Part 1 数据获取

郎大为 J.D. Power

数据获取　1小时

摘要： R 语言如何从各种方式读取数据，连接数据库，通过sql语句调用数据，从本地读取excel等各种文件数据。

- 本地文档的读写
- 连接数据库
- 获取网络数据
- 其他的函数

本地文档的读写

控制台的输出

用户可以使用print或只输入对象名，即在屏幕上得到显示。如果需要对输出有格式上的要求，则利用format函数进行调整。

```
set.seed(1)
out <- data.frame(x1=runif(3)*10,x2=c('a','b','c'))
## print(out)
out <- format(out,digits=3)
out
```

```
    x1 x2
1 2.66  a
2 3.72  b
3 5.73  c
```

控制台的输出

```
cat(paste(out$x2, out$x1, sep='='), sep='\n')
```

```
a=2.66
b=3.72
c=5.73
```

控制台的输入

用户也可以通过控制台进行交互输入。readline函数可以输入单个字符串数据，而scan函数则可以输入多个数值数据。

```
x <- readline()
x <- scan()
```

本地文本输出

```
output <- file('output.txt')
cat(1:100,sep='\t',file=output)
close(output)
```

使用file函数建立一个文件连接。前面使用过的cat函数可以直接将数据对象输出到文件连接中，如果文件中已经有内容，可以在cat函数中设置append参数为真，即表示是新增在文件尾部。

本地文本输入

```
output <- file('output.txt')
input <- scan(file=output)
close(output)
```

要注意的是scan读入的内容应该是一致的类型，不可能同时读入字符和数值。

字符串的输入输出

如果只需要处理字符串的输入输出，可以考虑使用readLines和writeLines这一对函数。

```
output <- file('output.txt')
writeLines(as.character(1:12),con=output)
input <- readLines(output)
```

小练习

读取用户R语言已经安装的每个扩展包的DESCRIPTION文件。

```
path <- .libPaths()[1]
doc.names <- dir(path)
doc.path <- sapply(doc.names,function(names) paste(path,names,'DESCRIPTION',sep='/'))
doc <- sapply(doc.path, function(doc) readLines(doc))
```

在处理这种大量外部文件时，可以充分利用R来自动化处理。

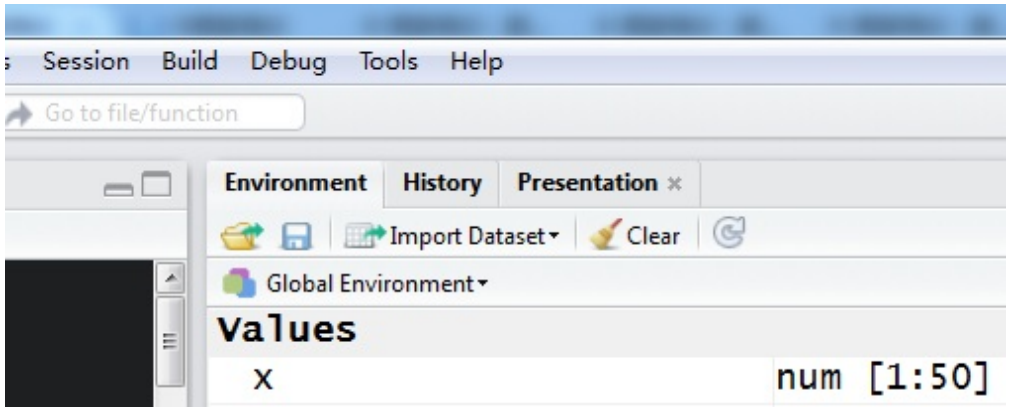
数据表的读写

read.table和write.table

```
write.table(iris, file='iris.csv',sep=',')
data <- read.table(file='iris.csv',sep=',')
data <- read.table(file=file.choose() ,sep=',')
```

```
data <- read.table('clipboard')
```

另一种更方便的导入方法是利用Rstudio的功能，在Environment面板选择“import dataset”也是一样的。



读取SPSS和SAS数据文件

foreign包中有大量读取外部数据的函数

- `statadata <- read.dta("C:/temp/statafile.dta")`
- `spssdata <- read.spss("C:/temp/spssfile.sav")`
- `sasdata <- read.xport("C:/temp/sasfile.xpt")`

数据库的读写

数据库的读写

- R 语言和关系型数据库的两种配合方式。
  - 用SQL来提取需要的数据，存为文本再由R来读入。
  - 将R与外部数据库连接，直接在R中操作数据库。
- 连接方式的两种选择：
  - ODBC方式，需要安装RODBC包并安装ODBC驱动
  - DBI方式，可以根据已经安装的数据库类型来安装相应的驱动

RODBC的函数

函数	描述
odbcConnect(dsn, uid="", pwd="")	打开ODBC数据库的链接
sqlFetch(channel, sqtable)	从数据库中读一张表，转成数据框
sqlQuery(channel, query)	提交一条SQL查询语句，返回结果
sqlSave(channel, mydf, tablename = sqtable, append = FALSE)	把数据框写入到数据库的表中
sqlDrop(channel, sqtable)	从数据库中删除一张表
close(channel)	关闭链接

windows下的连接准备

1. R下载安装RODBC包。
2. 下载安装MySQL ODBC。
3. 控制面板->管理工具->数据源（ODBC）->双击->添加->选中mysql ODBC driver一项;填写：data source name 一项填入你要使用的名字，自己随便命名，例如：mysql_data;
   - description一项随意填写，例如mydata
   - TCP/IP Server 填写服务器IP，本地数据库一般为：127.0.0.1
   - user 填写你的mysql用户名
   - password 填写你的mysql密码 -然后数据库里会出现你的mysql里的所有数据库，选择一个数据库。 确定。

SQLite

- SQLite是一款轻型的数据库，是遵守ACID的关系型数据库管理系统，它包含在一个相对小的C库中。
- 它是D.RichardHipp建立的公有领域项目。
- 占用资源较少
- 使用SQLite来展示R与数据库的连接
- http://www.sqlite.org (http://www.sqlite.org)

SQLite 下载安装

- 查看w3cschool的安装教程:
- http://www.w3cschool.cc/sqlite/sqlite-installation.html (http://www.w3cschool.cc/sqlite/sqlite-installation.html)

SQLite 与 R

- 使用RSQLite包来完成

```
## install.packages("RSQLite")
library(RSQLite)
```

- 操作数据:

```
db <- datasetsDb()
dbListTables(db)
dbReadTable(db, "CO2")
dbGetQuery(db, "SELECT * FROM CO2 WHERE conc < 100")
dbDisconnect(db)
```

SQLite 与 R

操作数据II

```
tmp <- dbConnect(SQLite(), dbname = "testDB.db")
dbListTables(tmp)
dbReadTable(tmp, "iris")
dbWriteTable(tmp, "arrests", datasets::USArrests)
dbGetQuery(tmp, "SELECT * FROM arrests limit 3")
dbGetQuery(tmp, "SELECT * FROM arrests WHERE Murder < 10")
dbGetQuery(tmp, "SELECT * FROM iris limit 3")
dbDisconnect(db)
```

Web数据抓取

结构化的网页数据抓取

之前提到的很多输入函数都可以直接读取网页上存放的文档。例如read.table可以读取那些保存为类EXCEL表格形式的数据文档。而有的结构化数据，直接是网页形式存在。比如美联储发布的汇率数据(http://www.federalreserve.gov/releases/h10/hist/)

```
library(XML)
# 美元兑澳元的汇率数据
url <- 'http://www.federalreserve.gov/releases/h10/hist/dat00_al.htm'
tables <- readHTMLTable(url,
                        stringsAsFactors = FALSE,
                        header=F)
data <- tables[[3]]
```

小练习

抓取中国地震局的数据 (http://data.earthquake.cn/datashare/globeEarthquake_csn.html) 对经纬度和深度进行统计描述，并找出深度最大的地区和地震常发的地区

```
library(XML)
webpage <-'http://data.earthquake.cn/datashare/globeEarthquake_csn.html'
tables <- readHTMLTable(webpage,stringsAsFactors = FALSE)
# 在Windows下用下面的命令
# tables <- readHTMLTable(webpage,stringsAsFactors = FALSE,encoding="UTF-8")
raw <- tables[[6]]
data <- raw[,3:5]
names(data) <- c('lan','lon','deep')
sapply(data,class)
data$deep <- as.numeric(data$deep)
summary(data$deep)
raw[which.max(data$deep),]
data <- transform(data,lan=as.numeric(lan),
                  lon=as.numeric(lon))
hist(data$lan,40)
hist(data$lon,40)
```

这数据将用到后续的字符串处理环节和可视化部分

非结构化的网页数据抓取

XML包中的htmlParse和getNodeSet

- htmlParse负责抓取页面数据并形成树状结构
- getNodeSet负责对抓取的数据根据XPath语法来选取特定的节点集合
- XPath语法简介 (http://www.w3school.com.cn/xpath/index.asp)
- kindle爬虫 (http://www.supstat.com.cn/blog/2015/03/31/amazon-kindle/)

使用API

豆瓣API

豆瓣电影API的文档 (http://developers.douban.com/wiki/?title=movie_v2)

```
https:        //api.douban.com/v2/movie/subject/5323968
```

```
library(RCurl)
library(XML)
library(RJSONIO)
id <- '5323968'
api <- 'https://api.douban.com/v2/movie/subject/'
url <- paste(api,id,sep='')
res <- getURL(url,.opts = list(ssl.verifypeer = FALSE))
reslist <- fromJSON(res)
str(reslist)
reslist$summary
reslist$images
imageurl <- reslist$images[2]
browseURL(imageurl)
download.file(imageurl,destfile='movie.jpg',mode = "wb")
```

案例与练习

练习一：抓取所有的250部最佳电影

练习二：使用API搜索电影得分

```
library(RCurl)
library(XML)
library(RJSONIO)
movieScoreapi <- function(x) {
  api <- 'https://api.douban.com/v2/movie/search?q={'
  url <- paste(api,x,'}',sep='')
  res <- getURL(url,
                .opts = list(ssl.verifypeer = FALSE))
  reslist <- fromJSON(res)
  name <- reslist$subjects[[1]]$title
  score <- reslist$subjects[[1]]$rating$average
  return(list(name=name,score=score))
}
movieScoreapi('僵尸世界大战')
```

其他函数

保存 Rdata

- .Rdata是R支持的数据格式,可以很方便的保存和调用
- 使用save函数完成保存
  - 可以保存多个R对象(数据,参数,模型,列表...)
- 使用load函数载入已经保存的Rdata

```
save(iris,my_fun,file = "demo.Rdata")
load("demo.Rdata")
```

时间对象的处理

- Date对象
- POSIXct POSIXt 对象

```
Sys.Date()
Sys.time()
system.time(Sys.sleep(2))
```

```
[1] "2016-10-07"
[1] "2016-10-07 16:52:06 CST"
   user  system elapsed
   0.00    0.00    2.03
```

时间对象的处理

- Date对象
- POSIXct POSIXt 对象

```
class(Sys.Date())
```

```
[1] "Date"
```

```
class(Sys.time())
```

```
[1] "POSIXct" "POSIXt"
```

时间对象的处理

基本运算

- Date对象
  - 加减单位为day
- POSIXct POSIXt 对象
  - 单位为seconds

```
today = Sys.Date()
format(today, "%d-%b-%Y")
```

```
[1] "07-Oct-2016"
```

```
Sys.Date() + 2
```

```
[1] "2016-10-09"
```

时间对象的处理

基本运算

```
time = Sys.time()
time
```

```
[1] "2016-10-07 16:52:08 CST"
```

```
time -3600
```

```
[1] "2016-10-07 15:52:08 CST"
```

```
seq(today, length.out=10, by="1 week")
```

```
 [1] "2016-10-07" "2016-10-14" "2016-10-21" "2016-10-28" "2016-11-04" "2016-11-11" "2016-11-18"
 [8] "2016-11-25" "2016-12-02" "2016-12-09"
```

时间对象的处理

基本运算

- difftime 计算两个时间之间的间隔
- units 设置间隔的单位

```
difftime(time1 = "2016-09-22", time2 = "2016-08-31",
         units = "weeks")
```

```
Time difference of 3.1429 weeks
```

```
as.Date("2016-09-22") - as.Date("2016-09-22")
```

```
Time difference of 0 days
```

```
## Error
## "2016-09-22" - "2016-09-22"
```

时间对象的处理

基本运算

- difftime 计算两个时间之间的间隔
- units 设置间隔的单位

```
interval = difftime(time1 = "2016-09-22",
         time2 = "2016-08-31",
         units = "weeks")
interval
```

```
Time difference of 3.1429 weeks
```

```
units(interval)<-"days"
interval
```

```
Time difference of 22 days
```

时间对象的处理

基本运算

- format 更换时间对象的格式
- strptime 将一个字符对象提取为时间对象

```
# ?format.POSIXct
# ?strptime
format(Sys.time(), "%A %B %d %X %Y %Z")
```

```
[1] "Friday October 07 16:52:08 2016 CST"
```

```
x <- c("1jan1960", "2jan1960", "31mar1960", "30jul1960")
z <- strptime(x, "%d%b%Y")
```

时间对象的处理

基本格式

FORMAT	MEANING	EXAMPLE	FORMAT	MEANING	EXAMPLE
%a	星期简写	Thu	%b	月份简写	Sep
%A	星期全称	Thursday	%B	月份全称	September
%u	星期数字	1-7(Mon=1)	%m	月份数字	(01-12)
%w	星期数字	0-6(Sun=0)	%W(%V%U)	周数	00-53
%H	小时	(00-23)	%d	日期	(01-31)
%I	小时	(01-12)	%e	日期	(1-31)
%p	AM/PM	AM/PM	%j	年中日期	(001-366)
%S	秒数	(00-61)	%M	分钟	(00-59)
%y	年份(简)	(00-99)	%Y	年份(全)	(2016)
%C	世纪	20			

时间对象的处理

定制格式

FORMAT	MEANING	EXAMPLE
%x	日期	%y/%m/%d
%X	时间	%H:%M:%S
%c	日期时间	%a %b %e %H:%M:%S %Y

时间对象的处理

其他运算

- 周末
  - weekdays
- 月份
  - months
- 季度
  - quarters
- julian

时间对象的处理

其他运算

```
weekdays(today)
```

```
[1] "Friday"
```

```
months(today)
```

```
[1] "October"
```

```
quarters(today)
```

```
[1] "Q4"
```

```
julian(today)
```

其他常用的函数

获取数据集信息

- ls()
- names()
- str()
- levels()
- dim()
- class()
- head(mydata, n=10)
- tail(mydata, n=5)

缺失值

- is.na() 检测是否为缺失
- 用索引操作来重编码
- 在计算中对NA的剔除
    - na.rm选项
    - complete.cases()
    - na.omit()

字符串处理

字符串处理概要

在文本数据挖掘日趋重要的背景下，在处理字符这种非结构化数据时，你需要能够熟练的操作字符串对象。

- 获取字符串长度：nchar()
- 字符串分割：strsplit()
- 字符串拼接：paste()
- 字符串截取：substr()
- 字符串替代：gsub()
- 字符串匹配：grep()

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。

```
fruit <- 'apple orange grape banana'
nchar(fruit)
```

```
[1] 25
```

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，需要设定分隔符。下面我们是用空格来作为分隔符将fruit分为四个元素。

```
strsplit(fruit,split=' ')
```

```
[[1]]
[1] "apple"  "orange" "grape"  "banana"
```

```
fruitvec <- unlist(strsplit(fruit,split=' '))
```

字符串拼接

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。另一个相似的函数paste0是设置无需分隔符的拼接。

```
paste(fruitvec,collapse=',')
```

```
[1] "apple,orange,grape,banana"
```

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。

```
substr(fruit, 1,5)
```

```
[1] "apple"
```

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。

```
gsub('apple','strawberry',fruit)
```

```
[1] "strawberry orange grape banana"
```

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。

```
grep('grape',fruitvec)
```

```
[1] 3
```

编码规范

编码规范

1. 文件名
2. 变量名
3. 单行长度
4. 缩进
5. 赋值
6. 空格
7. ";"
8. 注释

编码规范

- 不要(avoid)使用`attach`
- 函数报错用`stop`
- 避免使用S4类而使用S3类

编码规范

文件名

- 以.R结尾
- 以有意义的名称命名
- Google:
  - GOOD: predict_ad_revenue.R
  - BAD: foo.R

编码规范

变量名

- 不要(dont)使用下划线_或-
- 使用点或者驼峰式命名
- 函数名(FunctionName)不使用点. ,首字母大写
- 常数以k开头
    - good: variable.name variableName
    - bad: variable_name
- FunctionName
    - GOOD: CalculateAvgClicks
    - BAD: calculate_avg_clicks , calculateAvgClick

空格

- = 前后空格
- 运算符号前后空格
- , 之后空格
- GOOD:

```
tab.prior <- table(df[df$days.from.opt < 0, "campaign.id"])
total <- sum(x[, 1])
total <- sum(x[1, ])
```

空格

- BAD:

```
tab.prior <- table(df[df$days.from.opt<0, "campaign.id"])  # Needs spaces around '<'
tab.prior <- table(df[df$days.from.opt < 0,"campaign.id"])  # Needs a space after the comma
tab.prior<- table(df[df$days.from.opt < 0, "campaign.id"])  # Needs a space before <-
tab.prior<-table(df[df$days.from.opt < 0, "campaign.id"])  # Needs spaces around <-
total <- sum(x[,1])  # Needs a space after the comma
total <- sum(x[ ,1])  # Needs a space after the comma, not before
```

ifelse

- GOOD: `if (debug)`
- BAD: `if(debug)`

ifelse

- GOOD:

```
if (is.null(ylim)) {
  ylim <- c(0, 0.06)
}

if (is.null(ylim))
  ylim <- c(0, 0.06)
```

- BAD:

```
if (is.null(ylim)) ylim <- c(0, 0.06)
if (is.null(ylim)) {ylim <- c(0, 0.06)}
```

ifelse

- GOOD:

```
if (condition) {
  one or more lines
} else {
  one or more lines
}
```

- BAD:

```
if (condition) {
  one or more lines
}
else {
  one or more lines
}
```

ifelse

- GOOD:

```
if (condition) {
  one or more lines
} else {
  one or more lines
}
```

- BAD:

```
if (condition)
  one line
else
  one line
```

编码规范

其他问题

- 单行长度: 80
- 缩进
  - 用两个空格而不是tab
- 赋值
  - 用<-不用=
- ";"
  - 不使用!

编码规范

其他问题

- 注释
    - # Comments
- 单元检测
    - 命名为:originalfilename_test.R.