Rashika Singh
IST 687

Lecture Notes- 5

Connecting to External Sources:

R data import/export which are a range of methods for obtaining data from a wide variety of programs and formats

Two threads:

Data in a discrete "flat" file format

Data in non-discrete format, i.e., "system" oriented such as a relational database

R-supplied data sets are discrete files

data() #loads specific data set

help() #gives information about the data set

Connecting R to external data sources: discrete files

Utilize RStudio to import Dataset option using Tab delimited, Comma delimited, Decimal

Example- Using RStudio import function on comma- and tab-delimited data sets.

Connecting R to Discrete files.

Used for reading discrete spreadsheet like data. R Packages include:

RODBC (Windows)

xlsReadWrite (Windows)

xlsx (Mac)

XLConnect (Mac)

gdata

Function in gdata- read.xls function

Is("package:gdata") #list contents of gdata package

Code:

Install.packages("gdata") #get package

Library("gdata")

testFrame<-read.xls ("http___") #read data

View(testFrame) #view results of read.xls

Str(testFrame) #get i

Cleansing:

- Remove header rows
- Remove unneeded columns
- Remove last few rows
- Copy first column to a column with a good name
- Remove first column

Code:

```
testFrame<-testFrame[-1:-3,]  #remove 1st 3 rows
testFrame<-testFrame[.1:5] #keep 1st 5 columns
tail(testFrame,5) #bottom 5 rows
testFrame<-testFrame[-58:-62,]
testFrame #view testFrame post Cleansing
```

Transformation:

- Remove dots on front of state names
- Convert "factor"/ character data to numeric via a custom developed function…Numberize
- Recommend viewing "testFrame" at various cleansing and transformation steps tp see effect of R statement.

Code:

```
testFrame$region<-testFrame[,1] #give 1st Column a name..region
testFrame<-testFrame[,-1]
testFrame$region<-str_replace(testFrame$region,\\.,"") #remove dots in front of state name

# Numberize() - Gets rid of commas and other junk and
# converts to numbers
# Assumes that the inputVector is a list of data that
# can be treated as character strings
Numberize <- function(inputVector)
{
```

```
inputVector<-gsub(",","", inputVector) # Get rid of commas
inputVector<-gsub(" ","", inputVector) # Get rid of spaces
return(as.numeric(inputVector))
#Apply Numberize function to columns in testFrame and give columns a new name
testFrame$april10census<-Numerize(testFrame$X)
```

Data is hard to curate in JSON, XML. Examples of it include data of hospitals which has doctors, patients, etc.

For Non-discrete data access:

Database connectivity packages

- RMySQL
- ROracle
- RPostgresSQL
- RSQlite
- RMongo
- RODBC

Code:

MySQL R code

```
#establish R connection to GKMySQL
Conmysql<-odbcConnect("GKMySQL")
#assign SQL table list
Tblsmysql<-sqlTables(conmysql)
#View Northwind tables
Tblsmysql
#assign SQL query script to datamysql
Datamysql<-sqlQuery(conmysql, paste("select * from Products"))
```

SQLDF R Package

```
Install.packages("sqldf")
```

```
Library("sqldf"
Sqldf('select mtcars.mpg from mtcars')
Sqldf('select AVG(mtcars.mpg) from mtcars where cyl=4')
AVG(mtcars.mpg)
sapply(Variable, Function, optional paramters)
#get mean for each column in mtcars
Sapply(mtcars,mean)
```

Tapply:

Tapply-similar to sapply but more granular where you group variables.

```
Tapply(Summary Variable, Group Variable, Function)
#get the mean MPG for each CYL
tapply(mtcars$mpg, mtcars$cyl, mean)

tapply(mtcars$mpg, mtcars$cyl, meanPlusSD)
meanPlusSD<-function(v) {
t<-mean(v)+sd(v)
retun(t)
}
```

Remote applications are database "servers". Data is too large to store in local memory and local Disk and we cannot make copies of large "system" databases. So we always analyze on current "official" source content vs copy.

JSON:

JSON is to make data available to other developers' part of open data movement, JSON is more flexible doesn't support hierarchical data. JSON is machine readable, easy to parse, web-based tools available