Prog. Brief Reference Sheet

Marco Sousa

JULY 2019

Contents

1 Introduction (future me, you better not suck bro)

The purpose of this document is simply to add *quick and brief* notes/suggestions regarding programming.
USE THE TABLE OF CONTENTS PAGE. Click them to skip to the section.
You can technically copy paste, but I'd be careful, sometimes it doesn't ASCII correctly.
Not necessarily a tutorial, but can be very informative.
Also note that my R code looks like file .R code; not console line code (with > and +).
Special thanks to the UMassD professors for their instruction.
Special thanks to Prof Yan Donghui who much of this is taken from, and also for his teaching.

This will be updated as I have time x.x
Thank you for your time, and enjoy.

2 RStudio

A good intro source for R-PRogramming:
https://sites.google.com/site/rprogrammingumassd/home
Another source for more R Programming:
https://www.datacamp.com/

2.1 Elementary Vector Operations

Vectors are [1:n]
Creating a vector:

```
mydata <- c(2.9, 'string',TRUE, value, value,...)
```

List of numbers:

```
numberList <- 25:30
# Note: Will contain 25,26,27,28,29,30  (BOTH start and endpoint)
```

Access a value in VECTOR:

```
mydata[1], or  mydata[1:n]
```

Can test entire vector using logic

```
mydata > 3
#Result: [1]  FALSE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE
```

Adding names:

```
names(mydata) <- c('name1', 'name2',...);
```

Changing a particular name:

```
names(mydata)[1] <- 'newname'
```

letters is a pre-defined R vector of the 26 lowercase letters

```
letters [1:5]
#Result: [1] "a" "b" "c" "d" "e"
```

Finding a vector value associated with a name!

```
mydata['name2']
```

Important note about VECTOR logic!

```
mydata > 3 ##returns TRUE FALSE vector
mydata[mydata > 3] ##returns VALUES which are TRUE
```

MODE tells you the type of values in a vector

```
mode(mydata)
```

DIM returns the dimensions of the data structure

```
dim(mydata)
#Result: 2 5 (if it's a 2 by 5, for example)
```

DIM can also make a vector into a data frame structure

```
dim(mydata) <- c(m,n) (makes it an m by n structure)
dim(mydata) <- NULL (makes structure into vector)
```

ADDING/INSERTING a vector value is complicated...
Appending is discouraged because it copies the vector and adds the last part repeatedly in a for loop. Consider Python/Java for list appending in for loops.

DELETING a vector value

```
mydata2[-c(1,3)]; (deletes at location 1 and 3)
```

Making a list of vectors (list is a data structure)

```
mydatalist <- list(mydata,mydata1,mydata2,...)
## RESULT: [[1]] [1] 1 3 6,  [[2]]  [1]  "red"  "blue"  "green"  "black  ",[[3]]...
```

Accessing a list

```
mydatalist [[1]][2]
```

LENGTH can be found of vectors

```
length(mydata)
```

2.2 Elementary Matrix Operations

Matrix function:

```
matrix(c(value,value ,...) ,nrow,ncolumn,byrow)
A<-matrix(c(1,2,4,7,5,9),2,3,  byrow=T); #example
```

Class tells object type, so matrix or vector...etc.

```
class (A)
```

typeof tells MODE of the ELEMENTS of a matrix

```
typeof(A)
```

MATRIX dimensions

```
dim(A),nrow(A), nrow(A) #fairly obvious
```

Changing column name or ROW name

```
colnames(A) <- c("COL1", "COL2",...)
rownames(A) <- c("ROW1", "ROW2",...)
```

Transpose a matrix:

```
A <- t(A)
```

Deleting a column or columns:

```
##Deleting a single column
Data$genome <- NULL

##Deleting multiple columns
Data[1:2] <- list(NULL) #If you do not include the list, it will not work on the frame
#or
Data <- Data[,-(2:3)]              # vector
Data <- Data[,-(2:3),drop=FALSE] # still a data.frame
```

Matrix multiplication:

```
##Element-wise multiplication:
A*C (sizes 2x3 * 2x3)
##Linear Algebraic multiplication:
A %*% B
```

Binding rows/columns:

```
##rbind STACKS rows on top of one another of same dimension
rbind(A,C)
#Result: puts row of A on top of B in single matrix

##cbind STACKS columns similarly
```

Return/Replace Diagonal values along matrix

```
##Return diagonal values
diag(F)
##Replace diagonal values
diag(F)<-c(1,1);
```

2.3 Sequence Operations

Create a sequence

```
s4<-seq(length=51,from=-1,by=0.2)
```

Remember: Can return value at index n with vector[n]

Repeat a number x, n times:

```
rep(x,n)
#Fill a matrix:
B <- matrix(rep(1,6),3,2)
```

A sequence can be repeated:

```
This repeats x 2 times and stores into y:
x<-1:4;
y<-rep(x,2)
#Result:(1234)(1234)
#This repeats each x i times.
i<-rep(2,4)
z<-rep(x,i)
#Result:(11)(22)(33)(44)
#Note the difference from above: rep(x,2) versus rep(x,rep(2,4))
w<-rep(x,x)
#Result: 1223334444
x<-rep(rep(1:4,rep(2,4)),3)
#Looks complicated by remember the TWO parts of the rep function
#Result: (11223344)(11223344)(11223344) - Notice how it repeats the inner 3 times
```

2.4 Data Frames

A data frame is a matrix-like structure whose columns have the same length but may be of differing data types (e.g., numeric, logical, factor and character and so on).

Creating a data frame:

```
dataFrame <- data.frame(x = 1:2, y = 1:10, z=rep('A', 10))
#x,y,z are column names. It will fill to the largest column's dimension (this means columns must align, so x
    =1:3 would not work for example.
```

2.5 Logical Operators and NA

Logical Operators:

```
<, <=, >, >=, ==, !=
#Element-wise AND: &, Element-wise OR: |
#Logical AND: &&, Logical OR: ||
##The logical operators only consider the first element of a vector
```

When an element or value is "not available," the location is reserved as NA.
In general, any operation on an NA is NA.
The function is.na(x), where x is a vector, gives a logical vector of the same length as x with values set to be true if the corresponding element in x is NA.
The function is.na() is also good for finding locations in which a value is NA.

2.6 Files and Linear Fitting

Read in files, such as .dat,.txt, .csv (Comma seperated files), SPSS, and so on

```
tmp<-read.table("car.txt", header=TRUE, sep=",")
```

Note that excel files are xlsx, and should be saved as .csv first

```
dataFile<-read.csv("storeSales.csv",na.strings="N/A");
```

Other files can be imported from top right (Import Dataset)
Take a column and assign it to a vector:

```
x<-tmp[,1]
y<-tmp[,2]
```

Sorting vectors; index.return keeps track of the index for each value

```
xs<-sort(x, index.return =TRUE);
```

lm collects data to fit linear models

```
mylm<-lm(y ~ x);
```

The plot function:

```
plot(x,y,xlab="Age", ylab="Price",cex.lab=1.5,cex.axis=1.5, bty="l",pch=20, font.axis=2, font.lab=2);
```

Create the fitted line from lm

```
points(xs$x, fitted (mylm)[xs$ix],type="l",lwd=2)
#Note that xs$x and xs$ix select x from xs, and ix from xs
```

2.7 Conditionals (If,else)

```
if (test Expression) {
 statement
 }
```

```
if (x<=2) {print("less than 3")
} else if (x>=4) {print("greater than 3")
} else print("equal to 3")
```

IMPORTANT NOTE: the } MUST exist on the next line
Since R reads line by line, if it is on the same line, it will COMPLETE the statement and the next

line will error.

2.8 Loops (For,While,Repeat,Apply)

The for loop operates similarly to pythonic loops w/ a sequence
FOR Loop:

```
x <- seq(1, 100, by=1);
xsquared = NULL
for (n in 1:100)
{
    xsquared[n] = x[n]^2
}
```

A minor note about the for loop:

```
##The -1 will not subtract JUST the variable; add parenthesis to do so.
for (i in 1:variable -1) {
#do things
}
```

WHILE Loop:

```
x<-0;
while (x < 10) {
    x <- x + 4;
    print (x);
    if (x == 8) {
        break;
    }
}
```

REPEAT Statement

```
sum <- 1;
repeat
{
    sum <- sum + 2;
    print(sum);
    if (sum > 11)
    break;
}
```

apply(x, MARGIN, FUN, ARGs) for 2-dimensional data

```
x <- cbind(x1 = 3, x2 = c(4:1, 2:5));
dimnames(x)[[1]] <- letters [1:8];
apply(x, 2, mean);
col.sums <- apply(x, 2, sum);
row.sums <- apply(x, 1, sum);
```

Apply a function to each cell of a ragged array
tapply(vector, factor, FUN) Apply a funct

```
x<-c(1,2,3,4,5,6,7,8);
names(x)<-c("a","a","b","a","b","a","b","a");
tapply(x,factor(names(x)),mean);
```

lapply() and apply()

```
x <- list(a = 1:10, b = (-3:3)^2, lv = c(TRUE,FALSE,FALSE,TRUE));
# compute the list mean for each list element
lapply(x, mean);
sapply(x,mean);
```

2.9 Functions

General FUNCTION:

```
myfunction<-function(argument1, argument2, ..., argumentn )
{
        R statements; ##Body of a function
        return(output);
}
```

Some notes about functions:
You can make a function in the CONSOLE. It can be edited with fix(func)
However, it is better to save functions in .R files!
Also note that RStudio can run Python/C++ scripts
5 R function examples are provided below:

2.9.1 Number of positive items in a list

```
nPositives<-function(x)
{
  n<-length(x);
  counts<-0;
  for(i in 1:n)
  {
    if(x[i]>0) {
      counts<-counts+1;
    }
  }
  return(counts);
}
```

2.9.2 Finding non-duplicate items in a list

```
nonDup<−function(x)
{
  y<−NULL;
  while(length(x)>=1)
  {
    tmp<−(1:length(x))[x==x[1]];
    if (length(tmp)==1) y<−c(y,x[1]);
    x<−x[−tmp];
  }
  return(y);
}
>x<−c(1,2,3,2,5,3,6);
>nonDup(x)
```

2.9.3 Frequency of letters

```
letterFreq<−function(x)
{
    n<−length(x);
    fTable<−matrix(0,26,1);
    for(i in 1:n)
    {
        idx<−(1:26)[letters==tolower(x[i])];
        fTable[idx] <− fTable[idx] + 1;
    }
    return(data.frame(letters,fTable));
}
>x<−c("D","a","t","a","s","c","i","e","n","c","e","i","s","f","u","n");
>letterFreq(x)
```

Note:idx¡-(1:26)[letters==tolower(x[i])];
The above BRACKET RETURNS the number index in which the case is TRUE.

2.9.4 List of primes within n

```
is.prime<−function(n)
{
  if (n==2) return(1);
  if (n==1 || n%%2==0) {
    return(0);
  }
  m<−floor(sqrt(n))+1;
  for(i in 2:m)    {     if (n %% i ==0) return(0);
    }
  return(1);
}

print.primes<−function(n, h)
{
```

```
  z<-tapply(1:n,as.factor(1:n), is.prime);
  primes<-(1:n)[z==1];
  plot(primes, col="blue", type="b", pch=5,
       xlab="Index",ylab="");
  text(1:length(primes),primes+h,labels=primes);
  return(length(primes));
}
>print.primes(100,2);
```

Note the above uses tapply, which applies a function to each term in a designated sequence. Also note n is how far out in primes and h simply adjusts labels.

2.9.5 Second max value of a list

```
secMax<-function(x)
{
infty<-10^20; #define this as a very small number;
n<-length(x);
if(n <2) return(-infty);
mysec<--infty;
mymax<-x[1];
for(i in 2:n)
{
        if(x[i] > mymax)
        {
                mysec<-mymax;
                mymax<-x[i];
        }
        else
        {
                if(x[i] > mysec)
                {
                        mysec<-x[i];
                }
        }
}
return(mysec);
}
```

2.10 Probability Distributions

2.10.1 Uniform distribution

In interval [a,b]

```
runif(n, min = a, max = b)
```

2.10.2 Normal distribution

With mean u and standard deviation sigma

```
rnorm(n, mean = u, sd = sigma)
```

2.10.3 Binomial distribution

With size n and success probability prob

```
rbinom(n, size, prob)
```

2.10.4 Poisson distribution

With parameter lambda

```
rpois(n, lambda)
```

2.10.5 t distribution

With df degrees of freedom, and non-central parameter (by default 0)

```
rt(n, df, ncp)
```

2.10.6 Sampling data from a list

```
sample(x, size, replace = FALSE, prob = NULL)
```

x:Either a vector of one or more elements from which to choose, or a positive integer.
n:A positive number, the number of items to choose from. See 'Details.'
size:A non-negative integer giving the number of items to choose.
replace:Should sampling be with replacement?
prob:A vector of probability weights for obtaining the elements of the vector being sampled.

2.11 Graphs and Descriptive Stats

```
##Histogram:
hist(rnorm(100,0,1),breaks=8);
##Boxplot:
x<-read.table("newbedfordTempMonth.Data");
names(x)<-c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec");
```

```
boxplot(x [,1],  col="orange", cex.lab=1.5, cex.axis=1.5, font=2);
boxplot(x, col="orange", cex.lab=1.5, cex.axis=1.5, font=2);

##Stem-and-leaf plot:
x<-c  (2.21,2.24,2.28,2.33,2.37,2.52,2.79,2.81,2.82,2.82,2.83,2.83)  ;
stem(x, scale=2);

##QQ-plot
qqnorm(rnorm(100,0,1));
abline(0,1);

qqnorm(runif(100,0,1));
abline(0,1);

##Uncomment when running for the first time
##install.packages("moments");
library(moments);

x<-read.table("eruption.Data");
kurtosis(x);
skewness(x);
```

```
#Data Visualization with some examples:

numbers <- c(3574,1328,6548,1316,1053,626);
colors <- c("blue", "green", "orange", "cyan", "cadetblue4","darkmagenta");
cats <- c("Connecticut", "Maine", "Massachusetts", "New Hampshire", "Rhode Island", "Vermont");

##Barplot
barplot(numbers, names.arg=cats, col=colors, cex.lab=1.5, cex.axis=1.5, cex.names=1.5, font=2, font.main=3);

##Pie Chart
pie(numbers, labels=cats, col=colors, font=2, font.main=3);

##Pareto Chart
install.packages(  qcc  );
library(qcc);
names(numbers) <-cats;
pareto.chart(numbers, ylab= Frequency , main= , col=colors, font=2, cex.lab=1.5, cex.axis=1.5);

##Box Plot (Gives 5 Number Summary, Tukey)
x<- read.table(...);
boxplot(x, col= orange , cex.lab=1.5, cex.axis=1.5, font=2);

##Dotplot
stripchart(x, method= stack , pch=19, col= blue , offset=0.5, ,cex.lab=1.5,cex.axis=1.5);

##Histogram
hist(x, main= , xlab= Length (seconds) , ylab= Count , breaks=20, freq=TRUE, cex.lab=1.5, cex.
    axis=1.5);

##Stem and Leaf Plot
stem(x,scale=2);

##Grouped-Bar Chart
#What the Data looks like:
```

```
Soybean Spect ImgSeg Heart Wine WDBC Robot Madelon
CF 84.43 68.02 48.24 68.26 79.19 88.70 55.12 41.20
RP 71.83 61.11 47.69 60.54 70.79 85.41 55.19 35.50
BC2 72.34 56.28 49.91 59.10 70.22 85.38 50.21 35.37
EA 76.59 56.55 51.31 59.26 70.22 85.41 50.32 37.19

cfr<-read.table("CFrt.Data", header=TRUE);
row.names<-c("CF","RP","BC2","EA");
colors<-c("blue","green","pink","orange");
barplot(as.matrix(cfr),main="", ylab = "Rate", cex.lab=0.8, cex.main=1.4, xpd=FALSE, beside=TRUE, col=
    colors,ylim=c(30,90));
legend("topright", legend=c("CF","RP","BC2","EA"), fill=colors);

##Grouped-Bar Chart - UC Enrollment example:
NYR<-15; NRACE<-9; x<-matrix(0,NYR,NRACE+2);
x[1,]<-c(1996 ,44796 ,6477 ,5883 ,20251 ,1510 ,2807 ,71008 ,6992 ,6994 ,166718);
x[2,]<-c(1997 ,46237 ,6350 ,6211 ,20072 ,1491 ,3142 ,71658 ,7730 ,6971 ,169862);
x[3,]<-c(1998 ,47066 ,6023 ,6499 ,19764 ,1399 ,3293 ,71009 ,8196 ,10394 ,173643);
x[4,]<-c(1999 ,48462,5722 ,6782 ,19739 ,1265 ,3469 ,72606 ,8200 ,12165 ,178410);
x[5,]<-c(2000 ,49735 ,5654 ,7043 ,20083 ,1144 ,3618 ,73714 ,9139 ,13225 ,183355);
x[6,]<-c(2001 ,52339 ,5572 ,7455 ,21386 ,1140 ,3938 ,75647 ,10168 ,14258 ,191903);
x[7,]<-c(2002 ,55852 ,5807 ,7973 ,22976 ,1204 ,4290 ,77833 ,11165 ,14197 ,201297);
x[8,]<-c(2003 ,57951 ,6103 ,8232 ,24771 ,1232 ,4733 ,79145 ,11708 ,14516 ,208391);
x[9,]<-c(2004 ,58762 ,6049 ,8120 ,24992 ,1208 ,5006 ,77655 ,11816 ,14301 ,207909);
x[10,]<-c(2005 ,60297 ,6083 ,8214 ,25518 ,1188 ,5182 ,77374 ,11456 ,13768 ,209080);
x[11,]<-c(2006 ,63064 ,6281 ,8217 ,26907 ,1196 ,5424 ,78453 ,11469 ,13287 ,214298);
x[12,]<-c(2007 ,65534 ,6647 ,8437 ,28613 ,1258 ,5700 ,78957 ,12182 ,12706 ,220034);
x[13,]<-c(2008 ,68031 ,7137 ,8550 ,30981 ,1264 ,5811 ,79080 ,12996 ,12190 ,226040);
x[14,]<-c(2009,69236 ,7508 ,8653 ,33398 ,1344 ,5994 ,79134 ,13809 ,11452 ,230528);
x[15,]<-c(2010 ,69871 ,7990 ,8636 ,35993 ,1539 ,17544 ,77715 ,15176 ,0 ,234464);

#Combine Other and No response (i.e., column 7 and 10==>column8)
#Combine Native and Fillipino (i.e., column 4 and 6==>column4)
y<-matrix(0,NYR,NRACE+2);
for(i in 1:5) {y[,i]<-x[,i];}
y[,4]<-y[,4]+x[,6];
y[,6]<-x[,8];
y[,7]<-x[,7]+x[,9]+x[,10];
y[,8]<-x[,11];
z<-y[,2];y[,2]<-y[,4];y[,4]<-z; ##Swap Asian and Native
for(i in 1:NYR) { y[i,]<-y[i,]*100/y[i,8];}
z<-t(y[,2:7]);
colors<-c("blue","red","gold","grey","cyan", "brown");
ethnics<-c("Other/Unknown", "White", "Hispanic", "Asian",

b<-barplot(z, main="UC enrollment 1996-2010", col=colors, las=2, cex.names=0.5, axes=FALSE, legend.text=
    FALSE, xlim=c(0,28),width=1);
legend("bottomright", bty="n", cex=0.9, legend=ethnics, fill=rev(colors));

##Pyramid Plot - US Population example

library(plotrix);
#The Pop. Data
m1950<-c(8236164,6714555,5660399,5311342,5606293,5972078,5624723,5517544,5070269,4526366,
4128648,3630046,3037838,2424561,1628829,1001798,504958,236828);
m2000<-c(9810733,10523277,10520197,10391004,9687814,9798760,10321769,11318696,11129102,9889506,
8607724,6508729,5136627,4400362,3902912,3044456,1834897,1226998);
f1950<-c(7927407,6485130,5458869,5305256,5875535,6270182,
```

```
5892284,5728842,5133704,4544099,4143540,3605074,3021637,2578375,1783120,1150609,620386,340073);
f2000<−c(9365065,10026228,10007875,9828886,9276187,9582576,
10188619,11387968,11312761,10202898,8977824,6960508,5668820,5133183,4954529,4371357,3110470,3012589);

agelabels<−c("0−4","5−9", "10−14","15−19", "20−24", "25−29",
"30−34", "35−39", "40−44","45−49", "50−54","55−59",
"60−64", "65−69", "70−74","75−79","80−84","85+");
GPS<−18;
ma1950<−matrix(0,1,GPS); ma2000<−matrix(0,1,GPS);
fa1950<−matrix(0,1,GPS); fa2000<−matrix(0,1,GPS);
pop1950<−sum(m1950+f1950); pop2000<−sum(m2000+f2000);
ma1950p<−m1950/pop1950; fa1950p<−f1950/pop1950;
ma2000p<−m2000/pop2000; fa2000p<−f2000/pop2000;
males<−matrix(0, GPS,2);
males [,1] <−t(ma1950p); males[,2]<−t(ma2000p);
females<−matrix(0,GPS,2)
females [,1] <−t(fa1950p); females[,2]<−t(fa2000p);

myPyramid<−pyramid.plot(males∗100, females∗100, labels=agelabels, lxcol=c("blue","green"), rxcol=c("blue","
    green"), laxlab=seq(0,10,by=2), raxlab=seq(0,10,by=2), top.labels=c("Males","Age","Females"), gap=3);
mtext("US population by age and gender",3,2,cex=1.5);
legend(par("usr") [1], GPS,c("1950","2000"),fill=c("blue","green"));

#To restore the margins and background
par(mar=myPyramid,bg="transparent");

##Bubble Chart − Crime Example

crime<−read.csv("http://datasets.flowingdata.com/crimeRatesByState2005.tsv", header=TRUE, sep="\t");
radius<−sqrt( crime$population/pi);
symbols(crime$murder, crime$burglary, circles=radius,inches=0.35, fg="white", bg="red", xlab="Murder Rate",
    ylab="Burglary Rate");
text(crime$murder, crime$burglary, crime$state, cex=0.5);

##Bubble Chart − US Population by States
library (maps);
library (ggplot2);
#Get US map
usa <− map_data("state");
myData = data.frame(name=c("FL","CO","CA","MA","NY"),

lat=c(29,39,38,42,43),
long=c(−82,−105,−120,−71,−75),
pop=c (19.3,5.2,38.0,6.6,19.6) )

#Not exactly sure how this works out...
ggplot() + geom_path(data = usa, aes(x=long, y=lat, group = group)) + geom_point(data = myData, aes(x=long
    , y=lat, size=pop), color = "red");

##More data to analyze:
Age Male (1950) Male (2000) Female (1950) Female (2000)
0−14 20611118 30854207 19871406 29399168
15−29 16889713 29877578 17450973 28687649
30−44 16212536 32769567 16754830 32889348
45−64 15322898 30142586 15314350 31810050
65−84 5560146 13182627 6132490 17569539
85+ 236828 1226998 340073 3012589
```

```
#Can make a scatterplot, or scatterplot by proportion, or by pyramid!
```

2.12 Hypothesis Testing

```
##t-test
x<-c(49, 50, 47.5, 49.6, 48.6, 49.0, 48.6, 49, 49.4, 50.2);
t.test(x, alternative="two.sided", mu=50);

##Wilcox test
x <- c(1.83, 0.50, 1.62, 2.48, 1.68, 1.88, 1.55, 3.06, 1.30);
y <- c(1.67, 0.647, 1.35, 2.05, 1.06, 1.29, 1.06, 3.14, 1.09);
wilcox.test(x, y, paired = TRUE, alternative = "greater");

##Kolmogorov-Smirnov test
ks.test(rnorm(100,0,1), runif(100,-4,4));

##Shapiro-Wilk test for normality
shapiro.test(rnorm(100,0,1));
shapiro.test(runif(100,-4,4));
```

2.13 Logistic Regression

A tutorial: https://www.datacamp.com/community/tutorials/logistic-regression-R
An alternative tutorial:
http://rstudio-pubs-static.s3.amazonaws.com/14336_8e6cdabdc9434030b194dd9ba7b22ea2.html

```
## download for testing Smarket dataset
install.packages("ISLR");
library(ISLR)
## Viewing the distribution of the data?
par(mfrow=c(1,8))
for(i in 1:8) {
    hist(Smarket[,i], main=names(Smarket)[i])
}

## To show correlations! ***
install.packages("corrplot")
library(corrplot)
correlations <- cor(Smarket[,1:8])
corrplot(correlations, method="circle")

## pairs() will create a scatterplot matrix
pairs(Smarket, col=Smarket$Direction)


## LOGISTIC MODEL! ***
glm.fit <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume, data = Smarket, family = binomial)
summary(glm.fit)

```

```
## Logistic prediction function! ***
glm.probs = predict(glm.fit, type = "response")
glm.probs[1:n] ## Views selected probabilities in vector

##making predictions from each probability
glm.pred <- ifelse(glm.probs > 0.5, "Up", "Down")

##Creating a table of correct/wrong predictions.
##Smarket[,9] accesses the 9th column (the binomial column; Direction)
table(glm.pred,Smarket[,9])

##mean value to see how well logistic fit is; [,9] is Direction.
mean(glm.pred == Smarket[,9])

## train allows you to select rows to test with.
train = Year<2005
glm.fit <- glm(.. subset = train)
glm.probs <- predict(...newdata = Smarket[!train,])
Direction.2005 = Smarket$Direction[!train]
table(glm.pred, Direction.2005)
mean(glm.pred == Direction.2005)
```

There are necessary conditions for Logistic regression and this site MAY outline them: https://www.statisticshowto.datasciencecentral.com/assumptions-conditions-for-regression/

Some additional notes:

```
## The par function allows you to edit how many figures can be plotted, and the margin.
## par(mar) tells you the current plot margins
par("mar")
## dev.off() will reset par in case there is an issue w/ dimensions
## Smarket[,n] returns the WHOLE column for index n ***
```

2.14 dplyr

A Tutorial: https://genomicsclass.github.io/book/pages/dplyr_tutorial.html

```
install.packages("dplyr")
library(dplyr)
```

2.14.1 Some comments

dplyr is for cleaning and ordering data. There are other ways, but I like this one.

```
#Know that you don't have to use the following to save to a new dataFrame
{. ->> newDataFrame }
#Instead, you can also just do it normally:
newDataFrame <- myData %>% select( blah blah blah)
```

Since this is an old version, my examples might instead save in the former, where the latter is much nicer.

2.14.2 select()

```
## Select chooses COLUMNS with arguments
## select(datafile, columnName1,columnName2,...)
sleepData <- select(msleep, name, sleep_total)
head(sleepData)
## can select with other arguments
head(select(msleep, -name))
head(select(msleep, name:order))
head(select(msleep, starts_with("sl")))

##Some common arguments:
##ends_with() = Select columns that end with a character string
##contains() = Select columns that contain a character string
##matches() = Select columns that match a regular expression
##one_of() = Select columns names that are from a group of names
```

2.14.3 filter()

```
## filter selects ROWS with argument
## filter(data, argument to slect row)
 filter (msleep, sleep_total >= 16)
 filter (msleep, order %in% c("Perissodactyla", "Primates"))

##See boolean operators (>, <, >=, <=, !=, %in%)
```

2.14.4 Pipe Operator

```
head(select(msleep, name, sleep_total))

msleep %>%
    select(name, sleep_total) %>%
    head

## Are effectively the same; does multiple operations orderly left to right pipeing output to next as input.
## The below suggests one way to save data in between piping
{. ->> newDataFrame }
```

2.14.5 arrange()

```
## To arrange (or ORDER) ROWS by a particular column
head(arrange(msleep, order))
msleep %>% arrange(order) %>% head

msleep %>%
    select(name, order, sleep_total) %>%
    arrange(order, sleep_total) %>%
```

```
    filter (sleep_total >= 16)

#This orders rows BIGGEST to SMALLEST (because of desc) for Length then Width.
arrange( iris , desc(Sepal.Length),desc(Sepal.Width))
```

2.14.6 mutate()

Adds new COLUMNS to the data frame

```
## In this occasion we are STORING rem_proportion as name of new column BASED on sleep_rem / sleep_total.
## We also saved it as a new data frame with the last line!
msleep %>%
    mutate(rem_proportion = sleep_rem / sleep_total) %>%
    {. ->> newmsleep }
```

2.14.7 summarise()

```
##summarise() provides summary statistics.
msleep %>%
    summarise(avg_sleep = mean(sleep_total),
              min_sleep = min(sleep_total),
              max_sleep = max(sleep_total),
              total = n())

# Note the n() function counts the number of observations
```

2.14.8 groupby()

```
##group_by() splits the data frame by a variable, applies a function to the individual data frames, and then
    combine the output.
## The following example outputs a frame of summary statistics for each taxonomic order.
msleep %>%
    group_by(order) %>%
    summarise(avg_sleep = mean(sleep_total),
              min_sleep = min(sleep_total),
              max_sleep = max(sleep_total),
              total = n())
```

2.14.9 Filtering out NA rows

```
## The below filters out na values for column a.
 filter (!is.na(column_a))

## Here's an example in use:
```

```
msleep %>%
      filter (!is.na(sleep_rem)) %>%
     summarise(
          avg_sleeprem = mean(sleep_rem),
          totalrem = n())
```

2.15 R Comments

Comments done with .
Ctrl + Shift + C will comment out a highlighted block.
An introduction to docstrings for R funcrtions can be found here: /urlhttps://cran.r-project.org/web/packages/docstring/

3 Python

TBA
Wish I had more time to fill this out; love Python ;-;

4 Java

yikes

5 Notes to self

I should look at my old data and relearn ANOVA analysis and Tukey Analysis and correlations and such...

sum stuff I gott test

```
testPlot <- ggplot() + geom_path(data = usa, aes(x=long, y=lat, group = group)) + xlab("Longitude") + ylab("Latitude") + ggtitle("2010 Population Bubble Chart")


testPlot + geom_point(data = newtestSelect, aes(x=Longitude, y=Latitude, size=Pop2010), color = "red") + geom_text(data=newtestSelect, aes(x=Longitude,y=Latitude), label=newtestSelect$Abbreviation);


##I also can slice out 2 and 11 to make a better map:


testPlot + geom_point(data = newtestSelect2, aes(x=Longitude, y=Latitude, size=Pop2010), color = "red") + geom_text(data=newtestSelect2, aes(x=Longitude,y=Latitude), label=newtestSelect2$Abbreviation, nudge_x = 0, nudge_y = 0, fontface = "bold");
```

```
#Pairs example:
data( iris );
pairs( iris [1:4], main = "", pch = 21,
bg = c("red", "green3", "blue")[unclass( iris $Species)]) ;

#Chernoff Faces example:
##Uncomment if running for the first time
##nstall.packages("aplpack");
library(aplpack);
##May need to fix Tcl/Tk
##May need to install XQuartz/X11
data( iris );
##postscript("irisFaces.eps")
faces( iris [,1:4],

labels=c(rep("ST",50),rep("VS",50), rep("VG",50)));

##dev.off();

#Chernoff Faces ex. 2:
USArrests<-read.table("USArrests.Data", header=TRUE);
faces(USArrests);


#Radial plot example R:
##Uncomment if used for the first time
##install.packages("plotrix");
```

```
library(plotrix);
radmat<-matrix(sample(1:4,40,replace=TRUE),nrow=4);
radial.plot(radmat,rp.type="l",
radial.pos=seq(0,20*pi/11.1,length.out=10),
label.pos=seq(0,20*pi/11.1,length.out=10),start=pi/2,
clockwise=TRUE, labels=2001:2010,radial.lim=c(0.2,4),
cex=1.5, font=2, lty=1, lwd=2);
legend(-5,4,c("Black","Red","Green","Blue"),col=1:4,lty=1);
```

```
##CLUSTERING TECHNIQUES

##Kmeans clustering
kmeans(x, centers, iter.max = 10, nstart = 1, algorithm = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen")
      , trace=FALSE)
# centers = either # clusters, or a set of initial (distinct) cluster centers
# iter.max = maximum # iterations allowed
# nstart = # random starts for K-means if centers is a number
# algorithm determines the choice of initial cluster centers

#Example: K-means clustering of 2-F Gaussian Data
require(graphics);
x <- rbind(matrix(rnorm(100, sd = 1), ncol = 2), matrix(rnorm(100, mean = 1, sd = 1), ncol = 2));

colnames(x) <- c("x", "y");
cl <- kmeans(x, 2);
colors<-rep("blue",nrow(x)); colors[cl$cluster==2]<-"red";
pchs<-rep(17,nrow(x)); pchs[cl$cluster==2]<-19;
plot(x, col = colors, pch=pchs);
txts<-c(rep("1",50), rep("2",50));
text(x,labels=txts);

#Example: K-means clustering on Iris data (pairwise)
data(iris);
mykms<-kmeans(iris[,1:4],centers=3);
pairs(iris[1:4], main = "", pch = 21, bg = c("red", "green3", "blue")[mykms$cluster]);


# Agglomerative Clustering

library(cluster);
agnes(x, diss = inherits(x, "dist"), metric = "euclidean", stand = FALSE, method = "average", par.method,
      keep.diss = n < 100, keep.data = !diss);
# metric = euclidean or manhattan
# method = complete , single or average .

#Example: Agglo clustering on iris



Duplicated, tail, and, rev
```