### 环境安装

Windows: 当前版本3.0，从2.15.3到3.0，由于底层的兼容性问题，有很多包都不能用了，所以对于初学者，推荐使用2.15.3

**CRAN镜像的网址：**

<http://ftp.ctex.org/mirrors/CRAN/> CTEX.ORG

<http://cran.csdb.cn/> Computer Network Information Center, CAS, Beijing

<http://mirror.bjtu.edu.cn/cran> Beijing Jiaotong University, Beijing

<http://cran.dataguru.cn> Dataguru (a, Guangzhou)

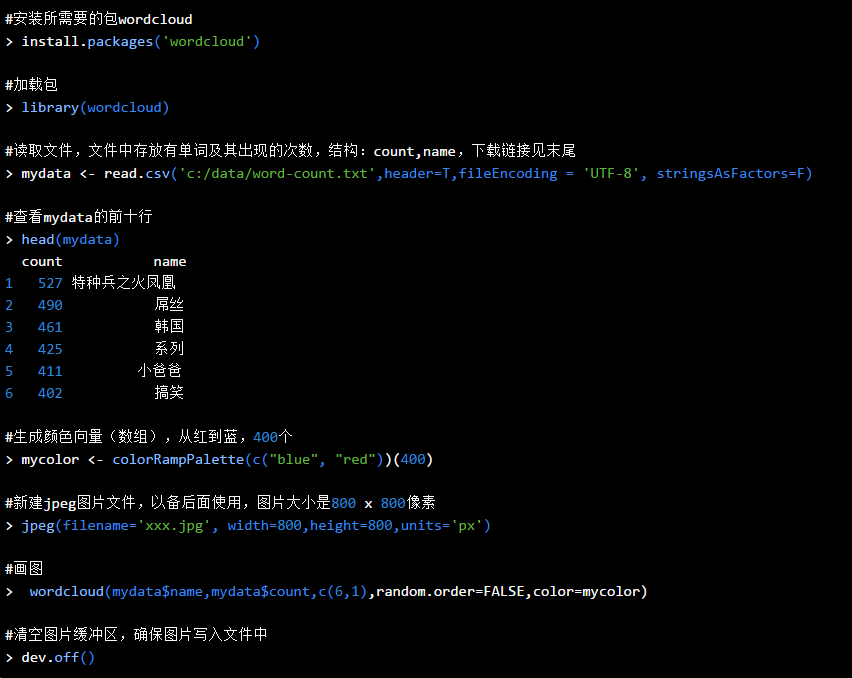
<http://mirrors.ustc.edu.cn/CRAN/> University of Science and Technology of China

<http://mirrors.xmu.edu.cn/CRAN/> Xiamen University

Linux: 高效开发环境RStudio

入门教程：<http://www.dataguru.cn/portal.php?mod=view&aid=3251#comment>

### 测试：画一个标签云



### 测试：画社交关系图

借助R语言的igraph包将用户的社交关系以图形化的方式展现出来，以歌手为例，据根据用户分享的歌曲，使用协同过滤算法计算歌手之间的关联关系。使用了R语言的可视化包igraph，具体代码如下：



### Rtools使用

R通过调用各种package可以实现很多新的应用。丰富的package资源正是R如此受追捧的一个很重要的原因。每位R User在使用R的时候都会积累一些自己编写的常用的函数，将它们打成一个package是一个非常好的办法。虽然做成一个标准的R包并发布在CRAN上比较麻烦，但是如果仅仅是做成一个简单的package供自己使用是非常容易的

**编写包**：

假设我们有自定义函数f1和f2以及数据矩阵d1，将其做成package，名字为test1，使用package.skeleton会在工作路径下自动生成package的框架。如下所示：

setwd("D:/RData/workdir")

f1 <- function(aa){return (aa)}

f2 <- function(aa){return(aa+2)}

d1 <- matrix(1:6,2,3)

package.skeleton(name="test1", list=c("f1","f2","d1"))

会在D:\RData\workdir内发现新生成了文件夹test1，里面包含data、man和R三个子文件夹以及DESCRIPTION和Read-and-delete-me这两个文件。其中Read-and-delete-me可以通过文本工具打开阅读后删除，DESCRIPTION可以参照标准的规范进行修改后保存（不做修改也不影响正常运行）。data文件夹保存数据（本例中对应d1），R文件夹保存函数（本例中对应f1和f2），man文件夹存放.Rd文件，用来生成帮助文件。我们只需要修改每个.Rd文件就行，用文本工具打开并定位到“title”开头这一行，写入该函数或者数据的名称即可（比如d1），保存关闭。其实按照标准的做法，下面的description和examples之类的都要补全，但是只有title是必须补全的，否则编译会出错。

**编译包：**

打开CMD，定位到D:\RData\workdir，进行编译，如下所示：

D:

cd D:\RData\workdir

Rcmd build test1

注意：如果没有把R语言的路径添加到环境变量中，运行Rcmd是会报错的，这时有两个办法：

A。把Rcmd.exe所在路径添加到系统路径中

B。运行时添加Rcmd.exe的全路径，比如：

"C:\Program Files\R\R-2.15.0\bin\i386\Rcmd.exe" build test1

运行之后，查看D:\RData\workdir 目录下多了一个文件：

test1\_1.0.tar.gz

**注意：**按照之前帖子里的说法是运行Rcmd build --binary test1 之后会生成zip文件，但是我添加了--binary参数之后报错，最终生成的也是tar.gz格式的，不知道是为什么），tar.gz格式的无法RGui环境中直接添加（RGui环境中，只能选择Zip包安装）。

**输入/输出**

> Print(“Hello world”)

> x=1:10

> x=x+1

> x

[1] 2 3 4 5 6 7 8 9 10 11

x=1:10表示产生 1 2 3 4 5 6 7 8 9 10这些数字，然后x=x+1表示每个数都加1，最后输入x后表示显示x这个变量。

### 常用命令

以下是R编程常用的一些命令（不包括统计分析与作图）。

**#运算符**

数学运算：

+，-，\*，/，^，%%。(加，减，乘，除，乘方，求余。)

比较运算：

>，<，>=，<=，==，!=。(大于，小于，大于等于，小于等于，等于，不等于。)

逻辑运算：

&，|，!。(与，或，非。)

##################################

#初始基本操作

##################################

> getwd() #获取工作目录。

> setwd("F:/R/myfun") #设定工作目录为F:/R/myfun。

> help(com)

> ?com

#获得对命令com的说明。

> example(com)

#命令com的使用实例。

> args(com)

#查看命令com的变量格式。

> library()

#列出已安装的包。

> library(AER)  #或用

> require(AER)

#载入包AER。

> library(help = AER)

#获取包AER的信息。

> detach(package:AER)

#去除载入的包AER。

> search()

#列出已载入的包。

> data()

#列出已载入的包中的所有数据集。

> data(package = .packages(all.available = TRUE))

#列出已安装的包中的所有数据集。

> try(data(package = "AER") )

#列出包AER里的所有数据集。

> data(Affairs, "BankWages")

#读入数据集Affairs和BankWages(这些数据集需已在data()中列出)。

> help(Affairs)

#获取数据集Affairs的信息。

> attach(Affairs)

#贴上数据集Affairs，其作用是可以直接对数据集里的变量进行操作。

> detach(Affairs)

#上述操作的逆操作。

##################################

#对一般对象的基本操作

##################################

>objects()

>ls()

#列出所有对象。

> mode(x)

#查看对象x的模式：空，数值，字符，逻辑，复数，列表，函数(NULL，numeric，character，logical，complex，list，function)。

> class(x)

#查看对象x的类型：除了mode里列出的几种类型外，还有整数，矩阵，因子，阵列，数据框，时间序列(integer，matrix，factor，array，data frame，ts)等其他类型。mode主要用于区别数据存放的方式，而class是一种更细微的分类方式，比如矩阵，就是一种更“有序”的数据存放方式。此命令比mode常用。

> as.matrix(x)

#把对象x转为矩阵型。

> as.numeric(x)

#把对象x转为数值型。

> str(x)

#查看对象x的结构。str是structure的缩写。

> rm(x)

#移除对象x。

> rm(list=ls(all=TRUE))

#移除所有对象。

##################################

#与向量有关的基本操作

##################################

> x = c(1,2,4)

#生成元素依次为1,2,4的向量x，这里的c是concatenate的意思。注意其类型是数值，不是矩阵。

> x = c("a","b","cd")

#生成元素依次为"a","b","cd"的字符向量x。

> x = paste("a","b","cd")

#生成"a b cd"的字符x。

> x[a:b]

#向量x的第a到b个元素。

> x[-i]

#剔除向量x第i个元素所得的向量。

> length(x)

#向量x的长度。

> x = seq(a, b, length = n)

#生成以一个n维数值型向量x，第一个元素为a，最后一个元素为b，中间元素依次等距递增。(假设a<b)

> x = seq(a, b, c)

#生成一个数值型向量x，第一个元素为a，其后元素依次加c，直到最后一个元素加c大于b。(假设a<b)

> x = seq(a)

#从1开始生成一个递增或递减数值型向量x，最后一个元素绝对值为小于等于|a|的最大整数。

> x = a:b

#生成一个从a递增(减)到b的数值型向量x。(a和b都是整数)

> x = rep(v, n)

#对向量v进行n次复制生成新的向量x。

> x = rep(v, each = n)

#依次对向量v的每个元素复制n此生成新的向量x。

> x = round(v)

#生成一个向量x，其中每个元素是v对应元素的最近整数。

> order(x)

#获得向量x第i大元素在向量中的位置。

> rank(x)

#获得向量x每个元素大小位置。

> sort(x)

#对向量x从小到大进行排序。降序：sort(x, decreasing = TRUE)。

> tapply(x,f,g)

#根据因子f对向量x分类执行函数g。

> split(x,f)

#向量x按因子f分类。

> diff(x)

#返回向量x的差分向量。

> cumsum(x)

#返回向量x的累加向量。

##################################

#与矩阵有关的基本操作

##################################

> M = matrix(0,c(m,n))

#生成m行n列的0矩阵。

> M = rbind(X,Y)

#按行合并矩阵X和Y形成新矩阵M。(X和Y列数需相同）

> M = cbind(X,Y)

#按列合并矩阵X和Y形成新矩阵M。(X和Y行数需相同）

> colnames(M)

#矩阵M的列名。

> rownames(M)

#矩阵M的行名。

> nrow(M)

#矩阵M的行数。

> ncol(M)

#矩阵M的列数。

> diag(M)

#矩阵M的对角线元素形成的向量

> M = diag(x)

#生成以向量x为对角线元素，其他位置元素为0的矩阵M。

> dim(M)

#矩阵M的维度。

> M[i,]

#矩阵M第i行。(数值型)

> M[i,,drop = FALSE]

#矩阵M第i行。(矩阵型)

> M = M[-i,]

#删除矩阵M第i行。

> M[,j]

#矩阵M第j列。

> M[i,j]

#矩阵M第i行j列元素。

> t(M)

#矩阵M的转置。若M为数值型向量，则t(M)为矩阵型行向量。

> X%\*%Y

#矩阵X乘矩阵Y。若Y是数值型的向量，R会自动判断其为行向量还是列向量。若X与Y为维度匹配的数值型向量，则返回的是矩阵型向量的内积。

> x%o%y

#数值型向量x与y的外积(矩阵型)。

> X\*Y

#矩阵X与矩阵Y的Hadamard乘积。加、减、除、求余的规则和乘相同，即相同位置的元素进行运算。

> eigen(M)$val

> eigen(M)$vec

#求矩阵M的特征值和特征向量。

> solve(M)

#矩阵M求逆。

> solve(A,b)

#求解线性方程Ax=b。

> apply(M, dimcode, f, fargs)

#对矩阵M的行(dimcode=1)或列(dimcode=2)依次进行函数f操作，f的变量(arguments)方正fargs里。

##################################

#与列表有关的基本操作

##################################

> L = list(a = , b = , c = ,...)

#建立列表L。

> L$a

> L[[a]]

#返回列表L里的对象a。

> L$a = NULL

#去除列表L里的对象a。

> names(L)

#列出列表L里的对象名。

> unname(L)

#去掉列表L里的对象名。

> lapply(...)

#list apply。功能与apply类似(参考上面的apply)，用于列表型数据。

> sapply(...)

#simplified apply。功能与lapply类似，区别在于函数结果的类型不是列表(list)。

##################################

#与数据框有关的基本操作

##################################

#数据框是一种特殊的列表，所以对列表适用的函数往往对数据框也适用。此外，数据框也有矩阵型数据的特征，所以一些适用于矩阵型数据的函数，不如rbind，cbind，apply等也可以作用在数据框上。

> Data = data.frame(...)

#创建数据框Data。

> fix(Data)

#编辑数据框Data。

> head(Data)

#显示数据框Data的前几行。

> attribute(Data)

#列出数据框Data的组成部分。

> names(Data)

#显示数据框Data的变量名。

> row.names(Data)

#显示数据框Data的行名。

> Data$name1

#数据框Data中名为name1的变量。

> Data[i]

#数据框Data中第i个变量形成的数据框。

> merge(D1,D2)

#合并数据框D1和D2，需要D1和D2中有至少一个相同的变量。

##################################

#与逻辑型数据有关的基本操作

##################################

> is.data.frame(x)

#判断是否对象x是数据框。类似命令有is.ts(x)，is.numeric(x)等。

> all(x>a)

#判断是否对象x的每个元素都大于a。

> any(x>a)

#判断对象x的元素中是否存在一个大于a。

> x>y

#判断x的每个元素是否大于y的每个元素。

> x[x>a]

#向量x中大于a的元素组成的新向量。

> subset(x, x>a)

#向量x中大于a的元素组成的新向量。与上面例子的区别在于若向量元素里有NA，上面的例子会保留在结果中，而subset命令会剔除掉。

> which(x, x>a)

#返回向量中大于a的元素的位置。

> x = ifelse(b, u, v)

#生成一个与b(逻辑向量)维度相同的数值向量，若b[i]为TRUE，则x[i]为u，反之为v。

### 读取Excel文件

read.table(),read.csv(),read.delim()直接读取EXCEl文件时，都会遇到一下问题：“在读取‘.xls’的TableHeader时遇到不完全的最后一行”。

**方法1：**xls另存为csv格式然后用read.csv：

data<-read.csv("D:\\work\\data\\1.csv",header = F)

**方法2：**xls另存为txt格式然后用read.table：

data<-read.table("D:\\work\\data\\1.txt",header = F)

**方法3**：打开EXCEL，全选里面的内容，点击复制，然后在R中输入一下命令：

data <- read.table("clipboard", header = T, sep = '\t')

### 数据类型

**（1）向量（vector）**

我们通常接触的数据主要是数字、字符和逻辑（真和假，是或不是）类型的。数据有一个个的，也有一串串一批批的。在R里面，最基本的数据类是向量，即一串有序数据；但vector是虚拟类，没有父类型，它包含了在其他语言里面常说的基本数据类型如整型、字符型和逻辑型等:

> getClass("vector")

R语言处理数据的最基本单位是向量，而不是原子数据。所以向量又称为原子向量（atomic vector），R语言的数据单位里面它最小（也最大，没有谁是它的父母）。但由于vector是虚拟类，不管用什么方式你都不可能获得类型名称叫“vector”的对象，只能获得它的直接子类的对象。

一个向量可以是一串数字（n个数字，向量长度为n），也可以是1个数字（n个数字，向量长度为1）：

> x <- c(1,2,3)

> x

[1] 1 2 3

> y <- c("赵匡胤", "钱学森", "孙思邈")

> y

[1] "赵匡胤" "钱学森" "孙思邈"

c( )是R的一个函数，表示将括号里面的内容连接起来成为一个向量。R提供了一些产生特殊向量的函数，如seq( )和rep( )，具体用法直接在R里面先输入问号（?）和函数名去查询。vector是虚拟类，本身不指定数据的存储类型，但赋值以后就马上会有数字型（numeric）、字符型（character）、逻辑型（logical）等实际类别，比如上面的变量x和y，用class( )函数获得的类型分别是数值型和字符型。

一个向量只属于一种类型，如果改变了一个元素的值可能会改变该向量的类型。

向量元素的引用/提取用下标法如 x[2]，***R语言的下标从1开始编号（而不是0）。***

**（2）因子（factor）**

R定义了一类非常特殊的数据类型：因子。比如我们的实验获得了10个数据，前5个数据来自对照样品CK，其余属于处理样品TR，R语言中可以用下面方法标识这10个数据的样品属性：

> sample <- rep(c("CK","TR"), each=5)

> sample <- factor(sample)

> sample

[1] CK CK CK CK CK TR TR TR TR TR

Levels: CK TR

因子的种类称为水平（level）。上面的样品sample因子有两个水平：CK和TR。因子类数据很特殊：

使用因子类数据是因为R是针对统计应用的语言。使用因子以后，数据的统计会完全不同。比如上面的两个样品10个测定数值如果是：

> value <- rnorm(10)

> value

[1] 1.44368380 -1.99417898 0.60279037 0.75186610 1.08372729 -0.16189030

[7] -0.05617801 1.03601538 -0.87932814 -0.32429184

求样品的平均值就可以这么做：

> tapply(value, sample, mean)

CK TR

0.37757771 -0.07713458

gl( )函数也可以方便地产生因子：

> sample <- gl(2, 5, labels = c("CK", "TR"))

> sample

[1] CK CK CK CK CK TR TR TR TR TR

Levels: CK TR

**（3）矩阵（matrix）**

矩阵的继承关系比较复杂，它和数组（array）的关系既是父亲又是儿子，还是孙子：

如果你愿意，也可以把矩阵称为数组，但事实上它们是不同的类。向量数据可以转成矩阵，下面代码将10个元素的x转成2行5列的矩阵：

> x <- 1:10

> dim(x) <- c(2,5)

> x

[,1] [,2] [,3] [,4] [,5]

[1,] 1 3 5 7 9

[2,] 2 4 6 8 10

> dim(x)

[1] 2 5

dim( )是一个函数，它获取或设置数据的维度。注意x数据的行列排列顺序：先列后行。但是矩阵内数据的下标读取方式是先行后列。x[2, 1]是第2行第1列的值，x[2, ]表示第2行的所有数据，x[ ,2]表示第2列的所有数据。

> x[2,1]

[1] 2

> x[2,]

[1] 2 4 6 8 10

> x[ ,2]

[1] 3 4

把1个向量转成矩阵还可以使用matrix( )函数，参数nrow设置行数，ncol设置列数：

> matrix(1:10, nrow=2)

[,1] [,2] [,3] [,4] [,5]

[1,] 1 3 5 7 9

[2,] 2 4 6 8 10

几个长度相同的向量也可以合并到一个矩阵，cbind( )函数将每个向量当成一列（按列）合并，rbind( )按行合并：

> x <- 3:6

> y <- 4:7

> z <- 1:4

> cbind(x,y,z)

x y z

[1,] 3 4 1

[2,] 4 5 2

[3,] 5 6 3

[4,] 6 7 4

> rbind(x,y,z)

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

x 3 4 5 6

y 4 5 6 7

z 1 2 3 4

不同向量的数据类型要相同，否则转换成矩阵后数据类型会变样。下面代码将height内的数字全都转成了字符类型，这可能不是你想要的结果：

> name <- c("赵匡胤", "钱学森", "孙思邈")

> height <- c(172, 175, 168)

> info <- rbind(name,height)

> info

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

name "赵匡胤" "钱学森" "孙思邈"

height "172" "175" "168"

> class(info)

[1] "matrix"

身高是数值，但跟姓名混合组成矩阵后就变成字符型了（输出结果中用双引号引起来）。矩阵元素可通过下标引用,多维矩阵可以只用一个下标，请注意info[3]获得的数据：

> info[1,]

[1] "赵匡胤" "钱学森" "孙思邈"

> info[,1]

name height

"赵匡胤" "172"

> info[3]

[1] "钱学森"

**（4）列表（list）**

列表由向量直接派生而来，nameList是它的子类，listOfMethods是它家孙子。那么列表是什么样子的呢？看这：

> gene <- list(agi="AT1G00010", gene.mode=c("AT1G00010.1", "AT1G00010.2", "AT1G00010.3"), expression=matrix(1:10, ncol=2))

> gene

$agi

[1] "AT1G00010"

$gene.mode

[1] "AT1G00010.1" "AT1G00010.2" "AT1G00010.3"

$expression

[,1] [,2]

[1,] 1 6

[2,] 2 7

[3,] 3 8

[4,] 4 9

[5,] 5 10

列表可以组合不同的数据类型，甚至可以是其他列表，各组成数据的类、长度、维数都可以不一样。

**（5）数据框（data frame）**

R语言中，一个矩阵内的数据类型要求都要相同，这对生物类数据不大适用，因为我们的数据经常是既有数字又有字符类标记。R语言提供了另外一种更灵活的数据类型：数据框。可以将几个不同类型但长度相同的向量用data.frame( )函数合并到一个数据框，它的模样就像二维数组。但要注意：合并的几个向量长度必需一致。

> name <- c("赵匡胤","钱学森","孙思邈")

> height <- c(172,175,168)

> info <- data.frame(name,height)

> info

name height

1 赵匡胤 172

2 钱学森 175

3 孙思邈 168

用as.data.frame( )函数可以将二维矩阵转成数据框。虽然数据框的外观和二维矩阵差不多，但它却不是从矩阵而是从列表派生来的，它是数据（.data）是列表数据，列名称（names）就是列表中各项的名称，另外还有行名称（row.names）：

数据框的每列是一个向量，称为列向量。列向量只有两种类型，要么是数字型，要么是因子型。从文件读取或其他类型数据转换成数据框的数据，如果不是数值型，会被强制转换成因子型。有时候数值型（尤其是整型）向量也会被转成因子，这点应该注意。

数据框可以用数字下标取数据，也可以用列名称下标取数据，但是两种方式所获数据的类型是不一样的，按列名称下标方式取得的数据仍然是数据框：

> info[,1]

[1] 赵匡胤 钱学森 孙思邈

Levels: 钱学森 孙思邈 赵匡胤

> class(info[,1])

[1] "factor"

> info["name"]

name

1 赵匡胤

2 钱学森

3 孙思邈

> class(info["name"])

[1] "data.frame"

为什么要注意这个区别？因为看起来像是同样的数据，在一些对类型要求很严格的操作（比如作图）中得到完全不一样的结果。数据框还有一种数据提取方式，得到因子或向量：

> class(info["name"])

[1] "data.frame"

> class(info$name)

[1] "factor"

> class(info$height)

[1] "numeric"

**（6）特殊值数据**

为确保所有数据都能被正确识别、计算或统计等，R定义了一些特殊值数据：

NULL：空数据

NA：表示无数据

NaN：表示非数字

inf：数字除以0得到的值

判断一个object （x）是不是属于这些类型有相应的函数：

is.null(x)

is.na(x)

is.nan(x)

is.infinite(x)

**（7）获取数据类型信息的一些有用函数**

R语言的对象“类”很多，虽然我们不可能一一去详细学习，但接触到一类新数据时我们需要了解一些基本信息才能进行进一步的操作。R提供了一些非常有用的方法（函数）。

getClass( )函数我们前面已经见过了，它的参数是表示类的字符串。

class( )可获取一个数据对象所属的类，它的参数是对象名称。

str( )可获取数据对象的结构组成，这很有用。

mode( )和storage.mode( )可获取对象的存储模式。

typeof( )获取数据的类型或存储模式。

要了解这些函数能干什么可以在R里面查询，方法是用问号加上面的函数名就可以，如：?str

### 横向数据合并merge

> id <- c(1:4)

> name <- c("Jim","Tony","Lisa","Tom")

> score <- c(89,22,78,78)

> s1 <- data.frame(id,name)

> s2 <- data.frame(id,score)

> s3 <- merge(s1,s2,by="id")

> s3

id name score

1 1 Jim 89

2 2 Tony 22

3 3 Lisa 78

4 4 Tom 78

### 纵向数据合并rbind

> id <- c(1,2,3)

> name <- c("Jame","Kevin","Sunny")

> s1 <- data.frame(id,name)

> id <- c(4,5,6)

> name <- c("Sun","Frame","Eric")

> s2 <- data.frame(id,name)

> s3 <- rbind(s1,s2)

> s3

id name

1 1 Jame

2 2 Kevin

3 3 Sunny

4 4 Sun

5 5 Frame

6 6 Eric

### quantmod 包

quantmod 包默认是访问 yahoo finance 的数据，其中包括上证和深证的股票数据，还有港股数据。上证代码是 ss，深证代码是 sz，港股代码是 hk

比如茅台：6000519.ss，万科 000002.sz，长江实业 0001.hk

在R的控制台里使用如下命令：

> library(quantmod)

> setSymbolLookup(WK=list(name='000002.sz',src='yahoo'))

> getSymbols("WK")

[1] "WK"

> chartSeries(WK)

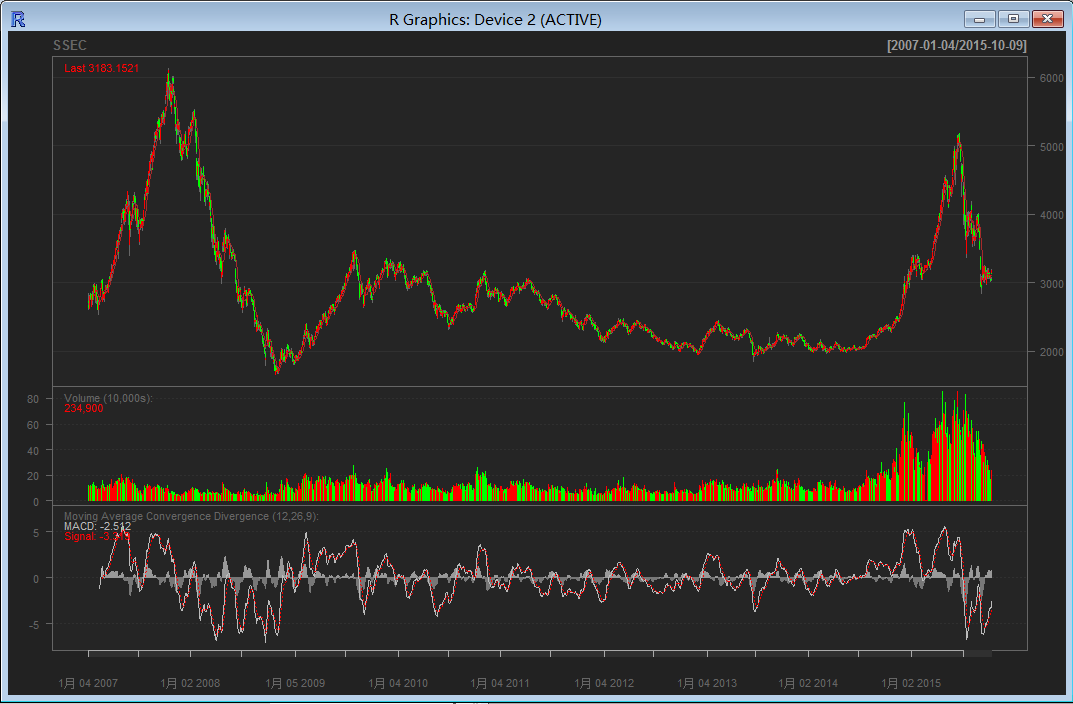
万科股票测试：



上证指数：

> getSymbols("^SSEC")

> chartSeries(SSEC, up.col='red', dn.col='green', TA="addVo(); addMACD(); addSMA(n=10)")



### 数据库操作

R通过RODBC方式进行数据存取的主要函数如下：

函数名称 函数描述

odbcConnect(dsn, uid="", pwd="") # 建立并打开连接

sqlFetch(channel, sqtable) # 从数据库读取数据表，并返回一个数据框对象

sqlQuery(channel, query) # 向数据库提交一个查询，并返回结果

sqlSave(channel, mydf, tablename = sqtable, append = FALSE) # 将一个数据框写入或更新(append=True)到数据库

sqlDrop(channel, sqtable) # 从数据库删除一个表

sqlClear(channel, sqtable) # 删除表中的内容

sqlTables(channel) # 返回数据库中表的信息

sqlColumns(channel, sqtable) # 返回数据库表sqtable列的信息

close(channel) # 关闭连接

连接mysql:

1. 安装mysql odbc
2. 配置odbc
3. 读取数据：

> library(RODBC)

> ch <- odbcConnect('mysql\_localhost',uid='root',pwd='pass')

> data <- sqlFetch(ch,'dcc\_task\_result')