36-401 R Exercise

This document is an exercise to help you learn R, while incorporating some Exploratory Data Analysis (EDA).

We will be using the stat programming language R in this class to analyze data in class.

R is a freely available, multi-platform (Windows, Linux, Unix, MAC OS) and powerful program for analysis and graphics similar to S-Plus. It provides a programming language that is very flexible and can easily be extended by the user. It also allows the user to produce publication-quality graphics. The CMU computer labs have R installed already. If you are planning on using your own computer and need to install R, see p. 1 of your handout. Ask if you have problems.

You should carefully work through the sequence of commands on the following pages, and understand how R implements crucial calculations such as basic arithmetic and pulling elements and vectors out of matrices.

These could be done in "standard" R, or could be executed in the "Console" of R Studio. The latter is **strongly** recommended.

Using R as a calculator:

Try the following at the command line (>):

```
23+12
2*9 + 12
3/5
exp(2)          what does this represent?
2^2+2
2^(2+2)
```

We can evaluate an expression at the command line but the answer is only temporarily stored. We can't get it back unless we type in the expression again.

Assigning/Creating Objects:

We assign a value to a object/variable/word using the =. (You will also see <- used for this purpose.)

```
> x = 23+12
> y = 2*9 + 12
```

x and y are objects in the R workspace

```
> x
> y
```

If you lose track of the objects you've defined,

```
> ls()
```

If you want to remove an object,

```
> rm(x)
```

Variables can start with a letter, digit, or period. Can't be a number by itself.
Get in the habit of naming your variables to represent the data you're assigning.

```
(mean.crime.rate, no.of.students)
```

It is a nightmare trying to remember what you named something or keep the variables: x, x2, x.2 straight. Name your variables well.

Vectors & Matrices:

```
> x  =  c(3, 5, 1, 67)
> y = seq(1,4)
> z = c(seq(3,9,by=2),27)

> x[1]
> y[3]
> z[7]

> x+y
> x+z

> x*y
> y*z

> m1 = matrix(0,3,4)
> m2 = matrix(seq(1,16),4,4)
> m3 = matrix(rep(2,12),ncol=4)

> m1+m3
> m1+m2
> m1/m3
> m1*m2   ##multiplication element-by-element
> m1*m3
> m1%*%m3      ##matrix multiplication
> m1%*%m2

> m1[1]
> m2[1,]
> m3[,1]
> m1[2,3]
```

Asking Questions about your Objects:

Think of your object as a vector of values from a random variable.
Each entry is one realization of the random variable (or the answer of a person to one question). How long is the vector?

You might want to know which people answered yes (entry of 1) to the question.

```
> q1 = sample(c(0,1),10,replace=T)   ##creating 10 random yes/no values
> q1
> q1==1         ##returns T/F vector
> which(q1==1) ##returns the locations of the Trues
```

Or who smoked more than 2 times the past week?

```
> q2 = sample(c(0,1,2,3),15,replace=T) ##random values from 0 to 3
> q2
> q2>2
> which(q2>2)
```

Can use `table( )` and `summary( )` to get overall info about your objects.

Functions:

We will be using functions already written in R and writing our own (later).
A function is a collection of commands under one name.
Arguments are passed into the function; output (if any) is returned.

function.name = function(arg1, arg2, arg3…..){

    commands

    return(out1, out2, out3,….)
}

All arguments need to be entered for the function to work. Often there are default values already coded into the function. A common argument `na.rm/na.omit` tells R what to do if there's missing data in the vector (NA). Type `help(fxn.name)` to learn more.

```
> help(median)
> vec = sample(seq(1:30),30,replace=F)
> vec
> median(vec)
> vec2  = vec
> vec2[4] = NA
> median(vec2)
> median(vec2, na.rm=T)
```

Exploratory Data Analysis:

Let's look at some real data.
One of the nice things about R is its list of packages that include lots of sample datasets.

```
> library(MASS)  ##loads the MASS library into your workspace
> help(hills)    ## pulls up the help documentation for the hills dataset
> attach(hills)  ##creates variables out of each named column (dist, climb, time)
> dim(hills)     ##hills is a matrix; returns the # of rows (n), #of cols
> dist           ##checking the individual variables
> climb
> time
```

Often the best way to display EDA information about your variables is with plots and graphs. A well-constructed visual summary can quickly get across the points you're trying to make. These graphs should be supplemented with numerical summaries.

Numerical Summaries

How many observations do we have?

```
> nrow(hills)
```

What are some summary measures of each variable?

```
> summary(hills)
> var(dist)  ##see also sd(dist)
> var(climb)
> var(time)
```

What's the difference between the above and `var(hills)`?

Plotting and Graphics

Scan the handout for different types of graphs of `dist`, `climb`, and `time`.
Which types of plots give useful information? Which don't?
Which graphs belong with continuous variables? With categorical variables?

```
> hist(dist,col=2,main="Distribution of Race
Distance",xlab="Miles")
> hist(dist,breaks=10,col=2,main="Distribution of Race
Distance (Increasing the # of Bins)",xlab="Miles")
```

```
> boxplot(climb,col="blue",main="Distribution of Race
Elevation Gained",ylab="Feet")
> hist(time,col=5,main="Distribution of Race Record
Times",xlab="Minutes")
```

Describe these univariate distributions; what features do we notice?
Do any of them appear to be distributed normally?

Now looking at their bivariate relationships:

```
> plot(dist,climb,xlab="Miles",ylab="Feet",pch=16)
> title("Race Distance vs. Race Elevation")
```

Describe their relationship. Are there any short races with large elevation gains?

```
> plot(climb,time,xlab="Feet",ylab="Minutes",pch=16)
> title("Race Elevation vs. Race Record Time")
```

Describe this relationship. Are there any unusual races?

See help documentation on `plot` for information on how to change x-limits/y-limits using the `xlim,ylim` arguments. Also use `pch,cex,col` to control the characters.

Let's add a possible underlying regression function to the plot using the `abline` function. See `help(abline); abline` requires a slope and an intercept.

A hypothesized underlying regression function of E[time] = 15 + 0.01*climb

```
> abline(15, 0.01,lwd=2,col=2)
```

Does the line seem like a good fit?

What about E[time] = 12 + 0.02*climb? `> abline(12, 0.02,lwd=2,col=4)`

Let's add the "point of averages", the average climb and the average time, to the plot using `points(mean(climb),mean(time),cex=2,pch=16).` This point will be on the best regression line.

On your own, experiment with colors, labels, etc. Look at help(plot).

A good graph is a WELL-LABELED graph.

Reading Data into the R Workspace

We can read in an xls file or a dat file or a txt file using the read.table( ) command.
`read.table("filename.xls")`

If your file is in another directory, you need to put the whole pathname.
(Use front slashes.)
`stat.read = read.table("//Desktop/filename.xls")`

You can work with xls files or csv files (ease of use depends on platform).
If you read in an .csv file, you need to use `stat.read = read.table("/Desktop/statread.csv",sep=",") or read.csv("")`

~~Go to Blackboard and~~ download the `brainintel.txt` dataset under today's Lecture Notes. Save it in your R working directory.

```
> brainintel = read.table("brainintel.txt")
```

Make sure you have a matrix of 40 rows and 4 columns (`dim(brainintel)`). The 1st column is a measure of Full-Scale IQ; the 2nd column is weight (in pounds); the 3rd column is height (inches); and the 4th column is the brain size (total MRI pixels).

```
> FSIQ = brainintel[,1]        ##can just assign columns instead of using attach
> Weight = brainintel[,2]      ## have to assign when cols have no names
> Height = brainintel[,3]
> MRI = brainintel[,4]
```

With the time left at the end of today's class, use EDA to learn about the individual variables and their relationship to each other. What's going on with FSIQ and MRI?

Simulating Data

We can simulate observations from a specific distribution using parameter values of our choice using R. Because we know the truth about our simulated data, we can use simulations to check if our approach is valid. i.e. Does it return the answers it should?

To start:

`rnorm(n,mean,sd)`	generates random numbers from a normal distribution
`rt(n,df)`	generates random numbers from a t distribution

(also explore dnorm, pnorm, qnorm; we have functions for uniform, exponential, etc)

```
> x = rnorm(10,0,1)        ####NOTE: rnorm uses STDEV not VAR
> summary(x)
> mean(x)
> var(x)
> hist(x, main="10 Simulated Normal Observations)

> y = rnorm(100,0,1)
> summary(y)
> mean(y)
> var(y)
> hist(y, main="100 Simulated Normal Observations")
```

Compare the histograms; do they look normal?
Which simulation better represents the truth?

Now let's look at the t-distribution. We'll compare two distributions with different degrees of freedom: 5 and 37.

```
> x = rt(100,5)
> hist(x,main="100 Obs from t-Dist with df = 5")
> x = rt(100,37)
> hist(x,main="100 Obs from t-Dist with df = 37")
```

Compare them against each other and to the 100 normally distributed observations.

Simulating Data from a Regression Model

Let's say we have an underlying regression function of E[Y] = 42 – 3X and normally distributed error (mean zero, variance 25). Y = 42 – 3X + eps where eps~N(0,25).

We want to simulate data from this model in R:
Our X values are fixed; let's use: 0, 0.5, 1, 1.3, 6.7, 12

```
> x = c(0,0.5,1,1.3,6.7,12)  ##c() creates a list/vector
```

Now we find the Y by plugging in the X's into our regression function and adding noise

```
> y = 42-3*x + rnorm(6,0,5)
```
##we need 6 random errors with mean 0, stdev 5

Plot the (x,y) data; add the underlying regression function to the plot using `abline()`.

```
plot(x,y,pch=16)    #pch = 16 gives closed circles
abline(42,-3,lwd=2,col=2)
```
Now generate a second set of responses from a model with the same underlying function but where the errors are N(0,9). (using the same x's)

```
> y2 = 42-3*x + rnorm(6,0,3)
```

Add these points to your graph as "+"s.
```
> points(x,y2,pch=3)
```

Compare the different spreads.
Is it apparent that the both sets of errors are normally distributed?
Is it apparent that the first set of errors has a higher variance?

Our Regression Framework

Some useful functions for us: *type help(fxn.name) for more info*

mean(x)	finds the mean of a vector
var(x)	finds s^2 of a vector (sample variance)
sd(x)	finds the standard deviation, s, of a vector
sum(x)	sums up all the entries in a vector
sqrt(x)	finds the square root of a number/vector
lm(y~x)	finds a least squares/MLE estimated linear regression equation
anova(lm.obj)	analysis of variance for a linear regression model
pt(q,df)	finds the exact probability for a t-distribution (instead of the tables)

We'll use several more, but start looking at the help documentation for these.