**Coursera Data Science Course 3 – Week 3**

**1. Subsetting and Sorting**

**Recall how to subset data for a data frame  
set.seed(13435)  
X <- data.frame(“var1”=sample(1:5),”var2”=sample(6:10),”var3”=sample(11:15))** // creates a data frame with variables, each containing 5 data points **X <- X[sample(1:5),]; X$var2[c(1,3)] = NA** //creating the three columns to format the data frame, also makes #1 and #3 in var2 to be NA **X**

**Subsetting  
X[,1]** // will print the first column **X[,”var1”]** // will print the first column **X[1:2,”var2”]** //will print the first two rows, and the second column

**Can also use logic  
X[(X$var1 <= 3 & X$var3 > 11),]** // will print all rows in which the two logical statements hold true  
- can also use OR

**Missing values  
X[which(X$var > 8), ]** // automatically ignores missing values

**Sorting  
sort(X$var1)** //puts them in increasing order (1 2 3 4 5)  
**sort(X$var1,decreasing=TRUE)** //puts them in decreasing order (5 4 3 2 1)  
**sort(X$var2,na.last=TRUE)** // puts them in increasing order and the NA last

**X[order(X$var1),]** // order the rows for increasing order, can also do multiple variables etc

**Ordering with plyr  
library(plyr)  
arrange(X,var1)** // basically the same thing as order  
**arrange(X, desc(var1))** // also works

**Add rows and columns  
X$var4 <- rnorm(5)**  //adds an extra column var4  
**Y <- cbind(X,rnorm(5))** bind the column to the right side of X.

**2. Summarizing Data**

**Example:** Baltimore Restaurant Data

**Classic getting data from web again**// insert the create directory code  
**fileUrl <- “the link”  
download.file(fileUrl, destfile=”./data/restaurants.csv”, method = “curl”)  
restData <- read.csv(“./data/restaurants.csv”)**

**head(restData, n=3)** // looks at the first three rows of the data **tail(restData, n=3)** // looks at the last three rows of the data

head and tail are by default n=6

**Some good summary methods**

**1. summary(restData)** – for all variables it will give you some summary. How many mcdonalds there are, median of certain data etc

**2. str(restData)** – gives you the number of variable there are, how many levels etc and dimensions

**3. quantile(restData$councilDistrict,na.rm=TRUE)** – splits things up into quantiles and tells us how many data points are in each percentile.

**3b. Quantile(restData$councilDistrict, probs=c(0.5,0.75,0.9))** – in this case splits the data into 50, 75, and 90%

**4. Table(restData$zipCode, useNA=”ifany”)** // will give us a table of zip codes. How many of each zip code there is. Will tell us missing values if we use the useNA part

**4b. Table(restData$councilDistrict, restData$zipCode)** // can also do 2D tables.

**Check for missing values**

**sum(is.na(restData%councilDistrict))** // returns a # **any(is.na(restData$councilDistrict))** // boolean **all(restData$zipCode > 0)** // Boolean

**Row and column sums  
colSums(is.na(restData))** // tells u the sum of NA data points there is for each column **all(colSums(is.na(restData))=0)** // same for each row

**Values with specific characteristics  
table(restData$zipCode %in% c(“21212”))** // returns all that is true and false in the statements in two separate columns, can also do c(“21212”,”21213”) and the default is OR, not AND

**restData[restData$zipCode %in% c(“21212”, “21213”,]** // prints out all data points in which the zip code is 21212 or 21213

**Cross tabs  
data(UCBAdmissions)  
DF = as.data.frame(UCBAdmissions)  
summary(DF)  
xt <- xtabs(Freq ~ Gender+ Admit, data=DF)** //variable you want to display, in our case is gender and admit. Also have to specify the data cross tabbed (in our case DF) **xt**

**Flat tables  
warpbreaks$replicate <- rep(1:9, len = 54)** // warpbreaks is the name of the data set. Adding another replicate **xt = xtabs(breaks ~.,data=warpbreaks)  
xt**

if the table is hard to read, we can **ftable(xt)** where it will put everything into the same table

**Size of data  
fakeData = rnorm(1e5)  
object.size(fakeData)** // tells you the size in bytes  
**print(object.size(fakeData), units=”Mb”)** // converts the size in Mb

**3. Creating New Variables**

-raw data doesn’t have some values we are looking for, so we will make our own  
**Common variables to create:**-missingness indicators  
-“cutting up” quantitative variables  
-applying transforms

**Example:** Baltimore restaurants (again)

Once again //code to get data from the web

**Creating sequences**

**s1 <- seq(1,10,by=2) ; s1** //create a sequence from 1 to 10 in which each term separates by every 2 number. (i.e. 1 3 5 7 9)

**s2 <- seq(1,10,length=3) ; s2** //create a sequence of 3 terms with equal spacing from 1 to 10 (1.0 , 5.5, 10.0)

**x <- c(1,3,8,25,100); seq(along = x)**

**Subsetting variables  
restData$nearMe = restData$neighborhood %in% c(“Roland Park”, “Homeland”)  
table(restData$nearMe)** // will give true or falses ☺

**restData$zipWrong = ifelse(restData$zipCode < 0, TRUE, FALSE)  
table(restData$zipWrong,restData$zipCode < 0)** // To check how many invalid zip codes there are

**Create Categorical Variables  
restData$zipGroups = cut(restData$zipCode,breaks=quantile(restData$zipCode))** // cut cuts up the data in accordance to our conditions, in our case by each quantile, then they will become factor variables. 0-25%, 25-50% etc, then can show which zip falls into which cluster **table(restData$zipGroups, restData$zipCode)** // can show which zipcode falls into which quantile! ☺

**EZ CUT  
library(Hmisc)  
restData$zipGroups = cut2(restData$zipCode,g=4)** // automatically chops it into 4 pieces **table(restData$zipGroups)**

**Create Factor Variables  
restData$zcf <- factor(restData$zipCode)   
restData$zcf[1:10]** // the zipcode becomes factors  
**What are levels?  
yesno <- sample(c(“yes”,”no”),size=10,replace=TRUE)** //size 10 vector of yes’s and no’s **yesnofac = factor(yesno, levels=c(“yes”,”no”))  
relevel(yesnofac,ref=”yes”)**

**If we cut the data, we produce factor variables  
library(Hmisc)  
restData$zipGroups = cut2(restData$zipCode,g=4)  
table(restData$zipGroups)** //create 4 factors (zip groups) on top of data

**“Mutate” Function – same as the cut  
library(Hmisc), library(plyr)  
restData2 = mutate(restData, zipGroups=cut2(zipCode,g=4))  
table(restData2$zipGroups)**

**Common Functions  
Abs(x)** – abs value **Sqrt(x)** = square root **ceiling(x)  
floor(x)  
round(x, digits=n)  
signif(x, digits=n)  
cos(x), sin(x)  
log(x)  
log2(x),log10(x)  
exp(x)**

**4. Reshaping Data**

**Recall Tidy Data**-Each variable in a column  
-Each observation in a row  
-Each table/file stores data about one kind of observation

**Reshaping  
library(reshape2)  
head(mtcars)**

**We can first melt the data frame:  
mtcars$carname <- rownames(mtcars)  
carMlet <- melt(mtcars,id=c(“carname”,”gear”,”cyl”),measure.vars=c(“mpg”,”hp”))** // tells the function which variables are ID variables and which are measurement variables, also makes the data set tall and skinny in the sense that it is divided up by mpg vs hp

**Then we can recast this data frame:  
cylData <-dcast(carMelt, cyl ~ variable)** // tells us how many of each value there are **cylData <- dcast(carMelt, cyl ~ variable,mean)** // takes the mean of each mpg/hp

**Averaging value:  
tapply(InsectSprays$count, InsectSprays$spray, sum)** // for each value of spray, we will sum up the counts. Will show how many of each type of spray there is in total

OR  
**spIns = split(InsectSprays$count, InsectSprays$spray)** // gives you the list of the values of each type of spray

THEN  
**sprCount= lapply(spIns,sum)** //adds the number of data in the spIns list up  
**unlist(sprCount)** // turns it back into a vector for easier calculations  
**sapply(spIns, sum)** // gives you sup of values as well

We can also do:  
**ddply(InsectSprays,.(spray),summarize,sum=sum(count))**-allows for creating a variable, you can set this to spraySums:  
**spraySums <- ddply(insectSprays,.(spray),summarize,sum=ave(count,FUN=sum))**Then you can take dim and head and etc to it

Also check out:  
**acast** – for casting multi-dimensional arrays  
**arrange** – for faster reordering without using order() commands  
**mutate** – for adding new variables

**5. Managing Data Frames with dplyr- Introduction**

**Dplyr**-key data structure in statistics and R  
-one observation per row  
-each column represents a variable/measure/characteristic  
-primary implementation = default R implementation  
-other implementations, particularly relational databases systems  
-optimized version of plyr  
-no “new” functionality, but simplifies functionality in R  
-provides “grammar” for data manipulation  
-very fast b/c it’s C++

**Dplyr Verbs 101  
select** – return a subset of columns  
**filter** – extract a subset of rows using logical conditions  
**arrange –** reorder rows  
**rename** – rename variables  
**mutate** – add new or transform existing variables  
**summarize/summarise** – generate summary statistics of different variables

**Dplyr properties**=first argument is a data frame  
-subsequent argument is what to do with it, refer to columns directly without $ operator.   
-result is a new data frame  
-formatting is really important

**6. Managing Data Frames with dplyr- Basic Tools**

**Example:** Air Pollution in Chicago  
**Chicago <- readRDS(“Chicago.rds”)**

**head(select(Chicago, city:dptp))** – easy to look at subset of columns just by naming their names, don’t have to find the indices of “city” and “dptp”

**chic.f <- filter(Chicago, pm25tmean2 > 30)** // can also do multiple logical statements **head(chic.f, 10)** // gives you 10 data points where pm25tmean2 > 30

**Chicago <- arrange(Chicago, date)** // arranges the data by date  
**Chicago <- arrange(Chicago, desc(date))**  // arranges the data by descending date

If I want to simply the name of the variables:  
**Chicago <- rename(Chicago, pm25 = pm25tmean2, dewpoint = dptp)**

**Chicago <- mutate(Chicago, pm25detrend = pm25 – mean(pm25, na.rm = TRUE))** // compares the data to its mean

**Chicago <- mutate(Chicago, tempcat = factor(1 \* (tmpd > 80), labels = c(“cold”, “hot”)) + )** // if its lower than 80 its cold, otherwise its hot **hotcold <- group\_by(Chicago, tempcat)  
hotcold**

**Summarize(hotcold, pm25 = mean(pm25, na.rm = TRUE), o3 = max(o3tmean2), no2 = median(no2tmean2))** // tells us some numbers about each hot and cold days including the pm 25 mean, the max o3 level, and the median no2

**Chicago <- mutate(Chicago, year = asPOSIXlt(date)$year + 1900)  
years <- group\_by(Chicago, year)  
summarize(year,s blah blah blah)** //summary for each of the variables by year

**Chicago %>% mutate(month = as.POSIXlt(date)$mon + 1) %>% summarize(blah blah blah)** // will show all the stats for each month

Dplyr can also:  
-work with other data frame “backends”  
-large tables as well  
-SQL interface

**7. Merging Data**

**Example**: Peer review data

//insert code to download and read.csv some data files

**Merge()  
Important Parameters:** x, y, by, by.x, by.y, all

**MergedData = merge(reviews,solutions,by.x=:solution\_id”,by.y=”id”, all = TRUE)** // will merge on solution ID. Solution ID will take the place of the ID. And all = TRUE leaves all the data that aren’t merged alone

**Default merges all common column names**  
**intersect(names(solutions),names(reviews))** // first check for all that intersects between the two data set  
**mergedData2 = merge(reviews,solutions, all=TRUE)**

**Join from plyr also works, faster but less full featured  
df1 = data.frame(id=sample(1:10), x=rnorm(10))  
df2 = data.frame(id=sample(1:10), y=rnorm(10))  
arrange(join(df1,df2), id)** // only converge on the condition of common name, such as id. If I want x and y on each id, that’s how I join them.

Join multiple sets: **dfList = list(df1, df2, df3)  
join\_all(dfList)**