36-401 Fall 2015 R Introduction

We will be using the stat programming language R in this class to analyze data in class. This documents contains some intro R materials, which also incorporates Exploratory Data Analysis (EDA). There is also a longer version of R introductory material available on the course website by Professor Rebecca Nugent. Other R resources are listed including an R reference card (pdf) that lists many of the common commands used in R.

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.

R-studio is a user-friendly environment to run R (and it is free) and it is easy to use. You are strongly encouraged to use R-studio (https://www.rstudio.com/) for this class. Here is a short and nice introductory material on R-studio. (http://dss.princeton.edu/training/RStudio101.pdf)

Open R on your lab computer (under Statistics); open a new text file (Word, Notepad,…), or you may go to File > New File > R Script (or File>New script for some operating system)

First, we'll learn how to change the R working directory and store our work.

Storing your R Work and Commands (p. 3-4)

How do we get help with R commands?

Working with the R Help pages (p. 5-13)

Using R as a calculator:

Try the following at the command line (>):

```
23+12
2*9 + 12
3/5
exp(2)
2^2+2
2^(2+2)
```

what does this represent?

Why are the last two answers different? How does R evaluate expressions?

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: p.16

We assign a value to a object/variable/word using either the <- sign or =.

```
> 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)
```

Seriously. 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: (p. 16-21)

```
> 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: (p. 23-24)

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 (p. 31-38)

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 regression line 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)
```

Good line?? Does it match the data?

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. Details can be found in the handout.

A good graph is a WELL-LABELED graph.

Reading Data into the R Workspace (p.25)

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("//pathname/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 `read.csv("")` or

```
stat.read<-read.table("/pathname/statread.csv",sep=",")
```

You can also read in data directly from a URL without downloading it

```
stat.read<-read.table("http://url/statread.csv",sep=",")
```

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.

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.

Basic R Markdown (not required for the class, but recommended)

R Markdown is an extension of R as a tool to create documents that embed your code and the calculations it produces, in ordinary text, which can also be formatted, contain figures and equations, etc.

To use it, you will need to install R package

```
>install.packages("rmarkdown")
>install.packages("knitr")
```

Here is quick tour of how R Markdown works (http://rmarkdown.rstudio.com/).

Let's create a short R Markdown file:

1. In R studio: File > New > R Markdown. You can choose the output format for your document (Html, pdf, or word). This will open a new *.Rmd* file.
2. The created file contains some already-written lines, including how to include R code with its output and plots. You may replace them using your own code or text.
3. Now click the Knite button, a new html/pdf/word file will be created.

The title goes first in the document, and specifies the type of output.

```
---
title: "Sample Document"
output: html_document
---
```

Paragraphs of texts can be included directly in the file. You may also embed calculations between texts by inline R code. Here is one example

```
The total number of students is  `r 89+90`
```

The output will be “The total number of students is 179”.

Big chunk of R code is included by

```
```{r}  your R code goes here  ```
```

By default, the code will run and the output will be printed in the knitted file. You have the option not to include the output by using `echo=FALSE` option.

```
```{r, echo=FALSE} your R code goes here ```
```

Or if you just want to include code but not evaluating it, you can use `eval=FALSE` option

```
```{r, eval=FALSE} your R code goes here ```
```

The link of R Markdown Basics (http://rmarkdown.rstudio.com/authoring_basics.html) provides some guideline on how to include other objects like Header, List, Table, Latex equation etc.

Please also see R Markdown Cheat for a really nice introduction. (https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf)
```