtcars$carb,as.factor(mtcars$cyl))
```



如果两个输入数据都是因子，输出的就是棘状图：

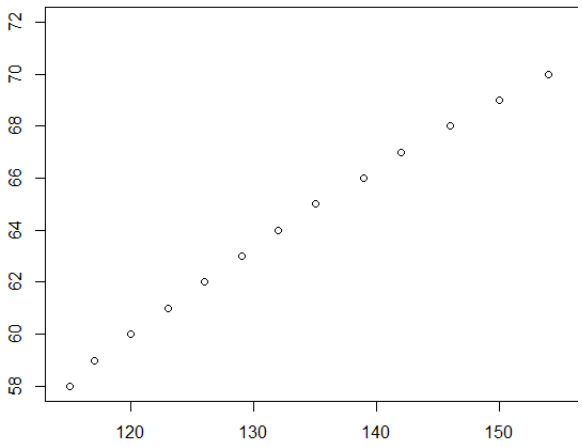
```
> plot(as.factor(mtcars$cyl),as.factor(mtcars$carb))
```



Plot()函数还可以接受函数公式：

```
plot(women$height~women$weight)
```

输出的是二者的关系图：



还可以用 plot()函数直接绘制线性回归的结果：

```
fit <- lm(height~weight,data=women)
```

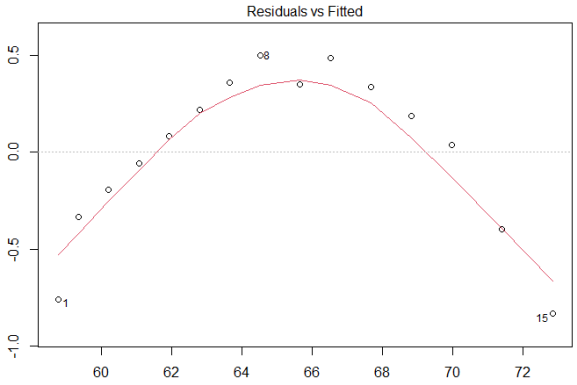
fit 的值是线性回归的结果：

```
Call:
lm(formula = height ~ weight, data = women)

Coefficients:
(Intercept)       weight
    25.7235       0.2872
```

直接用：

```
plot(fit)
```



Plot()函数支持多种属性的数据格式。

Par()函数：

Par()并不能直接接用来绘图，而是对绘图参数进行调整

```
par()
```

直接敲 par()，会跳出关于绘图的所有参数设置

例如我们绘制 mtcars 数据的因子数据图时，加上颜色：

```
plot(as.factor(mtcars$cyl),col=c("red","green","blue"))
```

函数介绍——自定义函数

编写函数就是为了减少重复代码的书写，从而让脚本更加简洁高效，增加可读性

在 R 中，如果直接写函数不加括号，就会显示函数的源代码

一个完整的 R 函数需要包括函数名称、函数声明、函数参数、函数体

函数名称：函数的命名最好和函数的功能相关，可以使字母和数字的组合，但必须是字母开

头
函数声明：利用 function 函数来声明，用来告诉 R 这个东西是函数

下面我们来编写一个简单的 R 函数，这个函数的功能是计算偏度和峰度值：
偏度（skewness），是统计数据分布偏斜方向和程度的度量，是统计数据分布非对称程度的数字特征
峰度（peakness; kurtosis），又称峰态系数，表征概率密度分布曲线在平均值处峰值高低的特征数

```
mystats <- function(x,na.omit=FALSE) {
  if(na.omit)
    x <- x[!is.na(x)]
  m <- mean(x)
  n <- length(x)
  s <- sd(x)
  skew <- sum((x-m^3/s^3))/n
  kurt <- sum((x-m^4/s^4))/n-3
  return(c(n=m,mean=m,stdev=s,skew=skew,kurtosis=kurt))
 }
```

第一个选项参数 x 就是要计算的数据，是一个数值向量，第二个选项是 na.omit，用于删除缺失值，默认取值为 False，然后是大括号，用于写函数的主体
然后是逻辑判断：
如果有缺失值，那么 x 只取不包含缺失值的 x 的值，！是取反的意思，is.na()是取数据集中的缺失值。
然后我们定义 m 为数值向量 x 的平均值，n 为数值向量 x 的长度，s 为数值向量 x 的标准差
然后写下计算 skew（偏度）的公式：

```
skew <- sum((x-m^3/s^3))/n
```

计算峰度的公式：

```
kurt <- sum((x-m^3/s^4))/n-3
```

最后使用一个 return 函数返回函数的值

```
return(c(n=m,mean=n,stdev=s,skew=skew,kurtosis=kurt))}
```

这里我们要输出的是向量的个数 n，平均值 m，标准差 s，偏度值和峰度值
然后可以使用一下这个函数：

```
> x <- 1:100
> mystats(x)
       n     mean     stdev      skew kurtosis
50.50000 50.50000 29.01149 45.22571 47.31820
```

下面来介绍一下 R 中的循环控制函数：
函数内部通过循环实现向量化操作，循环的三部分：条件判断，是真是假，用于循环执行的结构，表达式
首先看一下 for 循环：

```
for(i in 1:10){print("hello,world")}
```

然后是 while 循环：

```
i=1;while(i<=10){print("hello,world");i=i+1;}
```

分号；表示一个语句完结

```
i=1;while(i<=10){print("hello,world");i=i+2;}
```

i=i+2 时语句会少一半

if else 的结构

```
score=70;if (score>60){print("passed")} else {print("failed")}
```

还可以简写 ifelse

```
ifelse(score>60,print("passed"),print("failed")
```

这节课可以参考用来写 d-dematel 函数

项目实操——数据分析实战

通过实际案例进行数据分析，了解数据分析的实质

项目实操——线性回归（一）

回归，通常指用一个或多个预测变量，也成自变量或者解释变量，来预测响应变量，也称因变量、标效变量或者结果变量的方法

回归分析主要用于分析自变量对因变量的影响

重点是：如何建立模型、抽象出数学公式、哪些因素与模型有关、需要利用多少样品、模型的准确率有多高、在实际运用中还是否有效？

表8-1 回归分析的各种变体

回归类型	用 途
简单线性	用一个量化的解释变量预测一个量化的响应变量
多项式	用一个量化的解释变量预测一个量化的响应变量，模型的关系是n阶多项式
多元线性	用两个或多个量化的解释变量预测一个量化的响应变量
多变量	用一个或多个解释变量预测多个响应变量
Logistic	用一个或多个解释变量预测一个类别型响应变量
泊松	用一个或多个解释变量预测一个代表频数的响应变量
Cox比例风险	用一个或多个解释变量预测一个事件（死亡、失败或旧病复发）发生的时间
时间序列	对误差项相关的时间序列数据建模
非线性	用一个或多个量化的解释变量预测一个量化的响应变量，不过模型是非线性的
非参数	用一个或多个量化的解释变量预测一个量化的响应变量，模型的形式源自数据形式，不事先设定
稳健	用一个或多个量化的解释变量预测一个量化的响应变量，能抵御强影响点的干扰

最简单的线性回归：普通最小二乘回归法（OLS）

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_{i1} + \cdots + \hat{\beta}_k X_{ki} \quad i = 1 \cdots n$$

$\hat{Y}_i$	第i次观测对应的因变量的预测值（具体来讲，它是在已知预测变量值的条件下，对Y分布估计的均值）
X_{ji}	第i次观测对应的第j个预测变量值
$\hat{\beta}_0$	截距项（当所有的预测变量都为0时，Y的预测值）
$\hat{\beta}_j$	预测变量j的回归系数（斜率表示X_j改变一个单位所引起的Y的改变量）

我们可以使用 lm()函数来进行线性回归分析，lm 是 linear model，线性回归模型的简称

这个函数的格式是：

lm(formula, data, subset, weights, na.action, method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE, contrasts = NULL, offset, ...)

formula：是要进行拟合的模型形式，写成一个公式，例如，y ~ ax+b

data：是要使用的数据集，是数据框的格式

表8-2　R表达式中常用的符号

符　号	用　途
~	分隔符号，左边为响应变量，右边为解释变量。例如，要通过x、z和w预测y，代码为y ~ x + z + w
+	分隔预测变量
:	表示预测变量的交互项。例如，要通过x、z及x与z的交互项预测y，代码为y ~ x + z + x:z
*	表示所有可能交互项的简洁方式。代码y~ x * z * w可展开为y ~ x + z + w + x:z + x:w + z:w + x:z:w
^	表示交互项达到某个次数。代码y ~ (x + z + w)^2可展开为y ~ x + z + w + x:z + x:w + z:w
.	表示包含除因变量外的所有变量。例如，若一个数据框包含变量x、y、z和w，代码y~.可展开为y ~ x + z + w
-	减号，表示从等式中移除某个变量。例如，y ~ (x + z + w)^2 - x:w可展开为y ~ x + z + w + x:z + z:w
-1	删除截距项。例如，表达式y ~ x - 1拟合y在x上的回归，并强制直线通过原点
I()	从算术的角度来解释括号中的元素。例如，y~x+(z+w)^2将展开为y~x+z+w+z:w。相反，代码y ~ x + I((z + w)^2)将展开为y ~ x + h，h是一个由z和w的平方和创建的新变量
function	可以在表达式中用的数学函数。例如，log(y) ~ x + z + w表示通过x、z和w来预测log(y)

一般在回归分析中，都喜欢用 fit 这个变量名来定义结果，寻找回归模型的过程被称为拟合
如果后面 data 参数中指定了数据集，那么前面公式中的变量就可以直接写变量名字（注意，因变量在波浪线左边，自变量在右边）

```
fit <- lm(weight ~ height,data = women)
```

回归结果，可以使用 summary()函数查看详细的分析结果：

```
Call:
lm(formula = weight ~ height, data = women)

Coefficients:
(Intercept)       height
     -87.52         3.45
```

```
summary(fit)
```

```
Call:
lm(formula = weight ~ height, data = women)

Residuals:
    Min      1Q  Median      3Q     Max
-1.7333 -1.1333 -0.3833  0.7417  3.1167

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -87.51667    5.93694  -14.74 1.71e-09 ***
height        3.45000    0.09114   37.85 1.09e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.525 on 13 degrees of freedom
Multiple R-squared:  0.991,     Adjusted R-squared:  0.9903
F-statistic:  1433 on 1 and 13 DF,  p-value: 1.091e-14
```

首先是 call 这一列，是列出使用的回归的公式。
然后是 residuals，表示残差，残差是真实值和预测值之间的差，例如数据第一行，真实的值是 58，将 58 代入预测公式，得出的预测值 y，y 与 58 之间的差值就是残差，残差给出了四个值，最小值、最大值、中位值、四分之一的值、四分之三的值，这四个值越小，说明预测模型越精确。
Coefficients：系数项，intercept：截距项（当 x 等于 0 时，与 y 轴的相交点）
Estimate 是项系数的值，pr 就是 pvalue，是假设 x 与 y 不相关时候的概率，这个值也是小于 0.05 比较好，residual standard error 残差标准误，表示残差的标准误差，这个也是越小越好。

```
Multiple R-squared:  0.991,  Adjusted R-squared:  0.9903
```

这两个值称为 R 方判定系数，是衡量模型拟合质量的指标，它是表示回归模型所能解释的响应变量的方差比例，比如此处，就代表这个模型可以解释 99.1%的数据，只有 0.9%的数据不符合这个模型，取值在 0-1 之间，值越大于好。
最后是 F-statistic（F 统计量），这个值说明模型是否显著，也是用 pvalue 来衡量，也是值越小越好。
得出回归模型是：
Weight=3.45*height-87.52
是一个一元一次方程
在线性回归的结果中，一般先看 F 统计量，如果 F 统计量不显著（pvalue 不小于 0.05），那么这个模型就没有价值了，需要重新进行拟合，如果小于 0.05，再看 R 方差，模型能解释多少变量。
项目实操——线性回归（二）
线性拟合常用函数：

表8-3 对拟合线性模型非常有用的其他函数

函 数	用 途
summary()	展示拟合模型的详细结果
coefficients()	列出拟合模型的模型参数（截距项和斜率）
confint()	提供模型参数的置信区间（默认95%）
fitted()	列出拟合模型的预测值
residuals()	列出拟合模型的残差值
anova()	生成一个拟合模型的方差分析表，或者比较两个或更多拟合模型的方差分析表
vcov()	列出模型参数的协方差矩阵
AIC()	输出赤池信息统计量
plot()	生成评价拟合模型的诊断图
predict()	用拟合模型对新的数据集预测响应变量值

使用 predict()函数可以用拟合模型对新的数据集进行预测
直接使用 plot()函数可以对拟合结果进行绘图：

```
plot(fit)
```

会生成四幅图：残差拟合图、正态分布 qq 图、大小位置图以及残差影响图
Abline()函数可以绘制出拟合曲线，但这个命令属于低级绘图命令，必须在高级绘图的基础上完成，我们先绘制身高与体重的散点图：

```
plot(women$height,women$weight)
> abline(fit)
```

最小二乘法的原理就是找到一条直线（拟合直线），使残差平方和最小
一般拟合曲线很少是直线，大部分都是曲线，也就是多项式的回归
还是 women 这个数据集，我们用多项式回归试一下：
先定义一个 fit2 变量
将体重作为因变量，身高与身高的平方作为自变量

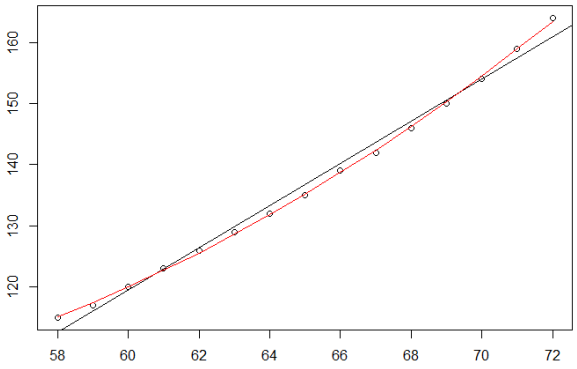
```
fit2 <- lm(weight ~ height+I(height^2),data = women)
```

可以对比两次回归的曲线差异

```
plot(women$height,women$weight)
abline(fit)
```

这次使用 lines()函数，这个函数能把点连成线，横坐标是身高数据，纵坐标是根据拟合模型的得出的预测值，为了增加比较的差异性，我们给第二个曲线增加颜色：

```
lines(women$height,fitted(fit2),col="red")
```

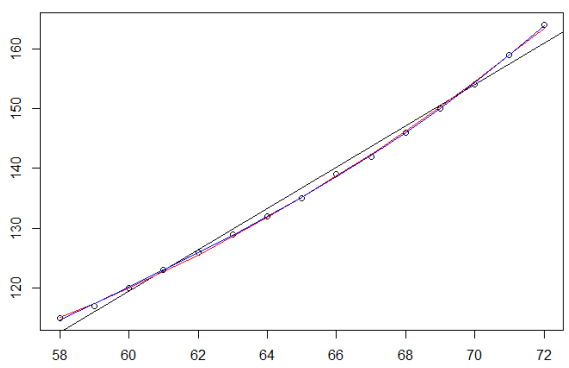


对比很明显，带有二次项的回归模型能够更好的拟合数据，使得更多的点落在曲线上

那么，三次项的回归模型效果是不是更好呢？我们再来拟合一下

```
fit3 <- lm(weight ~ height+I(height^2)+I(height^3),data=women)
plot(women$height,women$weight)
abline(fit)
> lines(women$height,fitted(fit2),col="red")
> lines(women$height,fitted(fit3),col="blue")
```

拟合结果如图：



结果表明，继续增加多项式可以提高拟合度，但是其实没有必要，因为用于拟合的数据集，只是用于建模的数据集，不一定适合真实的数据，过多的拟合也是纸上谈兵。

项目实操——多元线性回归

当预测变量不止一个时，简单现象回归就变成了多元线性回归，相当于求解多元方程，而且和方程式求解不同的是，这些变量的权重还不一样，有些大 ，有些小，有些或者是没有多大影响，下面我们来看一个案例：

以 state.x77 这个数据集为例，求解出一个拟合模型，然后可以根据某些指标预测出犯罪率：

首先我们将 state.x77 这个矩阵数据转化成数据框，因为 lm()函数输入数据必须是数据框的格式

这里为了简化问题，我们只取四个指标进行回归分析：population、illiteracy、income、frost

```
states <- as.data.frame(state.x77[,c("Murder",'Population','Illiteracy','Income',"Frost")])
```

还是使用一个 lm()函数，定义一个 fit 变量，用于存储模型结果

```
fit <- lm(Murder ~ Population+Illiteracy+Income+Frost,data = states)
```

```
Call:
lm(formula = Murder ~ Population + Illiteracy + Income + Frost,
    data = states)

Coefficients:
(Intercept)    Population    Illiteracy        Income         Frost
  1.235e+00     2.237e-04     4.143e+00     6.442e-05     5.813e-04
```

使用 summary()函数查看模型的详细过程结果

```
> summary(fit)
```

```
Call:
lm(formula = Murder ~ Population + Illiteracy + Income + Frost,
    data = states)

Residuals:
    Min      1Q  Median      3Q     Max
-4.7960 -1.6495 -0.0811  1.4815  7.6210

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.235e+00  3.866e+00   0.319   0.7510
Population  2.237e-04  9.052e-05   2.471   0.0173 *
Illiteracy  4.143e+00  8.744e-01   4.738 2.19e-05 ***
Income      6.442e-05  6.837e-04   0.094   0.9253
Frost       5.813e-04  1.005e-02   0.058   0.9541
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.535 on 45 degrees of freedom
Multiple R-squared:  0.567,     Adjusted R-squared:  0.5285
F-statistic: 14.73 on 4 and 45 DF,  p-value: 9.133e-08
```

我们可以使用 coef()函数单独查看各项的系数，根据系数项和截距值就能写出拟合方程

```
coef(fit)
```

```
 (Intercept)    Population    Illiteracy        Income         Frost
1.2345634112 0.0002236754 4.1428365903 0.0000644247 0.0005813055
```

```
options(digits = 4)
```

可以通过 options()的 digits 参数设置显示的位数

```
> coef(fit)
(Intercept)  Population  Illiteracy      Income       Frost
  1.235e+00   2.237e-04   4.143e+00   6.442e-05   5.813e-04
```

下面我们来列举一个更复杂的例子，在很多研究中，变量会有交互项，也就是变量相互之间并不是独立的，例如 mtcars 数据集，汽车的重量与马力之间存在着交互，质量大会影响到马力。交互项就是解释变量之间存在相关性

我们来拟合一下 mtcars 数据集中，每加仑汽油行驶里程数（mpg）变量与马力（hp）以及车重（wt）之间的关系

```
fit <- lm(mpg ~ hp+wt+hp:wt,data = mtcars)
summary(fit)
```

```
Call:
lm(formula = mpg ~ hp + wt + hp:wt, data = mtcars)

Residuals:
   Min     1Q Median     3Q    Max
-3.063 -1.649 -0.736  1.421  4.551

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 49.80842    3.60516   13.82  5.0e-14 ***
hp          -0.12010    0.02470   -4.86  4.0e-05 ***
wt          -8.21662    1.26971   -6.47  5.2e-07 ***
hp:wt        0.02785    0.00742    3.75  0.00081 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.15 on 28 degrees of freedom
Multiple R-squared:  0.885,     Adjusted R-squared:  0.872
F-statistic: 71.7 on 3 and 28 DF,  p-value: 2.98e-13
```

从结果中可以看见，车重与马力的交互项是非常显著的，这就说明响应变量 mpg 与其中一个预测变量的关系依赖于另一个预测变量的水平，其实就是说，每加仑汽油行驶里程数域与汽车马力的关系需要依赖汽车的不同而不同。

对于多元线性回归存在的一个问题就是，如果存在多个变量，就需要考虑变量之间的相互影响和回归时的组合关系，也就是如何从众多可能的模型中选择最佳的模型呢？

比如是不是需要包括所有的变量，还是需要去掉一些对模型贡献不显著的变量，是否需要添加多项式或者是交互项来提高拟合度呢？

那么对于以上的问题，我们可以使用 AIC()函数来比较模型。

AIC()函数是 An Information Criterion 的简称，称为赤池信息准则，这个准则考虑了模型统计拟合度以及用来拟合的参数数目

计算得到的 AIC 值越小越好，越小说明模型用较少的参数就可以获得足够的拟合度

例如上面拟合犯罪度的案例

Ctrl+UP 调出历史代码列表

对比一下两个 fit 的拟合度

```
fit1 <- lm(Murder ~ Population+Illiteracy+Income+Frost,data = states)
fit2 <- lm(Murder ~ Population+Illiteracy,data = states)
AIC(fit1,fit2)
```

```
     df   AIC
fit1  6 241.6
fit2  4 237.7
```

结果显示，fit2 的拟合度更好，但是如果变量数过多，那么组合起来的拟合模型数将是巨大的，再用 AIC()两两比较就不太可行了

这个时候，对于变量的选择可以使用逐步回归法和全子集回归法。

逐步回归法中，模型会依次添加或者删除一个变量直到达到某个节点为止，这个节点就是继续添加或者删除变量，模型不再继续变化。如果每次是增加变量，那么就是向前逐步回归，如果每次是删除变量，那么就是向后逐步回归。

全子集回归法是取所有可能的模型，然后从中计算出最佳的模型，很显然全子集回归法要比逐步回归法要好，因为会检测到所有的模型，把可能的模型都纳入考虑之中，但是缺点就是如果变量数太多，会涉及到大量的计算，运算会比较慢。

这两种方法都需要通过 R 的扩展包来实现，MASS 包中的 stepAIC()函数可以进行逐步回归法的计算

```
> library(MASS)
```

```
> states <- as.data.frame(state.x77[,c("Murder",'Population','Illiteracy','Income',"Frost")])
> fit <- lm(Murder ~ Population+Illiteracy+Income+Frost,data = states)
> stepAIC(fit,direction = "backward")
```

```
Start:  AIC=97.75
Murder ~ Population + Illiteracy + Income + Frost

             Df Sum of Sq RSS   AIC
- Frost       1       0.0 289  95.8
- Income      1       0.1 289  95.8
<none>                   289  97.7
- Population  1      39.2 328 102.1
- Illiteracy  1     144.3 433 116.0

Step:  AIC=95.75
Murder ~ Population + Illiteracy + Income

             Df Sum of Sq RSS   AIC
- Income      1       0.1 289  93.8
<none>                   289  95.8
- Population  1      43.7 333 100.8
- Illiteracy  1     236.2 525 123.6

Step:  AIC=93.76
Murder ~ Population + Illiteracy

             Df Sum of Sq RSS   AIC
<none>                   289  93.8
- Population  1      48.5 338  99.5
- Illiteracy  1     299.6 589 127.3

Call:
lm(formula = Murder ~ Population + Illiteracy, data = states)

Coefficients:
(Intercept)   Population   Illiteracy
   1.651550     0.000224     4.080737
```

Leaps 包中的 regsubsets()函数可以进行全子集回归的计算

但是拟合效果最佳但没有实际意义的模型是没有用的，所以我们始终要对数据有所了解。

项目实操——回归诊断

找到回归模型之后，我们仍需要解决以下的问题：

这个模型是否是最佳的模型？

模型最大程度满足 OLS 模型的统计假设？

模型是否经得起更多数据的检验？

如果拟合出来的模型指标不好，该如何继续下去？

我们需要从多个维度对回归分析的结果进行诊断

除了利用 summary()函数统计出来的各个指标进行检验，还可以用 plot()函数进行绘图，可以生成评价拟合模型的四幅图

下面我们来举个例子：

我们还是使用 women 这个数据集：

```
fit <- lm(weight ~ height,data = women)
par(mfrow=c(2,2))
plot(fit)
```

#par()函数可以修改 plot()函数的绘图参数，par()函数的 mfrow 选项参数可以定制图形的分布，默认是一张图上显示一幅图，可以使这个参数等于 c 这个向量，`mfrow=c(2,2)`表示横排显示两幅，纵排显示两幅，这样就可以将四幅图显示在一个画面内。

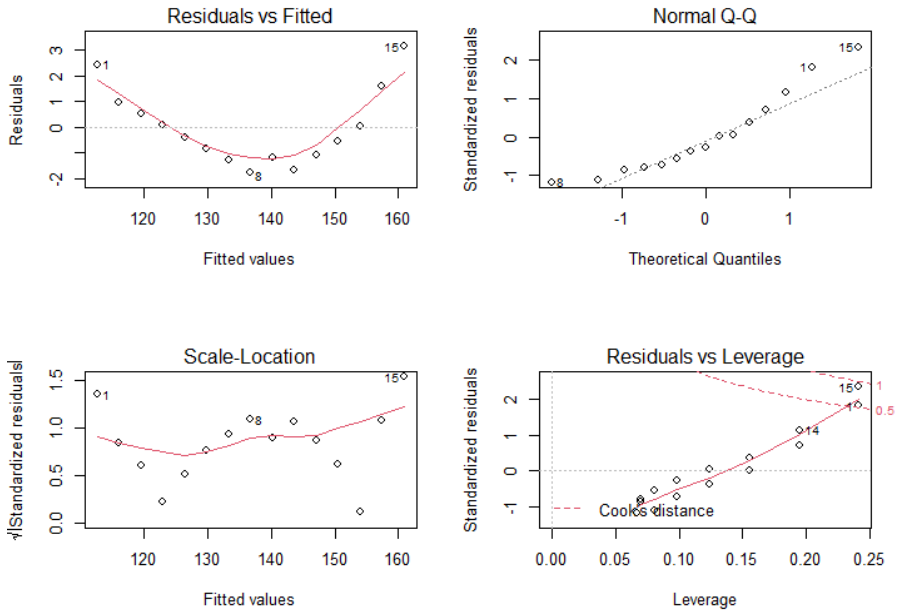
不是所有的数据都可以使用 OLS 模型进行拟合的，需要满足以下的前提条件：

正态性：对于固定的自变量值，因变量需要呈正态的分布

独立性：自变量之间相互独立

线性：因变量与自变量之间为线性相关

同方差性：因变量的方差不随自变量的水平不同而变化，也可以称作不变方差



第一幅图是残差与拟合的图，这幅图用来表示因变量与自变量是否呈线性关系，图中的点是残差值的分布，线为拟合曲线。如果图中是一个曲线的分布，说明可能存在二次项的分布

第二幅图是 R 中比较常见的 qq 图，QQ 图用来描述正态性，如果数据呈正态分布，那么在 QQ 图中就是一条直线

第三幅图是位置与尺寸图，用来描述同方差性，如果满足不变方差的假设，那么图中水平线周围的点应该是随机分布的

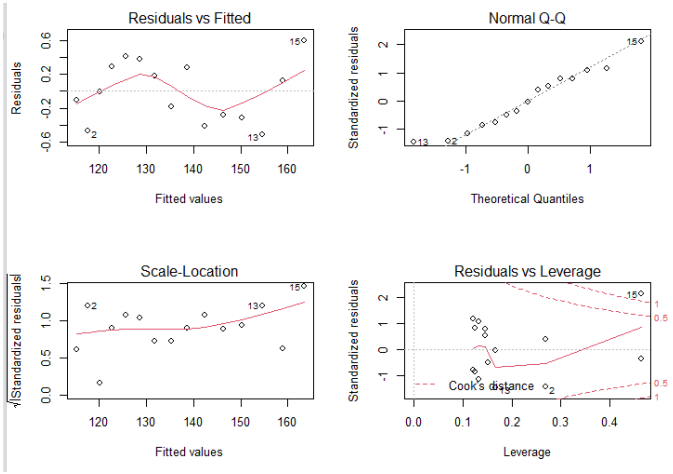
第四幅图是残差与杠杆图，提供了对单个数据值的观测，从图中可以看到哪些点偏差较远，可以用来鉴别离群点、高杠杆点和强影响点，高杠杆点表示它是一个异常的预测变量的组合；强影响点表示这个值对模型参数的估计产生的影响较大，如果删除或者转换就可能违背客观数据的事实

但是这四幅图没办法说明数据是否具有独立性

只能从收集的数据中验证，要判断收集的数据是否独立，我们可以拟合函数再加一个二次项，再绘制一次图看一下。

```
fit2 <- lm(weight ~ height+I(height^2),data = women)
> opar <- par(no.readonly = TRUE)
> par(mfrow=c(2,2))
> plot(fit2)
```

结果明显好一些：



抽样法验证：

1、数据集中有 1000 个样本，随机抽取 500 个数据进行回归分析；

2、模型建好之后，利用 predict()函数，对剩余的 500 个样本进行预测，比较残差值

3、如果预测准确，说明模型可以，否则就需要调整模型。

项目实操——方差分析（一）

方差分析称为 analysis of variance，简称 ANOVA，也称为变异数分析，用于两个及两个以上样本均属差别的显著性检验，从广义上来讲，方差分析也属于回归分析的一种，只不过线性回归的因变量一般是连续型变量，而当自变量是因子时，研究关注的重点通常会从预测转向不同组之间差异的比较，这就是方差分析。

方差分析会大量用在科学研究中，例如实验设计时，进行分组比较，例如药物研究实验，处理组与对照组进行比较

方差分析也分为很多种，比如单因素方差分析，双因素方差分析、协方差分析、多元方差分析、多元协方差分析（R 语言实战这本书中有详细介绍）

方差分析多采用 aov()函数进行分析，aov()函数的用法与 lm()函数类似。下表中列出了 R 中 aov()函数的符号用法：

表9-4　R表达式中的特殊符号

符　号	用　法
~	分隔符号，左边为响应变量，右边为解释变量。例如，用A、B和C预测y，代码为y~ A + B + C
+	分隔解释变量
:	表示变量的交互项。例如，用A、B和A与B的交互项来预测y，代码为y~ A + B + A:B
*	表示所有可能交互项。代码y~ A * B * C可展开为y ~ A + B + C + A:B + A:C + B:C + A:B:C
^	表示交互项达到某个次数。代码y ~ (A + B + C)^2可展开为y ~ A + B + C + A:B + A:C + B:C
.	表示包含除因变量外的所有变量。例如，若一个数据框包含变量y、A、B和C，代码y ~ .可展开为y ~ A + B + C

各种方差分析公式的写法：

表9-5　常见研究设计的表达式

设　计	表　达　式
单因素ANOVA	y ~ A
含单个协变量的单因素ANCOVA	y ~ x + A
双因素ANOVA	y ~ A * B
含两个协变量的双因素ANCOVA	y ~ x1 + x2 + A*B
随机化区组	y ~ B + A（B是区组因子）
单因素组内ANOVA	y ~ A + Error(Subject/A)
含单个组内因子（W）和单个组间因子（B）的重复测量ANOVA	y ~ B * W + Error(Subject/W)

变量的顺序很重要：

顺序很重要！

当自变量与其他自变量或者协变量相关时，没有明确的方法可以评价自变量对因变量的贡献。例如，含因子A、B和因变量y的双因素不平衡因子设计，有三种效应：A和B的主效应，A和B的交互效应。假设你正使用如下表达式对数据进行建模：

```
Y ~ A + B + A:B
```

有三种类型的方法可以分解等式右边各效应对y所解释的方差。

类型I（序贯型）

效应根据表达式中先出现的效应做调整。A不做调整，B根据A调整，A:B交互项根据A和B调整。

类型II（分层型）

效应根据同水平或低水平的效应做调整。A根据B调整，B依据A调整，A:B交互项同时根据A和B调整。

类型III（边界型）

每个效应根据模型其他各效应做相应调整。A根据B和A:B做调整，A:B交互项根据A和B调整。

R默认调用类型I方法，其他软件（比如SAS和SPSS）默认调用类型III方法。

项目实操——方差分析（二）

单因素方差分析：

这里我们使用 multcomp 包中的 cholesterol 数据集进行演示，这个数据集是 50 个患者接受降低胆固醇治疗的五种疗法的数据，50 个患者分为 10 人一组，每天服用 20 克药物这个方案只有治疗方案这一个因子，所以被称为单因素方差分析

首先使用

```
library(multcomp)
attach(cholesterol)
```

这里使用 attach()函数将数据集的列写入内存，这样便可以直接使用数据集的每一页，而不用每次都使用$来引用数据集，也就是表明以下的命令都在数据集下进行。

```
table(trt)
```

使用 table()计算每一列不同因子的频数

```
trt
 1time 2times 4times  drugD  drugE
    10      10      10      10      10
```

接下来使用 aggregate()函数进行分组统计数据的平均值：

```
aggregate(response,by=list(trt),FUN=mean)
```

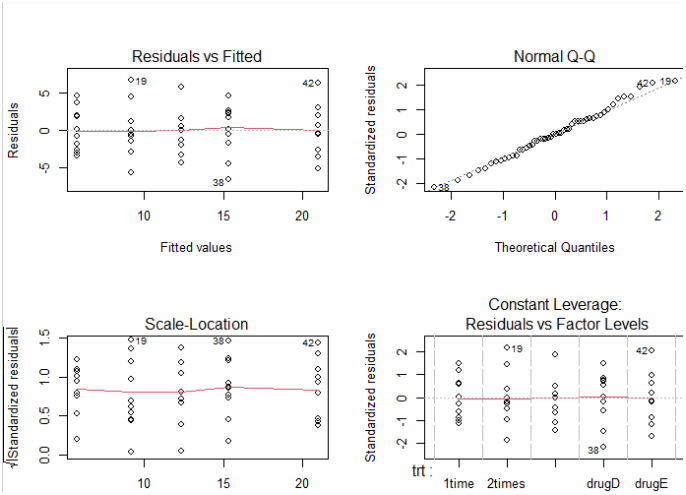
进行方差分析：

```
fit <- aov(response ~ trt,data = cholesterol)
summary(fit)
```

```
            Df Sum Sq Mean Sq F value   Pr(>F)
trt          4 1351.4   337.8   32.43 9.82e-13 ***
Residuals   45  468.8    10.4
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

方差分析的结果主要看 F 值和 P 值，F 值越大说明组间差异越显著，这里就是五组之间平均值的差别，而 P 值只是用来衡量 F 值，P 值越小说明 F 值越可靠，同样可以使用 plot()函数绘制方差的结果：

```
par(mfrow=c(2,2))
> plot(fit)
```



下面我们再使用 lm()函数做一下方差分析，比较一下二者之间的差别：

因为线性拟合要求预测变量是数值型，所以当 lm()函数的预测变量是因子时，就会将一系列与因子水平相对应的数值型对照变量来代替因子，不过 P 值和 F 值的意义是不变的

```
fit.lm <- lm(response ~ trt,data = cholesterol)
summary(fit.lm)
```

```
Call:
lm(formula = response ~ trt, data = cholesterol)

Residuals:
    Min      1Q  Median      3Q     Max
-6.5418 -1.9672 -0.0016  1.8901  6.6008

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    5.782      1.021   5.665 9.78e-07 ***
trt2times      3.443      1.443   2.385   0.0213 *
trt4times      6.593      1.443   4.568 3.82e-05 ***
trtdrugD       9.579      1.443   6.637 3.53e-08 ***
trtdrugE      15.166      1.443  10.507 1.08e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.227 on 45 degrees of freedom
Multiple R-squared:  0.7425,    Adjusted R-squared:  0.7196
F-statistic: 32.43 on 4 and 45 DF,  p-value: 9.819e-13
```

下面介绍一个单因素协方差的例子，选取 multcomp 包中的 litter 数据集，其中 weight 是这个数据集的响应变量：

首先使用 table()函数统计分组情况：

```
table(litter$dose)
```

然后分组统计一下平均数：

```
aggregate(weight,by=list(dose),FUN=mean)
```

```
  Group.1       x
1       0 32.30850
2       5 29.30842
3      50 29.86611
4     500 29.64647
```

使用方差分析检验一下组间是否有显著的差异

越基础性的效应越需要放在表达式前面，最好首先是协变量，然后是主效应，以此类推：

```
summary(fit1)
```

```
           Df Sum Sq Mean Sq F value  Pr(>F)
gesttime    1  134.3  134.30   8.049 0.00597 **
dose        3  137.1   45.71   2.739 0.04988 *
Residuals  69 1151.3   16.69
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

F 值的检验表明，怀孕时间与幼崽出生的体重有关，控制怀孕时间，药物剂量与体重相关，证明药物是有影响的，而不是随机产生的

双因素方差分析案例：

使用内置的 toothgrowth 数据集：

首先使用 xtabs()函数统计频率，结果是一个二维的列联表，再统计一下平均值

```
xtabs(~supp+dose)
```

```
    dose
supp 0.5  1  2
  OJ  10 10 10
  VC  10 10 10
```

```
aggregate(len,by=list(supp,dose),FUN=mean)
```

```
  Group.1 Group.2     x
1      OJ     0.5 13.23
2      VC     0.5  7.98
3      OJ     1.0 22.70
4      VC     1.0 16.77
5      OJ     2.0 26.06
6      VC     2.0 26.14
```

分组统计平均数的结果显示，剂量越小，二者的差异越明显

使用 factor()函数将 dose 这列的数据转化为因子数据，在进行方差分析：

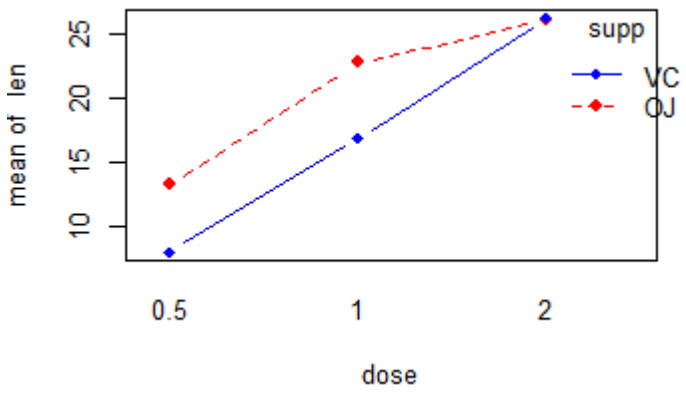
两个变量的方差分析再加上两个变量还有交互，所以进行方差分析的时候，应该用*号连接两个自变量：

```
fit <- aov(len ~ supp*dose,data = ToothGrowth)
```

```
> summary(fit)
            Df Sum Sq Mean Sq F value   Pr(>F)
supp         1  205.4   205.4  15.572 0.000231 ***
dose         2 2426.4  1213.2  92.000  < 2e-16 ***
supp:dose    2  108.3    54.2   4.107 0.021860 *
Residuals   54  712.1    13.2
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

使用“HH”包中的 interaction.plot()函数可以非常直观的对这个结果进行可视化，而且对任意数据因子涉及的主效应和交互效应都进行了展示：

```
interaction.plot(dose,supp,len,type="b",col=c('red','blue'),pch=c(16,18),main='interaction between Dose and Supplement Type')
```



接下来是多元协方差分析的例子：
这里以 MASS 包中的 UScereal 数据集进行展示：
前面的预备操作和上面的类似，先使用 attach()将数据集写入内存，然后将数据中的非因子型数据转化为因子，再进行方差分析。

```
attach(UScereal)
shelf <- factor(shelf)
aggregate(cbind(calories,fat,sugars),by=list(shelf),FUN=mean)
```

#aggregate 只能接受一个变量进行分组，所以这里使用 cbind()将三个因变量组合成一个数据框
这里有三个因变量，一个自变量

```
fit <- manova(cbind(calories,fat,sugars)~ shelf)
summary(fit)
```

```
          Df Pillai approx F num Df den Df    Pr(>F)
shelf      2 0.4021   5.1167      6    122 0.0001015 ***
Residuals 62
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

可以使用 summary.aov()函数分别查看每个变量的结果

项目实操——功效分析

这节课程我们讨论一下，在数据分析的筹备阶段，我们应该选择多少样本，在一个分析中，如果样本数量过小，那么就算 pvalue 值非常小，非常显著，也是不可信的。
功效分析（power analysis）可以帮助在给定置信度的情况下，判断检测到给定效应值时所需的样本量，反过来，它也可以在给定置信度水平的情况下，计算在某样本量内能检测到给定效应值的概率。

表10-1　pwr包中的函数

函　数	功效计算的对象
pwr.2p.test()	两比例（n相等）
pwr.2p2n.test()	两比例（n不相等）
pwr.anova.test()	平衡的单因素ANOVA
pwr.chisq.test()	卡方检验
pwr.f2.test()	广义线性模型
pwr.p.test()	比例（单样本）
pwr.r.test()	相关系数
pwr.t.test()	t检验（单样本、两样本、配对）
pwr.t2n.test()	t检验（n不相等的两样本）

功效分析的理论基础：

		判　断	
		拒绝H_0	不拒绝H_0
真实的	H_0为真	I型错误	正确
	H_0为假	正确	II型错误

第一类错误：弃真，第二类错误：存伪

1、样本大小指的是实验设计中每种条件/组中观测的数目。

2、 显著性水平（也称为alpha）由I型错误的概率来定义。也可以把它看做是发现效应不发生
的概率。

3、功效通过I减去II型错误的概率来定义。我们可以把它看做是真实效应发生的概率。

4、 效应值指的是在备择或研究假设下效应的量。效应值的表达式依赖于假设检验中使用的
统计方法。

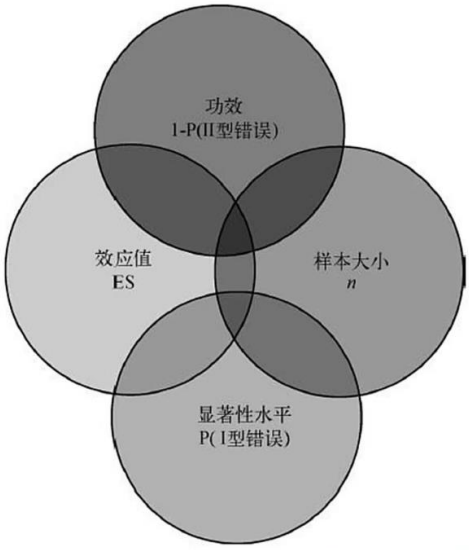


图10-1　在功效分析中研究设计的四个基本量。给定任意三个，你可以推算第四个

所以我们根据要检验的显著性水平、功效和效应值来推算所需要的样品数，R 中利用 pwr 包来进行功效分析。

在 pwr 中包含了多种功效分析的函数，根据不同的假设检验选择不同的函数：

下面介绍线性回归功效分析的案例

$F2=R^2/1-R^2$，即模型解释度（模型方差平方和 ssr）与平均数解释度（误差平方和 sse）之比，F2 效应值越大，样本越小；

V=n-u-1 为误差自由度，与样本数和自变量个数相关，误差自由度越搞，说明样本越多，房差越大，F2 效应值越小，即解释度越小。

U 为自变量个数，与误差自由度正相关，即个数越多，所需的样本越多

Power 功效，一般小于 0.95，但差距不大，排除假阴性的水平之，power 越大，v 就越大

```
pwr.f2.test(u=3,sig.level = 0.05,power = 0.9,f2=0.0769)
```

```
     Multiple regression power calculation

              u = 3
              v = 184.2426
             f2 = 0.0769
      sig.level = 0.05
          power = 0.9
```

结果表明，v=184.2426，也就是说假定显著性水平为 0.05，在 90%置信度的情况下，至少需要 185 个受试者才可以。

下面介绍方差分析功效分析的案例

假设现在两组样品做单因素方差分析，要达到 0.9 的功效，效应值为 0.25，并选择 0.05 的显著性水平，那么每组需要多少样品量呢？可以使用 pwr.anova.test()函数进行分析：其中选项 K 是组的个数，n 是各组的样本大小也就是我们要求的样本量，f 是效应值，sig.level 还是显著性水平，power 为功效水平：

```
pwr.anova.test(k=2,f=0.25,sig.level = 0.05,power = 0.9)
```

```
     Balanced one-way analysis of variance power calculation

              k = 2
              n = 85.03128
              f = 0.25
      sig.level = 0.05
          power = 0.9

NOTE: n is number in each group
```

最终求得 n=85.03，所以每一组中至少要有 86 个样本

项目实操——广义线性模型

在现实生活中，大部分数据都是无规则分布的，要通过数据分析找到规律。

广义线性模型扩展了线性模型的框架，它包含了非正态因变量的分析，在 R 中可以通过 glm()函数进行广义线性回归分析（具体可以参考 R 语言实战 p283）

这里我们使用 breslow 数据集进行演示，这个数据集是遭受轻微或严重性癫痫病的病人的年龄和发病数进行的记录

响应变量是后八周内癫痫病发病数 sumy，预测变量为治疗条件(trt)，年龄(age)以及前八周的基础癫痫病发病数(base)，之所以包含年龄(age)以及发病数(base)，是因为他们对响应变量有潜在的影响，属于协变量，在解释这些协变量之后，才能知道药物治疗对癫痫病的影响

```
data(breslow.dat,package="robust")
summary(breslow.dat)
```

接下来使用 glm()函数进行泊松回归：

```
attach(breslow.dat)
fit <- glm(sumY ~ Base+Age+Trt,data = breslow.dat,family = poiss
on(link = 'log'))
summary(fit)
```

```
Call:
glm(formula = sumY ~ Base + Age + Trt, family = poisson(link = "log"),
    data = breslow.dat)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-6.0569  -2.0433  -0.9397   0.7929  11.0061

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   1.9488259  0.1356191  14.370  < 2e-16 ***
Base          0.0226517  0.0005093  44.476  < 2e-16 ***
Age           0.0227401  0.0040240   5.651 1.59e-08 ***
Trtprogabide -0.1527009  0.0478051  -3.194   0.0014 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 2122.73  on 58  degrees of freedom
Residual deviance:  559.44  on 55  degrees of freedom
AIC: 850.71

Number of Fisher Scoring iterations: 5
```

在线性回归中使用的函数在广义线性模型中同样可以使用，我们可以使用 coef()函数查-/看回归结果的系数项：

```
coef(fit)
```

```
(Intercept)          Base           Age Trtprogabide
 1.94882593    0.02265174    0.02274013   -0.15270095
```

由于泊松回归的系数是对数形式，我们可以再对其取指数：

```
exp(coef(fit))
```

```
(Intercept)          Base           Age Trtprogabide
  7.0204403     1.0229102     1.0230007     0.8583864
```

结果表明，当年龄增加一岁时，平均数的癫痫发病数将乘以 1.023，也就是癫痫发病数的期望值将乘以 1.023，这就代表，年龄的增加与较高的癫痫发病率存在关联。

项目实操——logistic 回归

当通过一系列连续型或类别型预测变量来预测而执行结果变量时，logistic 回归时一个非常有用的工具。

这里我们以 affair 包中的数据为例来阐述 logistics 回归的过程：

先导入数据集，再使用 summary()函数做一个简单的统计，之后使用 table()函数统计一下出轨的频数，prop.table()函数统计一下频率：

```
data(Affairs,package="AER")
summary(Affairs)
table(Affairs$affairs)
> prop.table(table(Affairs$affairs))
```

```
         0          1          2          3          7         12
0.75041597 0.05657238 0.02828619 0.03161398 0.06988353 0.06322795
```

性别的频数：

```
prop.table(table(Affairs$gender))
```

```
   female      male
0.5241265 0.4758735
```

Logistic 回归需要结果是二值型的，所以我们将其转为二值型的结果，定义一个 ynaffairs（年度出轨次数）：

```
Affairs$ynaffairs[Affairs$affairs>0] <- 1
> Affairs$ynaffairs[Affairs$affairs==0] <- 0
```

可以用过 head()函数查看字段是否添加成功

```
> head(Affairs)
   affairs gender age yearsmarried children religiousness education occupation rating ynaffairs
4        0   male  37        10.00       no             3        18          7      4         0
5        0 female  27         4.00       no             4        14          6      4         0
11       0 female  32        15.00      yes             1        12          1      4         0
16       0   male  57        15.00      yes             5        18          6      5         0
23       0   male  22         0.75       no             2        17          6      3         0
29       0 female  32         1.50       no             2        17          5      5         0
```

接下来将这个变量转化为因子：

```
 Affairs$ynaffairs <- factor(Affairs$ynaffairs,levels = c(0,1),
labels = c("NO","Yes"))
> table(Affairs$ynaffairs)

 NO Yes
451 150
```

现在 ynaffairs 就属于二值型的变量，可以进行 logistic 回归分析了（这一步去其实是把离散值转化为二项分布）

使用 glm()函数进行 logistic 回归分析（这个数据集的变量比较多，先使用 attach()函数将数据集写入内存，使用 R 的自动补齐功能）：

```
fit <- glm(ynaffairs ~ gender+age+yearsmarried+children+religiousness+education+occupation+rating,data = Affairs,family = binomial(link=logit))
summary(fit)
```

```
Call:
glm(formula = ynaffairs ~ gender + age + yearsmarried + children +
    religiousness + education + occupation + rating, family = binomial(link = logit),
    data = Affairs)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.5713  -0.7499  -0.5690  -0.2539   2.5191

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)    1.37726    0.88776   1.551 0.120807
gendermale     0.28029    0.23909   1.172 0.241083
age           -0.04426    0.01825  -2.425 0.015301 *
yearsmarried   0.09477    0.03221   2.942 0.003262 **
childrenyes    0.39767    0.29151   1.364 0.172508
religiousness -0.32472    0.08975  -3.618 0.000297 ***
education      0.02105    0.05051   0.417 0.676851
occupation     0.03092    0.07178   0.431 0.666630
rating        -0.46845    0.09091  -5.153 2.56e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 675.38  on 600  degrees of freedom
Residual deviance: 609.51  on 592  degrees of freedom
AIC: 627.51

Number of Fisher Scoring iterations: 4
```

根据回归结果，将不显著的变量去除，重新拟合一次：

```
> fit1 <- glm(ynaffairs ~ age+yearsmarried+religiousness+rating,data = Affairs,family = binomial(link=logit))
```

```
> summary(fit1)

Call:
glm(formula = ynaffairs ~ age + yearsmarried + religiousness +
    rating, family = binomial(link = logit), data = Affairs)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6278  -0.7550  -0.5701  -0.2624   2.3998

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   1.93083    0.61032   3.164 0.001558 **
age          -0.03527    0.01736  -2.032 0.042127 *
yearsmarried  0.10062    0.02921   3.445 0.000571 ***
religiousness -0.32902    0.08945  -3.678 0.000235 ***
rating       -0.46136    0.08884  -5.193 2.06e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 675.38  on 600  degrees of freedom
Residual deviance: 615.36  on 596  degrees of freedom
AIC: 625.36

Number of Fisher Scoring iterations: 4
```

由于两个回归结果是嵌套关系，可以使用 anova 对两者进行方差分析，对于广义线性回归，可以使用卡方检验：

```
anova(fit,fit1,test = "Chisq")
```

卡方检验结果值并不显著

```
Analysis of Deviance Table

Model 1: ynaffairs ~ gender + age + yearsmarried + children + religiousness +
    education + occupation + rating
Model 2: ynaffairs ~ age + yearsmarried + religiousness + rating
  Resid. Df Resid. Dev Df Deviance Pr(>Chi)
1       592     609.51
2       596     615.36 -4  -5.8474   0.2108
```

证明两次结果差别不大，证明两次拟合的结果一样好，说明性别、教育程度等不显著变量不会显著的提高方程的预测精度。

在 logistic 回归中，响应变量是 y=1 的对数优势比，回归系数的含义是，当其他预测变量不变时，一单位预测变量的变化可能引起的响应变量对数优势比的变化，对回归系数取指数，就能回归正常的数值。

```
> coef(fit1)
  (Intercept)           age  yearsmarried religiousness        rating
   1.93083017   -0.03527112    0.10062274   -0.32902386   -0.46136144
> exp(coef(fit1))
  (Intercept)           age  yearsmarried religiousness        rating
    6.8952321     0.9653437     1.1058594     0.7196258     0.6304248
>
```

可以使用 predict()函数根据拟合模型的结果（如此处的 fit1）对新数据进行验证，首先创建一个包含你感兴趣的预测变量值的虚拟数据集，testdata，这里为了方便，我们都取数据的平均值。

```
testdata <- data.frame(rating=c(1,2,3,4,5),age=mean(Affairs$ag
e),yearsmarried=mean(Affairs$yearsmarried),religiousness=mean
(Affairs$religiousness))
> head(testdata)
```

```
  rating      age yearsmarried religiousness
1      1 32.48752     8.177696      3.116473
2      2 32.48752     8.177696      3.116473
3      3 32.48752     8.177696      3.116473
4      4 32.48752     8.177696      3.116473
5      5 32.48752     8.177696      3.116473
```

```
testdata$prob <- predict(fit1,newdata = testdata,type = "respon
se")
testdata
```

```
  rating      age yearsmarried religiousness      prob
1      1 32.48752     8.177696      3.116473 0.5302296
2      2 32.48752     8.177696      3.116473 0.4157377
3      3 32.48752     8.177696      3.116473 0.3096712
4      4 32.48752     8.177696      3.116473 0.2204547
5      5 32.48752     8.177696      3.116473 0.1513079
```

下面我们再看看年龄的影响，重新生成一个测试数据：

```
testdata <- data.frame(rating=mean(Affairs$rating),age=seq(17,5
7,10),yearsmarried=mean(Affairs$yearsmarried),religiousness=me
an(Affairs$religiousness))
```

#这里 seq(17,57,10)的意思是，17 岁到 57 岁，以 10 为间隔生成年龄数，相当于一个以 10 为差值的年龄等差数列。

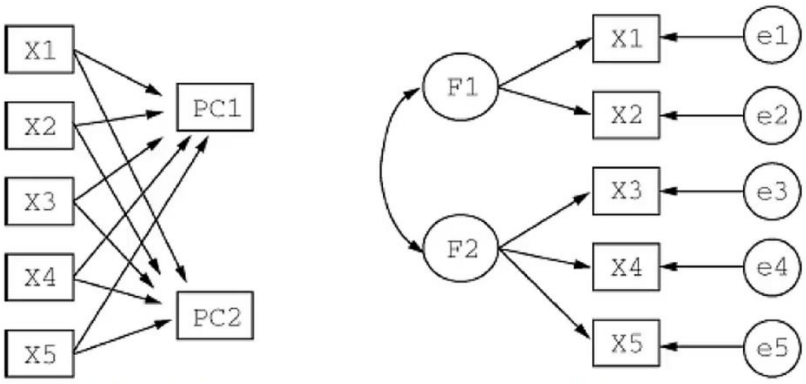
```
testdata$prob <- predict(fit1,newdata = testdata,type = "respon
se")
testdata
```

```
    rating age yearsmarried religiousness      prob
1 3.93178  17     8.177696      3.116473 0.3350834
2 3.93178  27     8.177696      3.116473 0.2615373
3 3.93178  37     8.177696      3.116473 0.1992953
4 3.93178  47     8.177696      3.116473 0.1488796
5 3.93178  57     8.177696      3.116473 0.1094738
```

结果显示，当其他变量的增长，婚外情的概率将从 0.31 降到 0.10，利用这种方法可以研究每一个预测变量对结果概率的影响

项目实操——主成分分析

主成分分析和因子分析都是用来探索和简化多变量复杂分析的方法。主成分分析，也简称为 PCA，是一种数据降维技巧，它能将大量相关变量转化为一组很少的不相关变量，这些无关的变量成为主成分，主成分其实是对原始变量重新进行线性组合，将原先众多具有一定相关性的指标，重新组合为一组的新的相互独立的综合指标。



(a) 主成分分析模型　　(b) 因子分析模型

主成分分析和因子分析模型。图中展示了可观测变量（X1到X5）、主成分（PC1、PC2）、因子（F1、F2）和误差（e1到e5）

R 中内置的 printcomp()函数可以进行主成分分析，这里我们使用 psych 包进行分析

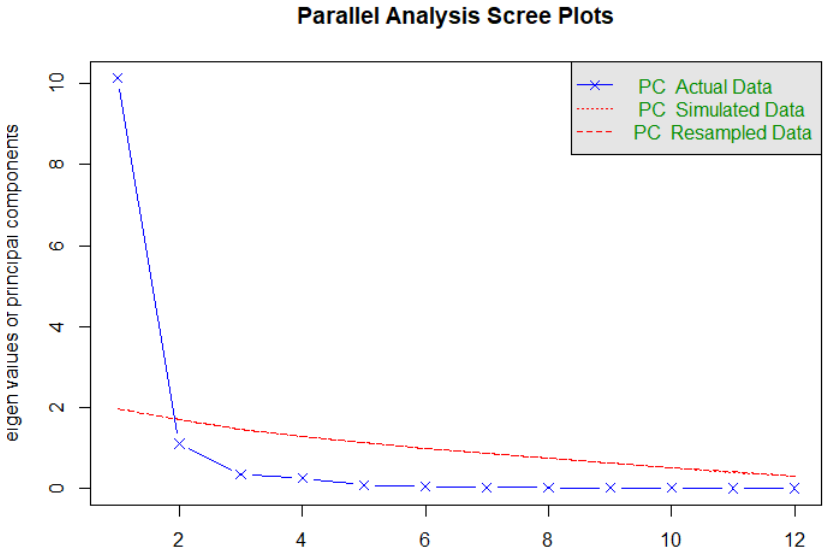
主成分分析与因子分析的步骤：

数据预处理、选择分析模型、判断要选择的主成分/因子数目、选择主成分/因子、旋转主成分/因子、解释结果、计算主成分或因子得分。

这里我们使用 USjudgeratings 数据集进行举例分析：

选择 PCA 分析，通过绘制碎石图选择需要的主成分数目：

```
fa.parallel(USJudgeRatings,fa="pc",n.iter = 100)
```



接下来使用 principle()函数进行主成分分析,nfactors 是主成分因子的数目，rotate 是旋转角度，scores 表示是否要计算主成分得分，默认为不需要：

```
pc <- principal(USJudgeRatings,nfactors = 1,rotate = "none",sco
res = FALSE)
```

```
Principal Components Analysis
Call: principal(r = USJudgeRatings, nfactors = 1, rotate = "none",
    scores = FALSE)
Standardized loadings (pattern matrix) based upon correlation matrix
       PC1      h2     u2 com
CONT -0.01 9.6e-05 0.9999   1
INTG  0.92 8.4e-01 0.1563   1
DMNR  0.91 8.3e-01 0.1660   1
DILG  0.97 9.4e-01 0.0613   1
CFMG  0.96 9.3e-01 0.0723   1
DECI  0.96 9.2e-01 0.0764   1
PREP  0.98 9.7e-01 0.0299   1
FAMI  0.98 9.5e-01 0.0469   1
ORAL  1.00 9.9e-01 0.0091   1
WRIT  0.99 9.8e-01 0.0195   1
PHYS  0.89 8.0e-01 0.2014   1
RTEN  0.99 9.7e-01 0.0275   1

                PC1
SS loadings    10.13
Proportion Var  0.84

Mean item complexity =  1
Test of the hypothesis that 1 component is sufficient.

The root mean square of the residuals (RMSR) is  0.05
 with the empirical chi square  12.56  with prob <  1

Fit based upon off diagonal values = 1
```

这就是PCA分析的结果，其中，pc1栏是指观测变量与主成分的相关系数，如果nfactors=2或者3，那么还会有pc2、pc3等主成分，h2栏指成分公因子的方差，是主成分对每个变量的方差解释度，u2一栏是成分唯一性，方差不能被主成分解释的比例，proportion var表示每个主成分对数据集的解释程度，这里可以看到第一主成分pc1解释了所有变量84%的方差，我们将score参数设置为true，就可以获得每个变量的得分

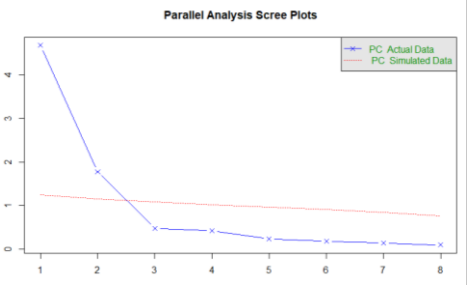
接下来我们使用Harman23.cor数据集进行分析：

表14-3　305个女孩的身体指标间的相关系数（Harman23.cor）

	身　高	指　距	前　臂	小　腿	体　重	股骨转子间径	胸　围	胸　宽
身高	1.00	0.85	0.80	0.86	0.47	0.40	0.30	0.38
指距	0.85	1.00	0.88	0.83	0.38	0.33	0.28	0.41
前臂	0.80	0.88	1.00	0.80	0.38	0.32	0.24	0.34
小腿	0.86	0.83	0.80	1.00	0.44	0.33	0.33	0.36
体重	0.47	0.38	0.38	0.44	1.00	0.76	0.73	0.63
股骨转子间径	0.40	0.33	0.32	0.33	0.76	1.00	0.58	0.58
胸围	0.30	0.28	0.24	0.33	0.73	0.58	1.00	0.54
胸宽	0.38	0.41	0.34	0.36	0.63	0.58	0.54	1.00

在这个数据集中，数据是由变量的相关系数组成而非原始的数据集，先进行筛选主成分因子，利用平行分析得出碎石图：

```
fa.parallel(Harman23.cor$cov,n.obs = 302,fa="pc",n.iter = 100)
```



只有两个x在y=1之上，所以选择两个主成分因子，接下来进行主成分的分析

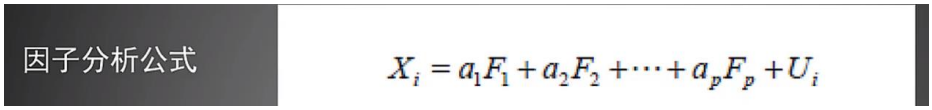
下面介绍一下主成分的旋转（这里只展示代码，具体的统计学知识查看统计学书本以及

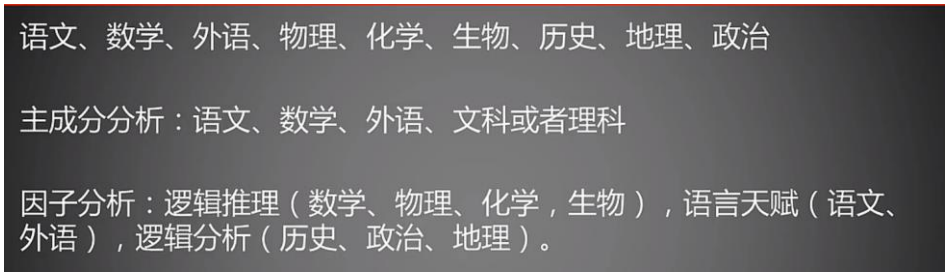
R 语言实战 p303）

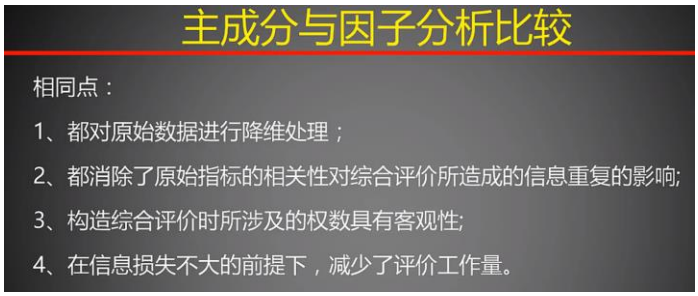
```
pc <- principal(Harman23.cor$cov,nfactors = 2,rotate = 'Varimax')
```

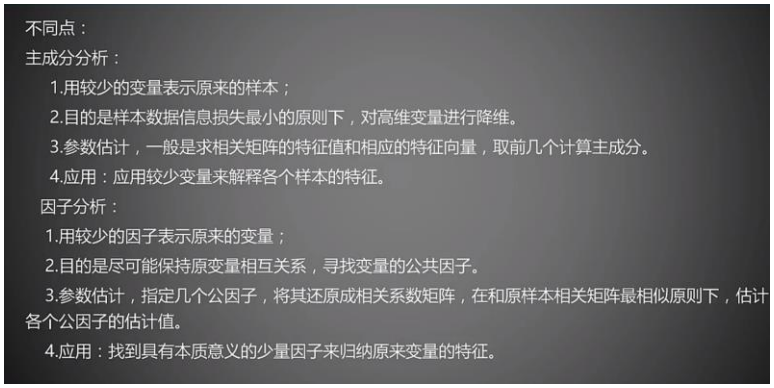
项目实操——因子分析

探索性因子分析法（exploratory factor analysis），简称 EFA，是一系列用来发现一组变量的潜在结构的方法，它通过寻找一组更小的、潜在的或隐藏的结构来解释已观测到的、显式的变量间的关系。









下面列举一个因子分析的案例：

这里我们使用 factanal()函数进行因子分析，使用 ability.cov 数据集进行演示：

首先对数据集进行处理，利用 option()函数将数据保留两位小数

```
options(digits = 2)
covariances <- ability.cov$cov#定义 covariences 变量，取数据集中的 cov 列，因为 ability.cov 数据集是一个列表
```

转化为系数矩阵（原来表里面的数据是方差，然后用 cov2cor()函数转化为相关系数）：

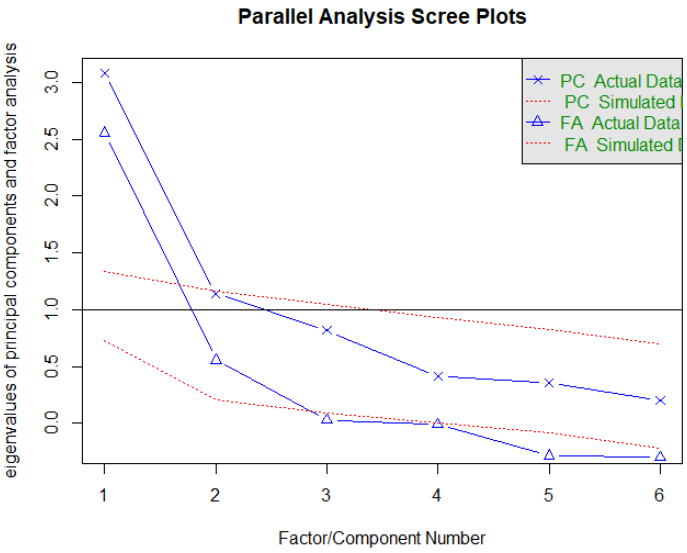
```
correlations <- cov2cor(covariances)
```

	general	picture	blocks	maze	reading	vocab
general	1.00	0.47	0.55	0.34	0.58	0.51
picture	0.47	1.00	0.57	0.19	0.26	0.24
blocks	0.55	0.57	1.00	0.45	0.35	0.36
maze	0.34	0.19	0.45	1.00	0.18	0.22
reading	0.58	0.26	0.35	0.18	1.00	0.79
vocab	0.51	0.24	0.36	0.22	0.79	1.00

同样使用 fa.parallel()函数判断参与分析的因子个数，这里我们把选项参数 fa 设置为‘Both’表示即研究主成分也研究因子分析：

```
fa.parallel(correlations,fa="both",n.obs = 112,n.iter = 100)
```

n.obs 是观测的数量，也就是样本数，可以通过 abilitycov$n.cobs 计算得出



观察图中结果，显示要提取两个因子。FA 分析是看特征值数大于 0，pc 看特征值数大于 1

```
fa <- fa(correlations,nfactors = 2,rotate = "none",fm="pa")
#nfactors 是指因子数，rotate 是指需不需要旋转，fm 是用于进行因子分析的方法
```

```
> fa
Factor Analysis using method =  pa
Call: fa(r = correlations, nfactors = 2, rotate = "none", fm = "pa")
Standardized loadings (pattern matrix) based upon correlation matrix
         PA1   PA2   h2    u2 com
general 0.75  0.07 0.57 0.432 1.0
picture 0.52  0.32 0.38 0.623 1.7
blocks  0.75  0.52 0.83 0.166 1.8
maze    0.39  0.22 0.20 0.798 1.6
reading 0.81 -0.51 0.91 0.089 1.7
vocab   0.73 -0.39 0.69 0.313 1.5

                       PA1  PA2
SS loadings           2.75 0.83
Proportion Var        0.46 0.14
Cumulative Var        0.46 0.60
Proportion Explained  0.77 0.23
Cumulative Proportion 0.77 1.00

Mean item complexity =  1.5
Test of the hypothesis that 2 factors are sufficient.

The degrees of freedom for the null model are  15  and the objective function was  2.5
The degrees of freedom for the model are 4  and the objective function was  0.07

The root mean square of the residuals (RMSR) is  0.03
The df corrected root mean square of the residuals is  0.06

Fit based upon off diagonal values = 0.99
Measures of factor score adequacy
                                                 PA1  PA2
Correlation of (regression) scores with factors   0.96 0.92
Multiple R square of scores with factors          0.93 0.84
Minimum correlation of possible factor scores     0.86 0.68
```

正交旋转法：两个因子之间不相关，斜交旋转法：两个因子之间相关

（后面的多看书-R 语言实战）

项目实操——购物篮分析

下面我们使用 arules 包以及 groceries 数据集进行演示：

由于这个数据没有办法直接在 r 中展示出来，我们需要使用 inspect()函数查看数据集的内容，展现出来的效果是类似于商品小票一样的数据

使用 arules 包中的 apriori()函数进行分析建模，这个函数就相当于线性回归分析中的 lm()或者是 aov()函数，是关联规则挖掘算法

Support=0.01 表示最小支持度是 0.01，confidence=0.5 代表最小置信度是 50%

```
fit <-
apriori(Groceries,parameter=list(support=0.01,confidence=0.5))
```

通过 summary()函数来查看统计信息，用 inspect()函数查看给出的规则：

```
> inspect(fit)
     lhs                                rhs                support    confidence coverage   lift     count
[1]  {curd,yogurt}                   => {whole milk}       0.01006609 0.5823529  0.01728521 2.279125  99
[2]  {other vegetables,butter}       => {whole milk}       0.01148958 0.5736041  0.02003050 2.244885 113
[3]  {other vegetables,domestic eggs} => {whole milk}      0.01230300 0.5525114  0.02226741 2.162336 121
[4]  {yogurt,whipped/sour cream}     => {whole milk}       0.01087951 0.5245098  0.02074225 2.052747 107
[5]  {other vegetables,whipped/sour cream} => {whole milk} 0.01464159 0.5070423  0.02887646 1.984385 144
[6]  {pip fruit,other vegetables}    => {whole milk}       0.01352313 0.5175097  0.02613116 2.025351 133
[7]  {citrus fruit,root vegetables}  => {other vegetables} 0.01037112 0.5862069  0.01769192 3.029608 102
[8]  {tropical fruit,root vegetables} => {other vegetables} 0.01230300 0.5845411  0.02104728 3.020999 121
[9]  {tropical fruit,root vegetables} => {whole milk}      0.01199797 0.5700483  0.02104728 2.230969 118
[10] {tropical fruit,yogurt}         => {whole milk}       0.01514997 0.5173611  0.02928317 2.024770 149
[11] {root vegetables,yogurt}        => {other vegetables} 0.01291307 0.5000000  0.02582613 2.584078 127
[12] {root vegetables,yogurt}        => {whole milk}       0.01453991 0.5629921  0.02582613 2.203354 143
[13] {root vegetables,rolls/buns}    => {other vegetables} 0.01220132 0.5020921  0.02430097 2.594890 120
[14] {root vegetables,rolls/buns}    => {whole milk}       0.01270971 0.5230126  0.02430097 2.046888 125
[15] {other vegetables,yogurt}       => {whole milk}       0.02226741 0.5128806  0.04341637 2.007235 219
```

结果表明，在酸奶出现的地方，购物小票中就一定会出现纯牛奶，支持度达到了 0.01，说明两者的关联很强。