R 语言学习笔记11

*Ning Liu, 04/06/2015*

官网：<http://www.r-project.org/>

下载地址：<http://ftp.ctex.org/mirrors/CRAN/>

# 安装package

        >packages->install packages->slect scan->slect package

        >library(meta) #加载package

       > help(package="meta") #访问package的帮助信息

# Help

其中有四种类型的帮助:

* help（functionname） #函数帮助文档，用?号也是一样的。
* help.search('keyword') #对已经安装的包搜索关键词，用??号功能一样。
* help(package='packagename') #显示已经安装的包的描述和函数说明
* RSiteSearch('keyword') #在官方网站上联网搜索

# **一些注意事项：**

* **对象（object）**是指可以赋值给变量的任何事物，包括常量、数据结构、函数，甚至图形。对象都拥有某种模式，描述了此对象是如何存储的，以及某个类。
* 对象名称中的句点（.）没有特殊意义。美元符号**“$”**指定一个对象中的某些部分。例如， A$x是指数据框A中的变量x。
* R不提供多行注释或块注释功能。你必须以#作为多行注释每行的开始。出于调试目的，你也可以把想让解释器忽略的代码放到语句if(FALSE){. .. }中。将FALSE改为TRUE即允许这块代码执行。
* 变量无法被声明。它们在首次被赋值时生成。
* 将一个值赋给某个向量、矩阵、数组或列表中一个不存在的元素时， R将自动扩展这个数据结构以容纳新值。或者减少以适应新的结构。
* R中没有标量。标量以单元素向量的形式出现。
* \n表示新行， \t为制表符， \'为单引号， \b为退格
* R中的下标不从0开始，而从1开始。
* 清屏和删除变量:    清屏： Ctrl + L;  删除变量: single varible - rm(object); all variables - rm(list =ls())

# 数据类型

数值型(numeric)、字符型(character)、逻辑型（TRUE/FALSE）、复数型（虚数）和原生型（字节）

# 5. 数据结构

str() 用于显示对象的结构，attributes()

标量, 向量，矩阵，数组，列表

## 5.1 向量 (Vetor)

用于存储数值型、字符型或逻辑型数据的一维数组

A<- c( , , , , )

其中, c()是向量赋值函数

## 5.2 矩阵(Matrix)

A<- matrix(vetor, nrow=“”, ncol= “”, byrow= “TURE or FALSE”, dimnames = list(“rownames”,“colnames”)

其中vector包含了矩阵的元素的一个**向量**， nrow和ncol用以指定行和列的维数(**标量**)， dimnames包含了可选的、以字符型**向量**表示的行名和列名。选项byrow则表明矩阵应当按行填充（byrow=**TRUE**）还是按列填充（byrow=**FALSE**） ，**默认情况下按列填充**。

## 5.3 数组（Array）

A <- array(vetor, dimensions, dimnames)

*vector*包含了数组中的数据的**向量**， *dimensions*是一个**数值型向量**，给出了各个维度下标的**最大值**，而*dimnames*是可选的、各维度名称标签的列表（**字符型向量**）

dim()函数可将向量转化成数组或矩阵。

                    例1：xx <- 1:24; dim(xx) <- c(3, 4, 2);

                    例2：zz <- 1:10; dim(zz) <- c(2, 5);

## 5.4 数据框 (Dataframe)

A<- data.frame(col1,col2,col3,…, row.names=””)

列向量col1, col2, col3,… 可为任何类型（如字符型、数值型或逻辑型）。其中row.names=””指定作为实例标识符 - case identifier的列向量（行数据的唯一性变量）。每一列的名称可由函数names指定。

**选取数据框中元素**的方式有若干种。

* 下标记号，亦可直接指定列名. Eg: *A*[1, 2]; *A*[c(“”,””)]
* **$** 它被用来选取一个给定数据框中的某个特定变量. Eg: *A*$col

可以联合使用函数attach()和detach()或单独使用函数with()来简化代码. 先用attach(*A*), 指定数据框，后面就可以直接写数据框中的变量名而不用加”$”, 结束的时候用detach(*A*)结束指定（注意：原始变量中存在和数据框中重名的变量，那么原始变量的优先级高于数据框中的变量，而出错），防止名字的冲突可以用函数with(*A*,{……..}),”{}”中语句提到的变量为数据框*A*中的变量,如提到外部变量则用”<<-”代替”<-”

## 5.5因子Factor

变量可归结为名义型、有序型或连续型变量。名义型变量是没有顺序之分的类别变量。有序型变量表示一种顺序关系，而非数量关系.连续型变量可以呈现为某个范围内的任意值，并同时表示了顺序和数量.

类别（名义型）变量和有序类别（有序型）变量在R中称为因子

factor(A, order=TURE, levels=c(A1,A2,… ))

*假设向量A*，factor(*A*) 则是一个存储原始向量*A*每个元素序号的整数向量，其中每个元素的序号默认将向量*A*中的元素按照首字母的顺序。从而转换为一个名义向量，若为factor(*A*,ordered=TURE), 怎转换为一个有序向量, 并且用 “levels=c(, , , …)”自定义排列顺序。

## 5.6列表List

列表就是一些对象（或成分，component）的有序集合。可能是若干向量、矩阵、数据框，甚至其他列表的组合

Alist <- list(object1,object2,…..)

给列表的对象命名（name为名字）

Alist <- list(name1=object1, name2=object2,…..)

你也可以通过在**双重方括号**中指明代表某个成分的数字或名称来访问列表中的元素。此例中， mylist[[2]]和mylist[["name2"]]均指第二个对象

# 6. 输入与输出

## 6.1读取键盘输入

1. 创建一个空数据框（或矩阵），其中变量名和变量的模式需与理想中的最终数据集一致
2. 针对这个数据对象调用文本编辑器，输入你的数据，并将结果保存回此数据对象中.

mydata <- data.frame(v1=numeric(0),v2=character(),…)

mydata <- edit(mydata)

## 6.2 读取表格文件

读取本地表格文件的主要函数是**read.table()**

newdata<- read.table(file, header=TURE or FALSE, seq=””, row.names=”name”)

其中， file是一个带分隔符的ASCII文本文件， header是一个表明首行是否包含了变量名的逻辑值（TRUE或FALSE）， sep用来指定分隔数据的分隔符(“,”,” ”,”\t”， row.names是一个可选参数，用以指定**一个或多个**表示行标识符的变量。

该函数读取数据后将存为 data.frame格式，而且所有的字符将被转为因子格式，如果你不想这么做需要记得将参数stringsAsFactors设为FALSE, 或者用colClasses为每一列指定一个类，例如logical（逻辑型）、 numeric（数值型）、 character（字符型）、 factor（因子）。与之类似 的函数是**read.csv()**专门用来读取csv格式。

在Excel中打开，框选所有的样本然后“复制”。在R语言中输入如下命令：

data=read.table('clipboard',T)

这里read.table是R读取外部数据的常用命令，T表示第一行是表头信息，整个数据存在名为data的变量中。另一种更方便的导入方法是利用Rstudio的功能，在workspace菜单选择“import dataset”也是一样的。

## 6.3 Web

如果是想抓去网页上的某个表格，那么可以使用XML包中的**readHTMLTable()**函数。例如我们想获得google统计的访问最多的1000名网站数据，则可以象下面这样做。关于这个函数可以参考这篇[博文](http://xccds1977.blogspot.com/2011/09/xml.html)。

[url](http://inside-r.org/r-doc/base/url) <- 'http://www.google.com/adplanner/static/top1000/'

[data](http://inside-r.org/r-doc/utils/data) <- readHTMLTable([url](http://inside-r.org/r-doc/base/url))

[names](http://inside-r.org/r-doc/base/names)([data](http://inside-r.org/r-doc/utils/data))

[head](http://inside-r.org/r-doc/utils/head)([data](http://inside-r.org/r-doc/utils/data)[[2]])

## **6.4 读取文本文件**

有时候需要读取的数据存放在非结构化的文本文件中，例如电子邮件数据或微博数据。这种情况下只能依靠**readLines()**函数，将文档转为以行为单位存放的list格式。例如我们希望读取wikipedia的主页html文件的前十行。

[data](http://inside-r.org/r-doc/utils/data) <- [readLines](http://inside-r.org/r-doc/base/readLines)('http://en.wikipedia.org/wiki/Main\_Page',n=10)

另外，**scan()**也有丰富的参数用来读取非结构化文档。

**批量读取本地文件**

在批量读取文档时一般先将其存放在某一个目录下。先用**dir()**函数获取目录中的文件名，然后用**paste()**将路径合成，最后用循环或向量化方法处理文档。例如：

doc.names <- [dir](http://inside-r.org/r-doc/base/dir)("path")

doc.path <- [sapply](http://inside-r.org/r-doc/base/sapply)(doc.names,[function](http://inside-r.org/r-doc/base/function)([names](http://inside-r.org/r-doc/base/names)) [paste](http://inside-r.org/r-doc/base/paste)(path,[names](http://inside-r.org/r-doc/base/names),sep='/'))

doc <- [sapply](http://inside-r.org/r-doc/base/sapply)(doc.path, [function](http://inside-r.org/r-doc/base/function)(doc) [readLines](http://inside-r.org/r-doc/base/readLines)(doc))

## 6.5导入netCDF数据

library(ncdf)

nc<-nc\_open(“file”)

mync<- get.var.ncdf(nc, myvar)

对于包含在netCDF文件mynetCDFfile中的变量myvar，其所有数据都被读取并保存到了一个名为myarray的R数组中。

**写入文件**  
**write.table()**与**write.csv()**函数可以很方便的写入表格型数据文档，而**cat()**函数除了可以在屏幕上输出之外，也能够输出成文件。  
  
另外若要与MySQL数据库交换数据，则可以使用**RMySLQ**包。

# 7. 处理数据对象的实用函数

**length**(*object*) 显示对象中元素/成分的数量  
**dim**(*object*) 显示某个对象的维度  
**str**(*object*) 显示某个对象的结构  
**class**(*object*) 显示某个对象的类或类型  
**mode**(*object*) 显示某个对象的模式  
**names**(*object*) 显示某对象中各成分的名称  
**c**(*object*, *object*,…) 将对象合并入一个向量

**cbind**(*object*, *object*, …) 按列合并对象  
**rbind**(*object*, *object*, …) 按行合并对象  
**Object** 输出某个对象  
**head**(*object*) 列出某个对象的开始部分  
**tail**(*object*) 列出某个对象的最后部分  
**ls**() 显示当前的对象列表  
**rm**(*object*, *object*, …) 删除一个或更多个对象。语句rm(list = ls())将删除当前工作环境中的几乎所有对象\*  
**newobject** <- edit(*object*) 编辑对象并另存为newobject  
**fix**(*object*) 直接编辑对象

# 8. R中的运算符

## 8.1 算数运算符

**+** #加; − #减; \* #乘; / #除; ^或\*\* #求幂; x%%y #求余(x mod y); x%/%y #整数除法

## 8.2 逻辑运算符

< 小于; <= 小于或等于; > 大于; >= 大于或等于; == 严格等于; != 不等于; !x 非x; x | y x或y; x & y X和y; isTRUE(x) 测试x是否为TRUE

# 9. 数据管理

## 9.1 创建变量

变量名←表达式, or

mydata <- transform(mydata,

sumx=x1+x2,

meanx=(x1+x2)/2)

transform()函数的一个示例。这种方式简化了按需创建新变量并将其保存到数据框mydata中的过程。

## 9.2变量的重编码

variable[condition] <- expression

将仅在condition的值为TRUE时执行赋值

## 9.3变量的重命名

Library(reshape)

fix() , or

dataframe <- rename(dataframe, c(oldname=”newname”, oldname=”newname”), or

names(data)[] <- c( , , …)

## 9.4缺失值

缺失值符号NA

函数is.na()允许你检测缺失值是否存在

返回一个相同大小的对象，如果某个元素是缺失值，相应的位置将被改写为TRUE，不是缺失值的位置则为FALSE。

## 9.5重编码某些值为缺失值

variable[condition] <- NA

## 9.6 在分析中排除缺失值

含有缺失值的算术表达式和函数的计算结果也是缺失值

数值函数都拥有一个na.rm=TRUE选项

函数na.omit()移除所有含有缺失值的观测

## 9.7 日期值

as.Date(x, "input\_format")

**input\_format:**

%d 数字表示的日期（0~31） 01~31

%a 缩写的星期名 Mon

%A 非缩写星期名 Monday

%m 月份（00~12） 00~12

%b 缩写的月份 Jan

%B 非缩写月份 January

%y 两位数的年份 07

%Y 四位数的年份 2007

## 9.8 数据类型转换

判断 转换

is.numeric() as.numeric()  
is.character() as.character()  
is.vector() as.vector()  
is.matrix() as.matrix()  
is.data.frame() as.data.frame()  
is.factor() as.factor()  
is.logical() as.logical()

## 9.9数据排序

newdata <- data[order(data$v1, - data$v2), ]

减号即可得到降序的排序结果

## 9.10数据集的合并

### 9.10.1 添加列

newdata <- merge(data1, data2, by=c(“ID1” ,”ID2”)

将两个数据框按照ID1和ID2进行了合并

### 9.10.2添加行

newdata <- rbind(data1, data2)

两个数据框必须拥有相同的变量，不过它们的顺序不必一定相同

## 9.11数据集取子集

### 9.11.1选入（保留）变量

dataframe[row indices, column indices]

### 9.11.2剔除（丢弃）变量

myvars <- names(data) %in% c(“var1”,”var2”)

newdata<- data[!myvars], or

newdata <- data[c(-1,-2)]

### 9.11.3选入观测

newdata <- data[which(expressions), , ], or

newdata <- subset(data, expressions, select=c( , , , )]

冒号运算符from:to。在这里，它表示了数据框中变量from到变量to包含的所有变量。

## 9.12数值和字符处理函数

### 9.12.1数学函数

Like: abs(),exp(),…..

当这些函数被应用于数值向量、矩阵或数据框时，它们会作用于每一个独立的值。

### 9.12.2统计函数

Like: mean(),sd(),sum(),….

### 9.12.3概率函数

[dpqr] distribution\_abbreviation()

其中第一个字母表示其所指分布的某一方面：  
d = 密度函数（density）  
p = 分布函数（distribution function）  
q = 分位数函数（quantile function）  
r = 生成随机数（随机偏差）

### 9.12.4其他实用函数

**length**(*x)* 对象x的长度

**seq**(*from*, *to*, *by*) 生成一个序列

**rep**(*x*, *n*) 将*x*重复*n*次

**cut**(*x*, *n*) 将连续型变量*x*分割为有着*n*个水平的因子

**pretty**(*x*, *n*) 创建美观的分割点。通过选取*n*+1个等间距的取整值，将一个连续型变量*x*分割为*n*个区间。

**cat**(. .., file ="myfile",append =FALSE) 连接...中的对象，并将其输出到屏幕上或文件中（如果声明了一个的话）:firstname <- c("Jane"), cat("Hello" ,firstname, "\n")

## 9.13将函数应用于矩阵和数据框

和matlab一样，R语言以向量为基本运算对象。也就是说，当输入的对象为向量时，对其中的每个元素分别进行处理，然后以向量的形式输出。R语言中基本上所有的数据运算均能允许向量操作。不仅如此，R还包含了许多高效的向量运算函数，这也是它不同于其它软件的一个显著特征。向量化运算的好处在于避免使用 循环，使代码更为简洁、高效和易于理解。本文来对apply族函数作一个简单的归纳，以便于大家理解其中的区别所在。  
  
所谓apply族函数包括了apply, sapply, lappy, tapply等函数，这些函数在不同的情况下能高效的完成复杂的数据处理任务，但角色定位又有所不同。  
  
**apply()**

处理对象是**矩阵或数组**，它逐行或逐列的处理数据，其输出的结果将是一个向量或是矩阵。下面的例子即对一个随机矩阵求每一行的均值。要注意的是apply与其它函数不同，它并不能明显改善计算效率，因为它本身内置为循环运算。

Apply(x,margin,fun,…..)

其中， x为数据对象， MARGIN是维度的下标， FUN是由你指定的函数，而...则包括了任何想传递给FUN的参数。在矩阵或数据框中， MARGIN=1表示行， MARGIN=2表示列。

**lappy()**

处理对象是向量、列表或其它对象，它将向量中的每个元素作为参数，输入到处理函数中，最后生成结果的格式为列表。在R中数据框是一种特殊的列表，所以数据框的列也将作为函数的处理对象。下面的例子即对一个数据框按列来计算中位数与标准差。

f.data <- [data.frame](http://inside-r.org/r-doc/base/data.frame)(x=[rnorm](http://inside-r.org/r-doc/stats/rnorm)(10),y=[runif](http://inside-r.org/r-doc/stats/runif)(10))

[lapply](http://inside-r.org/r-doc/base/lapply)(f.data,FUN=[function](http://inside-r.org/r-doc/base/function)(x) [list](http://inside-r.org/r-doc/base/list)([median](http://inside-r.org/r-doc/stats/median)=[median](http://inside-r.org/r-doc/stats/median)(x),[sd](http://inside-r.org/r-doc/stats/sd)=[sd](http://inside-r.org/r-doc/stats/sd)(x))）

**sapply()**

它和lappy()是非常相似的，但其输出格式则是较为友好的矩阵格式。

[sapply](http://inside-r.org/r-doc/base/sapply)(f.data,FUN=[function](http://inside-r.org/r-doc/base/function)(x) [list](http://inside-r.org/r-doc/base/list)([median](http://inside-r.org/r-doc/stats/median)=[median](http://inside-r.org/r-doc/stats/median)(x),[sd](http://inside-r.org/r-doc/stats/sd)=[sd](http://inside-r.org/r-doc/stats/sd)(x)))

[class](http://inside-r.org/r-doc/base/class)(test)

**tapply()**

专门用来处理分组数据的，其参数要比sapply多一个。我们以iris数据集为例，可观察到Species列中存放了三种花的名称，我们的目的是要计算三种花瓣萼片宽度的均值。其输出结果是数组格式。

[head](http://inside-r.org/r-doc/utils/head)([iris](http://inside-r.org/r-doc/datasets/iris))

[attach](http://inside-r.org/r-doc/base/attach)([iris](http://inside-r.org/r-doc/datasets/iris))

[tapply](http://inside-r.org/r-doc/base/tapply)(Sepal.Width,INDEX=Species,FUN=[mean](http://inside-r.org/r-doc/base/mean))

与tapply功能非常相似的还有**aggregate()**，其输出是更为友好的数据框格式。而**by()**和上面两个函数是同门师兄弟。  
  
另外还有一个非常有用的函数**replicate()**，它可以将某个函数重复运行N次，常常用来生成较复杂的随机数。下面的例子即先建立一个函数，模拟扔两个骰子的点数之和，然后重复运行10000次。

game <- [function](http://inside-r.org/r-doc/base/function)() {

n <- [sample](http://inside-r.org/r-doc/base/sample)(1:6,2,[replace](http://inside-r.org/r-doc/base/replace)=T)

[return](http://inside-r.org/r-doc/base/return)([sum](http://inside-r.org/r-doc/base/sum)(n))

}

[replicate](http://inside-r.org/r-doc/base/replicate)(n=10000,game())

最后一个有趣的函数**Vectorize()**，它能将一个不能进行向量化运算的函数进行转化，使之具备向量化运算功能。

## 9.14整合与重构

### 9.14.1 转置

t( )

### 9.14.2整合数据（相当于数据的分类汇总）

aggrerate(x, by, FUN)

*x*是待折叠的数据对象， *by*是一个变量名组成的列表，这些变量将被去掉以形成新的观测，而*FUN*则是用来计算描述性统计量的标量函数，

## 9.15控制流

为了理解贯穿本节的语法示例，请牢记以下概念：

 语句（statement）是一条单独的R语句或一组复合语句（包含在花括号{ } 中的一组R语句，使用分号分隔）；

 条件（cond）是一条最终被解析为真（TRUE）或假（FALSE）的表达式；

 表达式（expr）是一条数值或字符串的求值语句；

 序列（seq）是一个数值或字符串序列。

### 9.15.1重复和循环

**for循环**重复地执行一个语句，直到某个变量的值不再包含在序列seq中为止。

**for (n in seq) ｛expr}**

R中最基本的是for循环，其中n为循环变量，seq通常是一个序列。n在每次循环时从seq中顺序取值，代入到后面的expr语句中进行运算。下面的例子即是以for循环计算30个[Fibonacci](http://en.wikipedia.org/wiki/Fibonacci_number)数。

x <- c(1,1)  
for (i in 3:30) {  
    x[i] <- x[i-1]+x[i-2]  
}

**while循环**当不能确定循环次数时，我们需要用while循环语句。

while (condition) {expr}

在condition条件为真时，执行大括号内的expr语句。下面即是以while循环来计算30个Fibonacci数。

x <- c(1,1)  
i <- 3  
while (i <= 30) {  
    x[i] <- x[i-1]+x[i-2]  
    i <- i +1  
}

### 9.15.2条件

**if语句**用来进行条件控制，以执行不同的语句。

if (conditon) {expr1} else {expr2}

若condition条件为真，则执行expr1，否则执行expr2。**ifesle()**函数也能以简洁的方式构成条件语句。下面的一个简单的例子是要找出100以内的质数。

x <- 1:100  
y <- rep(T,100)  
for (i in 3:100) {  
    if (all(i%%(2:(i-1))!=0)){  
        y[i] <- TRUE  
        } else {y[i] <- FALSE  
                }  
}  
print(x[y])  
  
在上面例子里，**all()**函数的作用是判断一个逻辑序列是否全为真，%%的作用是返回余数。在if/else语句中一个容易出现的错误就是else没有放在｝的后面，若你执行下面的示例就会出现错误。  
logic = 3  
x<- c(2,3)  
if (logic == 2){  
    y <- x^2  
}  
else {  
  y<-x^3  
}  
show(y)

**ifelse**结构

ifelse(cond, statement1, statement2)

**switch**结构(类似其他语言的case)

switch(expr, statements)

其中的...表示与*expr*的各种可能输出值绑定的语句。

switch(i,

i1 = “”

i2 = “”

….

)

# 10. 对象和类

R是一种基于**对象(Object)**的语言，所以你在R语言中接触到的每样东西都是一个对象，一串数值向量是一个对象，一个函数是一个对象，一个图形也是一个对象。基于对象的编程(OOP)就是在定义类的基础上，创建与操作对象。  
  
对象中包含了我们需要的数据，同时对象也具有很多**属性(Attribute)**。其中一种重要的属性就是它的**类(Class)**，R 语言中最为基本的类包括了数值(numeric)、逻辑(logical)、字符(character)、列表(list)，在此基础上构成了一些复合型 的类，包括矩阵(matrix)、数组(array)、因子(factor)、数据框(dataframe)。

查看class的函数：

**attributes(model)；str(); mode(x)； class()； unclass()（消除类）**

我们下面来用一个简单线性回归的例子来了解一下对象和类的处理。

*# 创建两个数值向量*

x <- [runif](http://inside-r.org/r-doc/stats/runif)(100)

y <- [rnorm](http://inside-r.org/r-doc/stats/rnorm)(100)+5\*x

*# 用线性回归创建模型，存入对象model*

model <- [lm](http://inside-r.org/r-doc/stats/lm)(y~x)

好了，现在我们手头上有一个不熟悉的对象model，那么首先来看看它里面藏着什么好东西。最有用的函数命令就是**attributes(model)**，用来提取对象的各种属性，结果如下：  
  
> attributes(model)  
$names  
 [1] "coefficients"  "residuals"     "effects"       
 [4] "rank"          "fitted.values" "assign"       
 [7] "qr"            "df.residual"   "xlevels"       
[10] "call"          "terms"         "model"         
  
$class  
[1] "lm"  
  
可以看到这个对象的类是“lm”，这意味着什么呢？我们知道对于不同的类有不同的处理方法，那么对于modle这个对象，就有专门用来处理lm类对象的函 数，例如plot.lm()。但如果你用普通的函数plot()也一样能显示其图形，Why？因为plot()这种函数会自动识别对象的类，从而选择合适 的函数来对付它，这种函数就称为**泛型函数（generic function）**。你可以用**methods(class=lm)**来了解有哪些函数可适用于lm对象。  
  
好了，我们已经知道了model的底细了，你还想知道x的信息吧。如果运行attributes(x)，会发现返回了空值。这是因为x是一个向量，对于向量这种内置的基本类，attributes是没有什么好显示的。此时你可以运行**mode(x)**，可观察到向量的类是数值型。如果运行mode(model)会有什么反应呢？它会显示lm类的基本构成是由list组成的。当然要了解对象的类，也可以直接用**class()**，如果要消除对象的类则可用**unclass()**。  
  
从上面的结果我们还看到names这个属性，这如同你到一家餐厅问服务生要一份菜单，输入**names(model)**就相当于问model这 个对象：Hi，你能提供什么好东西吗？如果你熟悉回归理论的话，就可以从names里头看到它提供了丰富的回归结果，包括回归系数 （coefficients）、残差（residuals）等等，调用这些信息可以就象处理普通的数据框一样使用$符号，例如输出残差可以用 model$residuals。当然用泛型函数可以达到同样的效果，如residuals(model)，但在个别情况下，这二者结果是有少许差别的。

# 11. 字符串处理

尽管R语言的主要处理对象是数字，而字符串有时候也会在数据分析中占到相当大的份量。特别是在文本数据挖掘日趋重要的背景下，在数据预处理阶段你需要熟练的操作字符串对象。当然如果你擅长其它的处理软件，比如Python，可以让它来负责前期的脏活。  
  
**获取字符串长度：nchar()**能够获取字符串的长度，它也支持字符串向量操作。注意它和length()的结果是有区别的。  
  
**字符串粘合：paste()**负责将若干个字符串相连结，返回成单独的字符串。其优点在于，就算有的处理对象不是字符型也能自动转为字符型。  
  
**字符串分割：strsplit()**负责将字符串按照某种分割形式将其进行划分，它正是paste()的逆操作。  
  
**字符串截取：substr()**能对给定的字符串对象取出子集，其参数是子集所处的起始和终止位置。  
  
**字符串替代：gsub()**负责搜索字符串的特定表达式，并用新的内容加以替代。**sub()**函数是类似的，但只替代第一个发现结果。  
  
**字符串匹配：grep()**负责搜索给定字符串对象中特定表达式 ，并返回其位置索引。**grepl()**函数与之类似，但其后面的"l"则意味着返回的将是逻辑值。  
  
**一个例子：**  
我们来看一个处理邮件的例子，目的是从该文本中抽取发件人的地址。该文本[在此](http://www.box.com/s/g8udktb6q59o88oagehi)可以下载到。邮件的全文如下所示：  
----------------------------  
Return-Path: skip@pobox.com  
Delivery-Date: Sat Sep  7 05:46:01 2002  
From: skip@pobox.com (Skip Montanaro)  
Date: Fri, 6 Sep 2002 23:46:01 -0500  
Subject: [Spambayes] speed  
Message-ID: <15737.33929.716821.779152@12-248-11-90.client.attbi.com>  
  
If the frequency of my laptop's disk chirps are any indication, I'd say  
hammie is about 3-5x faster than SpamAssassin.  
  
Skip  
----------------------------

*# 用readLines函数从本地文件中读取邮件全文。*

[data](http://inside-r.org/r-doc/utils/data) <- [readLines](http://inside-r.org/r-doc/base/readLines)('data')

*# 判断对象的类，确定是一个文本型向量，每行文本是向量的一个元素。*

[class](http://inside-r.org/r-doc/base/class)([data](http://inside-r.org/r-doc/utils/data))

*# 从这个文本向量中找到包括有"From:"字符串的那一行*

email <- [data](http://inside-r.org/r-doc/utils/data)[[grepl](http://inside-r.org/r-doc/base/grepl)('From:',[data](http://inside-r.org/r-doc/utils/data))]

*#将其按照空格进行分割，分成一个包括四个元素的字符串向量。*

from <- [strsplit](http://inside-r.org/r-doc/base/strsplit)(email,' ')

*# 上面的结果是一个list格式，转成向量格式。*

from <- [unlist](http://inside-r.org/r-doc/base/unlist)(from)

*# 最后搜索包含'@'的元素，即为发件人邮件地址。*

from <- from[[grepl](http://inside-r.org/r-doc/base/grepl)('@',from)]

在字符串的复杂操作中通常会包括**正则表达式（Regular Expressions）**，关于这方面内容可以参考?regex

# 12. 程序查错

基本的除错方法是跟踪重要变量的赋值情况。在循环或条件分支代码中加入显示函数能完成这个工作。例如**cat('var',var,'\n')**。

**cat**(. .., file ="myfile",append =FALSE) 连接...中的对象，并将其输出到屏幕上或文件中（如果声明了一个的话）:firstname <- c("Jane"), cat("Hello" ,firstname, "\n")

在确认程序运行正常后，可以将这行代码进行注释。好的编程风格也能有效的减少出错的机会。在编写代码时先写出一个功能最为简单的部分，然后在此基础上逐步添加其它复杂的功能。对输出结果进行绘图或统计汇总也能揭示一些潜在的问题。  
  
另一种避免出错的方法是尽量使用函数。使用函数能将一个大的程序分解成几个小型的模块。每一个函数模块只负责某一种功能的实现。这样使程序更容易理解和维 护，而且容易针对各函数的输入、计算、输出分别进行查错调试。R语言中函数的运行不会影响到全局变量，所以使用函数基本上不会有什么副作用。  
  
但是在使用函数时需要注意的问题是输入参数的不可预测性。未预料到的输入参数会产生奇怪的或是错误的输出，所以在函数起始部分就要用条件语句来检查参数的正确与否。如果输入参数不正确，可以用下面的语句来停止程序执行**stop('your message here.')**。  
  
对函数进行调试的重要工具是**browser()**，它可以使我们进入调试模式逐行运行代码。在函数中的某一行插入browser()后，在函 数执行时会在这一行暂停中断，并显示一个提示符。此时我们可以在提示符后输入任何R语言的交互式命令进行检查调试。输入n则会逐行运行程序，并提示下一行 将运行的语句。输入c会直接跳到下一个中断点。而输入Q则会直接跟出调试模式。  
  
**debug()**函数和browser()是相似的，如果你认为某个函数，例如fx(x)，有问题的话，使用debug(fx)即可进入调试模式。它本质上是在函数的第一行加入了browser，所以其它提示和命令都是相同的。其它与程序调试有关的函数还包括:

* **trace**(fx1,fx2),该函数可以让我们在调用fx1函数时自动调用fx2，通常我们将browser作为fx2；
* **setBreakpoint**(filename,linenumber),该函数可在程序的某一行设置中断点；
* **traceback**()，在程序出现报错时，可使用该函数回溯程序调用过的函数名

有兴趣的各位可参见相关R语言帮助文档，或是《The Art of R Programming》的第十三章。