R语言

基本语法

变量定义：`a <- "Hello, world!"`

打印：`print(a)`，打印多个变量时使用`cat(a,b,c)`

索引：从1开始，首尾都包含

注释：`#test`

R不支持多行注释，但可以使用if(FALSE)语句注释

获取包含R包的库位置：`.libPaths()`

获取已安装的所有软件包列表：`library()`

重要范例

Rstudio用法

ctrl+1 光标移动到source

ctrl+2 光标移动到console

Ctrl+L 清理控制台

command+shift+c 批量注释或取消注释

文件

读文件

```
data = read.csv(paste0(getwd(),'/data/data.csv'))
```

写文件

`write.csv(data,"data/data.csv",row.names = FALSE)`

字符

字符处理

`paste(c("data","csv"),collapse = ".")`

添加列

`data[,code := gsub(x = contract, pattern = "[^/]+/([^\\./]+)\\.[^\\./]+", replacement = "\\1")]`

时间戳格式转时间格式

`as.POSIXct(plotData[,timestamp],origin = '1970-01-01')`

数据帧

筛选

`dataUSDT <- data[(grepl("\\.usdt$",contract)  & exchange != 'coinbase')]`

根据某列分列表

`split(dataUSDT, by = 'exchange',keep.by = TRUE)`

对列表元素依次操作

`lapply(splitedData,function(d)unique(d$code))`

计算时取本组元素

```
V_R_Exc <- dataUSDT[,.(
  code = code,
  time = time,
  v_r_exc = volume/.SD[code == "btc"][['volume']]
),by = c('exchange','timestamp')]
# 按交易所和时间分组，每组的btc交易量为.SD[code == "btc"][['volume']]
```

排列

`arrange(v_r,timestamp,-v_r) # 按照timestamp, vr降序排列`

`v_r <- v_r[order(timestamp, -v_r),] # 按照timestamp, vr降序排列`

缺失值处理

`plotData[is.na(plotData)] <- 0`

`plotData[,v_r:= plotData[,v_r] %>% replace(is.na(.), 0)]`

创造新列使用变量内容而非变量名本身

`result[,(i):=content]`

绘图

横轴为时间

```
ggplot2::ggplot(
     data = plotData, ggplot2::aes(x = time, y = v_r, group = code, colour = code)) +
     ggplot2::geom_line() +
     ggplot2::scale_x_datetime(date_labels = '%Y-%m-%d', date_breaks = '3 month') +
     ggplot2::xlab('Time') +
     ggplot2::ggtitle(label = 'Time ~ Count', subtitle = 'subtitle') +
     ggplot2::ylab('Count') +
     ggplot2::theme(legend.position = "right",
             legend.direction = "vertical",
             axis.text.x = ggplot2::element_text(angle = 45))
```

保存

`ggplot2::ggsave("graph/coinNumber.png")`

数据类型

对象无需声明数据类型，变量定义时R对象的数据类型变为变量的数据类型。

返回数据类型的函数：`class(a)`

Vectors 向量

- 最基本的R语言数据对象
- **单元素向量**
  - 即使在R语言中只写入一个值，它也将成为长度为1的向量，并且属于六类原子向量。
  - c(1)的效果与直接写入1一致
- **多元素向量**
  - 对数值数据使用冒号运算符
    - `v <- 3.8:11.4`
    - `[1]  3.8  4.8  5.8  6.8  7.8  8.8  9.8 10.8`
  - 使用sequence (Seq.)序列运算符
    - `seq(5, 9, by = 0.4)`
    - `[1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0`
  - 使用c()函数进行组合
    - `s <- c('apple','red',5,TRUE)`
    - `[1] "apple" "red" "5" "TRUE"`，类型为character(会强制转换成最宽泛的类)
- **向量索引**
  - 从1开始
  - logical变量或0/1也可用于索引

- FALSE或0代表对应位置舍弃，索引-x代表第x项舍弃后的剩余部分

```
t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
u <- t[c(2,3,6)]
v <- t[c(TRUE,FALSE,FALSE,FALSE,FALSE,TRUE,FALSE)]
x <- t[c(-2,-5)]
y <- t[c(0,0,0,0,0,0,1)]

[1] "Mon" "Tue" "Fri"
[1] "Sun" "Fri"
[1] "Sun" "Tue" "Wed" "Fri" "Sat"
[1] "Sun"
```

- **向量运算**
  - 加减乘除相同长度的向量
  - 若不等长，较短的向量将被循环

```
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11)
#V2 becomes c(4,11,4,11,4,11)
```

- **向量排序**
  - 可以使用sort()函数排序，有参数decreasing默认为false（默认升序）
  - 除数字外也可以进行字符排序

```
v <- c(3,8,4,5,0,11, -9, 304)
sort.result <- sort(v)
revsort.result <- sort(v, decreasing = TRUE)
```

六类原子向量

- **logical**
  - 逻辑型，真或假
  - TRUE, FALSE
- **numeric**
  - 数字
  - 12.3, 5
- **integer**
  - 整型
  - 2L, 0L
- **complex**
  - 复数型
  - 3+2i
- **character**

    - 字符
    - ‘a’, "abc", '23.4'
    - 单双引号无差异 字符和字符串无差异?
    - 字符修改函数

      `gsub(x = contract, pattern = "[^/]+/([^\\./]+)\\.[^\\./]+", replacement = "\\1")`
      表示形如“coinbase/btc.usdt”返回“btc”
    - 字符连接函数

      `paste(c("btc","usdt"), collapse = ".")`
      表示得到“btc.usdt”
    - 字符判定函数

      `grepl("\\.usdt",contract)`
      表示字符串内含有“.usdt”
- **raw**
    - 原型 16进制
    - "Hello" 被存储为 48 65 6c 6c 6f
    - `v <- charToRaw("Hello")`

Lists 列表

- 列表可以包含许多不同类型的元素，如向量、函数、矩阵甚至另一个列表。
- **列表创建**
    - `list1 <- list(c(2,5,3),21.3,sin)`
    - 将显示的结果为：

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

[[2]]   [1] 21.3

[[3]]   function (x)  .Primitive("sin")
```

- **列表元素命名**
    - `names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")`
- **列表元素访问**
    - 元素索引访问：`list_data[1]`
    - 元素名称访问：`list_data$A_Matrix`
- **列表元素修改**
    - 增删只能在末尾，修改可以在任意位置
    - 增：`list_data[4] <- "New element"`
    - 删：`list_data[4] <- NULL`
    - 改：`list_data[3] <- "updated element"`
- **列表合并**

- 将所有列表放在一个c()函数中
- `list <- c(list1,list2)`

- **列表转向量**
  - 转为向量后可使用向量的代数运算
  - unlist()函数，输入列表输出向量
  - `v <- unlist(list1)`

Metrices 矩阵

- 二维矩形数据集，可以使用matrix()创建
- 矩阵的转置：t()，用例为`t(M)`
- **创建矩阵**
  - 基本语法：`matrix(data, nrow, ncol, byrow, dimnames)`
  - 数据是成为矩阵的数据元素的输入向量
  - nrow是要创建的行数
  - ncol是要创建的列数
  - byrow是一个逻辑线索。 如果为TRUE，则输入向量元素按行排列
  - dimname是分配给行和列的名称(可选)

    ```
    rownames = c("row1", "row2", "row3", "row4")
    colnames = c("col1", "col2", "col3")
    P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames,
    colnames))
    ```

  - 将显示的结果为：

    ```
    col1 col2 col3
    ```

    row1 3 4 5
    row2 6 7 8
    row3 9 10 11
    row4 12 13 14

- **访问元素**
  - 访问元素：`P[4,2]`
  - 访问行/列：`P[2,]` 或 `P[,3]`
- **矩阵运算**
  - 参与运算的矩阵行数列数分别相等
  - 结果均为逐项运算
  - `result <- matrix1 / matrix2`

Arrays 数组

- 数组可以有任意维度，使用一个 dim 属性创建所需的维数，向量长度不足的循环补足
- 数组只能存储数据类型
- **数组创建**
  - `a <- array(c('green','yellow'),dim = c(3,3,2))`
  - 将显示的结果为：

```
, , 1

     [,1]     [,2]     [,3]
[1,] "green"  "yellow" "green"
[2,] "yellow" "green"  "yellow"
[3,] "green"  "yellow" "green"

, , 2

     [,1]     [,2]     [,3]
[1,] "yellow" "green"  "yellow"
[2,] "green"  "yellow" "green"
[3,] "yellow" "green"  "yellow"
```

- **行列命名**

```
result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames =
list(row.names,column.names,matrix.names))

, , Matrix1

    COL1 COL2 COL3
ROW1    5   10   13
ROW2    9   11   14
ROW3    3   12   15

, , Matrix2

    COL1 COL2 COL3
ROW1    5   10   13
ROW2    9   11   14
ROW3    3   12   15
```

- **元素访问**
  - 反正空着就是全都要
  - `result[3,,2]`
- **元素操作**
  - 对数组元素的操作通过访问矩阵的元素来执行

```
matrix1 <- array1[,,2]
matrix2 <- array2[,,2]
result <- matrix1+matrix2
```

- **跨矩阵元素运算**
  - apply()函数：`apply(x, margin, fun)`
    - x是一个数组
    - *margin是所使用的数据集的名称*
    - fun是要应用于数组元素的函数
  - 例：计算所有矩阵中数组行中元素的总和
  - `result <- apply(testarray, c(1), sum)`

Factors 因子

- 使用向量创建的 r 对象。 它将向量与向量中元素的不同值一起存储为标签
- **因子创建**
  - 函数factor()
    - 向量：`apple_colors <- c('green','green','yellow','red','red','red','green')`
    - 因子：`factor_apple <- factor(apple_colors)`
    - 将显示的结果为：

      ```
      [1] green  green  yellow  red   red   red   green
      Levels: green red yellow
      ```

  - 函数gl()
    - `gl(n, k, length = n*k, labels = 1:n, ordered = FALSE)`
    - n是整数，级别的个数
    - k是整数，每个级别的重复个数
    - length为结果的长度
    - labels为可选参数，向量因子水平的标签
    - `gl(3, 2, labels = c("Tampa", "Seattle","Boston"))`

      ```
      Tampa Tampa Seattle Seattle Boston Boston
      Levels: Tampa Seattle Boston
      ```

    - `gl(2, 2, 20)`

      ```
      [1] 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2
      Levels: 1 2
      ```

  - 判定：`is.factor(factor_data)`

- Factors 与Data Frames：在创建具有文本数据列的任何数据框时，R语言将文本列视为分类数据并在其上创建因子
- **指定级别顺序**
  - 使用新的等级次序再次应用因子函数，改变因子中的等级顺序
  - `new_order_data <- factor(factor_data,levels = c("East","West","North"))`

Data Frames 数据帧

- 表格对象，每列可以包含不同类型数据，可以是数字，因子或字符类型
- 列名称应为非空，行名称应该唯一，每个列应包含相同数量的数据项。
- **数据帧创建**
  - 使用data.frame()创建

```
emp.data <- data.frame(
    emp_id = c (1:5),
    emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
    salary = c(623.3,515.2,611.0,729.0,843.25),
    start_date = as.Date(c("2012-01-01","2013-09-23","2014-11-15","2014-05-
11","2015-03-27")),
    stringsAsFactors = FALSE
)
```

    - 将显示的结果为

```
emp_id    emp_name     salary     start_date
1     1     Rick         623.30     2012-01-01
2     2     Dan          515.20     2013-09-23
3     3     Michelle     611.00     2014-11-15
4     4     Ryan         729.00     2014-05-11
5     5     Gary         843.25     2015-03-27
```

- **获取信息**
  - str()函数，可以看到行列数、列名、每列数据类型、具体数据的前若干个字符
  - summary()函数，可以获得数据的统计摘要和性质，长度、类型、最大最小均值等
- **数据提取**
  - 按列名提取整列
    - `emp.data$emp_name`
    - `emp.data[,"emp_name"]`
    - `emp.data[,c("emp_name","salary")]`
  - 按数字索引提取
    - `emp.data[1]` 单索引默认按列提取
    - `emp.data[c(3,5),c(2,4)]`
    - `emp.data[1:2,]`
  - subset

        - `subset(data,colname1 == c(name1,name2) & colname2 > 300)`
    - 提取数据帧信息
        - 获取列名向量 `colnames(x)`
        - 获取行名向量 `rownames(x)`
        - 获取行数 `nrow(x)`
        - 获取列数 `ncol(x)`
- **增删列**
    - 直接用新列名修改
    - 增：`emp.data$dept <- c("IT","Operations","IT","HR","Finance")`
    - 删：`emp.data$dept <- NULL`
- **增删行**
    - 新建列数相同的新数据帧，然后使用rbind()函数合并
    - 增：`emp.finaldata <- rbind(emp.data,emp.newdata)`
    - 删除行需要向量技巧
    - `emp.data <- emp.data[-c(1,3),]`，其中第一项为要删除的行号向量（当前第几行）

数据帧处理

数据帧合并 cbind/rbind/merge

- cbind
    - 根据列进行合并，即叠加所有列，m列的矩阵与n列的矩阵cbind()最后变成m+n列
    - 合并前提：cbind(a, b)中矩阵a、b的行数相符
- rbind
    - 根据行进行合并，即叠加所有行，m行的矩阵与n行的矩阵rbind()最后变成m+n行
    - 合并前提：rbind(a, b)中矩阵a、b的列数相符
- merge
    - 根据列合并，m列的矩阵与n列的矩阵合并至多m+n列，按照列名相同的列进行合并
    - `merge(x, y, by = intersect(names(x), names(y)), by.x = by, by.y = by, all = FALSE, all.x = all, all.y = all, sort = TRUE, suffixes = c(".x",".y"), no.dups = TRUE, incomparables = NULL, ...)`
    - x,y为两个数据帧，by为作为合并根据的共同列名，all = TRUE 表示对应数据帧包含而另一数据帧不包含的行将被保留并填入NULL
    - incomparables表示用于比较的列上删去的值，常用incomparables = NA
    - 若两个数据帧没有共同列，合并后将返回`nrow(x)*nrow(y)`行，y的每一行信息将在每个x的行上重复

数据帧结构重塑 stack/unstack

- stack
    - 将多维数据压缩为一维向量
    - `stack(x, select, drop=FALSE, ...)`
    - 生成的仍为dataframe，第一列为压缩好的一维向量，第二列为对应列的标签
    - select为要选择或者删去的列名向量
- unstack
    - `unstack(x, form, ...)`

- form为公式类型，左侧是值，右侧是因子类型，每个水平形成一列
- 例：`unstack(x, values~ind)`
-

data.table

创建

```
library(data.table)
dft <- data.table(data1,data2,data3)
```

由data.frame转化：`dt1 <- as.data.table(df1)`

互相转化：table->frame `setDF(dtt)` frame->table `setDT(dtt)`

copy()：强行深复制 分配新存储地址（data.table是浅复制只改变指针）

访问

- 返回数据向量：`dft[,colname]`，`dft[[1,2]]`
- 返回表：`dft[,"colname"]`，`dft[1,2]`（均为dataframe的方法，在dataframe中返回numeric）
- 提取行：
  - `dft[1]`或`dft[1,]`（返回表）
  - 把某列设为key `setkey(dft,colname)`，然后可以行名提取 `dft["Bob",2]`
  - 可以设置多个key便于on参数筛选
- 提取列：
  - `dft[,1]`或`dft[,"colname"]`或`dft[,list(col1,col2)]`或`dft[,.(col1,col2)]`（返回表）
  - `dft[,colname]`或`dft[[3]]`或`dft$colname`（返回向量）
  - 双括号只能加数字或字符串
  - table中默认寻找的是变量名 故字符串变量（非列名）直接找会报错，需要with = FALSE
  - 列名修改：`setnames(dft,newnames)`或`colnames(dft)<-newnames`
- 逻辑提取
  - `dft[weight>40&height<170]`
  - `dft[c(T,F,T,T)]`（列不可以）
  - `dft[.("Bob",60),on=.(name1,weight)]`多列查找 等价于 name1 == Bob & weight == 60
- 增删行列
  - 删除：!或-
  - 排序
    - 根据某列对行排序 `dft[order(colname)]`
    - 按列名将列排序并接受 `setcolorder(dft,rev(names(dft)))`
- 计算
  - 参数1：哪些行 参数2：算什么 参数3：分组计算
  - 计算多个 表输出 指定列名`dft[,.(wm=mean(weight),ws=sum(weight))]`
  - 三参数分组 `dft[,mean(weight),by=height>150]`

- 合并
  - 按列名融合 `dt1[dt2,on="name1"]`，不同列名匹配`dt1[dt2,on="name1==friend"]`

特殊细节

- **.N** 代表行的数量 分组时代表每一组行的数量
- **.SD** 代表整个数据框
- **.SDcols** 指定.SD中包含的列
- 多个[]串联连续运算 `dft[weight>50][height>100][order(height)]`
- **%between%和%inrange%**
  - `dft[weight>=50&weight<=60]`
  - `dft[weight %between%c(50,60)]`
  - `dft[weight %inrange%c(50,60)]`
- **%like%** 字符串中含有某个字符 `dft[name1%like%"a"]`
- **%>%** 左值发送给右表达式，并作为右表达式函数的第一个参数
  - `plotData[,v_r] %>% replace(is.na(.), 0)]`

table数据接口

```
dataw <- data.table(a=1:10,b=2:11)
fwrite(dataw,"dataw.csv")
fwrite(dataw,"dataw.txt")
fwrite(dataw,"dataw.dat")

fread("dataw.csv")
fread(file="dataw.csv")
```

- 默认第一行作为列名（header=T），col.names改变列名
- nrow控制读几行 -1全部，0列名。

---

dplyr

数据筛选

- filter
  - 按行筛选子数据集
  - `filter(data, colname1 == choice1 & colname2 == choice2)`
- select
  - 按列筛选子数据集
  - `select(data, colname1:colname3)`
  - `select(data, starts_with("Petal"))` 选取变量名前缀包含Petal的列
  - `select(data, ends_with("Width"))` 选取变量名后缀包含Width的列

- `select(data, -contains("etal"))` 选取变量名不包含etal的列

数据排序

- arrange
  - 各行按某几列依次排序
  - `arrange(data,colname1,colname2,colname3)` （排序重要性1>2>3）

数据变形

- mutate
  - 直接利用已有的数据生成新变量且加在dataframe最后
  - `mutate(data, new1 = ..., new2 = ...)`
- transform
  - 也能达到同样的效果
  - `transform(data, new1 = ..., new2 = ...)`
- transmute
  - 只保留新生成的行列
  - `transmute(data, new1 = ..., new2 = ...)`

summerise()?

---

ggplot2

qplot快速绘图

- 散点图 `qplot(x=mpg, y=wt, data=df, geom = "point")`
- 平滑曲线 `qplot(x=mpg, y=wt, data = df, geom = "smooth")`
- 类似支持
  - 箱线图 boxplot
  - 小提琴图 violin
  - 点图 dotplot
  - 直方图 histogram
  - 密度图 density

ggplot

- 本质是图层叠加，依靠'+'号实现，越靠后的位置图层越高
- 起始明确从ggplot()开始，一个语句一幅图

初始化

载入数据空间、选择数据以及选择默认aes（美学，需要分组调整就写在aes内部，否则写在外面）

```
p <- ggplot(data = , aes(x = , y = ))
```

图层绘制

geom*和stat*，绘图与统计变换

```
ggplot(data= NULL, aes(x = x, y = y)) +
  geom_point(color = "darkred",stat = "sum")
```

与

```
ggplot(data= NULL, aes(x = x, y = y)) +
  stat_sum(color = "darkred",geom = "point")
```

效果相同。

时间格式绘图

```
ggplot2::ggplot(data = plotData, ggplot2::aes(x = time, y = count, group = type, colour
= type)) +
  ggplot2::geom_line() +
  ggplot2::scale_x_datetime(date_labels = '%Y-%m-%d', date_breaks = '3 month') +
  ggplot2::xlab('Time') +
  ggplot2::ggtitle(label = 'Time ~ Count', subtitle = 'subtitle') +
  ggplot2::ylab('Count') +
  ggplot2::theme(legend.position = "right",
                 legend.direction = "vertical",
                 axis.text.x = ggplot2::element_text(angle = 45))
```

保存图像

```
ggplot2::ggsave("graph/coinNumber.png")
```

指标名称及作用

- alpha：透明度，$0 \leq alpha \leq 1$
- color：线或点的颜色，见对照表
- fill：曲线下方的填充颜色
- linetype：线型，如"dashed"(虚线)
- size：点的大小或线的粗细

变量

- **变量名**：字母，数字和点或下划线字符，以字母或不以数字后跟的点开头
- **变量赋值**：可以使用向左，向右和等于运算符来为变量分配值
- **变量的数据类型**：获取分配给它的R对象的数据类型，可以在程序中使用同一个变量时更改变量的数据类型
- **变量查找**
  - 函数ls()返回工作空间中当前可用的所有变量
  - 以点(.)开头的变量被隐藏，可以使用“all.names = TRUE”参数列出
  - 仅查找在名称中带有某个指定字符的对象，则通过设定选项pattern(或pat)来实现
  - `ls(pattern = "var")`
- **变量删除**
  - 函数rm()删除给定变量 `rm(var.1)`，删除多个`rm(list = c("array1","array2"))`
  - 组合使用ls()和rm()可以全部删除 `rm(list = ls())`

函数

```
function_name <- function(arg_1, arg_2, ...) {
  Function body
}
```

- 传参原理：按照顺序和名称
- 例子：含默认参数的函数

```
new.function <- function(a = 3, b = 6) {
  result <- a * b
  print(result)
}
```

运算符

算术运算符

运算符	效果
+ - * /	两向量对应位置四则运算
%%	两个向量对应位置求余
%/%	两个向量对应位置求商(商与余数)
^	将第二向量作为第一向量对应位置的指数

关系运算符

- 支持：<，>，<=，>=，==，!=
- 将第一向量的每个元素与第二向量的相应元素进行比较
- 比较的结果是布尔值向量

逻辑运算符

- 只适用于逻辑，数字或复数类型的向量
- 除了实部和虚部均为0的数字，其他数字或复数均被认为是逻辑值TRUE
- &：元素逻辑AND运算符，两向量对应位置均为TRUE
- |：元素逻辑OR运算符，两向量对应位置至少一个为TRUE
- &&：逻辑AND运算符，两个向量的第一个元素都为TRUE时给出TRUE
- ||：逻辑OR运算符，两个向量的第一个元素至少一个为TRUE时给出TRUE
- !：逻辑非运算符，对向量的每个元素给出相反的逻辑值

赋值运算符

- <- 或 = 或 <<-：左分配，符号右侧的值赋给符号左侧
- -> 或 ->>：右分配，符号左侧的值赋给符号右侧

其他运算符

- : 冒号运算符，按顺序创建一系列数字，如`2:8`
- %in% 用于标识元素是否属于向量
- %*% 将矩阵与其转置相乘（矩阵乘法）

条件和循环语句

if-else if-else 语句

```
if(boolean_expression 1) {
// Executes when the boolean expression 1 is true.
} else if( boolean_expression 2) {
// Executes when the boolean expression 2 is true.
} else {
// executes when none of the above condition is true.
}
```

switch 语句

```
switch(expression, case1, case2, case3....)
```

- 元素未命名：`switch(3,2*3,sd(1:5),runif(3))`
- 元素命名：`switch("fruit", drink="water", meat = "beef", fruit = "apple", vegetable="cabbage")`

- 如果 expression 的值不是字符串，那么它被强制为整数。
- 若expr的计算结果为整数，且值在1~case个数之间时，则函数返回相应位置的值
- 如果表达式求值为字符串，那么该字符串与元素的名称匹配
- 如果有多个匹配，则返回第一个匹配元素。

while 循环

```
while (test_expression) {
    statement
}
```

repeat 循环

```
repeat {
    commands
    if(condition) {
        break
    }
}
```

for 循环

可以传递字符向量，逻辑向量，列表或表达式

```
for (变量 in 条件) {
    循环体
}
```

控制语句

- break：循环立即终止，并且程序控制在循环之后的下一语句处恢复，可用于循环或switch语句
- next：跳过本次迭代，并开始循环的下一次迭代

数据接口

CSV

- 读取

  ```
  data <- read.csv("input.csv")
  ```

- 写入

  ```
  write.csv(data,"output.csv")
  ```

其中会多一列X（row.names），可以通过命令删除

```
write.csv(data,"output.csv", row.names = FALSE)
```

EXCEL

（需要安装包）

- 读取

```
data <- read.xlsx("input.xlsx", sheetIndex = 1)
```

统计函数

平均数 中位数 众数

- 平均数

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

x为待求向量，$0 \leq trim \leq 0.5$表示去掉最大最小值的比例，na.rm = TRUE意味着去除NA值。

- 中位数

```
median(x, na.rm = FALSE)
```

- 众数

```
result <- getmode(x)
```

众数可以是数字也可以是字符

回归

- 一元线性回归
  - 回归 `relation <- lm(y~x)`
  - 预测 `result <-  predict(relation,data)`
- 多元线性回归
  - 回归 `model = lm(y ~ x1+x2+x3..., data = input)`
  - 预测 用法一致
- 逻辑回归
  - 回归 `model = glm(formula = y ~ x1+x2+x3..., data = input, family = binomial)`
- 最小二乘
  - 回归 `model <- nls(yvalues ~ b1*xvalues^2+b2,start = list(b1 = 1,b2 = 3))`

- 第一项为公式，第二项为data(略)，第三项为初始参数值

分布

- 正态分布
  - dnorm() 概率分布 `y <- dnorm(x, mean = 2.5, sd = 0.5)`
  - pnorm() 累积分布 `y <- pnorm(x, mean = 2.5, sd = 2)`
  - rnorm() 正态随机数 `y <- rnorm(50)`
- 二项分布
  - dbinom(x, size, prob)
  - pbinom(x, size, prob)
  - rbinom(n, size, prob)

分析

- 协方差分析

```
result1 <- aov(mpg~hp*am,data = input)
result2 <- aov(mpg~hp+am,data = input)
anova(result1,result2)
```

  aov为方差分析 anova为协方差分析

- 时间序列分析
  - 函数原型 `timeseries.object.name <-  ts(data, start, end, frequency)`
  - 后三个参数可选，frequency为单位时间内采样频率，12为月频，4为季频，6为10min，24*6为一天中固定的10min频率采样

```
rainfall.timeseries <- ts(rainfall,start = c(2012,1),frequency = 12)
```

- 卡方检验
  - 确定两个分类变量之间是否存在显著相关性
  - `chisq.test(data)`
  - data为表类型的数据，结果显示p值小于0.05则相关

高级功能

- 决策树
  - 需要安装包 `install.packages("party")`
  - 函数原型 `ctree(formula, data)`
  - 图像 `plot(model.tree)`
- 随机森林
  - 需要安装包 `install.packages("randomForest")`
  - 函数原型 `randomForest(formula, data)`
- 生存分析

- 预测特定时间发生的时间
- 需要安装包 `install.packages("survival")`
- `survfit(Surv(time,event)~1)`