R

陈华珊

社会发展战略研究院

介绍

◆ R 是什么？

- R 是一种高级计算机语言，用于数据分析及绘图
  - 进行各种简单及高级的统计建模
  - 生成高质量的图形
  - 作为一种计算机语言，可以进行任意扩展
- R 的作者是新西兰奥克兰大学统计学系 **R**oss Ihaka 与 **R**obert Gentleman
- R 是开源软件，核心代码由“R Core Team”负责维护
- R 的官方网站是

  `http://www.R-project.org`

  镜像网站

  `http://cran.R-project.org`

◆ 优势

- R 语言的高效
  - 面向对象的程序语言 vs. 面向过程的语言（Stata、SPSS）

    Computer time is inexpensive in comparison with personnel time.
  - 编译解释代码（`library(compile)`）
  - 直接调用 C 语言进行优化（`Rcpp`）
- 可嵌入性
  - `rJava`
  - `rPython`、`rpy2`
  - `shiny`
- 高质量的图形输出（`ggplot2`）
- 完整的工作流

- 数据导入
- 分析
- 制表
- 可视化
- 研究报告

- 面对大数据的计算能力
  - 连接 hadoop、mongodb等各类数据库
  - 多核多线程、并行计算
  - GPU计算

◆ 社会学研究与 R 统计包

❖ 线性及广义线性模型

- 线性回归模型：`car`（Fox, *An R and S-PLUS Companion to Applied Regression*, Sage, 2002）；
- F 及 wald 检验：`lmtest`；
- 多重比较：`multcomp`；
- 假设检验：`pscl`：非嵌套模型的假设检验（`vuong()`）；`rms`：线性及广义线性模型假设检验；

❖ 定类及计数数据

- Binomial logit 、probit模型、Poisson回归、loglinear模型：`MASS（glm()）`;
- multinomial logit model：`nnet`, `multinomRob`;
- ordered logit and probit model：`MASS（polr()）`;
- multinomial probit model：`MNP`,
- 流动表模型：`gnm`;
- 列联表数据的图形展示：`vcd`;

❖ 其它回归模型

- 混合效应模型：`nlme`；
- 事件史模型：`survival`(Therneau and Grambsch, *Modeling Survival Data*, Springer, 2000)、`eha`、`survrec`、`frailtypack`、`rms`；
- 结构方程模型：`sem`, `polycor`, `pls`；
- 广义潜变量模型：`lavaan`,`MplusAutomation`；

❖ 其它模型

- 网络分析：`sna`, `network`, `igraph`；`latentnet`, `statnet`（`ergm`, `hergm`, `tergm`）；
- 因果推断：`Matching`、`MatchIt`、`optmatch`；
- 空间分析：`spatial`、`gstat`; `sp`、`rgdal`、`maptools`；
- Bootstrapping：`boot`（Davison and Hinkley, *Bootstrap Methods and Their Application* Cambridge, 1997）、`bootstrap`（Efron and Tibshirani, *An Introduction to the Bootstrap* Chapman and Hall, 1993）

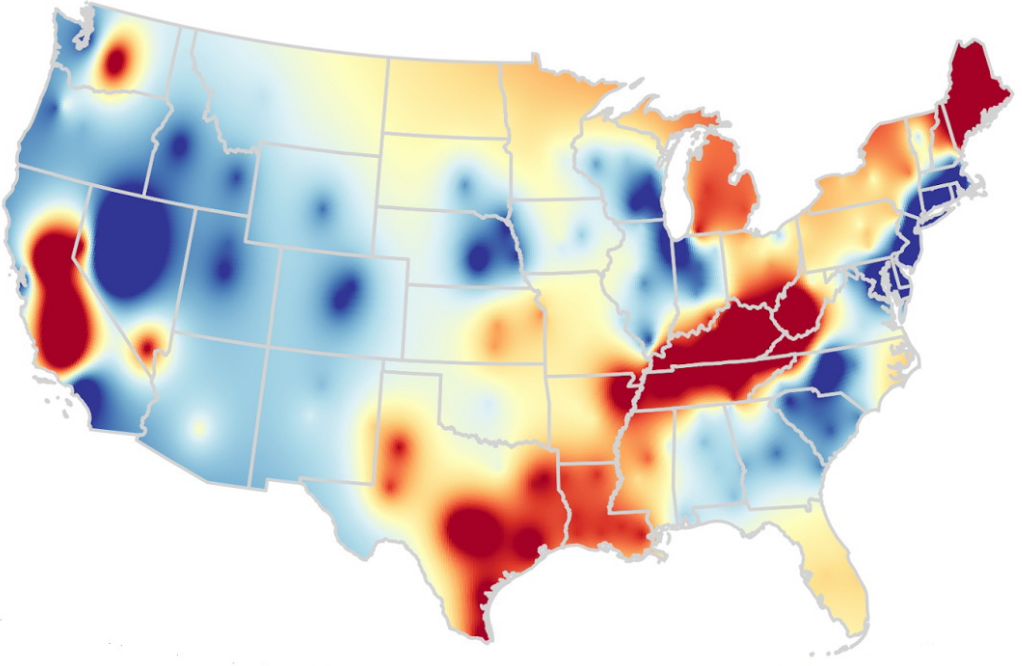
❖ 抽样及抽样调查数据分析

- 抽样设计：`sampling`、`pps`、`simFrame`、`SamplingStrata`；
- 分析：`survey`、`reweight`
- 缺失值处理：`mix`, `pan`(Shafer, Analysis of Incomplete Multivariate Data *Chapman and Hall, 1997*)、`mi`、`MImix`、`mitools`；

◆ 大数据及高性能计算

- Hadoop: `rmr`、`RHIPE`;
- 大型数据: `biglm`、`bigmemory`、`ff`;
- 并行计算: `snow`、`parallel`、`foreach`; `Rmp`、`pdbMPI`;
- GPU: `gputools`、`rgpu`;

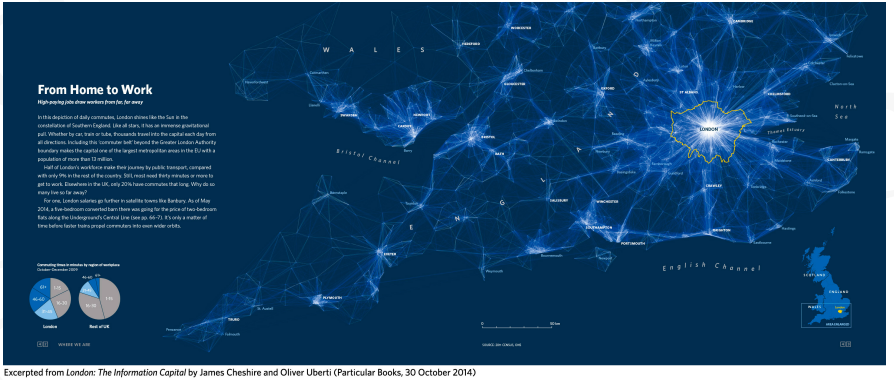
◆ 示例



美国家庭交通消费支出（2012）来源：*consumer expenditure survey*

undefined

图片来源：*Paul Butler, Facebook*



Excerpted from *London: The Information Capital* by James Cheshire and Oliver Uberti (Particular Books, 30 October 2014)

http://theinformationcapital.com/

❖ 可交互图形

file:///e:/Huashan/Stat/R/Lectures/北大新传/demo/clickme/data-force_directed.html

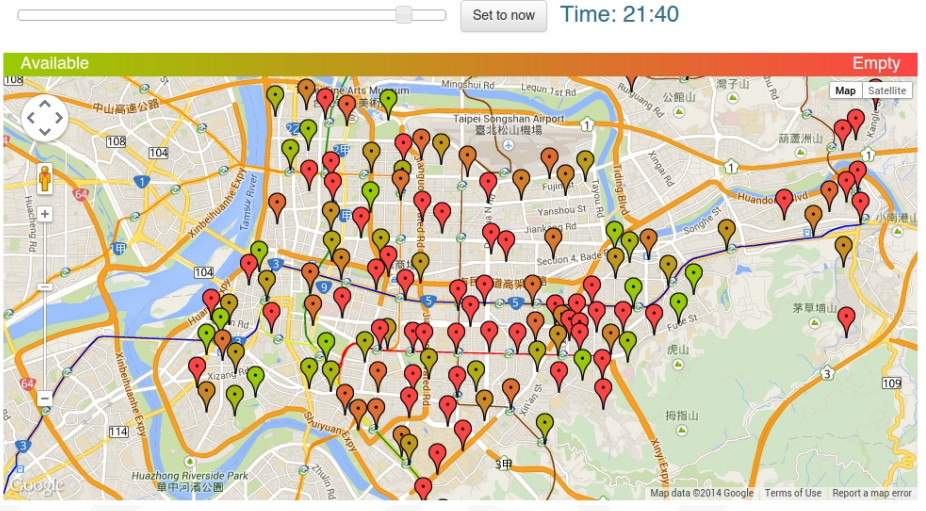
file:///e:/Huashan/Stat/R/Lectures/北大新传/demo/radial_network.html

file:///E:/Huashan/Stat/R/Lectures/%E5%8C%97%E5%A4%A7%E6%96%B0%E4%BC%A0/dem

- ❖ Shiny
- ❖ 股市数据信息
- 数据来自新浪或雅虎财经

❖ 台北公共自行车用量预测

- 项目目标:
  - 预测给定自行车站任意时间的自行车可用量;
  - 对一段时期内可用量的趋势做可视化的呈现;
- 网站: `https://yaoch29.shinyapps.io/taipei-city-bike-prediction`
- 源代码: `https://github.com/hsu-yc/taipei-city-bike-prediction`



安装及学习资源

◆ 下载及安装

❖ 安装 R

- R 主页 https://www.r-project.org/

- CRAN （The Comprehensive R Archive Network）`https://cran.r-project.org/`
- R 系统可以安装在绝大多数操作系统上，Windows、Mac、Linux 等等；

- 32 bit vs. 64 bit:
  - 32位的操作系统能支持的最大内存为 $2^{32}/1024^3 = 4G$;
  - 32位的 R 最大使用内存为3G。由于 Windows 的内存管理机制，真正 R 能使用的内存为2G，在 linux 下，R 能使用3G内存。
  - 64位的操作系统能支持的最大内存为 $2^{64}/1024^3 = 172 \times 10^8 G$;
  - 32位的 R 在64位系统下只能使用4G内存;
  - 由于 R 缺少64位的整型数据结构，能支持的最大元素数量为 2^32。

❖ R 控制台

- R 采用控制台（console）的方式进行人机互动；
- 在控制台中输入表达式，由 R 系统进行求值，输出计算结果或者图形；逐条输入表达式，并重复上述过程；
- 使用上下键，可以调出历史命令；
- 每个 R 会话（session）就是一个独立的交互进程；

❖ 安装 R 包

- 数量众多的 R packages，>5000
- Task Views
- 安装 R 包使用命令

```
install.packages("ggplot2")
```

❖ 安装 R Studio

- RStudio 是 R 的整合开发环境 (IDE)
- http://www.rstudio.com/

◆ 文档及帮助

❖ 文档

- 官方文档: `http://cran.r-project.org/manuals.html`
- R Journal: `http://journal.r-project.org/`

❖ 帮助

- 获取帮助文档
  - 在 `console` 输入`?function_name` 即可调出关于该函数的帮助文档;
  - 在 `RStudio` 中按 `F1` 调出在线帮助;
  - 如果不知道用哪个函数,

    `help.search("regression")`
- 很多R包的文档包含例子:

  `example(lm)`
- 或者

  `demo("topic")`
- 有些包额外附带了简介或梗概:

  `vignette()`

❖ 其他资源

- R-Bloggers: `http://r-bloggers.com/`
- 讨论组: `http://stackoverflow.com/`
- 邮件列表 R-help: `https://stat.ethz.ch/mailman/listinfo/r-help`

◆ 如何寻求帮助

- 可再现的例子
  - 最精简的，可重复的数据
  - 最精简的可运行的代码
  - 减少第三方依赖
  - 运行环境的描述：操作系统、R、R包的版本
- 描述清楚数据输入的结构，期待的结果，处理的逻辑
- 更多阅读 `http://www.r-bloggers.com/three-tips-for-posting-good-questions-to-r-help-and-stack-overflow/`

R 语言基础

◆ 一个示例

以中国综合社会调查（CGSS）为例

- 下载网址 `http://www.cnsda.org/index.php?r=projects/view&id=93281139`

```
1  library(haven)
2  gss = read_dta('CGSS2013（居民问卷）发布版.dta')
3  dim(gss)
4  View(gss)
5  names(gss)
6
7  lbl = sapply(gss, attr, 'label')
8  Encoding(lbl) = 'GB2312'
9  gss_lbl = data.frame(var = names(gss), lbl = lbl)
```

◆ 赋值及计算

❖ 赋值

在控制台窗口（console）中输入

```
x<-42+8
x
## [1] 50
x = 42+8
x
## [1] 50

# `()` 的作用
(x<-42+8)
## [1] 50

# 多重赋值
# 将 6 同时赋给对象 a 和 b
```

```
14  b<-a<-6
15  a
16  ## [1] 6
17  b
18  ## [1] 6
19
20  # 如果忘了给表达式赋值
21  42 + 8
22  ## [1] 50
23  (x = .Last.value)
24  ## [1] "knitr"     "stats"     "graphics"  "grDevices" "utils"
    "datasets"
25  ## [7] "base"
```

❖ 数学运算符

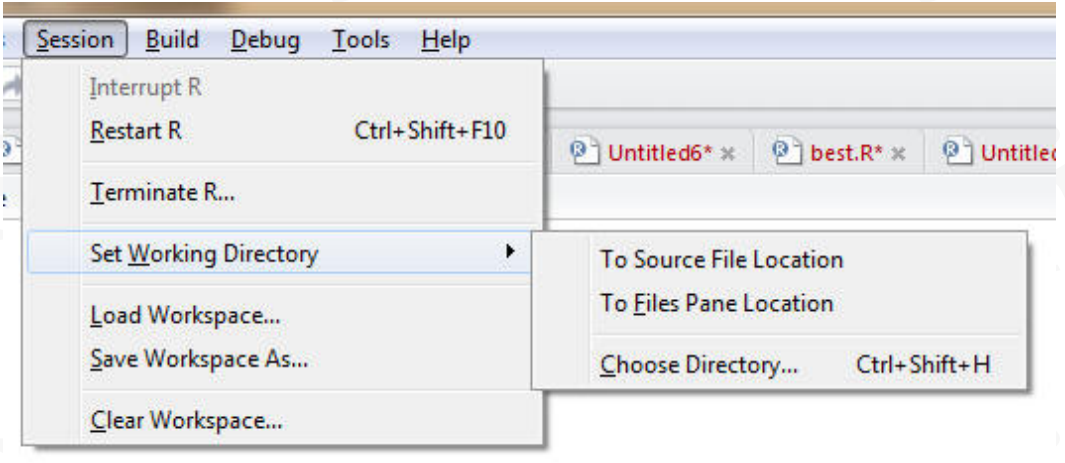
```
1 + 2 + 3
## [1] 6
1 + 2 * 3
## [1] 7
(1 + 2) * 3
## [1] 9
```

❖ 运行已有的 R 脚本命令

```
1  # `instruct.r` 是当前工作目录下的文本文件，包含一些 `R` 指令集。`source()` 函数调入该文件并执行指令。
2  source("instruct.R")
```

◆ 工作路径

- 当读写数据、载入脚本、保存图形时，通常需要指定文件路径，为了减少录入，可以设置 R 的工作路径;
- `setwd(),getwd()`
- `RStudio`



◆ 工作空间

- 在每个R会话中，所有创建的对象均保存在工作空间（workspace）；
- 每次退出 R 均会提示“Save workspace image?”
- 保存当前工作空间

```
save.image(file="")
```

- 其它相关函数：
  - `ls()`
  - `rm()`
  - `rm(list=ls()`

◆ 对象

❖ 命名规则

❖ 对象命名规则

- 所有的东西都是对象（“everything is an object”）。这一点和传统统计软件的脚本语言有着根本的不同。对象就是 R 语言的基本元素。
- 起始字符必须是罗马字母，其他可以使用的有数字0-9，“.”。但不能使用下划线(_)以及保留字符，如：

```
break for function if in next repeat  return  while
```

- 对象名称区分大小写
- 避免使用系统对象名字，如：

```
c t C F(alse) T(rue) diff range tree var data
```

❖ 对象列表

向量（vector）	一维，同类型元素
列表（list）	一维，不同类型元素
矩阵（matrix）	二维，同类型元素
数组（array）	二维或多维，同类型元素
数据框（data.frame）	二维，每列元素同类型

❖ 向量

❖ 创建

向量可以通过多种方式来创建，其中最常使用的是c()函数，该函数用来将几个参数数值合并，比如:

```
x<-c(1,2,3,4)
x
## [1] 1 2 3 4

# 我们用 c() 创建了一个向量x，包含 1，2，3，4 四个值。[1]表示开始的第一个数值是向量 x 的第一个元素
y<-c(6,7,8)
# 再建一个y向量，包含 6，7，8
c(y,x,y)
##  [1] 6 7 8 1 2 3 4 6 7 8
```

❖ 创建

向量和矩阵都可以包括数值、字符和逻辑值 (用TRUE, FALSE表示)

```
1  mydata<-c(2.9,3.4,3.4,3,7,3,7,2,8,2,5,2,4,2.4)
2  mydata > 3
3  ##  [1] FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
   FALSE  TRUE
4  ## [12] FALSE  TRUE FALSE
```

❖ 创建字符型向量

向量还可以包含字符。字符必须用单引号或双引号标明。创建字符串向量也用c()函数。

```
x<-c("a","b")
x
## [1] "a" "b"
```

如果c()函数中既包含数值参数又包含字符串参数，R会自动将所有的元素变成字符串。

```
x<-c(20,30, "a")
x
## [1] "20" "30" "a"
```

❖ 筛选过滤

```
a <- 1:10
a[4]
## [1] 4
a[a>6]
## [1]  7  8  9 10
head(a)
## [1] 1 2 3 4 5 6
head(a, 3)
## [1] 1 2 3
tail(a)
## [1]  5  6  7  8  9 10
```

❖ 筛选过滤

which() 函数

```
a = sample(10)
which(a > 6)
## [1]  1  4  8 10
a[which(a > 6)]
## [1]  8  7 10  9
which.min(a)
## [1] 6
which.max(a)
## [1] 8
```

❖ 匹配

```
a1 = c(4, 11)
a1 %in% a
## [1]  TRUE FALSE
a[a1 %in% a]
## [1] 8 4 3 6 5
match(a1, a)
## [1]  3 NA
```

❖ 排序

```
b <- sample(1:10)
b[order(b)]
##  [1]  1  2  3  4  5  6  7  8  9 10
# [1] 1 2 3 4 5 6 7 8 9 10
```

❖ 操作符与函数

```
`+ - * /`
```

乘法操作符

```
c(1,2,3) %*% c(3,2,1) # 内乘法
##      [,1]
## [1,]   10
c(1,2,3) %o% c(3,2,1) # 外乘法
##      [,1] [,2] [,3]
## [1,]    3    2    1
## [2,]    6    4    2
## [3,]    9    6    3
```

默认情况下, 所创建的向量为行向量。用转置函数 `t()` 可创建列向量

```
t(1:4)
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
```

❖ 命名

为向量中的某个元素命名，以后可以通过引用该名称进行操作。用c()创建带名称的向量:

```
x<-c(a=1, b=2)
x
## a b
## 1 2

# 创建了向量x，包含2个元素，第一个元素命名为"a"，第二个元素命名为"b"
x["a"]
## a
## 1
# 不需要给向量中的每个元素都命名，只将第一个元素命名为"a"
x<-c(a=1,2)
```

❖ 属性

所有的对象都有属性

```
names(x)
## [1] "a" ""
attributes(x)
## $names
## [1] "a" ""
attr(x, 'names')
## [1] "a" ""
(attr(x, 'mm') = 9)
## [1] 9
```

修改向量对象, 大多数属性丢失

```
attributes(x[1])
## $names
## [1] "a"
```

能够得到保存的三个基本属性:

- 名称 `names()`
- 维度 `dim()`
- 类名 `class()`

❖ 数组与矩阵

❖ 创建

使用 dim()` 函数可以把一个向量转化成矩阵或数组

```
mydata<-c(2.9,3.4,3.4,3,7,3,7,2,8,2,5,2,4,2.4)
dim(mydata)<-c(2,7)
mydata
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]  2.9  3.4    7    7    8    5  4.0
## [2,]  3.4  3.0    3    2    2    2  2.4
dim(mydata)<- NULL
# 这个赋值语句删除了mydata对象的dim属性，将其复原为向量。注意NULL必须为大写。
mydata
##  [1] 2.9 3.4 3.4 3.0 7.0 3.0 7.0 2.0 8.0 2.0 5.0 2.0 4.0 2.4
```

❖ 创建

通过matrix()函数可以把一个或几个向量合并，产生一个新的矩阵对象。

```
x<-c(1,2,3,4,5)
X<-c(11,12,13,14,15)
M<-matrix(c(x,X),ncol=2)
# matrix函数需要两个参数：1）向量参数；2）ncol参数，指定这个矩阵的列数
M
##      [,1] [,2]
## [1,]    1   11
## [2,]    2   12
## [3,]    3   13
## [4,]    4   14
## [5,]    5   15
```

❖ 创建

如果向量参数的元素个数（长度）不是ncol参数的整数倍，R会用循环的方式按该向量参数将向量长度补齐。比如

```
matrix(c(1,2,3,4,5), ncol=3)
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    1
matrix(c(1,2,3,4), ncol=3)
##      [,1] [,2] [,3]
## [1,]    1    3    1
## [2,]    2    4    2
```

❖ 矩阵运算

矩阵的运算 `+ - * /` 都适用。但是矩阵的乘法使用 **"%*%"** 运算符，矩阵与实数或向量相乘使用 **"*"** 运算符

矩阵的转置使用 `t()` 函数

❖ 筛选及过滤

[i,]、[,j]、[i,j]分别用来获得矩阵的第i行的所有元素、第j列的所有元素、第i行j列元素。比如有个$5x5$的Y矩阵:

```
Y = matrix(1:15, ncol=5)
Y
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    2    5    8   11   14
## [3,]    3    6    9   12   15
Y[,1]
## [1] 1 2 3
Y[1,]
## [1]  1  4  7 10 13
Y[2,1]
## [1] 2

Y[, c(1,2)]
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
Y[c(1,3),]
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    3    6    9   12   15
# 用 `i:j` 来指定元素的范围。比如
Y[, 1:3]
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

❖ 筛选及过滤

在范围前加上一个负号 (-)，表示剔除该范围的元素。比如

```
Y[-1,]
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2    5    8   11   14
## [2,]    3    6    9   12   15

Y[-2, -1]
##      [,1] [,2] [,3] [,4]
## [1,]    4    7   10   13
## [2,]    6    9   12   15
```

❖ 条件筛选

除了直接指定范围外，还可以使用逻辑判断符 “>”、“<”、“>=”、“<=”、“==”

```
Y > 5
##       [,1]  [,2] [,3] [,4] [,5]
## [1,] FALSE FALSE TRUE TRUE TRUE
## [2,] FALSE FALSE TRUE TRUE TRUE
## [3,] FALSE  TRUE TRUE TRUE TRUE
Y[Y > 5]
##  [1]  6  7  8  9 10 11 12 13 14 15
# 请注意以上两个语句的不同。y>5返回的是逻辑值T（真）或F（假），而y[y>5]才是列出y向量中大于5的元素
```

❖ 矩阵合并

两个函数rbind()、cbind()提供了矩阵的合并功能。rbind()对两个列数相同的矩阵按行合并，cbind()对两个行数相同的矩阵按列合并。

```
x<-matrix(1:10,ncol=2)
# x 是 5x2 的矩阵
y<-matrix(1:20,ncol=4)
# y 是 5x4 的矩阵
z<-cbind(x,y)
z
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    6    1    6   11   16
## [2,]    2    7    2    7   12   17
## [3,]    3    8    3    8   13   18
## [4,]    4    9    4    9   14   19
## [5,]    5   10    5   10   15   20
rbind(x,y[,1:2])
##      [,1] [,2]
```

```
15    ##  [1,]    1    6
16    ##  [2,]    2    7
17    ##  [3,]    3    8
18    ##  [4,]    4    9
19    ##  [5,]    5   10
20    ##  [6,]    1    6
21    ##  [7,]    2    7
22    ##  [8,]    3    8
23    ##  [9,]    4    9
24    ## [10,]    5   10
```

❖ 数组(ARRAY)

在R里，数组可以被看做是一个多维的矩阵（最多可以达到8维）。建立一个数组使用：

```
x<-array(1:24,c(3, 4, 2))
# 建立一个三维的数组对象。显示的时候，R是从最高维按顺序至最低维，在每一步显示一个二维的矩阵。
x
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
```

```
## [1,]   13   16   19   22
## [2,]   14   17   20   23
## [3,]   15   18   21   24
x[,2,]
##      [,1] [,2]
## [1,]    4   16
## [2,]    5   17
## [3,]    6   18
```

❖ 列表

❖

列表 (list) 表示对象的集合，它可以同时包含多个对象，同时每个对象可分别属于不同类型，比如数值类型向量和字符串类型向量。列表中的对象可命名。

```
alist<-list(c(0,1,2),1:10)
alist
## [[1]]
## [1] 0 1 2
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10
alist[[1]]
## [1] 0 1 2
alist[1]
## [[1]]
## [1] 0 1 2
```

```
13
14    blist<-list(c("a","b","c"),1:5,"name")
15    blist[[3]]
16    ## [1] "name"
```

❖

可以对列表中的对象命名，并可通过名称来引用一个对象。对集合中的对象名称的引用通过 “list$name” 的形式进行。例如：

```
clist<-list(x=matrix(1:10,ncol=2),y=c("a","b"),z=blist)
clist$x
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    3    8
## [4,]    4    9
## [5,]    5   10
clist$x[1:3,1:2]
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    3    8
clist$z[[2]]
```

```
15    ## [1] 1 2 3 4 5
```

❖ 数据集

数据集（data frame）是一种特殊的列表，但在形式上和矩阵类似。但与矩阵不同的是，数据集可以同时包含数值和字符型对象

```
df<-data.frame(V1=1:10, V2=6:15, V3=2:11)
df
##    V1 V2 V3
## 1   1  6  2
## 2   2  7  3
## 3   3  8  4
## 4   4  9  5
## 5   5 10  6
## 6   6 11  7
## 7   7 12  8
## 8   8 13  9
## 9   9 14 10
## 10 10 15 11
# 比较以下几种筛选方式的结果
```

```
df$V1
##  [1]  1  2  3  4  5  6  7  8  9 10

df['V1']
##    V1
## 1   1
## 2   2
## 3   3
## 4   4
## 5   5
## 6   6
## 7   7
## 8   8
## 9   9
## 10 10

df[, 'V1']
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
33
34  df[1]
35  ##    V1
36  ## 1   1
37  ## 2   2
38  ## 3   3
39  ## 4   4
40  ## 5   5
41  ## 6   6
42  ## 7   7
43  ## 8   8
44  ## 9   9
45  ## 10 10
46
47  df[, 1]
48  ##  [1]  1  2  3  4  5  6  7  8  9 10
49  # 如果变量名含有空格或特殊字符，则使用成对的 ``backticks (`)`` 符号
50  names(df)<-c('v1', 'na me', 'v3')
```

```
df$`na me`
## [1]  6  7  8  9 10 11 12 13 14 15
```

◆ 数据类型

❖ 数据类型

整数（integer）	`1:10`
实数（numeric, double）	`c(1.1, 3.14, 10)`
复数（complex）	`c(1+i, 3-2i)`
字符（character）	`c("a", "b", "COS")`
因子（factor）	`as.factor(letter[1:3])`
逻辑（logical）	`c(TRUE, FALSE)`
日期（date）	`as.Date()` `Sys.Date()`
缺失值（missing data）	`NA`
原始数据（raw）	`as.raw(48)`

❖ 实数（Numeric）

- 从类型上说，实数（numeric）包括浮点型（double）和整型（integer），但在实践中，实数即是指浮点型。
- .Machine 输出当前系统的数值特征，包括最大整数、精度等。
- 实数的格式化输出使用 format() 函数。

```
# trim: 是否去除前后空格
format(c(1,10,100,1000), trim = FALSE)
## [1] "   1" "  10" " 100" "1000"
format(c(1,10,100,1000), trim = TRUE)
## [1] "1"    "10"   "100"  "1000"
# nsmall: 小数点后位数
format(13.7, nsmall = 3)
## [1] "13.700"
# 科学记数法
format(2^16, scientific = TRUE)
```

```
## [1] "6.5536e+04"
```

❖ 类型判断

```
# False, 因为 R 默认存储数据的格式是double浮点型
identical(1, as.integer(1))
## [1] FALSE
# 在数值后加后缀 L, 表示整型
identical(1L, as.integer(1))
## [1] TRUE
# True
identical(1, as.numeric(1))
## [1] TRUE
# True
identical(1, 1.)
## [1] TRUE
```

❖ 逻辑

- 逻辑型用大写的 TRUE 和 FALSE 表示, 可缩写为 T 或 F。

❖ 示例

```
x <- 1:10
# x 中大于5的偶数
(x%%2==0) | (x > 5)
##  [1] FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
TRUE
x[(x%%2==0) | (x > 5)]
## [1]  2  4  6  7  8  9 10
y <- 5:15
x %in% y
##  [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
TRUE
x[x %in% y]
```

`!x`	非 `x`
`x & y`	`x` 且 `y`，对向量元素操作，返回向量
`x && y`	`x` 且 `y`，对向量操作，返回单个值
`x | y`	`x` 或 `y`，对向量元素操作，返回向量
`x || y`	`x` 或 `y`，对向量操作，返回单个值
`xor(x, y)`	异或，对向量元素操作，返回向量
`x < y`	$x < y$
`x > y`	$x > y$
`x <= y`	$x \leq y$
`x >= y`	$x \geq y$
`x == y`	$x = y$
`x != y`	$x \neq y$
`isTRUE(x)`	`x` 是否为真
`all(...)`	是否所有的参数为真
`any(...)`	是否至少一个参数为真
`identical(x,y)`	检验两个对象是否完全相等
`all.equal(x,y)`	检验两个对象是否近似相等
`x %in% y`	`x` 的元素在 `y` 中是否存在

```
## [1]  5  6  7  8  9 10
any(x>5)
## [1] TRUE
all(x>5)
## [1] FALSE
```

❖ 为何不相等?

- 逻辑操作符的返回值既不是 TRUE, 也不是 FALSE? 当向量中包含元素为 NA 或 NaN 时。
- 如果必须要得到单一的TRUE 或 FALSE 值, 例如在条件判断中 (if), 那么就避开使用 == 或!=。
- 使用 identical()。

```
1  name <- "Nick"
2  if(name=="Nick") TRUE else FALSE
3  ## [1] TRUE
4
5  name <- NA
6  if(name=="Nick") TRUE else FALSE
7  ## Error in if (name == "Nick") TRUE else FALSE: missing value
   where TRUE/FALSE needed
8  if(identical(name, "Nick")) TRUE else FALSE
9  ## [1] FALSE
```

❖ 为何不相等?

- `all.equal()` 检验两个对象是否相等, 并考虑一定的精度误差。当差异低于设定的容忍度时, 即返回 `TRUE`;
- `all.equal()` 返回 `TRUE` 或描述差异的字符。应避免在条件判断中使用。

```
(x <- sqrt(2))
## [1] 1.414214
x^2
## [1] 2
x^2==2
## [1] FALSE
all.equal(x^2, 2)
## [1] TRUE
all.equal(x^2, 1)
## [1] "Mean relative difference: 0.5"
isTRUE(all.equal(x^2, 1))
## [1] FALSE
```

❖ 字符

- 字符型的定义使用成对单引号或双引号，' '或" ";
- 常用函数

❖ 编码

```
# 系统编码
localeToCharset()
## [1] "CP936"

x<-'中文'
Encoding(x)
## [1] "unknown"

enc2native(x)
## [1] "中文"
enc2utf8(x)
```

`cat()`	连接多个字符对象，并打印到控制台
`paste()`	连接多个字符对象，并返回字符对象
`print()`	打印
`substr()`	提取或替换子串
`strtrim()`	根据指定的宽度，裁剪字符向量元素
`strsplit()`	将字符向量元素拆分成多个子串
`grep()`	查找子串，返回匹配位置
`grepl()`	查找子串，返回逻辑向量
`agrep()`	类似 `grep()``，模糊匹配
`regexpr()`	查找子串，返回第一个匹配位置
`sub()`	替换子串
`gsub()`	全局替换匹配的子串
`tolower(), toupper()`	转换为小写、大写
`noquote()`	输出字符向量，但是不显示引号
`nchar()`	字符长度
`letters, LETTERS`	小写、大写英文字母

```
12  ## [1] "中文"
13
14  # 查看字符编码
15  (r1 = charToRaw(x))
16  ## [1] d6 d0 ce c4
17  rawToChar(r1)
18  ## [1] "中文"
19
20  iconv(x, "GB2312", "UTF-16LE", toRaw = T)
21  ## [[1]]
22  ## [1] 2d 4e 87 65
```

❖ 因子

- R 中，类别变量有一种特殊的表示方式，不同于 spss 或 stata。类别变量在R中用 factor 表示，其本质上也是一个向量对象，只不过将类别的各个层次在其内部用数字表示并将其名称保存在 levels 属性中。
- 使用 factor() 创建因子对象

```
1  factor(rep(1:2, 4), labels=c("trt.1", "trt.2"))
2  ## [1] trt.1 trt.2 trt.1 trt.2 trt.1 trt.2 trt.1 trt.2
3  ## Levels: trt.1 trt.2
4  factor(rep(1:3, 4), labels=c("low", "med", "high"), ordered=TRUE)
5  ##  [1] low  med  high low  med  high low  med  high low  med
   high
6  ## Levels: low < med < high
```

- 常用函数

'levels(x)`	返回或设置 x 的类别描述（level）
'nlevels(x)`	返回 x 类别个数
'relevel(x, ref)`	重新排序 x 的类别，并使 ref 参数所设定的类为第一个类别
'reorder()`	根据参数重新排序类别
'gl()`	根据参数所设定的类别模式生成因子对象
'cut(x, breaks)`	将连续变量 x 切分成因子，参数 breaks 控制间隔

❖ 示例

```
x<-c(1,2,3)
(y<-as.factor(x))
## [1] 1 2 3
## Levels: 1 2 3
attributes(y)
## $levels
## [1] "1" "2" "3"
##
```

```
## $class
## [1] "factor"
levels(y)<-c('a', 'b', 'c')
y
## [1] a b c
## Levels: a b c

# 用 `unclass()` 移除 `y` 的类属性，于是我们可用默认的 `print` 方法来查看 `y` 的内部结构
unclass(y)
## [1] 1 2 3
## attr(,"levels")
## [1] "a" "b" "c"
```

❖ slide

```
# 将无序因子转化为有序因子
y.ordinal<-factor(x, ordered = TRUE)
attributes(y.ordinal)
```

```
## $levels
## [1] "1" "2" "3"
##
## $class
## [1] "ordered" "factor"

# 移除多余的的类别
y<-factor(f)
## Error in factor(f): object 'f' not found
levels(y)[2] = NA

# 或更加通用的方法，可用于向量、列表、数据集
y<-droplevels(y)

# `gdata` 提供了也提供了类似的方法，但是会对类别层次按照字符进行重新排序，慎用！
library(gdata)
drop.levels(cgss)
```

❖ slide

```
# 将连续变量转换为类别变量
factor(cut(1:20,
         breaks=c(0,2,3, Inf), right=FALSE),
         labels=c( "1 or less", "2", "3 +"))
##  [1] 1 or less 2         3 +       3 +       3 +       3 +
3 +
##  [8] 3 +       3 +       3 +       3 +       3 +       3 +
3 +
## [15] 3 +       3 +       3 +       3 +       3 +       3 +
## Levels: 1 or less 2 3 +
```

❖ 时间及日期

- 关于日期与时间更多的信息，`?DateTimeClasses`
- 常用函数

`Sys.Date()`	当前日期
`as.Date()`	将字符类型对象转换为日期类型对象
`format.Date()`	格式化
`seq.Date()`	生成日期序列
`cut.Date()`	生成日期间隔
`weekdays,months,quarters`	提取天、月、季
`julian(x, origin)`	自 `origin` 以来的天数

- `as.Date()`也可计算天数，即自1970-01-01以来的天数，如果早于1970年01月01日则用负数表示。

```
as.Date('1970-01-01')
## [1] "1970-01-01"
```

❖ POSIXlt vs. POSIXct

- R 提供两个表示日期与时间的类型，“POSIXlt” 和 “POSIXct”，其中 POSIXct 是一个数值向量，表示自1970年01月01日以来的秒数，而 POSIXlt 则是一个命名向量列表，表示：

  sec 0--61: 秒 min 0--59: 分钟 hour 0--23: 小时 mday 1--31: 一个月中的天数 mon 0--11: 月 year 自1900年以来的年数 wday 0--6: 一周中的天数，以周日为起点 yday 0--365: 一年中的天数 isdst 夏日制标识，正数表示夏日制，0表示没有实行夏日制，负数表示未知信息。

❖ slide

```
as.POSIXlt(Sys.time(),"GMT")
## [1] "2015-10-18 15:40:01 GMT"
as.POSIXct(Sys.time(),"GMT")
## [1] "2015-10-18 23:40:01 CST"
```

```
unclass(as.POSIXlt(Sys.time(),"GMT"))
## $sec
## [1] 1.42028
##
## $min
## [1] 40
##
## $hour
## [1] 15
##
## $mday
## [1] 18
##
## $mon
## [1] 9
##
## $year
## [1] 115
```

```
##
## $wday
## [1] 0
##
## $yday
## [1] 290
##
## $isdst
## [1] 0
##
## attr(,"tzone")
## [1] "GMT"
unclass(as.POSIXct(Sys.time(),"GMT"))
## [1] 1445182801
```

❖ 字符转换成时间

- 将字符串转换成 `date` 对象需要指明格式信息;
- 格式说明用 `%+关键字` 来说明:

'%a'	英文工作日名称缩写
'%A'	英文工作日名称
'%d'	月中天数
'%B'	英文月份名称
'%b'	英文月份名称缩写
'%m'	数字表示的月份 (01-12)
'%y'	年份（两位数）
'%Y'	年份（四位数）

- 示例

```
dates.1 <- c("5jan2008", "19aug2008", "2feb2009", "29sep2009")
as.Date(dates.1, format="%d%b%Y")
## [1] NA NA NA NA
dates.2 <- c("5-1-2008", "19-8-2008", "2-2-2009", "29-9-2009")
as.Date(dates.2, format="%d-%m-%Y")
## [1] "2008-01-05" "2008-08-19" "2009-02-02" "2009-09-29"
```

❖ 创建日期序列

```
seq.Date(as.Date("2011/1/1"), as.Date("2011/1/31"), by="week")
## [1] "2011-01-01" "2011-01-08" "2011-01-15" "2011-01-22" "2011-01-29"
seq.Date(as.Date("2011/1/1"), as.Date("2011/1/31"), by="3 days")
##  [1] "2011-01-01" "2011-01-04" "2011-01-07" "2011-01-10" "2011-01-13"
##  [6] "2011-01-16" "2011-01-19" "2011-01-22" "2011-01-25" "2011-01-28"
## [11] "2011-01-31"
seq.Date(as.Date("2011/1/1"), by="week", length.out=10)
##  [1] "2011-01-01" "2011-01-08" "2011-01-15" "2011-01-22" "2011-01-29"
##  [6] "2011-02-05" "2011-02-12" "2011-02-19" "2011-02-26" "2011-03-05"
```

❖ 日期运算

- 可以对日期对象加减天数
- 日期对象可以加减
- 日期可以比较

```
jan1 <- as.Date("2011/1/1")
(jan8 <- jan1 + 7) # Add 7 days to 2011/1/1
## [1] "2011-01-08"
jan1 - 14 # Subtract 2 weeks from 2011/1/8
## [1] "2010-12-18"
jan8 - jan1 # Number of days between 2011/1/1 and 2011/1/8
## Time difference of 7 days
jan8 > jan1 # Compare dates
## [1] TRUE
# Use format to extract parts of a date object or change the appearance
format.Date(jan8, "%Y")
## [1] "2011"
format.Date(jan8, "%b-%d")
```

```
14    ## [1] "一月-08"
```

❖ 缺失值

❖ NA

- 缺失值用 NA (Not Available)表示；NA 可表示任意类型的缺失；
- 可以直接将 NA 赋予向量或矩阵对象中的某一个元素，比如：

  x[2]<-NA

❖ 缺失值判断

```
x <- c(4, 7, 2, 0, 1, NA)
x==NA
## [1] NA NA NA NA NA NA
is.na(x)
## [1] FALSE FALSE FALSE FALSE FALSE  TRUE
anyNA(x)
## [1] TRUE
is.na(x) <- 99
```

❖ 包含缺失值的统计量

在R中，有一些包或函数要求数据不能含有缺失值，比如max()、mean()、var()、cor()函数，这种情况下，要使用na.action或na.rm, na.exclude等参数, 否则函数报错。

```
x <- c(4, 7, 2, 0, 1, NA)
var(x)
## [1] NA
mean(x)
## [1] NA
mean(x, na.rm=T)
## [1] 2.8
mean(x[!is.na(x)])
## [1] 2.8
```

❖ NaN

- 无法表示的量（0/0）或者数值类型的缺失值用 NaN (Not a Number)表示；

```
(y <- x/0)
## [1] Inf Inf Inf NaN Inf  NA
is.nan(y)
## [1] FALSE FALSE FALSE  TRUE FALSE FALSE
is.na(y)
## [1] FALSE FALSE FALSE  TRUE FALSE  TRUE
```

❖ NULL

- 未定义的对象用 `NULL` 表示;
- 通常作为函数或表达式的 (未定义) 返回值;

```
x = c(NULL, 1)
x
## [1] 1

is.null(list())     # FALSE (on purpose!)
## [1] FALSE
is.null(integer(0)) # FALSE
## [1] FALSE
is.null(logical(0)) # FALSE
## [1] FALSE
```

❖ 类型判断及转换

- R 所有的对象都具有类型属性，使用 `is._type_()` 函数来判断；使用 `typeof()` 来查看数据类型；
- 使用 `as_type_()` 函数进行类型转换；
- 转换时的一般规则：
  - 逻辑值转换为实数时，`FALSE = 0, TRUE = 1`；
  - 实数转换为逻辑型时，非零值都为真 (`TRUE`)；
  - 字符向量转换为因子向量时，向量中的所有元素按照字符排序，自动编码并设置标签 (`level`)；
  - 表示数值的字符可以自动转换为实数；

```
1  x <- 1:10
2  is.numeric(x)
3  ## [1] TRUE
4
```

```
5   sum(x>5)  # 逻辑型自动转换为整型
6   ## [1] 5
7
8   x = as.numeric(c("-.1"," 2.73 ","B"))
9   x
10  ## [1] -0.10  2.73    NA
11  format(x, nsmall=2)
12  ## [1] "-0.10" " 2.73" "   NA"
```

❖ 类型强制

- 向量的所有元素必须为同一类型;
- 当需要组合不同类型的向量时，发生类型强制转换;
- 转换的顺序

  logical > integer > double > character

```
1  c(1,F)
2  c(1,'a')
3
4  x <- c(FALSE, FALSE, TRUE)
5  sum(x)
```

R语言编程

◆ 控制结构

`if()...else`	
`ifelse(test, yes, no)`	
`switch()`	
`for()`	执行有限次数循环
`while()`	执行循环，直到触发条件为 `FALSE`
`repeat`	执行循环，直到触发 `break`
`break`	跳出循环
`next`	停止当前迭代，进入下一次迭代

❖ if()...else

- 一定要加 `()`;
- `condition` 必须为单个的逻辑值;
- 当语句较为复杂时, 可以使用 `{}`;

```
1   if(condition) expression if TRUE
2
3   if(condition) {
4       expressions if TRUE
5   }
6
7   if(condition) expression if TRUE else expression if FALSE
8
9   if(condition) {
10      expressions if TRUE
11  } else {
```

```
12        expressions if FALSE
13    }
```

❖ 示例

计算随机样本$x_1 ... x_n$的中位数

当变量值的项数 N 为奇数时，处于中间位置的变量值即为中位数；当N为偶数时，中位数则为处于中间位置的2个变量值的平均数。

$$median(x) = \begin{cases} \frac{1}{2}x_{(\frac{n}{2})} + \frac{1}{2}x_{(1+\frac{n}{2})}, & \text{\& if n is even} \\ x_{(\frac{n+1}{2})} & \text{\& if n is odd} \end{cases}$$

```
1     x <- c(5,4,2,8,9,10)
2     n <- length(x)
3     sort.x <- sort(x)
4
5     if(n%%2==0) {
6       median <- (sort.x[n/2]+sort.x[1+n/2])/2
7     } else median <- sort.x[(n+1)/2]
```

```
8
9   median
10  ## [1] 6.5
```

❖ ifelse()

- `ifelse(test, yes, no)`
- 返回值的结构与 `test` 一致，
- 对 `test` 中的每个元素进行条件判断，若为TRUE，则返回值为 `yes`，反之为 `no`

```
(x <- seq(0,2,len=6))
## [1] 0.0 0.4 0.8 1.2 1.6 2.0
ifelse(x <= 1, "small", "big")
## [1] "small" "small" "small" "big"   "big"   "big"
(y <- matrix(1:8, nrow=2))
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
ifelse(y>3 & y <7, 1, 0)
##      [,1] [,2] [,3] [,4]
```

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

❖ switch()

- `switch(EXPR, ...)`
- 根据 `EXPR` 分别执行不同的命令;
- `EXPR` 必须为单个元素的字符或数值, 不能是一个向量;
- `...` 是以逗号分隔的多个表达式, 其形式为 `name=expression` 或 `number=expression`;
- 如果某个 `EXPR` 条件后面没有表达式, 则执行下一句表达式;
- 如果 `EXPR` 没有找到匹配, 则执行第一个未命名的表达式 (作为默认值);

```
central <- function(y, measure) {
  switch(measure,
         Mean = ,
         mean = mean(y),
         median = median(y),
```

```
6             geometric = prod(y)^(1/length(y)),
7             "Invalid Measure")
8     }
9
10    y <- runif(100)
11    central(y, "mean")
12    ## [1] 0.4493669
13    central(y, "Mean")
14    ## [1] 0.4493669
15    central(y, "Median")
16    ## [1] "Invalid Measure"
17    central(y, "geometric")
18    ## [1] 0.2729004
19    central(y, "nomatch")
20    ## [1] "Invalid Measure"
```

❖ for 循环

```
for(var in seq) {
    expressions
}
```

- `var` 在每次迭代中取值发生变动;
- `seq` 必须为向量;
- 迭代的次数等于 `seq` 的长度;
- 当有多个表达式时, 使用 `{}`;

```
# Calculate 10! using a for loop
f <- 1
for(i in 1:10) {
    f <- f*i
    cat(i, f, "\n")
}
## 1 1
## 2 2
```

```
## 3 6
## 4 24
## 5 120
## 6 720
## 7 5040
## 8 40320
## 9 362880
## 10 3628800
f
## [1] 3628800
factorial(10)
## [1] 3628800
```

❖ while 循环

```
while(cond) {
    expressions
}
```

- 一定要检查 cond 的退出机制，否则陷入死循环

```
# Calculate 10! using a while loop
i <- 10
f <- 1
while(i>1) {
    f <- i*f
    i <- i-1
    cat(i, f, "\n")
}
## 9 10
## 8 90
## 7 720
```

```
## 6 5040
## 5 30240
## 4 151200
## 3 604800
## 2 1814400
## 1 3628800
f
## [1] 3628800
factorial(10)
## [1] 3628800
```

❖ repeat 循环

```
repeat {
    expressions
    if(cond) break
}
```

- 一定要有 break

```
# Calculate 10! using a repeat loop
i <- 10
f <- 1
repeat {
    f <- i*f
    i <- i-1
    cat(i, f, "\n")
    if(i<1) break
}
## 9 10
```

```
## 8 90
## 7 720
## 6 5040
## 5 30240
## 4 151200
## 3 604800
## 2 1814400
## 1 3628800
## 0 3628800
f
## [1] 3628800
factorial(10)
## [1] 3628800
```

❖ 向量化 vs 循环

```
1   a <- 1:100
2
3   # 向量运算
4   b <- sin(a)
5
6   # 循环
7   b <- vector()
8   for (i in seq_along(a)) {
9       b[i] <- sin(a[i])
10  }
11
12  system.time(replicate(10000, b <- sin(a)))
13  ##    user  system elapsed
14  ##    0.17    0.00    0.17
15
16  b <- vector()
```

```
17  system.time(replicate(10000, for (i in  seq_along(a)    ) {b[i] <-
    sin(a[i])}))
18  ##    user  system elapsed
19  ##    1.76    0.00    1.76
```

◆ 函数

- 函数也是一种对象，包含多个表达式，并使用参数；
- 函数声明

```
function name <- function(argument list) {
    body
}
```

- 参数列表用逗号分隔，可有三种书写形式：
  - name
  - name = default value
  - ...
- 如果函数体只有一句表达式，则 `{}` 可忽略；
- 函数通常是命名的，某些情况下也可以是匿名的，例如 `lapply()` 的参数；
- 函数返回值可为任意类型，但仅能返回一个对象；

```
return(x)
```

❖ 示例

file://attach/demo_2.r

◆ 高阶函数

apply	操作矩阵或数组，返回数组、矩阵或向量
lapply	操作向量或列表，返回列表
sapply	操作向量或列表，返回向量或矩阵
tapply	操作向量，根据因子分组，返回向量
mapply	操作多个向量，返回列表
vapply	与sapply同，操作列表，返回向量或列表
rapply	与lapply类似，操作向量与列表，返回向量

❖ apply示例

```
(m <- matrix(1:8, 2, 4))
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

```
5
6   apply(m, 1, sum) # 相当于 rowSums(m)
7   ## [1] 16 20
8
9   apply(m, 2, sum) # 相当于 colSums(m)
10  ## [1]  3  7 11 15
11
12  apply(m, 1:2, function(x) x / 2)
13  ##      [,1] [,2] [,3] [,4]
14  ## [1,]  0.5  1.5  2.5  3.5
15  ## [2,]  1.0  2.0  3.0  4.0
```

❖ lapply, sapply, vapply 示例

```
(l <- list(a = 1:5, b = 6:10))
## $a
## [1] 1 2 3 4 5
##
## $b
## [1]  6  7  8  9 10

lapply(l, sum) # 返回列表
## $a
## [1] 15
##
## $b
## [1] 40

sapply(l, sum) # 返回向量
##  a  b
```

```
## 15 40

(X <- sapply(3:5, seq))
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] 1 2 3 4
##
## [[3]]
## [1] 1 2 3 4 5

sapply(X, fivenum)
##      [,1] [,2] [,3]
## [1,]  1.0  1.0    1
## [2,]  1.5  1.5    2
## [3,]  2.0  2.5    3
## [4,]  2.5  3.5    4
```

```
35  ## [5,]  3.0  4.0    5
36
37  (v <- c("Min."=0, "1st Qu."=0, "Median"=0, "3rd Qu."=0, "Max."=0))
38  ##    Min. 1st Qu.  Median 3rd Qu.    Max.
39  ##       0       0       0       0       0
40
41  vapply(X, fivenum, v)
42  ##         [,1] [,2] [,3]
43  ## Min.     1.0  1.0    1
44  ## 1st Qu.  1.5  1.5    2
45  ## Median   2.0  2.5    3
46  ## 3rd Qu.  2.5  3.5    4
47  ## Max.     3.0  4.0    5
```

❖ mapply 示例

当要对多个对象进行操作时，用mapply。

```
mapply(function(x, y) seq_len(x) + y,
       c(a =  1, b = 2, c = 3),  # names from first
       c(A = 10, B = 0, C = -10))
## $a
## [1] 11
##
## $b
## [1] 1 2
##
## $c
## [1] -9 -8 -7
```

数据管理

◆ 从导入数据开始

❖ 导入文本

❖ 读取 **csv**

```
1  df <- read.csv(textConnection("a,b,c, 1,2,3, 4,5,6, 7,8,9"))
2  write.csv(df, file='dummy.csv')
3  dd <- read.csv('dummy.csv')
```

❖ 读取固定宽度文本

```
x <- read.fwf(file='fixed_width_data.txt',
              widths=c(6, 1, 5, 9, 4, 9, 4, 9))
head(x)

library(readr)
x <- read_fwf(file='fixed_width_data.txt',
              fwf_widths(c(6, 1, 5, 9, 4, 9, 4, 9)))
head(x)
```

❖ 导入 Stata

以中国综合社会调查 (CGSS) 为例

- 下载网址 `http://www.cnsda.org/index.php?r=projects/view&id=93281139`

```
1  library(haven)
2  gss = read_dta('CGSS2013（居民问卷）发布版.dta')
3  dim(gss)
4  View(gss)
5  names(gss)
6
7  lbl = sapply(gss, attr, 'label')
8  Encoding(lbl) = 'GB2312'
9  gss_lbl = data.frame(var = names(gss), lbl = lbl)
10
11 library(foreign)
12 d1<-read.dta(file = 'e:/cgss2003data.dta',
```

```
13                  convert.factors = TRUE, missing.type = FALSE)
14  # stata 文件中所包含的数值标签、变量标签等信息
15  # 数值标签 val.labels
16  attr(dat, 'var.labels')
17  attr(dat, 'labels.table')
```

❖ 导入 SPSS

```
1  library("foreign")
2  x <- read.spss("d:/se.sav", use.value.labels = TRUE, to.data.frame
   = FALSE)
```

❖ 导入 Excel

❖ 读取剪贴板

```
1  read.clipboard(header = TRUE, . . . ) #assumes headers and tab or
   space delimited
2  read.clipboard.csv(header=TRUE, sep=',',...) #assumes headers and
   comma delimited
3  read.clipboard.lower(diag=TRUE, names=NULL,...) #read in a matrix
   given the lower off diagonal
4  read.clipboard.upper(diag=TRUE, names=NULL,...)
```

Excel 相关R包:

- readxl
- xlsx
- xlsReadWrite

```
1  library(readxl)
2  datasets <- system.file("extdata/datasets.xlsx", package = "readxl")
```

```
3    read_excel(datasets)
```

◆ 存储

- 保存对象及其所处环境（Environment）设置

```
save(a list of object, file="file with path")
```

- 保存当前工作空间

```
save.image(file="")
```

- 载入对象

```
load(file="")
```

- 保存单个对象

```
saveRDS(obj, file="") x<-readRDS("filename")
```

◆ 查看数据

- 编辑数据

`fix()`

- 列出首、尾 n 个 case 数据
  - `head(x, n)`
  - `trail(x, n)`
- 对比两个数据集是否相同

```
1  clone<-iris
2  clone[1, 1] <-9
3  # 检验属性是否都相同
4  attr.all.equal(iris, clone)
5  # 检验两个对象是否相同，如果相同返回True，否则报告差异
6  all.equal(target = iris, current = clone, tolerance= 0.1)
```

```
# 检验两个对象是否相同，返回逻辑值
identical(iris, clone)
```

- 检查两个数据集是否有相同的变量名

```
cnames<-intersect(colnames(df1), colnames(df2))
# 提取变量名相同的部分数据
df1sub = df1[, match(cnames, colnames(df1))]
df2sub = df2[, match(cnames, colnames(df2))]
```

◆ 筛选数据

❖ 获取索引

```
1  # 获取向量中最大数、最小数的索引
2  a <- c(1,2,0,3,7,0,0,0)
3  which.min(a)
4  which.max(a)
5  which(a == min(a))  # returns both 3 and 6
6
7  # 按条件获取索引
8  which(a == min(a))
```

❖ [操作符

对于向量、矩阵、数组、列表可使用下述操作符:

- `x[i]`、`x[i, j, ...]`: `i, j` 为整数、字符或逻辑向量;
- `x[[i]]`、`x[[i, j, ...]]`: `i, j` 为整数或字符向量;
- `x$name`

❖ [

```
d <- matrix(1:15, ncol =3)
d[c(1,6)]
## [1] 1 6
d[[2]]
## [1] 2
# `[[` 只支持一个元素
d[[2:3]]
## Error in d[[2:3]]: attempt to select more than one element
# i = 逻辑向量
```

```
d[which(d[, 1]>3), ]
##      [,1] [,2] [,3]
## [1,]    4    9   14
## [2,]    5   10   15
# i = 字符向量
rownames(d) = LETTERS[1:5]
d[c('B', 'D'), ]
##   [,1] [,2] [,3]
## B    2    7   12
## D    4    9   14
```

❖ [.data.frame

```
df <- data.frame(x = c(1,2,3), y = c(3,4,5))
# 按行
df[c(1,3), ]
##   x y
## 1 1 3
## 3 3 5
class(df[c(1,3), ])
## [1] "data.frame"
# 按列
df[, 2]
## [1] 3 4 5
class(df[, 2])
## [1] "numeric"

select1 = c(T, T)
select2 = c(T, F)
df[, select1]
```

```
##   x y
## 1 1 3
## 2 2 4
## 3 3 5
df[, select2]
## [1] 1 2 3
# `drop` 对结果是否降维。默认值为当返回值仅有一列时降维，而当返回值仅有一行时则不降维
df[, 2, drop = F]
##   y
## 1 3
## 2 4
## 3 5
```

❖ 条件筛选并赋值

```
1    df[which(df$x > 2), 'y'] = 10
```

❖ subset()

以自带数据 iris 为例:

```
1  # iris 数据有5个变量, `"Sepal.Length" "Sepal.Width" "Petal.Length"
   "Petal.Width" "Species"`
2  names(iris)
3
4  # 筛选行数据
5  iris.x <- subset(iris, Petal.Width > 1.5 )
6
7  # 筛选变量
8
9  # 去除 Species 变量
10 iris.x <- subset(iris, select= -Species)
11 iris.x <- subset(iris, drop = Species)
12
13 # 去除多个变量
14 iris.s <- subset(iris, select= -c(Species, Petal.Width))
```

```
15
16  # 仅保留部分变量
17  iris.s <- subset(iris, select= c(Species, Petal.Width))
18
19  # 按照起始变量范围选择
20  iris.s <- subset(iris, select= Petal.Length:Species)
21  iris.s
```

◆ 查重

- `unique.data.frame()` 可对数据框（data.frame）进行去重;
- `duplicated.data.frame()` 返回是否重复的逻辑值;

```
df = data.frame(a = 1:4, b = sample(4))
df = rbind(df, df[2:3, ])

unique(df)
##   a b
## 1 1 3
## 2 2 4
## 3 3 1
## 4 4 2
duplicated(df)
## [1] FALSE FALSE FALSE FALSE  TRUE  TRUE
```

◆ 变量重命名

```
1   # 1
2   names <- c("john", "tim", "andy")
3   ages <- c(50, 46, 25)
4   mydata <- data.frame(names,ages)
5   names(mydata)
6   ## [1] "names" "ages"
7   (names(mydata) = c('names1', 'ages1'))
8   ## [1] "names1" "ages1"
9
10  # 2
11  library(reshape)
12  mydata <- rename(mydata, c(ages="age"))
```

◆ 变量数值再编码

❖ 连续变量离散化

```
ii <- 0:8; names(ii) <- ii
cut(ii, breaks=c(0, 2, 4, 6, 8))
## [1] <NA>  (0,2] (0,2] (2,4] (2,4] (4,6] (4,6] (6,8] (6,8]
## Levels: (0,2] (2,4] (4,6] (6,8]

# 使用`fivenum` 杜克五基（Tukey's five number summary），即最低数、四分位数、中位数、75%位数，最大数。
as.factor(cut(ii, fivenum(ii)))
## [1] <NA>  (0,2] (0,2] (2,4] (2,4] (4,6] (4,6] (6,8] (6,8]
## Levels: (0,2] (2,4] (4,6] (6,8]

symnum(ii, cut= 2*(0:4), sym = c("0-2", "3-4", "5-6", ">6"))
##   0   1   2   3   4   5   6   7   8
## 0-2 0-2 0-2 3-4 3-4 5-6 5-6  >6  >6
```

```
14  ## attr(,"legend")
15  ## [1] 0 '0-2' 2 '3-4' 4 '5-6' 6 '>6' 8
16
17  # 类似 Stata 的语法, 使用 `car` 包中的 `recode()`
18  library(car)
19  x<-rep(1:3,3)
20  x
21  ## [1] 1 2 3 1 2 3 1 2 3
22  recode(x, "c(1,2)='A'; else='B'")
23  ## [1] "A" "A" "B" "A" "A" "B" "A" "A" "B"
24  recode(x, "1:2='A'; 3='B'")
25  ## [1] "A" "A" "B" "A" "A" "B" "A" "A" "B"
```

❖ 定类变量再编码

```
###假设 r3 为 factor,
levels(r3)
###[1] "农业(农村)户口" "城镇(非农)户口" "其他"
###过滤 r3="其他" 的数据
which(dat$r3 == levels(r3)[3])
###或者
which(dat$r3 == "其他")
```

或者使用 car 包

```
require(car)
df <- data.frame(a=letters[1:26],1:26)
df2 <- within(df, a <- recode(a, 'c("a","b","c")="a"'))
head(df2)
##   a X1.26
## 1 a     1
## 2 a     2
## 3 a     3
```

```
9     ## 4 d     4
10    ## 5 e     5
11    ## 6 f     6
```

◆ 缺失值

```
1   data <- data.frame(x = c(1, 2, 3), y = c(0, 10, NA))
2   na.omit(data)
3   ##   x  y
4   ## 1 1  0
5   ## 2 2 10
6   complete.cases(data)
7   ## [1]  TRUE  TRUE FALSE
8
9   data<-as.matrix(data)
10  data<-na.omit(data)
11
12  # 编码为缺失值
13  data[data$x==3, "x"] <- NA
14  ## Error in data$x: $ operator is invalid for atomic vectors
```

◆ 排序与组合

❖ 排序

```
1   # 单个变量排序
2
3   data[sort(data$v1), ]
4   # 逆排序
5   data[sort(data$v1, decreasing = TRUE), ]
6   # 另见 rev()
7
8   # 多个变量排序
9   library(reshape)
10  sort_df(data, vars=c('id', 'city'))
11  # 按照数据集 data 中的所有变量排序
12  sort_df(data, vars=names(data))
```

❖ 组合

```
x <- 1:3
y <- 1:3
z <- outer(x,y,FUN="-")
z
##      [,1] [,2] [,3]
## [1,]    0   -1   -2
## [2,]    1    0   -1
## [3,]    2    1    0
x <- c("A", "B", "C", "D")
y <- 1:9
z <- outer(x, y, paste, sep = "")
z
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## [1,] "A1" "A2" "A3" "A4" "A5" "A6" "A7" "A8" "A9"
## [2,] "B1" "B2" "B3" "B4" "B5" "B6" "B7" "B8" "B9"
## [3,] "C1" "C2" "C3" "C4" "C5" "C6" "C7" "C8" "C9"
```

```
17   ## [4,] "D1" "D2" "D3" "D4" "D5" "D6" "D7" "D8" "D9"
```

◆ 分组计算生成新变量

例子: 需要根据一个分组变量, 计算均值, 并将计算结果作为新变量保存在原数据集中。

也即是将数据集

```
1    id  group  value
2    1   a      10
3    2   a      20
4    3   b      100
5    4   b      200
```

变成

```
1    id  group  value  grp.mean.values
2    1   a      10     15
3    2   a      20     15
4    3   b      100    150
```

```
5    4   b       200     150
```

- 方法一: 使用 `plyr::ddply` 函数

```
1    require(plyr)
2    ddply(dat, "group", transform, grp.mean.values = mean(value))
3
4      id group value grp.mean.values
5    1  1     a    10              15
6    2  2     a    20              15
7    3  3     b   100             150
8    4  4     b   200             150
```

- 方法二: 由于这里正好是计算均值, 所以可以直接使用 `stats::ave()` 对定类变量的各组求均值

```
1    df$grp.mean.values <- ave(df$value, df$group)
```

- 方法三: 使用 `merge` 及 `aggregate` 组合

```
1    merge(x, aggregate(value ~ group, data = x, mean),
2          by = "group", suffixes = c("", ".mean"))
```

◆ 宽数据与长数据

宽数据转换为长数据: 标识变量不变, 其它变量每个作为一行, 插入到新生成的数据中。

在 R 中有很多种方式实现数据的分组或重构, 但大多数底层函数难以理解, 操作繁琐。 reshape2 包针对数据集类型对象的长宽数据转换需要提供了两个封装好的函数 melt() 和 cast()。

melt() 函数将宽数据转换为长数据, 即将多列数据堆积成单列多行数据。它需要两个关键参数, id 变量以及 measure 量度变量 (将被合并的变量), 默认假设除 id 变量之外的其它变量均为量度变量。

```
1  # reshape2 示例
2  library(reshape2)
3  names(airquality) <- tolower(names(airquality))
4  melt(airquality, id=c("month", "day"))
```

线性模型

R作图--ggplot2介绍

RR扩展

◆ 网络爬虫示例

◆ 文本分析

◆ 社会网络分析

◆ 在线统计 Shiny 应用

◆ 并行计算