**Coursera Data Science Course 3**

**1. Raw and Processed Data**

**Data**- values of qualitative/quantitative variables, belong to a set of items

**Raw Data**- original source of data, hard to use, data analysis includes processing  
**Processed Data**- data ready for analysis, processing includes merging, subsetting, transforming, may have standards and **all steps are recorded**

**Example for Processing:** Small chunks of a long DNA sequence are binded to the slide. Multiple copies of the same sequence are made. Complementary base of each letter in the sequence is attached to the slide one at a time and are color coded. Each time a base binds, we can see the different color images it creates. Follow from image to image and see which color is the brightest to determine the sequence.

**2. Turning Raw Data into Tidy Data**

**Need:** (1) Raw data, (2) Tidy data set, (3) Code book describing each variable and its values in the tidy data set, (4) Explicit and exact recipe you used to go from (1) -> (2) + (3)

**What is Raw Data?**-Strange binary file from measurement machine  
-Unformatted Excel file with 10 worksheets  
-JSON data from twitter API  
-Numbers entered from scientific experiments

**Raw data is only raw if we did not edit it, therefore:**-no software allowed  
-no number manipulation  
-no removal of any data  
-no summarization of data

**What is Tidy Data?**  
-Each variable measured per column  
-Each observation of that variable per row   
-A table for each “kind” of variable (Ex: Twitter vs Facebook)  
-Multiple tables need column in the table that links them

**Should Do for Tidy Data:**  
-Row at the top of each file with variable names  
-Variable names are readable such as “AgeAtDiagnosis”  
-Data saved in one file per table

**What is a code book?**The code book should contain some extra information about the data-Info about variables in the data set that are not in the tidy data (maybe UNITS)  
-Info about the summary choices made (Median vs Mean etc)  
-Info about experimental study designed used

**Should Do for Code Book:**  
-Use Word/.md/.txt  
-“Study design” section  
-“Code book” that describes each variable and its units

**Instruction List:**-Use R (or maybe Python)  
-Input is raw data and output is processed, tidy data  
-NO PARAMETERS

Note that IF we can’t script every step, provide instructions on what to do!

Instruction list is so important!

**3. How to Download a File with R**

**Where are we?**

**getwd()** – gets working directory  
**setwd()** – sets a different wd  
Relative – setwd(“../”) goes one flight up  
Absolute – just type the entire path  
Window uses “\\” instead of “/”

**How to create or check directories?**

**file.exists(“directoryName”)** – checks existence  
**dir.create(“directoryName”)** – create directory

You can do something like:

If (!file.exists(“data”)) {  
 dir.create(“data”)  
}

**To Download a File:** *url, destfile, method* are important parameters to note

**Example:** Baltimore Traffic Cameras – right click and click on “Copy Link Address”, then save link into

**fileUrl <- linkName**  
**download.file(fileUrl, destfile = “./data/cameras.csv”, method = “curl”)** – note that curl is important to specify for https (secure) websites, especially on Macs  
**list.files(“./data”)** – can check if the data file has been made

Then we can date it like so: dateDownloaded <- date()

-Definitely can use download.file() with http  
-Maybe can use it with https on Windows  
-If using https on Mac, set method=”curl”

**4. How to read Downloaded/Local Files**

**Read.table()***file, header, sep, row.names, nrows*

**Back to the Camera Data**cameraData <- read.table(“./data/cameras.csv”, sep = “,”, header = TRUE)  
head(cameraData)

Also recall if you use read.csv, it sets sep=”,” and header = TRUE automatically

**Other Important Parameters Include:**  
quote – are there any quoted values? Use quote = “” for none or to get rid of reading accidentaa;l ‘ or ‘ in the data  
na.strings – character represents missing value  
nrows – however many rows do we want to read in a file?  
skip – number of lines to skip before we start to read the file

**5. How to Read Excel Files**

The only thing different for getting the file from the internet is use destfile=”./data/cameras.xlsx” instead of .csv

library(xlsx)  
cameraData <- read.xlsx(“./data/cameras.xlsx”, sheetIndex=1, header=TRUE)

We can also read specific rows and columns  
colIndex <- some number:some other number and rowIndex as well

Misc  
-write.xlsx can write out an excel file  
-readl.xlsx2 faster but more unstable  
XLConnect is a good package to check out

**6. How to read XML**

**What is XML?**-extensive markup language  
-frequently used to store structured data  
-a lot of internet applications  
-two parts  
- markup – labels that give the text structure  
- content – the actual text of the document

**Tags, elements, and attributes**-start tags: <section>  
-end tags: </section>  
-empty tags: <line-break />

Example: <Greeting> Hello, world </Greeting>

Attributes are components of the label  
Example: <img src= “jeff.jpg” alt=”instructor”/>  
<step number=”3”> Connect A to B. </step>

XML files have A LOT of tags

**Read XML files into R**

**Library(XML)  
fileUrl <- someUrl  
doc <- xmlTreeParse(fileUrl, useInternal=TRUE)** // loads R into a structured object  
**rootNode <- xmlRoot(doc)** //you can get the “wrapper” of the document. If you execute this command, you’ll have access to a particular element to that xml file.  
**xmlName(rootNode)** // tells you the name of the root node or the “wrapper” itself  
**names(rootNode)** // tells you the names of the all the nested elements within the root node  
  
**rootNode[[1]]** // gives you all the information about the first element of the list (Belgian waffles)  
**rootNode[[1]][[1]]** // only gives you the first line of the first element of the list

**xmlSApply(rootNode,xmlValue)** // allows you to extract different parts of the file passing a parsed object (such as the XML value) and it’s going to go through and get every single value of every sing tagged element

**XPath – How deep into the node are we at? What are we calling?**/node – top level node  
//node – node at any level  
node[@attr-name] – node with an attribute name  
node[@attr-name=’bob’] – node with the attribute name ‘bob’

**xpathSApply(rootNode,”//name”,xmlValue)** // will get you the name of each node **xpathSApply(rootNode,”//price”,xmlValue)** // will get you the prices of each node

**Ex:** Baltimore Raven source code  
Use htmlTreeParse to parse an html file  
**scores <- xpathSApply(doc,”li[@class=’score’]”, xmlValue)** // note that li stands for list items. And this will extract all the scores from the website and list them in an object

There’s extra links included at the end of the XML slides

**7. How to Read JSON**

**What is JSON?**  
-stands for Javascript Object Notation  
-lightweight data storage  
-common format for APIs  
-data can be doubles, strings, Boolean, array, objects

**Library(jsonlite)  
jsonData <- fromJSON(link)  
names(jsonData)  
names(jsonData$owner)** // gives all the owner name  
  
you can also write data frames to JSON  
**myjson <- toJSON(iris, pretty=TRUE)  
cat(myjson)**Convert back from JSON

**Iris2 <- fromJSON(myjson)  
head(iris2)**

**8. The data.table Package**

**Library(data.table)  
DF = data.frame(xrnorm(9), y=rep(c(“a”,”b”,”c”) each = 3), z = rnorm(9)))** // creates a table in the data frame format, with columns x, y, and z – each with 3 rows of variables.  
**head(DF,3)**

**DT = data.table(x=rnorm … same as DF)  
head(DT,3)**

**Tables()** // see all the data tables in memory

**DT[2, ]** // subset the second row of data **DT[DT$y=”a”,]** // subset all data in which y = a  
**DT[,c(2:3)]**  // DOES NOT WORK when you try to subset columns this way!!!!

Instead, use this…:  
Every comma separation represents a different expression  
So you can do something like this:  
**DT[,list(mean(x),sum(z))]** // will print out two numbers. One is the mean of x values and the other is the sum of z values.  
**DT[,table(y)]** // will print out a b c, 3 3 3  
  
**Adding a new column is also very easy:  
DT[,w:=z^2]** // adds a new column w where the value of w = z^2

You can also do **multiple operations**:  
**DT = [,m:= {tmp <- (x+z); log2(tmp+5)}]** // will create a new column m spitting out whatever you expected the answer to be

You can also do some **conditions** with it:  
**DT[,b:= mean(x+w), by=a]** // will create two different number variables depending on if the value of a is true or false

**Special Variables:**

.N – an integer, length 1, containing the number

**Set.seed(123)  
DT <- data.table(x=sample(letters[1:3], 1E5, TRUE))  
DT <- [, .N, by=x]**

Letters contains a lot of “a” “b” and “c”s in it and this function counts the amount of times each letter appears

This will return  
x N  
a 33387  
c 33201  
b 33412

**Keys  
DT <- data.table(x=rep(c(“a”,”b”,”c”), each = 100), yrnorm(300))  
setkey(DT, x)  
DT[‘a’]**

This will only look at all the variables in the column x where x = ‘a’ while giving column y actual numbers

**Joins  
DT1 <- data.table(x=c(‘a’, ‘a’, ‘b’, ‘dt1’), y=1:4)  
DT2 <- data.table(x=c(‘a’, ‘b’, ‘dt2’), z=5:7)  
setkey(DT1, x); setkey(DT2, x)  
merge(DT1, DT2)**

**Fast reading  
big\_df <- data.frame(x=rnorm(1E6), y=rnorm(1E6))** // this will create a gigantic data frame  
**file <- tempfile()  
write.table(big\_df, file=file, row.names=FALSE, col.names=TRUE, sep=”\t”, quote=FALSE)  
system.time(fread(file))**

write your data frame into a file instead of just read.table