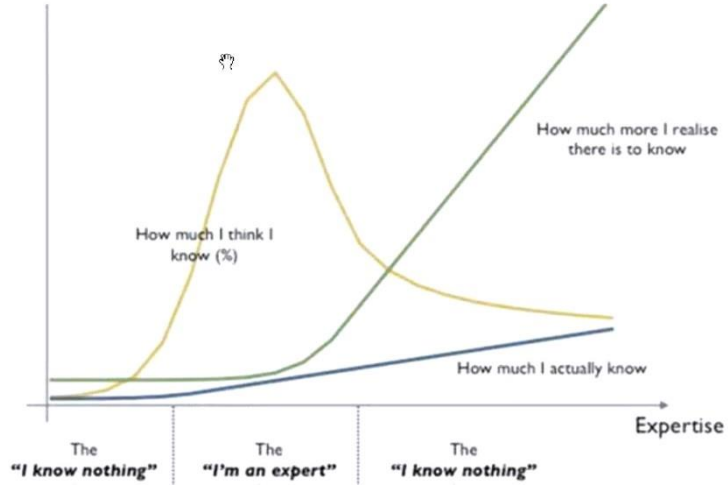
---
title: "R Base"
output: ioslides_presentation

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo =T, collapse = T)
```

数据科学

学习的三个阶段

Neural Networks: proceed with caution



学习的三个阶段
The "I know nothing" Stage
The "I'am an expert" Stage
The "I know nothing" Stage

学习的越深入，就会发现需要学习的东西越来越多．数据分析这个坑也很深

三个月时间只是对数据分析整体上有一个了解，每一块的内容都不可能进行很深入的学习，等三个月结束之后，想要向某一个方向发展，还要进行进一步深入的学习．

对于零基础入门一个新的领域或学习一个新的知识，先看视频，视频是最快的入门途径．再去看书．视频一般只偏重于教师认为重要的内容，而书则会把所有可能的知识点都列出来．

先看视频, 抓住重点, 有一定基础时再去看书, 补充视频的知识点.

尽量使用英文在谷歌来搜索问题, 国内还没有一个特别全的论坛, 使用百度搜索出来的大部分是一个个的帖子.

统计软件

偏编程　R　Python　SAS　STATA　Matlab
偏菜单　Spss　Evieus　SAS EG

偏菜单的入手快, 但要把操作背后的原理理解清楚.

R 偏向于统计
Python 偏向于编程, 国内可能会要求比较多, 因为有的公司可能要与前端等进行配合.
Evieus 和 Stata 金融方向应用比较多, 偏时间序列
Stata 是第四代的编程语言, 可能是所有编程语言中最容易掌握的. 不擅长数据清洗, 擅长回归分析和时间序列.
SAS EG 一些较新的模型可能不支持
Matlab 工科, 控制, 矩阵算法, 使用矩阵做神经网络比较多.

数学与统计
统计 > 矩阵(线性代数) > 数据分析, 只要开始接触到算法, 就要开始补数学和统计和基础. 机器学习都是偏重于算法的.

#几个常用的分布及之间的关系, 期望, 标准差, 二项分布, 期望是 np, 标准差是 npq, 正太分布期望是 μ, 标准差是 σ.

伯努利分布 > 二项分布 > 泊松分布 > 正态分布 > 中心极限 > 假设检验 > 传统回归 > 其它的模型或多或少都是基于回归的

##主要内容

- 初识 R
- 问答的正确姿势
- R 对象
- 读写数据
- 取子集
- 向量运算, 构造
- R 中的日期和时间
- 字符处理函数

#初识 R

R 背景

- R 是 S 语言的一种"方言"
- S 是一种编程语言，由 john chambers 和现已关闭的贝尔工作时共同研发
- 始于 1976 年，1988 年用 C 语言重写（版本 3.0），易于跨平台使用，并且加入了统计功能，1998 年发布版本 4.0，使用至今
- 经过各种公司收购后最终被 TIBCO 公司买到，其间衍生出了 S-PLUS 产品（加入了一些图形界面和好用的工具）
- 1993 年 R 问世，1995 年 R 变成免费软件，1997 年 R 核心开发小组成立（有很多当时负责开发 S-PLUS 的成员）
- 缺点: 数据都存储进内存, 基于 40 年前的技术开发, 不是万能的

R 优点

- 免费的~
- 可以分析任何类型的数据
- 更新速度以周计算
- 顶尖水准的制图功能
- 可以从各种类型的数据源导入数据
- 可以被整合到其他语言编写的应用程序
- 可以被应用到多个平台
- 有各式各样的 GUI(Graphical User Interface 图形用户界面)

R 是一种区分大小写的解释性语言，R 的主要功能是由程序内置函数、用户自编函数以及对对象的创建和操作所提供的，每个对象都有一个类属性

#在命令行中输入回车而不是执行命令时要使用 shift+enter
#在命令行窗口中向上向下方向键选择历史命令

R 区分大小写, 如果不想区分大小写, 必须要进行说明. 用函数来操作对象.

RStudio 打开 Rbase.Rmd, 如果有乱码, 使用 File > Reopen with encoding... > UTF-8

Tools > options > General > 更改工作目录 > 设置缩进 Code: Tab with: 4
R 中的缩进是为了好看, 即代码的可读性. 可重复性.

选中某个代码段 > run 或 ctrl+Enter 执行

Ctrl+shift+C 快速注释多行
Ctrl+Alt+I 插入代码块

R 语言也可以用来编程, 甚至编写游戏. 把 python 的练习全部用 R 实现一遍.

工作路径

工作路径是 R 寻找所有读写文件的默认目录, 建议使用固定目录

* 查询和设置工作路径的几种方式：
命令行:getwd()/setwd()
工具栏(Session->Set Working directory, Ctrl+Shift+H). 这里路径中使用正斜杠
    - "../" 上级目录; "./R" 此目录下文件夹

把 R 相关的 csv 文件放在工作目录下, 必须先要新建目录, 才能设置此目录为工作路径, 否则就会出错.

```{r}

#设置工作路径, 注意不能使用\, 因为\表示转义, 如\n 表示回车, \t 表示制表符, 如果必须使用\, 就使用两个\\, 第一个\表示转义, 第二个表示\
setwd("D:/King/Desktop/18 期/05 R/Data")
setwd("D:\\King\\Desktop\\18 期\\05 R\\Data")

getwd()

setwd("..") #返回上一级的目录
setwd('./Data') #设置工作路径为当前目录的下级目录
```

```{r, collapse=T}
getwd()
```

* 创建新的文本代码并且运行:source()
* 查看工作路径下所有文件:dir(), 显示当前加载的所有包 search()
* 查看当前环境中所有对象 ls(), 删除指定对象 rm()

```{r}
head(dir(), 2)
```

开始编程

```{r}
print("hello world")
```

在 R 里输入的东西叫表达式： <- 称为赋值运算符，把一个值赋给另一个符号, #是注释符, R 会忽略#右边的内容

```{r}
"hello world"
x <- 1
# 使用等号来赋值时只能把右边的值赋给左边的变量
x = 1
1 -> y

# print 是函数，把符号 x 的值传递给它
print(x)
x
# [1]表示元素在向量里的序号
```

问答的正确姿势

##问问题

- 线上提问，集思广益，老师也会不定期查看反馈
- [人大经济论坛五区](http://bbs.pinggu.org/forum.php?gid=148)

- 自己寻找解答是第一步，数据分析师应该具备的基本能力
- R 中每个包都有帮助文档（英文），也可以'Google it'或者'baidu it'。
- 经常逛逛论坛，解答别人问题的过程也是自己系统化巩固知识的过程，尤其是当你自己找到某个问题的答案时。

##函数帮助

```{r, eval=F}
?rnorm
help.search("rnorm")
```

```

```{r, collapse=T}
args("rnorm")
rnorm

?print #查看函数 print 的帮助文件
args(print) #查看函数的参数
```

##怎么问问题

- 通过什么样的步骤能够重新得到问题结果
- 期望的结果是什么
- 取而代之的结果是什么
- 你做过的所有尝试
- 你使用的 R 版本，系统
- [例子贴](http://bbs.pinggu.org/forum.php?mod=viewthread&tid=5557023&extra=)

标题的例子：
- R3.3.2 win10 64 位 lm 函数在处理较大的数据框是报错
- 对矩阵用主成分分析，什么是U, D, V^T

- 该做的：描述目的；详尽；精简；友善；试试跟踪；论坛大法
- 不该做的：轻易的认为发现了 bug；不要让别人替你干活；没有详尽解释问题的情况下寻求帮助；问论坛里常见的问题
[提问的智慧 how to ask questions the smart way](https://github.com/ryanhanwu/How-To-Ask-Questions-The-Smart-Way)

##怎么找答案

- 从帮助文件中查找
- 网上寻找
- 问一个精通的朋友
- 在论坛上发帖
- 编程问题去[stackoverflow](http://stackoverflow.com/)
- 统计分析问题去[crossvalidated](http://stats.stackexchange.com/)

- 提问圣地还是论坛，人大经济论坛或者 stackoverflow
- [stackoverflow](http://stackoverflow.com/)（在【r】标签下）；
- [R maling list](https://www.r-project.org/mail.html)用于软件问题（先阅读说明）；
- [crossvalidated](http://stats.stackexchange.com/)用于一般性问题
```

#R 对象

对象 object

在 R 里所有接触和操作的所有东西都称为对象，有五种基本的数据对象，即底层数据对象类型：

字符型 character
数值型 numeric
整形 integer
复数型 complex
逻辑型 logical(TRUE/FALSE)

- 最常见的基本对象是向量，包含相同类型的多个对象，不同对象组成的序列叫做列表.

和对象有关的东西：属性 attribute. 不是所有对象都有属性，但是属性是对象的一部分，常见属性有:

- 名字 names，维度名字 dimnames，维度 dimensions(matrices arrays)，类属性 class，长度属性 length，其他用户定义的属性，可以直接用 attributes()查看

数字

- R 中的数字通常被叫做数值型对象，基本上所有的数字都被当做双精度实数处理
- 只想要整数的话，需要在数字后面加 L 做后缀

```{r}
c(class(1), class(1L))
[1] "numeric" "integer"
```

- 特殊数字 Inf，表示无穷大, -Inf 表示无穷小

```{r}
c(1/0, 1/Inf)
[1] Inf   0
```

- 特殊的值 NaN，表示数字型未定义值，可以看成非数字，也可认为是缺失值, NA 表示一般性的未定义值

```{r}
0/0
[1] NaN
```

向量 c

冒号用来创建等差数列，冒号:创建的向量是整形的向量

```{r}
# 整形
x <- 1:20;
x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
class(x)
[1] "integer"
```

c()函数用来创建向量，把数据对象连结在一起(concatenate) , c 创建的向量是数值型的向量

```{r}
# TRUE 可以缩写为 T，数字 1 代表 TURE, FALSE-F-0
x <- c(T, F);
x
[1]  TRUE FALSE
class(x)
[1] "logical"
y <- c(1+0i, 2+4i)
y
[1] 1+0i 2+4i
class(y)
[1] "complex"
```

也可以通过 vector()创建初始向量，或者类型对应的函数如 integer()
vector(mode = "logical", length = 0)

vector 是一个函数, vector 中的对象都是它的参数，通过给函数的参数赋不同的值来实现不同的效果. 使用 R 就是通过调整函数的参数来使用函数操作对象

```{r}
```

```
# 会针对数据对象类型赋值默认值
x <- vector("numeric", length=10)
x
 [1] 0 0 0 0 0 0 0 0 0 0
class(x)
[1] "numeric"

# 创建一个长度为 10 的字符型空向量
y <- vector("character", 10)
y
 [1] "" "" "" "" "" "" "" "" "" ""
class(y)
[1] "character"

# 创建一个长度为 10 的整形空向量
integer(10)
 [1] 0 0 0 0 0 0 0 0 0 0
# 创建一个长度为 10 的字符型空向量
character(10)
 [1] "" "" "" "" "" "" "" "" "" ""
logical(10)
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
complex(10)
 [1] 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i
```

```{r}
# 创建向量
# 创建一个字符型的向量
x <- c("a", "b", "c")
x
[1] "a" "b" "c"
# 查看对象的类属性
class(x)
[1] "character"

# 创建一个数字型的向量
y <- c(1, 2, 3)
y
[1] 1 2 3
class(y)
[1] "numeric"
```

```
# 创建一个整形的向量，一般情况下很少区分数值型和整形
z <- c(1L, 2L, 3L)
z
[1] 1 2 3
class(z)
[1] "integer"

# 创建复数型的向量
m <- c(1+1i, 2+1i, 3+1i)
m
[1] 1+1i 2+1i 3+1i
class(m)
[1] "complex"

# 创建逻辑型的向量
n <- c(T, F, T, T)
n
[1]  TRUE FALSE  TRUE  TRUE
class(n)
[1] "logical"
```

强制类型转换

logical	numeric
T	1
F	0

任意非 0 数值转换为 logical 类型都为 T

如果创建的数据对象类型不一致，会创建一个最低级公共类型(least common denominator)的向量，即强制向低级转换. 例如向量中的类型必须要一致，否则程序就会强制进行类型的转换.
强制类型转换是程序自动进行的转换，还可以手动进行转换. 如果某一列数值型的数据如年龄中包含着一个字符型的数据，在读取数据时会自动把所有数值型的转换为字符型的. 所以这种情况下要特别注意.

```{r}
# 把 1.7 强制转换为字符型
x <- c(1.7, "a")
x
[1] "1.7" "a"
class(x)
```

```
[1] "character"

x <- c(T, 2)
x
[1] 1 2
class(x)
[1] "numeric"

x <- c("a", T)
x
[1] "a"       "TRUE"
class(x)
[1] "character"
```

强制转换是在后台进行，是系统进行的默认的转换，以上例子里最低级是字符型，字符型 < 数字型 < 逻辑型

函数转换类型

手动进行数据类型的转换.

以 as.开头的函数转换

```
```{r}
x <- 0:6;x;class(x)
[1] 0 1 2 3 4 5 6
[1] "integer"
y <- as.numeric(x);y;class(y)
[1] 0 1 2 3 4 5 6
[1] "numeric"
z <- as.logical(x);z;class(z)
[1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[1] "logical"
m <- as.character(x);m;class(m)
[1] "0" "1" "2" "3" "4" "5" "6"
[1] "character"
n <- as.complex(x);n;class(n)
[1] 0+0i 1+0i 2+0i 3+0i 4+0i 5+0i 6+0i
[1] "complex"
```
```

转换一定要符合逻辑，不成功会返回 NA 值，比如字符转换数字

```{r}
x <- c("b", "a");as.numeric(x) #会返回 NA，表示缺失值.
[1] NA NA
Warning message:
NAs introduced by coercion
```

缺失值和无穷大无穷小

缺失值有两种:
NA，表示一般意思上的缺失值
NaN，只表示数值运算产生的缺失值

```
1/0
[1] Inf
```
返回 Inf，为无穷大，可以直接参与运算

```
-1/0
[1] -Inf
```
返回-Inf，为无穷小

```
class(Inf)
[1] "numeric"
```

```
0/0
[1] NaN
```
返回 NaN，数值型运算的缺失值

Practices

1. 创建数值向量: 1, 99, Inf, -3.58, -Inf，并且将向量赋值给对象 a

```
a <- c(1, 99, Inf, -3.58, -Inf);a;class(a)
[1]    1.00 99.00      Inf -3.58    -Inf
[1] "numeric"
```

2. 创建整数向量: -1, -9, 1, 25，将向量赋值给 b，并且用 class()函数核实类型

```
b <- c(-1L, -9L, 1L, 25L);b;class(b)
```

```
[1] -1 -9  1 25
[1] "integer"
b <- as.integer(c(-1, -9, 1, 25));b;class(b)
[1] -1 -9  1 25
[1] "integer"
```

3. 创建差值为 1 的等差数列，从-100 到 0，赋值给 c，并且查看 c 的类型

```
c <- (-100:0);class(c)
[1] "integer"
```

4. 用两种方法创建指定长度为 15 的复数向量

```
vector("complex", 15)
 [1] 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i
complex(15)
 [1] 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i
```

5. 将上题中的复数向量强制转换为字符型向量，赋值给 d，并用 class()查看

```
d <- as.character(complex(15));d;class(d)
 [1] "0+0i" "0+0i" "0+0i" "0+0i" "0+0i" "0+0i" "0+0i" "0+0i" "0+0i"
[10] "0+0i" "0+0i" "0+0i" "0+0i" "0+0i" "0+0i"
[1] "character"
```

列表 list

- 和向量相似，不同的是每个元素可以是不同类型的对象，通过[[]]索引
向量中每个元素的类型必须相同，列表中的元素的类型可以不一致.
- 也可以给每个对象命名:*mylist <- list(name1=object1, name2=object2, ...)*

```
```{r}
a <- 'a'
a <- c("a", "b")
# R 中单一的数值 1 可以理解为长度为 1 的向量.R 中几乎所有的运算都是向量化的运算.
x <- list(1, a, T, b=c(1+4i, 2-3i));x
[[1]]
[1] 1

[[2]]
[1] "a" "b"

[[3]]
[1] TRUE
```
```

```
$b
[1] 1+4i 2-3i

# 如果没有对列表中的元素进行命名，系统就会以[[1]], [[2]]这样进行索引，命名的元素则以元素名为索引.
```

矩阵 matrix

可以看成是具有维度属性的特殊向量，维度为2的向量，矩阵中每个元素的类型都是一致的.
这里的矩阵就是线性代数中的矩阵. matlab 矩阵运算. 行列式
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
如果不设置某个参数，则使用默认值.
行数 nrow，列数 ncol
dim(m) 查看矩阵的维度

```{r}
#查看 matrix 函数的参数
args(matrix)
function (data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
NULL
```

有默认值的参数可以不用给定，会自动使用默认值，没有默认值的参数要给定，否则就会出错.

创建矩阵

方法 1：使用 matrix 创建矩阵

```{r}
#创建一个 2 行 3 列的矩阵，按列排列，即先排第一列, 1, 2，再排第 2 列, 3, 4，再排第 3 列, 5, 6.
matrix(1:6, 2, 3)
     [, 1] [, 2] [, 3]
[1, ]    1    3    5
[2, ]    2    4    6

#如果想要按行排列，则使用 byrom=True.
matrix(1:6, 2, 3, byrow=T)
     [, 1] [, 2] [, 3]
[1, ]    1    2    3
[2, ]    4    5    6

# dimnames 为维度的名字，即行和列的名字.
m <- matrix(nrow=2, ncol=3, dimnames=list(c("行 1", "行 2"), c("列 1", "列 2", "列 3")));m
```

```
     列 1  列 2  列 3
行 1   NA   NA   NA
行 2   NA   NA   NA

#查看维度
dim(m)
[1] 2 3

# 也可用: attributes(m)查看维度信息
attributes(m)
$dim
[1] 2 3

$dimnames
$dimnames[[1]]
[1] "行 1" "行 2"

$dimnames[[2]]
[1] "列 1" "列 2" "列 3"

# 默认按列排列
m <- matrix(1:6, nrow=2, ncol=3, byrow=F);m
       [, 1] [, 2] [, 3]
[1, ]      1      3      5
[2, ]      2      4      6

n <- matrix(1:6, 2, 3, byrow = T, dimnames=list(c("x1", "x2"), c("y1", "y2", "y3")));n
    y1 y2 y3
x1   1   2   3
x2   4   5   6

```

方法 2: 通过修改向量的维度信息创建按列排列的矩阵

```
```{r}
x <- 1:6
x
[1] 1 2 3 4 5 6
dim(x)
NULL

m <- 1:6;
dim(m) <- c(2, 3)
```
```

```
m
     [, 1] [, 2] [, 3]
[1, ]    1    3    5
[2, ]    2    4    6
# m 是一个向量，没有维度，通过 dim()对向量的维度进行赋值，从而转换为矩阵，所以说矩阵可以看成是特殊的向量，是具有维度属性的向量. 但这种方法只能按列排列，不能按行排列. 把对象的某个属性提取出来，然后赋予新值，或者把对象的子集提取出来，赋予新值，原来的对象就会发生变化. 这种方法和思想会经常用到
```

方法 3: 通过 cbind()和 rbind()创建，即绑定(binding)行或列

cbind(..., deparse.level = 1)

```{r}
x <- 1:3;y <- 10:12;x;y
[1] 1 2 3
[1] 10 11 12
#按列合并, x 中的元素作为列, y 中的元素作为行
cbind(x, y)
     x  y
[1, ] 1 10
[2, ] 2 11
[3, ] 3 12
#按行合并, x 中的元素作为行, y 中的元素作为列
rbind(x, y)
  [, 1] [, 2] [, 3]
x    1    2    3
y   10   11   12
```

cbind()和 rbind()不但可以合并向量，还可以合并矩阵

```{r}
x <- matrix(1:6, 2, 3);y <- 10:12;x;y
     [, 1] [, 2] [, 3]
[1, ]    1    3    5
[2, ]    2    4    6
[1] 10 11 12
#  按行合并时列数必须要一致, x 为 2 行 3 列的矩阵, y 中有 3 个元素，把 y 中的 3 个元素加到矩阵 x 的第 3 行中.
rbind(x, y)
  [, 1] [, 2] [, 3]
     1    3    5
```

```
      2     4     6
y    10    11    12
```

数组 array

数组 array 和矩阵的区别在于 array 维度大于 2
横截面数据和面板数据. 横截面数据不关心时间或者是同一时间的数据, 在 Excel 中是一个表格, 在 R 中则是一个数组. 面板数据相当于横截面数据加上一个时间的维度.

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

创建数组

```{r}
args(array)
function (data = NA, dim = length(data), dimnames = NULL)
NULL

array1 <- array(1:12, c(2, 2, 3));array1
, , 1

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

, , 2

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

, , 3

     [, 1] [, 2]
[1, ]    9   11
[2, ]   10   12

dimname1 <- c("A1", "A2");dimname2 <- c("B1", "B2", "B3");dimname3 <- c("c1", "c2")
x <- array(1:12, dim=c(2, 3, 2), dimnames = list(dimname1, dimname2, dimname3));x
, , c1
```

```
   B1 B2 B3
A1  1  3  5
A2  2  4  6

, , c2

   B1 B2 B3
A1  7  9 11
A2  8 10 12

array2 <- array(1:12, c(2, 2, 3), dimnames=list(c("x1", "x2"), c("y1", "y2"), c("z1", "z2", "z3")));array2
, , z1

   y1 y2
x1  1  3
x2  2  4

, , z2

   y1 y2
x1  5  7
x2  6  8

, , z3

   y1 y2
x1  9 11
x2 10 12

dimname2
[1] "B1" "B2" "B3"
dim(dimname2)
NULL
```

向量，列表，矩阵，数组小结

向量和列表都是若干个元素组成的序列，向量中的元素类型相同，列表中的元素类型可以不同.
矩阵和数组是在向量的基础上多了维度的属性，矩阵的维度为 2，数组的维度为 2 个以上.

数据框 data.frame

数据框是 R 的一个重要数据类型，用来存储表格数据

- 可认为是特殊类型的列表，但一般的列表中每个元素的长度可以不同，数据框中每列(每类)的元素长度必须相同
- 不同的函数的参数类型是不同的，有的可以处理向量，有的可以处理矩阵，有的可以处理列表. 因为数据框是一种特殊类型的列表，列表中的每一个元素就是数据框中的每一列，所以只要是可以处理列表的函数都可以处理数据框.
- 不同列可以是不同的类型(矩阵是相同的)
- 特殊属性：行名 row.names()或 rownames()，注意没有类似的列名用法 col.names()，只有 colnames()
- 可以通过读取表格函数 read.table()或 read.csv()读取数据框
- 可以通过调用 data.matrix()将数据框转化为矩阵
- 数据框就是表格. R 中的数据都是存储在数据框中的. 第一列是数据框的第 1 个元素，第二列是数据框的第 2 个元素…

```
data.frame(..., row.names = NULL, check.rows = FALSE,
                check.names = TRUE, fix.empty.names = TRUE,
                stringsAsFactors = default.stringsAsFactors())
```

```r
x <- data.frame(foo=1:4, bar=c(T, T, F, F));x
  foo   bar
1   1  TRUE
2   2  TRUE
3   3 FALSE
4   4 FALSE

#nrow(x), ncol(x)分别返回 x 这个数据框的行数和列数
c(nrow(x), ncol(x))
[1] 4 2

# 提取出数据框的行名. 可以提取出某个属性，给它赋上新值.
rownames(x)
[1] "1" "2" "3" "4"
row.names(x)
[1] "1" "2" "3" "4"

# 提取列的名字 "foo" "bar"
colnames(x)
[1] "foo" "bar"
```

```
# 可以提取出某个属性，给它赋上新值.
rownames(x) <- c("y1", "y2", "y3", "y4");x
    foo   bar
y1    1  TRUE
y2    2  TRUE
y3    3 FALSE
y4    4 FALSE
colnames(x) <- c("FOO", "BAR");x
    FOO   BAR
y1    1  TRUE
y2    2  TRUE
y3    3 FALSE
y4    4 FALSE
```

Practices

1. 创建列表 L，包含两个元素：名字为 a 的数值向量 1, 3, 2；名字为 b 的逻辑向量 F, T, F

```
L <- list(a=c(1, 3, 2), b=c(F, T, F));L
$a
[1] 1 3 2

$b
[1] FALSE  TRUE FALSE
```

2. 创建矩阵 M，矩阵将 1 到 8 八个数字按行排列, 2 行四列，用至少两种方法创建此矩阵

```
M <- matrix(1:8, 2, 4, byrow=T);M
     [, 1] [, 2] [, 3] [, 4]
[1, ]    1    2    3    4
[2, ]    5    6    7    8
N <- rbind(1:4, 5:8);N
     [, 1] [, 2] [, 3] [, 4]
[1, ]    1    2    3    4
[2, ]    5    6    7    8
```

3. 创建三维数组 A，将 1 到 8 八个数字分配到 2 乘 2 乘 2 的三个维度上，每个维度按数字 1 到 6 依次命名

```
# 会自动把 1:2, 3:4, 5:6 转换为字符型向量赋值给行名
A <- array(1:8, dim=c(2, 2, 2), dimnames=list(1:2, 3:4, 5:6))
```

```
A
, , 5

   3 4
1 1 3
2 2 4

, , 6

   3 4
1 5 7
2 6 8
```

4. 创建数据框 D，包含两列: ID 列列名 ID，从 1 到 3；数学成绩列，列名是 Math，成绩是 80, 78, 90. 并且查看数据框 D 的行数和列数

```
D <- data.frame(ID=1:3, Math=c(80, 78, 90));D
  ID Math
1  1   80
2  2   78
3  3   90
nrow(D)
[1] 3
ncol(D)
[1] 2
```

5. 将上述生成的四个对象放在列表 Mix 中，每个对象名为 List, Matrix, Array, Dataframe.

```
Mix <- list(list=L, Matrix=M, Array=A, Dataframe=D)
Mix
$list
$list$a
[1] 1 3 2

$list$b
[1] FALSE  TRUE FALSE


$Matrix
     [, 1] [, 2] [, 3] [, 4]
[1, ]    1    2    3    4
[2, ]    5    6    7    8

$Array
, , 5
```

```
  3 4
1 1 3
2 2 4

, , 6

  3 4
1 5 7
2 6 8


$Dataframe
  ID Math
1  1   80
2  2   78
3  3   90
```

因子

- 用于创建分类型数据，两种类型：有序/无序．使用有序的情况很少，一般用无序即可．只要是因子，都可以存储为因子.
    - 一般由函数 factor()创建，有序变量需要指定参数 ordered=TRUE，等级从小到大
- 输入是字符型向量，有一个单独的属性 levels，本质上是整数变量带有了 levels 的属性，每个整数有一个标签，可以用 unclass(x)查看
    - 以一个整数向量的形式存储类别值，同时由一个字符串组成的内部向量映射到整数上
- 重要的原因在于建模函数会对因子进行特殊处理，
- 使用字符型标签作为因子比数字更有解释性

创建因子

```
factor(x = character(), levels, labels = levels,
       exclude = NA, ordered = is.ordered(x), nmax = NA)
```

```{r}
x <- factor(c("yes", "yes", "no", "yes"), ordered=T);x
[1] yes yes no  yes
Levels: no < yes
# 说明:
[1] yes yes no  yes  # 默认是按照字母顺序表排序的
```

Levels: no < yes # Levels 返回因子的类别，因子就是在向量的基础上多了一个等级 Levels 的信息，用来表示分类. 这里说明创建的因子有两个类别，分别为 yes 和 no. ordered 表示有序的因子，在 Leavels 中显示的因子会显示大于小于号.

```
# unclass()去掉变量本身的类属性，查看变量本身的存储机制.
unclass(x)
[1] 2 2 1 2
attr(, "levels")
[1] "no"    "yes"

#  说明:
[1] 1 1 2 1   #表示因子实际上是以整数型来存储的, 1 1 2 1,
attr(, "levels")
[1] "yes" "no"  表示每个整数都有一个 Levels 的标签或属性, 1 的标签为"yes", 2 的标签为"no"，即说明存储的 1 表示是 yes, 2 表示的是 no
```

```
#  因为因子是按整数型来存储的，所以可以手动强制转换为数值型的向量 1 1 2 1
as.numeric(x)
[1] 2 2 1 2
```

```
#看每个等级的频数
table(x)
x
 no yes
  1   3
```

```
#自定义 Levels 因子/等级顺序，默认以字母表顺序排序. 自定义因子的顺序在做图和回归分析时都会用到.
factor(c("男", "女", "男", "女"))
[1] 男 女 男 女
Levels: 男 女

factor(c("男", "女", "男", "女"), levels=c("女", "男"))
[1] 男 女 男 女
Levels: 女 男
```

```
#  有序与无序因子
factor(c("男", "女", "男", "女"), levels=c("女", "男"), ordered=T)
[1] 男 女 男 女
Levels: 女 < 男
```

```
# levels 定义的先后顺序不同，因子存储时的值就不同
x <- factor(c("男", "女", "男", "女"), levels=c("女", "男"), ordered=T)
```

```
unclass(x)
[1] 2 1 2 1    # levels 中给定的第 1 个元素代表 1，第 2 个元素代表 2，以此类推，这里 1 表示女, 2 表示男，所以以 2, 1, 2, 1 来进行存储
attr(, "levels")
[1] "女" "男"

y <- factor(c("男", "女", "男", "女"), levels=c("男", "女"), ordered=T)
unclass(y)
[1] 1 2 1 2   # levels 中给定的第 1 个元素代表 1，第 2 个元素代表 2，以此类推，这里 1 表示男, 2 表示妇，所以以 1, 2, 1, 2 来进行存储
attr(, "levels")
[1] "男" "女"
```

gl 创建特殊因子 Generate Factor Levels

gl(n, k, length = n*k, labels = seq_len(n), ordered = FALSE)

```{r}
args(gl)
function (n, k, length = n * k, labels = seq_len(n), ordered = FALSE)
NULL

#n 表示有几个等级, k 表示每个等级重复几次, length 表示总长度

#表示有 2 个等级，每个等级重复 4 次
gl(2, 4)
[1] 1 1 1 1 2 2 2 2
Levels: 1 2

#如果 length 为总长度 n*k 的倍数，则把因子重复几倍．称为默认循环机制.
gl(2, 4, 16)
 [1] 1 1 1 1 2 2 2 2 1 1 1 1 2 2 2 2
Levels: 1 2

#不建议 length 不是 n*k 的倍数
gl(2, 4, 20)
 [1] 1 1 1 1 2 2 2 2 1 1 1 1 2 2 2 2 1 1 1 1
Levels: 1 2

# labels 表示给每个等级一个标签
gl(2, 8, labels = c("女", "男"), ordered=T)
 [1] 女 女 女 女 女 女 女 女 男 男 男 男 男 男 男 男
Levels: 女 < 男
```

```
gl(3, 2, 12, labels=c("小学", "初中", "高中"), order=T)
 [1] 小学 小学 初中 初中 高中 高中 小学 小学 初中 初中 高中 高中
Levels: 小学 < 初中 < 高中

gl(2, 1, 20)
 [1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
Levels: 1 2
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
```

R 中默认的强制转换机制与默认循环机制.

数据类型总结

底层类型:

character 字符型, numeric 数值型, integer 整形, complex 复数型, logical 逻辑型
不同的类型之间可以强制类型转换
基本类型:

vector 向量 元素类型相同

list 列表 元素类型可以不同

matrix 矩阵型 在向量的基础上多了一个二维的 dim 维度属性

array 数组 在向量的基础上多了一个三维及三维以上的 dim 维度属性

dataframe 数据框 可以看成是元素长度相同的列表, 类似于 excel 数据表和数据库表

factor 因子 在向量的基础上多了一个 Levels 的属性

很多包中都可能有属于这个包的特殊的数据类型

判断 NA 值

- 缺失值可以用 NaN(未定义的数学运算)或 NA(其他缺失值)表示
- NA 值也有类的概念, 缺失值可以是整形的, 也可以是字符型的等
- 判断 NA 值, 使用函数 is.na(), is.nan()
- NaN 属于 NA, 但是 NA 不属于 NaN, NaN 仅用于数学运算

```{r}
# numeric 类型
class(NaN)
[1] "numeric"

class(NA)
[1] "logical"
# logical 类型，由于逻辑型可以转换为任意的数据类型，使用 NA 来表示其它缺失值时，内部其实进行了一次强制类型转换

x <- c(1, 2, NA, NaN, 3)
x
[1]   1   2  NA NaN   3
is.numeric(x)
[1] TRUE
# NA 和 NaN 强制进行了一次数据的转换

#  判断是否是缺失值
is.na(x)
[1] FALSE FALSE  TRUE  TRUE FALSE

#判断是否是数值型缺失值, NA 包含 NaN, NaN 不包含 NA.
is.nan(x)
[1] FALSE FALSE FALSE  TRUE FALSE

# is.  是判断, as.  是手动强制转换
如 is.matrix, is.frame 等都可以使用. as.matrix, as.frame 只要能够转换为对应的数据类型，就都能够转换.
is.numeric(c(1, 2, 5))
[1] TRUE
```

命名

names()给向量，列表，数据框的列命名，可以使代码更具可读性

```{r}
x <- 1:3;names(x)
NULL
#此时为 NULL

#对向量的列进行命名
names(x) <- c("a", "b", "c");x
```

```
a b c
1 2 3

#等价于:
x <- c(a=1, b=2, c=3);x
a b c
1 2 3

y <- list(1:3, c(T, F));y
[[1]]
[1] 1 2 3

[[2]]
[1]  TRUE FALSE

names(y) <- c("a", "b");y
$a
[1] 1 2 3

$b
[1]  TRUE FALSE
```

dimnames() 给矩阵或数组命名

```{r}
z <- matrix(1:4, 2, 2);z
     [, 1] [, 2]
[1, ]    1    3
[2, ]    2    4

dimnames(z) <- list(c("a", "b"), 1:2);z
  1 2
a 1 3
b 2 4

m <- matrix(1:4, 2, 2);m
     [, 1] [, 2]
[1, ]    1    3
[2, ]    2    4
dimnames(m) <- list(c("a", "b"), c("c", "d"));m
  c d
a 1 3
b 2 4
```

```
# t(m)  对矩阵 m 进行转置
t(m)
   a b
c 1 2
d 3 4
```

把向量强制转换为数据框.

```
x  <-  c(1:10)
as.data.frame(x)
      x
1     1
2     2
3     3
4     4
5     5
6     6
7     7
8     8
9     9
10  10

x  <-  c("a",  "b",  "c")
as.data.frame(x)
   x
1  a
2  b
3  c
```

Practices

1. 创建因子向量("小学", "初中", "高中", "小学", "高中"), 赋值给 a, 有序, 顺序为 小学<初中<高中

```
a <- factor(c("小学", "初中", "高中"), levels=c("小学", "初中", "高中"), ordered=T);a
[1]  小学  初中  高中
Levels:  小学  <  初中  <  高中
```

2. 使用 gl()函数创建因子: 3 个小学, 3 个初中, 3 个高中, 此为一个循环, 共 10 个循环, 即总共 90 个元素, 有序, 小学<初中<高中

gl(3, 3, 90, labels=c("小学", "初中", "高中"), ordered=T)

3. 说出 NA 及 NaN 的区别，对于向量 x <- c(NA, NaN, 1, 2, NaN, NA)，口述 is.na(x)及 is.nan(x)的结果，并且用代码验证

```
x <- c(NA, NaN, 1, 2, NaN, NA)
is.nan(x)
[1] FALSE  TRUE FALSE FALSE  TRUE FALSE
is.na(x)
[1]  TRUE  TRUE FALSE FALSE  TRUE  TRUE
```

4. 创建长度为 4 的数值型向量，命名；创建 2 乘 2 的空矩阵，命名

```
x <- numeric(4);x
[1] 0 0 0 0
names(x) <- c("a", "b", "c", "d");x
a b c d
0 0 0 0
```

#把 NA 循环 4 次，类似于 matrix(1, 2, 2)

```
y <- matrix(NA, 2, 2);y
     [, 1] [, 2]
[1, ]   NA   NA
[2, ]   NA   NA
dimnames(y) <- list(c("a", "b"), c("X", "Y"));y
   X  Y
a NA NA
b NA NA
```

5. 创建数据框 x <- data.frame(x=1:3, y=2:4)，将列名 x, y 重命名为 y, z；将行命名为 a, b, c. (提示，使用函数 colnames(), rownames())

```
x <- data.frame(x=1:3, y=2:4);x
  x y
1 1 2
2 2 3
3 3 4
rownames(x) <- c("a", "b", "c");x
  x y
a 1 2
b 2 3
c 3 4
colnames(x) <- c("y", "z");x
  y z
a 1 2
b 2 3
c 3 4
```

#读写数据

键盘输入

代码: 易读, 可重复性

有两种方式: 文本编辑器和代码输入

文本编辑器: 1. 创建空数据框或者提供已有数据框; 2.通过 fix()函数打开文本编辑器编辑.注意字母列会被转化为因子列
但一般很少使用文本编辑器, 不符合代码的可重复性

```{r, eval=F}
mydata <- data.frame(a=1:2, b=c("a", "b"))
# edit(mydata)与 fix(mydata)等价, fix(mydata)会弹出数据编辑器, 在其中修改即可
mydata <- edit(mydata)
fix(mydata)
```

代码输入: 直接将数据输入成字符串格式, 用 read.table 读取, 字符串由 text 指定

```{r}
mydata <- "
+ y1 y2
+ 1 a
+ 2 b"
mydata
[1] "\ny1 y2\n1 a\n2 b"
read.table(text=mydata, header=T)
  y1 y2
1  1  a
2  2  b
```

读写数据常用函数

在 R 里读取数据时会用到几个基本函数:

- 读取表格形式数据: read.table(), read.csv(), 读取以行列形式存储的文本数据, 返回数据框
- 逐行读取文本文件 readLines(), 可以读取任何格式的文本文件, 如爬虫爬取的网页文件
- 读取 R 代码文件 source()(对应的相反的函数 dump)
- 读取被逆句法分析成文本文件的 R 代码文件 dput(对应的相反的函数 dget)
- load()和 unserialize()函数能把二进制文件读取到 R

对应的数据存储函数有:

write.table(); write.csv(); writeLines(); dump(); dput(); save(); serialize()

读数据

read.table(), read.csv()读取表格数据, 返回一个数据框。

```
?read.csv
read.table(file, header = FALSE, sep = "", quote = "\"",
                   dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
                   row.names, col.names, as.is = !stringsAsFactors,
                   na.strings = "NA", colClasses = NA, nrows = -1,
                   skip = 0, check.names = TRUE, fill = !blank.lines.skip,
                   strip.white = FALSE, blank.lines.skip = TRUE,
                   comment.char = "#",
                   allowEscapes = FALSE, flush = FALSE,
                   stringsAsFactors = default.stringsAsFactors(),
                   fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)

read.csv(file, header = TRUE, sep = ", ", quote = "\"",
               dec = ".", fill = TRUE, comment.char = "", ...)

read.csv2(file, header = TRUE, sep = ";", quote = "\"",
                 dec = ", ", fill = TRUE, comment.char = "", ...)
```

* file: 文件路径/文件名/连接
* header: 指明是否有表头(read.table 默认为 F, read.csv 默认为 T, 注意在使用 read.table 读取有表头的文件时, 一定要手动设置 header=T, 才能正确读取)
* sep: 指明列是如何分割的(read.table 默认空格, read.csv 默认逗号)
* colClasses: 字符向量(长度是列长度), 指定每列的数据类型(指定会很大程度加快读取速度, 尤其是数据量大时, 可以先读取前面几行, 通过循环找到每列的类型, 再赋值给这个参数读取所有数据)
* nrows: 读入的行数
* comment.char 指定注释符, 此行注释符右边的字符会被忽略(默认为#, 指定为空""可以提

高效率), 数据中一般是没有注释符的. 对于 read.table(), comment.char 默认为"#", 在读取数据时会扫描所有的数据查找是否有注释符, 可以把注释符设置为"", 这样就不会去在整个数据中查找注释符了, 就会提高读取的效率.
* skip: 从开始跳过的行数, skip 和 nrows 通常配合使用
* stringsAsFactors: 字符作为因子处理, 默认为 T, 一般都要设为 F, 只有特殊的情况下, 如性别才会设置为 T

```{r}
x <- read.csv("sale.csv")  #要先把 sale.csv 放在工作目录下
x <- read.csv("sale.csv", sep=" ") # 如果把 sep 设置为" ", 则会把所有的数据读取成一列.

x <- read.csv("sale.csv", nrows=4) #读取多少行
x <- read.csv("sale.csv", header=T, skip=2)  #默认 header=T, 即默认把第一行作为表头, 如果设置跳过多少行的同时设置 header=T, 就会把下一行作为表头, 就会出错. 可以使用 col.names 来设置表头
x <- read.csv("sale.csv", header=T, skip=2, col.names=c("year", "market", "sale", "profit"))
x

#对于数据量特别大的情况, 不可能手动去指定每一列的列名, 可以先读取两行数据, 存储到变量中, 在读取数据时, 设置 coClasses 为这个变量的列名称即可. 如家庭用电量的分析中, 如果需要跳过多少行, 就必须要设置表头.

x1 <- read.csv("sale.csv", nrows=2, comment.char = "", stringsAsFactors = F)
x <- read.csv("sale.csv", skip=1, col.names=colnames(x1))

#在 Environment 中点击 x 的向右箭头. 就会出现对数据的描述, 第 2 列是字符型的, 在读的时候会默认把字符型的列转换为因子. 可以把它设置为 F, 就不会把字符列转换为因子. factor 是用来表示分类的变量, 而实际数据中如名字列, 地址列等都不是分类的变量, 所以不需要转换为因子. 一般都要设置为 F, 如果有需要, 可以手动进行转换.
x <- read.csv("sale.csv", stringsAsFactors=F)

#把 comment.char 设置为""可以提高数据读取的效率
x<=read.csv("sale.csv", comment.char="")
```

读取大型数据

对于大型表格数据而言, 如下几点可以提效而且防止 R 卡住

- 仔细阅读 read.table 的帮助文档, 会发现大量优化函数的信息
- 估算一下读取的数据集需要多少存储空间

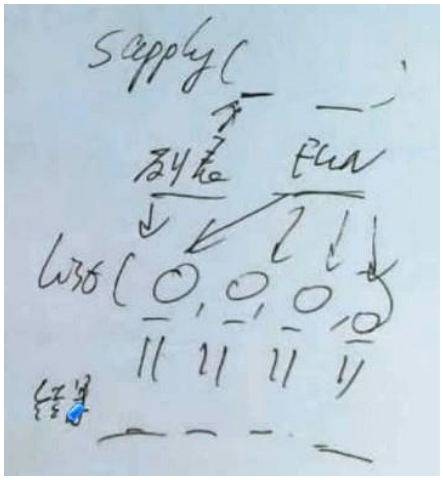
- 如果文件没有注释行的话，设置 comment.char=""
- coClasses 参数非常重要，如果不指定的话 R 会遍历每一列来确定每列的数据类型，数据集不太大时可以，但是如果很大会消耗内存且拖慢运行速度，如果能告诉 R 每一列的数据类型，通常速度能提升两倍. 如果每列都是数值型，可以直接设置 colClasses="numeric".

```{r}
x <- read.csv("sale.csv", colClasses = c("integer", "charactor", "integer", "integer"))

# 把所有列都读取为 character 类型的.
x <- read.csv("sale.csv", colClasses = "charactor")

#同样，可以使用这种方法来获取每一列的数据类型，使用 class(x)只能获取变量 x 整体的数据类型，不能得到每一列的数据类型，要使用一个循环语句来完成.

#使用 sapply()函数
#Apply a Function over a List or Vector
#sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
#X 为一个列表，FUN 表示一个方程，sapply(X, FUN)表示列表中的每一个元素都执行一次 FUN，返回每一个执行的结果. datafram 就是一个特殊的列表，sapply 可以处理列表，也就可以处理 datafram，如果 FUN 为 class，就表示对 datafram 中的每一列都执行 class，就返回每一列的数据类型.
```




```
x1 <- read.csv("sale.csv", nrows=2, comment.char = "", stringsAsFactors = F)
cc <- sapply(x1, class) #返回 x1 的每一列的数据类型，得到一个字符型的向量

x <- read.csv("sale.csv", colClasses = cc) #读取 2 行数据，得到数据集中每一列的数据类型，再赋值给 x 的 colClasses.

```

```{r, eval=F}
init <- read.table("table.txt", nrows=100)
cla <- sapply(init, class)
```

```
tabAll <- read.table("table.txt", colClasses=cla)
```

```{r}
y <- read.table("sale.csv")  #读取数据格式不正确，要设置 header=T，分隔符为逗号
y <- read.table("sale.csv", header=T, sep=", ")
```

##write.csv()

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"),
            fileEncoding = "")
```

```
write.csv(...)
write.csv2(...)
```

```{r}
x <- read.csv("sale.csv")

write.csv(x, "test.csv", row.names=F) #默认情况下会把行名 1, 2, 3, ...也写入到文件中，使用 row.names=F 就不会把行名写入到文件中.
```

- 设置 nrows，不会加速，但是提高内存使用率，如果能告诉 R 读取多少行信息的话, R 能计算出将占用多大内存，不用边读取边计算了．设置的比实际行数多一些也没关系，仍能准确读取.

内存计算

当调用读数据函数的时候，会把整个数据读取进电脑的 RAM 里，所以你需要知道这些数据需要多少内存，首先需要了解你的电脑:多少内存，哪些应用是吃内存大户，是否多个用户，操作系统.
粗略的计算: 1, 500, 000 行 120 列数据，认为都是数值，而由于数值型默认为双精度，双精度在 32 位 64 位系统中都占 8 位

```{r}
1500000*120*8/2^20
```

读取数据还要占用一些额外的内存，所以经验上讲要用 read.table()来读取数据的话，会占用

大约两倍计算结果的内存

Practices

给出的 hw1_data.csv 文件有 153 个观测, 6 个变量

1. 估计读取数据时大概占用的内存空间

2. 尽可能高效的使用 read.table()读取全部数据集到对象 a, 如指定所有列的数据类型, 注释符为空

a <- read.table("hw1_data.csv", header=T, sep=", ", colClasses="numeric", comment.char="")

点击 Environment 中的 a, 就会在新的工作区打开 a. 相当于执行了一条 view(a)的语句

3. 使用 read.csv 读取数据集到对象 b, 读取第 21 到第 100 行, 注意 b 中列名是否正确, 如不正确, 如何在读取的时候指定列名
b1 <- read.csv("hw1_data.csv", nrows=10)
cl <- colnames(b1) #抽取出列名
b <- read.csv("hw1_data.csv", skip=20, nrows=80, col.names=cl)

#子集

向量取子集

- [] 返回的对象和原对象类型相同(可能被简化), 从一个对象提取多个元素
- [[]] 从列表或者数据框中提取元素, 且仅可提取一个元素, 返回的对象类型可以不同, 一般用在列表和数据框中
- $ 可以从有命名的列表或数据框中提取元素, 和[[]]类似. 但只能通过名字来取子集

三种索引类型: 数字索引, 逻辑索引, 名字索引

先确定取子集的方式, 是使用单中括号, 双中括号还是$
再确定索引的类型, 是数字索引, 逻辑索引, 还是名字索引

数字索引

 # 数字索引可以理解为位置索引. 给定元素的位置.

```{r}
x <- c("a", "d", "c", "c", "d", "a")
# 返回长度为 1 的字符型向量
x[1]
[1] "a"
```

```
x[1:4]
[1] "a" "d" "c" "c"
x[c(1:4)]
[1] "a" "d" "c" "c"

x[1, 2, 4]
Error in x[1, 2, 4] : incorrect number of dimensions
x[c(1, 2, 4)]
[1] "a" "d" "c"

#间接索引，一般情况下使用的都是间接索引，通过一定的方法得到数值型的向量，再取这些索引的值.
index <- c(1, 3, 5)
x[index]
[1] "a" "c" "d"

# 索引中的负号. 只能用于数字索引，跳过某索引位置的几个数，取剩余其他索引位置上的数
x[-c(1, 3)]
[1] "d" "c" "d" "a"
```

逻辑索引

比较运算符

字母可以使用比较运算符进行比较 >, <, >=, <=, !=, ==，如

```
"z">"a"
[1] TRUE
```

条件运算符

#且，可以使用 1 个或 2 个 &

```
T & F
[1] FALSE
T && F
[1] FALSE
```

#或，可以使用 1 个或 2 个 |

```
T | F
[1] TRUE
T || F
[1] TRUE
```

#非

```
!T
[1] FALSE
```

```{r}
# 拿 x 中的每一个元素与 "a" 进行比较，得到一组逻辑型的向量，其中包含了内建的循环机制，因为比较的二者长度不一致，自动把"a"循环了 6 次.
```

```
x>"a"
[1] FALSE  TRUE  TRUE  TRUE  TRUE FALSE
```

```
# 先计算 x>"a"的值，得到长度为 6 的逻辑型向量，再从 x 中取子集，把为真的位置上的值就取出，为假的位置的值就舍弃
```

```
x[x>"a"]
[1] "d" "c" "c" "d"
```

```
##字母可以通过字典排序法来排序???
x>"a"
```

列表取子集

单中括号 [] 取子集

```
# 前后类型一致，取多个元素
```

```{r}
y <- list(a=1:3, b=c(T, F, T, T), c=matrix(4, 2, 2));y
$a
[1] 1 2 3

$b
[1]  TRUE FALSE  TRUE  TRUE

$c
     [, 1] [, 2]
[1, ]    4    4
[2, ]    4    4
```

[] 数字索引，通过位置进行索引

```
y[c(1, 3)]
$a
[1] 1 2 3
```

```
$c
     [, 1] [, 2]
[1, ]    4    4
[2, ]    4    4
```

 # [] 逻辑索引，通过真假进行索引.

逻辑型向量要与 x 中的元素等长. 无论是数字索引还是逻辑索引，都返回一个列表，即取的子集和原集合的类型是一致的.

```
y[c(T, F, T)]
$a
[1] 1 2 3

$c
     [, 1] [, 2]
[1, ]    4    4
[2, ]    4    4
```

 # [] 名字索引

```
y["b"]
$b
[1]  TRUE FALSE  TRUE  TRUE

 y[c("a", "c")]
$a
[1] 1 2 3

$c
     [, 1] [, 2]
[1, ]    4    4
[2, ]    4    4
```

 # 双中括号 [[]] 选取子集

一般用用列表和数据框中，数据框可以看成是特殊的列表. 只能取一个元素.
[[]] 很少用逻辑索引

 # [[]] 数字/位置索引

[[]]返回第 1 个列表元素的内容. 比[]选取的更深一层. 因为返回的是元素的内容，如果元素的内容不同，就无法把它们组织到一起，所以[[]]只能取一个元素的内容.

```
y[[1]]
[1]  1  2  3

# []只取第一级的元素，得到一个列表，不会再对列表元素的内容进行选取
y[1]
$a
[1]  1  2  3
```

 # [[]] 名字索引

```
y[["a"]]
[1]  1  2  3
y[["c"]]
      [,  1]  [,  2]
[1,  ]       4       4
[2,  ]       4       4
```

 # $ 名字索引

```
#  与[[]]方式类似, $  名字索引只能取一个元素的内容，但$只能使用名字取子集
$后面的名字不能是计算得到的，必须是在定义中直接存在的名字
```

```
```{r}
x <- list(a=1:4, b=0.6, c="hello");x
$a
[1] 1 2 3 4

$b
[1] 0.6

$c
[1] "hello"

name <- "a"
x[name]
$a
[1] 1 2 3 4

#  使用$方式取子集时，必须直接使用定义中存在的名字，不能使用间接的方式．当索引不到结果就返回 NULL

x$name
NULL
```
```

```
x$a
[1] 1 2 3 4
```

```
y$b
[1]  TRUE FALSE  TRUE  TRUE
```

取子集-特殊用法

多次取子集

[[]] 中可以使用数值型向量，如 y[[c(1, 3)]], 先取出 y 的第 1 个元素的内容，再在内容中取第 3 个元素，相当于 y[[1]][[3]]，也相当于 y[[1]][3].

##这三种写法等价. 注意: [[]]中使用向量取子集时向量中只能是数字索引

```
y[[c(1, 3)]]
[1] 3
y[[1]][[3]]
[1] 3
y[[1]][3]
[1] 3
```

```
x <- list(a=list(10, 12, 14), b=c(3.14, 2.81));x
$a
$a[[1]]
[1] 10

$a[[2]]
[1] 12

$a[[3]]
[1] 14

$b
[1] 3.14 2.81
```

```
x[[c(1, 3)]]
[1] 14
```

```
x[[1]][[3]]
[1] 14
```

```
x[[1]][3]
```

```
[1] 14
```

间接索引取子集

```
# 通过定义或计算出来的索引变量来取子集
name <- "c"

# 单中括号[]的间接索引，以名字的方式选取，得到列表 c
y[name]
$c
     [, 1] [, 2]
[1, ]    4    4
[2, ]    4    4

# 双中括号[[]]的间接索引，以名字的方式选取，得到列表 c 的内容
y[[name]]
     [, 1] [, 2]
[1, ]    4    4
[2, ]    4    4

# $的间接索引，返回 NULL
# 对于$，只能直接使用名字来选取，不能使用间接的方式选择.
y$name
NULL
```

名字索引简写匹配

必须是名称的前几位简写，只有当名字的简写能唯一匹配到内容时才能使用，且只能用在双方括号和$符号中

$ 使用名字的简写来取子集

```
y <- list(a=1:3, b=c(T, F, T, T), c=matrix(4, 2, 2));y
$a
[1] 1 2 3

$b
[1]  TRUE FALSE  TRUE  TRUE

$c
     [, 1] [, 2]
[1, ]    4    4
[2, ]    4    4
```

```
names(y)
[1] "a" "b" "c"
names(y) <- c("aa", "aawake", "bb");y
$aa
[1] 1 2 3

$aawake
[1]  TRUE FALSE  TRUE  TRUE

$bb
     [, 1] [, 2]
[1, ]    4    4
[2, ]    4    4

y$aawake
[1]  TRUE FALSE  TRUE  TRUE
# $还可以用名字的简写, 但是必须是名称的前几位, 并且能取到唯一的值, 如这里不能写成是 y$a, 因为不能取到唯一值, 名字简写方式可以用在以后很多地方, 如参数匹配.
y$aaw
[1]  TRUE FALSE  TRUE  TRUE

# 双中括号 [[]] 使用名字的简写来取子集

x <- list(aardvark=1:5);x
$aardvark
[1] 1 2 3 4 5

# 使用$取子集时可以直接使用名字的缩写来取子集, 只要能取到唯一的元素
x$a
[1] 1 2 3 4 5

# 使用双中括号取子集时默认情况下不会部分匹配, 即默认为精确匹配
x[["a"]]
NULL

# 需要调整参数, 取消默认的精确匹配.
x[["a", exact=F]]
[1] 1 2 3 4 5

y <- list(a=1:3, b=c(T, F, T, T), c=matrix(4, 2, 2))
names(y) <- c("aa", "aawake", "bb")

y[["aaw"]]
NULL
```

```
y[["aaw", exact=F]]
[1]  TRUE FALSE  TRUE  TRUE
```

矩阵取子集

```{r}
x <- matrix(1:18, 3, 6);x
     [, 1] [, 2] [, 3] [, 4] [, 5] [, 6]
[1, ]    1    4    7   10   13   16
[2, ]    2    5    8   11   14   17
[3, ]    3    6    9   12   15   18
```

对矩阵取子集时一般都是把某一列取出来. 所以可以使用[], 直接取某个元素没有意义.

```
# 通过 (i, j) 索引取子集, i 为行索引, j 为列索引, 取第 i 行第 j 列的元素
# 取第 1 行第 2 列的元素
x[1, 2]
[1] 4

# 取第 2 行, 第 1, 3 列的元素
x[2, c(1, 3)]
[1] 2 8

#取 1, 3 行和 2, 4 列中相交的元素, 返回一个矩阵
x[c(1, 3), c(2, 4)]
     [, 1] [, 2]
[1, ]    4   10
[2, ]    6   12

# 得到的子集如果只有一个或一串数字时, 程序就会自动把结果简化为向量的形式, 但是如果不想转换为向量, 想得到矩阵, 就使用 drop=F, 即禁止向下简化

# 取 1, 3 行和第 1 列的值, 返回一个向量
x[c(T, F, T), 1]
[1] 1 3

#使用 drop=F 得到两行一列的矩阵
x[c(T, F, T), 1, drop=F]
     [, 1]
[1, ]    1
[2, ]    3
```

#取 1, 3 行所有的列

```
x[c(1, 3), ]
```

```
     [, 1] [, 2] [, 3] [, 4] [, 5] [, 6]
[1, ]    1    4    7   10   13   16
[2, ]    3    6    9   12   15   18
```

#取 2, 4 列所有的行

```
x[, c(2, 4)]
```

```
     [, 1] [, 2]
[1, ]    4   10
[2, ]    5   11
[3, ]    6   12
```

取第 1 行的所有值

```
x[1, ]
```

```
[1]  1  4  7 10 13 16
```

取第 2 列的所有值

```
x[, 2]
```

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

取第 1 行 2 列的值, 并且以矩阵方式返回

```
x[1, 2, drop=F]
```

```
     [, 1]
[1, ]    4
```

```
x
```

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

查看 x, 默认的行名和列名就是取子集的表达式, 同样, 任意创建一个列表, 如果不定义列名和行名, 默认的列名和行名也是索引的表达式, 如 list(1:3, c("a"))

```
x <- list(1:3, c("a"));x
```

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

[[2]]
[1] "a"
```

```
x[[1]]
```

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

```
x[[2]]
[1]  "a"
```

数据框取子集

数据框的每列作为长度相同的列表，也可以用与列表类似的方法取子集.

```
```{r}

#在创建数据框时，如果不指定 stringsAsFactors，则自动会把字符型的列转换为因子 factor. 这与读取数据时相同，字符会自动转换为因子. 在创建数据框时字符也会自动转换为因子，如果不想转换，就要指定 stringsAsFactors=F

score <- data.frame(ID=1:6,
                    math=c(90, 89, 70, 60, 56, 10),
                    chn=c(60, 88, 99, 61, 59, 100),
                    sex=c("男", "女", "男", "女", "男", "男"),
                    stringsAsFactors=F)

class(score$sex)
[1] "character"

# 由于在这里希望 sex 列转换为因子，所以不用设置 stringsAsFactores.
score <- data.frame(ID=1:6,
                    math=c(90, 89, 70, 60, 56, 10),
                    chn=c(60, 88, 99, 61, 59, 100),
                    sex=c("男", "女", "男", "女", "男", "男")
                    )
score
  ID math chn sex
1  1    90  60   男
2  2    89  88   女
3  3    70  99   男
4  4    60  61   女
5  5    56  59   男
6  6    10 100   男

class(score$sex)
[1] "factor"

#把 sex 列的数据取出来
score$sex
[1] 男 女 男 女 男 男
```
```

```
Levels: 男 女

# 取出 sex 为男的所有行
# 第 1 步: 返回 TRUE, FALSE 的逻辑向量.
score$sex=="男"
[1]  TRUE FALSE  TRUE FALSE  TRUE  TRUE

# 第 2 步: 取出 sex 为男的所有行
score[score$sex=="男", ]
  ID math chn sex
1  1   90  60  男
3  3   70  99  男
5  5   56  59  男
6  6   10 100  男

# 两个逻辑型向量的 "且"
c(T, F)&c(F, F)
[1] FALSE FALSE

#取出数学和语文成绩都>=60 的人，还是逻辑型的索引，可以使用复杂的条件得到逻辑型的向量作为索引，甚至还可以使用函数，只要能够得到逻辑型的向量或数值型的向量，就可以使用对应的逻辑索引或数值索引.
score[score$math>=60 & score$chn>=60, ]
  ID math chn sex
1  1   90  60  男
2  2   89  88  女
3  3   70  99  男
4  4   60  61  女

```

通过取子集的方法生成新的元素，替换元素

当索引不存在的列或者元素的时候，会生成新的列或者元素

```{r}
x <- 1:3;x[4] <- 100;x
[1]   1   2   3 100

#会自动补全 x[5]，x[6]为 NA
x[7] <- 10;x
[1]   1   2   3 100  NA  NA  10
```

```
x <- data.frame(a=T);x
      a
1  TRUE

x[2] <- "lalala";x
      a      V2
1  TRUE  lalala

x["c"] <- 1+2i;x
      a      V2      c
1  TRUE  lalala  1+2i

x$d <- 5L;x
      a      V2      c  d
1  TRUE  lalala  1+2i  5

y <- c(1, 2, 4);y
[1] 1 2 4

#对于不存在的元素，返回 NA
y[4]
[1] NA

y[c(1, 3, 4, 5)]
[1]   1   4  NA  NA

#可以取出对象的属性，再通过更改对象的属性来达到更改对象的目的．也可以对取出来的子集进行修改，从而达到修改对象的目的
y <- c(1, 2, 4);y
[1] 1 2 4

y[2] <- -2;y
[1]   1  -2   4

#可以添加新值
y[4] <- 10;y
[1]   1   2   4  10

#自动把 5, 6 个元素设置为 NA
y[7] <- 5;y
[1]   1   2   4  10  NA  NA   5

# 向量和列表都可以使用类似的方法进行赋值
```

上面学习的所有取子集的方法都可以通过类似的方法对子集进行修改，注意设置的新值的类型和长度要与原来的一致.

对于数据框元素的替换

```
score <- data.frame(ID=1:6,
                    math=c(90, 89, 70, 60, 56, 10),
                    chn=c(60, 88, 99, 61, 59, 100),
                    sex=c("男", "女", "男", "女", "男", "男"),
                    stringsAsFactors=F)
score
```

	ID	math	chn	sex
1	1	90	60	男
2	2	89	88	女
3	3	70	99	男
4	4	60	61	女
5	5	56	59	男
6	6	10	100	男

```
#对已有数据进行更改，把 100 的值赋给第 6 的行中的 math.
score[score$ID==6, "math"] <- 100;score
```

	ID	math	chn	sex
1	1	90	60	男
2	2	89	88	女
3	3	70	99	男
4	4	60	61	女
5	5	56	59	男
6	6	100	100	男

```
#添加一个新列，并赋值
score$eng <- c(58, 89, 35, 64, 67, 86);score
```

	ID	math	chn	sex	eng
1	1	90	60	男	58
2	2	89	88	女	89
3	3	70	99	男	35
4	4	60	61	女	64
5	5	56	59	男	67
6	6	100	100	男	86

```
#abs 取绝对值
abs(-1)
[1] 1
abs(c(-1, -2, 1))
```

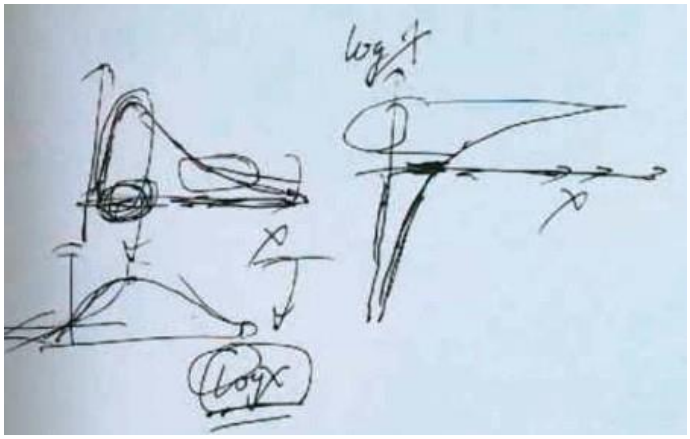
[1] 1 2 1

#查看语文和数学的差值，看是否偏科，规定凡是大于 20 分的，都确定为偏科

```
score$div <- abs(score$math-score$chn);score
```

	ID	math	chn	sex	eng	div
1	1	90	60	男	58	30
2	2	89	88	女	89	1
3	3	70	99	男	35	29
4	4	60	61	女	64	1
5	5	56	59	男	67	3
6	6	100	100	男	86	0

#假设英语成绩是右偏的成绩，对它取 log. logx 把右偏分布左边集中的数据分散开，把右边分散的数据集中起来，这样就纠正了右偏数据的偏度



```
score$lgeng <- log(score$eng);score
```

	ID	math	chn	sex	eng	div	lgeng
1	1	90	60	男	58	30	4.060443
2	2	89	88	女	89	1	4.488636
3	3	70	99	男	35	29	3.555348
4	4	60	61	女	64	1	4.158883
5	5	56	59	男	67	3	4.204693
6	6	100	100	男	86	0	4.454347

删除已存在的某列数据

```
score$lgeng <- NULL;score
```

	ID	math	chn	sex	eng	div
1	1	90	60	男	58	30
2	2	89	88	女	89	1
3	3	70	99	男	35	29
4	4	60	61	女	64	1
5	5	56	59	男	67	3

6 6 100 100 男 86 0

```

## Practices

小写的 letters 代表小写的 a-z, 大写的 letters 代表大写的 A-Z

1. 查看 letters 所代表的字母顺序向量(直接在命令行中输入 letters)

```
letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i"
[10] "j" "k" "l" "m" "n" "o" "p" "q" "r"
[19] "s" "t" "u" "v" "w" "x" "y" "z"
LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I"
[10] "J" "K" "L" "M" "N" "O" "P" "Q" "R"
[19] "S" "T" "U" "V" "W" "X" "Y" "Z"
```

- 取自己的姓对应的字母, 将姓取出来

```
matrix(letters)
letters[c(26, 8, 21)]
[1] "z" "h" "u"
```

- 取除了自己的姓以外所有的字母

```
letters[-c(26, 8, 21)]
 [1] "a" "b" "c" "d" "e" "f" "g" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t
" "v"
[21] "w" "x" "y"
```

- 取所有偶数位置的字母(提示:2 乘以 1:2 可得到前两个偶数)

```
letters[(1:13)*2]
 [1] "b" "d" "f" "h" "j" "l" "n" "p" "r" "t" "v" "x" "z"
```

2. 对于数据框(创建数字框的代码自己理解和执行):

```
x <- as.data.frame(matrix(1:25, 5, 5, dimnames=list(letters[1:5], letters[22:26])));x
  v  w  x  y  z
a 1  6 11 16 21
b 2  7 12 17 22
c 3  8 13 18 23
d 4  9 14 19 24
e 5 10 15 20 25
```
```

- 写出得到子集 7, 8, 12, 13, 17, 18 的代码; 2, 4, 12, 14, 22, 24 的代码(每个结果需要按照名字索引, 按照位置索引各一个代码, 总共 4 个)

```
x[c("b", "c"), c("w", "x", "y")]
```

```
  w  x  y
b 7 12 17
c 8 13 18
```

```
x[2:3, 2:4]
```

```
  w  x  y
b 7 12 17
c 8 13 18
```

```
x[c("b", "d"), c("v", "x", "z")]
```

```
  v  x  z
b 2 12 22
d 4 14 24
```

```
x[c(2, 4), c(1, 3, 5)]
```

```
  v  x  z
b 2 12 22
d 4 14 24
```

- 通过逻辑索引得到 3 到 5 行的子集(可以引用任何一列作为逻辑条件)

```
x[x$v>=3, ]
```

```
  v  w  x  y  z
c 3  8 13 18 23
d 4  9 14 19 24
e 5 10 15 20 25
```

- 通过逻辑索引得到 3 到 5 行且是 3 到 5 列的子集, 逻辑索引的条件需是行名和列名

```
x[rownames(x)>"b", colnames(x)>"w"]
```

```
  x  y  z
c 13 18 23
d 14 19 24
e 15 20 25
```

- 加一个新列, 新列的所有值都是逻辑值 TRUE, 新列名为"o"

```
x$o <- T;x
```

```
  v  w  x  y  z    o
a 1  6 11 16 21 TRUE
b 2  7 12 17 22 TRUE
c 3  8 13 18 23 TRUE
d 4  9 14 19 24 TRUE
```

e 5 10 15 20 25 TRUE

缺失值的过滤和替换

- is.na(); complete.cases()得到逻辑向量，通过取子集剔除缺失值
- na.omit()直接剔除(对象一般是数据框)

向量缺失值的过滤

```{r}
x <- c(1, 2, NA, 4, NA, 5)
bad <- is.na(x);bad
[1] FALSE FALSE TRUE FALSE TRUE FALSE
# 取出 x 中的缺失值
x[bad]
[1] NA NA

# 取出 x 中的非缺失值
x[!bad]
[1] 1 2 4 5

# 使用 complete.cases 对多个向量或数据框的多个列进行缺失值的判断
x <- c(1, 2, NA, 4, NA, 5);y <- c("a", "b", NA, "d", "e", NA)
# 只要 x, y 对应位置上有一个缺失值，就返回 FALSE
good <- complete.cases(x, y);good
[1] TRUE TRUE FALSE TRUE FALSE FALSE
x[good]
[1] 1 2 4
y[good]
[1] "a" "b" "d"
```

数据框缺失值的替换

```{r}
score <- data.frame(ID=1:6,
                    math=c(90, 89, 70, 60, 56, 10),
                    chn=c(60, 88, 99, 61, 59, 100),
                    sex=c("男", "女", "男", "女", "男", "男"),
                    stringsAsFactors=F)

# 把数据框中的某些值设置为 NA，用来表示原始数据中的缺失值
score[2, 3] <- NA;score[3, 2] <- NA;score[5, 2] <- NA;score
  ID math chn sex
```

```
1   1    90   60   男
2   2    89   NA    女
3   3    NA   99   男
4   4    60   61   女
5   5    NA   59   男
6   6    10  100   男

# is.na()判断向量中是否有缺失值，所以只能对某一列进行判断，不能对整个数据框使用
is.na(score$math)
[1]  FALSE  FALSE   TRUE  FALSE   TRUE  FALSE

score[is.na(score$math),  ]
   ID  math  chn  sex
3   3    NA   99   男
5   5    NA   59   男

#  取出来缺失值
score[is.na(score$math),  "math"]
[1]  NA  NA

#把所有的缺失值替换为 1
score[is.na(score$math),  "math"]  <-  1;score
   ID  math  chn  sex
1   1    90   60   男
2   2    89   NA    女
3   3     1   99   男
4   4    60   61   女
5   5     1   59   男
6   6    10  100   男
```

 # 数据集缺失值的处理

 # R 数据集

?data

Data Sets
Description
Loads specified data sets, or list the available data sets.
Usage
data(..., list = character(), package = NULL, lib.loc = NULL,
        verbose = getOption("verbose"), envir = .GlobalEnv)

#显示所有的数据集
data()

#加载 airquality 数据集
data("airquality")

#查看某数据集的帮助信息
?airquality
x <- airquality
# 查看数据集 x
View(x)
View(airquality)

# 查看 airquality 的汇总信息
summary(airquality)

数据集去除缺失值

```{r}
airquality[4:6, ]
  Ozone Solar.R Wind Temp Month Day
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
# 提取出非缺失值的所有数据
# 方法一
#
good <- complete.cases(airquality)
airquality[good, ][4:6, ]
# 方法二
na.omit(airquality)[4:6, ]
```

针对某一列过滤掉缺失值

```
x <- airquality
#返回逻辑型向量, Ozone 列是缺失值的返回 TRUE
is.na(x$Ozone)
# Ozone 是缺失值的返回 FALSE, 不是缺失值的返回 TRUE
!is.na(x$Ozone)

#根据单列 Ozone 列取出所有的非缺失值,
x1 <- x[!is.na(x$Ozone), ];x1
```

#两个向量对应位置都是非缺失值是返回 TRUE，对应位置有一个缺失值就返回 FALSE
```
complete.cases(c(1，2，NA，4)，c(NA，2，3，4))
```
```
[1] FALSE  TRUE FALSE  TRUE
```

 # 根据多列过滤掉缺失值

只要多列中有一个缺失值，就把整行的数据都过滤掉. 两列对应位置上都没有缺失值是返回 TRUE，有一个缺失值就返回 FALSE
```
complete.cases(x$Ozone，x$Solar.R)
```

等价于 complete.cases(x[, 1:2]). x[, 1:2]提取出第 1, 2 列的所有行，返回一个数据框，使用 complete.cases 对数据框进行判断，对数据框的每一行中的缺失值进行判断，有缺失值就返回 FALSE，如果一行中所有的数据都没有缺失值，就返回 TRUE
```
complete.cases(x[，1:2])
```

#把 Qzone 和 Solar.R 这两列中有缺失值的行都去除掉.
```
x2 <- x[complete.cases(x$Qzone，x$Solar.R)，];x2
x3 <- x[complete.cases(x[，1:2])，];x3
```

 # na.omit()直接过滤掉数据框中存在缺失值的行

na.omit() 把 x 中所有存在缺失值的行都去掉，以上两种方法都是通过取子集的方法剔除的. na.omit 则是直接对数据框进行操作.
```
x4 <- na.omit(x);x4
```

Practices

读取数据集 hw1_data.csv

- 用两种方式得到去除所有 NA 值后的数据框

```
x <- read.csv("hw1_data.csv")
x1 <- na.omit(x)
```
不指定具体的某一列时, x[complete.cases(x),] 会对每一列进行操作，过滤掉 x 中所有存在缺失值的行.
```
x2 <- x[complete.cases(x)，]
```

- 仅去除第一列的 NA 值，得到相应的数据框
```
x3 <- x[!is.na(x$Qzone)，]
```

attach, detach 和 with

R 中的全局环境与搜索路径

当 R 遇到一个变量后, 先在全局环境中查找, 再在此搜索路径中查找. 当引用某变量时每次都要指定表名较为繁琐时, 可用 attach()将数据框添加到 R 的搜索路径中

ls() #当前环境下的所有变量
rm(a) #删除当前全局环境下的某个变量
rm(list=ls()) #删除当前全局环境下的所有变量. 全局环境直观的可以理解为当前变量操作所处的环境.或者理解为能够存储当前工作变量, 系统函数和自定义函数的框架. 相当于是若干个函数和全局变量组成的一个闭环.

search() #返回当前环境下的搜索列表

如 c()这个函数不在搜索列表中, 使用?c 查看 c 的帮助, 它位于 base 这个包中, 即 package:base 这个包中. 在执行一个函数时, 就从 search()的搜索列表中一层层的去查找这个包. 在 R 中使用任何函数时都会有一个一层层搜索的过程. 如果在全局环境.GlobalEnv 中创建一个名为 c 的函数, 再去执行 c 时就不是执行创建向量这个函数了. 而是执行全局环境下的自定义 c 函数.

base 包是放在全局环境最后面的, 在用户加载其它包时, 可能会与系统自带的包存在重名的情况, 这时就默认使用最新加载的包, 以避免冲突.

使用 tools > install packages 来安装 ggplot2
或者使用命令行来安装
install.packages("ggplot2", dependencies = TRUE, lib="C:/R/R-3.4.4/library")

install.packages {utils} R Documentation
Install Packages from Repositories or Local Files

Description
Download and install packages from CRAN-like repositories or from local files.

Usage
```
install.packages(pkgs, lib, repos = getOption("repos"),
                 contriburl = contrib.url(repos, type),
                 method, available = NULL, destdir = NULL,
                 dependencies = NA, type = getOption("pkgType"),
                 configure.args = getOption("configure.args"),
                 configure.vars = getOption("configure.vars"),
                 clean = FALSE, Ncpus = getOption("Ncpus", 1L),
                 verbose = getOption("verbose"),
                 libs_only = FALSE, INSTALL_opts, quiet = FALSE,
```

keep_outputs = FALSE, ...)

Arguments

pkgs	character vector of the names of packages whose current versions should be downloaded from the repositories. If repos = NULL, a character vector of file paths of '.zip' files containing binary builds of packages. (http:// and file://URLs are also accepted and the files will be downloaded and installed from local copies.) Source directories or file paths or URLs of archives may be specified with type = "source", but some packages need suitable tools installed (see the 'Details' section). If this is missing or a zero-length character vector, a listbox of available packages is presented where possible in an interactive **R**session.
lib	character vector giving the library directories where to install the packages. Recycled as needed. If missing, defaults to the first element of .libPaths().
repos	character vector, the base URL(s) of the repositories to use, e.g., the URL of a CRAN mirror such as "https://cloud.r-project.org". For more details on supported URL schemes see url. Can be NULL to install from local files, directories or URLs: this will be inferred by extension from pkgs if of length one.
contriburl	URL(s) of the contrib sections of the repositories. Use this argument if your repository mirror is incomplete, e.g., because you burned only the 'contrib' section on a CD, or only have binary packages. Overrides argument repos. Incompatible with type = "both".
method	download method, see download.file. Unused if a non-NULL available is supplied.
available	a matrix as returned byavailable.packages listing packages available at the repositories, or NULL when the function makes an internal call toavailable.packages. Incompatible withtype = "both".
destdir	directory where downloaded packages are stored. If it is NULL (the default) a subdirectory downloaded_packages of the session temporary directory will be used (and the files will be deleted at the end of the session).
dependencies	logical indicating whether to also install uninstalled packages which these packages depend on/link to/import/suggest (and so on recursively). Not used if repos = NULL. Can also be a character vector, a subset ofc("Depends", "Imports", "LinkingTo", "Suggests", "Enhances"). Only supported if lib is of length one (or missing), so it is unambiguous where to install the dependent packages. If this is not the case it is ignored, with a warning. The default, NA, means c("Depends", "Imports", "LinkingTo"). TRUE means to use c("Depends", "Imports", "LinkingTo", "Suggests") for pkgs andc("Depends", "Imports", "LinkingTo") for added dependencies: this installs all the packages needed to runpkgs, their examples, tests and vignettes (if the package author specified them correctly). In all of these, "LinkingTo" is omitted for binary packages.

type	character, indicating the type of package to download and install. Will be "source"except on Windows and some macOS builds: see the section on 'Binary packages' for those.
configure.args	(Used only for source installs.) A character vector or a named list. If a character vector with no names is supplied, the elements are concatenated into a single string (separated by a space) and used as the value for the --configure-args flag in the call to R CMD INSTALL. If the character vector has names these are assumed to identify values for --configure-args for individual packages. This allows one to specify settings for an entire collection of packages which will be used if any of those packages are to be installed. (These settings can therefore be re-used and act as default settings.) A named list can be used also to the same effect, and that allows multi-element character strings for each package which are concatenated to a single string to be used as the value for --configure-args.
configure.vars	(Used only for source installs.) Analogous toconfigure.args for flag --configure-vars, which is used to set environment variables for the configurerun.
clean	a logical value indicating whether to add the --clean flag to the call to R CMD INSTALL. This is sometimes used to perform additional operations at the end of the package installation in addition to removing intermediate files.
Ncpus	the number of parallel processes to use for a parallel install of more than one source package. Values greater than one are supported if the make command specified bySys.getenv("MAKE", "make")accepts argument -k -j *Ncpus*.
verbose	a logical indicating if some "progress report" should be given.
libs_only	a logical value: should the --libs-onlyoption be used to install only additional sub-architectures for source installs? (See alsoINSTALL_opts.) This can also be used on Windows to install just the DLL(s) from a binary package, e.g. to add 64-bit DLLs to a 32-bit install.
INSTALL_opts	an optional character vector of additional option(s) to be passed to R CMD INSTALLfor a source package install. E.g., c("--html", "--no-multiarch"). Can also be a named list of character vectors to be used as additional options, with names the respective package names.
quiet	logical: if true, reduce the amount of output.
keep_outputs	a logical: if true, keep the outputs from installing source packages in the current working directory, with the names of the output files the package names with '.out' appended. Alternatively, a character string giving the directory in which to save the outputs. Ignored when installing from local files.
...	Arguments to be passed todownload.file or to the functions for binary installs on macOS and Windows (which accept an argument "lock": see the section on 'Locking').

library(ggplot2)
#把 ggplot2 这个包加载到搜索路径中，这个包就会出现在搜索路径中第 2 的位置上．全局环境位于第 1 位．只要加载一个新的包，就把它放在第 2 的位置上，除全局环境之外的包都会

依次向后移动.
http://www.cookbook-r.com/Graphs/

使用 attach 把 R 中的数据包加载到搜索列表中

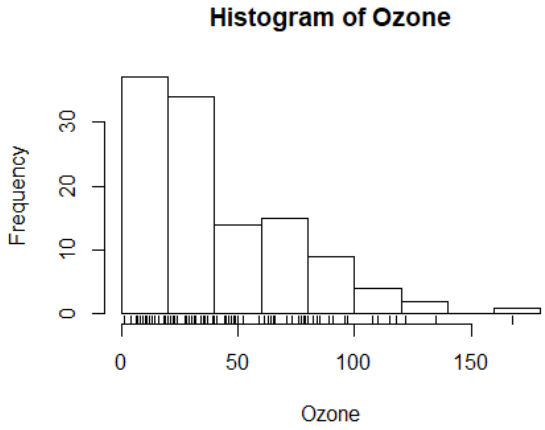
```{r}
attach(airquality)
search()
#查看搜索列表, 发现 airquality 位于第 2 个了, 把数据框也加载到搜索列表中了. 这样就可以像使用函数一样直接使用数据框中的列名了. 相当于把列名作为变量加载到了搜索路径中.
Ozone
#就不用再输入 airquality$Ozone 了.
hist(Ozone)
#作 Ozone 的直方图 Histogram of Ozone, 现在就不用指定数据框了.
# 但如果全局变量中定义了一个 Ozone 的变量, 再去作图, 就不会使用 airquality 中的 Ozone 了.
Ozone <- 1:10;hist(Ozone)
#现在就使用全局环境中的 Ozone 变量了, 不会再使用 airquality 中的 Ozone. 所以使用 attach 的时候不要与全局环境中的变量重名.

summary(Ozone)
detach(airquality)
#从搜索路径中删除 airquality
search()
```

with 方法

另一种方式是使用函数*with(表名, 语句)*, 一般用于制图, 使用 with 时, 后面所有的列都不用再指定 data 名了

```{r}
with(airquality, summary(Ozone))
with(airquality, {hist(Ozone);rug(Ozone)})
#rug 在直方图的基础上添加一个地毯图或密度图
```




```

## 向量运算，构造

# 向量化运算符

* 数学运算符: +, -, *, /, %%取余, %/%整除即取商
* 比较运算符: >=, >, <, <=, !=, ==
* 条件运算符: &, |, !, xor()异或
异或: 如果两个参数不同则返回 1，两个参数相同则返回 0

xor(1, 0)=1
xor(1, 1)=0
xor(0, 0)=0

异或满足交换率与结合率，任何数与 0 的异或都为它本身.
xor(1, 0, 1, 1, 0, 0, 1, 0, 1, 0) = xor(1, 1, 1, 1, 1)，因为任何数与 0 的异或都等于它本身，所以可以直接把 0 去掉，然后再查看 1 的个数 ，偶数个时得到的结果为 0，奇数个时得到的结果为 1.

异或在二进制计算中经常用到. 010 异或 110 = 100, 2 异或 6 = 4
十进制数与二进制数的转换 15/2=7+1, 7/2=3+1, 3/2=1+1 15-->1111，从 3/2 开始取商，记为最高位, 3/2 取余，记为第 2 位. 7/2 取余，第 3 位, 15/2 取余，第 4 位

1111 --> 2^(4-1)+2^(3-1)+2^(2-1)+2^(1-1)=8+4+2+1=15

# 向量化运算

# 即使是两个数相加，也认为是两个长度为 1 的向量相加，得到一个长度为 1 的向量. 规则
```

就是对应位上的数字相加.

```{r}
x <- 1:4;y <- 6:9;z <- 2:3
x+y
[1]  7  9 11 13
x+z
[1] 3 5 5 7
# 长度不一时, 短的循环, 且长的是短的的倍数. 循环的根源就是因为向量化运算.

z <- 2:4
x+z
[1] 3 5 7 6
Warning message:
In x + z : longer object length is not a multiple of shorter object length
# 二者的长度不是倍数时也能循环, 但是会出现一个警报信息. 所以对于很复杂的数, 尽量保证进行向量化运算的两者长度致或者是整数倍.

5 %% 2 #取余
[1] 1
5 %/% 2 #求商
[1] 2

x <- 1:4;y <- 6:9;x;y
[1] 1 2 3 4
[1] 6 7 8 9
y %% x
[1] 0 1 2 1
y %/% x
[1] 6 3 2 2
```

```{r}
x <- 1:4;y <- 6:9;x;y
[1] 1 2 3 4
[1] 6 7 8 9
y==8
[1] FALSE FALSE  TRUE FALSE
x>=2
[1] FALSE  TRUE  TRUE  TRUE
(y==8)&(x>=2)
[1] FALSE FALSE  TRUE FALSE
(y==8)|(x>=2)
[1] FALSE  TRUE  TRUE  TRUE
```

```
xor(y==8, x>=2) #异或，两个值相同返回假，两个值不同返回真
[1] FALSE  TRUE FALSE  TRUE
xor(c(T, T, F), c(F, T, F))
[1]  TRUE FALSE FALSE
```

```
## 矩阵运算

```{r}
m <- matrix(1:4, 2, 2);n <- matrix(rep(10, 4), 2, 2);m;n
m <- matrix(1:4, 2, 2);n <- matrix(10, 2, 2);m;n
     [, 1] [, 2]
[1, ]    1    3
[2, ]    2    4
     [, 1] [, 2]
[1, ]   10   10
[2, ]   10   10

#向量化运算
m*n
     [, 1] [, 2]
[1, ]   10   30
[2, ]   20   40

# 矩阵乘法
m%*%n
     [, 1] [, 2]
[1, ]   40   40
[2, ]   60   60

# 矩阵乘法解读

A <- matrix(1:6, 2, 3);A
     [, 1] [, 2] [, 3]
[1, ]    1    3    5
[2, ]    2    4    6
B <- matrix(1:6, 3, 2);B
     [, 1] [, 2]
[1, ]    1    4
[2, ]    2    5
[3, ]    3    6
```
```

A%*%B

```
     [, 1] [, 2]
[1, ]   22   49
[2, ]   28   64
```

两个矩阵相乘，左边矩阵的列数必须等于右边矩阵的行数，A(2*3)%*%B(3*2)，得到一个 2*2 的矩阵．A(2*3)%*%B(3*2) = C(2*2)，相乘的两个矩阵中间两个数相等，相乘后保留左右两边的数．C(1, 1) = A(1,)*B(, 1)，即 C 的第一个元素是 A 的第一行乘以 B 的第一列．C(i, j) 等于 A 的第 i 行，乘以 B 的第 j 列．C(i, j) = A(i,) * B(, j)．C 的第 1 列可以看成是 A 中的元素根据 B 中的第 1 列进行线性组合后得到的，即 C(, 1)=A*B(, 1)，22=1*1+3*2+5*3，28=2*1+4*2+6*3

```

# 矩阵乘法总结

理解一:

C(i, j)=A(i)*B(j), C 的第 i, j 个元素等于 A 的第 i 行乘以 B 的第 j 列

```
|A1 A2| |B1 B2 B3| |A1*B1+A2*B4, A1*B2+A2*B5, A1*B3+A2*B6| |C1 C2 C3|
|A3 A4| %*% |B4 B5 B6| = |A3*B1+A4*B4, A3*B2+A4*B5, A3*B3+A4*B6| = |C4 C5 C6|
|A5 A6| |A5*B1+A6*B4, A5*B2+A6*B5, A5*B3+A6*B6| |C7 C8 C9|
```

理解二:

根据 B 对 A 进行线性组合的变换，得到一个新的矩阵

C 的第一列是 A 的两列根据 B 中第一列的两个元素进行线性组合得到的结果.

```
A1          A2          C1
A3 * B1  +  A4 * B4  =  C4
A5          A6          C7
```

C 的第二列是 A 的两列根据 B 中第二列的两个元素进行线性组合得到的结果.

```
A1          A2          C2
A3 * B2  +  A4 * B5  =  C5
A5          A6          C8
```

理解三: 可以把 A 的每一列看成是数据的每一个变量，线性组合就是回归的表达式，就可以通过逆矩阵得到数据的系数.

常见的基本运算函数

函数	功能	示例
mean	均值	mean(1:10)
```

sd	标准差	sd(1:3)
scale	正态化	scale(1:3)
var	方差	var(1:3)
median	中位数	median(1:10)
sum	求和	sum(1:10)
rowSums	每行和	rowSums(matrix(1:10, 2, 5))
unique	去重	unique(c(1, 2, 1, 3, 2))

#养成查看帮助文件的习惯
?mean
#均值，方差，标准差，中位数，求和的对象只能是向量，如果参数是一个矩阵，会强制转换为向量. 如果是数据框，就只能对数据框中的某一列来求，使用循环或 apply 系函数对数据框的每一列求值.

```{r}
mean(1:10)
[1] 5.5

mean(c(1:10, NA))  #返回 NA，任何值与 NA 的计算都返回 NA
[1] NA

mean(c(1:10, NA), na.rm=T)  #计算时去除掉 NA 值.
[1] 5.5

sd(1:3)  # 计算的是样本的标准差
[1] 1

# scale 标准化，正态化，不是归一化. 标准化即为减去均值/标准差，标准化的主要目的是消除量纲的影响，只能对向量进行标准化.
# scale 的返回值中有两个参数, scaled:center 即均值, scaled:scale 为标准差

scale(1:10)
           [, 1]
 [1, ] -1.4863011
 [2, ] -1.1560120
 [3, ] -0.8257228
 [4, ] -0.4954337
 [5, ] -0.1651446
 [6, ]  0.1651446
 [7, ]  0.4954337
 [8, ]  0.8257228
 [9, ]  1.1560120
[10, ]  1.4863011
attr(, "scaled:center")
```

```
[1] 5.5
attr(, "scaled:scale")
[1] 3.02765

# unique 去重，可以对向量，也可以对数据框和矩阵即表格进行操作
unique(c(1, 2, 1, 3, 2))
[1] 1 2 3

x <- data.frame(x1=c(1, 1, 3), x2=c(2, 2, 3));x   #x 中有两行相同的数据，会被剔除掉，成为一个 2 行 2 列的矩阵，去除整行都重复的数据. 只有整行都重复的数据才会被去除掉.

  x1 x2
1  1  2
2  1  2
3  3  3
unique(x)
  x1 x2
1  1  2
3  3  3
```

```{r}
# rowsSums 求每行的和，是对数据框或矩阵即表格型数据的每行求和，前提是表格的元素都是数值型的或者都能转换为数值型的.

x <- matrix(1:12, 3, 4);x
     [, 1] [, 2] [, 3] [, 4]
[1, ]    1    4    7   10
[2, ]    2    5    8   11
[3, ]    3    6    9   12
#行和
rowSums(x)
[1] 22 26 30

#行均值
rowMeans(x)
[1] 5.5 6.5 7.5
#列和
colSums(x)
[1]  6 15 24 33

#列均值，对列或列操作的函数只有 rowSums, rowMeans, colSums, colMeans 这 4 个函数，没有如行方差之类的函数.
colMeans(x)
```

[1] 2 5 8 11

#这 4 个函数也有 na.rm=T 的参数.
colSums(airquality) #因为前两列都有缺失值，所以前两列的结果都为 NA
colSums(airquality, na.rm = T) #去掉缺失值
```

## ##常见的基本运算函数

都只能对向量进行运算

| 函数 | 功能 | 示例 |
|---|---|---|
| **abs** | 绝对值 | abs(-1.2) |
| **sqrt** | 平方根 | sqrt(1:10) |
| **ceiling** | 取整+1 | ceiling(pi) |
| **floor** | 取整 | floor(pi) |
| **round** | 小数位数 | round(pi, digits=2) |
| **sin** | 三角函数 | sin(pi/6) |
| **log, exp** | 对数指数 | exp(1);log(exp(2)) |

类似的还有 cos, log2, log10

```{r}
abs(c(-1, 2, -3))
[1] 1 2 3
sqrt(c(4, 9))
[1] 2 3
ceiling(c(3.1, 2.9)) #向上取整
[1] 4 3
floor(c(3.1, 2.9)) #向下取整
[1] 3 2
round(c(pi, 4.289, 5.786), digits=2)  # 四舍五入，保留几位小数
[1] 3.14 4.29 5.79
sin(c(pi/6, pi/4))
[1] 0.5000000 0.7071068
exp(1)
[1] 2.718282
exp(c(1, 2))
[1] 2.718282 7.389056

log(exp(2)) #取以 e 为底的对数
[1] 2
log(3, base=3) #取以 3 为底的对数
```
```

[1] 1
log2(4) #以 2 为底的对数，只有 log2 和 log10，没有其它的特殊对数函数了.
[1] 2
log10(1000) #以 10 为底的对数
[1] 3
```

## 创建随机数

一些函数可以用于模拟已知概率分布中的数字和变量，而正态分布是比较重要的，可以通过指定一个均值和标准差生成符合正态分布的随机变量

# 正态分布随机数

帮助文件
?rnorm

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to mean and standard deviation equal to sd.

Usage

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)

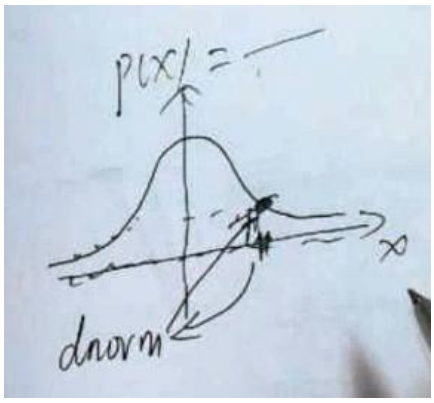
Arguments

x, q
vector of quantiles.
p
vector of probabilities.
n
number of observations. If length(n) > 1, the length is taken to be the number required.
mean
vector of means.
sd
```

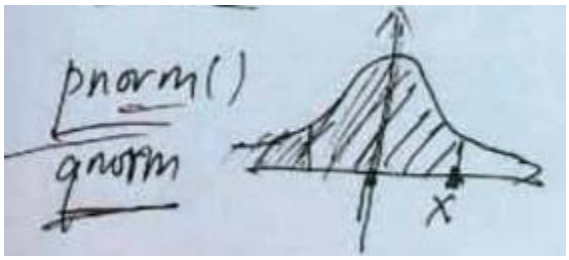
vector of standard deviations.
log, log.p
logical; if TRUE, probabilities p are given as log(p).
lower.tail
logical; if TRUE (default), probabilities are P[X ≤ x] otherwise, P[X > x].

- rnorm: 模拟出 n 个正态随机变量. r 表示 random, norm 表示 normal distribution. rnorm(n, mean = 0, sd = 1)中的第一个参数是创建多少个随机数, 第二个参数是均值, 默认为 0, 第三个参数是标准差, 默认为 1.

- dnorm: 已知分位点计算对应正态分布概率密度. (probabilty desity 为正态分布概率密度曲线上点的值). dnorm 一般用来作图, 给定一个 x, 概率密度曲线上对应的值就知道了, 给定所有的 x, 就能确定概率密度曲线上所有的值, 这个分布就确定了. 严格来说, 对于连续概率密度来说, 对应某个点的概率密度为 0,



- pnorm: 已知分位点求累积概率密度. (cumulative probabilty distribution, 正态分布累积概率分布密度), 给出分位点 x, 得到对应的正态分布累积概率密度值.



- qnorm: 给定累积概率分布密度得到对应的分位数, qnorm 是已知正态分布的前提下, 根据分布去计算分位点. 而专门计算分位点的函数 quantile 则是根据向量来计算分位点的, pnorm 和 qnorm 一般在计算分位点时使用, 分位点一般在假设检验中计算, 假设检验中一般已经给出相应的函数了, 也就不需要计算分位点了.

对于标准正态分布, 95%的置信区间对应的分位点是 1.96. -1.96-1.96 之间的概率是 95%.

```
pnorm(1.96)  pr(x<=1.96)
[1]  0.9750021
qnorm(0.975)
[1]  1.959964
```

```
qnorm(0.995)
[1] 2.575829
```

在标准正态分布中如果 Phi 表示累积分布函数，那么 pnorm(q)=Phi (q), qnorm(p)=Phi^(-1)(p)

对于任意一个概率分布，都有四种前缀
每种函数都有不同的参数，每个概率分布对应的四种函数中关于分布的参数都是一致的

- r(random)用来产生随机数; d(density)表示概率分布密度; p 累积概率分布密度; q(quantile)表示分位数
- 例如对于伽马分布，有 rgamma(), dgamma(), pgamma(), qgamma()

```
dgamma(x, shape, rate = 1, scale = 1/rate, log = FALSE)
pgamma(q, shape, rate = 1, scale = 1/rate, lower.tail = TRUE, log.p = FALSE)
qgamma(p, shape, rate = 1, scale = 1/rate, lower.tail = TRUE, log.p = FALSE)
rgamma(n, shape, rate = 1, scale = 1/rate)
```

- 对于泊松分布，有 rpois(), dpois(), ppois(), qpois()

```
dpois(x, lambda, log = FALSE)
ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)
qpois(p, lambda, lower.tail = TRUE, log.p = FALSE)
rpois(n, lambda)
```

- rpois: 给定 lambda 值产生的符合泊松分布的随机数，如果是泊松分布，只有一个参数 lambda，正态分布 norm 有 2 个参数，分别是均值和方差.

```
# 产生 10 个标准正态分布随机数
rnorm(10, 0, 1)
 [1] -1.6529257 -0.1644907 -0.1782421  0.4202467  2.0644688  0.1222271
 [7] -0.9344950 -0.7873228  0.5806802 -1.1131172

# 产生 10 个服从均值为 5，标准差为 0.1 的正态分布随机数，因为标准差很小，所以数据都集中在 5 的周围.
rnorm(10, 5, 0.1)
 [1] 5.100165 5.104755 5.028688 5.020089 5.066352 5.028846 4.912233
 [8] 5.059072 5.165502 5.134770
```

均匀分布随机数

- 对于均匀分布，有 runif(), dunif(), punif(), qunif()

```
dunif(x, min = 0, max = 1, log = FALSE)
punif(q, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE)
qunif(p, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE)
```

runif(n, min = 0, max = 1)

# 创建 10 个从-5 到 5 的服从均匀分布的随机数
runif(10, -5, 5)
```
 [1]  2.414477  1.673665 -4.245911  2.550708  4.443636  4.076298 -4.222902
 [8] -3.739893  4.289168  3.355048
```

二项分布随机数

dbinom(x, size, prob, log = FALSE)
pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
rbinom(n, size, prob)

Arguments

x, q
vector of quantiles.
p
vector of probabilities.
n
number of observations. If length(n) > 1, the length is taken to be the number required.
size
number of trials (zero or more).
prob
probability of success on each trial.
log, log.p
logical; if TRUE, probabilities p are given as log(p).
lower.tail
logical; if TRUE (default), probabilities are P[X ≤ x], otherwise, P[X > x].

#rbinom. 最常用的是二项分布，二项分布的概率密度函数是 P(k)=C(n, k)P^k*(1-p)^(n-k), rbinom 的两个参数即为 n 和 p
具有固定概率的独立事件的多次实验中出现 n 次的概率就是二项分布的概率密度函数，如射中靶子的概率为 0.8，10 次射击，射中 7 次的概率即为 P(7)=C(10, 7)*0.8^7*(1-0.8)^3.也可以理解为投硬币，只是正面朝上的概率是 0.8，投了 10 次，有 7 次正面朝上的概率.

#射中概率为 0.7，每射击 20 次为一个样本，多次重复，射中的次数就服从二项分布.
rbinom(10, 20, 0.7) 就得到重复进行 10 次实验射中的次数.
rbinom(10, 20, 0.7)
```
 [1] 15 15 15 12 14 11 13 12 14 14
```
# 当射中的概率越大时，得到的射中次数就越多.
rbinom(10, 20, 0.9)

[1] 17 18 18 18 18 20 18 20 19 19

投一次硬币实际上为伯努利分布，二项分布就相当于进行了多次的伯努利分布的实验.

设置种子创建随机数

在任何情况下，从任意分布模拟随机数字时，设置随机数字生成器种子(seed)是非常重要的，电脑生成的数字并不是真正的随机数，但是看起来是随机的，所以可以做到重复生成同样一组随机数，这些数叫做伪随机数(pseudo random number)，这样就满足了代码可重复性的要求.

既然已经是随机数了，为什么还要重复呢. 在某些模型中，如在随机森林中，是通过随机的过程进行计算的，为了在多次模拟中得到相同的结果，就需要给它设置一个种子，使随机数产生的过程和结果都是一样的，这样随机森林才能得到相同的结果.

```{r}
set.seed(1);rnorm(4) #只要设置的种子一样，生成的随机数就是一样的. 每一个种子对应一组随机数的集合，设置种子之后再选取随机数就是从这些集合中选择. 随机数种子只能是整数. 这是为了在特定情况下满足代码的可重复性. 每一个种子就对应了一种抽取随机数的方法，一个种子对应一种随机性，但无论设置种子是多少，得到的随机数都是服从正态分布的. 设置种子只是为了满足代码可重复性的要求.

rnorm(4)
[1] -0.74366961 -1.02486852  0.03890159 -0.15516067
set.seed(1);rnorm(4);rnorm(5)
[1] -0.6264538  0.1836433 -0.8356286  1.5952808
[1]  0.3295078 -0.8204684  0.4874291  0.7383247  0.5757814

#rnorm 就是从标准正态分布中选择的随机数，所以得到的数字都是分布在 0 附近的. 得到的数据一定服从标准正态分布. 如果生成足够多的随机数，用这些随机数作图，得出的图形就是标准正态分布的图形.

set.seed(1);rnorm(4)
[1] -0.6264538  0.1836433 -0.8356286
[4]  1.5952808
set.seed(1);rnorm(5)
[1] -0.6264538  0.1836433 -0.8356286
[4]  1.5952808  0.3295078
每次设置一次种子都会重置随机数池，然后再从池中的第一个随机数开始选择.

# set.seed()也可以用于产生服从其它类型分布的随机数. 如

set.seed(2);runif(8)
[1] 0.1848823 0.7023740 0.5733263 0.1680519 0.9438393 0.9434750 0.1291590
```

```
[8] 0.8334488
set.seed(2);runif(10)
 [1] 0.1848823 0.7023740 0.5733263 0.1680519 0.9438393 0.9434750 0.129159
0
 [8] 0.8334488 0.4680185 0.5499837
```

随机抽样

sample()能够从指定的特定对象集合中随机取样

```{r}
# 设置种子的不放回抽样
set.seed(1);sample(1:10, 4)
[1] 3 4 5 7
set.seed(1);sample(1:10, 8)
[1] 3 4 5 7 2 8 9 6

# 从 letters 中不放回的抽取 5 个数字.
sample(letters, 5)
[1] "f" "w" "y" "p" "n"

#不设置抽取的个数时默认抽取与向量等长的个数，只是把原数据重新随机排列了一样. 可以用于变量的随机重排,
sample(1:10)
 [1] 1 2 9 5 3 4 8 6 7 10

#设置 replace=T 有放回的抽取
sample(1:10, replace=T)
 [1] 4 5 6 5 2 9 7 8 2 8

# 对于不放回抽样，每次抽取的样本量要小于总体的长度. 对于放回抽样，抽取的样本量可以大于总体的长度
sample(1:10, 15)
Error in sample.int(length(x), size, replace, prob) :
  cannot take a sample larger than the population when 'replace = FALSE'

sample(1:10, 15, replace=T)
 [1] 10 3 7 2 3 4 1 4 9 4 5 6 5 2 9
```

sample 抽取训练集和测试集

#sample 可以用来抽取训练集和测试集. 可以设置种子来得到可重复的结果, 但一般不设置种子, 多次抽取来确定自己的模型是否合理, 如果不同的抽取得到的结果偏差太大, 则这个模型是不合理的.

```{r}
#查看 airquality 的行数
n <- nrow(airquality);n
[1] 153

#抽出来 70%的数, ariquality 一共有 n=153 行, n*0.7=107, 从 airquality 中随机抽取出 107 行数据, 得到的是行索引值
  [1]  37  40 111  68  27 151  16 127  90  81  48  65  71
 [14]  26  74  11  39  29 137 120  60 103 116  54   9  43
 [27]  92 128  79 105 106 143  47 108  77  88 141 124  34
 [40]  22 101  57  98  21  83 125 113  59  75  41 138  95
 [53] 136 133 132 109  31  76 140 110  49  25  17 121  51
 [66]  12  23  62  82 129  64  78  67 146  52 131 139 102
 [79]  20  13  24 153  66  36  18   4  28 112  87  84  73
 [92] 152  63  42 104  33  80  96  91  32  93   8 114  99
[105]  89   7  53

#抽出来 70%的数, 得到的是行号
index <- sample(1:n, n*0.7); index

#根据行索引 index 从 airquality 中取子集, 得到训练集
train <- airquality[index, ];train

#余下的 30%的数据作为测试集
test <- airquality[-index, ];test

```

随机数总结

- 对于任意一个概率分布均有四个前缀 r, d, p, q
- sample 常用于建立测试集, 建立随机数之前一定设定种子

##常见的向量构建函数

创建重复数*rep(x, times, each)*

rep {base}　　R Documentation

Replicate Elements of Vectors and Lists

Description

rep replicates the values in x. It is a generic function, and the (internal) default method is described here.

rep.int and rep_len are faster simplified versions for two common cases. They are not generic.

Usage

rep(x, ...)

rep.int(x, times)

rep_len(x, length.out)

Arguments

x	**a vector (of any mode including a list) or a factor or (for rep only) a POSIXct or POSIXlt or Date object; or an S4 object containing such an object.**
...	further arguments to be passed to or from other methods. For the internal default method these can include: times an integer-valued vector giving the (non-negative) number of times to repeat each element if of length length(x), or to repeat the whole vector if of length 1. Negative or NA values are an error. A double vector is accepted, other inputs being coerced to an integer or double vector. length.out non-negative integer. The desired length of the output vector. Other inputs will be coerced to a double vector and the first element taken. Ignored if NA or invalid. each

	non-negative integer. Each element of x is repeated each times. Other inputs will be coerced to an integer or double vector and the first element taken. Treated as1 if NA or invalid.
times, length.out	see ... above.

```{r}
#1 重复 4 次
rep(1, 4)
[1] 1 1 1 1
rep(1, times=4)
[1] 1 1 1 1

#向量重复 4 次
rep(1:2, 4)
[1] 1 2 1 2 1 2 1 2
rep(c(1, 5), 4)
[1] 1 5 1 5 1 5 1 5
rep(c("男", "女"), 4)
[1] "男" "女" "男" "女" "男" "女" "男" "女"

#每个元素先重复，再下一个元素重复
rep(1:2, each=4)
[1] 1 1 1 1 2 2 2 2

#每个元素先重复 each 次，再整体上从头开始重复 times 次
rep(c(1, 5), each=4, times=3)
 [1] 1 1 1 1 5 5 5 5 1 1 1 1 5 5 5 5 1 1 1 1 5 5 5 5
```

创建等差数列

seq(from, to, by)
seq_along(x)
sequence(n)

?seq
seq {base}　　R Documentation
Sequence Generation

Description
Generate regular sequences. seq is a standard generic with a default method. seq.int is a primitive which can be much faster but has a few restrictions. seq_along and seq_len are very fast primitives for two common cases.

Usage

```
seq(...)

## Default S3 method:
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)

seq.int(from, to, by, length.out, along.with, ...)

seq_along(along.with)
seq_len(length.out)
```

Arguments

...	**arguments passed to or from methods.**
from, to	the starting and (maximal) end values of the sequence. Of length 1 unless just from is supplied as an unnamed argument.
by	number: increment of the sequence.
length.out	desired length of the sequence. A non-negative number, which for seq and seq.intwill be rounded up if fractional.
along.with	take the length from the length of this argument.

?sequence
sequence {base} R Documentation

Create A Vector of Sequences

Description

For each element of nvec the sequence seq_len(nvec[i]) is created. These are concatenated and the result returned.

Usage

```
sequence(nvec)
```

Arguments

nvec	**a non-negative integer vector each element of which specifies the end point of a sequence.**

```{r}
可以使用 from to by, 也可以使用 from by length.out

# 使用 1:5 这种方式只能创建公差为 1 的等差数列.
-5:-10
[1]  -5  -6  -7  -8  -9 -10
seq(2, 7, 0.5)
 [1] 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0
seq(1, 10, 2)
[1] 1 3 5 7 9
seq(-5, -10, -1)
[1]  -5  -6  -7  -8  -9 -10

#创建一个长度为 10, 公差为 3 的数列.
seq(0, by=3, length.out=10)
 [1]  0  3  6  9 12 15 18 21 24 27
seq(0, by=3, length=10)
 [1]  0  3  6  9 12 15 18 21 24 27

# seq_along(x)相当于 seq(1:length(x))
x <- c("a", "b", "c");seq_along(x)
[1] 1 2 3
seq(1:length(x))
[1] 1 2 3

#相当于 c(1:3, 1:4, 1:5)
sequence(3:5)
 [1] 1 2 3 1 2 3 4 1 2 3 4 5
c(1:3, 1:4, 1:5)
 [1] 1 2 3 1 2 3 4 1 2 3 4 5
```

##Practices

1. 创建 1 个数据框 a, 在保证可重复性的前提下, 每列要求如下:
    - rn 列, 创建 100 个满足均值为 5, 标准差为 2 的正态分布的随机数
    - lt 列, 随机抽取 100 个字母, 可重复
    - sa 列, 在 1 到 10 中随机抽取 100 个数, 可重复
    - rp 列, 1122334455 重复 10 次, 创造 100 个重复数
    - se 列, 创建 0 到 49.5 的等差数列, 差值为 0.5

```
set.seed(1)
a <- data.frame(rn=rnorm(100, 5, 2),
                lt=sample(letters, 100, replace = T),
                sa=sample(1:10, 100, replace = T),
```

```
rp=rep(1:5, each=2, times=10),
se=seq(0, 49.5, 0.5),
stringsAsFactors = F)
```

2. 对 rn 列求均值 r1，求标准差 r2，求方差 r3，求中位数 r4，求和 r5

```
r1 <- mean(a$rn);r1
[1] 5.217775
r2 <- sd(a$rn);r2
[1] 1.796399
r3 <- var(a$rn);r3
[1] 3.227048
r4 <- median(a$rn);r4
[1] 5.227818
r5 <- sum(a$rn);r5
[1] 521.7775
```

3. 求除了 lt 列的每行均值，除了 lt 列的每列和

```
#排除第 2 列
rs <- rowMeans(a[, -2])
#排除第 2 列
cs <- colSums(a[, -2]);cs
        rn         sa         rp         se
 521.7775   546.0000   300.0000  2475.0000
```

4. 生成新列 logrn，取 rn 列底数为 e 的对数生成此新列

```
a$logrn <- log(a$rn);a$logrn
```

#R 中的日期和时间函数

日期函数 as.Date()

R 用一种特殊的类表示日期和时间，如同之前的列表类

- 日期用 Date 类表示，不包含时间，内部以 1970-01-01 至今的天数存储，可以使用 as.Date() 函数强制转化字符串型时间为 Date 类
- 时间则由两个不同的类, POSIXct 和 POSIXlt 来表示，内部时间则是以 1970-01-01 08:00:00 至今的秒数来存储的

?as.Date 位于 base 包中.

as.Date {base} **R Documentation**

Date Conversion Functions to and from Character

Description

Functions to convert between character representations and objects of class "Date" representing calendar dates.

Usage

```
as.Date(x, ...)
## S3 method for class 'character'
as.Date(x, format, ...)
## S3 method for class 'numeric'
as.Date(x, origin, ...)
## S3 method for class 'POSIXct'
as.Date(x, tz = "UTC", ...)

## S3 method for class 'Date'
format(x, ...)

## S3 method for class 'Date'
as.character(x, ...)
```

Arguments

x	**An object to be converted.**
format	A character string. If not specified, it will try "%Y-%m-%d" then "%Y/%m/%d" on the first non-NA element, and give an error if neither works. Otherwise, the processing is via strptime
origin	a Date object, or something which can be coerced by as.Date(origin, ...) to such an object.
tz	a time zone name.
...	Further arguments to be passed from or to other methods, including format foras.character and as.Date methods.

lubridate 这个包中的时间处理函数会用的比较多.

```{r}
x <- as.Date(c("1970-01-01"));x
[1] "1970-01-01"
class(x)
```

[1] "Date"

#unclass 去类型化，返回 1，表明时间是使用整数进行存储的. 1970-01-01 之前的日期是负数
unclass(x)
[1] 0
unclass(as.Date("1969-01-01"))
[1] -365

#系统当前的日期
Sys.Date()
[1] "2018-05-04"
unclass(Sys.Date())
[1] 17655

t <- c("2015-11-04", "2018-10-03")
t1 <- as.Date(t);t1
[1] "2015-11-04" "2018-10-03"
#在 R 中只要是分类型变量，就把它转换为因子，只要是时间或日期，就转换为时间或日期类，以方便后期的数据处理，建模和作图.
class(t1)
[1] "Date"

#只有"2015-11-04"和"2015/11/04"这两种标准的日期格式的字符串才会被 as.Date 函数识别，其它非标准的则需要转化.
as.Date("2017/12/16")
[1] "2017-12-16"

unclass(as.Date("1970-02-01"))
[1] 31
```

许多绘图函数能识别日期时间对象，会给 x 轴一个特殊的格式

## ## 时间函数 POSIXct, POSIXlt

?DateTimeClasses

There are two basic classes of date/times. Class "POSIXct" represents the (signed) number of seconds since the beginning of 1970 (in the UTC time zone) as a numeric vector. Class"POSIXlt" is a named list of vectors representing

| | |
|---|---|
| **sec** | 0–61: seconds. |
```

min	0–59: minutes.
hour	0–23: hours.
mday	1–31: day of the month
mon	0–11: months after the first of the year.
year	years since 1900.
wday	0–6 day of the week, starting on Sunday.
yday	0–365: day of the year.
isdst	Daylight Saving Time flag. Positive if in force, zero if not, negative if unknown.
zone	(Optional.) The abbreviation for the time zone in force at that time: "" if unknown (but ""might also be used for UTC).
gmtoff	(Optional.) The offset in seconds from GMT: positive values are East of the meridian. Usually NA if unknown, but 0 could mean unknown.

时间由两种类型表示: POSIXct 和 POSIXlt, POSIX 是一个计算标准家族，规定了特定类型的计算机处理事件和数据表示的方法，有一个标准族规定如何表示日期和时间

- POSIXct 类中，时间使用非常大的整数来表示的，常用于将时间存储在数据框中，整数是至今的秒数
- POSIXlt 把时间当做列表来存储，存储跟给定时间相关的信息，例如这个时间是星期几，是一年中的第几天

可以在 POSIXct 和 POSIXlt 之间使用 as.POSIXct()或 as.POSIXlt()函数进行类型转换

```{r}
t2 <- as.POSIXct("2017-12-16 17:22:00");t2
[1] "2017-12-16 17:22:00 CST"

# 返回两种时间类型, POSIXct 和 POSIXt
class(t2)
[1] "POSIXct" "POSIXt"

#R 中的时间是以 1970 年上午 8:00:00 分为零点存储的，存储的是零点到现在的秒数.
unclass(t2)
[1] 1513416120
attr(, "tzone")
[1] ""

t3 <- as.POSIXlt("2017-12-16 17:22:00");t3
[1] "2017-12-16 17:22:00 CST"

# 返回两种时间类型, POSIXct 和 POSIXt
class(t3)
[1] "POSIXlt" "POSIXt"
```

#把时间拆分为列表，可以从中提取出天，日，小时，分，秒等. 月是以 0 开始的，年是以 1970 年开始的, wday 周日到周六是从 0 到 6.

```
unclass(t3)
$sec
[1] 0

$min
[1] 22

$hour
[1] 17

$mday
[1] 16

$mon
[1] 11

$year
[1] 117

$wday
[1] 6

$yday
[1] 349

$isdst
[1] 0

$zone
[1] "CST"

$gmtoff
[1] NA
```

#得到某日期是该年中的第多少天

```
t3$yday
[1] 349
```

##得到的系统时间是 POSIXct 格式的

```
x <- Sys.time();x
[1] "2018-05-04 13:58:37 CST"
unclass(x)
```

```
[1] 1525413517

# 转换为 POSIXlt 格式的时间
p <- as.POSIXlt(x);p
[1] "2018-05-04 13:58:37 CST"

# 把 POSIXlt 格式的时间 unclass 后转换为数据框的形式
as.data.frame(unclass(p))
        sec min hour mday mon year wday yday isdst zone gmtoff
1 37.15278  58   13    4   4  118    5  123     0  CST  28800

# 只能对 POSIXlt 格式的日期或时间使用, 不能用于 POSIXct 格式, 如 x$hour, x$min, x$sec
p$hour
[1] 13
p$min
[1] 58
p$sec
[1] 37.15278
```

泛型函数

```
# 泛型函数. 可以使用时间, 也可以使用日期. 每次只能求出一个时间或日期, 不能对向量使用.
x <- Sys.time();x
[1] "2018-05-04 13:58:37 CST"
# 转换为 POSIXlt 格式的时间
p <- as.POSIXlt(x);p
[1] "2018-05-04 13:58:37 CST"

weekdays(p)
[1] "星期五"
weekdays(x)
[1] "星期五"
# months 和 weekdays 可以根据系统语言显示为中文或英文.
months(p)
[1] "五月"
months(x)
[1] "五月"
# 第几季度, 只能显示 Q1, Q2, Q3, Q4
quarters(p)
[1] "Q2"
quarters(x)
[1] "Q2"
```

- weekdays()给定的时间或日期是星期几，不能对日期型向量求值，只能对单一的日期或时间求值.

```
weekdays(as.Date("2017-12-16  17:22:00"))
[1] "星期六"
weekdays(as.POSIXct("2017-12-16  17:22:00"))
[1] "星期六"
weekdays(as.POSIXlt("2017-12-16  17:22:00"))
[1] "星期六"
```

- months()给定的时间或日期在几月

```
months(as.Date("2017-12-16  17:22:00"))
[1] "十二月"
months(as.POSIXct("2017-12-16  17:22:00"))
[1] "十二月"
months(as.POSIXlt("2017-12-16  17:22:00"))
[1] "十二月"
```

- quarters()处于第几季度("Q1", "Q2", "Q3", "Q4")

```
quarters(as.Date("2017-12-16  17:22:00"))
[1] "Q4"
quarters(as.POSIXct("2017-12-16  17:22:00"))
[1] "Q4"
quarters(as.POSIXlt("2017-12-16  17:22:00"))
[1] "Q4"
```

```

## ## 时间转换函数 strptime

strptime()函数用于将任意格式的字符串表示的日期转换成时间对象 POSIXlt，用到了所谓的格式字符串
标准的时间日期格式的字符串可以直接使用函数，非标准的时间日期格式的字符串就要使用格式字符串

strptime {base}　　R Documentation

**Date-time Conversion Functions to and from Character**

**Description**

Functions to convert between character representations and objects of classes "POSIXlt"and "POSIXct" representing calendar dates and times.

**Usage**
```

S3 method for class 'POSIXct'
format(x, format = "", tz = "", usetz = FALSE, ...)
S3 method for class 'POSIXlt'
format(x, format = "", usetz = FALSE, ...)

S3 method for class 'POSIXt'
as.character(x, ...)

strftime(x, format = "", tz = "", usetz = FALSE, ...)
strptime(x, format, tz = "")

Arguments

x	An object to be converted: a character vector for strptime, an object which can be converted to "POSIXlt" for strftime.
tz	A character string specifying the time zone to be used for the conversion. System-specific (see as.POSIXlt), but "" is the current time zone, and "GMT" is UTC. Invalid values are most commonly treated as UTC, on some platforms with a warning.
format	A character string. The default for the format methods is "%Y-%m-%d %H:%M:%S" if any element has a time component which is not midnight, and "%Y-%m-%d" otherwise. If options("digits.secs") is set, up to the specified number of digits will be printed for seconds.
...	Further arguments to be passed from or to other methods.
usetz	logical. Should the time zone abbreviation be appended to the output? This is used in printing times, and more reliable than using "%Z".

##格式字符串
?strptime 在帮助文件中查找格式字母的含义

符号	含义	示例
%d	数字表示的日期	01-31
%a, %A	缩写，非缩写的星期名	Mon, Monday
%m	十进制月份	01-12
%b, %B	缩写，非缩写的月份	Jan, January
%y, %Y	两位数，四位数的年份	07, 2007
%H	小时 00-23	00-23
%M	分钟 00-59	00-59
%S	秒 00-61	00-61

```{r, warning=FALSE}
date <- c("01 10, 2012 10:40:11", "12 09, 2011 09:10:39");date
```

```
[1] "01 10, 2012 10:40:11" "12 09, 2011 09:10:39"

#格式化为时间, m-month, d-day, Y-Year, H-Hour, M-Min, S-Sec

t1 <- strptime(date, "%m %d, %Y %H:%M:%S");t1
[1] "2012-01-10 10:40:11 CST" "2011-12-09 09:10:39 CST"
t1 <- as.POSIXct(date, format="%m %d, %Y %H:%M");t1
[1] "2012-01-10 10:40:00 CST" "2011-12-09 09:10:00 CST"
t2 <- as.POSIXlt(date, format="%m %d, %Y %H:%M");t2
[1] "2012-01-10 10:40:00 CST" "2011-12-09 09:10:00 CST"
t3 <- strptime(date, format="%m %d, %Y %H:%M");t3
[1] "2012-01-10 10:40:00 CST" "2011-12-09 09:10:00 CST"

#格式化为日期
t4 <- as.Date(date, format="%m %d, %Y %H:%M");t4
[1] "2012-01-10" "2011-12-09"

class(t1)
[1] "POSIXct" "POSIXt"
```

日期和时间运算

日期和日间是使用整数存储的, 因子也是按整数存储的, 因子可以直接转换为整数参与运算

使用数学运算符(+, -,)和比较运算符(==, <=等)计算日期和时间, 必须是相同的时间类之间才能运算

```{r, error=T}
x <- as.Date("2012-01-01");x;class(x)
[1] "2012-01-01"
[1] "Date"
y <- strptime("9 01 2011 11:34:21", "%d %m %Y %H:%M:%S");y;class(y)
[1] "2011-01-09 11:34:21 CST"
[1] "POSIXlt" "POSIXt"
x-y
Error in x - y : non-numeric argument to binary operator
In addition: Warning message:
Incompatible methods ("-.Date", "-.POSIXt") for "-"
x <- as.POSIXlt(x)
x <- as.POSIXlt(x);class(x);class(y)
[1] "POSIXlt" "POSIXt"
```

```
[1] "POSIXlt" "POSIXt"
x;y;x-y
[1] "2012-01-01 UTC"
[1] "2011-01-09 11:34:21 CST"
Time difference of 356.8511 days

时间差值计算
t1 <- Sys.time();t1
[1] "2018-05-04 14:37:16 CST"
t2 <- as.POSIXct("2018-05-03 12:00:00");t2
[1] "2018-05-03 12:00:00 CST"

#超过 1 天就以天为单位
t1-t2
Time difference of 1.109216 days

t3 <- as.POSIXct("2018-05-03 15:00:00");t3
[1] "2018-05-03 15:00:00 CST"
#低于 1 天就以小时为单位
t1-t3
Time difference of 23.62118 hours

#类型为 difftime, 可以转换为整数型的
class(t1-t3)
[1] "difftime"
as.numeric(t1-t2)
[1] 1.109216
as.numeric(t1-t3)
[1] 23.62118
```

注意:
1. 只能时间-时间, 日期-日期, 不能日期-时间.
2. 会自动考虑到闰年和时区的差别
3. 日期时间运算符的好处在于能记录闰年, 时令, 时区

```{r}
x <- as.Date("2012-03-01");x
[1] "2012-03-01"
y <- as.Date("2012-02-28");y
[1] "2012-02-28"
x-y
Time difference of 2 days
```

```
x <- as.POSIXlt("2012-10-25 01:00:00");x
[1] "2012-10-25 01:00:00 CST"
y <- as.POSIXlt("2012-10-25 06:00:00", tz="GMT");y
[1] "2012-10-25 06:00:00 GMT"
y-x
Time difference of 13 hours
```

计算时间差

时间间隔有若干种计算的方式，计算时又常常需要考虑闰年和时区的影响，默认得到的结果是天数，也可以简单的换算成相应的时，分，秒，周．而如果要换算成月和年，需要用到序列或者外部包里的函数

base 包中常见的计算时间差的方式有:

- time1-time2
- difftime(time1, time2, units)

?difftime 查看 units 可取的值.

```
difftime(time1, time2, tz,
         units = c("auto", "secs", "mins", "hours",
                   "days", "weeks"))
```

```
t1 <- as.POSIXct("2016-12-17 12:00:00");t1
[1] "2016-12-17 12:00:00 CST"
t2 <- as.POSIXlt("2017-12-17 17:00:00");t2
[1] "2017-12-17 17:00:00 CST"
difftime(t1, t2, units = "weeks")
Time difference of -52.17262 weeks
difftime(t1, t2, units = "mins")
Time difference of -525900 mins
```

想要得到年和月，就不能使用 days/365 或 days/30，还要考虑闰年或每月不同的天数的影响，可以使用 seq 函数来间接完成.

#使用 seq 得到按时间排列的等差数列，再通过 length()查看间隔

- seq(time2, time1, by)

seq 从小时间到大时间, from ... to ...

?seq.Date

```
seq(from, to, by, length.out = NULL, along.with = NULL, ...)
```

"by" can be specified in several ways.

A number, taken to be in days.

A object of class difftime

A character string, containing one of "day", "week", "month", "quarter" or "year". This can optionally be preceded by a (positive or negative) integer and a space, or followed by "s".

精确计算时间间隔，可以用于精确计算年龄.

```{r}
t1 <- as.POSIXct("2015-12-17 13:00:00");t1
[1] "2015-12-17 13:00:00 CST"
t2 <- as.POSIXct("2017-12-17 12:00:00");t2
[1] "2017-12-17 12:00:00 CST"

#因为结束日期中还没有到 2017-12-17 13:00:00，所以只把 2015,2016 年算进去了，没有计算 2017 年. 从开始的时间点，每隔一年在结果中加上一个值，直到结束的时间点，如果结束的时间点还没有到，就到离结束时间点最近的时间点..
y <- seq(t1, t2, by="year");y
[1] "2015-12-17 13:00:00 CST" "2016-12-17 13:00:00 CST"

#得到精确的年份，但这种方法只能对一个时间间隔进行计算，不能用于向量的计算. 扩展包 lubridate 中有可以对向量进行运算的函数.
length(y)-1
[1] 1

t3 <- as.POSIXct("2000-12-17 23:00:00");t3
[1] "2000-12-17 23:00:00 CST"

# 指定间隔为 2 year
y2 <- seq(t3, t2, by="2 year");y2
[1] "2000-12-17 23:00:00 CST" "2002-12-17 23:00:00 CST" "2004-12-17 23:00:00 CST"
[4] "2006-12-17 23:00:00 CST" "2008-12-17 23:00:00 CST" "2010-12-17 23:00:00 CST"
[7] "2012-12-17 23:00:00 CST" "2014-12-17 23:00:00 CST" "2016-12-17 23:00:00 CST"
length(y2)
[1] 9

# 从近的时间向过去的远的时间计算
y3 <- seq(t2, t3, by="-3 year");y3
[1] "2017-12-17 12:00:00 CST" "2014-12-17 12:00:00 CST" "2011-12-17 12:00:00 CST"
```

[4] "2008-12-17 12:00:00 CST" "2005-12-17 12:00:00 CST" "2002-12-17 12:00:00 CST"

生成时间序列
```
seq(t3, by="2 year", length.out=10)
 [1] "2000-12-17 23:00:00 CST" "2002-12-17 23:00:00 CST" "2004-12-17 23:00:00 CST"
 [4] "2006-12-17 23:00:00 CST" "2008-12-17 23:00:00 CST" "2010-12-17 23:00:00 CST"
 [7] "2012-12-17 23:00:00 CST" "2014-12-17 23:00:00 CST" "2016-12-17 23:00:00 CST"
[10] "2018-12-17 23:00:00 CST"
```

还可以使用 by month, by week, by day 等.
```

## 计算时间差

as.difftime(tim, format = "%X", units = "auto")

```{r}
time1 <- as.POSIXlt("2017-11-04 11:59:00");time1
[1] "2017-11-04 11:59:00 CST"
time2 <- as.POSIXlt("2015-11-04 12:00:00");time2
[1] "2015-11-04 12:00:00 CST"
time1-time2
Time difference of 730.9993 days

difftime(time1, time2, units="weeks")
Time difference of 104.4285 weeks

td <- seq(time2, time1, by="year");td
[1] "2015-11-04 12:00:00 CST" "2016-11-04 12:00:00 CST"
length(td)
[1] 2

seq(as.POSIXlt("2015-11-04 12:00:00"), by="month", length.out = 4)
[1] "2015-11-04 12:00:00 CST" "2015-12-04 12:00:00 CST" "2016-01-04 12:00:00 CST"
[4] "2016-02-04 12:00:00 CST"
seq(as.POSIXlt("2015-11-04 12:00:00"), by="-1 month", length.out = 4)
[1] "2015-11-04 12:00:00 CST" "2015-10-04 12:00:00 CST" "2015-09-04 12:00:00 CST"
[4] "2015-08-04 12:00:00 CST"
```
```

```

## Practices

1. 将字符串"2010/11/04"和"11-04/2010"分别转换为日期 Date 类

```
as.Date("2010/11/04")
[1] "2010-11-04"
as.Date("11-04/2010", format="%m-%d/%Y")
[1] "2010-11-04"
```

2. 查看当前系统时间，查看系统时间类别，查看系统时间对应的周几，几月，哪个季度

```
t1 <- Sys.time();t1
[1] "2018-05-04 15:11:57 CST"
class(t1)
[1] "POSIXct" "POSIXt"
weekdays(t1)
[1] "星期五"
months(t1)
[1] "五月"
quarters(t1)
[1] "Q2"
```

3. 将字符串"11-04 2016, 33:59 23h", "12-21 2017, 23m59s 23h"分别转换为 POSIXct 和 POSIXlt 时间格式，要用到 as.POSIXct, as.POSIXlt, strptime 函数

```
as.POSIXct("11-04 2016, 33:59 23h", format="%m-%d %Y, %M:%S %Hh")
[1] "2016-11-04 23:33:59 CST"
as.POSIXlt("12-21 2017, 23m59s 23h", format="%m-%d %Y, %Mm%Ss %Hh")
[1] "2017-12-21 23:23:59 CST"
```

4. 计算从自己出生到现在的年，月，日，时，分，秒

```
t1 <- Sys.time();t1
[1] "2018-05-04 15:12:45 CST"
t2 <- as.POSIXct("1986-11-04 00:00:00");t2
[1] "1986-11-04 CST"
```

时间在保存时是使用小数保存的，使用 difftime 直接计算两个时间的差值得到的结果就是秒，秒除以 60 就是分，分除以 60 就是小时，小时除以 24 就是天. 但年和月就不是这样计算的，要使用 seq 函数间接计算.

```
difftime(t1, t2, units="days")
Time difference of 11504.63 days
```
```

```
difftime(t1, t2, units="hours")
Time difference of 276111.2 hours
difftime(t1, t2, units="mins")
Time difference of 16566673 mins
difftime(t1, t2, units="secs")
Time difference of 994000365 secs

# 计算多少年和多少个月
length(seq(t2, t1, "month"))-1
[1] 378
length(seq(t2, t1, "year"))-1
[1] 31
```

5. 生成从当前时间开始前 12 个月的降序时间序列

```
seq(t1, by="-1 month", length.out = 12)
 [1] "2018-05-04 15:12:45 CST" "2018-04-04 15:12:45 CST" "2018-03-04 15:1
2:45 CST"
 [4] "2018-02-04 15:12:45 CST" "2018-01-04 15:12:45 CST" "2017-12-04 15:1
2:45 CST"
 [7] "2017-11-04 15:12:45 CST" "2017-10-04 15:12:45 CST" "2017-09-04 15:1
2:45 CST"
[10] "2017-08-04 15:12:45 CST" "2017-07-04 15:12:45 CST" "2017-06-04 15:1
2:45 CST"
```

#字符处理函数

字符处理函数主要用于在字符变量中提取有用的信息，或者将有用的信息规范化，尤其是处理从网络上爬取的字符串的时候，需要从中提取出对象若干的属性信息，这时候就需要很熟悉相关字符处理函数的使用

下面我们例举一些常见的字符处理函数

- 连接字符串 paste

- 连接字符 paste(..., sep=" ", collapse=NULL), paste0, paste0 相当于 paste 中 sep 默认为""

```
?paste
```

Concatenate Strings

Description

Concatenate vectors after converting to character.

Usage

paste (..., sep = " ", collapse = NULL)

paste0(..., collapse = NULL)

Arguments

...	one or more **R** objects, to be converted to character vectors.
sep	a character string to separate the terms. Not NA_character_.
collapse	an optional character string to separate the results. Not NA_character_.

```{r}
#3 个长度为 1 的向量
paste("ai", "ya", "ya", sep="*")
[1] "ai*ya*ya"

#把多个向量对应位置的元素连接起来，两个向量的个数尽可能是倍数或一致，与向量的四则运算方法类似.
paste(c("ai", "ya", "ya"), c("1", "2"), sep="$")
[1] "ai$1" "ya$2" "ya$1"
paste(c("ai", "ya", "ya"), c("1", "2", "3"), c(F, F, T), sep="$")
[1] "ai$1$FALSE" "ya$2$FALSE" "ya$3$TRUE"

#collapse 把向量的元素连接起来.
paste(c("ai", "ya", "ya"), collapse="-")
[1] "ai-ya-ya"
# 先把多个向量对应位置上的元素连接起来，成为一个新的向量，再把这个向量各个元素连接起来.
paste(c("ai", "ya", "ya"), c("1", "2", "3"), sep="$", collapse="-")
[1] "ai$1-ya$2-ya$3"

paste0("ai", "ya", "ya")
[1] "aiyaya"
```

- 大小写转换 tolower, toupper

- 大小写:*tolower(x)*, *toupper(x)*

```{r}
tolower("The Smog Of The Sea")
[1] "the smog of the sea"
toupper("Tell Me Who You Are")
[1] "TELL ME WHO YOU ARE"
```

- strsplit, unlist 分解字符串

分解某个字符串:*strsplit(x, split, fixed=F)*, x 是字符串向量, split 表示间隔字符, 可以是""或 NULL 即将每个字符分开, fixed 表示固定模式, 表可以使用正则表达式, 不使用正则表达式的情况下设置为 T.得到一个 list. 可以使用 unlist()将列表简化成向量

Split the Elements of a Character Vector

Description

Split the elements of a character vector x into substrings according to the matches to substring `split` within them.

Usage

strsplit(x, split, fixed = FALSE, perl = FALSE, useBytes = FALSE)

Arguments

x	character vector, each element of which is to be split. Other inputs, including a factor, will give an error.
split	character vector (or object which can be coerced to such) containing regular expression(s) (unlessfixed = TRUE) to use for splitting. If empty matches occur, in particular if split has length 0, x is split into single characters. If split has length greater than 1, it is re-cycled along x.
fixed	logical. If TRUE match split exactly, otherwise use regular expressions. Has priority over perl.
perl	logical. Should Perl-compatible regexps be used?
useBytes	logical. If TRUE the matching is done byte-by-byte rather than character-by-character, and inputs with marked encodings are not converted. This is forced (with a warning) if any input is found which is marked as "bytes" (see Encoding).

```{r}
x <- c(as="asfef", "yuiop[", "stuff.blah.yech");x
```

```
   as
"asfef"      "yuiop["  "stuff.blah.yech"
```

```
#把目标中的每个元素按照分割符分解开, 得到的类型为 list, 原数据中分割符会丢失
strsplit(x, "e")
```

```
$as
[1] "asf" "f"

[[2]]
[1] "yuiop["
```

```
[[3]]
[1] "stuff.blah.y" "ch"

#把每个字母都分开
strsplit("abcd", NULL)
[[1]]
[1] "a" "b" "c" "d"

strsplit(c("abcd", "dsff"), NULL)
[[1]]
[1] "a" "b" "c" "d"

[[2]]
[1] "d" "s" "f" "f"
```

unlist 把列表转换为向量, 向量的元素类型必须相同, 所以只有当列表中的元素类型相同时, 才能使用 unlist 进行转换.

```
#把多个列表中的元素取出来放到一个向量中
strsplit(c("li lei", "han meimei"), " ")
[[1]]
[1] "li"   "lei"

[[2]]
[1] "han"      "meimei"

unlist(strsplit(c("li lei", "han meimei"), " "))
[1] "li"      "lei"     "han"     "meimei"

strsplit(c("abcd", "dsff"), NULL)
[[1]]
[1] "a" "b" "c" "d"

[[2]]
[1] "d" "s" "f" "f"

unlist(strsplit(c("abcd", "dsff"), NULL))
[1] "a" "b" "c" "d" "d" "s" "f" "f"

x <- list(a=1:3, b=c(1, 3));x
$a
[1] 1 2 3

$b
[1] 1 3
```

```
unlist(x)
a1 a2 a3 b1 b2
 1  2  3  1  3

x <- list(a=1:3, b=c("a", "b"));x
$a
[1] 1 2 3

$b
[1] "a" "b"

#会自动把数值型的向量转换为字符型的向量
unlist(x)
 a1  a2  a3  b1  b2
"1" "2" "3" "a" "b"

x <- list(a=1:3, b=c("a", "b"), c=matrix(4, 2, 2));x
$a
[1] 1 2 3

$b
[1] "a" "b"

$c
     [, 1] [, 2]
[1, ]    4    4
[2, ]    4    4

#matrix 也会转换为字符型的向量
unlist(x)
 a1  a2  a3  b1  b2  c1  c2  c3  c4
"1" "2" "3" "a" "b" "4" "4" "4" "4"

x <- list(a=1:3, b=c("a", "b"), c=data.frame(a=1:3, b=c("a", "b", "c")));x
$a
[1] 1 2 3

$b
[1] "a" "b"

$c
  a b
1 1 a
```

2 2 b
3 3 c

#数据框也转为字符型向量
unlist(x)
  a1   a2   a3   b1   b2 c.a1 c.a2 c.a3 c.b1 c.b2 c.b3
 "1" "2" "3" "a" "b" "1" "2" "3" "1" "2" "3"

```

# - substr, substring 抽取与给定位置替换

抽取或替换字符串中的字符: *substr(x, start, stop) <- value*, 关键在于指定位置的抽取和指定位置的替换

**Substrings of a Character Vector**

**Description**
Extract or replace substrings in a character vector.

**Usage**
substr(x, start, stop)
substring(text, first, last = 1000000L)
substr(x, start, stop) <- value
substring(text, first, last = 1000000L) <- value

**Arguments**

| | |
|---|---|
| x, text | a character vector. |
| start, first | integer. The first element to be replaced. |
| stop, last | integer. The last element to be replaced. |
| value | a character vector, recycled if necessary. |

```{r}
x <- c("bizarre", "love", "triangle", "we're", "meant[]")

# 对向量 x 中的每个元素从第 2 个字符开始, 提取到第 5 个字符, 实际上是把 2 和 5 都循环了 x 向量的向量长度次, 向量化运算的核心是一一对应运算
substr(x, 2, 5)
[1] "izar" "ove"  "rian" "e're" "eant"

# 向量化运算的核心是一一对应运算, 向量中第 1 个元素取[2, 1], 第 2 个元素取[2, 2], 第 3 个元素取[2, 3], 第 4 个元素取[2, 4], 第 5 个元素取[2, 1], 实际上是 2 循环了 5 次, 而 1:4 第 2 次循环中只取到了 1.
```
```

```
substr(x, 2, 1:4)
[1] ""      "o"     "ri"    "e'r"  ""

# 可以使用变量做为 start 和 stop 的值.
gap <- c(3, 2, 7, 3, 6)
substr(x, 2, gap)
[1] "iz"        "o"          "riangl"  "e'"        "eant["
```

substr 给定位置的替换

```
substr(x, 2, gap) <- "good";x
[1] "bgoarre"   "lgve"        "tgoodgle"  "wgore"       "mgood[]"

substr(x, 2, gap) <- "好玩";x
[1] "b 好玩 arre"   "l 好 ve"          "t 好玩 angle"  "w 好玩 re"       "m 好玩 nt[]"
#把原向量中抽取出的内容用"好玩"进行替换, 中文的"好玩"也作为 2 个字符来处理, 对于只抽取出 1 个字符的 love, 只替换 1 个"好"的字符. 对于抽取出超过 2 个字符的元素, 也是只替换前两个字符, 而把其余的字符保留在原向量中. 这里对于超出的元素, 没有使用默认的循环机制.
```

- Pattern Matching and Replacement

```
?sub
grep {base}
```

Description

grep, grepl, regexpr, gregexpr and regexec search for matches to argument pattern within each element of a character vector: they differ in the format of and amount of detail in the results.
sub and gsub perform replacement of the first and all matches respectively.

Usage

```
grep(pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE,
        fixed = FALSE, useBytes = FALSE, invert = FALSE)

grepl(pattern, x, ignore.case = FALSE, perl = FALSE,
         fixed = FALSE, useBytes = FALSE)

sub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
       fixed = FALSE, useBytes = FALSE)

gsub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
        fixed = FALSE, useBytes = FALSE)
```

```
regexpr(pattern, text, ignore.case = FALSE, perl = FALSE,
        fixed = FALSE, useBytes = FALSE)

gregexpr(pattern, text, ignore.case = FALSE, perl = FALSE,
         fixed = FALSE, useBytes = FALSE)

regexec(pattern, text, ignore.case = FALSE, perl = FALSE,
        fixed = FALSE, useBytes = FALSE)
```

Arguments

pattern	character string containing a regular expression (or character string forfixed = TRUE) to be matched in the given character vector. Coerced by as.character to a character string if possible. If a character vector of length 2 or more is supplied, the first element is used with a warning. Missing values are allowed except for regexpr andgregexpr.
x, text	a character vector where matches are sought, or an object which can be coerced by as.character to a character vector. Long vectors are supported.
ignore.case	if FALSE, the pattern matching is*case sensitive* and if TRUE, case is ignored during matching.
perl	logical. Should Perl-compatible regexps be used?
value	if FALSE, a vector containing the (integer) indices of the matches determined by grep is returned, and if TRUE, a vector containing the matching elements themselves is returned.
fixed	logical. If TRUE, pattern is a string to be matched as is. Overrides all conflicting arguments. fixed = TRUE: use exact matching.
useBytes	logical. If TRUE the matching is done byte-by-byte rather than character-by-character. See ‘Details’.
invert	logical. If TRUE return indices or values for elements that do *not*match.
replacement	a replacement for matched pattern in sub and gsub. Coerced to character if possible. For fixed = FALSE this can include backreferences "\1" to "\9" to parenthesized subexpressions ofpattern. For perl = TRUEonly, it can also contain "\U" or"\L" to convert the rest of the replacement to upper or lower case and "\E" to end case conversion. If a character vector of length 2 or more is supplied, the first element is used with a warning. If NA, all elements in the result corresponding to matches will be set to NA.

- chartr, sub 给定字符替换

给定字符的替换

- 替换字符，指定对象的替换:

    - *chartr(old, new, x)*，即将 x 中 old 指定的字符串替换成 new 指定的字符串，注意 old 和 new 长度相同，且字符串中每个对应的字符都要替换

    - *sub(pattern, replacement, x, ignore.case=FALSE)* 将 pattern 替换成 replacement, x 表示字符向量, ignore.case 表示是否忽略大小写, gsub()同理，只不过 sub()替换第一个匹配, gsub 替换所有匹配

?chartr

chartr {base}
Character Translation and Casefolding

Description
Translate characters in character vectors, in particular from upper to lower case or vice versa.

Usage
chartr(old, new, x)
tolower(x)
toupper(x)
casefold(x, upper = FALSE)

Arguments

x	**a character vector, or an object that can be coerced to character by as.character.**
old	a character string specifying the characters to be translated. If a character vector of length 2 or more is supplied, the first element is used with a warning.
new	a character string specifying the translations. If a character vector of length 2 or more is supplied, the first element is used with a warning.
upper	logical: translate to upper or lower case?.

```{r}
x <- c("better together", "never know", "Gone")

#也是匹配替换，把所有的"e"用"~"替换，所有的"r"都用"-"替换. 匹配和替换的长度要相等.
chartr("er", "~-", x)
[1] "b~tt~- tog~th~-" "n~v~- know"      "Gon~"
chartr("er", "*", x)
Error in chartr("er", "*", x) : 'old' is longer than 'new'

#把 x 中的"er"看成一个整体，"替换成的文字"也看成一个整体，并且只替换向量中每个元素中的第一个"er"
sub("er", "替换成的文字", x)
```

```
[1] "bett 替换成的文字 together" "nev 替换成的文字 know"      "G
one"
```

全局替换, 对 x 中的所有"er"进行替换
```
gsub("er", "替换成的文字", x)
[1] "bett 替换成的文字 togeth 替换成的文字" "nev 替换成的文字 kno
w" "Gone"
```

#ignore.case=T 忽略掉大小写
```
gsub("On", "12", c("On my way", "once again"), ignore.case=T)
[1] "12 my way"  "12ce again"
```

#在实际工作中 gsub 用的比较多, 比如淘宝或京东的评论, 删除掉其中非中文评论的部分, 只保留中文部分进行分析.
```

## - grep, grepl 字符串在字符向量中的匹配

- 单个字符串在字符向量中的匹配:*grep(pattern, x, ignore.case)*, *grepl()*, 返回匹配到的字符在向量中的位置, grepl 返回 x 中每个元素是否匹配到的逻辑值
# grep 和 grepl 只匹配, 不替换
#R 中有三种匹配方式, 数字位置索引, 逻辑索引和名称索引, 但逻辑索引用的比较多, 可以方便的取反, 并且可以进行运算. 因为 grepl 返回的是逻辑值, 所以在实际中用的比较多.

```{r}
txt <- c("arm", "foot", "lefroo", "bafoobar")
grep("foo", txt)
[1] 2 4
#返回匹配到的字符在向量中的位置, 就可以根据此位置来取子集了.
grepl("foo", txt)
[1] FALSE  TRUE FALSE  TRUE
```

- regexpr, gregexpr 正则匹配

- *regexpr(pattern, text, ignore.case)*在字符串 text 中寻找 pattern 的匹配, 并且返回每个字符中第一个匹配的位置, 类似的扩展函数 gregexpr()参数一样, 返回的是所有匹配位置的列表

grep 只能判断字符是否在元素中存在, 不返回匹配的内容在元素中的具体位置信息. regexpr 则返回具体的位置信息.

regexpr(pattern, text, ignore.case = FALSE, perl = FALSE,
        fixed = FALSE, useBytes = FALSE)

useBytes
```

logical. If TRUE the matching is done byte-by-byte rather than character-by-character. See ‘Details’.

The main effect of useBytes is to avoid errors/warnings about invalid inputs and spurious matches in multibyte locales, but for regexpr it changes the interpretation of the output. It inhibits the conversion of inputs with marked encodings, and is forced if any input is found which is marked as "bytes" see Encoding).

Caseless matching does not make much sense for bytes in a multibyte locale, and you should expect it only to work for ASCII characters if useBytes = TRUE.

```{r}
txt <- c("arm", "foot", "lefroo", "bafoobar")

#返回 txt 每个元素中 "第 1 个" 匹配的位置，没有匹配到的元素，就返回-1
regexpr("o", txt)
[1] -1  2  5  4
attr(, "match.length")
[1] -1  1  1  1
attr(, "useBytes")
[1] TRUE

#以列表格式返回元素中 "所有" 匹配的位置
gregexpr("o", c("look", "my god"))
[[1]]
[1] 2 3
attr(, "match.length")
[1] 1 1
attr(, "useBytes")
[1] TRUE

[[2]]
[1] 5
attr(, "match.length")
[1] 1
attr(, "useBytes")
[1] TRUE
```

- match, %in% 向量向量匹配

Value Matching

Description

`match` returns a vector of the positions of (first) matches of its first argument in its second.

`%in%` is a more intuitive interface as a binary operator, which returns a logical vector indicating if there

is a match or not for its left operand.

Usage

match(x, table, nomatch = NA_integer_, incomparables = NULL)

x %in% table

Arguments

x	vector or NULL: the values to be matched. Long vectorsare supported.
table	vector or NULL: the values to be matched against. Long vectors are not supported.
nomatch	the value to be returned in the case when no match is found. Note that it is coerced to integer.
incomparables	a vector of values that cannot be matched. Any value in xmatching a value in this vector is assigned the nomatchvalue. For historical reasons, FALSE is equivalent to NULL.

- *match(x, table, nomatch=NA_integer_)* 在向量 table 中寻找向量 x 的每个元素，返回第一个匹配值得位置，如果没有匹配返回 nomatch 指定的字符

```{r}
#拿 1:10 中所有元素逐个匹配 c(1, 3, 4, 9)中的元素，返回 x 中每个元素在 table 向量中第一个匹配的位置. 1 匹配，位置为 1, 2 不匹配，返回 nomatch 中默认的参数 NA, 3 匹配，返回 3 在 c(1, 3, 4, 9)中的位置 2, 4 匹配，返回位置 3, 9 匹配返回位置 4，就得到了(1 NA 2 3 NA NA NA NA 4 NA)
match(1:10, c(1, 3, 4, 9))
 [1]  1 NA  2  3 NA NA NA NA  4 NA
match(1:10, c(1, 3, 4, 9), nomatch=0)
 [1] 1 0 2 3 0 0 0 0 4 0

#返回逻辑值，如果匹配返回 TRUE，不匹配则返回 FALSE
match(1:10, c(1, 3, 4, 9), nomatch=0)>0
 [1]  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE
```

- *%in%*相当于*"%in%" <- function(x, table) match(x, table, nomatch=0)>0*，返回与左边向量长度相同的逻辑向量，逻辑向量中是值是符号左边向量中每个元素是否在右边向量中的逻辑值. 因为 %in% 返回的不是 NA，所以在 if 判断中经常使用到. That %in% never returns NA makes it particularly useful in if conditions.

```{r}
1:10 %in% c(1, 3, 4, 9)
```

```
 [1]  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE
FALSE

str <- c("a", "ab", "B", "bla", NA, "%", "Z")
x <- str %in% c(letters, LETTERS);x
[1]  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE
str[x]
[1] "a" "B" "Z"
```

%in% 在实际中的应用

某一个数据集中有一列 city, 值为 北京, 上海, 杭州, 广州, 武汉, 深圳

从 city 列表中抽取一级城市, 即北上广深.

```
city <- c("北京", "上海", "杭州", "广州", "武汉", "深圳")
fcity <- c("北京", "上海", "广州", "深圳")

city %in% fcity
[1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE
# 把匹配到的城市抽取出来
city[city %in% fcity]
[1] "北京" "上海" "广州" "深圳"
```

##Practices

- 将多个字符: "我", "爱", "你" 粘贴转换成 "我!爱!你"; 将字符串 c("我", "爱", "你") 转换成 "我&1-爱&2-你&3"

```
paste("我", "爱", "你", sep="!")
[1] "我!爱!你"
paste(c("我", "爱", "你"), 1:3, sep="&", collapse="-")
[1] "我&1-爱&2-你&3"
```

- 对于名字向量: c("jiang zhu", "san zhang", "si li", "wang lao") (先名后姓)
    - 对于原始向量, 将所有的小写变成大写

```
x <- c("jiang zhu", "san zhang", "si li", "wang lao")
toupper(x)
[1] "JIANG ZHU" "SAN ZHANG" "SI LI"     "WANG LAO"
```

    - 对于原始向量, 将名和姓分开, 将所有字符转化为一个字符向量

```
unlist(strsplit(x, " "))
[1] "jiang" "zhu"   "san"   "zhang" "si"    "li"    "wang"
"lao"
```

- 对于原始向量, 将所有的空格变成 &&

```
gsub(" ", "&&", x)
[1] "jiang&&zhu" "san&&zhang" "si&&li"     "wang&&lao"
```

- 对于原始向量, 找到每个字符中第一个空格的位置, 返回一个数值向量

```
gap <- regexpr(" ", x);gap
[1] 6 4 3 5
attr(, "match.length")
[1] 1 1 1 1
attr(, "useBytes")
[1] TRUE
```

- 根据上一题得到的数值向量, 只将每个名字的姓抽取出来

```
substr(x, gap+1, 20)
[1] "zhu"   "zhang" "li"    "lao"
```

- 对于字符串*c("我爱你", "我是朝阳群众", "吃瓜群众", "我爱京瘫")*
    - 查找字符串中包含"我"的字符的位置

```
x <- c("我爱你", "我是朝阳群众", "吃瓜群众", "我爱京瘫")
grep("我", x)
[1] 1 2 4
```

- 返回一个逻辑向量指明每个字符是否含有字符:"我"

```
grepl("我", x)
[1]  TRUE  TRUE FALSE  TRUE
```

- 对于字符向量*c("北京大学", “北京清华”, “上海师范”, "上海复旦", "山西大学")*, 返回一个逻辑向量表明在北京和上海的大学

```
x <- c("北京大学", "北京清华", "上海师范", "上海复旦", "山西大学
")
substr(x, 1, 2) %in% c("北京", "上海")
[1]  TRUE  TRUE  TRUE  TRUE FALSE
```

选学: data.table 包

#data.table

-继承自 data.frame

-所有作用于 data.frame 上的函数作用于 data.table. data.frame 可以直接替换多列, 如对 100 列的 data.frame, 提取出其中的 10 列, 统一进行某种操作. 但 data.table 只能一列一列的替换. data.table 与 data.frame 取子集的方式也不同.

-由 C 写成, 比在数据框上的操作快很多, 例如在取子集, 分组, 更新变量操作中

-需要学习一些新的语法

```{r}
library(data.table, quietly = T)
DF <- data.frame(x=rnorm(9), y=rep(c("a", "b", "c"), each=3), z=rnorm(9))
head(DF, 2)
##                    x y                   z
## 1 -0.4115108 a -0.05710677
## 2    0.2522234 a    0.50360797
DT <- data.table(x=rnorm(9), y=rep(c("a", "b", "c"), each=3), z=rnorm(9))
head(DT, 2)
##                     x y                 z
## 1: -0.6494716 a -0.4527840
## 2:    0.7267507 a -0.8320433
```

##查看内存中数据, 子集

```{r}
tables()
##          NAME NROW NCOL MB COLS    KEY
## [1, ] DT          9       3    1 x, y, z
## Total: 1MB
```

```{r}
DT[2, ]
##                    x y                 z
## 1: 0.7267507 a -0.8320433
DT[DT$y=="a", ]
##                     x y                 z
## 1: -0.6494716 a -0.4527840
## 2:    0.7267507 a -0.8320433
## 3:    1.1519118 a -1.1665705
```

##取子集的区别

-没有逗号只有数值向量时，按行取子集
-逗号后面即可以是索引也可以是表达式(旧版本只能是表达式)

```{r}
DT[c(2, 3)]
##                    x y                z
## 1: 0.7267507 a -0.8320433
## 2: 1.1519118 a -1.1665705
DT[, c(2, 3)]
##        y                z
## 1: a -0.4527840
## 2: a -0.8320433
## 3: a -1.1665705
## 4: b -1.0655906
## 5: b -1.5637821
## 6: b    1.1565370
## 7: c    0.8320471
## 8: c -0.2273287
## 9: c    0.2661374
```

##表达式计算列

-逗号后面的变量叫做表达式，来对数据进行各种不同方式的总结和筛选
-表达式是花括号括起来的一系列函数和命令的集合
-表达式返回的值是表达式最后一条命令的结果

```{r}
{     x=1
      y=2}
k={print(10);5}
## [1] 10
print(k)
## [1] 5
DT[, list(mean(x), sum(z))]
##                   V1              V2
## 1: 0.5632717 -3.053378
DT[, table(y)]
## y
```

```
## a b c
## 3 3 3
```

##创建新列

-用:=定义并赋值一个新变量
-如果我们在数据框中加入新变量, R 会把内存中原数据框整个复制一遍, 再在新的副本里添加新变量, 这样内存中就有了两个重复的数据框
-data.table 不会产生副本

```{r}
DT[, w:=z^2];DT
##                    x y                z                w
## 1: -0.6494716 a -0.4527840 0.20501333
## 2:    0.7267507 a -0.8320433 0.69229605
## 3:    1.1519118 a -1.1665705 1.36088684
## 4:    0.9921604 b -1.0655906 1.13548329
## 5: -0.4295131 b -1.5637821 2.44541430
## 6:    1.2383041 b    1.1565370 1.33757783
## 7: -0.2793463 c    0.8320471 0.69230242
## 8:    1.7579031 c -0.2273287 0.05167833
## 9:    0.5607461 c    0.2661374 0.07082910
```

##创建新列

```{r}
DT2 <- DT
DT[, y:=2]
head(DT, 3)
##                    x y                z              w
## 1: -0.6494716 2 -0.4527840 0.2050133
## 2:    0.7267507 2 -0.8320433 0.6922960
## 3:    1.1519118 2 -1.1665705 1.3608868
head(DT2, 3)
##                    x y                z              w
```

```
## 1: -0.6494716 2 -0.4527840 0.2050133
## 2:    0.7267507 2 -0.8320433 0.6922960
## 3:    1.1519118 2 -1.1665705 1.3608868
```

-此处 y 列从字符转化成数值会出警告，可用 DT[, , y:=2]
-如果想在内存中创建新的副本，需要使用 copy()函数

##执行多个函数创建新列

```{r}
DT[, m:={tmp <- x+z;log2(tmp+5)}];DT[1:5, ]
##                    x y                z              w              m
## 1: -0.6494716 2 -0.4527840 0.2050133 1.962639
## 2:    0.7267507 2 -0.8320433 0.6922960 2.291223
## 3:    1.1519118 2 -1.1665705 1.3608868 2.317692
## 4:    0.9921604 2 -1.0655906 1.1354833 2.300583
## 5: -0.4295131 2 -1.5637821 2.4454143 1.588183
```

```{r}
DT[, a:=x>0];DT[1:5, ]
##                    x y                z              w              m        a
## 1: -0.6494716 2 -0.4527840 0.2050133 1.962639 FALSE
## 2:    0.7267507 2 -0.8320433 0.6922960 2.291223    TRUE
## 3:    1.1519118 2 -1.1665705 1.3608868 2.317692    TRUE
## 4:    0.9921604 2 -1.0655906 1.1354833 2.300583    TRUE
## 5: -0.4295131 2 -1.5637821 2.4454143 1.588183 FALSE
```

##类 plyr 包命令

-在数据表中使用类似于 plyr 包的命令，以下执行分组汇总

```{r}
DT[, b:=mean(x+w), by=a];DT
##                    x y                z              w              m        a              b
## 1: -0.6494716 2 -0.4527840 0.20501333 1.962639 FALSE 0.6614663
## 2:    0.7267507 2 -0.8320433 0.69229605 2.291223    TRUE 1.8460879
## 3:    1.1519118 2 -1.1665705 1.36088684 2.317692    TRUE 1.8460879
```

```
## 4:    0.9921604 2 -1.0655906 1.13548329 2.300583    TRUE 1.8460879
## 5: -0.4295131 2 -1.5637821 2.44541430 1.588183 FALSE 0.6614663
## 6:    1.2383041 2    1.1565370 1.33757783 2.886519    TRUE 1.8460879
## 7: -0.2793463 2    0.8320471 0.69230242 2.473190 FALSE 0.6614663
## 8:    1.7579031 2 -0.2273287 0.05167833 2.707210    TRUE 1.8460879
## 9:    0.5607461 2    0.2661374 0.07082910 2.542724    TRUE 1.8460879
```

##特殊变量

-.N 是一个长度为 1 的整数, 类似于 count(), 表示某一组值在变量中出现的次数
-比起 table(DT$x)快很多

```{r}
set.seed(123)
DT <- data.table(x=sample(letters[1:3], 1E5, T))
DT[, .N, by=x]
##       x         N
## 1: a 33387
## 2: c 33201
## 3: b 33412
```

##键 Keys

```{r}
DT <- data.table(x=rep(letters[1:3], times=100), y=rnorm(300))
setkey(DT, x)
DT["a"][1:10]
##        x                y
##    1: a    0.2595897
##    2: a -0.8082840
##    3: a -2.3795501
##    4: a    0.8609006
##    5: a -0.3698375
##    6: a    0.3783155
##    7: a    0.3943500
##    8: a    2.5583766
```

```
##    9: a    1.2151264
## 10: a    1.1324966
```

-设置键以后会自动排序，取子集的速度也会大大增加

##合并

```{r}
DT1 <- data.table(x=c('a', 'a', 'b', 'dt1'), y=1:4)
DT2 <- data.table(x=c("a", "b", "dt2"), z=5:7)
setkey(DT1, x);setkey(DT2, x)
merge(DT1, DT2)
##        x y z
## 1: a 1 5
## 2: a 2 5
## 3: b 3 6
```

-两表的键名必须相同，很快就可以合并

##快速读取

```{r}
big_df <- data.frame(x=rnorm(1E6), y=rnorm(1E6))
file <- tempfile()
write.table(big_df, file=file, row.names=F, col.names = T, sep="\t", quote=F)
system.time(fread(file))
##
#Read 30.0% of 1000000 rows
#Read 51.0% of 1000000 rows
#Read 67.0% of 1000000 rows
#Read 88.0% of 1000000 rows
#Read 1000000 rows and 2 (of 2) columns from 0.035 GB file in 00:00:06
##        user    system elapsed
##        5.65        0.05        5.93
system.time(read.table(file, header = T, sep="\t"))
##        user    system elapsed
##      19.56        0.27      20.12
```

```

## ##资源

- 最新版本的 data.table 的说明
https://r-forge.r-project.org/scm/viewvc.php/pkg/NEWS?view=markup&root=datatable
- data.table 和 data.frame 的区别
http://stackoverflow.com/questions/13618488/what-you-can-do-with-data-frame-that-you-cant-in-data-table

-目前最快读取数据的包是 fst 包，查看 http://www.fstpackage.org/

| Method | Format | Time (ms) | Size (MB) | Speed (MB/s) | N |
|---|---|---|---|---|---|
| **readRDS** | bin | 1577 | 1000 | 633 | 112 |
| **saveRDS** | bin | 2042 | 1000 | 489 | 112 |
| **fread** | csv | 2925 | 1038 | 410 | 232 |
| **fwrite** | csv | 2790 | 1038 | 358 | 241 |
| **read_feather** | bin | 3950 | 813 | 253 | 112 |
| **write_feather** | bin | 1820 | 813 | 549 | 112 |
| **read_fst** | bin | 457 | 303 | 2184 | 282 |
| **write_fst** | bin | 314 | 303 | 3180 | 291 |
```