Getting and cleaning Data.

Downloading Files using R

know what directory you are working on

getwd() - get what you are in
setwd() - set wd
setwd(../ - go up a directory . windows go \\

dir.create(“directory name)
file.exists() - check first then create.

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

```
fileUrl <- “”
download.file(fileUrl, destfile = “ “ , method= “curl” )
dateDownalod <-date()  #save date for future reference.
```

Download.file(url, destfile, method) , include download fine
, from mac set method to curl . if https
list.files(“/data”)

read.table() - robust and flexible to read the data. , reads data into RAM , if big , read in chunks, file, header, sep, row.names., nrows. , quote, tell are what the values in does , na.strings = set character for representing missing values.
norms = .. .
skip == . .

set quote = “” , set this to not be confused with quotation marks .

read.table(‘’, sep =“,” , header = TRUE)

Reading excel files :
 read.xlsx()   or read.xsls2() (“”, sheetIndex = 1, header =TRUE)
read specific rows and columns using colIndex, rowIndex in read.xlsx()
write.xlsx() ,
read.xlx2 is faster.
XLConnect package is much better. check XLconnect vignette.

reading XML -
extensive markup language
Componenets - markups and content - will be between labels .

There will be start tags, end tags, empty tags,
Reading JSON,

```
library(XML )
fileURL
doc <- xmlTreeParse(fileURl, useInternal = TRUE)
rootNode <- xmlRoot(doc)
xmlName(rootNode)
rootNode[[1]][[1]] , - get individual items off each
xmlSApply(rootNode, xmlValue) , get all values with all the tags, use this to apply functions
xpathSAppley(rootNode, “//name?”, xmlValue)
```

“//listitensm = “score: ### relook at presentation slides, different language to extract markup from different ones. learn to scrape information directly using R. look up XML package tutorial , to programmatically extract information from websites.

READING JSON:
Javascript Objects Notation

- Lightweight , similar structure to XML , data scored as numbers , strong, all this and that. Used in GitHub to represent repos.
- Library(jsonlite)
- jsonData <- fromJSON(“url”)
- names(jsonData$owner)

myjson <- toJSON(iris, pretty = TRUE) , store to son in a nicely structured format.

data.table package = inherits from data.table. much faster at subsetting, grouping and updating is better.

```
library(data.table)
DF = data.frame(x)
DT = data.table.   #same way you create a data frame.
tables()  — tables instead of table
#same subsetting rules, except if subsetting done with only a single index, does something different,. uses expressions to summarize the data at a different way .
#add new column,
#select data, apply expressions,
```

#perform multiple step functions … %%%CHECK prez notes again%%%
set keys to sort the tables much faster. easier to facilitate joins as well .

READING DATA FROM mySQL:
each table is a dataset, each row is a structure.. Look at MYSQL structure for different tables. .. look u[columns and joins.

ucsDB <- dbConnect(MySQL, user=“” , host= “”) — also database, to see what tables.
 result <- dbGetQuery(usb, “show databases;”) , dbDisconnect.. show show True.
allTables <- dbListTables(hg19)
length(allTables) ..
dfListFeilds(hg19, “”) — get all the fields,
dbListFeilds(hg19, “select count(X) from affyU133Plus2)
head()
dbReadTable(hg19, “”) , read the table and extract one table at a time.

query <- dbSendQuery(hg19, “select * from iffy where mismatch is between)
affyMis <- fetch(query, n =10); quantile(affyMiss$msMAtchches) — use to get a small part of a large data
dbClearResult(query) — return to clear query
dbDisconnect(hg19) close the connection as soon as you don’t need it

Reading Data from HDf5 . — make notes
Reading Data from APIS . — make notes
Reading Data from Web
conn = url ()
htmlCode = readLines(con)
close(con)
htm code
html <- htmlTreeParse(url, useInternalNodes = T)
xpathSapply(html, “//title” , smlValue)
library(httr) — check again , use to GET package, response
use handles, paths and handles.

WEEK 3
reshaping data - recast function
tapply,
split, apply combine, — use split ,apply, then unlist and sapply()

Merging data - Merge() , by.x , by.y, , join().

arrange(join(df1, df2) , id) . arrange in increasing order by ID.

Week 4 :
editing text variables - tolower()
strsplit(names(cameraData) , “\\”)
splitNames[[5]]
sub(“_” , “’”, names(reviews)) - substitutes the underscores
gsub(“_” , “” , test name) - sub removes only first, sub removes all) .
grep () - find all the indexes where the thing appears
grepl() - return true or false
 subset using camerData[!grepl()]

Regular expression matching:
^i think - start of the line
morning$ - end of the line
[Bb][Uu][Ss][Hh] - all versions of capital or lower of the word
^[0-9][a-zA-Z] range of characters
[^?.]$, - anything that does not end with ? or period
9.11 - . means any character 9!11 , 9-11
fire|flood - Or expression
^([Gg]ood|[Bb]ad) - mix with parenthesis to get logic right
([Ww]\.)? - question mark means optional . \ means view it as a meta character and not a regular expression operator . “*” - repeat any number of times. (.*) - any character any number of times between parenthesis .
“+” - atleast once .
{1,5} - see something between one and 5 times
[^] - something that is not a space
\1 \2 - matching characters - repetition of a character “
“*” is greedy so it will look for the max length of the string .

DAPPLYR.
select(), filter(),
| arrange(), mutate(), and summarize().
Use select(cran, r_arch:country) to select all columns starting from r_arch and ending with country.

after group_by (cran, xzy), set default group to run operations on that default value.
pack_sum <- summarize(by_package,

```
                      count = n() ,
                      unique = n_distinct(ip_id),
                      countries = n_distinct(country) ,
                      avg_bytes =  mean(size))


quantile(pack_sum$count, probs = 0.99)

filter(pack_sum, count > 679)


# CHAINING — IMP
by_package <- group_by(cran, package)
pack_sum <- summarize(by_package,
                      count = n(),
                      unique = n_distinct(ip_id),
                      countries = n_distinct(country),
                      avg_bytes = mean(size))

# Here's the new bit, but using the same approach we've
# been using this whole time.

top_countries <- filter(pack_sum, countries > 60)
result1 <- arrange(top_countries, desc(countries), avg_bytes)

# Print the results to the console.
print(result1)


result2 <-
  arrange(
    filter(
      summarize(
        group_by(cran,
                 package
        ),
        count = n(),
        unique = n_distinct(ip_id),
        countries = n_distinct(country),
        avg_bytes = mean(size)
      ),
      countries > 60
    ),
    desc(countries),
    avg_bytes
  )

print(result2)

result3 <-
  cran %>%
  group_by(package) %>%
  summarize(count = n(),
            unique = n_distinct(ip_id),
            countries = n_distinct(country),
            avg_bytes = mean(size)
  ) %>%
  filter(countries > 60) %>%
  arrange(desc(countries), avg_bytes)

# Print result to console
print(result3)

cran %>%
  select(ip_id, country, package, size) %>%
  mutate(size_mb = size / 2^20) %>%
  filter(size_mb <= 0.5) %>%
  arrange(desc(size_mb))  %>%
  print
```

Gather takes multiple columns and collapses into key-value pairs, duplicating all other columns as needed. You use gather() when you notice that you have columns that are not variables.

-grade. Note the minus sign before grade, which says we
| want to gather all columns EXCEPT grade.

```
students2 %>%
  gather( sex_class, count, -grade ) %>%
  separate( col= sex_class, into =c("sex", "class")) %>%
  print
```

SPREAD function - best in use for turning values of columns, into separate columns,

```
sat %>%
  select(-contains("total")) %>%
  gather(part_sex, count, -score_range) %>%
  separate(part_sex, c("part", "sex")) %>%
  group_by(part,sex)
  mutate(total = sum(count),prop = count / total) %>%
    print
```