# Getting and cleaning data

# Week 1

Raw data -> Processing script -> tidy data

## 从网上下载东西

download.file(url, destfile, method, 。。。)

if (!file.exists("data")) {

dir.create("data")

}

fileUrl <- "https://data.baltimorecity.gov/api/views/dz54-2aru/rows.csv?accessType=DOWNLOAD"

download.file(fileUrl, destfile = "cameras.csv", method = "curl")

dateDownloaded <- date()

## reading local files

read.table()

read.csv()

## 读入excel文件

这个需要先加载一个包 library(xlsx)

read.xlsx(),read.xlsx2()

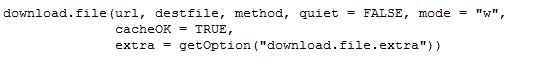
可以设置读入的行和列位置

注意：

在homework中有个从网上下载excel文件的操作，若直接使用

download.file(url,destfile)会发现下载的excel文件打不开(在windows系统上)

函数的用法是：





所以需要加入参数mode="wb"

## XML文件

例子1：

url='http://www.w3schools.com/xml/simple.xml'

doc<-xmlTreeParse(url, useInternal=T)

rootNode<-xmlRoot(doc)

xmlName(rootNode)

names(rootNode)

rootNode[[1]][[1]]

xmlSApply()

#用XPath的方法找到name

xpathSApply(rootNode,"//name",xmlValue)

例子2：抓取球队名称

### 例子2.2

url='http://espn.go.com/nfl/team/\_/name/bal/baltimore-ravens'

doc <- htmlTreeParse(url,useInternal=TRUE)

teams <- xpathSApply(doc,"//li[@class='team-name']",xmlValue)

**XML扩充内容**

[1] 课程里的资料<http://www.stat.berkeley.edu/~statcur/Workshop2/Presentations/XML.pdf>

[2]另外上传了一份整理的XML的学习文档

## JSON文件

文件格式示例如下，很规整



## data.table 包

#源于data.frame,速度更快，更有效

install.packages("data.table")

library(data.table)

# 数据创建和data.frame差不多， 子集提取不太一样

tables()

DF = data.frame(x=rnorm(9),y=rep(c("a","b","c"),each=3),z=rnorm(9))

DT = data.table(x=rnorm(9),y=rep(c("a","b","c"),each=3),z=rnorm(9))

DF[2]

DF[c(1,3),]

DT[2]

DT[,c(1,3)]

#sumarize data

DT[,list(mean(x),sum(z))]

DT[,table(y)]

# 增加新列

DT[,w:=z^2]

# count .N

#keys JOIN

DT1=data.table(x=c('a','b','b','dt1'),y=1:4)

DT2=data.table(x=c('a','b','dt2'),y=5:7)

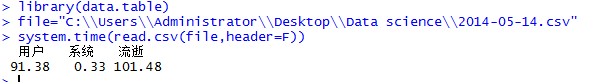
setkey(DT1,x)

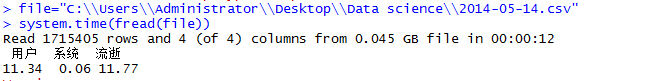
setkey(DT2,x)

merge(DT1,DT2)

#fread 实验读入速度 45m 170万行

Fread与read.csv的对比如下;





# Week 2学习笔记

## 1. R-Mysl

课程里面介绍的是安装RMySQL的package，若是在windows下，安装配置比较麻烦，还有一种方法是用RODBC包，并且这种方法也可以连接其他类型的数据库，具体可以参见：

<http://blog.163.com/zzz216@yeah/blog/static/162554684201362433011399/>

## 2.读取web网页

1. readLines

con = url("http://scholar.google.com/citations?user=HI-I6C0AAAAJ&hl=en")  
htmlCode = readLines(con)

close(con)

1. XML package

library(XML)  
url <- "http://scholar.google.com/citations?user=HI-I6C0AAAAJ&hl=en"  
html <- htmlTreeParse(url, useInternalNodes=T)  
xpathSApply(html, "//title", xmlValue)

## [1] "Jeff Leek - Google Scholar Citations"

xpathSApply(html, "//td[@id='col-citedby']", xmlValue)

## [1] "Cited by" "419" "310" "284" "183" "165"   
## [7] "156" "140" "126" "123" "50" "47"   
## [13] "40" "35" "27" "17" "14" "13"   
## [19] "13" "12" "8"

1. httr package  
   httr包可以通过以下的方式和上面的XML做同样的事情

library(httr)  
html2 = GET(url)  
content2 = content(html2,as="text")  
parsedHtml = htmlParse(content2,asText=TRUE)  
xpathSApply(parsedHtml, "//title", xmlValue)

## [1] "Jeff Leek - Google Scholar Citations"

1. 有用户名和密码的网页

library(httr)  
pg1 = GET("http://httpbin.org/basic-auth/user/passwd")  
pg1

## Response [http://httpbin.org/basic-auth/user/passwd]  
## Status: 401  
## Content-type:   
##

pg2<-GET("http://httpbin.org/basic-auth/user/passwd",authenticate("user", "passwd"))  
pg2

## Response [http://httpbin.org/basic-auth/user/passwd]  
## Status: 200  
## Content-type: application/json  
## {  
## "authenticated": true,   
## "user": "user"  
## }

使用handles

google = handle("http://google.com")  
pg1 = GET(handle=google,path="/")  
pg2 = GET(handle=google,path="search")

## 3.API

## 4.其他

sqldf package可以对R中的data frame执行SQL语句操作。

library(sqldf)

acs<-read.csv(file="getdata-data-ss06pid.csv")

ddd<-sqldf("select pwgtp1 from acs where AGEP < 50")

head(ddd)

# week 3学习笔记

## 1. subset and sort

sort

sort(x,decreasing=,na.last=)

order  
 #按照x的第一列和第三列排序 x[order(x[,1],x[,3]),] 使用plyr包中的arrange函数进行排序

library(plyr)  
arrange(x,var1)

## 2. summarize data

一般读入数据之后，先查看下数据的情况,做些简单的统计

head(data,n=)  
tail(data)  
summary(data) quantile()  
str(data)  
table() #做一些汇总的  
sum(is.na()) #查看缺失的情况  
xtabs() #列联表  
ftable()

## 3. 创建新的变量

创建二值变量

y<-ifelse(x<0,TRUE,FALSE)

创建分类变量

groups=cut(data,breaks=)

更简单的cut方式 library(Hmisc) Hmisc包

factor变量

factor()  
  
yesno<-sample(c("yes","no"),size=10,replace=T)  
yesnofac<-factor(yesno,levels=c("yes","no"))  
relevel(yesnofac,ref="yes")

## 4.reshape

library(reshape2)  
head(mtcars)  
  
mtcars$carname<-rownames(mtcars)  
carmelt<-melt(mtcars,id=c("carname","gear","cyl"),measure.vars=c("mpg","hp"))  
会将mpg与hp列变为1列  
head(carmelt)  
cyldata<-dcast(carmelt,cyl~variable,mean)

## 5.merge data

F1: merge(x,y,by,by.x,by.y,all)  
默认所有列名相同的 intersect(names(dat1),names(dat2))  
F2: plyr包的join函数，适合多个表关联

# week4 学习笔记

## 1. text variables

* tolower(),toupper() 大小写转换 经常会出现数据不一致的情况
* strsplit()  
  字符串分割，将字符串按照指定的分隔符进行分割，结果是个list
* gsub()
* grep(),grepl()
* package stringr
* str\_trim()去除空格

一般清理内容包括： 1. 所有字符统一小写 2. 最好不要有点，空格之类的 3. 变量名好理解的

## 2.正则表达式

dat<-readLines("dat.txt")  
dat

## [1] "I think it is OK"   
## [2] "you are so cool!"   
## [3] "in fact, i think this is a good question!"

* ^表示开始
* $表示结尾

r grep("^i",dat)

## [1] 3

r grep("!$",dat)

## [1] 2 3

* [0-9][a-zA-Z]
* . 表示任意字符
* 或者or good|well|better
* ^[Gg]ood|[Bb]ad 以good开头，或者含有bad  
  ^([Gg]ood|[Bb]ad) 以good或者bad开头
* ()? 括号里面是否匹配是可选的
* \*表示任意个，+表示至少一个
* {}可以设置匹配的最小和最大长度
* look for repetition of a particular phrase. +([a-zA-Z]+) + + 即寻找类似  
   so so blah blash

## 3.日期型数据

1. 查看当前的日期：  
   date() 返回的是一个字符串 d= Sys.Date() 返回的则是Date类型的变量
2. 格式转换 %d 日0-31 %a %A 星期几 %m 月份 %Y %y 年
3. as.Date() weekdays() months()
4. 除了基本的操作函数外，还有一个专门的包  
   处理日期

library(lubridate)  
ymd("20140723")

## [1] "2014-07-23 UTC"

dmy("03-4-1999")

## [1] "1999-04-03 UTC"

处理时间  
ymd\_hms() wday()

## 4. data resources

这个ppt里面给了好多的数据资源，许多数据都挺好的