

# 数据处理

# 本章内容

- 变量的创建、编码、命名
- 缺失值、日期值处理，数据类型转化，数据排序
- 数据集的合并，选取子集，使用**SQL**操作数据框，数据的整合与重构
- 控制流：条件与循环
- 用户自编函数

变量的创建、编码、命名

# 背景

- 将数据表示为矩阵或数据框仅是数据准备的第一步
- 多达**60%**的数据分析时间都花在了实际分析前数据的准备上

# 创建新变量

- 创建新变量或者对现有的变量进行变换  
变量名<-表达式
- “表达式”可以包含多种运算符和函数

# 算数运算符

运 算 符	描 述
+	加
-	减
*	乘
/	除
^或**	求幂
x%%y	求余 ( $x \bmod y$ )。5%%2的结果为1
x%/%y	整数除法。5%/2的结果为2

# 创建新变量例子

```
mydata<-data.frame(x1 = c(2, 2, 6, 4), x2 = c(3, 4, 2, 8))
```

```
mydata$sumx <- mydata$x1 + mydata$x2
```

```
mydata$meanx <- (mydata$x1 + mydata$x2)/2
```

- 上面例子可以使用attach()省去\$符:

```
attach(mydata)
```

```
mydata$sumx <- x1 + x2
```

```
mydata$meanx <- (x1 + x2)/2
```

# 变量重编码(Recoding Variables)

- **重编码**是指根据**已有变量**创建**新变量**的过程。比如：
  - 将一个连续型变量变为一组类别值
  - 将误编码的值替换为正确值
  - 基于一组分数创建一个表示及格/不及格的变量
- 要重编码数据，可以使用**R**中的逻辑运算符。逻辑运算符表达式返回**TRUE**或**FALSE**。



# 逻辑运算符

运 算 符	描 述
<	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

# 例子（1.准备数据：调查问卷）

```
manager <- c(1, 2, 3, 4, 5)
date <- c("10/24/08", "10/28/08", "10/1/08", "10/12/08", "5/1/09")
country <- c("US", "US", "UK", "UK", "UK")
gender <- c("M", "F", "F", "M", "F")
age <- c(32, 45, 25, 39, 99) #99代表年龄缺失
q1 <- c(5, 3, 3, 3, 2)
q2 <- c(4, 5, 5, 3, 2)
q3 <- c(5, 2, 5, 4, 1)
q4 <- c(5, 5, 5, NA, 2)
q5 <- c(5, 5, 2, NA, 1)
leadership <- data.frame(manager, date, country, gender, age, q1, q2,
q3, q4, q5, stringsAsFactors=FALSE)
```

# 例子（2.变量重编码）

新属性：年龄类别（age category）

↓  
leadership\$agecat[leadership\$age > 75] <- "Elder"  
leadership\$agecat[leadership\$age >= 55 &  
leadership\$age <= 75] <- "Middle Aged"  
leadership\$agecat[leadership\$age < 55] <-  
"Young"

## 例子（用within()改写）

```
leadership <- within(leadership,{  
  agecat <- NA  
  agecat[age > 75] <- "Elder"  
  agecat[age >= 55 & age <= 75] <-  
"Middle Aged"  
  agecat[age < 55] <- "Young" })
```

- **注意：** 如果不用<-赋值回给leadership，则在运行结果为临时值，原leadership不会被改变。
- The within() function is similar to the with() function, but allows you to modify the data frame.

# 变量重命名: `fix()`

- `fix()` 可以调用一个交互式的编辑器，单击变量名，然后在弹出的对话框中将其重命名。
  - 例如: `fix(leadership)`
- **`edit`和`fix`函数的区别**
  - 只用`edit(X)`在窗口中编辑修改X，关闭窗口后X没有改变，应使用 **`Y<- edit(X)`**；使用`fix(X)`在窗口中编辑修改X，关闭窗口后X就是修改后的值，不必使用`Y <- fix(X)`。

# 变量重命名，方法一：rename()

- **reshape**包中有一个**rename()**函数，可用于修改变量名

```
rename(dataframe, c(olddname="newname", oldname="newname", ...))
```

- 例

```
install.packages("reshape") #先安装包
```

```
library(reshape)
```

```
leadership <- rename(leadership,  
c(manager="managerID", date="testDate")  
)
```

# 变量重命名，方法二：names()

#查看所有变量名

```
names(leadership)
```

#将第二个变量重命名

```
names(leadership)[2] <- "testDate"
```

#将q1到q5重命名为item1到item5

```
names(leadership)[6:10] <- c("item1",  
"item2", "item3", "item4", "item5")
```

缺失值、日期值处理，数据类型转化，数据排序



# 缺失值（1）

- 在任何项目中，数据都可能由于未作答问题、设备故障或误编码数据而不完整。
- 在R中，**缺失值**以符号**NA**（Not Available，不可用）表示。不可能出现的值（例如，被0除的结果）通过符号NaN（Not a Number，非数值）来表示。

## 缺失值（2）

- 函数**is.na()**用以检测**缺失值是否存在**。

```
y <- c(1, 2, 3, NA)
```

```
#then the function
```

```
is.na(y)
```

```
#returns c(FALSE, FALSE, FALSE, TRUE)
```

# 缺失值（3）：重编码某些值为缺失值

- 在之前示例中，缺失的年龄值被编码为99。任何等于99的年龄值都应该被修改为NA。

```
leadership$age[leadership$age==99] <- NA
```

- 必须确保所有的缺失数据已在分析之前被妥善地编码为缺失值，否则分析结果将失去意义。

# 缺失值（4）：在分析中排除缺失值（1/2）

- 由于**含有缺失值**的算术表达式和函数的计算**结果也是缺失值**：需要**删除**。
- 例如：  
    `x <- c(1, 2, NA, 3)`  
    `y <- x[1] + x[2] + x[3] + x[4]`  
    `z <- sum(x)`
- 由于x中的第3个元素是缺失值，所以y和z也都是NA

# 缺失值（5）：在分析中排除缺失值（2/2）

- 多数的数值函数都有一个na.rm=TRUE选项

```
x <- c(1, 2, NA, 3)
```

```
y <- sum(x, na.rm=TRUE)
```

- 这里，y等于6。在使用函数处理不完整的数据时，请务必查阅它们的帮助文档（例，help(sum)）。

- **na.omit()**可以删除所有含缺失数据的行：

```
newdata <- na.omit(leadership)
```

# 日期值（1/6）

- 日期值通常以字符串的形式输入到R中，然后转化为以数值形式存储的日期变量。
- 函数`as.Date()`用于执行这种转化。其语法为`as.Date(x, "input_format")`，其中`x`是字符型数据，`input_format`则给出了用于读入日期的适当格式。

# 日期值（2/6）

表：日期格式

符 号	含 义	示 例
%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

## 日期值（3/6）

- 日期值的默认输入格式为yyyy-mm-dd  
`mydates <- as.Date(c("2007-06-22", "2004-02-13"))`
- 在**leadership**数据集中，日期是以mm/dd/yy的格式编码为字符型变量的。需要转化：  
`leadership$date <- as.Date(leadership$date, "%m/%d/%y")`



# 日期值（4/6）

- `Sys.Date()`可以返回当天的日期
- `date()`可以返回当前的日期和时间。
- 可以使用`format(x, format="output_format")`来输出指定格式的日期值

```
today <- Sys.Date()
```

```
format(today, format="%B %d %Y")
```

```
format(today, format="%A")
```

# 日期值（5/6）

- R的内部使用自1970年1月1日以来的天数表示日期，更早的日期表示为负数。可以在日期值上执行算术运算：

```
startdate <- as.Date("2004-02-13")
```

```
enddate <- as.Date("2011-01-22")
```

```
days <- enddate - startdate
```

计算周数： 

```
today <- Sys.Date()
```

```
dob <- as.Date("1956-10-12")
```

```
difftime(today, dob, units="weeks")
```

# 日期值（6/6）

- 将日期转化为字符型变量(不常用)  
`strDates <- as.character(dates)`

# 数据类型转换

- R提供了一系列用于判断某个对象的数据类型和将其转换为另一种数据类型的函数。

Test	Convert
<code>is.numeric()</code>	<code>as.numeric()</code>
<code>is.character()</code>	<code>as.character()</code>
<code>is.vector()</code>	<code>as.vector()</code>
<code>is.matrix()</code>	<code>as.matrix()</code>
<code>is.data.frame()</code>	<code>as.data.frame()</code>
<code>is.factor()</code>	<code>as.factor()</code>
<code>is.logical()</code>	<code>as.logical()</code>

- **is.datatype()**返回TRUE/FALSE,  
**as.datatype()**将参数**转换**为对应类型。

# 数据类型转换示例

```
a <- c(1,2,3)
```

```
a
```

```
is.numeric(a)
```

```
is.vector(a)
```

```
a <- as.character(a)
```

```
a
```

```
is.numeric(a)
```

```
is.vector(a)
```

```
is.character(a)
```

# 数据类型转换的用处

- 与控制流（如**if-then**）结合使用时，**is.datatype()**这样的函数将成为一类强大的工具，即允许根据数据的具体类型以不同的方式处理数据。
- 某些**R**函数需要接受某个特定类型（字符型或数值型，矩阵或数据框）的数据，**as.datatype()**这类函数可以在分析之前先行将数据转换为要求的格式。

# 数据排序

- 使用`order()`函数对一个数据框进行排序。

```
newdata <- leadership[order(leadership$age),]
```

↑  
数据框

↑  
要排序的变量

- 在排序变量的前边加一个**减号**即可得到降序的排序结果。

```
newdata <- leadership[order(-leadership$age),]
```

- 多变量排序

```
newdata <- leadership[order(leadership$gender,  
leadership$age),]
```

数据集的合并，选取子集，使用**SQL**操作数据框，数据的整合与重构



# 数据集的合并（Merging）

- 如果数据分散在多个地方，就需要在继续下一步之前将其合并。
  - 添加列（变量）
  - 添加行（观测）

# 添加列（添加变量）

- 要横向合并两个数据集，使用merge()函数。两个数据框是通过一个或多个共有变量进行联结的（inner join）。例如：
- total <- merge(dataframeA, dataframeB, by="ID")  
将dataframeA和dataframeB按照ID进行合并。
- total <- merge(dataframeA, dataframeB, by=c("ID","Country"))  
将两个数据框按照两个共有变量ID和Country进行了合并。

# 直接横向合并，即（AB）

- 直接**横向合并**两个矩阵或数据框，并且不需要指定一个公共索引，那么可以直接使用**cbind()**函数（column bind）：

`total <- cbind(A, B)`

- 这个函数将横向合并对象**A**和对象**B**。为了让它正常工作，每个对象必须拥有**相同的行数**，且要以**相同顺序**排序。

# 添加行（添加观测），即 $\begin{pmatrix} \mathbf{A} \\ \mathbf{B} \end{pmatrix}$

- 要**纵向合并**两个数据框（数据集），使用 `rbind()` 函数（row bind）。

```
total <- rbind(dataframeA, dataframeB)
```

- 两个数据框**必须拥有相同的变量**，不过顺序不必相同。如果 `dataframeA` 中拥有 `dataframeB` 中没有的变量，在合并之前做以下某种处理：
  - 删除 `dataframeA` 中的多余变量；
  - 在 `dataframeB` 中创建追加的变量并将其值设为 `NA`（缺失）。

# 数据集取子集

- 对变量和观测进行保留或删除的方法
  - 选入（保留）变量
  - 剔除（丢弃）变量
  - 选入观测
  - **subset()**函数——最简单的方法
  - 随机抽样

# 选入（保留）变量

- 从一个大数据集中选择有限数量的变量来创建一个新的数据集。
- 数据框中的元素是通过`dataframe[row indices, column indices]`来访问的。可以沿用这种方法来选择变量。例如：

```
newdata <- leadership[, c(6:10)]
```

- 等价于

```
myvars <- c("q1", "q2", "q3", "q4", "q5")  
newdata <- leadership[myvars]
```

# 剔除（丢弃）变量（1/2）

- 剔除（dropping）变量的原因有很多。例如，如果某个变量中有若干缺失值，则需要为进一步分析之前将其丢弃。
- 方法1：  

```
myvars <- names(leadership) %in% c("q3", "q4")  
newdata<-leadership[!myvars]
```

解释：(1) names(leadership)生成了一个包含所有变量名的字符型向量：  
c("managerID","testDate","country","gender","age","q1", "q2","q3","q4","q5")。

(2) names(leadership) %in% c("q3", "q4")返回了一个逻辑型向量，names(leadership)中每个匹配q3或q4的元素的值为TRUE，反之为FALSE：c(FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, FALSE)。

(3) 运算符非 (!) 将逻辑值反转：c(TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, FALSE, FALSE, TRUE)。

(4) leadership[c(TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, FALSE, FALSE,TRUE)]选择了逻辑值为TRUE的列，于是q3和q4被剔除了。

# 备注

- 选取变量时采用逻辑型向量注意：
- `leadership[c(TRUE, FALSE, FALSE),]`
  - 如果行数大于3：会得到1、4、7、10、13、16.....行
- `leadership[c(TRUE, FALSE),]`
  - 如果行数大于2：会得到1、3、5、7.....行



# 剔除（丢弃）变量（2/2）

- 方法2：因为知道q3和q4是第7个和第8个变量，也可以使用语句将其剔除：

```
newdata <- leadership[c(-7,-8)]
```

- 方法3：也可以使用以下语句剔除：

```
leadership$q3 <- leadership$q4 <- NULL
```

– **注意！**：此种方法会**直接作用**于leadership。

– 此语句实为两句：

- leadership\$q3 <- NULL
- leadership\$q4 <- NULL

# 选入观测

- 选择了第1行到第3行（前三个观测）  
`newdata <- leadership[1:3,]`
- 选择了所有30岁以上的男性  
`newdata <- leadership[leadership$gender=="M"  
&leadership$age > 30,]`

解释：(1) 逻辑比较`leadership$gender=="M"`生成了向量`c(TRUE, FALSE, FALSE, TRUE, FALSE)`。

(2) 逻辑比较`leadership$age > 30`生成了向量`c(TRUE, TRUE, FALSE, TRUE, TRUE)`。

(3) 逻辑比较`c(TRUE, FALSE, FALSE, TRUE, TRUE) & c(TRUE, TRUE, FALSE, TRUE, TRUE)`生成了向量`c(TRUE, FALSE, FALSE, TRUE, FALSE)`。

(4) 函数`which()`给出了向量中值为TRUE元素的下标。因此，`which(c(TRUE, FALSE, FALSE, TRUE, FALSE))`生成了向量`c(1, 4)`。

(5) `leadership[c(1,4),]`从数据框中选择了第一个和第四个观测。这就满足了我们的选取准则（30岁以上的男性）。

# subset()函数

- subset函数是选择变量和观测最简单的方法
- 例，选择所有age值大于等于35或age值小于24的行，**保留**了变量q1到q4：  
第2个参数：  
/ 逻辑表达式  

```
newdata <- subset(leadership, age >= 35 | age < 24, select=c(q1, q2, q3, q4))
```
- 例，选择所有25岁以上的男性，并保留了变量gender到q4和其间所有列：  

```
newdata <- subset(leadership, gender=="M" & age > 25, select=gender:q4)
```

# 随机抽样（1/2）

- 经常需要创建两份随机样本，使用其中一份样本构建预测模型，使用另一份样本验证模型的有效性。
- **sample()**函数能够让你从数据集中（**有放回或无放回地**）抽取大小为n的一个**随机**样本。

## 随机抽样（2/2）

- 例，从leadership数据集中随机抽取一个大小为3的样本：

```
mysample <-  
leadership[sample(1:nrow(leadership), 3,  
replace=FALSE),]
```

- sample()函数中的第一个参数是一个由要从中抽样的元素组成的向量。在这里，这个向量是1到数据框中观测的数量，**第二个参数**是要抽取的**元素数量**，第三个参数表示无放回抽样。

# 使用SQL 语句操作数据框

- 安装sqldf包  
install.packages("sqldf")
- 使用sqldf()函数在数据框上使用SQL中的**SELECT**语句。
- 例（注：mtcars包含于R基本安装中），  
library(sqldf)  
newdf <- **sqldf**("select \* from mtcars where carb=1 order by mpg")

# 数据的整合和重构：转置

- **转置**：反转行和列。使用**函数t()**。

- 例，

**#取mtcars数据集的子集cars**

**cars <- mtcars[1:5, 1:4]**

**#对cars转置**

**t(cars)**

# 数据的整合和重构： 分类汇总

- 分类汇总的函数为**aggregate**(x, **by**, FUN)
- 其中x是待分类汇总的数据对象； **by**是一个**变量名组成的列表**，用于对原有观测进行分类； **FUN**是用来计算描述性统计量的函数，它将被**用来计算新观测中的值**。例，

```
options(digits=3) #保留3位有效数字
```

```
attach(mtcars)
```

```
aggdata <- aggregate(mtcars, by=list(cyl,gear),  
                      FUN=mean, na.rm=TRUE)
```

注： **by**中变量必须在一个列表中，即使只有一个变量。



# 数据的整合和重构：reshape包

- reshape包是一套整合和重构数据集的**万能工具**。不过其入门有点难度
- 使用步骤
  - **首先**将数据**melt**，以使每一行都是一个唯一的标识符—变量组合。
  - **然后**将数据**cast**为任何形状。在重铸过程中，可以使用任何函数对数据进行整合。

# 准备测试数据

```
mydata <- read.table(header=TRUE, sep=" ",  
text="
```

```
ID Time X1 X2
```

```
1 1 5 6
```

```
1 2 3 5
```

```
2 1 6 1
```

```
2 2 2 4
```

```
")
```

The original dataset (mydata)

ID	Time	X1	X2
1	1	5	6
1	2	3	5
2	1	6	1
2	2	2	4

# melt

- **melt**将数据重构为这样一种格式：每个测量变量独占一行，行中带有要唯一确定这个测量所需的标识符变量。

# Using the reshape2 package

```
library(reshape2)
```

# melt data

```
md <- melt(mydata, id=c("ID", "Time"))
```

# melt后的数据集

The original dataset (mydata)

ID	Time	X1	X2
1	1	5	6
1	2	3	5
2	1	6	1
2	2	2	4

The melted dataset

ID	Time	Variable	Value
1	1	X1	5
1	2	X1	3
2	1	X1	6
2	2	X1	2
1	1	X2	6
1	2	X2	5
2	1	X2	1
2	2	X2	4

# cast

- **cast()**函数读取已融合的数据，并使用你提供的公式和一个（可选的）用于整合数据的函数将其重塑。调用格式为：

`newdata <- cast(md, formula, FUN)`

- 其中的md为已融合的数据，**formula**描述了想要的最后结果，而**FUN**是（可选的）数据整合函数。
- 例子见下页。

# Reshaping a Dataset

## With Aggregation

按id分别求variable均值

`cast(md, id~variable, mean)`

ID	X1	X2
1	4	5.5
2	4	2.5

(a)

按time分别求variable均值

`cast(md, time~variable, mean)`

Time	X1	X2
1	5.5	3.5
2	2.5	4.5

(b)

按id分别求time均值

`cast(md, id~time, mean)`

ID	Time1	Time2
1	5.5	4
2	3.5	3

(c)

mydata

ID	Time	X1	X2
1	1	5	6
1	2	3	5
2	1	6	1
2	2	2	4

`md <- melt(mydata, id=c("id", "time"))`

ID	Time	Variable	Value
1	1	X1	5
1	2	X1	3
2	1	X1	6
2	2	X1	2
1	1	X2	6
1	2	X2	5
2	1	X2	1
2	2	X2	4

## Without Aggregation

`cast(md, id+time~variable)`

ID	Time	X1	X2
1	1	5	6
1	2	3	5
2	1	6	1
2	2	2	4

(d)

`cast(md, id+variable~time)`

ID	Variable	Time1	Time2
1	X1	5	3
1	X2	6	5
2	X1	6	2
2	X2	1	4

(e)

`cast(md, id~variable+time)`

ID	X1 Time1	X1 Time2	X2 Time1	X2 Time2
1	5	3	6	5
2	6	2	1	4

(f)

# 控制流：条件与循环

# 控制流：条件与循环

- R拥有一般现代编程语言的标准控制结构。
- 基本概念：
  - 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。



# 条件执行

- 在条件执行结构中，一条或一组语句仅在满足一个指定条件时执行。
- 条件执行结构包括
  - if-else
  - ifelse
  - switch

# if-else结构 (1/3)

- 语法:
  - **if** (cond) statement
  - **if** (cond) statement1 **else** statement2
- 例,
  - 如果grade是字符向量, 则会被转换为因子。  
**if** (is.character(grade)) grade <- as.factor(grade)
  - 如果grade**不是**因子, 则会被转换为因子。  
**if** (!is.factor(grade)) grade <- as.factor(grade)  
**else** print("Grade already is a factor")

# if-else结构 (2/3)

- 注意:

正确!

```
score<-50
if(score>60){
    print("good")
}else{
    print("bad")
}
```

错误!

```
score<-50
if(score>60){
    print("good")
}
else{
    print("bad")
}
```

脚本语言实际上是一句一句地解释。到这R认为这是一个完整的if语句,已经结束。因此下面的else报错  
**Error:**  
**unexpected 'else' in "else"**

即: else不能出现在行首

# if-else结构（3/3）

- **if-else**可以进行嵌套。但仍要注意保证**else**的匹配问题：即不能在单独行的最开始写“**else**”。

# ifelse结构

- **ifelse**结构是if-else结构比较紧凑的向量化版本，其语法为：

**ifelse**(cond, statement1, statement2)

- 例，
  - **ifelse**(score > 0.5, print("Passed"), print("Failed"))
  - outcome <- **ifelse** (score > 0.5, "Passed", "Failed")

# switch结构

- 语法: `switch(expr, ...)`。例:  
    `feelings <- c("sad", "afraid")`  
    for (`i` in feelings)  
        print(  
            `switch(i,`  
                happy = "I am glad you are happy",  
                afraid = "There is nothing to fear",  
                sad = "Cheer up",  
                angry = "Calm down now"  
            )  
        )

# for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。语法为：

**for (var in seq) statement**

- 例1：

```
for (i in 1:10) print("Hello")
```

- 例2：

```
for (i in 1:10) {  
    print("hello")  
    print("haha")  
}
```

# while循环

- **while**循环重复地执行一个语句，直到条件不为真为止。语法为：

**while** (*cond*) statement

- 例：

```
i <- 10
```

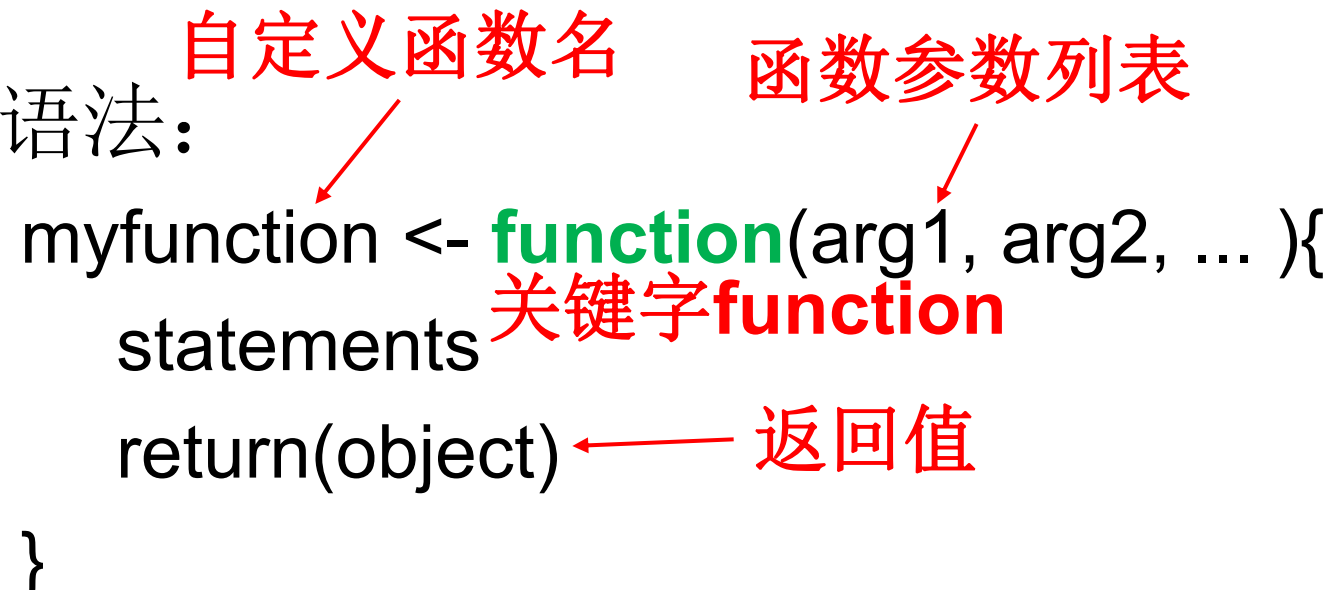
```
while (i > 0) {print("Hello"); i <- i - 1}
```

- 注意：在处理大数据集中的行和列时，**R中的循环可能比较低效费时**。只要可能，最好联用R中的内建数值/字符处理函数和**apply**族函数。



用户自编函数

# 用户自编函数（及例1）

- 语法：  




```
myfunction <- function(arg1, arg2, ... ){  
  statements  
  return(object)
```

- 例1：

```
mysum<-function(x,y){  
  z<-x+y  
  return(z)  
}  
mysum(1,2)
```

# 用户自编函数（例2）

#此函数输出当前日期，可让用户选择输出格式，在函数声明中为参数指定的值将作为其默认值。

```
mydate <- function(type="long") {  
  switch(type,    
    long = format(Sys.time(), "%A %B %d %Y"),  
    short = format(Sys.time(), "%m-%d-%y"),  
    cat(type, "is not a recognized type\n"))  
}  
mydate("long")  
mydate("short")  
mydate()  #调用默认类型"long"  
mydate("medium")
```