《R Cookbook》

中文笔记

说明：本笔记主要提炼了《R Cookbook》书中的重点内容并按笔者的经验整理而来，对于广大爱好者而言，不免会有偏差，笔记中也不乏翻译不到位之处，希望各位留言交流。

笔者信息

QQ：cador(2684929905)

邮箱：cador@sina.com

博客：www.youhaolin.com

2013 年 4 月

目　　录

@在 linux 系统中安装 R

在 Ubuntu 或 Debian 系统中，使用 apt-get 下载并安装 R。运行如下 sudo 代码需要一定的权限：

$ sudo apt-get install r-base

在 Red Hat 或 Fedora 系统中，使用 yum：

$ sudo yum install R.i386

除了基础包以外，我还建议安装文档包。比如在我的 Ubuntu 机器中，我安装了 r-base-html（因为我喜欢浏览文档链接），还安装了 r-doc-html，安装命令如下：

$ sudo apt-get install r-base-html r-doc-html

@获得帮助

使用 help 命令查看该函数的文档：

> help(*functionname*)

使用 args 函数，获取函数参数的简要说明：

> args(*functionname*)

使用 example 函数来查看该函数的例子：

> example(*functionname*)

如果你想知道更多关于 mean 函数。像这样使用 help 函数：

>help(mean)

这会打开一个函数文档的窗口或者在你的控制台呈现文档，要依赖你的系统平台。使用 help

命令的快捷方式，只需要敲入 ‘?’ 后面跟着函数名：

>?mean

使用 help.search 函数，在你的电脑中查找 R 文档：

>help.search("*pattern*")

一个典型的 patter 是一个函数名或者关键词。注意它应该加上引号。

为了方便起见，你也可以通过两个问题调用一个查找（在这种情况下，不需要加引号）：

>??*pattern*

使用 help 函数，并指定一个包名（而不是函数名）：

>help(package="*packagename*")

使用 vignette 函数，你可以查看你电脑中的所有小品文（*看了一下，可以找到相关的 PDF 文档*）列表：

>vignette()

通过加入包名，你可以查看某个包的小品文:

>vignette(package="packagename")

每个小品文都有一个名字，可以使用名字查看该小品文：

>vignette("*vignettename*")

在 R 中，使用 RsiteSearch 函数，通过关键词或短语来查找：

>RSiteSearch("*key phrase*")

在你的浏览器中，尝试使用这些站点来查找：

http://rseek.org

这是一个谷歌自定义搜索，它专注于 R-specific 网站。

http://stackoverflow.com/

StackOverflow 是一个可查找问题和答案，面向编程主题，比如数据结构、编码、图形。

http://stats.stackexchange.com/

Stack Exchange 在统计分析领域也是一个可查找问题及答案的站点，但是它比面向编程而言，更面向统计。

有时，你仅仅有一个通用的兴趣-比如贝叶斯分析，计量经济学，最优化，或者图形。CRAN 包含一套任务视图页面描述了可能有用的包。一个任务视图是一个伟大的起点，因了你对什么是可用的有了大概的理解。你可以在 *http://cran.r-project.org/web/views/*查看任务列表或按解决方案中所描述的那样查找它们。假如你碰巧知道了一个有用包 say 的名字，通过在线提醒注意到它。一个完整的，按字母顺序整理的包的列表在 *http://cran.r-project.org/web/packages/*是可用的，可通过相应链接查看汇总页面。

另请参阅

你可以下载并安装 sos 包，它提供了强有力的其它查找包的方式，在 *http://cran.r-project.org/web/packages/sos/vignettes/sos.pdf* 看该小品文。

@打印函数

cat 函数是 print 的另一种方法，它让你组合多个对象到一个连续的输出中：

```
>cat("The zero occurs at", 2*pi, "radians.", "\n")
```

The zero occurs at 6.283185 radians.

注意 cat 在对象间默认地旋转了一个空格符。你应该提供一个换行符(\n)来结束本行。

cat 函数也能打印简单的向量：

>fib <- c(0,1,1,2,3,5,8,13,21,34)

>cat("The first few Fibonacci numbers are:", fib, "...\n")

The first few Fibonacci numbers are: 0 1 1 2 3 5 8 13 21 34 ...

然而，一个严格的限定条件是，它不能打印复合数据结构，如矩阵和列表。如果非要这么做，则会产生一个错误：

>cat(list("a","b","c"))

Error in cat(list(...), file, sep, fill, labels, append) :

argument 1 (type 'list') cannot be handled by 'cat'

@类型转换

这些模式是不兼容的。为了从它们中产生向量，R 将 3.1415 转换为字符模式，这样它就能和”foo”兼容了：

>c(3.1415, "foo")

[1] "3.1415""foo"

>mode(c(3.1415, "foo"))

[1] "character"

@R 向量、索引及相关运算基础

cor 和 cov 函数可以分别计算两个向量间的相关性和协方差:

```
>x <- c(0,1,1,2,3,5,8,13,21,34)
>y <- log(x+1)
>cor(x,y)
[1] 0.9068053
>cov(x,y)
[1] 11.49988
```

所有这些函数对值都很挑剔。甚至向量中的一个 NA 值都会引起这些函数中的任何一个返回 NA 或者甚至会生一个隐藏的错误而终止算法：

```
>x <- c(0,1,1,2,3,NA)
>mean(x)
[1] NA
>sd(x)
[1] NA
```

R 像这样小心，还真让人受不了，但它这样做是正确的。你应该仔细考虑你的情形。NA 在你的数据中是无效数据吗？如果是，R 这样做是对的。如果不是，你可以通过设置 na.rm=TRUE 来重写该行为，它将会告诉 R 忽略掉 NA 值：

```
>x <- c(0,1,1,2,3,NA)
>mean(x, na.rm=TRUE)
[1] 1.4
>sd(x, na.rm=TRUE)
[1] 1.140175
```

>mean(dframe)

small medium big

1.245040 9.560325 99.946003

>sd(dframe)

small medium big

0.5844025 0.9920281 0.8135498

>var(dframe)

small medium big

small 0.34152627 -0.21516416 -0.04005275

medium -0.21516416 0.98411974 -0.09253855

big -0.04005275 -0.09253855 0.66186326

同样地，如果 x 是一个数据框或者一个矩阵，那么 cor(x)将会返回相关性矩阵，cor(x)将会返回协方差矩阵：

>cor(dframe)

small medium big

small 1.00000000 -0.3711367 -0.08424345

medium -0.37113670 1.0000000 -0.11466070

big -0.08424345 -0.1146607 1.00000000

>cov(dframe)

small medium big

small 0.34152627 -0.21516416 -0.04005275

medium -0.21516416 0.98411974 -0.09253855

big -0.04005275 -0.09253855 0.66186326

对于一个增量不是 1 的序列，使用 seq 函数：

>seq(from=1, to=5, by=2)

[1] 1 3 5

使用 rep 函数创建一系列重复值：

>rep(1, times=5)

[1] 1 1 1 1 1

另一种方法，你可以对输出序列指定一个长度，然后 R 会计算必要的增量：

>seq(from=0, to=20, length.out=5)

[1] 0 5 10 15 20

>seq(from=0, to=100, length.out=5)

[1] 0 25 50 75 100

>fib[-1] #忽略掉第一个元素

[1] 1 1 2 3 5 8 13 21 34

>fib[1:3] # 像以前一样

[1] 0 1 1

>fib[-(1:3)] #索引的求反运算，用于排除而不是选择

[1] 2 3 5 8 13 21 34

选择所有比中位数大的元素

v[v > median(v)]

选择所有上下 5%之外的元素

v[(v < quantile(v,0.05)) | (v > quantile(v,0.95))]

选择超过平均值正负两位标准差范围之外的所有元素

v[abs(v-mean(v)) > 2*sd(v)]

选择为 NA 或为 NULL 的所有元素

v[!is.na(v) & !is.null(v)]

一个最终的索引特性让你通过名字选择元素。假设向量有一个 names 属性，为每一个元素定义一个名字。通过把一个字符串的向量赋值给该属性可以实现：

>years <- c(1960, 1964, 1976, 1994)

>names(years) <- c("Kennedy", "Johnson", "Carter", "Clinton")

>years

Kennedy Johnson Carter Clinton

1960 1964 1976 1994

一旦名字被定义，你便可以通过名字引用单独的某个元素：

>years["Carter"]

Carter

1976

>years["Clinton"]

Clinton

1994

%%

求模运算符

%/%

整除运算符

%*%

矩阵乘法

%in%

如果左边的值发生在右边，返回为 TRUE，否则返回 FALSE

@查看与卸载包

>history(100) # Show 100 most recent lines of history

>history(Inf) # Show entire saved history

没有参数的情况下，search 函数返回加载包的列表。它会产生类似如下的输出：

>search()

[1] ".GlobalEnv""package:stats""package:graphics"

[4] "package:grDevices""package:utils""package:datasets"

[7] "package:methods""Autoloads""package:base"

detach 函数将卸载当前加载的一个包：

>detach(package:MASS)

library 函数，在没有参数的情况下，将打印已经安装的所有包列表。该列表可能会很长。

在 linux 电脑上，这些也许只是输出的前几行：

>library()

Packages in library '/usr/local/lib/R/site-library':

boot Bootstrap R (S-Plus) Functions (Canty)

CGIwithR CGI Programming in R

class Functions for Classification

cluster Cluster Analysis Extended Rousseeuw et al.

DBI R Database Interface

expsmooth Data sets for "Forecasting with exponential

smoothing"

. *(etc.)*

@查看包中的数据集

通过使用伴有一个 package 参数（不需要数据集名称）的 data 函数，你能看到 MASS 或者任意的包中可用的数据集列表：

>data(package="*pkgname*")

>installed.packages()[,c("Package","Version")]

Package Version

acepack "acepack""1.3-2.2"

alr3 "alr3""1.0.9"

base "base""2.4.1"

boot "boot""1.2-27"

bootstrap "bootstrap""1.0-20"

calibrate "calibrate""0.0"

car "car""1.2-1"

chron "chron""2.3-12"

class "class""7.2-30"

cluster "cluster""1.11.4"

. *(etc.)*

@选择镜像

1. 调用 chooseCRANmirro 函数：

>chooseCRANmirror()

R 将显示 CRAN 镜像的列表。

2.从列表中选择一个 CRAN 镜像，并且按 OK

3. 通过查看 repos 选项的第 1 个元素，获取该镜像的 URL:

>options("repos")[[1]][1]

4. 向.Rprofile 文件中增加该行:

options(repos="*URL*")

这里的 URL 是镜像的 URL

@R 脚本命令

使用 CMD BATCH 子命令，给出脚本名称与输出文件名称，运行 R 程序：

$ R CMD BATCH scriptfile outputfile

如果你想输出结果到标准输出或者你需要将命令行参数传到脚本中，考虑使用 Rscript 命令：

$ Rscript *scriptfile arg1 arg2 arg3*

输出写向标准输出，当然，这里 R 从调用 shell 脚本中继承。使用普通的定向，你能重定向输出到一个文件：

$ Rscript --slave myScript.R arg1 arg2 arg3 >results.out

>Sys.getenv("SHELL")

SHELL

"/bin/bash"

>Sys.setenv(SHELL="/bin/ksh")

@R 启动时都发生了什么？

这里简要地给出了当 R 启动时所发生的情况（键入 help(Startup)来查看详情）：

1. R 执行 Rprofile.site 脚本。这是网站级别的脚本，可以让系统管理员重写位置的默认选项。该脚本的路径为 R_HOME*/etc/Rprofile.site*（R_HOME 是 R 的主目录）

2. R 在当前目录中执行.Rprofile 脚本；或者，如果该文件不存在，执行你的主目录里

的.Rprofile 脚本。用户可以基于该脚本来定制 R。在主目录中的.Rprofile 脚本用于全局定制。在低级别目录中的.Rprofile 脚本，在 R 启动时，能够执行确定的定制；比如，当在某个确定项目的文件夹启动时，实现 R 的定制。

3.R 加载工作空间保存在.Rdata，若该文件存在于工作目录中。从该文件中重新加载你的工作空间，恢复对你临时变量和函数的访问。

4. 如果你定义了一个.First 函数，R 会执行。该函数对用户或项目来说是个定义启动初始化代码有用的地方。你可以在你的.Profile 或工作空间中定义它。

5. R 执行.First.sys 函数。本步会加载默认的包。该函数是 R 内建的，不会根据不同的用户或管理员而改变。观察发现 R 并不会加载默认的包，直到最后一步，当它执行了.First.sys 函数。在它之前，只有基础包被加载了。这是一个重要的事实，因为它意味着先前的步骤并不能假设除了基础包之外的包是可用的。它也解释了为什么在你的.Rprofile 脚本中尝试打开一个图形窗口会失败：因为 graphics 包还没有加载。

@输出重定向

你可以使用 file 参数，对 cat 的输出重定向：

>cat("The answer is", answer, "\n", file="*filename*")

使用 sink 函数对来自 print 和 cat 的所有输出进行重定向。使用一个文件名参数调用 sink 开始重定向控制台输出到文件。当你完成后，使用不带参数的 sink 函数关掉该文件，并且恢复输出到控制台：

>sink("*filename*") # Begin writing output to file

. . . *other session work* . . .

>sink() # Resume writing output to console

>sink("script_output.txt") # Redirect output to file

>source("script.R") # Run the script, capturing its output

>sink() # Resume writing output to console

@显示当前目录文件

list.files 函数显示你的当前目录下的内容：

>list.files()

>list.files(all.files=TRUE)

@mode 函数

这些对象都有一个”numeric”模式，因为它们作为一个数字存储；但是它们有不同的类型来解释。比如，一个 Date 对象由一个数字组成：

>d <- as.Date("2010-03-15")

>mode(d)

[1] "numeric"

>length(d)

[1] 1

@在向量中插值

新值将会被插入在由 after 所确定的位置上。比如，在一个序列的中间，插入 99：

>append(1:10, 99, after=5)

[1] 1 2 3 4 5 99 6 7 8 9 10

after=0 的特殊值意味着在向量的最前面插入新值：

>append(1:10, 99, after=0)

[1] 99 1 2 3 4 5 6 7 8 9 10

@stack 函数用法

下图展示了 stack 函数的用法：

```
> a
[1] 3 4 5 3
> b
[1] 3 2
> c
[1] 10 20 40 20 28 40
> stack(list(a=a,b=b,c=c))
   values ind
1       3   a
2       4   a
3       5   a
4       3   a
5       3   b
6       2   b
7      10   c
8      20   c
9      40   c
10     20   c
11     28   c
12     40   c
>
```

@do.call 使用 list 参数执行函数

把每一行存为一行的数据框。然后，再把这些一行的数据框保存进一个列表。使用 rbind 和 do.call 函数将行绑定进一个大的数据框中：

>dfrm <- do.call(rbind, obs)

@subset 取子集

使用 subset 函数。该 select 参数是一个字段名，或者需要选择的字段名称的向量：

>subset(dfrm, select=*colname*)

>subset(dfrm, select=c(*colname*1, ..., *colnameN*))

注意到，这里并没有对字段名加上引号。

该 subset 参数是一个选择行的逻辑表达式。在该表达式中，你可以把字段名称作逻辑表达式的一部分。在本例中，response 是数据框的一个字段，并且我们正在选择 response 为正的行：

>subset(dfrm, subset=(response > 0))

当你组合 select 和 subset 参数时，subset 是最有用的：

>subset(dfrm, select=c(predictor,response), subset=(response > 0))

@对每一个 List 元素应用一个函数

根据希望的结果形式，使用 lapply 或 sapply 函数。lapply 总是以 list 的形式返回结果，然而 sapply 如果可能的话，会以向量的形式返回结果：

>lst <- lapply(*lst*, *fun*)

>vec <- sapply(*lst*, *fun*)

@对每一行应用一个函数

>long

trial1 trial2 trial3 trial4 trial5

Moe -1.8501520 -1.406571 -1.0104817 -3.7170704 -0.2804896

Larry 0.9496313 1.346517 -0.1580926 1.6272786 2.4483321

Curly -0.5407272 -1.708678 -0.3480616 -0.2757667 -1.2177024

对所有行，你可以通过应用 mean 函数为每行计算平均的观察值。结果是一个向量：

>apply(long, 1, mean)

Moe Larry Curly

-1.6529530 1.2427334 -0.8181872

@对每一列应用一个函数

>results <- apply(mat, 2, fun)

@对数据的分组应用一个函数

>tapply(pop,county,sum)

Cook DuPage Kane Kendall Kenosha Lake(IL) Lake(IN) Will

3281966 147779 266269 106221 90352 91452 185794 70834

下一个例子按县计算平均人口：

>tapply(pop,county,mean)

Cook DuPage Kane Kendall Kenosha Lake(IL) Lake(IN) Will

468852.3 147779.0 133134.5 106221.0 90352.0 91452.0 92897.0 70834.0

@对并行的向量或列表应用一个函数

使用 mapply 函数。它将会对你的参数元素，应用函数 f：

>mapply(f, *vec*1, *vec2*, ..., *vecN*)

函数 f 希望每个参数都是一个向量。如果向量参数的长度不一致 ,将会应用 Recycling Rule。

lapply 函数对 list 参数也有效：

>mapply(f, *list*1, *list*2, ..., *listN*)

假如你有四个测试站和三个治疗方案：

>locations <- c("NY", "LA", "CHI", "HOU")

>treatments <- c("T1", "T2", "T3")

我们可以应用 outer 和 paste 来产生测试站和治疗方案的所有组合：

>outer(locations, treatments, paste, sep="-")

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

[1,] "NY-T1""NY-T2""NY-T3"

[2,] "LA-T1""LA-T2""LA-T3"

[3,] "CHI-T1""CHI-T2""CHI-T3"

[4,] "HOU-T1""HOU-T2""HOU-T3"

@产生可重现的随机数

>set.seed(165)

>runif(10) # 产生 10 个随机数

[1] 0.1159132 0.4498443 0.9955451 0.6106368 0.6159386 0.4261986 0.6664884

[8] 0.1680676 0.7878783 0.4421021

>set.seed(165) # 初始化到相同的已知状态

>runif(10) #产生相同的 10 个随机数

[1] 0.1159132 0.4498443 0.9955451 0.6106368 0.6159386 0.4261986 0.6664884

[8] 0.1680676 0.7878783 0.4421021

对一个向量，产生倒序：

```
> m<-c(34,23,20,94,3)
> rev(m)
[1]  3 94 20 23 34
> 
```

@为一个均值产生置信区间

>x <- rnorm(50, mean=100, sd=15)

>t.test(x)

One Sample t-test

data: x

t = 59.2578, df = 49, p-value < 2.2e-16

alternative hypothesis: true mean is not equal to 0

95 percent confidence interval:

97.16167 103.98297

sample estimates:

mean of x

100.5723

>t.test(x, conf.level=0.99)

One Sample t-test

data: x

t = 59.2578, df = 49, p-value < 2.2e-16

alternative hypothesis: true mean is not equal to 0

99 percent confidence interval:

96.0239 105.1207

sample estimates:

mean of x

100.5723

使用 wilcox.test 函数，设置 conf.int=TRUE:

>wilcox.test(x, conf.int=TRUE)

该输出会包含中位数的置信区间

@测试样本比例

使用 prop.test 函数，假设样本的大小为 n，并且样本包含 x 个成功的：

>prop.test(x, n, p)

该输出包含一个 p 值。依惯例，p 值小于 0.05，表明真正的比例不可能是概率 p。然而 p 值超过 0.05 不能提供这样的证据。

>prop.test(11, 20, 0.5, alternative="greater")

1-sample proportions test with continuity correction

data: 11 out of 20, null probability 0.5

X-squared = 0.05, df = 1, p-value = 0.4115

alternative hypothesis: true p is greater than 0.5

95 percent confidence interval:

0.3496150 1.0000000

sample estimates:

p

0.55

@正态性检验

>shapiro.test(x)

Shapiro-Wilk normality test

data: x

W = 0.9651, p-value = 0.4151

@游程检验

你的数据是一个二值序列：yes-no，0-1，true-false，或其它的二值数据。你想知道：该序列随机吗？ tseries 包包含 runs.test 函数，它用于检验一个序列的随机性。该序列应该是有两个水平的因子：

>library(tseries)

>runs.test(as.factor(s))

Runs.test 函数给出了一个 p 值。依惯例，p 值小于 0.05 表明该序列并不随机；然而，当 p

值大于 0.05 又提供不了这样的证据。

>library(tseries)

>s <- sample(c(0,1), 100, replace=T)

>runs.test(as.factor(s))

Runs Test

data: as.factor(s)

Standard Normal = 0.2175, p-value = 0.8279

alternative hypothesis: two.sided

@比较两个样本的均值

>t.test(x, y)

>t.test(x, y, paired=TRUE)

@以非参的方式比较两个样本的位置

你有来自两个总体的两个样本。你不知道总体的分布，但是你知道它们有相似的形状。你想知道：一个总体对另一个总体来说是偏左还是偏右呢？

你可以使用一个非参检测，Wilcoxon–Mann–Whitney 检验，它是通过 wilcox.test 函数实现的。对于配对样本，设置 paired=TRUE:

>wilcox.test(x, y, paired=TRUE)

对于非配对样本，设置 paired=FALSE:

>wilcox.test(x, y)

该数据是配对的，所以我们设置 paired=TRUE:

>wilcox.test(fav, unfav, paired=TRUE)

Wilcoxon signed rank test with continuity correction

data: fav and unfav

V = 0, p-value < 2.2e-16

alternative hypothesis: true location shift is not equal to 0

该 p 值本质上是 0. 从统计角度说，我们拒绝完成时间相同的假设。实际上来说，作出时间是不同的结论是合理的。

@相关性的显著性检验

你计算出了两个变量之间的相关性，但是你并不知道是否该相关性具有统计学意义。

Cor.test 函数可以计算 p 值和相关性的置信区间。如果变量来自正态分布总体，那么使用默认的 Pearson 相关性度量：

>cor.test(x, y)

对于非正态总体，使用 Spearman 方法替换：

>cor.test(x, y, method="Spearman")

该函数返回了几个值，包括来自于显著性检验的 p 值。依惯例，p<0.05 表明相关性有可能是显著的，然而 p>0.05 而不显著。

@等比例分组检验

你有来自两个或更多分组的样本。分组的元素都是二元值。使用 prop.test 函数，需要两个作为参数：

>ns <- c(*ns*1, *ns*2, ..., *nsN*)

>nt <- c(*nt*1, *nt*2, ..., *ntN*)

>prop.test(ns, nt)

这些都是平行的向量。第一个向量 ns 给出了每个组成功的数量。第二个向量 nt 给出了相应组的大小。输出包含一个 p 值。依惯例，p<0.05 表明可能分组的比例是不同的；然而，当 p>0.05 时，则不能提供这方面的证据。

@分组均值之间进行两两比较

将所有数据放到一个向量中，创建一个并行的因子来区分这些组。使用 pairwise.test 来执行均值的成对比较：

>pairwise.t.test(x,f) # x 包含数据，f 是分组因子

该输出包含一个 p 值的表，一个 p 值对应一个分组对。依惯例如果 p<0.05 那么两个分组间可能有相同的均值；然而，p>0.05 不能提供这样的证据。

>pairwise.t.test(comb$values, comb$ind)

Pairwise comparisons using t tests with pooled SD

data: comb$values and comb$ind

fresh jrs

jrs 0.0043 -

soph 0.0193 0.1621

P value adjustment method: holm

@检验两个样本是否来自同一个分布

你有两个样本，你想问：它们来自于相同的分布吗？

Kolmogorov–Smirnov 检验比较两个样本，检验它们是否来自于相同的分布。Ks.test 函数实现了该检验：

>ks.test(x, y)

该输出包包含一个 p 值。依惯例，p<0.05 表明两个样本（x 和 y）来自于不同的分布；然而，p>0.05 提供不了这样的证据。

@图形函数笔记

理解高级与低级绘图的区别是重要的。一个高级绘图函数开始一个新的画布。它初始化图形窗口（需要的时候创建它）；设置标度；也许会设置一些标题、标签之类的，然后添加到画布中。例子包括：

plot

通用绘图函数

boxplot

创建一个箱线图

hist

创建一个直方图

qqnorm

创建一个分位数-分位数（Q-Q）图

curve

函数图形

低级绘图函数不能开始一个新的画布。更确切地说，它向一个已存在的画布中增加一些东西：点、线、文本等等。例子包括：

points

增加点

lines

增加线

abline

增加一条直线

segments

增加线段

polygon

增加一个多边形

text

增加文本

比如 Zoo 包，实现了一个时间序列对象。如果你创建了一个 zoo 对象，并且调用 plot(z)，那么 zoo 包会画出来；它创建了为呈现时间序列而定制的图形。

对于传统绘图来说，lattice 是另一个包。它使用了一个更容易的绘图范式让你创建信息图变得更容易。结果，通常也更好看。

ggplot2 包也提供了另外一种绘图范式，它调用了绘图的语法。它使用了高级模块方法来绘图，这样使用构建与定制你的图形更加容易。这些图，通常也比传统的图更具吸引力。

为点添加图例

legend(x, y, labels, pch=c(pointtype1, pointtype2, ...))

根据线的类型，为线添加图例

legend(x, y, labels, lty=c(linetype1, linetype2, ...))

根据线的宽度，为线添加图例

legend(x, y, labels, lwd=c(width1, width2, ...))

为颜色添加图例

legend(x, y, labels, col=c(color1, color2, ...))

@在条形图中添加置信区间

你想在条形图中，增加置信区间。假设 x 是一个统计向量，比如均值的向量，并且 lower 和 upper 是对应置信区间界限的向量。gplots 库的 barplot2 函数能够呈现 x 的条形图和它的置信区间：

>library(gplots)

>barplot2(x, plot.ci=TRUE, ci.l=lower, ci.u=upper)

改变线条的类型、宽度和颜色

•lty="solid"or lty=1 (default)

•lty="dashed"or lty=2

•lty="dotted"or lty=3

•lty="dotdash"or lty=4

•lty="longdash"or lty=5

•lty="twodash"or lty=6

•lty="blank"or lty=0 (inhibits drawing)

本调用用于画图，比如，用虚线的样式画线：

>plot(x, y, type="l", lty="dashed")

使用 lwd 参数来控制线的宽度和厚度。默认值为 1。

>plot(x, y, type="l", lwd=2) # Draw a thicker line

使用 col 参数来控制线条颜色。默认为黑色。

>plot(x, y, type="l", col="red") # Draw a red line

@向直方图中添加密度估计

你有一个人数据样本的直方图，并且你想增加一条曲线来说明整体的密度。

使用 density 函数来近似样本的密度；然后使用 lines 函数来画出该近似值：

>hist(x, prob=T) # Histogram of x, using a probability scale

>lines(density(x)) # Graph the approximate density

直方图可以看出你数据的密度函数，但它太粗糙。一个平滑估计可以帮助你更好地可视化潜在分布。本例从伽马分布中取了一个样本，然后画出直方图，并且估计密度：

>samp <- rgamma(500, 2, 2)

>hist(samp, 20, prob=T)

>lines(density(samp))

@获得回归统计量

你想要获得关于回归的关键统计量和信息，比如 R^2，F 统计量，系数的置信区间，残差，方差分析表等等。

将回归模型保存在一个变量当中，比如 m：

>m <- lm(y ~ u + v + w)

然后使用函数抽取模型的回归统计量和信息：

anova(m)

方差分析表

coefficients(m)

模型系数

coef(m)

和 coefficients(m)一样

confint(m)

回归系数的置信区间

deviance(m)

残差平方和

effects(m)

正交效应的向量

fitted(m)

拟合 y 值的向量

residuals(m)

模型残差

resid(m)

和 residuals(m)一样

summary(m)

关键统计量，如 R^2, F 统计量和残差标准误（σ）

vcov(m)

主要参数的方差-协方差矩阵

@数据子集的回归

你想要对你的数据子集拟合一个线性模型，而不是整个数据集。

lm 函数有一个 subset 参数，指定了哪些数据元素应该用于拟合。该参数的值可以是任何的能够索引你数据的索引表达式。本例给出了仅前 100 个观察的拟合：

>lm(y ~ x, subset=1:100) # Use only x[1:100]

@在回归公式中使用表达式

你想要在计算后的值上进行回归，而不仅仅是变量上回归。但是回归公式的语法貌似不允许这样。

为计算后的值嵌套表达式在 I(...)运算符中。这样将强迫 R 计算该表达式并使用计算后的值回归。

如果你要在 u 和 v 的和上进行回归，那么你的表达式应该是这样的：

$yi = \beta 0 + \beta 1(ui + vi) + \varepsilon i$

那么，如何将该方程写成回归公式呢？本例不会有效：

>lm(y ~ u + v) # Not quite right

这里 R 将会把 u 和 v 翻译成两个单独的预测变量，每一个变量使用自己的表达式系数。同样地，假假如你的回归方程是：

$yi = \beta 0 + \beta 1 ui + \beta 2 ui$

$2 + \varepsilon i$

下面的公式也不会有效:

>lm(y ~ u + u^2) # 这是一个交互，而不是二次项

R 将会把 u^2 翻译成交互项，而不是 u 的平方。解决方法是在表达式外面加上 I(...) 运算符，它阻止表达式被翻译成回归公式。反而，它会强迫 R 计算表达式的值，然后吸收那个值直接用于回归。

第一个例子变成了：

>lm(y ~ I(u + v))

在如上命令中，R 计算 u+v 的和，再在和上，对 y 进行回归。

第二个例子，我们使用：

>lm(y ~ u + I(u^2))

这里 R 计算 u 的平方，并在 u+u^2 的和上进行回归。

>m <- lm(y ~ u + I(u^2))

>predict(m, newdata=data.frame(u=13.4)) # R plugs u and u^2 into the calculation

1

11531.59

@多项式回归

你想要在 x 的多项式上对 y 进行回归。在你的回归表达式中使用 poly(x,n)函数在 x 的 n 项式上进行回归。本例把 y 作为 x 的三次函数进行建模。

>lm(y ~ poly(x,3,raw=TRUE))

本例中的表达式与如下的三次回归方程相对应：

$yi = \beta 0 + \beta 1xi + \beta 2xi$

$2 + \beta 3xi$

$3 + \varepsilon i$

它更容易写:

>m <- lm(y ~ poly(x,3,raw=TRUE))

raw=TRUE 是必要的. 没有它，poly 函数计算正交的多项式而不是简单多项式。除了方便之外，一个巨大的优势是，当你从模型做预测时，R 会计算 x 的所有这些次方。没有它，每当你采用模型时，你都得自己计算 x2 和 x3。

这里有另外一个使用 poly 的好的理由。你不能以这样的方式写你的回归表达式：

>lm(y ~ x + x^2 + x^3) #并不会按照你想的方式做！

R 将会把 x^2 和 x^3 解释为交互项，而不是 x 的次方。由此产生的模型是一项线性回归，完全不像你所期望的那样。你应该像这样写回归表达式：

>lm(y ~ x + I(x^2) + I(x^3))

但这样显得十分多余。直接使用 poly。

@发现最佳的次方转换（Box-Cox 程序）

你想要通过对响应变量应用一个次方转换来改进线性模型。使用 box-cox 程序，它是通过 MASS 包中的 boxcox 函数实现的。该程序将指定一个次数，λ，如此以致将 y 转换成 y^{λ}来改进模型的拟合度：

```
>library(MASS)
>m <- lm(y ~ x)
>boxcox(m)
```

boxcox 函数将创建一个图表，画出的λ值及对应的产生模型的对数似然函数。我们想最大化对数似然函数，这样该函数在最佳值处画了一条线，同时在置信区间的边界也画了线。在本例中，看起来最佳值在-1.5 左右，这里的置信区间大约在$(-1.75, -1.25)$。

奇怪的是，boxcox 函数并不会返回最佳值处的λ值。而是，返回显示在图中的(x,y)值对。可见，发现最大对数似数值 y 对应的λ值是相当容易的。我们使用 which.max 函数：

```
>which.max(bc$y)
[1] 13
```

然后该步给出了对应λ的位置:

```
>lambda <- bc$x[which.max(bc$y)]
>lambda
[1] -1.515152
```

@画出回归的残差

你想要你的回归残差的可视化展现。

你通过从可用的图表中选择残差图来画出模型对象：

>m <- lm(y ~ x)

>plot(m, which=1)

通常 ,画出一个回归模型对象会产生几个诊断图表。你可以通过指定 which=1 选择残差图。

@诊断线性回归

你已经执行了一个线性回归。现在你想通过运行诊断检查来核实模型质量。

首先绘制模型对象，这步将产生几个诊断图。

>m <- lm(y ~ x)

>plot(m)

接着，通过观察残差的诊断图或者使用 car 包的 outlier.test 来区别可能的离群值。

>library(car)

>outlier.test(m)

最终，找到任何过度影响的观测。

@识别出影响点

你想识别出对回归模型有最强影响的观测。这对使用数据诊断可能的问题是有用的。

influence.measures 函数提供了对识别影响观测有用的几个统计量 ,并且它显示地用*进行

标记。该函数的主要参数是你的回归模型对象：

>influence.measures(m)

@自相关的残差检验(Durbin–Watson 检验)

你已经执行了一个线性回归，并且想检验自相关的残差。Durbin—Watson 检验能够检查自相关的残差。该检验通过 lmtest 包的 dwtest 函数实现。

>library(lmtest)

>m <- lm(y ~ x) #创建一个模型对象

>dwtest(m) # 检验模型的残差

输出包含 p 值。依惯例，如果 p<0.05 那么残差显著相关，反之，p>0.05 不是提供相关的证据。通过画残差的自相关函数(ACF)，你可以对自相关执行一个可视化的检查。

>acf(m) # 画模型残差的 ACF

这是建立简单线性回归模型，然后检验自相关的残差的第一个例子。检验返回的 p 值为 0.07755，这说明并不显著相关：

>m <- lm(y ~ x)

>dwtest(m)

Durbin–Watson test

data: m

DW = 1.5654, p-value = 0.07755

alternative hypothesis: true autocorrelation is greater than 0

第二个例子在残差中展示了自相关。P 值为 0.01298，如此自相关是肯定的：

>m <- lm(y ~ z)

>dwtest(m)

Durbin–Watson test

data: m

DW = 1.2946, p-value = 0.01298

alternative hypothesis: true autocorrelation is greater than 0

@产生预测区间

你正使用一个线性回归模型作预测。你想要知道预测的区间 :预测分布的范围。使用 predict 函数，并且指定 interval="prediction":

>predict(m, newdata=preds, interval="prediction")

@执行单因素方差分析

你的数据被分成组，并且组是正态分布的。你想要知道这些组是否有明显不同的均值。使用一个因子来定义组。然后应用 oneway.test 函数：

>oneway.test(x ~ f)

这里，x 是一个数值型的向量，且 f 是一个能够区分组别的因子。输出包含一个 p 值。依惯例，当 p 值<0.05 时表明两个或更多的组别具有显著不同的均值；反之，当 p>0.05 却提供不了这样的证据。

方差分析的基本假设是你的数据是正态分布的或者至少相当地接近钟形。否则使用 Kruskal–Wallis 检验。

注意到 oneway.test 输出说“(没有假设等方差)”。如果你知道各分组具有相等的方差，通过指定 var.equal=TRUE 你将获得一个更少的保守检验。

>oneway.test(x ~ f, var.equal=TRUE)

你也可以像如下这样使用 aov 函数来执行单因素方差分析：

>m <- aov(x ~ f)

>summary(m)

但是，aov 函数总是假设相等的方差，所以它没有 oneway.test 那么灵活。

@创建一个交互图

你正在执行多因素方差分析：使用两个或更我的分类变量作为预测变量。你想要一个在两个预测变量间可能交互的可视化检验。使用 interaction.plot 函数：

>interaction.plot(pred1, pred2, resp)

这里 pred1 和 pred2 是两个分类预测变量，resp 是响应变量。

Faraway 包包含一个 rats 数据集。在该数据集中，treat 和 poison 是分类变量，且 time 是响应变量。当绘制 poison 和 time 的关系时，我们正在寻找直的平行线，它可以表示线性关系。然而使用 interaction.plot 函数揭示了有些事情并不是对的：

>library(faraway)

>data(rats)

>interaction.plot(rats$poison, rats$treat, rats$time)

@在各分组的均值间找到差异

你的数据被分成很多组，一个方差检验表示这些分组有显著不同的均值。你想知道对所有分组这些均值的不同。

使用 aov 函数执行方差检验，会返回一个模型对象。将该模型对象应用到 TukeyHSD 函数中：

>m <- aov(x ~ f)

>TukeyHSD(m)

这里，x 是你的数据，y 是分组因子。你可以画出 TukeyHSD 的结果以获取差异的可视化呈现：

>plot(TukeyHSD(m))

方差检验是重要的，因为无论分组的均值是否不同，它都会告诉你。但是该检验并不会区别哪一个分组是不同的，并且它也不会报告他们之间的不同。TukeyHSD 函数可以计算这些差异 ,辅助你区别最大的。它使用由 John Tukey 发明的 “honest significant differences” 方法。

@执行稳健方差分析（Kruskal–Wallis 检验）

你的数据被分成组。这些组并不是正态分布，但是它们的分布具有相似的形状。你想要执行一个与方差分析相似的检验-你想要知道这些分组的中位数是否有显著的区别。

创建一个定义数据分组的因子。使用 kruskal.test 函数，它实现了 Kruskal–Wallis 检验。不像方差检验，该检验不会依赖数据的正态性：

>kruskal.test(x ~ f)

这里，x 是数据向量，f 是分组因子。输出包含一个 p 值。依惯例，当 p<0.05 时，表明两个或更多分组的中位数具有显著差异。反之，当 p>0.05 时，却提供不了这种证据。

正常的方差分析假设数据满足正态分布。它能够容忍一些偏差，但是极大的偏差将产生无意义的 p 值。Kruskal–Wallis 检验有一个方差分析的非参版本，也就是说它没有正态性的假设。然而，它假设一样形状的分布。当你的数据分布是非正态或未知的情况下，你应该使用 Kruskal–Wallis 检验。

@使用方差分析比较模型

你有相同数据的两个模型，你想要知道是否它们产生不同的结果。Anova 函数可以比较两个模型，并且可以告诉是否它们显著地不同：

>anova(m1, m2)

这里 m1 和 m2 是由 lm 返回的两个模型对象。Anova 的输出包含一个 p 值。依惯例，p<0.05 表明模型显著不同，反之 p>0.05 则提供不同这样的证据。

@汇总行和列

你想要汇总某个矩阵或数据框的行和列。使用 rowSums 函数汇总行：

>rowSums(m)

使用 colSums 汇总列：

>colSums(m)

@找到特殊值的位置

你有一个向量。你知道内容中的一个特殊值，你想要知道它的位置。Match 函数将在向量中查找特殊值，并返回它的位置：

```
>vec <- c(100,90,80,70,60,50,40,30,20,10)
>match(80, vec)
[1] 3
```

这里 match 返回 3，它是 vec 中 80 的位置。

R 中有用于找出最大最小值位置的函数—which.min 和 which.max：

```
>vec <- c(100,90,80,70,60,50,40,30,20,10)
>which.min(vec) # Position of smallest element
[1] 10
>which.max(vec) # Position of largest element
[1] 1
```

@找出成对的最小值或最大值

你有两个向量，v 和 w，你想要在这成对的元素中，找最小值或最大值。也就是，你想要计算：

min($v1$, $w1$), min($v2$, $w2$), min($v3$, $w3$), ...

或者:

max($v1$, $w1$), max($v2$, $w2$), max($v3$, $w3$), ...

分别使用 pmin(v,w)和 pmax(v,w)如下：

>pmin(1:5, 5:1) # Find the element-by-element minimum

[1] 1 2 3 2 1

>pmax(1:5, 5:1) # Find the element-by-element maximum

[1] 5 4 3 4 5

@产生所有几个因子的组合

你有两个或多个因子。你想要产生它们的所有组合，也就是笛卡儿乘积。使用 expand.grid 函数。这里 f 和 g 是两个因子：

>expand.grid(f, g)

@按两个字段排序

你想要使用两个字段作为排序键，对数据框的内容进行排序。通过两个字段排序与通过一个字段排序相似。Order 函数接受多个参数，这样我们给它两个排序键：

>dfrm <- dfrm[order(dfrm$key1,dfrm$key2),]

@展现一个对象的结构

调用 calss 函数，查看对象所属的类：

>class(x)

使用 mode 函数来去除面向对象的特征，揭示潜在的结构：

>mode(x)

使用 str 函数呈现内部结构和内容：

>str(x)

@为你的代码计时

你想要知道运行你的代码需要多少时间。这是很有用的，比如，当你正在优化你的代码，并且需要“before”和”after“数字来度量性能。

System.time 函数将执行你的代码，然后报告异常时间：

>system.time(aLongRunningExpression)

输出是三或五个数字，这依赖于你的平台。前三个通常是最重要的：

User CPU time

运行 R 的 CPU 秒数

System CPU time

运行系统的，典型的如输入/输出的 CPU 的秒数

Elapsed time

时钟的秒数

@禁止提示和错误信息

一个功能正在产生令人厌烦的错误信息或提示信息。你不想看到他们。在该函数外面加上 suppressMessage(...) 或 suppressWarnings(...)：

>suppressMessage(annoyingFunction())

>suppressuWarnings(annoyingFunction())

@从列表中获取函数参数

你的数据在一个列表结构中得到的。你想将数据传给一个函数，但该函数不接受列表作为一个参数。简单的情况，你需要把列表转成向量。对于更复杂的情形，do.call 函数将把列表分成单独的参数，并且调用你的函数：

>do.call(function, list)

@定义你自己的二元运算符

你想要定义你自己的二元运算符，让你的 R 代码更加精简可读。R 识别百分号(%...%)间的任何文本作为一个二元操作符。通过赋值两个参数的函数给它，创建并定义一个新的二元操作符。R 包含一个令人感兴趣的特征，就是可以让你自己定义二元操作符。在百分号(%...%)间的任何的文本都能被 R 作为二元操作符解释。R 预先定义了几个这样的运算符，比如%/%用于整除，%*%用于矩阵相乘。通过赋值一个函数给它，你能够创建一个新的二元运算符。本例将创建一个二元运算符，%+-%：

>'%+-%' <- function(x,margin) x + c(-1,+1)*margin

表达式 x %+-% m 计算 x ±m。这里它计算 100 ±(1.96 × 15)，一个标准 IQ 测试的两倍离差范围：

>100 %+-% (1.96*15)

[1] 70.6 129.4

注意我们在定义的时候引用二元运算符而不是使用的时候。定义你自己运算符的快感在于你

能够把常用的运算以简洁语法的方式包起来。如果你的应用频繁地连接字符串并且没有介入中间的空格，那么你可以为了这个目的定义一个二元连接字符串：

```
>'%+%' <- function(s1,s2) paste(s1,s2,sep="")
>"Hello" %+% "World"
[1] "HelloWorld"
>"limit=" %+% round(qnorm(1-0.05/2), 2)
[1] "limit=1.96"
```

定义你自己运算符的危险，在于代码在其它环境中可移性不是那么强。将函数定义的代码一同引用，否则 R 会报未定义运算符的错误。自定义运算符的优化级比乘法和除法高，但比求幂和序列创建低。如果我们忽略掉引号，那么我们会得到一个意想不到的结果：

```
>100 %+-% 1.96*15
[1] 1470.6 1529.4
```

R 将它作为(100 %+-% 1.96) * 15 解释。

@执行简单的正交回归

你想要使用正交回归创建一个线性模型，在正交回归里面，x 和 y 的方差是齐性的。使用 prcomp 来执行基于 x 和 y 的主成份分析。从产生的旋转计算斜率和截距：

```
>r <- prcomp( ~ x + y )
>slope <- r$rotation[2,1] / r$rotation[1,1]
>intercept <- r$center[2] - slope*r$center[1]
```

@在你的数据中找出聚集

你相信你的数据中包含着聚类。你想要识别出这些聚类。你的数据集 x，可能是一个向量，数据框或矩阵。假设 n 是你期望的聚类数量：

>d <- dist(x) #计算不同观测间的距离

>hc <- hclust(d) #形成层次聚类

>clust <- cutree(hc, k=n) # 把它们组织成 n 个最大的聚类

Clust 是一个从 1 到 n 的数字向量，代表 x 中每一个观测所在的聚类。

@预测一个二元值变量（逻辑回归）

你想要执行一个逻辑回归，逻辑回归模型用于预测二元事件发生的概率。调用 glm 函数使用 family=binomial 设置来执行逻辑回归。结果是一个模型对象：

>m <- glm(b ~ x1 + x2 + x3, family=binomial)

这里，b 是一个两水平因子（比如 TRUE 和 FALSE，0 和 1），x1 和 x2 和 x3 是预测变量。

使用模型对象 m 和 predict 函数来预测来自于新数据的概率：

>dfrm <- data.frame(x1=*value*, x2=*value*, x3=*value*)

>predict(m, type="response", newdata=dfrm)

@使用助法计算一个统计量

你有一个数据集和函数来计算数据集的统计量。你想要估计该统计量的置信区间。使用 boot 包。应用 boot 函数来计算统计量的自助复本：

>library(boot)

```
>bootfun <- function(data, indices) {
+ # . . . calculate statistic using data[indices] . . .
+ return(statistic)
+ }
>reps <- boot(data, bootfun, R=999)
```

这里，data 是原始数据集，它被存储在向量中或数据集中。统计函数(bootfun 在本例中)需要两个参数：在原数据集中的 data 和从 data 中进行自助法抽样的整数向量的索引。下一步，使用 boot.ci 函数从复本中估计一个置信区间：

```
>boot.ci(reps, type=c("perc","bca"))
```

```
>boot.data <- data.frame(x=x, y=y)
>reps <- boot(boot.data, stat, R=999)
```

999 个复本的选择是一个好的起点。你总是能够用更多的值重复自助法，来看是否结果有显著地改变。Boot.ci 函数能够从复本中估计出 CI。它实现了几种不同的算法，type 参数选择执行哪种算法。对每一个选择的算法，boot.ci 将返回产生的估计：

```
>boot.ci(reps, type=c("perc","bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 999 bootstrap replicates
CALL :
boot.ci(boot.out = reps, type = c("perc", "bca"))
Intervals :
Level Percentile BCa
```

95% (1.144, 2.277) (1.123, 2.270)

Calculations and Intervals on Original Scale

@表示时间序列数据

在 CRAN 中查找与 zoo 和 xts 相关的文档；这些文档包含参考手册，小品文以及快捷参考卡片。如果包已经安装在你电脑上，使用 vignette 函数查看它们的文档：

>vignette("zoo")

>vignette("xts")

@构造一个时间序列的子集

你想要从一个时间序列中选择一个或多个元素。你能够通过位置索引到一个 zoo 或 xts 的对象。使用一个或两个下标，依赖于是否该对象包含一个时间序列或多个时间序列：

ts[i]

从单个时间序列中，选择第 i 个观测

ts[j,i]

从多时间序列中选择第 j 个时间序列的第 i 个观测

通过一个日期对象，你能够索引到时间序列对象。使用一样的数据类型作为你时间序列的索引。本例假设索引包含 Date 对象：

>ts[as.Date("*yyyy-mm-dd*")]

你能够通过一个日期的序列来索引它:

>dates <- seq(*startdate*, *enddate*, *increment*)

>ts[dates]

window 函数通过开始和结束日期选择一个范围：

>window(ts, start=*startdate*, end=*enddate*)

xts 包提供了许多其它更便捷的方式来索引一个时间序列。请查看该包的文档。

@合并几个时间序列

你有两个或更多的时间序列。你想把它们合并为一个时间序列对象。使用 zoo 对象来呈现时间序列；然后使用 merge 函数合并它们：

>merge(ts1, ts2)

@计算移动平均

你想要计算时间序列的移动平均。使用 zoo 包的 rollmean 函数来计算 k 周期的移动平均：

>library(zoo)

>ma <- rollmean(*ts*, *k*)

这里 ts 是一个时间序列数据，在 zoo 对象中获取，k 是周期数。对于大多数金融应用，你想用 rollmean 来计算历史数据的均值；也就是说，对于每天，仅使用当天可用的数据。为了实现这个，需要指定 align=right。另外，rollmean 将“欺骗”并且使用将来那个时间实际上不可用的数据：

>ma <- rollmean(*ts*, *k*, align="right")

@通过日历周期应用函数

给定一个时间序列，你想要通过一个日历周期分组这些内容（如，周、月或年），然后对每

一个组应用一个函数。Xts 包包含用于按天、周、月、季、年处理时间序列的功能：

>apply.daily(*ts*, *f*)

>apply.weekly(*ts*, *f*)

>apply.monthly(*ts*, *f*)

>apply.quarterly(*ts*, *f*)

>apply.yearly(*ts*, *f*)

这里 ts 是一个 xts 时间序列，并且 f 是应用到每天、周、月、季或年的函数。如果你的时间序列是 zoo 对象，将它转换成一个 xts 对象，你就能访问这些函数；比如：

>apply.monthly(as.xts(*ts*), *f*)

@应用滚动功能

你想以滚动的形式向一个时间序列应用函数：在一个数据点计算该函数，然后移向下一个数据点，在那个点计算该函数，再移向下个点，如此等等。

在 zoo 包中使用 rollapply 函数。width 参数定义了来自时间序列 ts 在每个数据点有多少个数据点应该由函数 f 处理：

>library(zoo)

>rollapply(*ts*, *width*, *f*)

对许多应用，你可能设置 align="right"来避免使用在那时间不可用的历史数据来计算 f：

>rollapply(*ts*, *width*, *f*, align="right")

@画出自相关的函数

你想要画出你的时间序列的自相关函数（ACF）。使用 acf 函数：

>acf(ts)

@时间序列的自相关检验

你想要对你时间序列自相关的存在性进行检验。使用 Box.test 函数，它实现了用于自相关的 Box-Piece 检验：

>Box.test(ts)

@画出偏自相关函数

使用 pacf 函数：

>pacf(ts)

@发现两个时间序列的滞后相关性

你有两个时间序列，你想要知道是否它们之间有一个滞后的相关性。使用 ccf 函数来画出交叉相关函数，它将提示滞后的相关性：

>ccf(ts1, ts2)

@消除时间序列的趋势

你的时间序列数据包含趋势，你想要消除这种趋势。使用线性回归来识别趋势部分；然后从原始时间序列中减掉趋势部分。这两行说明了如何消除 zoo 对象 ts ,并将结果在于 detr 中：

>m <- lm(coredata(ts) ~ index(ts))

>detr <- zoo(resid(m), index(ts))

@拟合一个 ARIMA 模型

你想要对你的时间序列数据拟合一个 ARIMA 模型。在 forecast 包中的 auto.arima 函数能够选择正确的模型 order，并且对你的数据拟合好模型：

```
>library(forecast)
>auto.arima(x)
```

如果你已经知道模型的 order(p,d,q)，那么 arima 函数能够直接拟合模型：

```
>arima(x, order=c(p,d,q))
```

创建一个 ARIMA 模型涉及到三步：

识别模型的 order

对数据拟合模型，并给出系数

应用诊断的措施来验证模型

模型的 order 通常由三个整数表示(p,d,q)，这里的 p 是自回归的系数个数；d 是差分的阶数；q 是移动平均系数的个数。当我建立一个 ARIMA 模型的时候，我对于合适的 order 通常无线索可寻。使用 auto.arima，它会帮我找到最佳的 p,d,q 的组合：

```
>auto.arima(x)
Series: x
ARIMA(2,1,2)
Call: auto.arima(x = x)
Coefficients:
```

ar1 ar2 ma1 ma2

0.0451 -0.9183 -0.0066 0.6178

s.e. 0.0249 0.0252 0.0496 0.0517

sigma^2 estimated as 0.9877: log likelihood = -708.45

AIC = 1426.9 AICc = 1427.02 BIC = 1447.98

在该情况下，auto.arima 确定的最佳 order 是（2，1，2）。默认情况下，auto.arima 限制 p 和 q 在【0，5】范围内。如果你确信你的模型需要小于 5 个系数，使用 max.p 和 max.q 参数来限制查找范围；这会使得它更快。同样地，如果你相信你的模型需要更多的系数，使用 max.p 和 max.q 来扩展查找范围。如果你已经知道了你的 ARIMA 模型的 order，arima 函数能够快迅地根据数据拟合模型：

>arima(x, order=c(2,1,2))

Series: x

ARIMA(2,1,2)

Call: arima(x = x, order = c(2, 1, 2))

Coefficients:

ar1 ar2 ma1 ma2

0.0451 -0.9183 -0.0066 0.6178

s.e. 0.0249 0.0252 0.0496 0.0517

sigma^2 estimated as 0.9877: log likelihood = -708.45

AIC = 1426.9 AICc = 1427.02 BIC = 1447.98

该输出看起来和 auto.arima 相同。这里你不能看到的 arima 执行得更快。Auto.arima 和 arima 的输出都包含了对每个系数拟合的系数值与标准误差（s.e.）：

Coefficients:

ar1 ar2 ma1 ma2

0.0451 -0.9183 -0.0066 0.6178

s.e. 0.0249 0.0252 0.0496 0.0517

通过在一个对象中得到一个 ARIMA 模型，你能够通过使用 confint 函数找到系数的置信区间：

>m <- arima(x, order=c(2,1,2))

>confint(m)

2.5 % 97.5 %

ar1 -0.003811699 0.09396993

ar2 -0.967741073 -0.86891013

ma1 -0.103861019 0.09059546

ma2 0.516499037 0.71903424

该输出阐述了 ARIMA 建模的主要问题：并不是所有系数都必定是显著的。ar1 系数的置信区间为(−0.0038,+0.0940)。由于该区间包含 0 ,所以真实的系数也可能是 0 本身 ,这样 ar1 项是多余的。

对于 ma1 系数也是一样，它的置信区间为（−0.0066, +0.0496)。ar2 和 ma2 具有相似的显著性，因为它们并不包含 0。Auto.arima 和 arima 函数包含了用于拟合最佳模型的有用特征。比如，你可以迫使它们包含或排除一个趋势部分。查看帮助页获得详细信息。

最终警告：auto.arima 的危险之处在于它使用 ARIMA 建模看起来简单。ARIMA 建模并不简单。它比科学更艺术化，自动产生的模型只是一个起点。我强烈要求读者，在选定最终模型之前，去复习一本关于 ARIMA 建模的好书。

@移除非显著的 ARIMA 系数

你的 ARIMA 模型有一个以上的系数在统计学上不显著，你想要移除它们。

Arima 函数包含参数 fixed，它是一个向量。每个元素是 NA 或 0。对保留的系数使用 NA，对移除的系数使用 0.本例展示了一个 ARIMA(2, 1, 2)模型，将它的第一个 AR 系数和第一个 MA 系数强制赋为 0：

>arima(x, order=c(2,1,2), fixed=c(0,NA,0,NA))

通过查看置信区间，我们决定 ar1 和 ma1 是不显著的：

>m <- arima(x, order=c(2,1,2))

>confint(m)

2.5 % 97.5 %

ar1 -0.003811699 0.09396993

ar2 -0.967741073 -0.86891013

ma1 -0.103861019 0.09059546

ma2 0.516499037 0.71903424

既然用于 ar1 和 ma1 的置信区间包含 0，所以真实的系数就是 0。它们是第一和第三个系数，所以我们可以通过设定 fixed 的第一和第三个元素为 0 来移除它们：

>m <- arima(x, order=c(2,1,2), fixed=c(0,NA,0,NA))

>m

Series: x

ARIMA(2,1,2)

Call: arima(x = x, order = c(2, 1, 2), fixed = c(0, NA, 0, NA))

Coefficients:

ar1 ar2 ma1 ma2

0 -0.9082 0 0.5931

s.e. 0 0.0268 0 0.0522

sigma^2 estimated as 0.999: log likelihood = -711.27

AIC = 1428.54 AICc = 1428.66 BIC = 1449.62

Warning message:

In arima(x, order = c(2, 1, 2), fixed = c(0, NA, 0, NA)) :

some AR parameters were fixed: setting transform.pars = FALSE

由 arima 产生的警告信息是无害的。参数 transform.pars 的默认值是 TRUE，但是 arima 强制赋值为 FALSE，如果有必需，就是警告你。

可以看到 ar1 和 ma1 两个系数者为 0。留下的系数（ar2 和 ma2）仍然是显著的，通过它们的置信区间显示如下，这样我们有了一个合理的模型：

>confint(m)

2.5 % 97.5 %

ar1 NA NA

ar2 -0.9606839 -0.8557283

ma1 NA NA

ma2 0.4907792 0.6954606

@在 ARIMA 模型上执行诊断

你已经建立了一个 ARIMA 模型，你想要运行诊断检验来验证该模型。使用 tsdiag 函数。

本例将模型保存为 m，然后在模型上运行诊断：

```
>m <- arima(x, order=c(2,1,2), fixed=c(0,NA,0,NA), transform.pars=FALSE)
>tsdiag(m)
```

这些是基本的诊断，但它们是一个好的开始。在决定你的模型是可靠的之前，找一本关于 ARIMA 建模的好书，运行推荐的诊断检验。对于残差的额外的检验可以包括：

•正态性检验

•QQ 图

•直方图

•对拟合值的散点图

•时间序列图

@均值回归检验

你想要知道是否你的时间序列是均值回复的。对于均值回归的普通检验是 ADF 检验（Augmented Dickey−Fuller ），它是由 tseries 包中的 adf.test 函数实现的：

```
>library(tseries)
>adf.test(coredata(ts))
```

Adf.test 的输出包含一个 p 值。依惯例，如果 p<0.05 那么时间序列可能均值回复，反之，p>0.05 提供不了这样的证据。如果你的应用不希望去趋势或重新定位中心，使用

fUnitRoots 包中的 adfTest 函数替换：

```
>library(fUnitRoots)
>adfTest(coredata(ts1), type="nc")
```

设定 type="nc"，该函数既不会去趋势，又不会重新定位中心。设定 type="c"，该函数重新定位中心，但不会去趋势。adf.test 和 adfTest 这两个函数都会让你指定一个 lag 值，使得能够控制它们计算的精确统计量。这些函数提供了合理的默认值，但是任何严肃的用记应该学习下 ADF 检验的手册说明来决定用于她应用的合理的 lag 值。

@平滑一个时间序列

你有一个噪声时间序列。你想要平滑数据来消除噪声。KernSmooth 包有用于平滑的函数。使用 dpill 函数来选择一个初始的 bandwidth 参数；然后使用 locploy 函数来平滑数据：

```
library(KernSmooth)
gridsize <- length(y)
bw <- dpill(t, y, gridsize=gridsize)
lp <- locpoly(x=t, y=y, bandwidth=bw, gridsize=gridsize)
smooth <- lp$y
```

这里，t 是时间变量，y 是时间序列