STAT 515 - R Templates
Last Updated 8/15/2016 - Designed for use with McClave and Sincich's 13th Edition

If you get stuck and need to contact your instructor, please include a copy of the data set you are using and the code (or menu options) that you are trying.

"The Basics of R" section below is designed to be fairly easy to go through. A .pdf copy of that section and of the output can be downloaded and printed out to take notes on. This entire page is designed to be worked through while you are on a computer with both R and a copy of this page open.

The data sets for the text book can be found at: https://www.pearsonhighered.com/mathstatsresources/#author=M in the Statistics, 13/e section. They can be downloaded in TXT format to your computer in a zip file, and then be read in from there. When reading them in, the slashes must always be / forward slashes in your R code.

The Basics of R:

R, like the commercial S-Plus, is based on the programming language S. It can be downloaded for from www.r-project.org [At home, after reading the instructions!, choose: CRAN, then a mirror site in the US, then "Download R for Windows", then "install R for the first time", then "Download R 3.3.1 for Windows" -- there is also a link for the Mac or LINUX]. The CRAN page also has links to FAQs and other information about R.

One of the nice things about R (besides the fact that it's powerful and free) is that you can save everything you've done after you've finished each time. If you are working on a machine you have administrator access to it will ask you if you want to save the workspace when you quit the program (choose yes, and all the entered data and functions will be there next time you start it up). If you are on a networked machine where you don't have administrator access, or if you want to be able to access the saved data from several machines then you can load and save the `.Rdata` file on a disk or pen-drive (use the `Load Workspace` and `Save Workspace` options under the `File` menu).

At the heart of R are the various objects that you enter. At any time (when you are in R) you can type in the function `objects()` to see which ones you have entered previously. Presumably there aren't any there right now if you've never used R before.. R also has many built in objects, most of which are functions. Because these objects are always there, it will not list them when you type `objects()`.

A list of the various functions for R can be found in the manuals that are installed with the program. The various manuals can be found under the `Manuals` submenu of the `Help` menu. In particular, the *R Reference Manual* lists not only all the functions in the base package of R, but also some of those in the supplementary packages as well. Because this manual is nearly 1000 pages long, you probably shouldn't print out a copy!!!! A large number of supplementary packages (functions that others wrote and decided to make available to the public) can be found under the `Packages` menu by choosing `Load Package`. One popular one is the `MASS` library that is designed to

accompany Venables and Ripley's *Modern Applied Statistics with S*. It can also be loaded in by typing `library(MASS)`.

Once you know the name of a function, you can get help about it simply by typing `help(functionname)`. In R, parentheses indicate either mathematical grouping, like they usually do, or that you are applying a function. Brackets `[ ]` indicate that you are finding an element in a string.

The following demonstrates some of the basic ideas of how R syntax works. Simply read along, and whenever you see something in `type-writer font` simply cut and paste that into the R window and hit return.

```
x<-c(5,4,3,2)
```

The `<-` is the command that assigns what ever is on the left to the name that is on the right. The `c` indicates that an array or vector is going to be entered. Simply typing `x` now will return those four values. Typing

```
x[3]
```

will return the third value,

```
mean(x)
```

will return the mean, and

```
objects()
```

will show you that you now have an object called `x`. Other functions to try include `sd(x)`, `var(x)`, `median(x)`, and `summary(x)`.

R also has a variety of graphical functions built in.

```
hist(x)
```

will construct a histogram. Unfortunately it uses the right end-point rule by default and fudges a few things sometimes! Checking `help(hist)` will reveal that "A numerical tolerance of 1e-7 times the median bin size is applied when counting entries on the edges of bins."

We could get the correct histogram by insuring that the screen is set up wide enough and that we told it to use the left endpoint rule.

```
hist(x,breaks=seq(1,7,by=1),right=F)
```

Try it with 2,5 and see that the fudging messes things up!

If this was discrete data, the way to get the axis labeled correctly would be

```
hist(x,breaks=seq(1.5,5.5,by=1),right=F)
```

It also very easy to simulate data sets in R. For example, the following code will make 1,000 fake IQ scores from a normal distribution with mean 100 and standard deviation 15. (If you run this yourself you will get slightly different values since you will have your own random sample of 1,000 IQs!)

```
IQ<-rnorm(1000,mean=100,sd=15)
IQ
mean(IQ)
sd(IQ)
hist(IQ,right=F)
```

Besides just getting the result of a function, we can store the results somewhere. In the code above we put the values from the function rnorm into IQ. As another example,

```
medianofIQ<-median(IQ)
```

will store the median in `medianofIQ`.

We can use some built in function in R to check out how the empirical rule and Chebychev's inequality work for this data.

```
sum((IQ>=85)&(IQ<115))/1000
sum((IQ>=70)&(IQ<130))/1000
sum((IQ>=55)&(IQ<145))/1000
```

If we wanted to try this on a skewed data set, we could use something we will see later called the chi-squared distribution.

```
Newdat<-rchisq(1000,df=2)
hist(Newdat)
mean(Newdat)
median(Newdat)
```

We can check the percents again using:

```
sum((Newdat>=mean(Newdat)-sd(Newdat))&(Newdat< mean(Newdat)+sd(Newdat)))/1000
sum((Newdat>=mean(Newdat)-2*sd(Newdat))&(Newdat<
mean(Newdat)+2*sd(Newdat)))/1000
sum((Newdat>=mean(Newdat)-3*sd(Newdat))&(Newdat<
mean(Newdat)+3*sd(Newdat)))/1000
```

Percentiles can be gotten using the `quantile` function. For example:

```
quantile(IQ,.5)
quantile(IQ,.25)
quantile(IQ,.75)
```

will give the median, 1st quartile (25th-percentile), and 3rd quartile (75th-percentile). `help(quantile)` explains the nine different ways it has built in to calculate the quantiles. Using the option `type=6` gives the same result as Minitab and SAS (e.g. `quantile(IQ,.75,type=6)`).

R can also be used to write your own functions. For example, if you wanted a function that returned the inter-quartile range we could write:

```
IQRfun<-function(datavector,type=7){
Q3<-quantile(datavector,.75,type=type)
Q1<-quantile(datavector,.25,type=type)
as.numeric(Q3-Q1)}
```

After copying that function over, try `IQRfun(IQ)`. You can compare it to the built in function `IQR(IQ)` that didn't exist in previous editions of the program. The function above allows for the quantile type to change too, though.

The five number summary (plus the mean) can be gotten by using:

```
summary(IQ)
```

Unfortunately it isn't set up to easily allow you to change the type.

A boxplot is a graph based on the five-number summary:

```
boxplot(IQ)
boxplot(Newdat)
```

R is able to read data in both from files and from appropriately formatted web-pages. The following command will read in the data set contained at http://people.stat.sc.edu/habing/515/data/examp9p4.txt that is used in example 9.4 in the text.

```
e9p4 <-
read.table("http://people.stat.sc.edu/habing/515/data/examp9p4.txt",header=TRU
E)
```

Typing `e9p4` will show you the entire data set, and `attributes(e9p4)` will give you a list of the various names associated with this data set.

Try the following to see if you can tell what they are returning:

```
e9p4[,2]

e9p4[2,]

e9p4$value
```

The first one is giving the second column of the data set - the notation inside of the square brackets is always of the form "[which rows, which columns]". `e9p4[,2]` is saying to give the second column, and since it doesn't specify, all of the rows. The second one, `e9p4[2,]` is therefore giving the econd row of the data set. The third one, `e9p4$value`, uses one of the names you saw when you entered `attributes(e9p4)` to specify the name of a column. When using names in this way, it's always after a dollar sign.

If you wanted to, you could put the two variables in their own vectors using either:

```
group<-e9p4$group
value<-e9p4$value

group<-e9p4[,1]
value<-e9p4[,2]
```

NOTE: Many functions in R will not work directly on data you read in using `read.table`. You need to either specify which columns you wanted to use (like the above few lines) or if all of the columns are data, you could use`as.matrix(name.you.used.for.the.table)`.

If you want to use all of the data

We could also make subsets of the data. For example, try:

```
new<-e9p4[e9p4[,1]=='new',2]
new
stand<-e9p4[e9p4[,1]=='stand',2]
stand
```

What does these seem to return? Why?

The first part in the square brackets says which rows, all the ones where the second column has the value 1 (notice the double equal sign is used there). The second part in the square brackets are the columns, and here they are listed as a vector of names. (Check the original data set to make sure it was done correctly!).

Boxplots of the two could be made using either compare:

```
plot(group,value)
```
or
```
boxplot(value~group)
```

The ~ will be used later on when designing statistical models.

It is a lot easier to compare the median and other percentiles of distributions using boxplots than it is with histograms:

```
par(mfrow=c(2,1))
hist(new,right=F,breaks=seq(60,90,by=5))
hist(stand,right=F,breaks=seq(60,90,by=5))
par(mfrow=c(1,1))
```

Finally, one more plotting example for our simulated normal data set:

```
hist(IQ,freq=F,xlim=c(45,150),ylim=c(0,.04),right=F)
par(new=T)
plot(density(IQ),xlim=c(45,150),ylim=c(0,.04),xlab="",ylab="",main="")
```