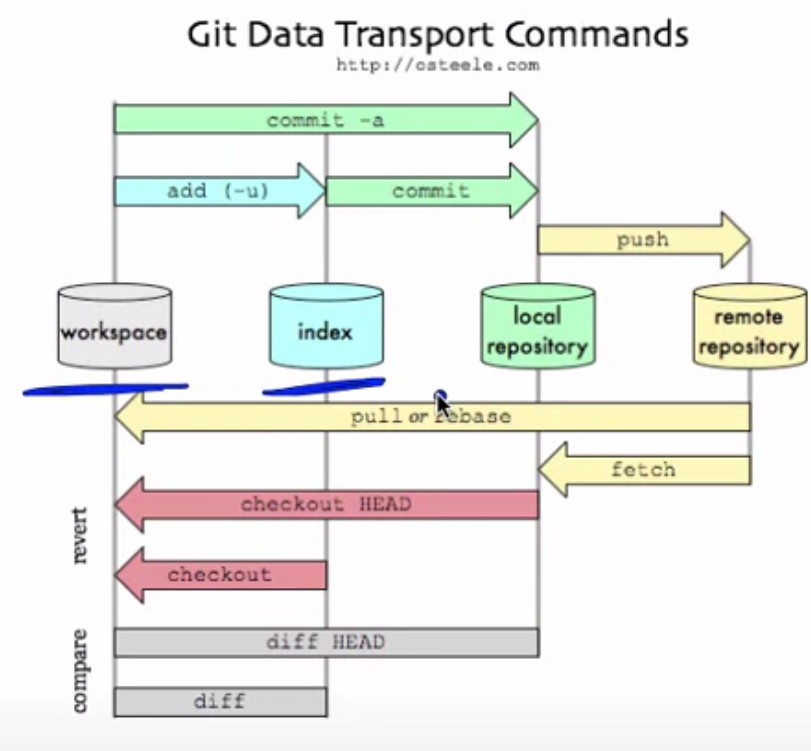
**COMMAND LINE: command flags arguments**

**pwd; clear; ls; cd; mkdir; touch; cp; rm; mv;**

**GIT: Peter-32; common password; peter@impactradius.com**

**git init [may only need first time]**

**git config --global user.name “Your name”**

**git config --global user.email “Your email”**

**git config --list**

**git config --global credential.helper wincred**

**git clone {url}**

**git add -A**

**git commit -m “Your message”**

**git push**

**OTHER GIT COMMANDS:**

**git remote -v**

**git remote add upstream**

**git fetch upstream**

**git checkout -b “Your branchname” [switch branches]**

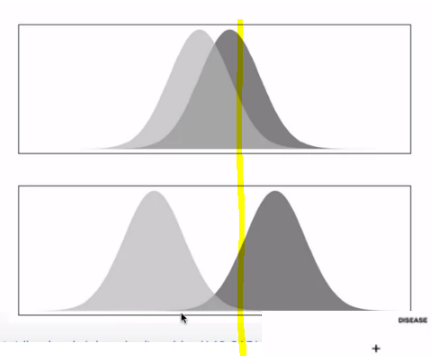
**git checkout master [switch to master branch]**

**git merge upstream/master**

**MARKDOWN FILES:**

**HEADINGS: #{primary heading} ##{secondary} ###{tertiary}**

**LISTS: \* [this symbol is a bullet]**

**DATA SCIENCE QUESTIONS: [in order of difficulty of answering]**

**describe; explore; infer; predict; cause-effect; mechanistic**

**YOUR EXPERIMENTS: Have a purpose; correlation is not causation;**

**Fix a variable OR equally stratify it OR randomize it; [Graph: how to predict]**

**avoid data dredging; have replication; measure variance; generalize;**

**prediction isn't inference**

**BASIC FUNCTIONS: getwd(); source(“file.r”); ls(); dir();**

**1:20 [sequence];c() [create vector]; list(); matrix(nrow=,ncol=);**

**factor(c(),levels=c()); is.na(x); data.frame(col1,col2)**

**paste(); args(func); unclass; rowSums; rowMeans; colSums; colMeans; quantile(probs=c()); gl();**

**mean(x, na.rm=FALSE); median(x); str() [what's in this object?];**

**CONTROL FUNCTIONS:**

**if(boolean){} else if {} else {}; for(i in 1:10) {}; for (i in seq\_along(x)) {}; for (letter in x) {};**

**while(boolean && boolean) {}; repeat {} [infinite]; next [skips 1 iteration]; return [leave function]**

**DISTRIBUTION FUNCTIONS:**

**rnorm; dnorm; pnorm; qnorm; can replace norm with pois or binom; set.seed(n); summary(x);**

**d: density; r: random gen; p: cum distr; q: quantile; sample(1:10, 4, replace = FALSE)**

**SAVING DATA: dump(); source() to get it back;**

**DATA TYPES: character; numeric; integer; complex; logical; vector; list; matrix; [class()]**

**DATA ATTRIBUTES: names; dims; class; length; [attributes()]**

**READING DATA USING READ.TABLE: arguments: file;header=FALSE;sep=;colClasses=c();nrows;comment.char;skip;stringAsFactors=TRUE;dec**

**read.csv has comma sep; read.csv2 has semicolon sep; read.delim has \t sep; all have header=true**

**READING DATA EFFICIENTLY:** read 100 rows; sapply get classes; read all rows;

**STORING/GETTING R OBJECTS: dput(y,file=”filename.R”); dget(“filename.R”)**

**CONNECTIONS: con <- file(“filename”, “r” “w” or “a”) [read write or append]**

**read.csv(con); close(con); readLines(con,10); close(con); con <- url(“www.abc.edu”, “r”);**

**SUBSETTING: [; [[; $; [1:4]; [x > “a”]; Matrix/Data Frame: x[1,2] x[,1], x[1,]**

**REMOVING NA: bad <- is.na(x); x[!bad]; good <- complete.cases(x,y)**

**MATRIX MULTIPLICATION: x %\*% y ;**

**FUNCTION SYNTAX: funName <- function(input1=defaultValue) {}**

**LOOKING IN A FUNCTION: ls(environment(functionName));**

**get(“var1”, environment(functionName));**

**DATES: as.Date(“1970-01-01”); class is POSIXct, POSIXlt; functions:weekdays, month, quarters**

**as.POSIXlt(Sys.time(), tz = “GMT”); sec, min, hour, mday, mon, year, wday, yday, isdst**

**strptime(Yourdatestring, “%B %d, %Y %H:%M“); use ?strptime for formatting**

**lapply(x, FUN, ...); sapply will simplify lists into vectors or matrices based on list length;**

**apply(x, dimNumbersToKeep [ie. c(1,3) or 1], FUN, ...)**

**mapply(); list(rep(1, 4), rep(2, 3)) is equivalent to mapply(rep, 1:2, 4:3)**

**tapply(x, YourListOfFactors, FUN, ...); f <- gl(3, 10); tapply(x, f, mean)**

**split(x, f, drop=FALSE [drops empty factors])**

**See notes for debugging tools.**

**See notes for optimizing script runtime with R Profiler.**

**READING DATA INTO R.**

**DOWNLOAD FILES: if (!file.exists(“data”)) { dir.create(“data”) }**

**fileUrl <- “www.abc/rows.csv”; download.file(fileUrl, destfile = “./data/abc.csv”, method = “curl”) [use curl only on macs]; list.files(“./data”);**

**READING EXCEL FILES: library(xlsx2); read.xlsx(“./directory”, sheetIndex=1, colIndex=c(), rowIndex=c(), header=TRUE); write.xlsx is similar;**

**READING XML: library(XML); fileUrl <- “url”; doc <- xmlTreeParse(fileUrl, useInternal=t)**

**rootNode <- xmlRoot(doc); xmlName(rootNode) [outer]; names(rootNode) [names of next level];**

**rootNode[[1]]; rootNode[[1]][[1]]**

**EXTRACT PARTS OF XML: /node [top level] //node [any level]**

**xpathSApply(rootNode, “//name”,xmlValue);**

**Pull all HTML: con = url(“urlHere”); htmlCode = readLines(con); close(con);**

**EXTRACT PARTS OF HTML: doc <- htmlTreeParse(fileUrl, useInternal=TRUE)**

**scores <- xpathSApply(doc, “//li[@class='score']”,xmlValue)**

**names <- xpathSApply(doc,"//div[@class='win-summary summary-side']//span[@class='person-name']",xmlValue) [this will look for the class, then look for the second class within it]**

**READING JSON: library(jsonlite); jsonData <- fromJSON(“JSONWebLink” or file);**

**READING MySQL: install mysql and rmysql on the internet;**

**DB <- dbConnect(MySQL(),user=”genome”, db=”dbname”, host=”genome-mysql.cse.ucsc.edu”)**

**result <- dbGetQuery(DB, “Put SQL HERE;”); dbDisconnect(DB); dbListTables(DB);**

**dbListFields(DB, “tableName”); dbReadTable(DB, “tableName”);**

**HDF5: library(rhdf5); created <- h5createFile(“example.h5”); See notes for more details.**

**READING API DATA: create an application;**

myapp = oauth\_app("twitter",

key="MfkhMjyH3iiR9uZNrsLXi4KZl",

secret="ymSQNUaHHLnTW238ttUScb7rcSKk8CF35fKrbA1yXRYRBp3hXB")

sig = sign\_oauth1.0(myapp,

token = "4449167113-7kIXZ0lYEg36crMRl3WjzJe31mUuEfbkRnpyaE5",

token\_secret = "MgIhoMRQE6Rh17vuD4n7pwCbS1btTYdwtdSGC9lCRkz3C")

**homeTL = GET("https://api.twitter.com/1.1/search/tweets.json", sig)**

json1 = content(homeTL) ## content will recognize that it is JSON data and it will convert it to an object

json2 = jsonlite::fromJSON(toJSON(json1)) ## this reformats it to a JSON file that is a data frame

**READING OTHER SOURCES: file; url; gzfile; bzfile; close connections afterwards**

**library(foreign); read.arff (Weka); read.dta (Stata); read.mtp (Minitab); read.octave (Octave);**

**read.spss (SPSS); read.xport (SAS);**

**library(jpeg); library(readbitmap); library(png); library(EBImage); [Images]**

**library(rdgal); library(rgeos); library(raster); [Geo Data]**

**library(tuneR); library(seewave); [Music Data]**

**RESHAPING DATA: library(reshape2);**

**data$newColumn <- rownames(data) [adds a new column using the row names]**

**dataMelt <- melt(data, id=c("col1","col2","col3"), measure.vars=c("mpg","hp"))**

**[all the values from column mpg and hp go into one column, a new categorical column is created which has one of the two values mpg or hp]**

**cylData <- dcast(carMelt, cyl ~ variable) [this is the opposite of melt]**

**MANIPULATE DATA:**

**USING SQL ON DF: library(sqldf); DF=sqldf('Query goes here'); [Thats all there is to sqldf]**

**X$var4 <- rnorm(5) [this can add a column]**

**colSums(is.na(restData)) [to see the # of NA values for each column]**

**table(data$col1,data$col2) [2 dimensional data counting occurrences]**

**xtabs(Freq ~ Gender + Admit,data=DF) [same as before but now we do SUM(Freq) in 2 dims]**

**xtabs(Freq ~.,data=DF) [same as before but the period means break out by every other column!]**

**ftable([put the xtabs data in here]) [flat tables, a more compact version of the xtabs data result]**

**seq(along = x) [adds an index]; x$newCol <- factor(x$col1) [creates a factor column]**

**relevel(factors, ref=”yes”) [yes will now be the lowest value]; as.numeric(x) [convert to numeric]**

**EDITING TEXT VARIABLES:**

**tolower(names(data)) [make the column names lowercase; there is also toupper()]**

**splitNames <- strsplit(names(data), “//.”)**

**firstElement <- function(x) {x[1]} [defines a function]**

**sapply(splitNames, firstElement) [applies the function defined above to each element]**

**sub(“\_”,””,names(data)) [substitutes the first occurrence of “\_” with “”]**

**gsub(“\_”,””,names(data)) [substitutes all occurrences of “\_” with “”]**

**grep(“Alameda”, data$column2) [returns the indices where the column value is like “Alameda”]**

**grepl(“Alameda”, data$column2) [returns the FALSE count and TRUE count]**

**grep(“Alameda”,data$column2,value=TRUE) [shows the value instead of the index]**

**nchar() [number of chars]; substr(string,1,7) [substring same as SQL];**

**paste(“jeff”,”rey”,”sep=””) [useful when putting things together like URLs; this concat function]**

**str\_trim(“string ”) [removes white spaces to a string]**

**REGULAR EXPRESSIONS: literals and metacharacters.**

**^ [start of a line]; $ [end of a line]; The square brackets list a set of characters we will allow;**

**capital and lower case characters are different; the ^ found inside brackets means allow everything except this list of characters; the period is a wildcard for any character; the pipe | translates to “or”;**

**flood|fire|earthquake [will match any line with one of these words]**

**^Good|bad [the Good must be found at the beginning of the line, but the bad does not because | ]**

**The ? means that the expression is optional; We must escape metacharacters with backslash;**

**The \* and + symbols mean repetition, \* means 0 or many, + means 1 or many;**

**Curly brackets let you specify the minimum and maximum number of matches**

**[Bb]ush( +[^ ]+){1,5} debate [there is a space, then not a space 1 to 5 times]**

**The normal parentheses are used to divide things and group order of operations; Normal parentheses can be used to remember text, we refer to matched text with \1, \2, etc.;**

**+([a-zA-Z]+) +\1 + [this example looks for lines where a word repeats]**

**The + and \* are greedy, putting a question mark after it will make it take the minimum length.**

**Working with Dates:**

**%d = day as number (0-31), %a = abbreviated weekday, %A = unabbreviated weekday,**

**%m = month (00-12), %b = abbreviated month, %B = unabbreviated month,**

**%y = 2 digit year, %Y = four digit year**

**format(d2, “%a %b %d) [takes a date and returns a string]**

**as.Date(x, “%d%b%Y”) [creates a date out of a string]**

**z[1] - z[2] Time difference of -1 days [Dates can be subtracted from one another]**

**weekdays(d2); months(d2); julian(d2) [# of days since 1970-01-01]**

**library(lubridate); ymd(“20140108”); mdy(“08/04/2013”); dmy(“03-04-2013”)**

**[lubridate can work with the common date formats to convert text to a date with these FUN]**

**ymd\_hms(“2011-08-03 10:15:03”, tz=”GMT”); wday(x); wday(x,label=TRUE);**

**Use classes Date, POSIXct, or POSIXlt.**

**GET FREE DATA:**

|  |  |
| --- | --- |
| **United Nations** | **http://data.un.org/** |
| US | http://www.data.gov/ |
| UK | http://data.gov.uk/ |
| France | http://www.data.gouv.fr/ |
| Ghana | http://data.gov.gh/ |
| AU | http://data.gov.au/ |
| Germany | http://www.govdata.de/ |
| Hong Kong | http://www.gov.hk/en/theme/psi/datasets/ |
| Japan | http://www.data.go.jp/ |
| Many more | http://www.data.gov/opendatasites |
| Gapminder; human health | http://www.gapminder.org/ |
| Survey data from US | http://www.asdfree.com/ |
| Infochimps Marketplace; various datasets some are free | http://www.infochimps.com/marketplace |
| Kaggle; offers data science competitions and interesting data | http://www.kaggle.com/ |
| Collections by data scientists research quality | http://bitly.com/bundles/hmason/1  https://delicious.com/pskomoroch/dataset  http://www.quora.com/Jeff-Hammerbacher/Introduction-to-Data-Science-Data-Sets  http://www.kdnuggests.com/gps.html  http://blog.mortardata.com/post/67652898761/6-dataset-lists-curated-by-data-scientists |
| Standford Large Network Data |  |
| UCI Machine Learning |  |
| KDD Nuggets Datasets |  |
| CMU Statlib |  |
| Gene expression omnibus |  |
| ArXiv Data |  |
| Public Data Sets on Amazon Web Services |  |
| **API** |  |
| twitter and twitteR package |  |
| figshare and rfigshare package |  |
| PLoS and rplos package |  |
| rOpenSci |  |
| Facebook and RFacebook package |  |
| Google maps and RGoogleMaps |  |

**Principles: Show Comparisions; how do you think the world works;**

**show many variables at once; use multiple plot types;**

**keep a code book; above all have a purpose. Also, if there are a lot of outliers or skewed data it is often helpful to take the log of the data to view the graphs;**

**Multivariate Plots: library("car"); scatterplotMatrix(x[,2:5]) [select all rows for your columns]**

**Save Plot to File: pdf(file = “myplot.pdf”); call plot and annotate; dev.off(); svg() [web format]**

**png(); jpeg() [efficient]; tiff(); bmp(); [these 4 are more efficient but don't re-size well] ?Devices**

**dev.copy(png, file = “myplot.png”) [This is another way: send current screen plot to a file];**

**Open Multiple Plot Windows: windows(); dev.cur(); dev.set(<Integer>) ?Devices**

**BASE PLOTS: (library: graphics)**

**Great Function: example() [shows graphical examples of use]**

**summary(column); boxplot(column, col=”blue”);**

**plot(x,y); hist(x);**

**with(data, plot(y, z),col=region) [scatterplot]; text(x+0.05,y+0.05,labels=as.character(1:5)) [label];**

**abline(v=;h=); lines; points; text; title; legend; legend=unique(data$col1), col=unique(data$col1)**

**par(mfrow=c(),bg [background]) [multiple plots]; COMMON ARGUMENTS:**

**target ~ feature; data = ; col =;column$;**

**subset(data, boolean)$column**

**pch [plot symbol]; lty [line type]; lwd [line width]; xlab; ylab [labels]; cex [scaling]; frame=FALSE; type=”l”; lwd=3;**

**plot(type=”l", ylim = c(-3,3)) [type="l" is line plot; limits included to remove outliers]**

**smoothScatter(x, y)**

**REGRESSION CODE:**

**model <- lm(target ~ feature, data); summary(model); plot(model); anova(model);**

**abline(model)**

**PLOT SUBSETS:**

**plot(x, y, type = “n”); points(x[boolean], y[boolean], col=)**

**LATTICE PLOTS: (library: lattice) specify all plot details in one step.**

**xyplot(target~feature | separation \* separation2, data=, layout=c()) [scatter plot];**

**splom [scatterplot matrix]; dotplot [dots plot];**

**bwplot [box and whiskers]; stripplot [another boxplot];**

**levelplot & contourplot [plot “image”]; histogram;**

**xyplot(y ~ x | f, panel = function(x, y, ...) {**

**panel.xyplot(x, y, ...)**

**panel.abline(h = median(y), lty = 2)**

**}) [This will draw a horizontal line at the median y value]**

**xyplot(y ~ x | f, panel = function(x, y, ...) {**

**panel.xyplot(x, y, ...)**

**panel.lmline(x, y, col = 2)**

**}) [This will draw a regression line to the data]**

**GGPLOT2: (library: ggplot2)** **requires DFs; use factors to use subsetting; made up for aesthetics [size= subsetByCol, shape= subsetByCol, color=subsetByCol] and geoms [points, lines];**

**qplot(x, y, data=,color=[ColumnToSubset]) [easiest plot; color makes auto legend];**

**qplot(x, y, data=,geom= c("point","smooth")) [some statistic; this shows a curved regression line];**

**qplot(x, data=, fill= [ColumnToSubset]) [Creates a histogram of x with each count];**

**qplot(x, y, data=,facets= .~[ColumnToSubset]) [subsets data in separate plots];**

**qplot(x, data=, facets= [ColumnToSubset] ~., binwidth=2) [separate histograms];**

**facets takes two arguments separated by ~, left subsets by rows, the right by columns (.~z)(z~.);**

**qplot(x, data=, geom = "density",color= [ColumnToSubset]) [Line based histogram]**

**(...,color=, geom="c("point","smooth"), method="lm") will create two regression & 95% CI**

**[specifying method="lm" will make a straight regression line rather than a curved one]**

**g <- ggplot(data, aes(x, y)); summary(g); [core plot; we require additions before it plots]**

**g + geom\_point() [can add to your plot using the plus operator] [this plots, default point used];**

**g + geom\_point() + geom\_smooth(); g + geom\_point() + geom\_smooth(method = "lm")[regression]**

**g + geom\_point() + facet\_grid(. ~ [ColumnToSubset]) + geom\_smooth(method = "lm") [graphs]**

**xlab(), ylab(), labs(), ggtitle(); theme() [global]; theme\_gray(), theme\_bw() [many defaults set]**

**aesthetics: color="red",size=4,alpha=1/2,aes(color=[ColumnToSubset]) [can be used geom\_point]**

**g + labs(title="") + labs(x = expression("log " \* PM[2.5]), y = "") [set title and xaxis, yaxis]**

**g + geom\_smooth(size=4, linetype=3, method="lm", se=FALSE) [se=95% CI; type is dot]**

**g + theme\_bw(base\_family="Times") [using a theme with many defaults, also new font specified]**

**g + geom\_line() [plots a line graph];**

**CATEGORIZE CONTINUOUS VARIABLES: [useful for facet\_grid on continuous vars]**

**cutpoints <- quantile(data$col1, seq(0, 1, length=4), na.rm=TRUE) [calculate the deciles of data];**

**data$newColumn <- cut(data$col1, cutpoints) [cut the data at the deciles and create a new factor];**

**levels(data$newColumn) [see the levels of newly created factor variables]**

**facet\_wrap([sub1] ~ [sub2], nrow = 2, ncol = 4) [makes panels]**

**OUTLIERS: exclude: g + geom\_line() + ylim(-3,3) [will remove those points from graph]**

**set window limits: g + geom\_line() + coord\_cartesian(ylim = c(-3, 3)) [included but out of window]**

**HIERARCHICAL CLUSTERING: Visualizing high dimensional data; Good for Exploring;**

**how do we... define close; group things; visualize groupings; interpret groupings;**

**Method: Find two closest points; remove them and create a point in the middle of them; repeat;**

**Distance Metric: euclidean distance, correlation similarity [cont;]; manhattan distance [binary];**

**Euclidean: Find difference of some dimensions, square it, add them, and take square root;**

**Manhattan: in terms of city blocks away, only move 4 directions; |A1-A2|+|B.. for each dimension;**

**PLOT: dist(dataFrame) [takes DF, gives all pairwise distances; default is euclidean]**

**distxy <- dist(dataFrame)**

**hClustering <- hclust(distxy) [further down the tree, the sooner it was clustered];**

**plot(hClustering) [cut the tree at a certain point to determine how many clusters there are];**

**BETTER PLOTS: myplclust(hClustering, lab=rep(1:3,each=4), lab.col=rep(1:3,each=4))**

**[This is done after you decide what the clusters are, here we know there are 3 clusters 4 points ea.]**

**dataMatrix <- as.matrix(DF)[sample(1:12),]; heatmap(dataMatrix) [runs hierarchical analysis on the rows of the table and the column of the table. Helps to visualize high dimension cluster data]**

**CLUSTER DISTANCES: complete linkage: take distance of furthest two points in 2 clusters.**

**average linkage: distances between the two centers of gravity;**

**K-MEANS CLUSTERING: summarize data, what data is similar and what is very different;**

**[requires the # of clusters, a distance metric, initial guess clusters centroids]**

**choose a spot for each cluster, it will then find which points belong to each cluster, then it finds the centroid of each cluster, then we find the points close to these new centroids, and it repeats.**

**kmeansObj <- kmeans(dataFrame, centers=3); [param: x, centers, iter.max, nstart]**

**names(kmeansObj); kmeansObj$cluster; [note: sometimes k-means don't make stable results]**

**K-MEANS PLOT: plot(x, y, col = kmeansObj$cluster, pch = 19, cex = 2)**

**points(kmeansObj$centers, col=1:3,pch=3,cex=3,lwd=3)**

**K-MEANS HEATMAPS: visualize clusters;**

**dataMatrix <- as.matrix(dataFrame)[sample(1:12), ]**

**kmeansObj2 <- kmeans(dataMatrix, centers=3)**

**par(mfrow=c(1,2))**

**image(t(dataMatrix)[, nrow(dataMatrix):1], yaxt="n")**

**image(t(dataMatrix)[, order(kmeansObj$cluster)], yaxt="n")**

**DIMENSION REDUCTION: Reduce the variables, and keep the ones that are uncorrelated;**

**SVD (singular value decomposition): you have matrix X, columns are vars, rows are observations.**

**X = UDVT ; U are orthogonal (left singular vectors), columns of V are orthogonal (right singular vectors), D is a diagonal matrix (singular values);**

**PCA (principle components analysis): uses SVD and is a similar technique. If you take the original data matrix and subtract the mean of each column to the respective column, and divide by the column standard deviation; then run SVD on the normalized matrix. The principle components would be equal to the right singular values of the matrix (the V matrix)**

**COMPONENTS OF SVD - u and v: svd1 <- svd(scale(dataMatrixOrdered))**

**par(mfrow=c(1, 3))**

**image(t(dataMatrixOrdered)[, nrow(dataMatrixOrdered):1])**

**plot(svd1$u[, 1], 40:1, , xlab="Row", ylab="First left singular vector", pch=19)**

**plot(svd1$v[, 1], xlab="Column", ylab="First right singular vector", pch=19)**

**COMPONENTS OF THE SVD - VARIANCE EXPLAINED: D Only contains diagonal;**

**plot(svd1$d, xlab="Column", ylab="singular value", pch=19)**

**plot(svd1$d^2/sum(svd1$d^2), xlab="Column", ylab="Prop. of variance explained", pch=19)**

**RELATIONSHIP TO PRINCIPLE COMPONENTS: to show pca and svd are similar;**

**svd1 <- svd(scale(dataMatrixOrdered))**

**pca1 <- prcomp(dataMatrixOrdered, scale = TRUE)**

**plot(pca1$rotation[, 1], svd1$v[, 1], pch=19, xlab="Principle Component 1", ylab="Right Singular Vector 1"); abline(c(0, 1)); [this shows that pca and svd give the same result]**

**VARIANCE EXPLAINED: If the first 5 columns are zeros and the second 5 columsn are ones, then svd shows that the first singular value is high, the rest are about zero. The first singular value explains 100% of the variation in the dataset.**

**MISSING DATA: library(impute); [we need to remove NA or it won't work]**

**dataMatrix2 <- imput.knn(dataMatrixOrdered)$data [it categorize the points with NA]**

**svd1 <- svd(scale(dataMatrixOrdered)); [the NA value will be replaced with the average of]**

**svd2 <- svd(scale(dataMatrix2)); plot(svd2$v[,1],pch=19) [others in its category]**

**COLORS: using better colors in graphs; R default isn't very good; color to communicate;**

**library(grDevices);**

**colorRamp: Take a palette of colors and return a function that takes values between 0 and 1, indicating the extremes of the color palette.**

**colorRampPalette: Take a palette of colors and return a funciton that takes integer arguments and returns a vector of colors interpolating the palette;**

**pal <- colorRamp(c("red", "blue")); pal(0) [255 0 0]; pal(1) [0 0 255]; pal(0.5) [127.5 0 127.5]**

**pal(0, 1, length = 10) [gives a sequence of colors between red and blue at equal intervals]**

**pal <- colorRampPalette(c("red", "yellow"))**

**pal(2) [#FF0000] [#FFFF00] [takes integer arguments to tell how many mixes of colors to make]**

**pal(10) [#FF0000] [#FF1C00] [#FF3800] [#FF5500] [#FF7100]**

**[#FF8D00] [#FFAA00] [#FFC600] [#FFE200] [#FFFF00]**

**CREATE YOUR OWN COLORS: library(RColorBrewer ); [Three types of palettes.]**

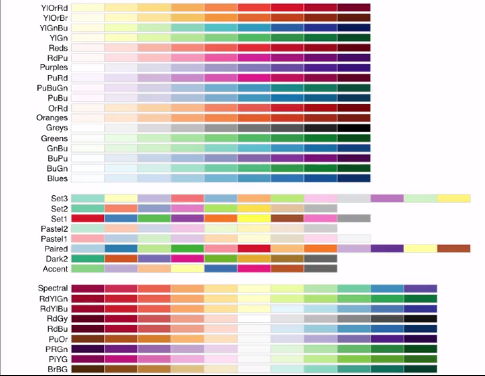
**[sequential palette, diverging palette, and qualitative palettes.]**

**Sequential: data is ordered from low to high**

**Diverging: data diverges and you want to show how it diverges from the mean**

**Qualitative: data that isn't ordered at all**

**[Top palette is sequential; next is qualitative; last is diverging palettes]**

****

**brewer.pal(3, "BuGn") [brewer.pal is the only useful function in this package]**

**arguments are, the number of colors you want in your palette (your primary colors), usually small 2,3,4. The second argument is the name of your palette. The helper function helps find the palette names.**

**cols <- brewer.pal(3, "BuGn")**

**pal <- colorRampPalette(cols)**

**image(volcano, col = pal(20)) [uses the initial palette of 3 colors and creates 20 colors]**

**RGB COLORS:**

**rgb() function can be used to produce combinations of red green and blue. There is a fourth parameter for the transparency of the color, 0 is completely transparent, 1 is not transparent.**

**library(colorspace) [isn't talked about in detail here, but might be worth exploring];**

**[use transparency when the plot points overlap to see more of a histogram view of the data.];**

**Data Project Basics and Analysis Folder:**

**Communicate your data analysis; New people repeating an experiment can help verify results;**

**There are barriers to conducting studies, so if others can't replicate your study you should allow them to reproduce your study by making data and computations available;**

**Use text and code, each chunk will do something, it's both human and machine readable;**

**knitr package needed; also RMarkdown is good; script everything, write everything down!;**

**Steps: define the question; define the ideal data set; determine what data you can access; obtain the data; clean the data; exploratory data analysis; statistical prediction/modeling; interpret results; challenge results; synthesize/write up results; create reproducible code;**

**ideal data set: descriptive - whole population; exploratory - random sample with many vars;**

**inferential - the right population, randomly sampled; predictive - a training and test data set from the same population; causal - data from a randomized study; mechanistic - data from all components of the system;**

**what you can access: free data on web; buy data (respect terms of use); generate it yourself**

**obtain data: try to obtain the raw data; reference the source; polite emails; record url and time;**

**clean data: raw -> tidy; if tidy understand how it came to be; understand the source of the data;**

**may need to reshape the data or re-sample; determine if data is good enough (o.w. get new data);**

**resampling methods: 1) half training, half test 2) K-fold cross validation (K-1 training and 1 test)**

**3) Mask one value randomly and use the rest of the data to predict that value, repeat n times;**

**stat prediction/modeling: based on your question, and results from exploratory analysis you choose your pursuit and method; account for transformations/processing that you did earlier;**

**Keep in mind, what are the measures and/or sources of uncertainty in yourW data set;**

**interpret results: use appropriate language, don't go beyond what you're analysis did; Good words are “prediction”, “it correlates with”, “variables associate with the outcome”, “analysis is descriptive”; give an explanation; interpret coefficients; interpret measures of uncertainty;**

**Challenge results: challenge question, data source, processing, analysis, conclusions, measures of uncertainty, choice of terms to include in models; think of potential alternative analyses;**

**Synthesize/write up: reduce down to less details for a coherent story; lead with the question; summarize the analyses into the story; don't include every analysis, only use it if it is needed for the story or needed to address a challenge; order analyses according to the story, rather than chronologically; include “pretty” figures that contribute to the story;**

**Your Analysis Folder Keeps: raw data, processed data, exploratory figures, final figures (pretty), raw / unused scripts, final scripts, R Markdown files, README.txt, text of analysis / report;**

**Name the files and make it easy to find where in the script files were generated; few final figures;**

**Keep dead end raw scripts; final script: only keep what is analyzed in the paper & comment well;  
Store only the final processed data step, and R code should recreate this file with the raw data!**

**README: Where you got the data (The URL), what is the data and what it is for, date accessed;**

**R Markdown: Easy in RStudio; replaces README file; Text and R code together;**

**Paper: tells a story; title, introduction that motivates problem, methods of statistics, results with measures of uncertainty, and conclusions including potential problems; Reference the statistical methods used (software and implementation); supplementary materials, code, and data; Don't include every analysis you performed!;**

**Coding:**

**RStudio>Preferences>Code: Tab width: 8, Margin: 80; Ctrl+a, Ctrl+i will indent your code;**

**Make functions for short tasks!; don't let a function do many things!;**

**Markdown: easy text-to-HTML conversion tool; easy read/write plain text format then convert;**

**\*text\* is italicized; \*\*text\*\* is bold; #, ##, ### creates headings (h1,h2,h3); - text is a list;**

**1. text is another list (2. text, 3. text); [Text description](http://www.URL) will give a hyperlink;**

**another hyperlink: [Text1][1] and [Text2][2], End of document: [1]: URL1 “Text1” [2]: URL2 “T**

**Newlines: use a double space at the end of a line;**

**R Markdown can execute code; .Rmd; R markdown is converted to Markdown with knitr package; R Markdown -> Markdown -> HTML; Create slides: R markdown -> slidify package;**

**RStudio > File > R Markdown creates a document, when done click “Knit HTML”; “Publish”;**

**R Markdown: \* text is unordered list; '''{r ChunkName} My R Code''' allows running R code;**

**============= makes what is above it the title; ''' {r, echo=FALSE} ''' will only show results!;**

**''' {r, results=”hide”} hides the results; ''' {r, echo=FALSE, results=”hide”} hides both;**

**When typing text outside of the r block you can pull variables into the text, to do this use the format 'r variableName'. ie. '''{r} number <- 32 ''' My favorite random number is 'r number'**

**'''{r, fig.height=4} Code here''' this will change the output figure size for your plot (also width);**

**'''{r, results=”asis”} library(xtable)**

**xt <- xtable(summary(fit)); print(xt, type = “html”)''' Shows a nice table! package required;**

**cache=TRUE allows the code chunk to be stored for future use in case it is slow! (careful);**

**other ideas for when to use R Markdown: manuals, technical documents, tutorials, reports (esp. if generated periodically), data preprocessing documents/summaries;**

**cacher package (extra): objectcode(“varname”); runcode function; loadcache(); effect; stderr;**

**The Project**

**Start with a good question that interests you; have a coherent focused question will help immensely; focused questions mean your job is easier and removes complexity;**

**DONTS: Don’t edit spreadsheets by hand; Don’t edit tables or figures with rounding and formatting; Don’t download data by clicking; Don’t move data around computer; Don’t “We’re just going to do this once…”; Don’t use a GUI; these are not reproducible;**

**All things done by hand MUST be documented so avoid doing anything by hand!;**

**Don’t keep exploratory figures and output early on, at the end of the project recreate these;**

**DO: do teach a computer to do it; use version control and many commits; Keep track of your programming environment: CPU (intel, AMD, ARM), OS (Windows, Mac OS, Linux), compilers, interpreters, command shell, languages (C, Python), libraries, R packages, websites, data repositories, remote databases, version numbers; sessionInfo() tells a lot about your environment;**

**Set your seed for random numbers; Always think if everything can be reproduced;**

**Process of generating conclusions about a population from a noisy sample; generating new knowledge to get answers about a population; Think of probability as being able to repeat an experiment over and over and over and the percentage of times that something happens defines that population parameter;**

**Probability**

**Given a random experiment (say rolling a die) a probability measure is a population quantity (an intrinsic property of the die) that summarizes the randomness. When we talk about probability we aren’t taking about something in the data we have, but as a conceptual thing that exists in the population that we would like to estimate; P(A||B)=P(A)+P(B)-P(A&&B);**

**Densities and mass functions for random variables are all we will need for probabilities of numeric outcomes of experiments; The density is not a statement about what occurs in the data, you’re talking about population quantities; We will collect data that will be used to estimate properties of the population; discrete probabilities: we will assign a probability to each value it can take; continuous probabilities: we will assign a probability to the ranges that it can take; The web site traffic on a given day is a discrete random variable, but it is interesting because there is no upper bound. We will likely use the Poisson distribution to model that;**

**A probability mass function evaluated at a value corresponds to the probability that a random variable takes that value; To be valid a pmf function must satisfy that it’s greater than zero and the sum of all possible values add to one; Binomial pmf: used for flipping coins. Poisson: used for modeling counts. Bernoulli: flip one coin;**

**EX: X = 0 represents tails and X=1 represents heads; upper case represents a potentially unrealized value of the random variable; so we can talk about the probability that capital X equals zero or one;**

**p(x) = (1/2)x (1/2)1-x for x = 0,1; p(x) = (θ)x (1- θ)1-x for x = 0,1; data can be used to estimate population proportion theta; An example is the probability that someone has hypertension or not;**

**A probability density function evaluates the probability in a range of values for continuous variables; All areas must be at least probability zero, and the total area under it must be one;**

**Beta continuous distribution is a triangle with width 1 and height 2, with it’s highest point being at point (1,2); The area under the curve can be found by the triangle area function; Also you can use pbeta(0.75, 2, 1) to find the area under the curve; The “p” stands for probability from a beta density of being less than 0.75; This “p” gives a result similar to the cumulative distribution function; “p” functions can take arrays as the first argument;**

**The cumulative distribution function of a random variable X returns the probability that the random variable is less than or equal to the value x; F(x) = P(X<=x); works for discrete and continuous; The survival function of a random variable X is defined as the probability that the random variable is greater than the value x; S(x) = P(X>x); S(x) = 1 – F(x); the cdf is found by integrating the pdf;**

**The quantiles align the observations from least to greatest and it finds the score such that 95% score worse and 5% score better than you; The αth quantile of a distribution function F is the point xα so that F(xα)= α; median is the 50th percentile; nearly always just use the common distributions to find quantiles; qbeta(-.5, 2, 1) gives the quantile at the 50th quantile; “q” stands for quantiles;**

**We treat the median of our population as the median of our population, and certain assumptions are made; The sample median is the estimator and the population median is the estimand; every estimator has an associated estimand, and this is the formal process of statistical inference, linking your sample to a population;**

**conditional probability P(B|A) = P(A&&B) / P(A); If you can count the outcomes and they are all equally likely then this next formula is correct: P(B|A) = (# of outcomes in A && B) / (# of outcomes in event A);**

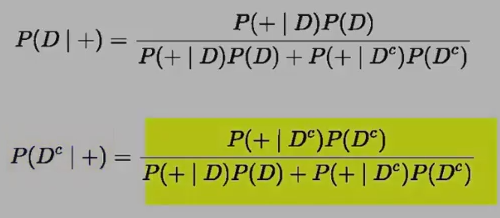
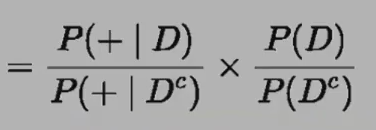
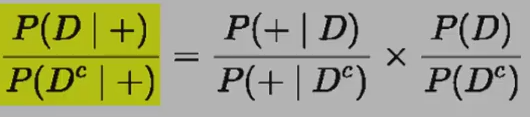
**Joint probability P(A&&B) = P(A)P(B|A) = P(B)P(A|B); If you can count the outcomes and they are all equally likely then this next formula is correct: P(A&&B) = (# of outcomes in A && B) / (Total # of outcomes);**

**Mutually exclusive events can’t occur at the same time; P(B|A) = 0; P(A&&B) = 0; P(B||A) = P(A) + P(B);**

**Independent events don’t affect each other’s probabilities; P(B|A) = P(B); P(A&&B) = P(A)P(B); P(A||B) = P(A) + P(B) + P(A)P(B);**

**Bayes’ rule finds P(B|A) given P(A|B); P(B|A) = P(A|B)P(B) / ( P(A|B)P(B) + P(A|Bc)P(B c) );**

**D: has disease; Sensitivity = P(+|D); Specificity = P(-|D c); Want both of these high for the test to be good; P(D|+) found by Bayes’ rule is the positive predictive value; For our example P(+|D) = .997 and P(-|D c) = .985; Given a positive test result, here are the probabilities of having the disease or not having the disease, only the numerator changes; we will divide these two results to work with the likelihood ratios;**

****

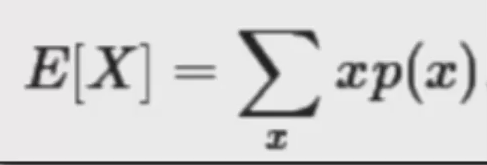
**If you take the probability and divide it by one minus the probability you get the odds; So we have the odds of the disease given a positive test result; The post-test odds of disease are equal to the diagnostic likelihood ratio times the pre-test odds of disease; post-test odds of D = DLR+ \* pre-test odds of D; For our example, DLR+ = .997/(1-.985)= about 66, DLR- = (1 - .997) / .985 = about .003**

**If a positive test, whatever your pre-test odds are, you multiply them by 66 to get your post-test odds; If a negative test, you multiply the pre-test odds by .003 to get your post-test odds;**

**We will assume the collection of random variables used are independent and from the same distribution;**

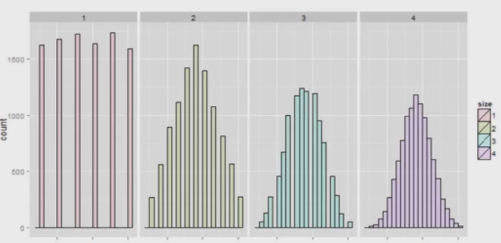
**Expected Values**

**The density and mass functions have characteristics and these are also characteristics of the random variables drawn from them. One of the most useful characterizations is called expected values, another type is quantiles covered above; The mean is the most useful of the expected values; The mean describes the center of the distribution and the variance and standard deviation tell us how spread out it is; Our sample mean will be an estimation of our population mean; Our sample variance will be an estimation of our population variance; Our sample standard deviation will be an estimation of our population standard deviation; The expected value is the mean;**

** We multiply x times the probability of each value;**

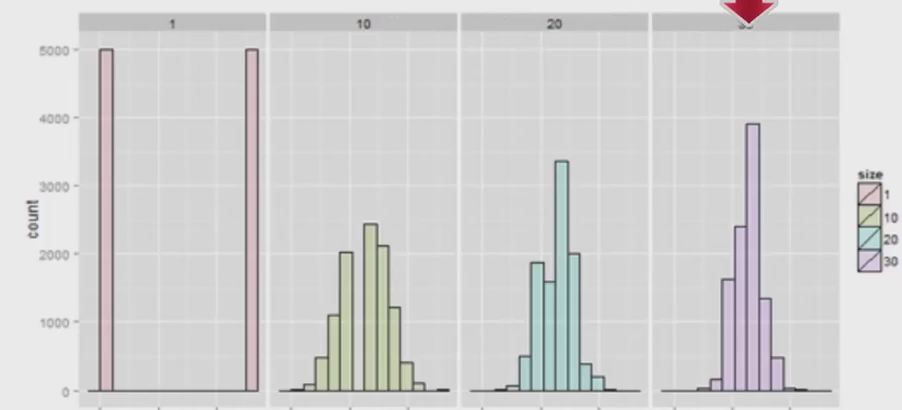
**The sample mean is the center of mass if we treat each sample point as equally likely;**

**The mean for a continuous rv. Is still the center of mass;**

**The average of random variables is itself a random variable; ie: Repeatedly rolling 6 dice and finding their average; Because it is a random variable it also has a distribution and that distribution has characteristics like an expected value and standard deviation; the center of this distribution is the same as the original distribution; This is useful when finding the population mean from the sample mean because they should be about the same (the sample mean is unbiased because its distribution is centered at what it’s trying to estimate);**

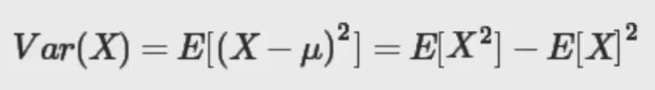
**This is a distribution of die rolls and averages of die rolls; the four figures are the averages of 1, 2, 3, and 4 die rolls;**

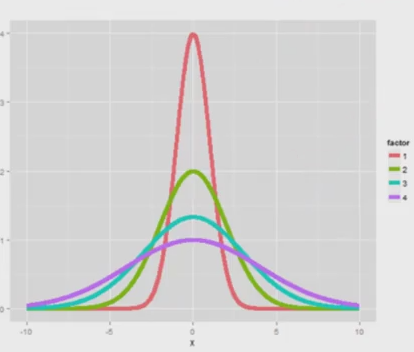
**With multiple rolls it becomes more Gaussian; notice that all four are centered at the same location;**

**This is the same but using flips of the coin one, ten, twenty, or thirty times; As the average is comprised of more coin flips, its distribution gets more concentrated around the mean, however the distribution is always centered at the same place;**

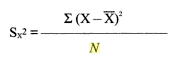
**Variance**

**The variance is how fat or how thin the distribution is around the mean; variance is exactly the expected square distance that the random variable is from the mean; A shortcut is to use the expected value of the random variable squared minus the square of the expected value of the random variable, as shown below; Below is the population variance;**

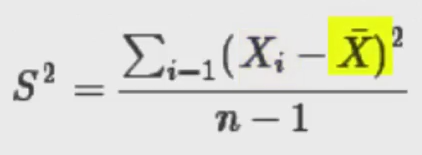
****

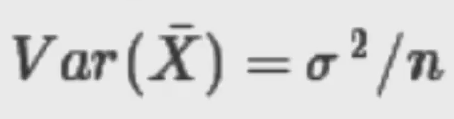
**Higher variance creates a more wide distribution;**

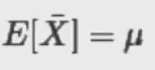
**The sample variance is:**

****

**The estimated population variance is:**

****

**The sample variance itself is a random variable, and the expected value is centered at the population variance; As you collect more data, the distribution of the sample variance will get more concentrated around the population variance it is trying to estimate; We use n-1 instead of n because the sample variance tends to underestimate the population variance; n-1 is called the degrees of freedom;**

****

**The average of the numbers that are sampled from a population (X bar) itself is a random variance, it has its own population mean and population variance; The variance of the sample mean decreases to zero as n accumulates more data; The sample variance S squared, estimates the population variance, sigma squared; standard deviation is the standard error; The variance of the sample mean is σ2 / n, the logical estimate is s2 / n, the logical estimate of the standard error is S / sqrt(n);**

**Sample standard error of the mean:**

****

**S, the standard deviation, is how variable the population is;**

**S / sqrt(n), the standard error, is how variable averages of random samples of size n from the population are;**

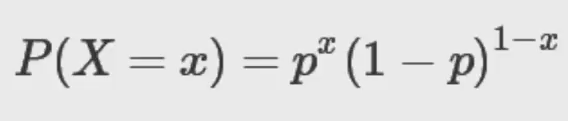
**Useful functions for simulations**

**sample(0 : 1, n, replace =TRUE) is for coin flips; rnorm(n) for normal distribution; runif(n) for uniform; rpois(n, lambda) is for Poisson distribution; sd(apply(matrix(sample(0:1, numsim\*n, replace=TRUE), numsim), 1, mean)) will simulate “numsim” simulations with n coinflips each and find the standard deviation of the means; choose(8, 7) will do 8 choose 7; rbinom for binomial;**

**var(x) the variance; var(x) / n the variance of the averages; sd(x) is the standard deviation; sd(x) / sqrt(n) is the standard deviation of the averages; The n refers to the number of observations each average is comprised of, not the “numsim”;**

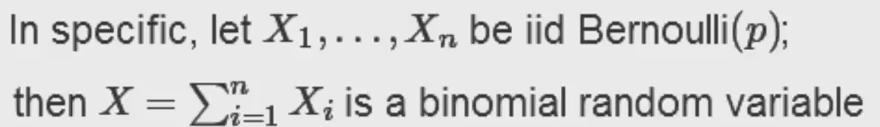
**Distributions**

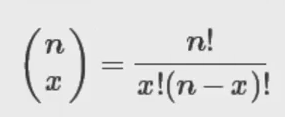
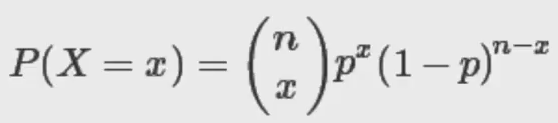
**Bernoulli arises as a result of a binary outcome like a coin flip; values 1 or 0, probabilities of p and 1-p;**

****

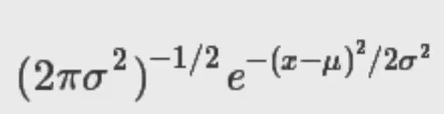
**mean: p; variance: p(1-p)**

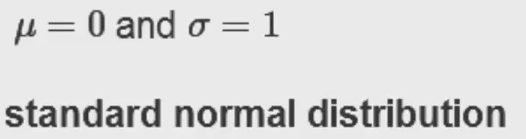
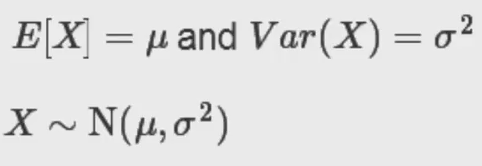
**binomial is the number of heads from flipping n coins with probabilities p and 1-p;**

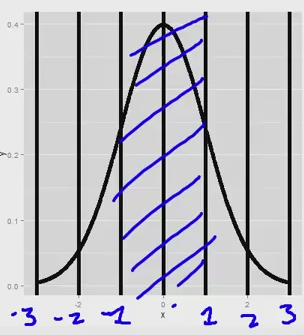
****

** **

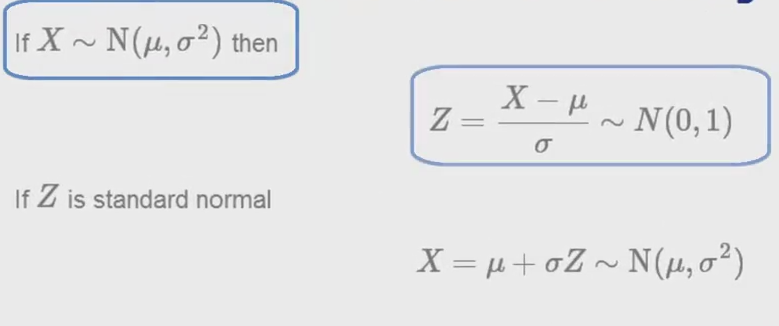
**n choose x counts the number of ways of selecting x items out of n without replacement disregarding the order of the items;**

**Normal distribution is the most important distribution; it has mean μ and variance σ2 if it’s associated density is formula is:**

****

**All normal are the same in respect to their probabilities in terms of mu plus or minus sigma; What changes is the numbers on the x axis**

**How to move from standard normal to normal or vise versa:**

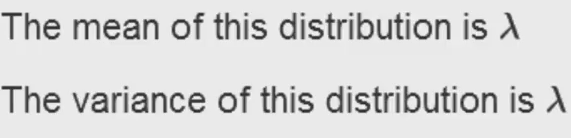
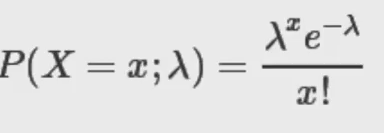
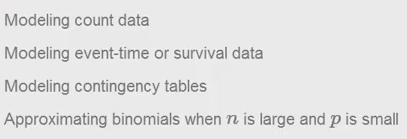
****

**Approximately 68%, 95%, and 99% of the normal density lies within 1, 2 and 3 standard deviations;**

**-1.28, -1.645, -1.96, and -2.33 are the 10th, 5th, 2.5th, and 1st percentiles of the standard normal;**

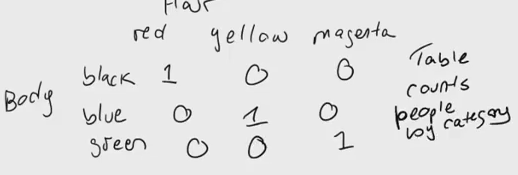
**qnorm(.95, mean=, sd=); pnorm(1.96, mean=, sd=)**

**The Poisson distribution is probably the second most useful distribution; it is used to model counts; the density formula is below, and x is a non-negative integer;**

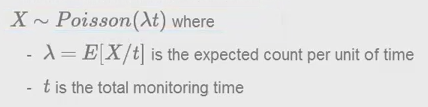
****

**Especially useful if count data is unbounded**

**Modeling the time before reoccurrence of symptoms**

**an example contingency table is shown below;**

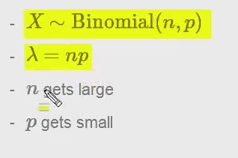
**it is a count by categories;**

**Poisson is used to model rates; **

**Example: The # of people who show up at a bus stop is Poisson with mean 2.5; If watching the bus stop for 4 hours, what is the probability that 3 or fewer people show up the whole time;**

**Ppois(3, lambda = 2.5 \* 4)**

**Poisson is an approximation of the binomial when ni s large and p is small;**

****