

Learning R

Matthew Krachey

CHAPTER 1

Session # 1

1. Introduction

R (<http://www.r-project.org>) is a free, open source statistical software package based on the S programming language. In contrast to SAS, R is a scripted language. That is, you can make decisions on the fly, and manipulate them in real time, in contrast to SAS. R also has a great graphical package, which will be the subject of later talks.

R is based on the S language, which has a cousin S-plus, which has some differences, but is essentially a version of R that you pay for. Industry often does, but I can't think of many researchers that do anymore.

2. Scripts

Go to file: new script. This generates a script file. Select text in the script file, hit ctrl-R, and it will send the text to R. This is a helpful way to keep your code organized, work, modify code, and end up with what you want.

Popular alternatives to the script editor include Tinn-R, ESS, and my preferred, Eclipse using the StatET package.

3. Objects

R treats everything as objects. Objects can be numbers, text, n-dimensional matrices, vectors, programs, output, etc. For example, suppose we wanted to assign the string of integers from 1 to 10 to the variable x. Then

```
> x = 1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
```

First, note that here the = is playing the role of the **assignment** operator, it is taking what is on the right side of the sign, and shoving it into what is on the left side. Note that if I want to see if two variables are equal, I **cannot** use the symbol = to do this comparison, if I do, I will actually make them equal to each other! (In this case, I need to use ==)

The : symbol only allows us to use integers. Suppose we wanted a string of numbers from 1 to 10 by .5, or of length 22:

```
> x1 = seq(1, 10, 0.5)
> x1
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
[16] 8.5 9.0 9.5 10.0
> x1 = seq(1, 10, length = 22)
> x1
[1] 1.000000 1.428571 1.857143 2.285714 2.714286 3.142857 3.571429
[8] 4.000000 4.428571 4.857143 5.285714 5.714286 6.142857 6.571429
```

```

[15] 7.000000 7.428571 7.857143 8.285714 8.714286 9.142857 9.571429
[22] 10.000000
> x == x1
[1] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

```

Note, by using the same name `x1`, the contents were overwritten. Also note that I wouldn't normally want to see if two vectors of unequal size are equal, but it's doable. It's important to be wary of this, and name things in a sensible, systematic fashion.

Notice that we have above used a function: `seq`. If you apply `help(seq)` or `?seq` you can get more information on how to use this, or any other function. Functions are part of what give R a great deal of power and flexibility. Also, if you think that a word should be involved in a function (i.e. autocorrelation) but cannot remember the function, you can play with `help.search("search-terms")`. There are too many functions to name that are built in to R. Later, we will learn how to add functions written by others, and finally write our own functions.

We can apply simple math functions to these objects with ease:

```

> x1 - 2
[1] -1.0000000 -0.5714286 -0.1428571 0.2857143 0.7142857 1.1428571
[7] 1.5714286 2.0000000 2.4285714 2.8571429 3.2857143 3.7142857
[13] 4.1428571 4.5714286 5.0000000 5.4285714 5.8571429 6.2857143
[19] 6.7142857 7.1428571 7.5714286 8.0000000
> x1^2
[1] 1.000000 2.040816 3.448980 5.224490 7.367347 9.877551
[7] 12.755102 16.000000 19.612245 23.591837 27.938776 32.653061
[13] 37.734694 43.183673 49.000000 55.183673 61.734694 68.653061
[19] 75.938776 83.591837 91.612245 100.000000
> x1/3
[1] 0.3333333 0.4761905 0.6190476 0.7619048 0.9047619 1.0476190 1.1904762
[8] 1.3333333 1.4761905 1.6190476 1.7619048 1.9047619 2.0476190 2.1904762
[15] 2.3333333 2.4761905 2.6190476 2.7619048 2.9047619 3.0476190 3.1904762
[22] 3.3333333
> x1^2 - x1/3
[1] 0.6666667 1.5646259 2.8299320 4.4625850 6.4625850 8.8299320
[7] 11.5646259 14.6666667 18.1360544 21.9727891 26.1768707 30.7482993
[13] 35.6870748 40.9931973 46.6666667 52.7074830 59.1156463 65.8911565
[19] 73.0340136 80.5442177 88.4217687 96.6666667
> x1/mean(x1)
[1] 0.1818182 0.2597403 0.3376623 0.4155844 0.4935065 0.5714286 0.6493506
[8] 0.7272727 0.8051948 0.8831169 0.9610390 1.0389610 1.1168831 1.1948052
[15] 1.2727273 1.3506494 1.4285714 1.5064935 1.5844156 1.6623377 1.7402597
[22] 1.8181818
> x1/max(x1)
[1] 0.1000000 0.1428571 0.1857143 0.2285714 0.2714286 0.3142857 0.3571429
[8] 0.4000000 0.4428571 0.4857143 0.5285714 0.5714286 0.6142857 0.6571429
[15] 0.7000000 0.7428571 0.7857143 0.8285714 0.8714286 0.9142857 0.9571429
[22] 1.0000000

```

```
> summary(x1)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.00   3.25   5.50   5.50   7.75   10.00
```

There is a number of important functions to be aware of in order to navigate around the R environment. Knowing what your working directory is can be important. We can get that by using the `getwd()` function.

```
> getwd()
[1] "C:/Users/Matthew/workspace/Tutorial"
```

Note, if you are using Windows, these direction of the slashes returned is opposite of the direction that windows uses. So if you copy a path from a "Get Info" in Windows, you will have to change the direction of the slashes. To set a different working directory, use the function `setwd("path-to-directory")`. You need the quotes for it to work. You can also use the menus in R to change the directory, just use `file > change dir`. I recommend using `setwd("path-to-directory")` at the header of any R script that you are likely to use often, it saves time over going through the menus.

4. Importing Data

This is possibly the hardest thing about using R, and once you have conquered this hurdle, you are well on your way to learning R. My best advice for importing data into R, **use comma-separated values: .csv**. R can handle Excel, SAS, Arc, tab-delimited and others, but by far the easiest approach is to use .csv files. You can easily convert any SAS (through Export Wizard) or Excel (through "Save As" csv) dataset into csv easily, and it just makes life a whole lot easier if that option is available.

The syntax for the import line, using .csv is:
`read.csv(file="path-to-file.csv",header=T)`

And, nicely, writing such a file is as easy as:
`write.csv(object,file="desired-path.csv")`

Note that when you put in the path, if you have already set the path you want, you can just put "filename.csv" instead of a full path.

5. Simulating Data

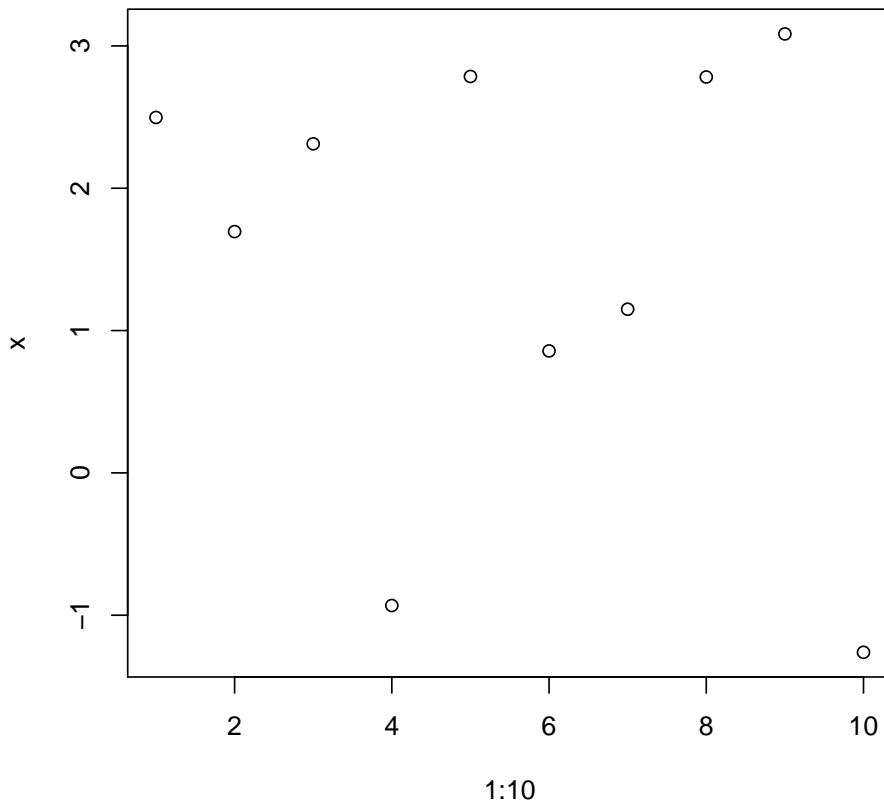
One strong advantage of R over most other packages is the ease at which one can simulate data. For example,

```
> x = rnorm(10, mean = 1, sd = 2)
> x
 [1] 0.19525195  3.47619508  2.11876482  1.00866468  1.49699015 -0.05216268
 [7] 0.50607338  2.87868709  0.07045554  1.05981765
> mean(x)
[1] 1.275874
> var(x)
[1] 1.473879
> min(x)
[1] -0.05216268
> max(x)
```

```

[1] 3.476195
> x[x > 0]
[1] 0.19525195 3.47619508 2.11876482 1.00866468 1.49699015 0.50607338 2.87868709
[8] 0.07045554 1.05981765
> plot(1:10, x)

```



this generates a sample of 10 normally distributed observations with mean 1 and a standard deviation of 2. Likewise, we can use `rbeta`, `rpois`, `rbinom`, `rgamma` etc to generate variables of those types. I recommend using `?rgamma` etc to see the input requirements for those functions.

5.1. Sampling observations. Suppose we want to randomly select 5 units from a group of 15 sites to do our study.

```

> sites = 1:15
> units = sample(sites, 5, replace = F)
> units

```

```
[1] 2 7 6 1 14
```

`sample()` can also be used for character variables

```

> sitechar = c()
> for (i in 1:20) {
+   for (j in c("A", "B", "C", "D", "E")) {
+     sitechar = c(sitechar, paste(j, i, sep = ""))

```

```

+     }
+ }
> units = sample(sitechar, 15)
> units
[1] "C15" "D4" "E5" "E11" "E1" "C12" "C14" "B3" "B11" "A19" "E19" "D17"
[13] "E3" "E10" "B10"

```

6. Organization

Before we end a first session, we have some organizational matters to sort out.

- (1) We have been creating objects. We can accumulate a ton of them, and they can take up a lot of memory
- (2) use the function `ls()` to see what objects there are
- (3) use `rm(object.name)` to remove an object from your session (i.e. erase it!)
- (4) Under the misc menu item, you can select remove all objects, or equivalently enter the following `:rm(list=ls(all=TRUE))` to erase everything.
- (5) If you want to put comments into your code, use the '#' sign to comment out everything in the line after the '#'

```

> ls()
[1] "a"                "den.fact"          "example"
[4] "fish.re"          "i"                 "inputs"
[7] "j"                "prodsum"           "ReedfrogPred"
[10] "ReedfrogPred.sort" "sitechar"          "sites"
[13] "units"            "x"                 "x1"
> rm(x1)
> ls()
[1] "a"                "den.fact"          "example"
[4] "fish.re"          "i"                 "inputs"
[7] "j"                "prodsum"           "ReedfrogPred"
[10] "ReedfrogPred.sort" "sitechar"          "sites"
[13] "units"            "x"
> rm(list = ls(all = TRUE))
> ls()
character(0)

```

7. Regression

To get back into the stats world, we will do one example of linear regression. The MASS package is standard on all R installations. This corresponds to the book *Modern Applied Statistics with S-Plus* by Venables and Ripley. Its a great reference book, and the MASS package loads all of the libraries in the book. Here, we will look at the hills dataset. This dataset contains the record time, distance and height climbed for 35 Scottish hill races.

```

> library(MASS)
> summary(hills)

```

	dist	climb	time
Min.	: 2.000	Min. : 300	Min. : 15.95
1st Qu.:	4.500	1st Qu.: 725	1st Qu.: 28.00

```

Median : 6.000   Median :1000   Median : 39.75
Mean   : 7.529   Mean   :1815   Mean   : 57.88
3rd Qu.: 8.000   3rd Qu.:2200   3rd Qu.: 68.62
Max.   :28.000   Max.   :7500   Max.   :204.62

```

```

> cat("Simple linear regression of the time as a function of distance",
+     fill = T)

```

Simple linear regression of the time as a function of distance

```

> model1 = lm(time ~ dist, data = hills)
> summary(model1)

```

Call:

```
lm(formula = time ~ dist, data = hills)
```

Residuals:

```

      Min       1Q   Median       3Q      Max
-35.745  -9.037  -4.201   2.849   76.170

```

Coefficients:

```

              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -4.8407     5.7562  -0.841   0.406
dist           8.3305     0.6196  13.446 6.08e-15 ***
---

```

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 19.96 on 33 degrees of freedom

Multiple R-squared: 0.8456, Adjusted R-squared: 0.841

F-statistic: 180.8 on 1 and 33 DF, p-value: 6.084e-15

```

> par(mfrow = c(3, 2))
> plot(hills$dist, hills$time, main = "Plot of Data")
> abline(model1)
> qqnorm(studres(model1))
> qqline(studres(model1))
> cat("Multivariate regression with non-zero intercept")

```

Multivariate regression with non-zero intercept

```

> model2 = lm(time ~ dist + climb, data = hills)
> summary(model2)

```

Call:

```
lm(formula = time ~ dist + climb, data = hills)
```

Residuals:

```

      Min       1Q   Median       3Q      Max
-16.215  -7.129  -1.186   2.371   65.121

```

Coefficients:

```

              Estimate Std. Error t value Pr(>|t|)
(Intercept) -8.992039   4.302734  -2.090   0.0447 *
dist         6.217956   0.601148  10.343 9.86e-12 ***
climb        0.011048   0.002051   5.387 6.45e-06 ***

```

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 14.68 on 32 degrees of freedom
Multiple R-squared: 0.9191, Adjusted R-squared: 0.914
F-statistic: 181.7 on 2 and 32 DF, p-value: < 2.2e-16

```
> qqnorm(studres(model2))
> qqline(studres(model2))
> cat("Multivariate regression with zero intercept")
Multivariate regression with zero intercept
> model3 = lm(time ~ -1 + dist + climb, data = hills)
> summary(model3)
```

Call:
lm(formula = time ~ -1 + dist + climb, data = hills)

Residuals:

Min	1Q	Median	3Q	Max
-18.089	-10.053	-5.539	-3.180	58.235

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
dist	5.605651	0.551046	10.173	1.05e-11 ***
climb	0.010280	0.002118	4.853	2.84e-05 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 15.41 on 33 degrees of freedom
Multiple R-squared: 0.9613, Adjusted R-squared: 0.959
F-statistic: 409.8 on 2 and 33 DF, p-value: < 2.2e-16

```
> qqnorm(studres(model3))
> qqline(studres(model3))
> cat("Multivariate regression with zero intercept and interaction")
Multivariate regression with zero intercept and interaction
> model4 = lm(time ~ -1 + dist + climb + dist:climb, data = hills)
> summary(model4)
```

Call:
lm(formula = time ~ -1 + dist + climb + dist:climb, data = hills)

Residuals:

Min	1Q	Median	3Q	Max
-22.825344	-3.869912	-0.007844	2.455847	61.504223

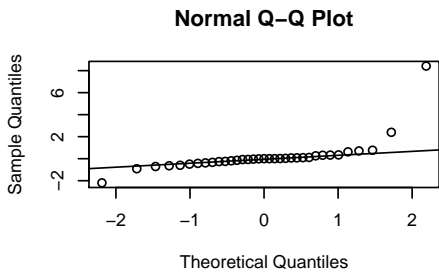
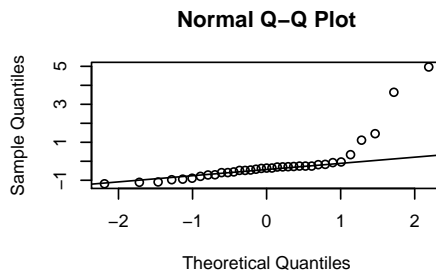
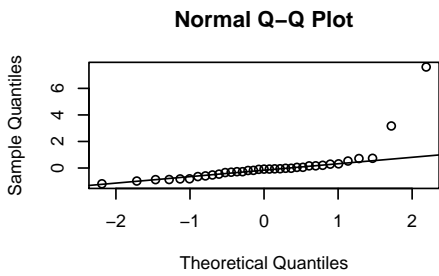
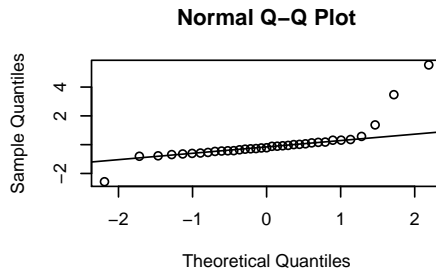
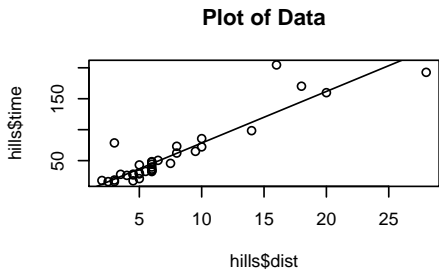
Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
dist	5.0794629	0.4896614	10.373	9.18e-12 ***
climb	0.0035501	0.0025614	1.386	0.175320
dist:climb	0.0006332	0.0001714	3.695	0.000818 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13.1 on 32 degrees of freedom
Multiple R-squared: 0.9729, Adjusted R-squared: 0.9703
F-statistic: 382.5 on 3 and 32 DF, p-value: < 2.2e-16

```
> qqnorm(studres(model4))  
> qqline(studres(model4))  
> cor(hills$climb, hills$dist)  
[1] 0.6523461
```



CHAPTER 2

Session # 2

1. Data manipulation

```
> fev = read.csv("fev.csv")
> summary(fev)
```

Age	FEV	Height	Gender
Min. : 3.000	Min. :0.791	Min. :46.00	Min. :0.0000
1st Qu.: 8.000	1st Qu.:1.981	1st Qu.:57.00	1st Qu.:0.0000
Median :10.000	Median :2.547	Median :61.50	Median :1.0000
Mean : 9.931	Mean :2.637	Mean :61.14	Mean :0.5138
3rd Qu.:12.000	3rd Qu.:3.119	3rd Qu.:65.50	3rd Qu.:1.0000
Max. :19.000	Max. :5.793	Max. :74.00	Max. :1.0000

Smoke
Min. :0.00000
1st Qu.:0.00000
Median :0.00000
Mean :0.09939
3rd Qu.:0.00000
Max. :1.00000

Here we see 3 continuous variables (Age, Height, FEV (a measure of vascular output)), and 2 categorical variables (Smoke (1=Smoker, 0= Non-smoker), and Gender (1= Male). I find this very cryptic to remember which is which gender and smoking class, so lets give those some sensible names.

There are a few ways we can subset data in R. In this case we will change the Gender and Smoke variables to factor character variables. There are a few ways that we can select a specific row within a column within R.

- (1) Use `attach(Name.Of.Dataset)`. Then the column names can be treated as variable names. Then, for Gender, `Gender[Gender==1]="Male"`
- (2) `$` allows you to subset within a dataset name, as is shown below
- (3) If you know what column number you are dealing with, then `fev[fev[,5]==1,5]="Male"`

Also, note that we use `[]` to denote subsetting within a dataset. If we are looking at a vector (`fev$Gender`, no spaces) then we don't need to worry about what column it is in, then we can put which rows we are looking for in the `[]`, otherwise we have to specify `[Desired Rows,Desired Columns]`. If you want all columns, it would be `[Rows,]` if you want all rows `[,Columns]`

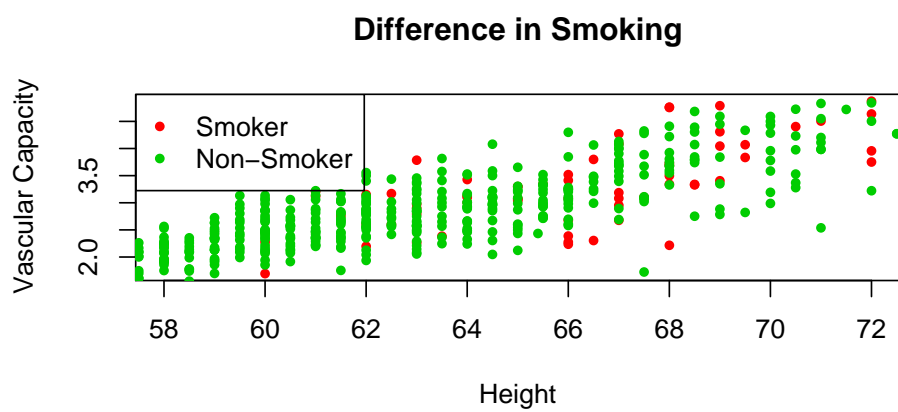
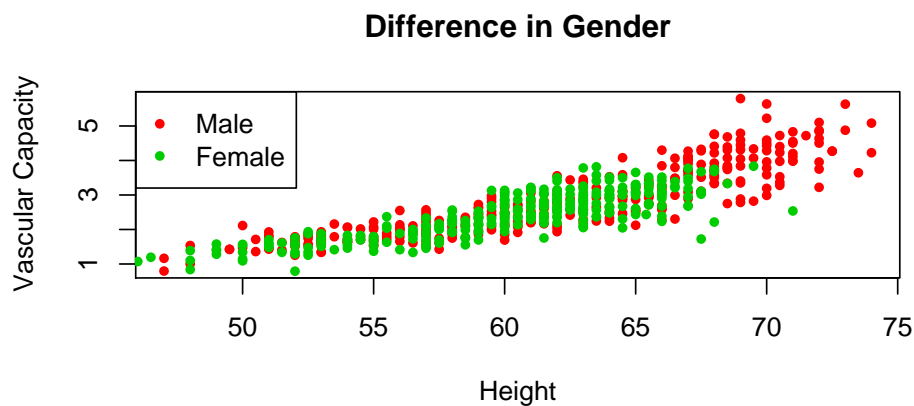
```
> fev$Gender[fev$Gender == 1] = "Male"
> fev$Gender[fev$Gender == 0] = "Female"
> fev$Smoke[fev$Smoke == 1] = "Smoker"
> fev$Smoke[fev$Smoke == 0] = "Non-smoker"
> fev$Gender = as.factor(fev$Gender)
```

```
> fev$Smoke = as.factor(fev$Smoke)
> summary(fev)
```

Age	FEV	Height	Gender	Smoke
Min. : 3.000	Min. :0.791	Min. :46.00	Female:318	Non-smoker:589
1st Qu.: 8.000	1st Qu.:1.981	1st Qu.:57.00	Male :336	Smoker : 65
Median :10.000	Median :2.547	Median :61.50		
Mean : 9.931	Mean :2.637	Mean :61.14		
3rd Qu.:12.000	3rd Qu.:3.119	3rd Qu.:65.50		
Max. :19.000	Max. :5.793	Max. :74.00		

Here we have set a factor (same as a class statement in SAS). This tells R the the values of Gender and Smoke are unique and are considered variable values, though not numeric. Now, if we want to do some data visualization:

```
> par(mfrow = c(2, 1))
> plot(FEV ~ Height, data = fev, col = 2, main = "Difference in Gender",
+      subset = Gender == "Male", ylab = "Vascular Capacity", pch = 20)
> points(FEV ~ Height, data = fev, col = 3, subset = Gender ==
+        "Female", pch = 20)
> legend("topleft", c("Male", "Female"), pch = c(20, 20), col = c(2,
+ 3))
> plot(FEV ~ Height, data = fev, col = 2, main = "Difference in Smoking",
+      subset = Smoke == "Smoker", ylab = "Vascular Capacity", pch = 20)
> points(FEV ~ Height, data = fev, col = 3, subset = Smoke == "Non-smoker",
+        pch = 20)
> legend("topleft", c("Smoker", "Non-Smoker"), pch = c(20, 20),
+        col = c(2, 3))
```



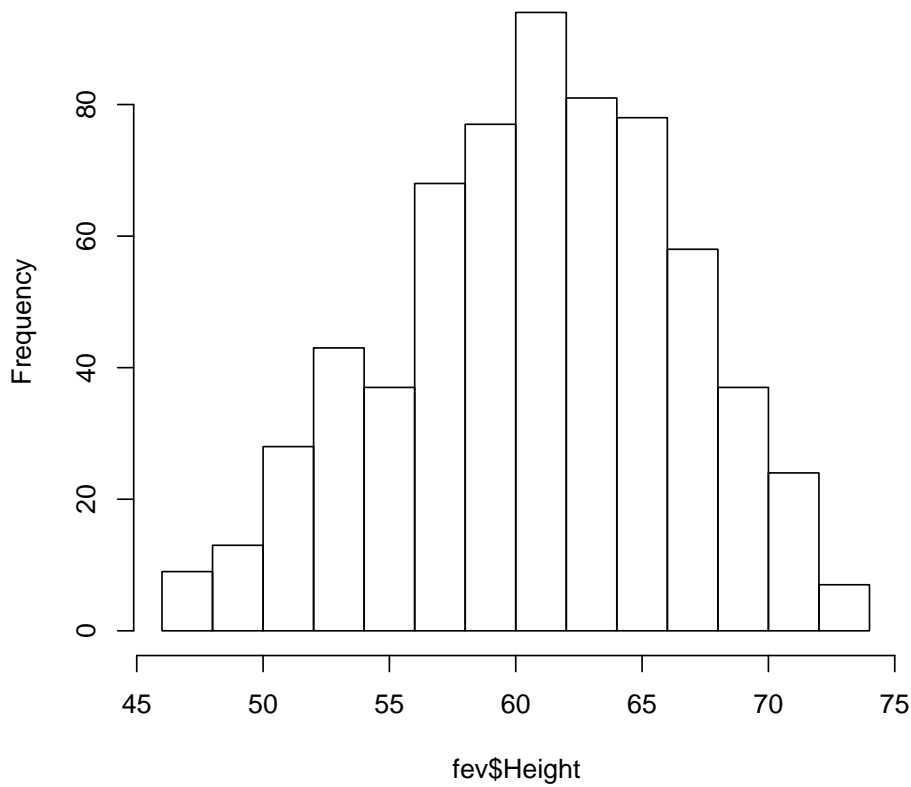
Note several arguments in the plot function:

- (1) When you give a dataset under the `data=` argument, you need to do `y-variable ~ x-variable`
- (2) `col=1` is black, `col=2` is green, `col=3` is red
- (3) `main=""` is the graph title line
- (4) `xlab=""`, `ylab=""` is the x and y axis label
- (5) `xlim=c(min,max)`, `ylim=c(min,max)` are the x and y axis ranges, where min and max must be entered by the user
- (6) `subset` reduces the subset of data to be plotted

For a histogram, simply

```
> hist(fev$Height)
```

Histogram of fev\$Height



```
> model1 = lm(FEV ~ Height, data = fev)
> summary(model1)
```

Call:

```
lm(formula = FEV ~ Height, data = fev)
```

Residuals:

Min	1Q	Median	3Q	Max
-1.751672	-0.266191	-0.004014	0.244745	2.119364

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	-5.432679	0.181460	-29.94	<2e-16 ***
Height	0.131976	0.002955	44.66	<2e-16 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4307 on 652 degrees of freedom

Multiple R-squared: 0.7537, Adjusted R-squared: 0.7533

F-statistic: 1995 on 1 and 652 DF, p-value: < 2.2e-16

```
> model2 = lm(FEV ~ Height * Age, data = fev)
> summary(model2)
```

```

Call:
lm(formula = FEV ~ Height * Age, data = fev)

Residuals:
    Min       1Q   Median       3Q      Max
-1.64871 -0.23420  0.01321  0.21743  1.80869

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.6636068  0.5098915  -1.301   0.194
Height       0.0458080  0.0087359   5.244 2.13e-07 ***
Age        -0.4181823  0.0561327  -7.450 2.99e-13 ***
Height:Age   0.0074975  0.0008801   8.518 < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3984 on 650 degrees of freedom
Multiple R-squared:  0.7899,    Adjusted R-squared:  0.7889
F-statistic: 814.4 on 3 and 650 DF,  p-value: < 2.2e-16
> model3 = lm(FEV ~ Height + Age, data = fev)
> summary(model3)

Call:
lm(formula = FEV ~ Height + Age, data = fev)

Residuals:
    Min       1Q   Median       3Q      Max
-1.50533 -0.25657 -0.01184  0.24575  2.01914

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -4.610466   0.224271 -20.558 < 2e-16 ***
Height       0.109712   0.004716  23.263 < 2e-16 ***
Age          0.054281   0.009106   5.961 4.11e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4197 on 651 degrees of freedom
Multiple R-squared:  0.7664,    Adjusted R-squared:  0.7657
F-statistic: 1068 on 2 and 651 DF,  p-value: < 2.2e-16
> model4 = lm(FEV ~ Height + Age + Gender, data = fev)
> summary(model4)

Call:
lm(formula = FEV ~ Height + Age + Gender, data = fev)

Residuals:
    Min       1Q   Median       3Q      Max
-1.37613 -0.24834  0.01051  0.25748  1.94538

```

```

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -4.448560   0.222966 -19.952 < 2e-16 ***
Height       0.104560   0.004756  21.986 < 2e-16 ***
Age          0.061364   0.009069   6.766 2.96e-11 ***
GenderMale   0.161112   0.033125   4.864 1.45e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4126 on 650 degrees of freedom
Multiple R-squared: 0.7746,      Adjusted R-squared: 0.7736
F-statistic: 744.6 on 3 and 650 DF,  p-value: < 2.2e-16
> model5 = lm(log(FEV) ~ Height + Age, data = fev)
> summary(model5)

Call:
lm(formula = log(FEV) ~ Height + Age, data = fev)

Residuals:
      Min       1Q   Median       3Q      Max
-0.64994 -0.08310  0.01055  0.09324  0.42156

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.971147   0.078332 -25.16 < 2e-16 ***
Height       0.043991   0.001647  26.71 < 2e-16 ***
Age          0.019816   0.003181   6.23 8.35e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1466 on 651 degrees of freedom
Multiple R-squared: 0.8071,      Adjusted R-squared: 0.8065
F-statistic: 1362 on 2 and 651 DF,  p-value: < 2.2e-16
> model6 = lm(log(FEV) ~ Height + Age + Gender, data = fev)
> summary(model6)

Call:
lm(formula = log(FEV) ~ Height + Age + Gender, data = fev)

Residuals:
      Min       1Q   Median       3Q      Max
-0.63616 -0.08685  0.01134  0.09035  0.40188

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.939555   0.078845 -24.599 < 2e-16 ***
Height       0.042986   0.001682  25.561 < 2e-16 ***
Age          0.021198   0.003207   6.610 8.03e-11 ***
GenderMale   0.031436   0.011714   2.684 0.00747 **
---

```

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1459 on 650 degrees of freedom

Multiple R-squared: 0.8092, Adjusted R-squared: 0.8083

F-statistic: 919 on 3 and 650 DF, p-value: < 2.2e-16

CHAPTER 3

Session # 3

1. Data Analysis continued

So far, we have seen a few ways to do simple subsetting of data. This week, I learned of another method, so I will share it with you

```
> rm(list = ls(all = TRUE))
> library(emdbook)
> data(ReedfrogPred)
> summary(ReedfrogPred)
```

density	pred	size	surv	propsurv
Min. :10.00	no :24	small:24	Min. : 4.00	Min. :0.1143
1st Qu.:10.00	pred:24	big :24	1st Qu.: 9.00	1st Qu.:0.4964
Median :25.00			Median :12.50	Median :0.8857
Mean :23.33			Mean :16.31	Mean :0.7216
3rd Qu.:35.00			3rd Qu.:23.00	3rd Qu.:0.9200
Max. :35.00			Max. :35.00	Max. :1.0000

```
> head(ReedfrogPred, 10)
```

	density	pred	size	surv	propsurv
1	10	no	big	9	0.9
2	10	no	big	10	1.0
3	10	no	big	7	0.7
4	10	no	big	10	1.0
5	10	no	small	9	0.9
6	10	no	small	9	0.9
7	10	no	small	10	1.0
8	10	no	small	9	0.9
9	10	pred	big	4	0.4
10	10	pred	big	9	0.9

```
> ReedfrogPred[1:10, ]
```

	density	pred	size	surv	propsurv
1	10	no	big	9	0.9
2	10	no	big	10	1.0
3	10	no	big	7	0.7
4	10	no	big	10	1.0
5	10	no	small	9	0.9
6	10	no	small	9	0.9
7	10	no	small	10	1.0
8	10	no	small	9	0.9
9	10	pred	big	4	0.4
10	10	pred	big	9	0.9

2. Sorting Data

One nice thing about R is that we typically do not need to sort data, as is often necessary in SAS. The `sort()` function will sort a vector, or series of vectors in increasing order, it can be made decreasing by adding the `decreasing=T` option.

If we want to sort a whole a whole data set based on one or more vectors, we are in essence changing the rows. So, as when we modify datasets by specifying the appropriate rows, we use the `order` function at the row identifier as below:

```
> attach(ReedfrogPred)
```

```
The following object(s) are masked from ReedfrogPred ( position 3 ) :
```

```
density pred propsurv size surv
```

```
The following object(s) are masked from ReedfrogPred ( position 4 ) :
```

```
density pred propsurv size surv
```

```
The following object(s) are masked from ReedfrogPred ( position 5 ) :
```

```
density pred propsurv size surv
```

```
> propsurv
```

```
[1] 0.9000000 1.0000000 0.7000000 1.0000000 0.9000000 0.9000000 1.0000000
[8] 0.9000000 0.4000000 0.9000000 0.7000000 0.6000000 0.7000000 0.5000000
[15] 0.9000000 0.9000000 0.9600000 0.9200000 0.8800000 1.0000000 0.9200000
[22] 0.9200000 0.9200000 0.8400000 0.2400000 0.5200000 0.1600000 0.3600000
[29] 0.5200000 0.8000000 0.3200000 0.4000000 0.9714286 0.9428571 0.9428571
[36] 0.8857143 0.8857143 1.0000000 0.9428571 0.9142857 0.1142857 0.3428571
[43] 0.3714286 0.4000000 0.6285714 0.3428571 0.8857143 0.4857143
```

```
> sort(propsurv, decreasing = T)
```

```
[1] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.9714286 0.9600000
[8] 0.9428571 0.9428571 0.9428571 0.9200000 0.9200000 0.9200000 0.9200000
[15] 0.9142857 0.9000000 0.9000000 0.9000000 0.9000000 0.9000000 0.9000000
[22] 0.9000000 0.8857143 0.8857143 0.8857143 0.8800000 0.8400000 0.8000000
[29] 0.7000000 0.7000000 0.7000000 0.6285714 0.6000000 0.5200000 0.5200000
[36] 0.5000000 0.4857143 0.4000000 0.4000000 0.4000000 0.3714286 0.3600000
[43] 0.3428571 0.3428571 0.3200000 0.2400000 0.1600000 0.1142857
```

```
> ReedfrogPred.sort = ReedfrogPred[order(propsurv, decreasing = T),
+ ]
```

```
> head(ReedfrogPred.sort)
```

```
density pred size surv propsurv
2      10 no big 10 1.0000000
4      10 no big 10 1.0000000
7      10 no small 10 1.0000000
20     25 no big 25 1.0000000
38     35 no small 35 1.0000000
33     35 no big 34 0.9714286
```

```
> detach()
```


CHAPTER 4

Last session

1. Functions

It may seem unlikely for many that learning how to run functions and loops in R would be helpful for those that aren't programmers or statisticians. However, its likely that many of us may do common things on the computer such as

- Sorting of data
- Common summaries of data
- Common types of analysis
- Common visualizations of data that you find helpful
- Tedious routines
- Many, many more

Here, we want to get summary of proportion of surviving reed frogs densities given the various factor (demographic) levels. Before we get carried away with functions, lets start with a for loop. First, we want to see if the variables are in factor form, so that we can use the function `expand.grid` (use `?expand.grid` to see what it does)

```
> library(emdbook)
> data(ReedfrogPred)
> summary(ReedfrogPred)
  density      pred      size      surv      propsurv
Min.   :10.00  no :24  small:24  Min.   : 4.00  Min.   :0.1143
1st Qu.:10.00  pred:24  big  :24  1st Qu.: 9.00  1st Qu.:0.4964
Median :25.00
Mean   :23.33
3rd Qu.:35.00
Max.   :35.00
      3rd Qu.:23.00  3rd Qu.:0.9200
      Max.   :35.00  Max.   :1.0000
> attach(ReedfrogPred)
The following object(s) are masked from ReedfrogPred ( position 3 ) :
  density pred propsurv size surv
The following object(s) are masked from ReedfrogPred ( position 4 ) :
  density pred propsurv size surv
The following object(s) are masked from ReedfrogPred ( position 5 ) :
  density pred propsurv size surv
> class(pred)
```

```

[1] "factor"
> class(size)
[1] "factor"
> class(density)
[1] "integer"

```

So size and pred are already factors, but density is not, so lets make it so. Then we will check the levels of the different parameters and use expand.grid to give us a matrix of subset combinations

```

> den.fact = as.factor(density)
> levels(pred)
[1] "no" "pred"
> levels(size)
[1] "small" "big"
> levels(den.fact)
[1] "10" "25" "35"
> inputs = expand.grid(x = levels(pred), y = levels(size), z = levels(den.fact))
> inputs
      x      y  z
1   no small 10
2  pred small 10
3   no   big 10
4  pred   big 10
5   no small 25
6  pred small 25
7   no   big 25
8  pred   big 25
9   no small 35
10  pred small 35
11  no   big 35
12  pred   big 35

```

Notice, the expand.grid makes a matrix of all of the subset combinations that we have. Now we will use the cat() function, which we use to output text and R variables as desired. We could do this by hand, and write 12 lines of code, iterating each factor as we go, but we would be likely to goof and forget a combination, not to mention its a lot of code to write.

To reach our pinnacle of laziness, we will apply a for-loop. A for-loop has the following structure

```

for(index.variable in items.to.run.index.on){
things indexed by index.variable
}

```

So, how do these works. Lets say we want to run an index on the numbers 1,2,3. Then we could have

```

for(i in 1:3){
things indexed by i
}

```

So as the loop starts, it will have the first value of the list, 1. *i* will have the value of 1 through all of the arguments between the braces. When all of the arguments have run their course, and we get to the end of the braces, then *i* will switch to the next value, 2. It will continue that way until it has exhausted all of the values of the list. This list need not be numbers, or in sequence.

For what we are going for, we get:

```
> for (i in 1:12) {
+   x = subset(ReedfrogPred, pred == inputs$x[i] & size == inputs$y[i] &
+     density == inputs$z[i])
+   cat("Mean(var) survival for predation=", as.character(inputs[i,
+     1]), ", size=", as.character(inputs[i, 2]), ", density=",
+     as.character(inputs[i, 3]), " is: ", round(mean(x$propsurv),
+     2), "(", round(var(x$propsurv), 2), ")", sep = "",
+     fill = T)
+ }
```

```
Mean(var) survival for predation=no, size=small, density=10 is: 0.93(0)
Mean(var) survival for predation=pred, size=small, density=10 is: 0.75(0.04)
Mean(var) survival for predation=no, size=big, density=10 is: 0.9(0.02)
Mean(var) survival for predation=pred, size=big, density=10 is: 0.65(0.04)
Mean(var) survival for predation=no, size=small, density=25 is: 0.9(0)
Mean(var) survival for predation=pred, size=small, density=25 is: 0.51(0.04)
Mean(var) survival for predation=no, size=big, density=25 is: 0.94(0)
Mean(var) survival for predation=pred, size=big, density=25 is: 0.32(0.02)
Mean(var) survival for predation=no, size=small, density=35 is: 0.94(0)
Mean(var) survival for predation=pred, size=small, density=35 is: 0.59(0.05)
Mean(var) survival for predation=no, size=big, density=35 is: 0.94(0)
Mean(var) survival for predation=pred, size=big, density=35 is: 0.31(0.02)
```

Let's notice a few things. First, we are using the index "i" to move us down the rows of the matrix input. Second, we use the inputs in two equivalent ways. Since inputs are an array of factors, the subset argument knows how to deal with it. The problem comes when we get to the cat argument, if we didn't include the "as.character()" argument, we'd end up with integers, not the values themselves, the as.character() function converts the factors into character values so that they are returned. Also, notice that when I want to include text of any form, I have to use quotes around it, where I do not use quotes when its for information provided by R. The sep="" argument just makes it so that there are no spaces between the arguments, I provide those myself where I think its appropriate. The fill=T command is important, try running this without it to see what it would do.

The basic forms of functions go as follows:

```
functionname=function(inputs){
R statements using inputs
Formulate desired outputs
return(outputs)
}
```

Example 1:

```
> prodsum = function(first, second) {
+   num.sum = first + second
+   num.diff = abs(first - second)
```

```

+   num.prod = first * second
+   return(num.sum, num.diff, num.prod)
+ }
> example = prodsum(5, 10)
> example
$num.sum
[1] 15

$num.diff
[1] 5

$num.prod
[1] 50
> example$num.sum
[1] 15

```

For a second, much more complicated example, below is a simplified function that I made to simulate tag-return data under the instantaneous rates formulation. A little background on what is going on here:

Suppose we tag R_i individuals in year i , and have x_{ij} tags from release cohort i returned in year j ($j > i$). We look at the probability that an individual would survive in year t , S_t and the probability of dying, and having the tag returned as f_t , then the instantaneous rates formulation just makes the following augmentations, generally:

$$(1) \quad S_t = \exp(-M_t - F_t)$$

$$(2) \quad f_t = \lambda \phi (1 - \exp(-M_t - F_t)) \frac{F_t}{M_t + F_t}$$

where M_t, F_t are the year-specific instantaneous natural and harvest mortality rates, respectively, λ is the tag-reporting rate, and ϕ is the combined probability of surviving tagging and retaining the tag throughout the study, typically, as in this case, its assumed that $\phi = 1$ and $M_t = M$ for all t .

The Brownie model is based on a joint multinomial likelihood. Let R_i be the number of tagged individuals released in year i , and x_{ij} be the number of tags released in year i and returned or reported to managers in year j . Then the likelihood function is

$$\Lambda = \prod_{i=1}^I \binom{R_i}{x_{ii}, x_{ii+1}, \dots, x_{iI}} \prod_{j=i}^J [P_{ij}^{x_{ij}}] P_{iI}^{R_i - \sum_{j=i}^J x_{ij}}$$

where,

$$P_{ij} = \begin{cases} f_j & \text{if } i=j; \\ (\prod_{k=i}^{j-1} S_k) f_j & \text{if } i < j \end{cases}$$

```

> fish.re = function() {
+   f = c(0.2, 0.5, 0.3, 0.4, 0.6, 0.3, 0.4)
+   M = 0.2
+   surv = rep(0, 7)
+   i = 1:7
+   surv[i] = exp(-M - f[i])
+   p = matrix(, nrow = 7, ncol = 8)
+   X = matrix(, nrow = 7, ncol = 8)

```

```

+   R = as.vector(rep(500, 7))
+   J = I_ = 7
+   lambda = 0.3
+   for (i in 1:(I_ - 1)) {
+     for (j in (i + 1):J) {
+       p[i, j] = lambda * prod(surv[i:(j - 1)]) * (1 - surv[j]) *
+         f[j]/(f[j] + M)
+     }
+     p[i, i] <- lambda * (1 - surv[i]) * f[i]/(f[i] + M)
+     p[i, (J + 1)] <- 1 - sum(p[i, (i:J)])
+     X[i, i:(J + 1)] <- rmultinom(1, size = R[i], prob = c(p[i,
+       i:(J + 1)]))
+   }
+   p[I_, J] = lambda * (1 - surv[I_]) * f[I_]/(f[I_] + M)
+   p[I_, (J + 1)] <- 1 - sum(p[I_, (I_:J)])
+   X[I_, I_:(J + 1)] <- rmultinom(1, size = R[I_], prob = c(p[I_,
+     I_:(J + 1)]))
+   return(X, R)
+ }
> a = fish.re()
> a$X
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]  21  34   9  12   8   2   1 413
[2,]  NA  45  12  15   7   1   0 420
[3,]  NA  NA  35  33  23   3   6 400
[4,]  NA  NA  NA  40  34  16   7 403
[5,]  NA  NA  NA  NA  69  19  13 399
[6,]  NA  NA  NA  NA  NA  36  30 434
[7,]  NA  NA  NA  NA  NA  NA  56 444
> a$R
[1] 500 500 500 500 500 500 500

```