Introduction to R

1. Installing R
2. First R
3. Getting Your Data Into R
4. Using Files of Commands
5. Factors and Tables
6. Getting Help in R
7. Simple Linear Regression
8. Plots
9. Logical Variables and Subsetting Your Data
10. Multiple Regression
11. sapply, tapply, and cut
12. Package Installation
13. More on Input and Output

1. Installing R

Instructions for installing R are at:

`http://it.chicagobooth.edu/research-computing/pdfs/R_Download-Install.pdf`

Quick instructions are as follows:

- Go to http://www.r-project.org/.
- On the left, click on **CRAN**, under `Download, Packages`.
- choose a mirror (I guess something geographically close).
- Click on `Download R for Windows` (or for `MacOSX,Linux`).
- click on `base`.
- click on the download for your platform (e.g. `Download R x.x.x for Windows`).

Note: http://www.r-project.org/ will take you to the main R page.
To go directly to the R download page you can use
http://www.cran.r-project.org/.
After than it should be pretty standard.
You will have an R icon on your desktop and all you have to do is (double) click on that.

2. First R

Get into R.

At the prompt (>) type x=2 (then hit return):

```
> x=2
```

Then,

```
> x
```

Now try,

```
> y=3
> z=x+y
> z
> w=sqrt(x)
> w
> ls()
> rm(x)
> ls()
```

Things like `x` above are *variables*. We can store information in variables and then operate on them (e.g. add them).

Something like `sqrt` is a *function*. Above, the function `sqrt` takes the *argument* `x` and returns the square root of the value in `x` which we put into `w`.

`ls` is also a *function*, but we give it no arguments. We do not keep what it returns, so the results are just printed to the screen. `ls` shows us all the variables we have created. What does `rm` do?

Try:

```
> x=2
> x*x
[1] 4
> x^2
[1] 4
> log(x)
[1] 0.6931472
```

Note you can nest (or *compose*) functions:

```
> x=2
> y=3

> w = log(sqrt(x+y^2))
```

Try:

```
> x=c(5,3,7)
> x
[1] 5 3 7
> x[1]
[1] 5
> x[3]
[1] 7
> x[1:2]
[1] 5 3
> mean(x)
[1] 5
```

Here the variable `x` is a *vector* of numbers instead of just one number as is the case with `x=2`.

In `R`, variables can be numbers, strings, vectors, matrices and all kinds of very useful things!

Arithmetic works with vectors:

```
> x=c(1,2,3)
> y=c(4,5,6)
> w=x+y
> w
[1] 5 7 9
> z = 2*x+y
> z
[1]  6  9 12
> v=log(x)
> v
[1] 0.0000000 0.6931472 1.0986123
```

Beside numbers, we work with character stings a lot:

```
> x= "Chicago Black Hawks"
> x
```

Anything after # is ignored by R so you can use it to make comments:

```
> x=c("rob","is","a" ,"leafs","fan") #a vector of strings
> x
```

You can use the `paste` function to put character string together

```
> y = paste("go","Hawks")
> y
> z = paste("go","Hawks",sep="--")
> z
```

You can also use indexing for assignment.
That is, you can change just a part of a vector.
Try:

```
> x=c(1,2,3,4)
> x
> x[c(1,4)] = c(10,11)
> x
> x[2:4] = 5:7
> x
```

Note: to clear all the variables you can use

```
> x=2
> y=c(4,7,9)
> ls()
> rm(list=ls())

> ls()
```

To get out of R use the q function:

```
> q()
```

Note that q is a function with no arguments here and you need the ()!

3. Getting Your Data Into R

First make a new directory (or folder) on your machine to work in.

Then download the file susedcars.csv to your directory (from the course data webpage).

Now get into `R`.

Now you have to *set the working directory*.

R has to know where to look for files to read in and where to write stuff to.

In Windows you can also use the menu item /File/Change dir.
In Rstudio you can use /Session/Set Working Directory/Choose Directory.

To check what the working directory is set to use:

```
> getwd()
```

To read your data in try:

```
> cdat = read.csv("susedcars.csv")
> ls()
```

The function will read in the data and store it in the variable `cdat`. Of course, you can use any name you want instead of `cdat`. Some people like long variable names and some people like short ones.

The cdat is a *data frame*, which is a special format `R` uses to store data sets.

There are lots of ways we can look at the data in a data frame.

To get a quick visual look at the date try:

```
> edit(cdat)
```

To summarize the data try:

```
> summary(cdat)
```

To see the number of rows (observations) and columns (variables) try:

```
> dim(cdat)
```

To see the names of the variables try:

```
> names(cdat)
```

You can think of a data frame as being like a matrix in that it has rows (corresponding to the observations) and columns (corresponding to the variables).

You can use indexing to access the numbers in the element frame:

```
> cdat[2,4]
[1] 46883
```

The car in the second observation has mileage (the 4th column) 46,883.

We can pull off a range. If you don't specify the row or colum, all are used.
To get the first 5 rows try:

```
> cdat[1:5,] #the first 5 rows
  price trim isOneOwner mileage year  color displacement
1 43995  550          f   36858 2008 Silver          5.5
2 44995  550          f   46883 2012  Black          4.6
3 25999  550          f  108759 2007  White          5.5
4 33880  550          f   35187 2007  Black          5.5
5 34895  550          f   48153 2007  Black          5.5
```

Two other ways to get a quick look at the data are:

```
> head(cdat) #print out the first few rows
> tail(cdat) #print out the last few rows
```

4. Using Files of Commands

Obviously, typing in commands the the R prompt $>$ will get old fast.

What we do is put our commands in a simple ascii file and then cut and paste them into R.

So, if you wanted to read in the used cars data and get a quick summary, you get create an ascii file with the following in it:

```
#read in and print summary of used cars data
cdat=read.csv("susedcars.csv")
print(summary(cdat))
```

Then you just cut and paste the above lines into the R prompt window. Try it.

You can do quite sophisticated work using this simple mechanism.

The ascii file stores your " R code" so that you have a record of what you did!!!

You can also run all the commands in a file using the function `source`.

Suppose the ascii file with the commands

```
#read in and print summary of used cars data
cdat=read.csv("susedcars.csv")
print(summary(cdat))
cat("dim of cdat is ",dim(cdat),"\n") #\n is a carriage return
```

is in your working directory and called "do.R".

Then at the prompt you just type

```
> source("do.R")
```

If your are using Windows, check out the menu
File/New Script.

5. Factors and Tables

A basic distinction is statistics is between numeric variables and categorical variables.

A numeric variable has units and measure "how much", e.g I have 100 *dollars*.

A categorical variable just says one of several states or *categories* may obtain.
e.g I may have gender of `male` or `female`.

The way you analyze a categorical variable is often very different from the way you analyze a numeric variable.

The `R` data frame keeps track of which variables are numeric and which are categorical. `R` calls categorical variables *factors* and the different possible categories are called *levels*.

Try:

```
x = c(1,1,2,2,4,4,4)
summary(x)
y = as.factor(x)
summary(y)
```

`x` is a numeric variable so it is summarized by its median and quantiles.

The `R` function `as.factor` converts a variable to a factor.

`y` is a catergorical variable, so it is summarized by the counts for the different categories.

Now try,

```
levels(y) = c("north","south","west")
summary(y)
```

`R` uses the term *levels* to refer to the names of the categories.

Now try

```
cdat = read.csv("susedcars.csv")
summary(cdat)
```

Notice the R guessed that some of the variables are numeric and that some are categorical. Numeric variables are summarized by quantiles and categorical variables are summarized by counts.

Try

```
plot(cdat$mileage)
```

Now try

```
cdat$year = as.factor(cdat$year) #convert year to a factor
plot(cdat$mileage)
```

The way R plots a variable depends on whether it is numeric or categorical.
Many important statistical methods in R will (quite reasonably) behave differently depending on whether a variable is numeric or categorical.

Try

```
table(cdat$color) #table of category counts
table(cdat$color)/nrow(cdat) #table of category percentages
```

We often look at pairs of categorical variables by making *two way* tables.

Try

```
tb = table(cdat$color,cdat$isOneOwner) #two way table of counts
tb
tb/nrow(cdat) #two way table of percentages
tbp = round(tb/nrow(cdat),2) #round to two digits.
tbp
```

6. Getting Help in R

Suppose I want help on the function `mean`.

```
> help(mean)
> ?mean
```

will both get you help on the function `mean`.

This is very easy, but, as you can see, the help is not always the easiest thing to read.

To get a webpage of help, try

```
> help.start()
```

You can click on packages and then base to get info on all R functions.

You can also try

```
> help.search("mean")
> apropos("mean")
```

to search R for functions related to the term "mean".

7. Simple Linear Regression

Get the simple Housedata.csv from the webpage and put it in your working directory.

Read in the data:

```
hd = read.csv("Housedata.csv")
dim(hd)
hd
```

There are just 15 observations giving the Size (thousands of square feet) and Price (thousands of dollars) of 15 houses.

To run the simple linear regression of Price on Size and store the results in the variable hlm:

```
hlm = lm(Price~Size,hd)
```

Remember "hlm" is just a variable name I made up. You can call the results whatever you want.

To get the usual "regression output" use

```
summary(hlm)
```

The R function `lm` returns a *list* with all kinds of information about the regression. To see all the information in the list try

```
names(hlm)
```

For example `hlm$coefficients` is the vector of regression coefficients.

```
hlm$coeff  #just have to give enough of the name to identify it
```

You can also store the results of `summary` is a variable and this gives you a whole other set of info on the regression:

```
shlm = summary(hlm)
names(shlm)
```

To get the estimate of σ and R^2 we can use

```
shlm$sigma
shlm$r.squared
```

8. Plots

Get the simple housedata:

```
hd = read.csv("Housedata.csv")
attach(hd)
```

Let's plot Size vs. Price:

```
plot(Size,Price)
```

The plot function has many optional arguments that we can use to change the appearance of the plot.

```
plot(Size,Price,xlab="size", ylab="price",col="blue",pch=16,cex=1.2)
```

xlab,ylab: axis labels.
col: color of plotted points
pch: plot symbol

cex: size of plotted symbol (default is 1).

You can add lines, points, and text to a plot.
Let's plot the y=Price vs the fitted values and then add the line with intercept 0 and slope 1.

```
houselm = lm(Price~Size,hd)
plot(Price,houselm$fitted,xlab="price",ylab="fitted values",col="blue",pch=20)
abline(a=0,b=1,col="red",lwd=2) #lwd: line width
```

You can change the appearance of the line with the argument `lty`

```
abline(v=mean(Price),col="green",lwd=3,lty=2) #v= gives a vertical line.
```

You can add points to a plot:

```
points(Price[1:10],houselm$fitted[1:10],pch=2,col="black") #redraw first 10.
```

You can add text to a plot.

```
n=nrow(hd)
plot(Size,Price,type="n") #type="n" sets of the plot but does not add the point
text(Size,Price,paste("p",1:n,sep=""),col="red") #add "p1", "p2" ... at points.
text(1.0,140,"house plot")
```

Let's run the house regression and add the least squared line to the plot. Let's also put the y vs fitted plot besid that plot, in a separate plot.

```
lmh = lm(Price~Size,hd) #run regression
par(mfrow=c(1,2)) #set up a 1 x 2 matrix of plots
plot(Size,Price,col="blue",cex=1.2) #plot Size vs. Price
abline(lmh$coef,lwd=2,col="red") #add regression line
plot(Price,lmh$fitted,ylab="fitted",col="blue",cex=1.2) # y vs yhat
abline(0,1,lw=2,col="red") #add line
```

Now let's use the cars data again.

```
cd = read.csv("susedcars.csv")
```

To get the histogram of the prices we use `hist`

```
hist(cd$price,xlab="used car price",main="car data",nclass=20,col="blue")
#nclass: number of bins.
```

Boxplots are are another hand way to display a numeric variable:

```
boxplot(cd$price)
```

What happens if we plot a numeric variable agains a categorical variable??

```
plot(cd$trim,cd$price)
```

This does a separate boxplot of the cars prices for the subsets of data defined by the levels of trim. We can easily see that prices are higher for trim 550!

Another way to get the same plot is:

```
boxplot(price~trim,cd)
```

You can use color in a scatterplot to get simple display involving three variables. Let's use the midcity housing data:

```
mchd = read.csv("midcity.csv")
attach(mchd)
plot(SqFt,Price,col=Nbhd,pch=20) #indicate neighborhood with color
legend("topleft",legend=c("N1","N2","N3"),col=1:3,,pch=rep(20,3))
```

9. Logical Variables and Subsetting Your Data

Another kind of variable which is very useful a logical variable.

```
> 1>2
> 2>1
> 2==1
> 2==2
> 2=>2
```

There are only two possible values, `TRUE` or `FALSE`.

What is really useful are vectors of logical values.

```
> x=1:10
> x<6
```

We can use this to pick off interesting subsets of our data.

```
> x=1:10
> x
> y=sample(1:10)# randomly permute 1 to 10
> y
> yg5 = y>5
> yg5
> xx = x[yg5]
> xx
```

Or,

```
> x=1:10
> y=sample(1:10)# randomly permute 1 to 10
> xx = x[y>5]
> x;y;xx
```

Let's to a separate histogram for the house price in each of the three neighborhoods in the midcity data:

```
par(mfrow=c(1,3)) #set up three plots in a 1 x 3 arrangement
mc = read.csv("midcity.csv")
attach(mc)
hist(Price[Nbhd==1],xlim=range(Price))
hist(Price[Nbhd==2],xlim=range(Price))
hist(Price[Nbhd==3],xlim=range(Price))
```

10. Multiple Regression

Let's use the midcity housing data.

How do we regress Price on SqFt *and* Bedrooms?

```
mc = read.csv("midcity.csv")
lmmc = lm(Price~SqFt+Bedrooms,mc)
print(summary(lmmc))
```

If you run a multiple regression where some of the x's are factors, R will automatically dummy up the factors:

```
mc = read.csv("midcity.csv")
summary(mc)
mc$Price = mc$Price/1000 #thousands of dollars
mc$SqFt = mc$SqFt/1000 #thousands of square feet.
mc$Nbhd = as.factor(mc$Nbhd) #was coded as numeric when read in.
summary(mc)
lmmc = lm(Price~SqFt+Nbhd,mc)
print(summary(lmmc))
plot(mc$SqFt,lmmc$fitted)
```

In the multiple regression ouput "Nbhd2" is the dummy for neighborhood 2 and "Nbhd3" is the dummy for neighborhood 3.

If you want to regress one variable in a data frame on all the other variables you can do it succinctly:

```
#let's get rid of the variable "Home" and "Offers" ,the first and third columns
mc = mc[,-c(1,3)] #drop first and third columns
print(summary(mc))
lmmc = lm(Price~.,mc)
print(summary(lmmc))
```

11. sapply, tapply, and cut

Let' use the mutual funds data.

```
mfd = read.csv("mfunds.csv")
print(head(mfd))
```

Try:

```
> rmns = sapply(mfd,mean)
> mns
> mean(mfd$drefus)
```

`sapply` will "apply" the given function (in this case `mean`) to each column (variable) and return a vector of results.

To plot the returns means vs. the return standard deviations:

```
> plot(sapply(mfd,sd),sapply(mfd,mean),xlab="sd of
return",ylab="mean return")
```

Can we use the fund labels instead of points?

```
mv = sapply(mfd,mean)
sv = sapply(mfd,sd)
plot(sv,mv,xlab="sd of return",ylab="mean return",type="n")
#type="n" means points not plotted
text(sv,mv,names(mfd),col="blue")
```

Now lets use the midcity housing data to learn about `tapply`.

```
mc = read.csv("midcity.csv")
attach(mc)
mean(Price[Nbhd==1])
tapply(Price,Nbhd,mean)
```

`tapply` broke the data up into subsets ("a table") using Nbhd and then applied the function `mean` to the values of Price within each subset.

How does "brick affect price" ?

```
tapply(Price,Brick,mean)/1000
```

We can use more than one factor to subgroup our data. If we want the average house price broken down by neighborhood and brickness we can use

```
tblm = tapply(mc$Price,list(mc$Nbhd,mc$Brick),mean)
tblm
```

This gives us the average price of a house broken down by neighborhood and brickness.

A very common thing to do is turn a numeric variable into a categorical variable by making ranges of the variable categories.

The R function cut does this for us.

```
x=1:10
xf = cut(x,breaks=c(0,3,8,10))
print(x)
print(xf)
table(xf)
levels(xf) = c("low","medium","high")
table(xf)
```

Let's use cut to see how Price is related to size in the midcity data.

```
mc = read.csv("midcity.csv")
price = mc$Price/1000
size = mc$SqFt/1000
sizeq = quantile(size,probs=c(.25,.5,.75))
sizeq
sizef = cut(size,c(0,sizeq,max(size)))
par(mfrow=c(1,2))
plot(tapply(size,sizef,mean),tapply(price,sizef,mean),
    xlab="average size",ylab="average price")
plot(sizef,price)
```

12. Package Installation

A key advantage to R is the many "packages" that have been written for R.
These are add-ons the the base system that give you additional functionality.
Often the latest research is available in these packages and, of course, the price is zero.

Installing a package is very easy.

To install a package, you just need to know its name.
For example, the name of the package providing data sets for the book "Introduction to Statistical Learning with R" by James et. al. is ISLR.

To install it:

```
> install.packages("ISLR")
```

After a package is installed you have to load it into your session everytime you use it. For the ISLR package try,

```
library("ISLR")
attach(Default)
ls(pos=2)
summary(Default)
```

Go to http://www.cran.r-project.org/, then click on "packages" (on the left), then click on "Table of available packages, sorted by date of publication ".

Wow!!!!

13. More on Input and Output

We have seen that we can use `read.csv` to read in data from a file in csv format.

We can also write a file in csv format:

```
#make a small ‘‘data set’’
set.seed(99)
mydat = data.frame(x=1:5,y=sample(1:5,5))
print(mydat)

#write it to file
write.csv(mydat,file="mydat.csv",row.names=FALSE)
temp = read.csv("mydat.csv")
print(temp)
```