

# One day R course\*

Casey Crisman-Cox

Fall 2025 Update

---

\*This is an abridged version of my R course notes available at [www.caseycrismancox.com/teaching](http://www.caseycrismancox.com/teaching)

# Contents

<b>1</b>	<b>Basics of R</b>	<b>3</b>
1.1	What is R . . . . .	3
1.2	Course aims and structure . . . . .	3
1.3	Installing R . . . . .	3
1.4	Using R as a Calculator . . . . .	7
1.5	Vectors and variables . . . . .	8
1.5.1	Naming variables . . . . .	9
1.5.2	Easy ways to create vectors . . . . .	12
1.5.3	Indexing . . . . .	12
1.5.4	Removing objects . . . . .	14
1.6	Matrices . . . . .	15
1.6.1	Matrix attributes . . . . .	18
1.6.2	Matrix operations . . . . .	22
1.7	Lists . . . . .	23
1.8	If and else . . . . .	27
1.9	Scripting and functions . . . . .	29
1.10	Packages and updating . . . . .	31
1.10.1	Updating R and R packages . . . . .	32
1.11	Getting help . . . . .	32
1.12	Exercises (break) . . . . .	33
<b>2</b>	<b>Data Frames</b>	<b>34</b>
2.1	Reading data . . . . .	34
2.2	Commands to use on Data . . . . .	36
2.2.1	Looking at the Variables . . . . .	36
2.2.2	Individual Variables . . . . .	40
2.2.3	Creating Subsets . . . . .	41
2.2.4	Classes . . . . .	47
2.3	Merging Data . . . . .	52
2.4	Generating New Variables . . . . .	57
2.4.1	Removing Variables . . . . .	58
2.5	Aggregating or summarizing Data . . . . .	59
2.6	Writing Data . . . . .	60
<b>3</b>	<b>Plotting</b>	<b>61</b>

3.1	Basic Plots . . . . .	61
3.2	Scatterplots and Layers . . . . .	63
3.3	Adding addition geoms . . . . .	76
3.4	Other plots . . . . .	77
	3.4.1 Histograms . . . . .	77
	3.4.2 Density . . . . .	80
3.5	Saving Plots . . . . .	83

# 1 Basics of R

## 1.1 What is R

You’ve already decided to learn R so I don’t need to write the congratulatory paragraph that opens nearly every R tutorial. But I will say a few nice things about R. Some of the things that R is good at

- New methods are frequently released with an R package or R code.
- If new methods don’t come with code you can write it yourself in R.
- I personally find data management easier to do in R.
- R plots are easy on the eyes.

## 1.2 Course aims and structure

At the end of today you should be able to

- Install/Update R and R packages
- Know where to look for R help
- Create simple programs and functions using R
- Use control statements to program iterative procedures
- Use R to read, save, and do some basic manipulations of data
- Create and save plots

## 1.3 Installing R

To install R for Windows

1. Go to <http://cloud.r-project.org/>
2. Click on “Download R for Windows”

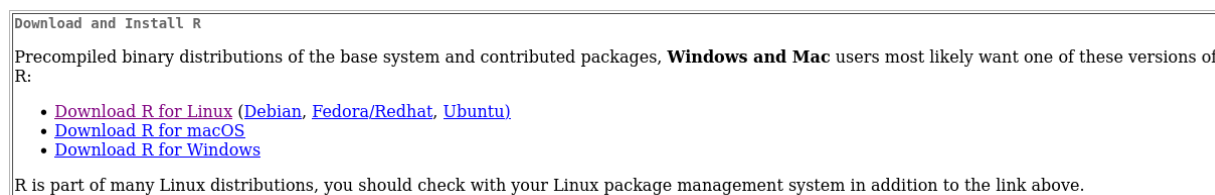


Figure 1: <http://cloud.r-project.org/>

3. Click on “base”

Subdirectories:

<a href="#">base</a>	Binaries for base distribution. This is what you want to <b>install R for the first time</b> .
<a href="#">contrib</a>	Binaries of contributed CRAN packages (for R $\geq$ 3.4.x).
<a href="#">old contrib</a>	Binaries of contributed CRAN packages for outdated versions of R (for R $<$ 3.4.x).
<a href="#">Rtools</a>	Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

**Figure 2:** <http://cloud.r-project.org/bin/windows/>

4. Finally click on the big button download at the top of the page and run the file that it downloads

[Download R-4.3.1 for Windows](#) (79 megabytes, 64 bit)  
[README on the Windows binary distribution](#)  
[New features in this version](#)

**Figure 3:** <http://cloud.r-project.org/bin/windows/base>

You now have R installed on your computer. Note the version number in the picture is old! But the process holds up.

To install R on a Mac is largely the same.

1. Go to <http://cloud.r-project.org/>
2. Click on “Download R for (Mac OS X)”

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux \(Debian, Fedora/Redhat, Ubuntu\)](#)
- [Download R for macOS](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

**Figure 4:** <http://cloud.r-project.org/>

3. Click on the version that matches your Mac

You now have R installed on your computer. Again the pictures are old, but the process is true.

For Linux users you’ll want to follow click on “Download R for Linux” find your distribution and follow the instructions.

When you’ve finished installing R open it up you should see something that looks like this:

As you can see in the picture, this version of R is version 3.1.2, if we wanted more information about the type of R we’re running we can use the command `sessionInfo()` and we get

## R for macOS

This directory contains binaries for the base distribution and of R and packages to run on macOS. R and package binaries for R versions older than 4.0.0 are only available from the [CRAN archive](https://cran-archive.r-project.org) so users of such versions should adjust the CRAN mirror setting (<https://cran-archive.r-project.org>) accordingly.

Note: Although we take precautions when assembling binaries, please use the normal precautions with downloaded executables.

R 4.3.1 "Beagle Scouts" released on 2023/06/16

Please check the integrity of the downloaded package by checking the signature:

```
pkgutil --check-signature R-4.3.1.pkg
in the Terminal application. If Apple tools are not available you can check the SHA1 checksum of the downloaded image:
openssl sha1 R-4.3.1.pkg
```

Latest release:

For Apple silicon (M1/M2) Macs:

[R-4.3.1-arm64.pkg](#)  
SHA1-hash: 14c018054f75bb37c1d90b33207343b83e9345  
(ca. 90MB, notarized and signed)

For older Intel Macs:

[R-4.3.1-x86\\_64.pkg](#)  
SHA1-hash: 1af8055a601d5de5df6db3956ecc8f745c2401  
(ca. 92MB, notarized and signed)

**R 4.3.1** binary for macOS 11 (**Big Sur**) and higher, signed and notarized packages.

Contains R 4.3.1 framework, Rapp GUI 1.79, Tcl/Tk 8.6.12 X11 libraries and Texinfo 6.8. The latter two components are optional and can be omitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from sources.

macOS Ventura users: there is a known bug in Ventura preventing installations from some locations without a prompt. If the installation fails, move the downloaded file away from the *Downloads* folder (e.g., to your home or Desktop)

Note: the use of X11 (including `tcltk`) requires [XQuartz](#) (version 2.8.5 or later). Always re-install XQuartz when upgrading your macOS to a new major version.

This release uses Xcode 14.2/14.3 and GNU Fortran 12.2. If you wish to compile R packages which contain Fortran code, you may need to download the corresponding GNU Fortran compiler from <https://mac.R-project.org/tools>. Any external libraries and tools are expected to live in `/opt/R/arm64` (Apple silicon) or `/opt/R/x86_64` (Intel).

Figure 5: <http://cloud.r-project.org/bin/macosx/>

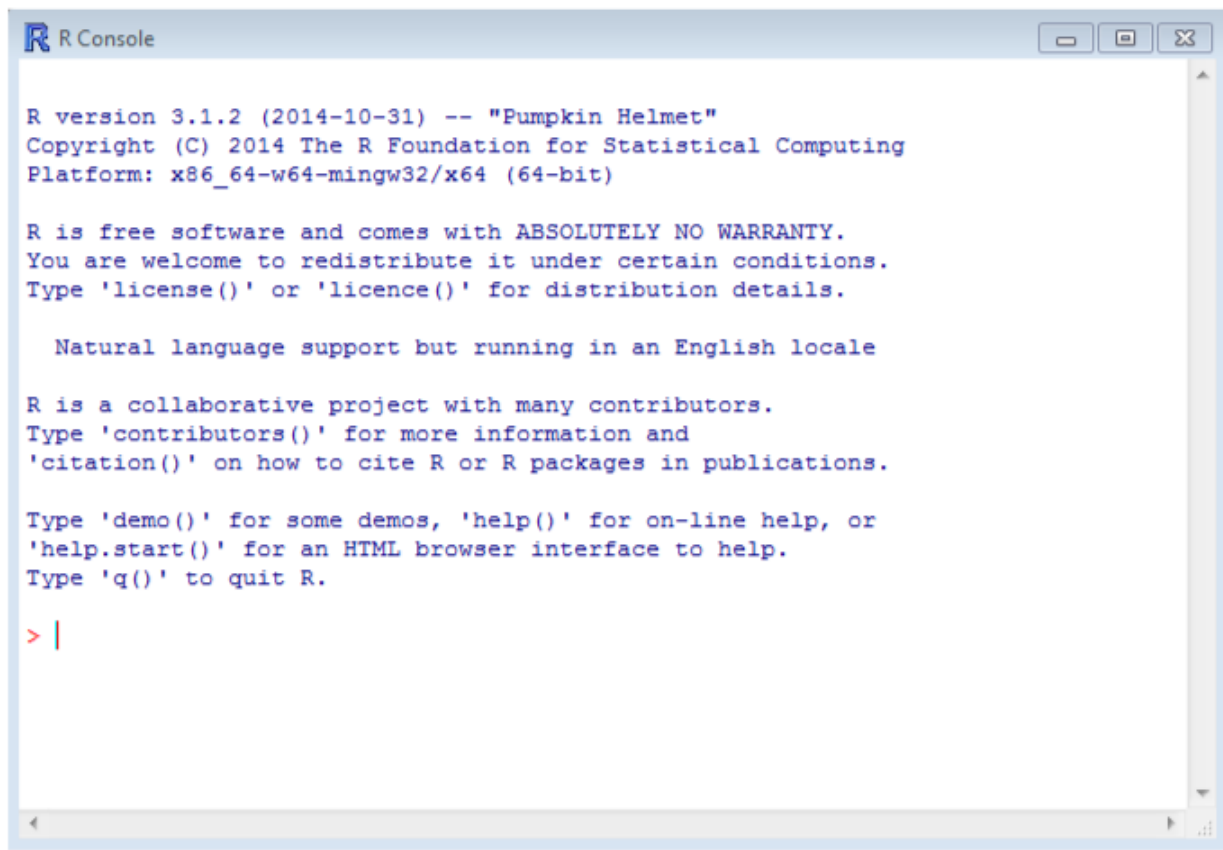


Figure 6: R Console

```
sessionInfo()
```

```
## R version 4.5.1 (2025-06-13)
## Platform: x86_64-pc-linux-gnu
## Running under: Ubuntu 24.04.2 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblas-p0.3.26.so; LAPACK v
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8      LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8      LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8  LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8     LC_NAME=C
##  [9] LC_ADDRESS=C             LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## time zone: America/Chicago
## tzcode source: system (glibc)
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets  methods    base
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.5.1    fastmap_1.1.1     cli_3.6.5         tools_4.5.1
##  [5] htmltools_0.5.7   rstudioapi_0.15.0 yaml_2.3.8         rmarkdown_2.25
##  [9] knitr_1.45        xfun_0.41         digest_0.6.37     rlang_1.1.6
## [13] evaluate_0.23
```

Which shows us the version of R we're using, our operating system (actually 4.5.1, again pictures are old!), and the packages we currently have loaded. Since you haven't loaded any packages yet, the packages listed are those that that R loads automatically each time it opens (base packages).

**Note** *In the above chunk I have the several packages loaded as part of making these notes, which means my output has "other attached packages" and "loaded via namespace" your output*

*will not have that.*

RStudio is an excellent alternative to the R console as it provides a nice system to edit your files while you're working on them and keep everything better organized. To download RStudio visit <https://posit.co/download/rstudio-desktop/> and find the installer that matches your system. I strongly recommend the use of RStudio over the regular R console for ease of use and organization.

## 1.4 Using R as a Calculator

Now that we've gone through that ordeal, let's actually use R for something. When we open up R we have the rather intimidating looking prompt staring at us. Whenever we see

```
>
```

It just means that R is waiting for us to give it something to do. Let's start with something simple

```
1+1
```

```
## [1] 2
```

Which gives our answer and returns us to the >. Now we don't have to fit everything on one line. If we don't type a full command R changes the > to a + to let us know that it needs more from us. For example:

```
> 2*  
+ 3
```

```
## [1] 6
```

If for some reason you get the + and you don't know what went wrong you can hit the escape button on your keyboard and that stops R and returns you to the >. Escape will terminate anything R is doing and return you to the > prompt.

All the basic operations work in R so +, -, \*, /, ^ do addition, subtraction, multiplication, division, and exponents just as we would expect them to do. Additionally, standard functions are available so:

```
log(10) #base= e
```

```
## [1] 2.302585
```



```
log(10, base=10)
```

```
## [1] 1
```

```
exp(1)
```

```
## [1] 2.718282
```

```
sin(0)
```

```
## [1] 0
```

```
acos(-1)
```

```
## [1] 3.141593
```

Note that `#` is how we use comments in R. A comment is just a remark we put with our code but don't want R to evaluate. So after the `#` R stops reading the line.

Also, R can't do the impossible so

```
log(0)
```

```
## [1] -Inf
```

```
log(-1)
```

```
## Warning in log(-1): NaNs produced
```

```
## [1] NaN
```

Where `-Inf` means  $-\infty$  and `NaN` means “Not a Number.”

Getting those is a sign that you need to reevaluate what you're doing.

## 1.5 Vectors and variables

Now we want to use R for more than just a calculator (your computer already has one of those). So now we want to expand what we can do, the first way we'll do that is by assigning the output of our calculations to a variable. In R, an assignment can take many forms, and all of the following are the same.

```
x <- exp(1)
```

```
x = exp(1)
```

```
exp(1) -> x
```

```
assign('x', exp(1))
```

For the most part, you'll only ever see the first two, and most R users prefer the `<-`. Once a value is assigned to variable we can use `x` like any other number and so

```
x
```

```
## [1] 2.718282
```

```
x-2
```

```
## [1] 0.7182818
```

```
log(x)
```

```
## [1] 1
```

If we want to assign a new value to `x` we just use the arrow again

```
x <- exp(2)
```

```
x
```

```
## [1] 7.389056
```

### 1.5.1 Naming variables

We can name variables anything. Within code it is often better to use descriptive names. The only rules about naming variable is that it can't start with a number or contain any symbols except for periods and underscores.

```
n <- 50 #Good but not descriptive
numberOfStates <- 50 #Good and descriptive
number.of.states <- 50 #Still good
number_of_states <- 50 #Still good
number-of-states <- 50 #Not good
```

```
## Error: object 'number' not found
```

**Note** *Check your understanding: The last one returned an error, why?*

Using dashes made R think we wanted to subtract the variable `number` minus the variable `of` minus the variable `students`. If these variables had existed we would have gotten a different error because R would think we wanted to assign the value 10 to this difference, which it would say is nonsense.

Notice that all of our output began with the symbol `[1]`, for example

```
2+2
```

```
## [1] 4
```

The `[1]` just means that R thinks of this as a **vector**. A vector is a collection of 1 or more values. The `[1]` just tells you that the value next to it is the first number in the vector. There's no reason why a variable in R has to have only one value. The simplest way to create vector is with the `c()` function. For example

```
x1 <- c(1, 2, 3, 4)
x1
```

```
## [1] 1 2 3 4
```

Notice that the `[1]` is still there to tell us that the number next to it is the first value in the vector. The `c` in this function just stands for “concatenate” and it can be used to bring lots of vectors together

```
x2 <- c(1, 0, -1, 1)
c(x2, x2, x2, x1, x1, x1, x2, x2, x1, x1)
```

```
## [1] 1 0 -1 1 1 0 -1 1 1 0 -1 1 1 2 3 4 1 2 3 4 1 2 3 4 1
## [26] 0 -1 1 1 0 -1 1 1 2 3 4 1 2 3 4
```

Where we can now see that whenever the output goes onto a second line we get a new indicator to tell us what position it is. So in the above we have `[1]` at the beginning of the output and then `[26]` to tell us the value that starts the second line is the 26th value in the vector.

Nearly all the functions we looked at before work on vectors. For instance

```
x1+x2
```

```
## [1] 2 2 2 5
```

```
x1/x2
```

```
## [1] 1 Inf -3 4
```

```
log(x1)
```

```
## [1] 0.0000000 0.6931472 1.0986123 1.3862944
```

And there are some nice functions to describe or aggregate vectors.

```
sum(x1) ## add all the elements
```

```
## [1] 10
```

```
cumsum(x1) ## add the elements before i to i
```

```
## [1] 1 3 6 10
```

```
prod(x1) ## multiply all the elements
```

```
## [1] 24
```

```
mean(x1) ## average
```

```
## [1] 2.5
```

```
median(x1) ## median
```

```
## [1] 2.5
```

```
var(x1) ## variance
```

```
## [1] 1.666667
```

```
sd(x1) ## standard deviation
```

```
## [1] 1.290994
```

If you don't remember some of these, you'll cover them in 602.

We can also sort the values within a vector

```
sort(x1)
```

```
## [1] 1 2 3 4
```

```
sort(x1, decreasing=TRUE)
```

```
## [1] 4 3 2 1
```

```
length(x1)
```

```
## [1] 4
```

### 1.5.2 Easy ways to create vectors

If we want to create a vector that follows a pattern, we don't need to take the time to type it in. For instance if we just want all the numbers between 1 and 15 in a vector we can use the colon.

```
1:15
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

```
5:2
```

```
## [1] 5 4 3 2
```

Notice that R reads the second one as a sequence from 5 to 2, and so it goes in decreasing order. The more general version of the colon is the `seq()` command

```
seq(0, 20)
```

```
## [1] 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

```
seq(0, 20, by=2)
```

```
## [1] 0 2 4 6 8 10 12 14 16 18 20
```

```
seq(0, 20, length.out=5)
```

```
## [1] 0 5 10 15 20
```

Finally the `rep` command allows you to repeat numbers

```
rep(10, 2)
```

```
## [1] 10 10
```

```
rep(x1, 3)
```

```
## [1] 1 2 3 4 1 2 3 4 1 2 3 4
```

```
rep(x1, each=3) #Repeats each number within x1 one at a time
```

```
## [1] 1 1 1 2 2 2 3 3 3 4 4 4
```

### 1.5.3 Indexing

Let's say we want to extract or replace a single number within a vector. In these cases we use the square brackets, for example

```
z <- seq(0, 6, by =2)
z[3] #3rd entry
```

```
## [1] 4
```

```
z[1:3] #1st three entries
```

```
## [1] 0 2 4
```

```
z[c(1, 3)] #Entries 1 and 3, note that we need c()
```

```
## [1] 0 4
```

```
z[-c(1,3)] #Everything but 1 and 3
```

```
## [1] 2 6
```

We can also extract based on a pattern using logical operators. Let's say we only want elements of `z` that are greater than 10. The logical statement is

```
z > 3
```

```
## [1] FALSE FALSE TRUE TRUE
```

Which returns a vector of `TRUE` and `FALSE` values to show if a particular element in `z` meets the condition we gave it. Now in order to use that to get the elements we want do the following:

```
z[z>3]
```

```
## [1] 4 6
```

The list of commonly used logical operators is shown in table 1

**Table 1:** Logical operators

Operator	Meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal
!=	Not equal
!	Not

Logical conditions can be strung together use & (and) and | (or)

```
z > 3 & z < 5
```

```
## [1] FALSE FALSE TRUE FALSE
```

```
z[z > 3 & z < 5]
```

```
## [1] 4
```

```
z[z < 3 | z > 5]
```

```
## [1] 0 2 6
```

### 1.5.4 Removing objects

We use the `ls()` command to view all the objects that we've created

```
ls()
```

```
## [1] "n"                "number_of_states" "number.of.states" "numberOfStates"
## [5] "x"                "x1"                "x2"                "z"
```

Now lets say we wanted to get rid of some things. For this we use the `rm()` command, but be careful, there's no undo for this.

```
rm(list='number.of.states')
ls()
```

```
## [1] "n"                "number_of_states" "numberOfStates"   "x"
## [5] "x1"                "x2"                "z"
```

```
rm(list=c('x1', 'y2')) #We can delete more than one thing at time.
```

```
## Warning in rm(list = c("x1", "y2")): object 'y2' not found
```

```
ls()
```

```
## [1] "n"                "number_of_states" "numberOfStates"   "x"
## [5] "x2"                "z"
```

```
rm(list=ls()) #We can delete everything
ls()
```

```
## character(0)
```

It's worth noting at this point that a vector doesn't have to be numbers it could be

```
x <- c('cat', 'dog', 'horse')
```

Until we get more into data analysis there isn't a whole lot of reason to get into strings. I will note that the `stringr` package contains many good tools for manipulating string variables should you find yourself needing to do that.

## 1.6 Matrices

A matrix is just a 2 dimensional version of the vector. To create a matrix you just need a vector of values and then tell R one of the dimensions

```
x <- 1:10  
matrix(x, nrow=2)
```

```
##      [,1] [,2] [,3] [,4] [,5]  
## [1,]    1    3    5    7    9  
## [2,]    2    4    6    8   10
```

```
matrix(x, ncol=2)
```

```
##      [,1] [,2]  
## [1,]    1    6  
## [2,]    2    7  
## [3,]    3    8  
## [4,]    4    9  
## [5,]    5   10
```

Notice that R fills in the numbers column-wise, but we can also fill in row wise

```
matrix(x, ncol=2, byrow=TRUE)
```

```
##      [,1] [,2]  
## [1,]    1    2  
## [2,]    3    4  
## [3,]    5    6  
## [4,]    7    8  
## [5,]    9   10
```

We can also use `cbind` and `rbind` to “bind” vectors together to make a matrix, bind a vector(s) to a matrix, or bind matrices together



```
x2 <- -10:-1
cbind(x, x2)
```

```
##      x  x2
## [1,] 1 -10
## [2,] 2  -9
## [3,] 3  -8
## [4,] 4  -7
## [5,] 5  -6
## [6,] 6  -5
## [7,] 7  -4
## [8,] 8  -3
## [9,] 9  -2
## [10,] 10 -1
```

```
rbind(x, x2)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## x      1    2    3    4    5    6    7    8    9    10
## x2   -10   -9   -8   -7   -6   -5   -4   -3   -2   -1
```

```
z <- 1:5
cbind(x, x2, z)
```

```
##      x  x2 z
## [1,] 1 -10 1
## [2,] 2  -9 2
## [3,] 3  -8 3
## [4,] 4  -7 4
## [5,] 5  -6 5
## [6,] 6  -5 1
## [7,] 7  -4 2
## [8,] 8  -3 3
## [9,] 9  -2 4
## [10,] 10 -1 5
```

Notice that there's no limit to the number of things we can bind together in one use of `cbind`.

The `diag` command has a few different uses.

```
diag(4) # 4 x 4 identity matrix
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1
```

```
diag(x) #A square matrix with diagonal = x
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    1    0    0    0    0    0    0    0    0    0
## [2,]    0    2    0    0    0    0    0    0    0    0
## [3,]    0    0    3    0    0    0    0    0    0    0
## [4,]    0    0    0    4    0    0    0    0    0    0
## [5,]    0    0    0    0    5    0    0    0    0    0
## [6,]    0    0    0    0    0    6    0    0    0    0
## [7,]    0    0    0    0    0    0    7    0    0    0
## [8,]    0    0    0    0    0    0    0    8    0    0
## [9,]    0    0    0    0    0    0    0    0    9    0
## [10,]   0    0    0    0    0    0    0    0    0   10
```

```
Z <- matrix(1:9, nrow = 3)
Z
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
diag(Z) #Extract the diagonal of a square matrix
```

```
## [1] 1 5 9
```

If for some reason you wanted to turn a matrix into vector there are few ways to do that

```
c(Z)
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

```
as.vector(Z)
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

if you have any doubts about whether something is a vector you can always check its class

```
class(x)
```

```
## [1] "integer"
```

```
class(Z)
```

```
## [1] "matrix" "array"
```

### 1.6.1 Matrix attributes

Just like with vectors we can use the square brackets to extract elements. For a matrix  $X$ , the command  $X[i, j]$  gives you the element from row  $i$ , column  $j$ .

```
X <- matrix(1:12, nrow=3)
```

```
X
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
X[2, 4]
```

```
## [1] 11
```

As before we can replace individual elements

```
X[3,2] <- 8
```

```
X
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    8    9   12
```

We can also extract whole rows and columns

```
X[1, ] #First row
```

```
## [1] 1 4 7 10
```

```
X[, 2] #Second Column
```

```
## [1] 4 5 8
```

```
X[1:2,] ##First two columns
```

```
##      [,1] [,2] [,3] [,4]
```

```
## [1,]    1    4    7   10
```

```
## [2,]    2    5    8   11
```

Notice that when we pull out just one row or column R converts it into a vector, we can use the `drop` argument to stop that

```
X[1, ,drop=FALSE]
```

```
##      [,1] [,2] [,3] [,4]
```

```
## [1,]    1    4    7   10
```

```
class(X[1, ,drop=FALSE])
```

```
## [1] "matrix" "array"
```

As before we can use the logical operators

```
X[, 2] == 8 # which rows have 8 in the second column?
```

```
## [1] FALSE FALSE  TRUE
```

```
X[X[, 2] == 8, ]
```

```
## [1] 3 8 9 12
```

For the most part R treats matrices as just vectors that are written differently, this means that if we ask R for things like length, mean, and standard deviation it gives it to us for all the values.

```
length(X)
```

```
## [1] 12
```

```
mean(X)
```

```
## [1] 6.666667
```

```
sd(X)
```

```
## [1] 3.626502
```

Some things will work on directly on matrices, such finding the shape

```
dim(X) #dimensions of X
```

```
## [1] 3 4
```

```
nrow(X) #rows of X
```

```
## [1] 3
```

```
ncol(X) #columns of X
```

```
## [1] 4
```

But what if we wanted means by column? This takes us to our first introduction of the `for` loop and the `apply` function. We will cover them in greater detail later but for now let's start with `for` loop.

```
mean.x <- rep(0, ncol(X)) #Recall that this creates a vector of 0s  
#equal to the length of ncol(X)  
for(i in 1:ncol(X)){  
  mean.x[i] <- mean(X[,i]) #What does this do?  
}  
mean.x
```

```
## [1] 2.000000 5.666667 8.000000 11.000000
```

```
apply(X, 2, mean) # Same thing
```

```
## [1] 2.000000 5.666667 8.000000 11.000000
```

```
colMeans(X) #Best way to do this!
```

```
## [1] 2.000000 5.666667 8.000000 11.000000
```

Both the loop and `apply` do the same thing, but sometimes one is easier to write than the other. So let's break down what these things do.

1. Before we even ran the `for` loop we created a vector in which to store the results. We filled the vector with 0s but we really could have filled them with anything. I like using 0s because it makes it easy to spot if something goes wrong. Zeros are also better than missing values `NA` because they don't involve changing types (non-number to number) as you fill in the vector.
2. We start the loop with `for(i in 1:ncol(X))`. This tells R that we're going to use a variable `i` that takes the values `1, 2, ..., ncol(X)`, and once `i` takes the last value in that sequence the loop is done.
3. The curly brackets tell R what we want repeated

The `apply` function on the other hand takes 3 arguments.

1. A matrix, in this case `X`.
2. A direction, 2 means that want to apply the function over columns, 1 means apply it over rows.
3. The function we want done to each column or row. Here, we used `mean`, but it could be any function, including one you write yourself once we get to writing functions.

Finally, for this specific example there is a built in function `colMeans` (and `rowMeans`) that is faster than either `for` or `apply`, but that won't be the case for every operation you want to do.

Note that one thing we can do with matrices that we can't do with vectors is name the rows and the columns. These names are just string vectors.

```
X <- diag(2)
colnames(X)
```

```
## NULL
```

```
colnames(X) <- c('left', 'right')
X
```

```
##      left right
## [1,]    1     0
## [2,]    0     1
```

```
colnames(X)[2] <- 'Right'
X
```

```
##      left Right
## [1,]    1     0
```

```
## [2,]    0    1
row.names(X) <- c('up', 'down')
X
```

```
##      left Right
## up      1     0
## down    0     1
```

```
X[, 'left'] ## We can use the names in place of numbers to index
```

```
##   up down
##   1    0
```

```
X['up', 'left']
```

```
## [1] 1
```

### 1.6.2 Matrix operations

Matrix math in R includes standard operations including arithmetic.

```
X <- matrix(1:4, nrow=2)
Y <- diag(2) #Identity matrix
X + Y
```

```
##      [,1] [,2]
## [1,]    2    3
## [2,]    2    5
```

```
X-Y
```

```
##      [,1] [,2]
## [1,]    0    3
## [2,]    2    3
```

Note that `*` performs *element-wise* multiplication. In the coming weeks you'll learn more about matrix operations, as you do you'll find the following functions/operators useful

1. Matrix multiplication `X %*% Y`
2. Matrix inversion `solve(X)`
3. Matrix transpose `t(X)`
4. Determinant `det(X)`

5. Eigenvalues and eigenvectors `eigen(X)`
6. Cholesky decomposition `chol(X)`

## 1.7 Lists

When R returns a list to us we can extract the elements of it using the dollar sign with the appropriate name. The names are given by the output, in the above example the names given to us are “values” and “vectors.” If we didn’t know the names we can look using the `names` command. Let’s make our own list and then try it

```
Y <- matrix(c(1, 0.5, 0.5, 1), nrow=2)
matrixList <- list(matrix = diag(4), #Identity matrix
                  Y = Y,
                  nrowY = nrow(Y))

names(matrixList)
```

```
## [1] "matrix" "Y"      "nrowY"
```

```
matrixList$matrix
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1
```

```
matrixList$Y
```

```
##      [,1] [,2]
## [1,]  1.0  0.5
## [2,]  0.5  1.0
```

Alternatively, we can still use brackets, but with lists we have to double them up to get the specific element extracted from the list. For example, compare

```
matrixList[3]
```

```
## $nrowY
## [1] 2
```



```
matrixList[[3]]
```

```
## [1] 2
```

Lists are very flexible because they are way to combine matrices of different dimensions with vectors, or to put many statistical models together in one group. We can also nest lists within lists.

```
matrixList$sizeY <- list(rows=nrow(Y), cols=ncol(Y))
```

If we wanted to extract just the columns from this list we could use either the names or the square brackets.

```
matrixList$sizeY$cols
```

```
## [1] 2
```

```
matrixList[[4]][[2]] #Same thing
```

```
## [1] 2
```

Finally, we have two more forms of apply that we can use on just lists. The first one we'll look at is `lapply` which is read "L- Apply" and stands for list apply. When we use `lapply` it performs some function that we want over the entire list. So if we wanted to know the length of each object in a list we could do the following.

```
lapply(matrixList, length)
```

```
## $matrix
```

```
## [1] 16
```

```
##
```

```
## $Y
```

```
## [1] 4
```

```
##
```

```
## $nrowsY
```

```
## [1] 1
```

```
##
```

```
## $sizeY
```

```
## [1] 2
```

Notice that `lapply` returns a list, this can be rather cumbersome, which is why we sometimes use `sapply` instead. The `sapply` command does the same thing but returns the results in

vector form if possible.

```
sapply(matrixList, length)
```

```
## matrix      Y nrowY  sizeY
##      16      4      1      2
```

Other \*ply functions exist, notably, `tapply` (apply a function over a group) and `mapply`, but I don't find myself using either of those very much, so we'll leave it at that.

In most of the really useful applications of these functions we would have a list where all the elements were of the same class. Let's say we have a bunch of matrices and want to know the column means of each one.

```
matrixList <- list(matrix1 = matrix(1:9, nrow=3),  #3 x 3
                   matrix2 = matrix(0:5, nrow=2),  #2 x 3
                   matrix3 = cbind(rnorm(3), 1))    #3 x 2
```

```
matrixList
```

```
## $matrix1
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## $matrix2
##      [,1] [,2] [,3]
## [1,]    0    2    4
## [2,]    1    3    5
##
## $matrix3
##      [,1] [,2]
## [1,] 0.2788785    1
## [2,] 0.2264007    1
## [3,] -0.6227685    1
```

```
lapply(matrixList, class) # make sure they're all matrices
```

```
## $matrix1
## [1] "matrix" "array"
```

```
##
## $matrix2
## [1] "matrix" "array"
##
## $matrix3
## [1] "matrix" "array"

lapply(matrixList, dim)  # check dimensions
```

```
## $matrix1
## [1] 3 3
##
## $matrix2
## [1] 2 3
##
## $matrix3
## [1] 3 2
```

```
lapply(matrixList, apply, 2, mean)
```

```
## $matrix1
## [1] 2 5 8
##
## $matrix2
## [1] 0.5 2.5 4.5
##
## $matrix3
## [1] -0.03916309 1.00000000
```

Notice that in the last one we used `apply` within `lapply`. We then just write the arguments that we would use with `apply` as additional arguments. This is something that we can generally do with functions in the `apply` family. For example

```
X <- matrix(c(1, NA, 1,1), nrow=2) #Row 2 has a missing value
mean(X[2, ]) #is NA
```

```
## [1] NA
```

```
mean(X[2, ], na.rm=TRUE) #Tells R to just ignore missing values
```

```
## [1] 1
```

```
apply(X, 1, mean) #Gives us that NA
```

```
## [1] 1 NA
```

```
apply(X, 1, mean, na.rm=TRUE) #add option na.rm=TRUE
```

```
## [1] 1 1
```

## 1.8 If and else

When we want to use logical conditions we can use `if` and `else` as separate commands. They have the following setup:

```
if(LOGICAL){  
  COMMAND1  
  COMMAND2  
}else{  
  COMMAND  
}
```

Notice the use of `{}` to contain the conditions. While you sometimes find code that does not use these (you don't need them for one line statements), I *strongly* encourage you to always be explicit and use them as much as possible. This makes your code less prone to breaking and much more readable to you, others, and, perhaps most importantly, your future selves.

Let's look at an example of a trivial if statement.

```
y <- FALSE  
if(y){  
  cat("Hello World")  
}else{  
  cat("Goodbye")  
}
```

```
## Goodbye
```

We can also nest if statements. Try the following: Generate a value of `test` and predict which name will be printed. Make sure you understand why a given name is being displayed.

```
test <- runif(1)  
print(test)
```

```
## [1] 0.186342
```

```
if(test < 1/2){  
  if(test < 1/3){  
    "Mary"  
  }else{  
    if(test < 0.4){  
      "Frank"  
    }else{  
      "Liz"  
    }  
  }  
}  
}else{  
  "Bob"  
}
```

```
## [1] "Mary"
```

Sometimes if and else can be quite cumbersome, and for special cases R comes with a neat `ifelse` command. This command takes the syntax

```
ifelse(LOGICAL,  
       IF TRUE: DO THIS,  
       ELSE: DO THIS)
```

This can be used on vectors of logicals in ways that don't make sense for the if-else constructs we used above. Let's try it:

```
test <- runif(10)  
print(test)
```

```
## [1] 0.17571876 0.81686901 0.92782836 0.56817635 0.82667148 0.03428013  
## [7] 0.98115622 0.50338785 0.66578606 0.78509240
```

```
ifelse(test < 1/2,  
       0,  
       1)
```

```
## [1] 0 1 1 1 1 0 1 1 1 1
```

As with if-else constructs we can also nest them

```
print(test)
```

```
## [1] 0.17571876 0.81686901 0.92782836 0.56817635 0.82667148 0.03428013  
## [7] 0.98115622 0.50338785 0.66578606 0.78509240
```

```
ifelse(test < 1/2,  
  ifelse(test < 1/3,  
    "Mary",  
    ifelse(test < 0.4,  
      "Frank",  
      "Liz")),  
  "Bob")
```

```
## [1] "Mary" "Bob" "Bob" "Bob" "Bob" "Mary" "Bob" "Bob" "Bob" "Bob"
```

Were you able to predict them all correctly? If you did then you understand what's going on here.

## 1.9 Scripting and functions

Now that we're starting to get the hang of doing things in R we're now at the point where we'll want to write them down so we can redo and replicate our work. Everything you do in R should take the form of a script (e.g., problem sets, research), so that you can edit, reproduce, remember what you've done, and share it with others. Our first script will be a program that generates some data and then provides some descriptive statistics of that data. To create a new script file in R go to `file>New script`. In RStudio go to `file>New>R Script`. In both cases we now have a blank file. Save this file somewhere (remember where) as "test1.R" and then enter the following

```
#####Heading#####  
##File: test1.R  
##Description: First R script  
  
#####Generate some data#####  
dat <- rnorm(1000) ##Creates a vector of normal draws  
  
#####Create a function to summarize it#####  
summarize <- function(x){ ##This creates a function that takes one  
  ##Argument, we've called x, it can be anything
```

```

##Make a vector with summary stats
ans <- c(Mean = mean(x),
        StDev = sd(x),
        Min = min(x),
        Median = median(x) ,
        Max = max(x))

return(ans) ## Return the list we created
} ## end the function
summarize(dat) ##run the function on the data

```

Once you have that typed, re-save the file. We can now run the file using the `source` command. To do this you'll want to have your working directory set to wherever you saved the file. You can set your working directory using `setwd()`

```

getwd() ##Returns the current working directory

## [1] "/home/cox/Dropbox/Rcourse_2021update"

setwd('~/.Dropbox/Rcourse_2021update') ##Change
getwd() ##Returns the new directory

```

```
## [1] "/home/cox/Dropbox/Rcourse_2021update"
```

**Note** All R scripts should be written with a working directory in mind and use "relative" rather than "absolute" paths. You should also never include a 'setwd' command in your scripts. When you send a script or project to someone it should be self contained in the sense that they should be able to download it and run it from whatever directory they save it to.

In my case this means that I set my working directory and then run:

```

source('test1.R', echo=TRUE)

##
## > dat <- rnorm(1000)
##
## > summarize <- function(x) {
## +   ans <- c(Mean = mean(x), StDev = sd(x), Min = min(x), Median = median(x),
## +           Max = max(x))
## +   return(ans .... [TRUNCATED]

```

```
##
## > summarize(dat)
##           Mean      StDev      Min      Median      Max
## -0.02238367  0.98726872 -3.87672640 -0.02719171  2.73763540
```

Alternatively you can run individual lines by highlight them in the file editor and press ctrl+enter (or command+enter). RStudio also has a source button built into the editor. We can also dispense with the full extension by changing our working directory.

Now that we've sourced the file the variable `dat` and the function `summarize` are now in our working space. To see this

```
ls()

## [1] "dat"      "i"        "matrixList" "mean.x"    "summarize"
## [6] "test"     "x"        "X"          "x2"        "y"
## [11] "Y"       "z"        "Z"
```

Which means we can now use our `summarize` function just like any of the built in R commands. For example

```
X <- cbind(rnorm(1000), 1:1000)
apply(X, 2, summarize)

##           [,1]      [,2]
## Mean   -0.01501621  500.5000
## StDev   1.03626887  288.8194
## Min    -3.85528930   1.0000
## Median -0.02429488  500.5000
## Max     3.32742782 1000.0000
```

## 1.10 Packages and updating

To install a package (in this case MASS) from CRAN (99.9% of the packages you want will be here) you just run the command

```
install.packages('MASS')
```

it may ask you to pick a mirror, it really doesn't matter which one you pick. Once it's installed you can load it.



```
library(MASS)
```

### 1.10.1 Updating R and R packages

To update R there are 3 steps

1. Download the new version
2. Install it
3. Uninstall the old version

In most cases that's all you'll need to do.

To update a package just run `install.packages()` again. RStudio has a button in the packages tab that says 'Check for Updates.'

## 1.11 Getting help

This is probably the most important part of the whole course. If you run into a problem, which will happen often, there are two things that are almost always true:

1. Someone else has had this problem
2. Someone has solved it.

**Finding out about a particular function:** The most common problems are related to particular functions that you want to know more about. In these cases the best place to start looking is the R help file. These can be accessed using the `?` command. For instance if we wanted to know more about the arguments in `log`, say we didn't know that it was base  $e$  or we didn't know how to change it we could type

```
?log
```

Which pulls up the help file. A typical R help file consists of a few sections

- **Description** What is the function supposed to do?
- **Usage** How does one typically type the command?
- **Arguments** What are all the arguments and what do they do?
- **Details** Additional information about how the function works
- **Value** What does the function return? If the function returns a list, what are the elements of that list?
- **See Also** Related functions that may be helpful
- **Example** Examples of how to use the function.

This is usually good enough to figure anything you want to know about a function, and running the examples at the bottom of the page can be helpful in understanding the output. Note that if for some reason `?` doesn't work you can also use type

```
help('%*%')
```

and it will do the same thing.

**You know what you want to do, but you don't know what function to use:** In these cases the commands `??` or `help.search` are your friends. They do a keyword search through the help files or all your packages to find what you're looking for. For example,

```
help.search("multivariate normal")
```

Searches the help files for mentions of multivariate normal. One result that looks promising is

```
MASS::mvrnorm    Simulate from a Multivariate Normal Distribution
```

Which means that there is a function in the MASS package called `mvrnorm`.

**If neither of those works:**

1. Google will almost certainly find you the answer you want. Googling 'How to do XYZ in R' will almost always guide you to the right place. There are few websites that deal with R questions and the answers are almost always helpful. Results from [www.stackoverflow.com](http://www.stackoverflow.com) are usually very helpful and easy to follow.
2. ChatGPT and similar AI can also be very helpful at answering your questions.

## 1.12 Exercises (break)

1. Look up the function `rnorm` using the `?` function. Read about its arguments and its related functions (`pnorm`, `dnorm`, etc), we will use it in the next problem.
2. Create a script file that does the following
  - a. Create a  $15 \times 3$  matrix, call it `X` where the first column is all 1s, the second column contains random draws from a normal with mean 1 and standard deviation 2 (hint: look at problem 1) and the last column contains random draws from the uniform distribution  $[0, 1]$  (use `??` or google to try and find the function for this). Use any of the methods discussed above to create the matrix. Look up and use the function `colMeans` to print the column means for each column and use `apply` to print standard deviations of each column to make sure you that you did this correctly (the standard deviation for  $U[0, 1]$  will be between 0.27 and 0.30)

- b. Create a vector `b` equal to `(-1, 2, 2)`. Then change the second value to `-2`.
  - c. Use `if` or `ifelse` commands to replace every negative value in `b` with `0`.
3. Install the following packages:
- `dplyr`
  - `readstata13`
  - `xtable`
  - `ggplot2`

## 2 Data Frames

In this section, we'll be focusing on one particular type of object: the data frame. Data frames in R are used for data manipulation and data analysis because they offer a few advantages over the standard matrix, the advantages that they offer are:

- Each column in a data frame can be of a different class (numeric, character, factor). All the columns in a matrix must be the same class (numeric, character).
- Data frames can be merged together, the `merge` command doesn't work on matrices
- Most canned regression models are designed to work with data frames rather than matrices
- It's easier to extract individual variables out of a data frame

Because data frames are pretty essential to most applications of R we'll be doing a lot of specific applications. Two common add-ons to data frames are data tables the tidyverse. We will briefly discuss these throughout, but our main focus will be on base R data frames since they are the work horse. You will at some point want to supplement your knowledge by using either `tidyr` or data tables (or both).

### 2.1 Reading data

One advantage of R over other statistical packages is that it has the ability to read many different kinds of data. The two standard read commands are for tab and comma separated data and they are `read.table` and `read.csv`, respectively. It's easy to save excel files into comma separated data (.csv), and I would recommend this over using tools explicitly designed for excel files. For many purposes the combination of `read.csv` will get you where you want to go however, there are lots of times when the data can only be obtained in Stata (.dta) or other proprietary formats.

The `foreign` package allows for reading older Stata files only, but it does allow for SAS,

SPSS, S+, minitab, .dbf files (GIS data is often in .dbf form) and other data formats, so you may also find that useful. For newer Stata files I like `readstata13`, but the `haven` package also works.

Other packages you may come across for reading data include: `readr` for most text data (csv, txt, tab), and `readxl` package for excel-style files (xls and xlsx). In addition to reading data from outside sources, many R packages (including the base packages) come prepackaged with datasets which can be accessed using the `data` function

```
# Tidy packages
library(dplyr)

# for stata files
library(readstata13)

##Notice that I use relative paths below. You should use the setwd()
##command that we learned before to change your working directory to
##directory that contains the datasets folder **before** trying these examples

##ordinary csv
NMC <- read.csv('Datasets/NMC_Supplement_v4_0.csv')

##stata dataset
FL2003 <- read.dta13('Datasets/FearonLaitin_CivilWar2003.dta')

## Warning in read.dta13("Datasets/FearonLaitin_CivilWar2003.dta"):
##   Factor codes of type double or float detected in variables
##
##   region
##
##   No labels have been assigned.
##   Set option 'nonint.factors = TRUE' to assign labels anyway.

# A warning. Let's do what it says
FL2003 <- read.dta13('Datasets/FearonLaitin_CivilWar2003.dta',
                    nonint.factors=TRUE,
                    convert.dates = FALSE) #annoying change in newer versions
```

```
class(NMC)
```

```
## [1] "data.frame"
```

```
class(FL2003)
```

```
## [1] "data.frame"
```

Notice that the class here is `data.frame` which is what we're into. Now we've read in the data we can take a look at it.

## 2.2 Commands to use on Data

### 2.2.1 Looking at the Variables

Once we've read in the data we may wish to look at it. This can be accomplished using the `View` command. This command opens up a new window where we can see the data just like we would using the `browse` command in Stata, there is also the command `fix` which is the equivalent of the `edit` command in Stata.

```
View(NMC)
```

```
fix(NMC)
```

There is also an easy way to just look at the first few observations of a `data.frame`. This is helpful just to see what the variables look like without actually looking at the whole dataset. This can be done using the `head` command. Additionally, the command `summary` can be used to get a summary of each column in the data frame; we can also look at just the variable names using the command `'colnames'`

```
head(FL2003) #Top 6
```

```
##  politycode year polity2 country cname cmark wars war warl onset ethonset
## 1          2 1945      10     USA  USA      1    0  0    0      0          0
## 2          2 1946      10     USA  USA      0    0  0    0      0          0
## 3          2 1947      10     USA  USA      0    0  0    0      0          0
## 4          2 1948      10     USA  USA      0    0  0    0      0          0
## 5          2 1949      10     USA  USA      0    0  0    0      0          0
## 6          2 1950      10     USA  USA      0    0  0    0      0          0
##  durest aim casename ended ethwar waryrs    pop      lpop gdpen gdptype gdpenl
## 1    NA  NA          NA    NA          140969 11.85630 7.626      3  7.626
## 2    NA  NA          NA    NA          141936 11.86313 7.654      3  7.626
```

```

## 3      NA  NA      NA      NA      142713 11.86859 8.025      3 7.654
## 4      NA  NA      NA      NA      145326 11.88673 8.270      3 8.025
## 5      NA  NA      NA      NA      147987 11.90488 8.040      3 8.270
## 6      NA  NA      NA      NA      152273 11.93343 8.772      0 8.040
##  lgdpenl1  lpopl1      region western eeuroplamerica
## 1 8.939319 11.85630 western democracies and japan      1      0      0
## 2 8.939319 11.85630 western democracies and japan      1      0      0
## 3 8.942984 11.86313 western democracies and japan      1      0      0
## 4 8.990317 11.86859 western democracies and japan      1      0      0
## 5 9.020390 11.88673 western democracies and japan      1      0      0
## 6 8.992185 11.90488 western democracies and japan      1      0      0
##  ssafrica asia nafrme colbrit colfra mtnest  lmtnest elevdiff Oil ncontig
## 1      0      0      0      1      0 23.9 3.214868      6280      0      1
## 2      0      0      0      1      0 23.9 3.214868      6280      0      1
## 3      0      0      0      1      0 23.9 3.214868      6280      0      1
## 4      0      0      0      1      0 23.9 3.214868      6280      0      1
## 5      0      0      0      1      0 23.9 3.214868      6280      0      1
## 6      0      0      0      1      0 23.9 3.214868      6280      0      1
##  ethfrac      ef plural second numlang relfrac plurrel minrelpc muslim
## 1 0.3569501 0.490957 0.691 0.125      3 0.596      56      28      1.9
## 2 0.3569501 0.490957 0.691 0.125      3 0.596      56      28      1.9
## 3 0.3569501 0.490957 0.691 0.125      3 0.596      56      28      1.9
## 4 0.3569501 0.490957 0.691 0.125      3 0.596      56      28      1.9
## 5 0.3569501 0.490957 0.691 0.125      3 0.596      56      28      1.9
## 6 0.3569501 0.490957 0.691 0.125      3 0.596      56      28      1.9
##  nwstate polity2l instab anocl deml ccode
## 1      0      10      0      0      1      2
## 2      0      10      0      0      1      2
## 3      0      10      0      0      1      2
## 4      0      10      0      0      1      2
## 5      0      10      0      0      1      2
## 6      0      10      0      0      1      2

```

```
tail(FL2003) #Last 6
```

```

##      politycode year polity2 country cname cmark wars war warl onset ethonset
## 6605      950 1994      5      FIJI FIJI      0      0      0      0      0      0

```

## 6606	950	1995	5	FIJI	FIJI	0	0	0	0	0	0
## 6607	950	1996	5	FIJI	FIJI	0	0	0	0	0	0
## 6608	950	1997	5	FIJI	FIJI	0	0	0	0	0	0
## 6609	950	1998	5	FIJI	FIJI	0	0	0	0	0	0
## 6610	950	1999	6	FIJI	FIJI	0	0	0	0	0	0
##	durest	aim	casename	ended	ethwar	waryrs	pop	lpop	gdpen	gdptype	
## 6605	NA	NA		NA	NA		784.00	6.664409	4.278853		2
## 6606	NA	NA		NA	NA		794.00	6.677083	4.313088		2
## 6607	NA	NA		NA	NA		803.00	6.688354	4.427134		2
## 6608	NA	NA		NA	NA		814.65	6.702759	4.309664		2
## 6609	NA	NA		NA	NA		827.19	6.718034	4.210803		2
## 6610	NA	NA		NA	NA		NA	NA	4.479345		2
##	gdpenl	lgdpenl1	lpopl1	region	western	eeurop	lamerica	ssafrica	asia		
## 6605	4.149	8.330654	6.647688	asia	0	0	0	0	1		
## 6606	4.279	8.361441	6.664409	asia	0	0	0	0	1		
## 6607	4.313	8.369410	6.677083	asia	0	0	0	0	1		
## 6608	4.427	8.395508	6.688354	asia	0	0	0	0	1		
## 6609	4.310	8.368615	6.702759	asia	0	0	0	0	1		
## 6610	4.211	8.345408	6.718034	asia	0	0	0	0	1		
##	nafrme	colbrit	colfra	mtnest	lmtnest	elevdiff	Oil	ncontig	ethfrac		
## 6605	0	1	0	0.4	0.3364722	1324	0	1	0.7105385		
## 6606	0	1	0	0.4	0.3364722	1324	0	1	0.7105385		
## 6607	0	1	0	0.4	0.3364722	1324	0	1	0.7105385		
## 6608	0	1	0	0.4	0.3364722	1324	0	1	0.7105385		
## 6609	0	1	0	0.4	0.3364722	1324	0	1	0.7105385		
## 6610	0	1	0	0.4	0.3364722	1324	0	1	0.7105385		
##	ef	plural	second	numlang	relfrac	plurrel	minrelpc	muslim	nwstate		
## 6605	0.5657309	0.49	0.44	6	0.7002	38	37	8	0		
## 6606	0.5657309	0.49	0.44	6	0.7002	38	37	8	0		
## 6607	0.5657309	0.49	0.44	6	0.7002	38	37	8	0		
## 6608	0.5657309	0.49	0.44	6	0.7002	38	37	8	0		
## 6609	0.5657309	0.49	0.44	6	0.7002	38	37	8	0		
## 6610	0.5657309	0.49	0.44	6	0.7002	38	37	8	0		
##	polity2l	instab	anocl	deml	cocode						
## 6605	5	0	1	0	950						
## 6606	5	0	1	0	950						

```
## 6607      5      0      1      0    950
## 6608      5      0      1      0    950
## 6609      5      0      1      0    950
## 6610      5      0      1      0    950
```

```
summary(FL2003[, 1:10]) ##Truncated the first 10 columns to save space
```

```
##      politycode      year      polity2      country
## Min.   :  2.0   Min.   :1945   Min.   : -10.0000   Length:6610
## 1st Qu.:230.0   1st Qu.:1964   1st Qu.:  -7.0000   Class :character
## Median :451.0   Median :1977   Median :  -3.0000   Mode  :character
## Mean   :450.6   Mean   :1976   Mean    : -0.4377
## 3rd Qu.:663.0   3rd Qu.:1989   3rd Qu.:   8.0000
## Max.   :950.0   Max.   :1999   Max.    : 10.0000
##
##                      NA's      :62
##      cname      cmark      wars      war
## Length:6610   Min.    :0.00000   Min.    :0.0000   Min.    :0.0000
## Class :character 1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000
## Mode  :character Median :0.00000   Median :0.0000   Median :0.0000
##                      Mean    :0.02436   Mean    :0.1552   Mean    :0.1389
##                      3rd Qu.:0.00000   3rd Qu.:0.0000   3rd Qu.:0.0000
##                      Max.    :1.00000   Max.    :4.0000   Max.    :1.0000
##
##      warl      onset
## Min.    :0.0000   Min.    :0.00000
## 1st Qu.:0.0000   1st Qu.:0.00000
## Median :0.0000   Median :0.00000
## Mean    :0.1346   Mean    :0.01679
## 3rd Qu.:0.0000   3rd Qu.:0.00000
## Max.    :1.0000   Max.    :1.00000
##
```

```
colnames(FL2003)
```

```
## [1] "politycode" "year"      "polity2"   "country"   "cname"
## [6] "cmark"      "wars"      "war"       "warl"      "onset"
## [11] "ethonset"   "durest"    "aim"       "casename"   "ended"
## [16] "ethwar"     "waryrs"    "pop"       "lpop"      "gdpen"
```



```
## [21] "gdptype"      "gdpenl"      "lgdpenl1"    "lpopl1"      "region"
## [26] "western"      "eeurop"      "lamerica"    "ssafrica"    "asia"
## [31] "nafrme"       "colbrit"     "colfra"      "mtnest"      "lmtnest"
## [36] "elevdiff"     "Oil"         "ncontig"     "ethfrac"     "ef"
## [41] "plural"       "second"      "numlang"     "relfrac"     "plurrel"
## [46] "minrelpc"     "muslim"      "nwstate"     "polity2l"    "instab"
## [51] "anocl"        "deml"        "ccode"
```

It's worth noting at this point that all of commands just mentioned work on matrices, and everything but `colnames` works on ordinary vectors.

## 2.2.2 Individual Variables

R treats data frames like a special version of a list. This means that to access individual elements we use the dollar sign. For example if we want just the summary of the `pop` variables in Fearon and Laitin we would type.

```
summary(FL2003$pop)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.   NA's
##      222   3217   8137   31787   20601 1238599    177
```

We could also use numbers to index like with matrices

```
summary(FL2003[,18]) ##But isn't the dollar sign easier?
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.   NA's
##      222   3217   8137   31787   20601 1238599    177
```

Extracted variables are just vectors and so we can treat them as such

```
## Doing vector stuff with variables
```

```
FL2003$pop[1:10]
```

```
## [1] 140969 141936 142713 145326 147987 152273 155000 157727 160475 163202
```

```
head(log(FL2003$pop))
```

```
## [1] 11.85630 11.86313 11.86859 11.88673 11.90488 11.93343
```

### 2.2.3 Creating Subsets

We can also use index to create subsets of data frames, for instance if we just wanted the COW codes and years we could do any of the following to create that subset.

```
##These all do the same thing
```

```
temp.dat <- FL2003[, c('ccode', 'year')]  
head(temp.dat)
```

```
##   ccode year  
## 1     2 1945  
## 2     2 1946  
## 3     2 1947  
## 4     2 1948  
## 5     2 1949  
## 6     2 1950
```

```
temp.dat <- FL2003[, c(53, 2)]  
head(temp.dat)
```

```
##   ccode year  
## 1     2 1945  
## 2     2 1946  
## 3     2 1947  
## 4     2 1948  
## 5     2 1949  
## 6     2 1950
```

```
temp.dat <- subset(FL2003, select=c('ccode', 'year'))  
head(temp.dat)
```

```
##   ccode year  
## 1     2 1945  
## 2     2 1946  
## 3     2 1947  
## 4     2 1948  
## 5     2 1949  
## 6     2 1950
```

```
## Tidy approach
temp.dat <- FL2003 %>%
  select(ccode, year)
head(temp.dat)
```

```
##  ccode year
## 1      2 1945
## 2      2 1946
## 3      2 1947
## 4      2 1948
## 5      2 1949
## 6      2 1950
```

Note that we have introduced the tidy %>% function. This operator connects functions in the tidyverse. Instead of `f(g(x))` we write `x %>% g() %>% f()` which can make for more readable code as it goes in the order of operation.

We can also subset based on rows

```
##These all do the same thing
temp.dat <- FL2003[FL2003$ccode ==2, ] ##Extract USA
head(temp.dat)
```

```
##  politycode year polity2 country cname cmark wars war warl onset ethonset
## 1      2 1945      10     USA  USA      1    0  0  0    0      0
## 2      2 1946      10     USA  USA      0    0  0  0    0      0
## 3      2 1947      10     USA  USA      0    0  0  0    0      0
## 4      2 1948      10     USA  USA      0    0  0  0    0      0
## 5      2 1949      10     USA  USA      0    0  0  0    0      0
## 6      2 1950      10     USA  USA      0    0  0  0    0      0
##  durest aim casename ended ethwar waryrs      pop      lpop gdpen gdptype gdpenl
## 1    NA  NA      NA      NA      NA      140969 11.85630 7.626      3  7.626
## 2    NA  NA      NA      NA      NA      141936 11.86313 7.654      3  7.626
## 3    NA  NA      NA      NA      NA      142713 11.86859 8.025      3  7.654
## 4    NA  NA      NA      NA      NA      145326 11.88673 8.270      3  8.025
## 5    NA  NA      NA      NA      NA      147987 11.90488 8.040      3  8.270
## 6    NA  NA      NA      NA      NA      152273 11.93343 8.772      0  8.040
##  lgdpenl1  lpopl1      region western eeuroplamerica
```

```
## 1 8.939319 11.85630 western democracies and japan      1      0      0
## 2 8.939319 11.85630 western democracies and japan      1      0      0
## 3 8.942984 11.86313 western democracies and japan      1      0      0
## 4 8.990317 11.86859 western democracies and japan      1      0      0
## 5 9.020390 11.88673 western democracies and japan      1      0      0
## 6 8.992185 11.90488 western democracies and japan      1      0      0
##   ssafrica asia nafrme colbrit colfra mtnest  lmtnest elevdiff Oil ncontig
## 1      0      0      0      1      0 23.9 3.214868    6280 0      1
## 2      0      0      0      1      0 23.9 3.214868    6280 0      1
## 3      0      0      0      1      0 23.9 3.214868    6280 0      1
## 4      0      0      0      1      0 23.9 3.214868    6280 0      1
## 5      0      0      0      1      0 23.9 3.214868    6280 0      1
## 6      0      0      0      1      0 23.9 3.214868    6280 0      1
##   ethfrac      ef plural second numlang relfrac plurrel minrelpc muslim
## 1 0.3569501 0.490957 0.691 0.125      3 0.596      56      28 1.9
## 2 0.3569501 0.490957 0.691 0.125      3 0.596      56      28 1.9
## 3 0.3569501 0.490957 0.691 0.125      3 0.596      56      28 1.9
## 4 0.3569501 0.490957 0.691 0.125      3 0.596      56      28 1.9
## 5 0.3569501 0.490957 0.691 0.125      3 0.596      56      28 1.9
## 6 0.3569501 0.490957 0.691 0.125      3 0.596      56      28 1.9
##   nwstate polity2l instab anocl deml ccode
## 1      0      10      0      0      1      2
## 2      0      10      0      0      1      2
## 3      0      10      0      0      1      2
## 4      0      10      0      0      1      2
## 5      0      10      0      0      1      2
## 6      0      10      0      0      1      2
```

```
temp.dat <- subset(FL2003, subset = ccode==2)
head(temp.dat)
```

```
##   politycode year polity2 country cname cmark wars war warl onset ethonset
## 1      2 1945      10      USA  USA      1      0      0      0      0      0
## 2      2 1946      10      USA  USA      0      0      0      0      0      0
## 3      2 1947      10      USA  USA      0      0      0      0      0      0
## 4      2 1948      10      USA  USA      0      0      0      0      0      0
## 5      2 1949      10      USA  USA      0      0      0      0      0      0
```

## 6	2	1950	10	USA	USA	0	0	0	0	0	0
##	durest	aim	casename	ended	ethwar	waryrs	pop	lpop	gdpen	gdptype	gdpenl
## 1	NA	NA	NA	NA			140969	11.85630	7.626	3	7.626
## 2	NA	NA	NA	NA			141936	11.86313	7.654	3	7.626
## 3	NA	NA	NA	NA			142713	11.86859	8.025	3	7.654
## 4	NA	NA	NA	NA			145326	11.88673	8.270	3	8.025
## 5	NA	NA	NA	NA			147987	11.90488	8.040	3	8.270
## 6	NA	NA	NA	NA			152273	11.93343	8.772	0	8.040
##	lgdpenl1	lpopl1					region	western	eeurop	lamerica	
## 1	8.939319	11.85630	western	democracies	and	japan		1	0	0	
## 2	8.939319	11.85630	western	democracies	and	japan		1	0	0	
## 3	8.942984	11.86313	western	democracies	and	japan		1	0	0	
## 4	8.990317	11.86859	western	democracies	and	japan		1	0	0	
## 5	9.020390	11.88673	western	democracies	and	japan		1	0	0	
## 6	8.992185	11.90488	western	democracies	and	japan		1	0	0	
##	ssafrica	asia	nafrme	colbrit	colfra	mtnest	lmtnest	elevdiff	Oil	ncontig	
## 1	0	0	0	1	0	23.9	3.214868	6280	0	1	
## 2	0	0	0	1	0	23.9	3.214868	6280	0	1	
## 3	0	0	0	1	0	23.9	3.214868	6280	0	1	
## 4	0	0	0	1	0	23.9	3.214868	6280	0	1	
## 5	0	0	0	1	0	23.9	3.214868	6280	0	1	
## 6	0	0	0	1	0	23.9	3.214868	6280	0	1	
##	ethfrac		ef	plural	second	numlang	relfrac	plurrel	minrelpc	muslim	
## 1	0.3569501	0.490957	0.691	0.125		3	0.596	56	28	1.9	
## 2	0.3569501	0.490957	0.691	0.125		3	0.596	56	28	1.9	
## 3	0.3569501	0.490957	0.691	0.125		3	0.596	56	28	1.9	
## 4	0.3569501	0.490957	0.691	0.125		3	0.596	56	28	1.9	
## 5	0.3569501	0.490957	0.691	0.125		3	0.596	56	28	1.9	
## 6	0.3569501	0.490957	0.691	0.125		3	0.596	56	28	1.9	
##	nwstate	polity2l	instab	anocl	deml	cocode					
## 1	0	10	0	0	1	2					
## 2	0	10	0	0	1	2					
## 3	0	10	0	0	1	2					
## 4	0	10	0	0	1	2					
## 5	0	10	0	0	1	2					
## 6	0	10	0	0	1	2					

```
#tidy
```

```
temp.dat <- FL2003 %>%
```

```
  filter(ccode==2)
```

```
head(temp.dat)
```

```
## politycode year polity2 country cname cmark wars war warl onset ethonset
## 1          2 1945        10     USA  USA      1    0    0    0    0    0
## 2          2 1946        10     USA  USA      0    0    0    0    0    0
## 3          2 1947        10     USA  USA      0    0    0    0    0    0
## 4          2 1948        10     USA  USA      0    0    0    0    0    0
## 5          2 1949        10     USA  USA      0    0    0    0    0    0
## 6          2 1950        10     USA  USA      0    0    0    0    0    0
## durest aim casename ended ethwar waryrs      pop      lpop gdpen gdptype gdpenl
## 1      NA  NA              NA      NA      140969 11.85630 7.626      3 7.626
## 2      NA  NA              NA      NA      141936 11.86313 7.654      3 7.626
## 3      NA  NA              NA      NA      142713 11.86859 8.025      3 7.654
## 4      NA  NA              NA      NA      145326 11.88673 8.270      3 8.025
## 5      NA  NA              NA      NA      147987 11.90488 8.040      3 8.270
## 6      NA  NA              NA      NA      152273 11.93343 8.772      0 8.040
## lgdpenl1  lpopl1              region western eeuroplamerica
## 1 8.939319 11.85630 western democracies and japan      1    0    0
## 2 8.939319 11.85630 western democracies and japan      1    0    0
## 3 8.942984 11.86313 western democracies and japan      1    0    0
## 4 8.990317 11.86859 western democracies and japan      1    0    0
## 5 9.020390 11.88673 western democracies and japan      1    0    0
## 6 8.992185 11.90488 western democracies and japan      1    0    0
## ssafrica asia nafrme colbrit colfra mtnest lmtnest elevdiff Oil ncontig
## 1      0    0    0      1    0 23.9 3.214868      6280 0    1
## 2      0    0    0      1    0 23.9 3.214868      6280 0    1
## 3      0    0    0      1    0 23.9 3.214868      6280 0    1
## 4      0    0    0      1    0 23.9 3.214868      6280 0    1
## 5      0    0    0      1    0 23.9 3.214868      6280 0    1
## 6      0    0    0      1    0 23.9 3.214868      6280 0    1
## ethfrac      ef plural second numlang relfrac plurrel minrelpc muslim
## 1 0.3569501 0.490957 0.691 0.125      3 0.596      56      28 1.9
## 2 0.3569501 0.490957 0.691 0.125      3 0.596      56      28 1.9
```

```
## 3 0.3569501 0.490957 0.691 0.125      3 0.596      56      28      1.9
## 4 0.3569501 0.490957 0.691 0.125      3 0.596      56      28      1.9
## 5 0.3569501 0.490957 0.691 0.125      3 0.596      56      28      1.9
## 6 0.3569501 0.490957 0.691 0.125      3 0.596      56      28      1.9
##   nwstate polity2l instab anocl deml ccode
## 1      0      10      0      0      1      2
## 2      0      10      0      0      1      2
## 3      0      10      0      0      1      2
## 4      0      10      0      0      1      2
## 5      0      10      0      0      1      2
## 6      0      10      0      0      1      2
```

Note that `subset` is used in base for both rows and columns. If we pull up the help page on `subset` (`?subset`) we can see that the `subset` argument takes a logical expression (in this case `ccode==2`) for selecting rows that we want. The `select` argument takes column names for the columns that we want. We can use them to together

```
temp.dat <- subset(FL2003,
                  subset = ccode==2,
                  select=c('year', 'polity2'))
head(temp.dat)
```

```
##   year polity2
## 1 1945      10
## 2 1946      10
## 3 1947      10
## 4 1948      10
## 5 1949      10
## 6 1950      10
```

```
dim(temp.dat)
```

```
## [1] 55  2
```

```
#tidy approach
temp.dat <- FL2003 %>%
  filter(ccode==2) %>%
  select(year, polity2)
head(temp.dat)
```

```
##   year polity2
## 1 1945      10
## 2 1946      10
## 3 1947      10
## 4 1948      10
## 5 1949      10
## 6 1950      10
```

```
dim(temp.dat)
```

```
## [1] 55  2
```

## 2.2.4 Classes

One thing you might have noticed when we ran `summary()` on the Fearon and Laitin data is that not all variables looked the same. For instance if we run

```
temp.df <- subset(FL2003, select=c(ccode, cname, region))
summary(temp.df)
```

```
##      ccode      cname
## Min.   :  2.0   Length:6610
## 1st Qu.:230.0   Class :character
## Median :451.0   Mode  :character
## Mean   :450.6
## 3rd Qu.:663.0
## Max.   :950.0
##
##                      region
## western democracies and japan      :1155
## e. europe and the former soviet union: 646
## asia                               :1096
## n. africa and the middle east      : 910
## sub-saharan africa                 :1593
## latin america and the caribbean    :1210
```

```
lapply(temp.df, class) ##lapply because it's really a type of list
```

```
## $ccode
## [1] "numeric"
##
```



```
## $cname
## [1] "character"
##
## $region
## [1] "factor"
```

We can see that we have a numeric variable, a character variable, and a factor variable. In general, R assigns these classes when we read the data, and most of the time it gets it right. Numeric and integer variables are variables that are all numbers. These are ordinary variables, they can be either continuous (population) or discrete (year) and R won't notice the difference. Everything we covered with numeric vectors last time works on these. Character variables are just strings. There's not too much special we can or would want to do with these. Factors, however, are an interesting construct.

**More on Factors** Factors are how R deals with categorical variables. In the Fearon and Laitin example region is stored as a factor. Running `summary` on a factor variable returns a table with a count of each category.

```
summary(FL2003$region)
```

```
##          western democracies and japan e. europe and the former soviet union
##                                1155                                646
##          asia                                n. africa and the middle east
##                                1096                                910
##          sub-saharan africa                                latin america and the caribbean
##                                1593                                1210
```

```
head(FL2003$region) ##includes info about the levels
```

```
## [1] western democracies and japan western democracies and japan
## [3] western democracies and japan western democracies and japan
## [5] western democracies and japan western democracies and japan
## 6 Levels: western democracies and japan ...
```

```
levels(FL2003$region) ##Just want to know the levels
```

```
## [1] "western democracies and japan"
## [2] "e. europe and the former soviet union"
## [3] "asia"
## [4] "n. africa and the middle east"
```

```
## [5] "sub-saharan africa"
## [6] "latin america and the caribbean"
```

```
nlevels(FL2003$region) ##Just want to the number of levels
```

```
## [1] 6
```

The first level is always considered the reference level (and dropped in regression). Factors can be troublesome when manipulating data. To get around this you may sometimes want to convert factors to characters when doing any manipulation. For example if we want to subset the data to remove one level from a factor R will do that but it won't drop that as a level, which can mess things up.

```
temp.df <- subset(FL2003, region=='western democracies and japan')
summary(temp.df$region) ##others still listed
```

```
##          western democracies and japan e. europe and the former soviet union
##                                1155                                0
##                asia                n. africa and the middle east
##                                0                                0
##                sub-saharan africa                latin america and the caribbean
##                                0                                0
```

We can tell R to convert all factors to characters when we read in the data. Likewise R sometimes messes up and creates factors where we don't want them (it will sometimes read a numeric or a character in as a factor). We can easily change between classes. The only transformation we need to be careful with is with factors to numeric:

```
FL2003 %>%
  select(pop) %>%
  head()
```

```
##      pop
## 1 140969
## 2 141936
## 3 142713
## 4 145326
## 5 147987
## 6 152273
```

```
FL2003 %>% ##change from numeric to character
  mutate(pop=as.character(pop)) %>%
  select(pop) %>%
  head()
```

```
##      pop
## 1 140969
## 2 141936
## 3 142713
## 4 145326
## 5 147987
## 6 152273
```

```
FL2003 %>% ##change from numeric to character to factor
  mutate(pop=as.character(pop),
         pop=as.factor(pop)) %>%
  select(pop) %>%
  head()
```

```
##      pop
## 1 140969
## 2 141936
## 3 142713
## 4 145326
## 5 147987
## 6 152273
```

```
FL2003 %>% ##change from numeric to character to factor to numeric
  mutate(pop=as.character(pop),
         pop=as.factor(pop),
         pop=as.numeric(pop)) %>%
  select(pop) %>%
  head() #WHOOOPS
```

```
##      pop
## 1    820
## 2    838
## 3    853
```

```
## 4 900
## 5 942
## 6 1018
```

```
##R numbers them by the level their in,
##so the first level (222) is converted to 1
```

```
FL2003 %>% ##change from numeric to character to factor to numeric
  mutate(pop=as.character(pop),
         pop=as.factor(pop),
         pop=as.character(pop),
         pop=as.numeric(pop)) %>%
  select(pop) %>%
  head() #What a relief
```

```
##      pop
## 1 140969
## 2 141936
## 3 142713
## 4 145326
## 5 147987
## 6 152273
```

Useful transformations:

**Table 2:** Useful functions for Converting Objects

Function	Use
<code>as.numeric</code>	Change a factor or character vector into numbers
<code>as.character</code>	Change a numeric or factor vector into a character string
<code>as.Date</code>	Change a character vector of dates in a Date object
<code>as.factor</code>	Change a character or numeric vector in factor
<code>as.matrix</code>	Change a vector or data frame into a matrix
<code>as.data.frame</code>	Change a matrix into a data frame

We've now also introduced `mutate` as the tidy tool for making variables. More on this in a moment.

Factors can be helpful when we have well-defined groups and/or what to do analysis by

groups. Some factor work:

```
x <- 1:5
factor(x)
```

```
## [1] 1 2 3 4 5
## Levels: 1 2 3 4 5
```

```
factor(x, levels = 1:10) ##Add extra levels that aren't in x
```

```
## [1] 1 2 3 4 5
## Levels: 1 2 3 4 5 6 7 8 9 10
```

```
factor(x, labels = c('blue',
                     'red',
                     'green',
                     'yellow',
                     'pink')) ##Relabel x
```

```
## [1] blue   red    green  yellow pink
## Levels: blue red green yellow pink
```

## 2.3 Merging Data

The `merge` function in R is important enough to merit its own section, although it's relatively easy to do. The function takes two data.frames and joins them together based on one or more columns that the user supplies. Let's start with a simple example.

```
## Create two data frames
temp.df <- data.frame(ccode= 1:5,
                      Var1= rnorm(5))
```

```
temp.df
```

```
##   ccode    Var1
## 1     1 0.4915405
## 2     2 2.3880690
## 3     3 -0.3598627
## 4     4 1.7588204
## 5     5 -0.6255602
```

```
temp.df2 <- data.frame(ccode= 1:5,
                      Var2 = runif(5))

temp.df2
```

```
##   ccode      Var2
## 1     1 0.14125892
## 2     2 0.17983749
## 3     3 0.93499223
## 4     4 0.09054334
## 5     5 0.93959143
```

```
temp.df3 <- merge(temp.df,
                  temp.df2,
                  by='ccode') ##The variable we want to merge on

temp.df3 ##Ta Da
```

```
##   ccode      Var1      Var2
## 1     1 0.4915405 0.14125892
## 2     2 2.3880690 0.17983749
## 3     3 -0.3598627 0.93499223
## 4     4 1.7588204 0.09054334
## 5     5 -0.6255602 0.93959143
```

A slightly more complicated example might be

```
temp.df <- data.frame(cow.code= 1:5,
                     Var1= rnorm(5))

temp.df
```

```
##   cow.code      Var1
## 1         1 0.8239589
## 2         2 -0.4775666
## 3         3 -1.3252303
## 4         4 1.0561053
## 5         5 -0.5021448
```

```
temp.df2 <- data.frame(ccode= 1:5,
                      Var2 = runif(5))
temp.df2
```

```
##   ccode      Var2
## 1     1 0.04125228
## 2     2 0.11860259
## 3     3 0.92095722
## 4     4 0.92815488
## 5     5 0.64746372
```

*##We want to merge of country codes, but they have different names*  
*##Not to fear*

```
temp.df3 <- merge(temp.df,
                  temp.df2,
                  by.x='cow.code', ##.x refers to the 1st data.frame
                  by.y='ccode') ##.y refers to the 2nd
```

temp.df3 *##Ta Da*

```
##   cow.code      Var1      Var2
## 1         1 0.8239589 0.04125228
## 2         2 -0.4775666 0.11860259
## 3         3 -1.3252303 0.92095722
## 4         4  1.0561053 0.92815488
## 5         5 -0.5021448 0.64746372
```

An even more complex example

*###Data sets with different countries*

```
temp.df <- data.frame(cow.code= 1:10,
                      Var1= rnorm(10))
```

temp.df

```
##   cow.code      Var1
## 1         1 1.10611844
## 2         2  0.00448964
## 3         3 -0.61339358
```

```
## 4      4 -0.72952498
## 5      5  1.05324282
## 6      6 -1.83213562
## 7      7  0.26908781
## 8      8 -0.90308338
## 9      9 -1.31283561
## 10     10 -0.69725279
```

```
temp.df2 <- data.frame(ccode= c(1:5, 11:15),
                        Var2 = runif(5))
temp.df2
```

```
##      ccode      Var2
## 1      1 0.5833650
## 2      2 0.5188172
## 3      3 0.8576136
## 4      4 0.3835781
## 5      5 0.2354128
## 6     11 0.5833650
## 7     12 0.5188172
## 8     13 0.8576136
## 9     14 0.3835781
## 10    15 0.2354128
```

*##We want to merge of country codes, but they have different countries*

```
temp.df3 <- merge(temp.df,
                  temp.df2,
                  by.x='cow.code',
                  by.y='ccode')
```

*temp.df3 ##Note it only contains overlapping countries*

```
##      cow.code      Var1      Var2
## 1      1 1.10611844 0.5833650
## 2      2 0.00448964 0.5188172
## 3      3 -0.61339358 0.8576136
## 4      4 -0.72952498 0.3835781
## 5      5  1.05324282 0.2354128
```



```
##All the countries from just the first data.frame
```

```
merge(temp.df,  
      temp.df2,  
      by.x='cow.code',  
      by.y='ccode',  
      all.x=TRUE)
```

##	cow.code	Var1	Var2
## 1	1	1.10611844	0.5833650
## 2	2	0.00448964	0.5188172
## 3	3	-0.61339358	0.8576136
## 4	4	-0.72952498	0.3835781
## 5	5	1.05324282	0.2354128
## 6	6	-1.83213562	NA
## 7	7	0.26908781	NA
## 8	8	-0.90308338	NA
## 9	9	-1.31283561	NA
## 10	10	-0.69725279	NA

```
##Same for the 2nd
```

```
merge(temp.df,  
      temp.df2,  
      by.x='cow.code',  
      by.y='ccode',  
      all.y=TRUE)
```

##	cow.code	Var1	Var2
## 1	1	1.10611844	0.5833650
## 2	2	0.00448964	0.5188172
## 3	3	-0.61339358	0.8576136
## 4	4	-0.72952498	0.3835781
## 5	5	1.05324282	0.2354128
## 6	11	NA	0.5833650
## 7	12	NA	0.5188172
## 8	13	NA	0.8576136
## 9	14	NA	0.3835781
## 10	15	NA	0.2354128

```
##All from both
merge(temp.df,
      temp.df2,
      by.x='cow.code',
      by.y='ccode',
      all=TRUE)
```

##	cow.code	Var1	Var2
## 1	1	1.10611844	0.5833650
## 2	2	0.00448964	0.5188172
## 3	3	-0.61339358	0.8576136
## 4	4	-0.72952498	0.3835781
## 5	5	1.05324282	0.2354128
## 6	6	-1.83213562	NA
## 7	7	0.26908781	NA
## 8	8	-0.90308338	NA
## 9	9	-1.31283561	NA
## 10	10	-0.69725279	NA
## 11	11	NA	0.5833650
## 12	12	NA	0.5188172
## 13	13	NA	0.8576136
## 14	14	NA	0.3835781
## 15	15	NA	0.2354128

We can turn to the real data to show that we can match on more than one variable.

```
mergedData <- merge(FL2003,
                    NMC,
                    by=c('ccode', 'year'), ##Variables to match on
                    all.x=TRUE) ##Keep all the values from FL2003
```

More information on `merge` can be found in its help file. It's very flexible and fairly straight forward (although you may need to take a few tries before you get it right).

## 2.4 Generating New Variables

We may be in the situation of needing to create new variables that we want to add to our data frame.

In most cases this is pretty easy. For instance if we wanted might notice that the Fearon and Laitin data doesn't contain logged GDP per capita. To create that we could do the following

```
###Creates and attaches the new variable to the data frame
```

```
FL2003$log.gdpen <- log(FL2003$gdpen)
```

```
# tidy
```

```
FL2003 <- FL2003 %>%
```

```
  mutate(log.gdpen = log(gdpen))
```

### 2.4.1 Removing Variables

Removing variables is also straight forward. We can do it one at a time or with the subset command.

```
FL2003$random <- NULL ##Remove this variable
```

```
##The %in% command is a logical function that takes two vectors and  
##for each value of in the 1st vector it asks:  
##Is this value in the 2nd vector?
```

```
##Example of %in% Returns 2 TRUE value
```

```
colnames(FL2003) %in% c('politycode', 'casename')
```

```
## [1] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [13] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [49] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
# Tidy: start over
```

```
FL2003 <- read.dta13("Datasets/FearonLaitin_CivilWar2003.dta",  
                    convert.dates = FALSE,  
                    nonint.factors = TRUE)
```

```
c('politycode', 'casename') %in% colnames(FL2003)
```

```
## [1] TRUE TRUE
```

```
FL2003 <- FL2003 %>%  
  select(! c(politycode, casename))  
c('politycode', 'casename') %in% colnames(FL2003)
```

```
## [1] FALSE FALSE
```

## 2.5 Aggregating or summarizing Data

Sometimes you want to aggregate data to different levels or by groups. The thing to remember here: `mutate` is when you want a new variable the *same length as the input data* and `summarize` is when you want to aggregate to some level (e.g., smaller than the input).

```
###Generate some data###  
newDat <- data.frame(ccode=rep(1:5, each=10),  
                     year = rep(1:5, 10),  
                     Var1 = rnorm(50))  
  
## This is a data frame with 5 countries and five years  
## But each country-year has two observations.  
## We want to aggregate to a sum for each country year  
  
#tidy  
output <- newDat %>%  
  group_by(ccode, year) %>% #group on these variables  
  summarise(SumVar = sum(Var1)) %>%  
  ungroup() #ungroup to save yourself from future mistakes
```

```
## `summarise()` has grouped output by 'ccode'. You can override using the  
## `.groups` argument.
```

```
output
```

```
## # A tibble: 25 x 3  
##   ccode year SumVar  
##   <int> <int>   <dbl>  
## 1     1     1     1  1.03  
## 2     1     2  -1.28  
## 3     1     3  -0.327  
## 4     1     4   1.17
```

```
## 5      1      5 -0.839
## 6      2      1  2.49
## 7      2      2 -1.02
## 8      2      3  1.79
## 9      2      4 -1.17
## 10     2      5  0.0349
## # i 15 more rows
```

## 2.6 Writing Data

Once we have our data all set we may want to save it. All of the read functions we used to read have writing equivalents.

```
write.csv(NMC, 'Datasets/NMC.csv')
save(list=c('FL2003',
            'NMC'),
      file='Datasets/DataSets.rdata')
save.image('Datasets/DataFrames.Rdata')
```

The write functions create individual data frame files that can be opened by excel or Stata, whereas the .Rdata files are specific to R and can contain any number of objects. Also, `save` lets you save specific objects, and `save.image` saves your entire workspace.

```
ls() #Everything
```

```
## [1] "dat"      "FL2003"   "i"        "matrixList" "mean.x"
## [6] "mergedData" "newDat"   "NMC"      "output"     "summarize"
## [11] "temp.dat"   "temp.df"  "temp.df2" "temp.df3"   "test"
## [16] "x"         "X"        "x2"       "y"          "Y"
## [21] "z"         "Z"
```

```
rm(list=ls())
```

```
ls() #Nothing
```

```
## character(0)
```

```
load('Datasets/DataFrames.Rdata')
```

```
ls() #It's all back
```

```
## [1] "dat"      "FL2003"   "i"        "matrixList" "mean.x"
```

```
## [6] "mergedData" "newDat"      "NMC"          "output"       "summarize"
## [11] "temp.dat"    "temp.df"     "temp.df2"     "temp.df3"     "test"
## [16] "x"           "X"           "x2"           "y"            "Y"
## [21] "z"           "Z"
```

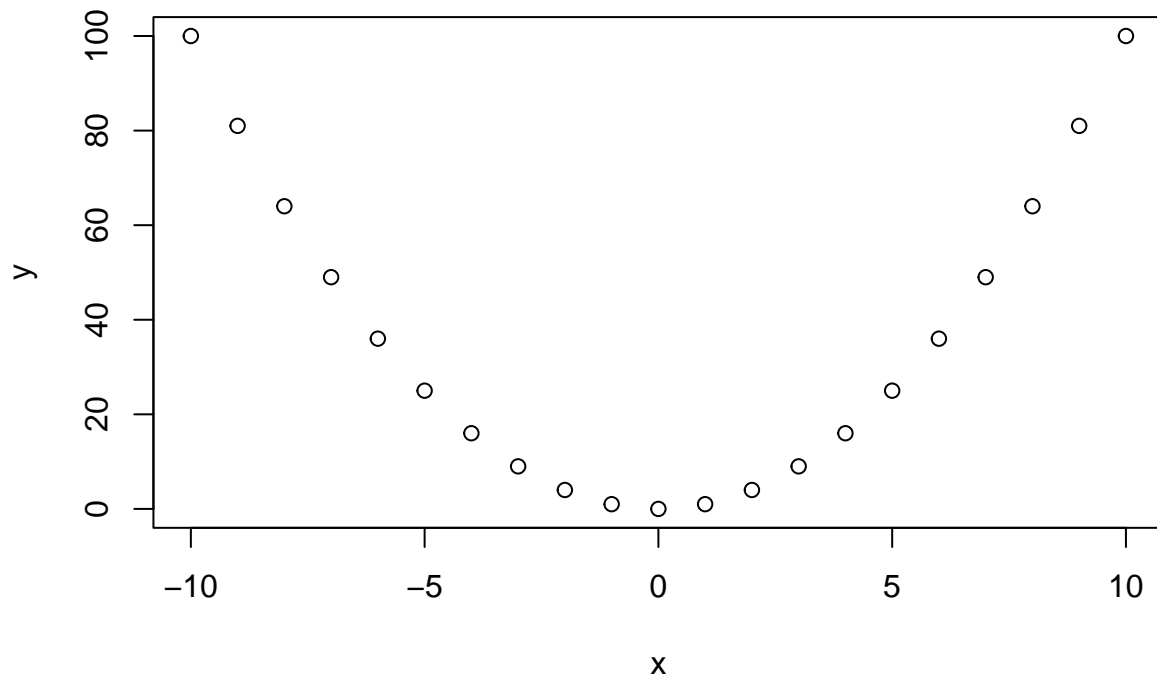
## 3 Plotting

R has three major plotting systems: `base`, `lattice`, and `ggplot`. All three do the same things and so we really only need to learn one. We will do some base plotting because it's easy for exploratory work, but focus on `ggplot` because it produces nicer looking plots, it's more consistent in syntax across different type of plots than base graphics, and the options make more sense to me. To use `ggplot` we need to use the `ggplot2` library (part of the tidyverse).

### 3.1 Basic Plots

Despite the good things about `ggplot` sometimes it is nice to do some basic, exploratory plots with base graphics.

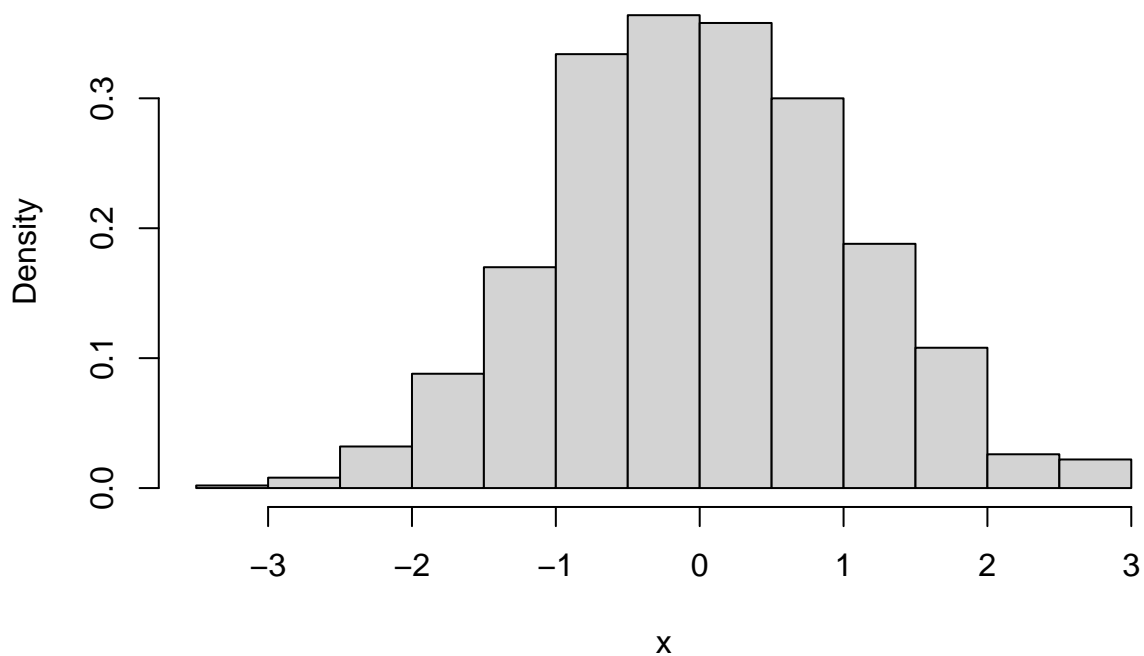
```
x <- -10:10
y <- x^2
plot(y~x)
dat <- data.frame(x=x, y=y)
with(dat, plot(y~x))
```



You can spice these up by using functions like `lines`, `points`, `rug`, or `curve`. To get a fast histogram we can do this:

```
x <- rnorm(1000)
hist(x, freq=FALSE) #To get a true histogram set freq=FALSE
```

**Histogram of x**



## 3.2 Scatterplots and Layers

We'll start with basic plots using data on the fuel economy of different cars.

```
library(ggplot2)

FE2013 <- read.csv("Datasets/FE2013.csv")
colnames(FE2013) ##Take a look at the variables

## [1] "ModelYear"          "Manufacturer"      "Division"
## [4] "Model"              "Displacement"      "Cylinder"
## [7] "FEcity"             "FEhighway"         "FEcombined"
## [10] "Guzzler"            "AirAspiration1"    "AirAspiration2"
## [13] "Gears"              "LockupTorqueConverter" "DriveSystem1"
## [16] "DriveSystem2"       "FuelType"          "FuelType2"
## [19] "AnnualFuelCost"     "IntakeValvesPerCyl" "ExhaustValvesPerCyl"
## [22] "Class"              "OilViscosity"      "StopStartSystem"
## [25] "FErating"           "CityCO2"           "HighwayCO2"
## [28] "CombinedCO2"
```

ggplot relies on layers which are connected using the + sign (which acts similarly to the %>% operator, above). The first layer is created using the ggplot command on a data.frame.

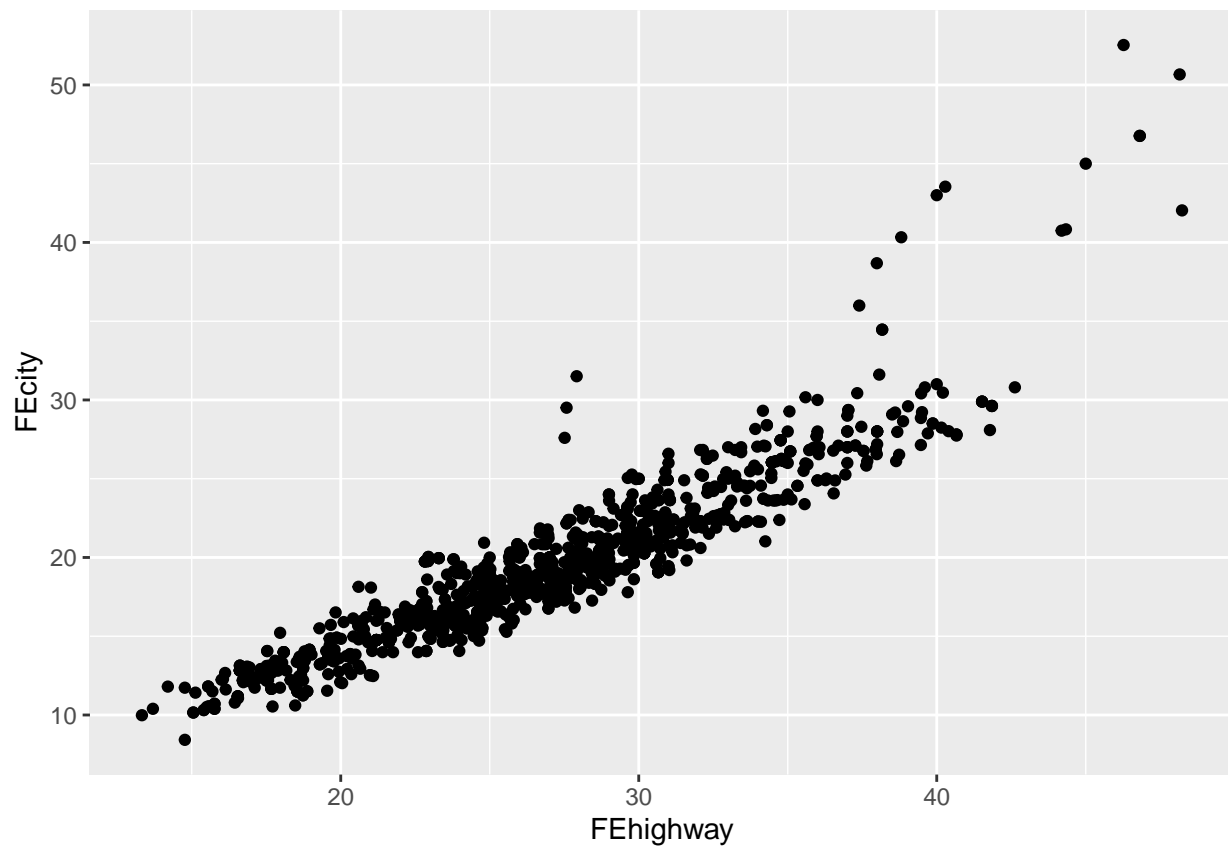
**Note** *ggplot works best with data frame and data table objects.*

```
plot1 <- ggplot(FE2013)
plot1 ##It's blank
```



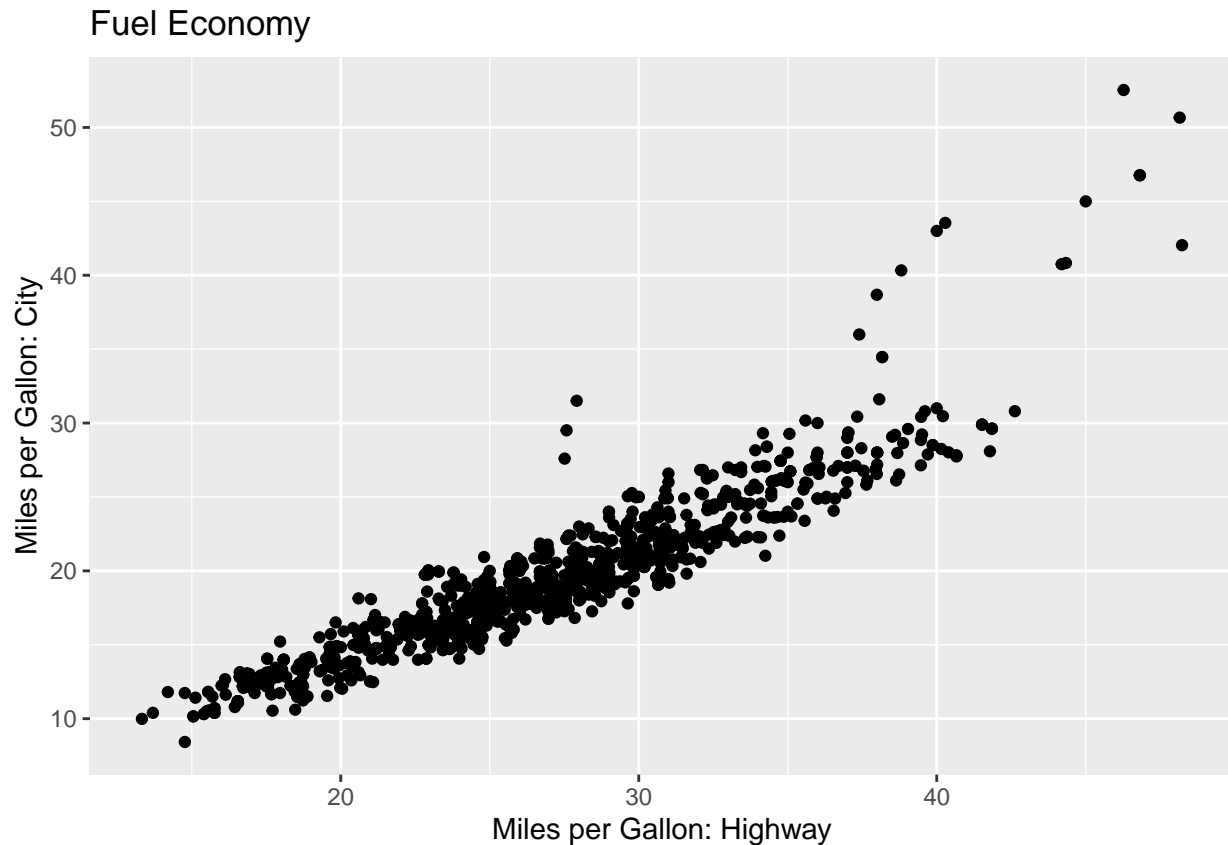
To create the scatterplot we need to add that layer to the plot

```
plot1 <- ggplot(FE2013) + ##Initial layer  
  geom_point(aes(x = FEhighway, y=FEcity))  
  
## geom_point is used to specify that we want a scatter plot  
## aes is used to specify the variables used in the plot  
  
print(plot1)
```



It's pretty straight forward to make changes to plot once it's created. Say we wanted to add a title and change the axis labels.

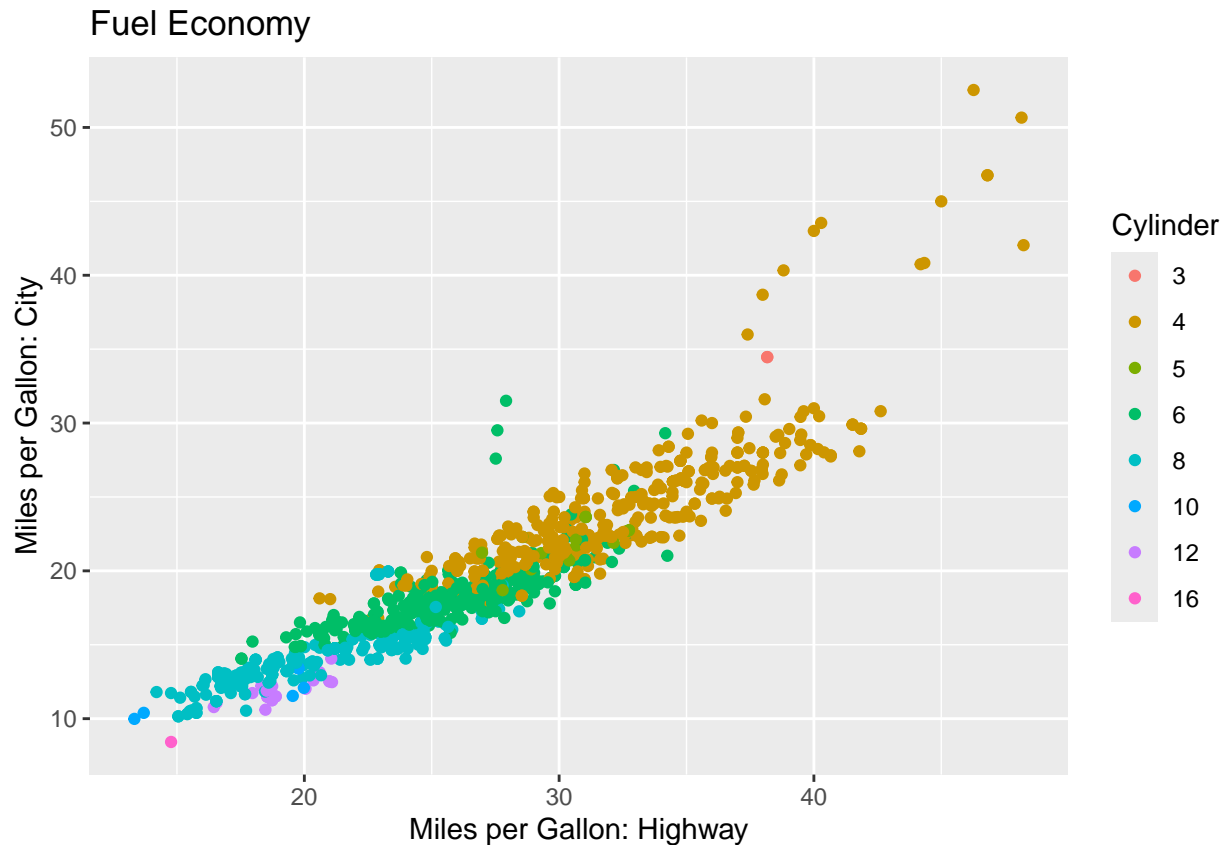
```
plot1 <- plot1 + ##Take plot1 and add  
  xlab('Miles per Gallon: Highway')+  
  ylab('Miles per Gallon: City')+  
  ggtitle('Fuel Economy')  
  
print(plot1)
```



We can add color to the plot by including a factor variable. In this case, let's color the observations by number of cylinders.

```
FE2013$Cylinder <- factor(FE2013$Cylinder) ##Need to convert to a factor

plot1 <- ggplot(FE2013) + ##Since we changed the data we need to start over
  geom_point(aes(x=FEhighway, y = FEcity, color= Cylinder))+
  xlab('Miles per Gallon: Highway')+
  ylab('Miles per Gallon: City')+
  ggtitle('Fuel Economy')
print(plot1)
```

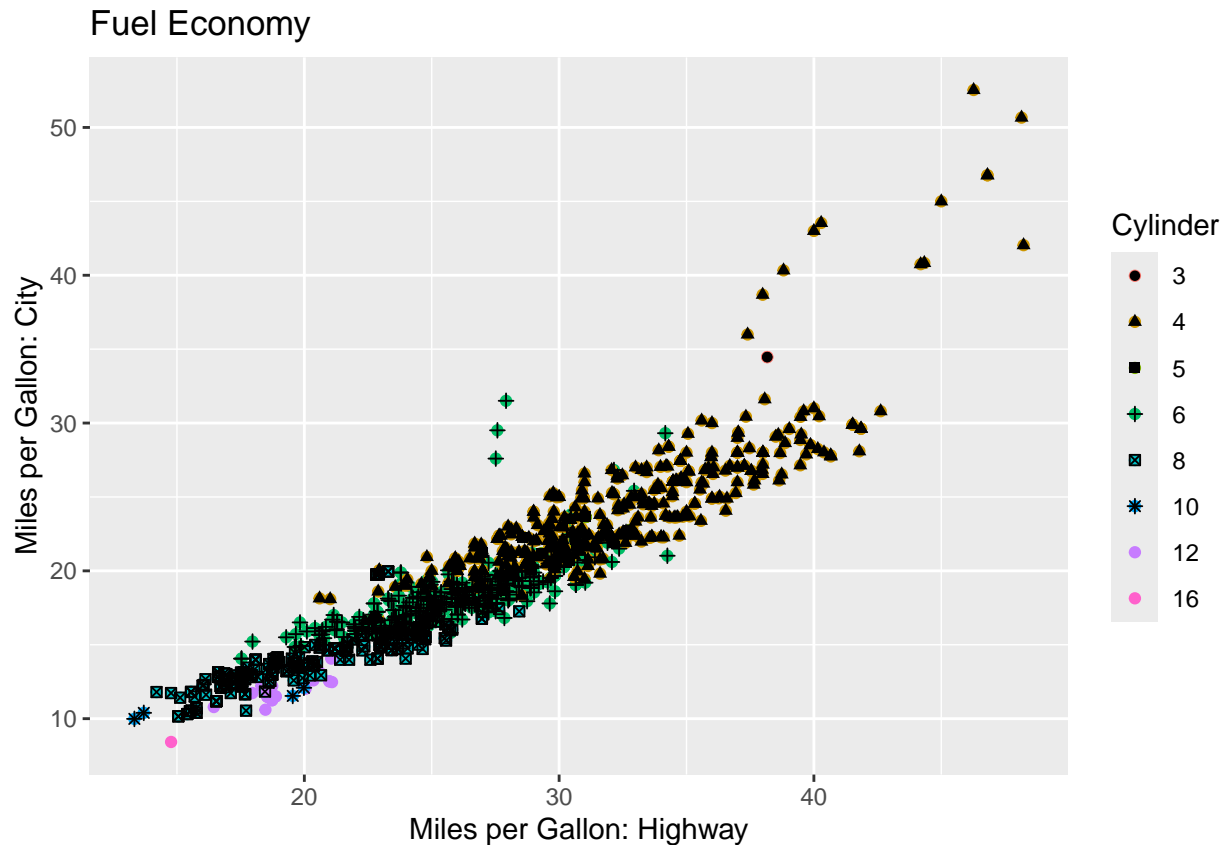


```
##Can use different shapes in place of color
plot1 <- plot1 +
  geom_point(aes(x=FEhighway, y = FEcity, shape= Cylinder))

print(plot1)
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because more
## than 6 becomes difficult to discriminate
## i you have requested 8 values. Consider specifying shapes manually if you need
## that many have them.

## Warning: Removed 23 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



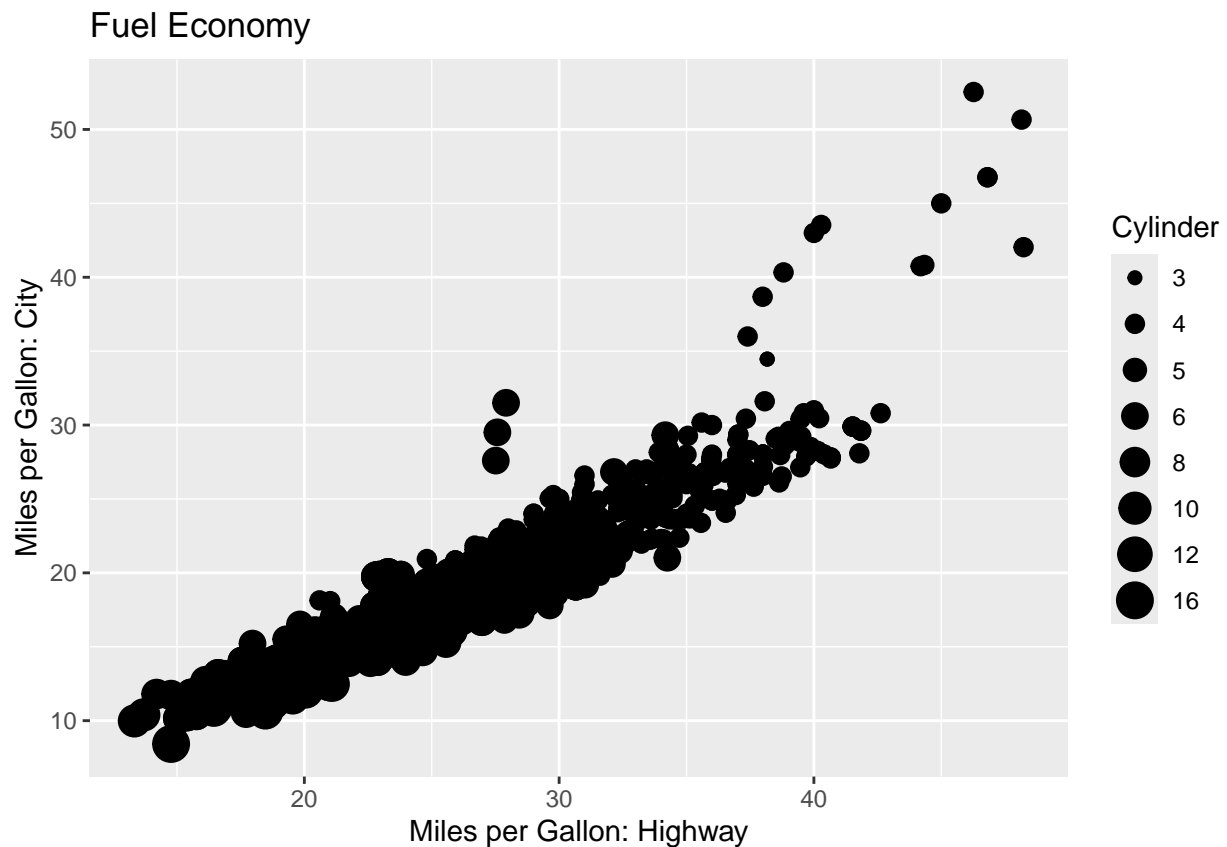
```
##Or sizes
plot1 <- plot1 +
  geom_point(aes(x=FEhighway, y = FEcity, size= Cylinder))
print(plot1)
```

```
## Warning: Using size for a discrete variable is not advised.
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because more
## than 6 becomes difficult to discriminate
```

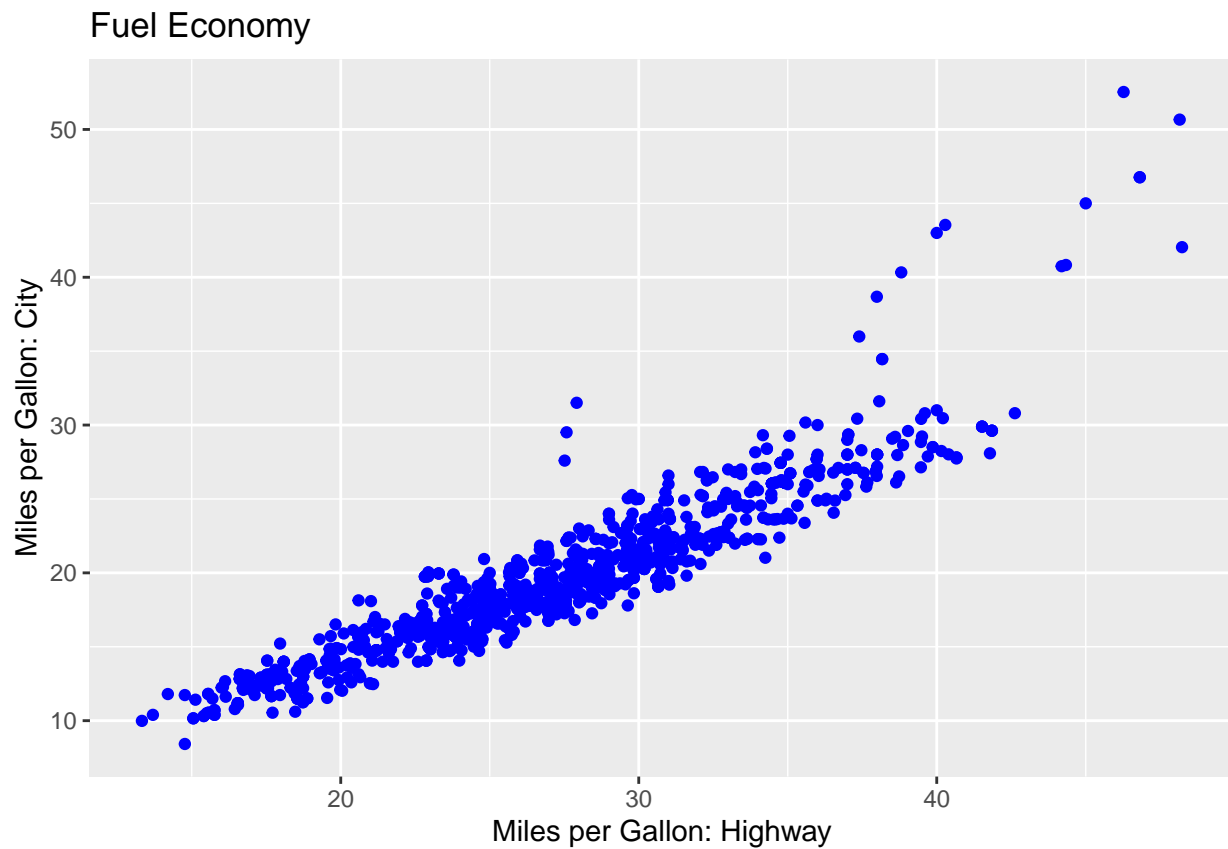
```
## i you have requested 8 values. Consider specifying shapes manually if you need
## that many have them.
```

```
## Warning: Removed 23 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

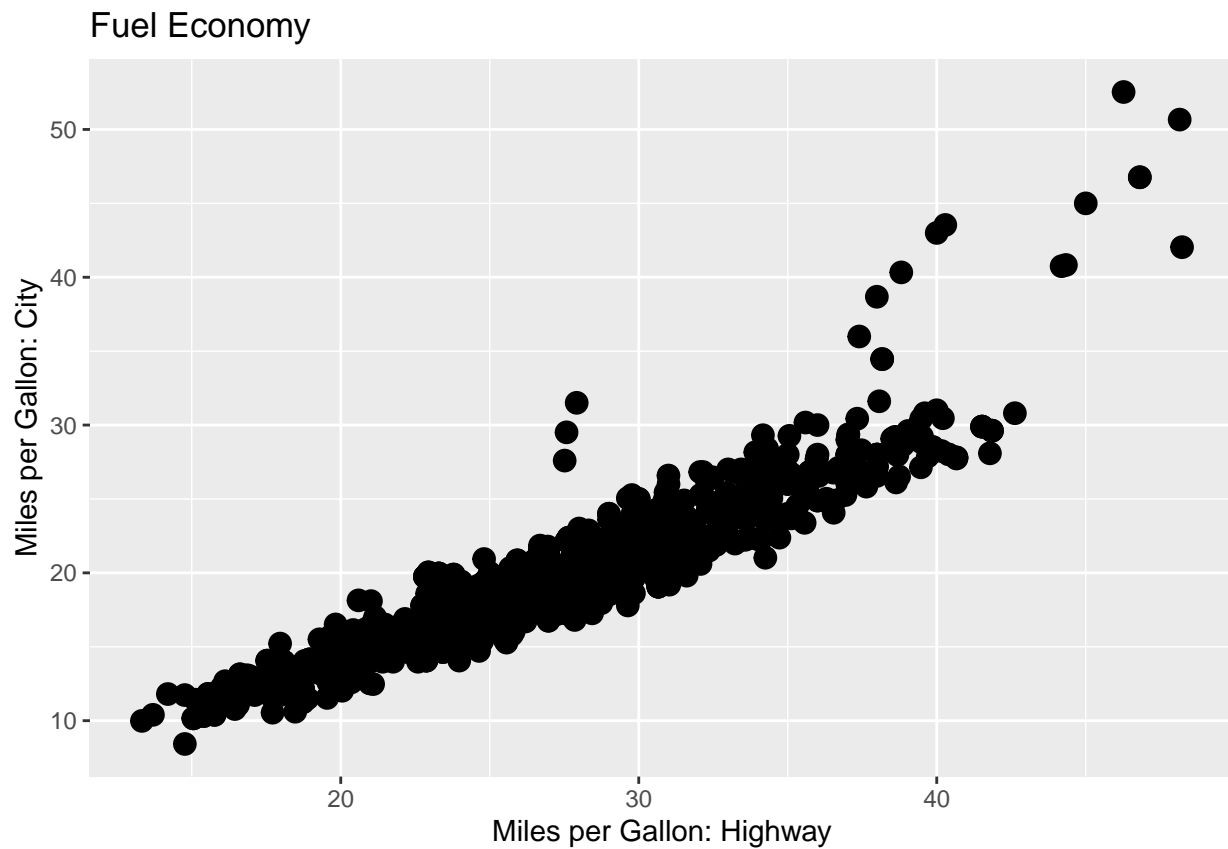


Alternatively we can adjust the color, shape, and size all the points if we do it within `geom_point` and outside `'aes'`

```
plot1 <- ggplot(FE2013) +
  geom_point(aes(x=FEhighway, y = FEcity), color="blue")+
  xlab('Miles per Gallon: Highway')+
  ylab('Miles per Gallon: City')+
  ggtitle('Fuel Economy')
print(plot1)
```

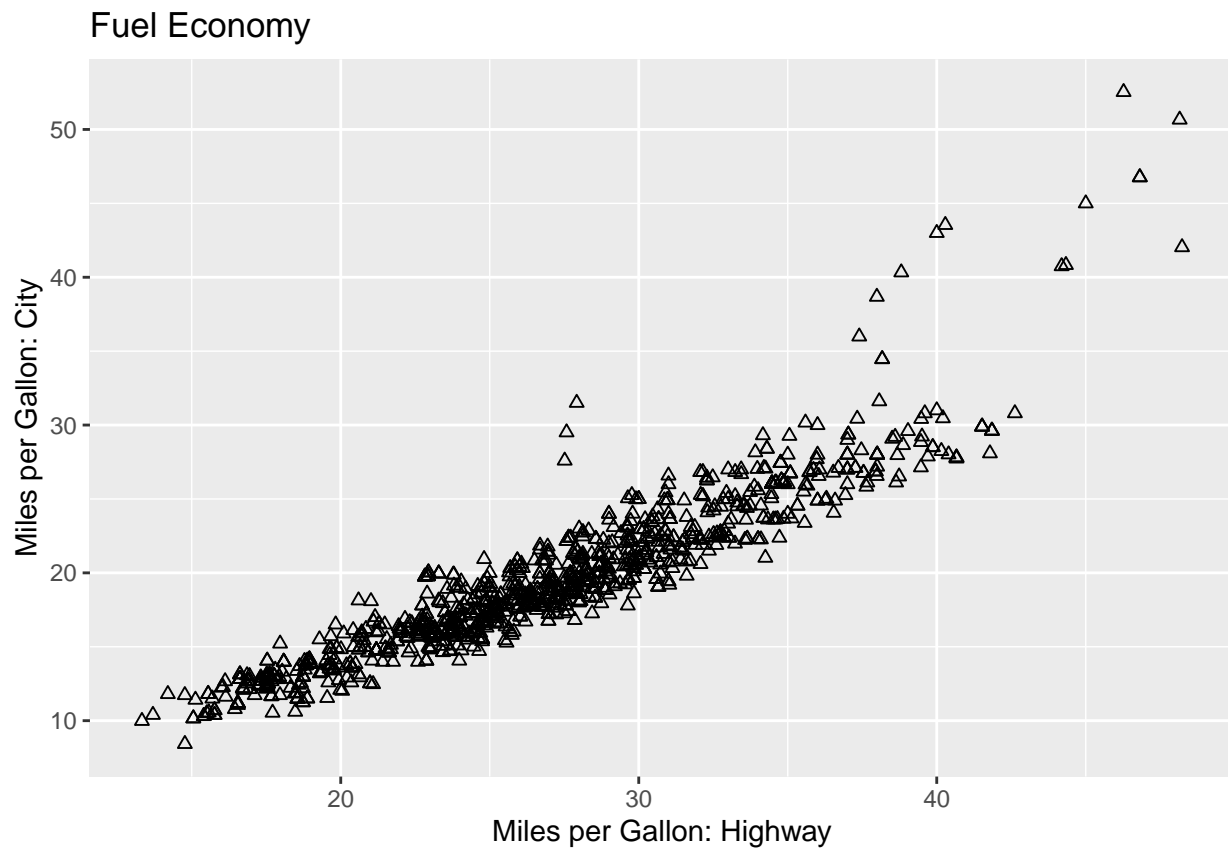


```
plot1 <- ggplot(FE2013) +  
  geom_point(aes(x=FEhighway, y = FEcity), size=3.5)+  
  xlab('Miles per Gallon: Highway')+  
  ylab('Miles per Gallon: City')+  
  ggtitle('Fuel Economy')  
print(plot1)
```



```
plot1 <- ggplot(FE2013) +  
  geom_point(aes(x=FEhighway, y = FEcity), pch=24)+  
  xlab('Miles per Gallon: Highway')+  
  ylab('Miles per Gallon: City')+  
  ggtitle('Fuel Economy')  
print(plot1)
```

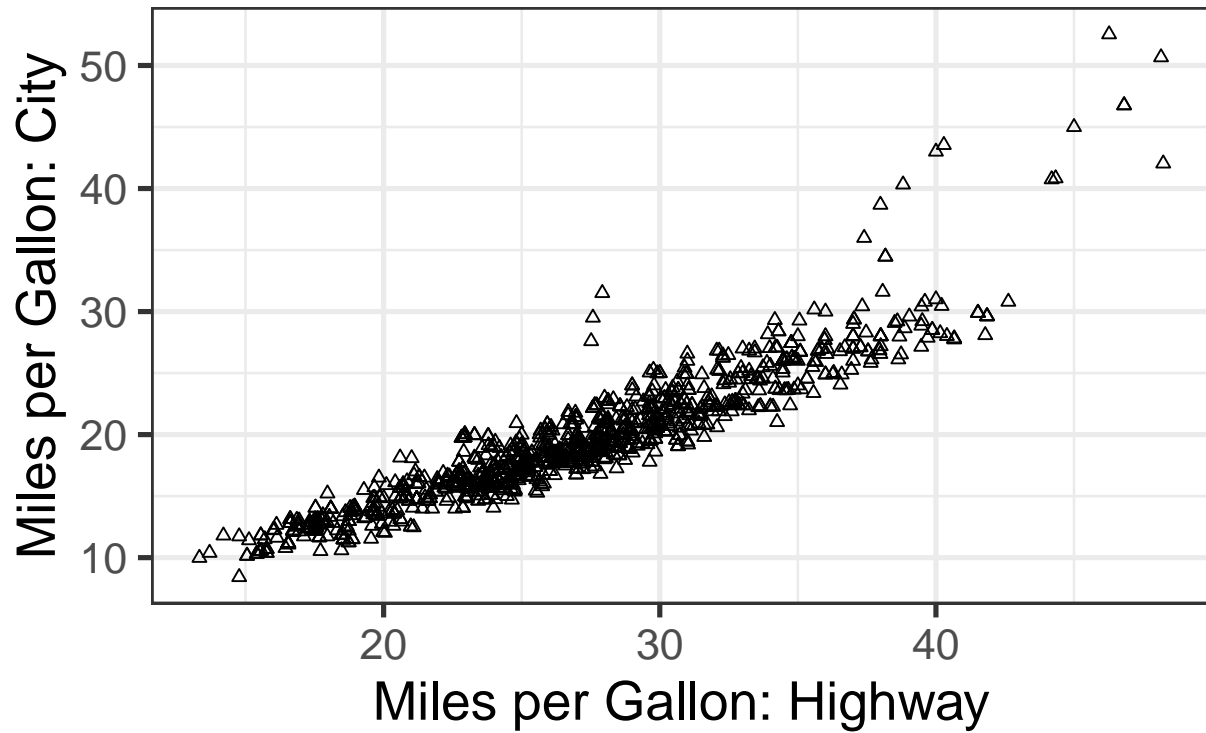




We can also change the background theme and the font size.

```
plot1 +  
  theme_bw(20)
```

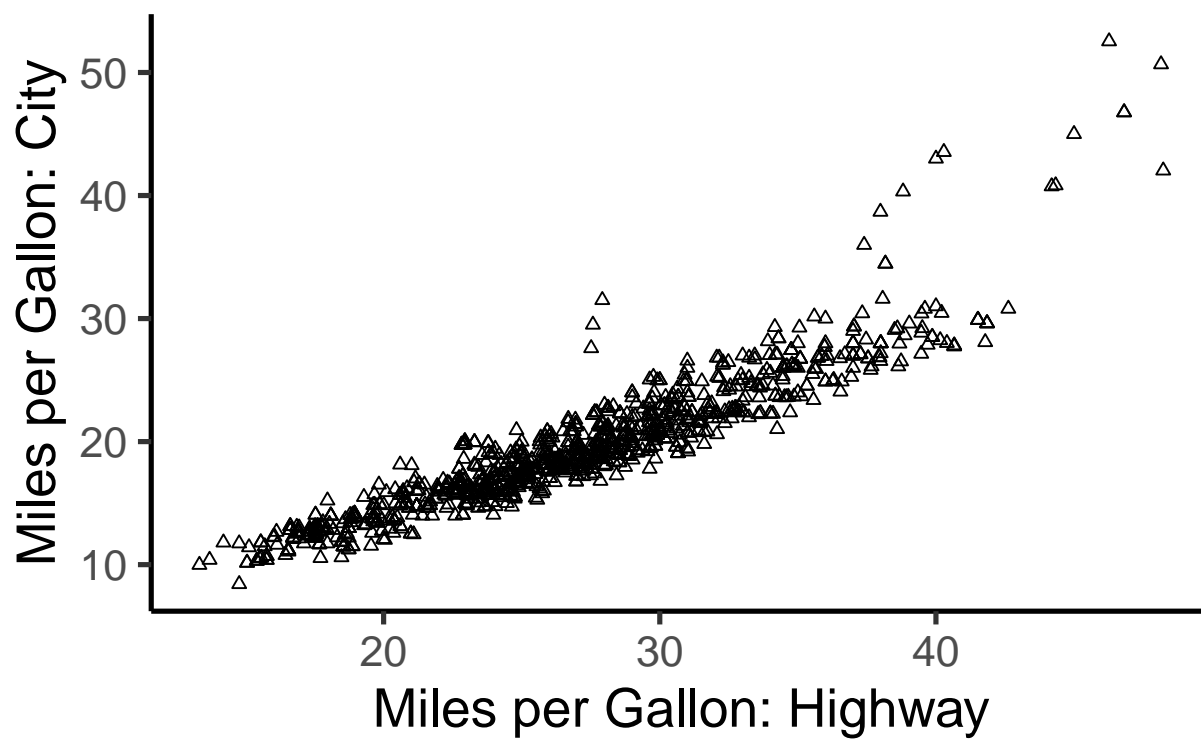
# Fuel Economy



```
##theme_bw changes the color theme,  
##20 means 20pt font
```

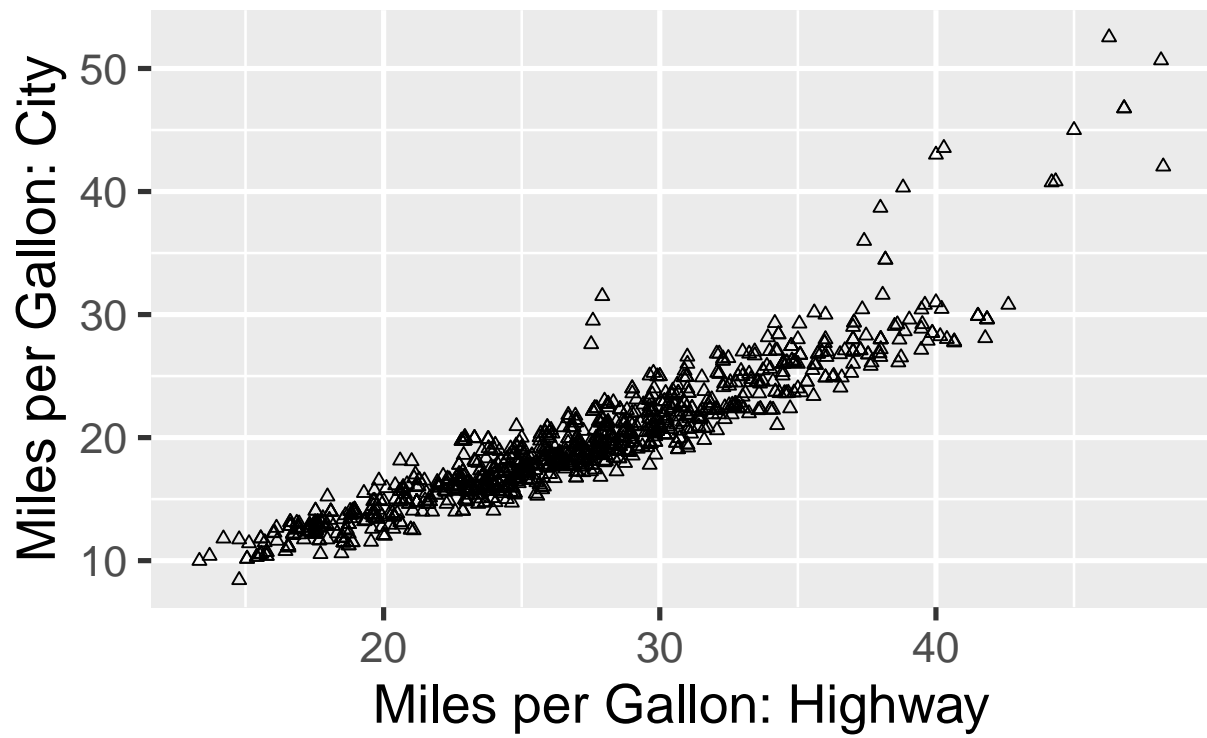
```
##Also  
plot1 +  
  theme_classic(20)
```

# Fuel Economy



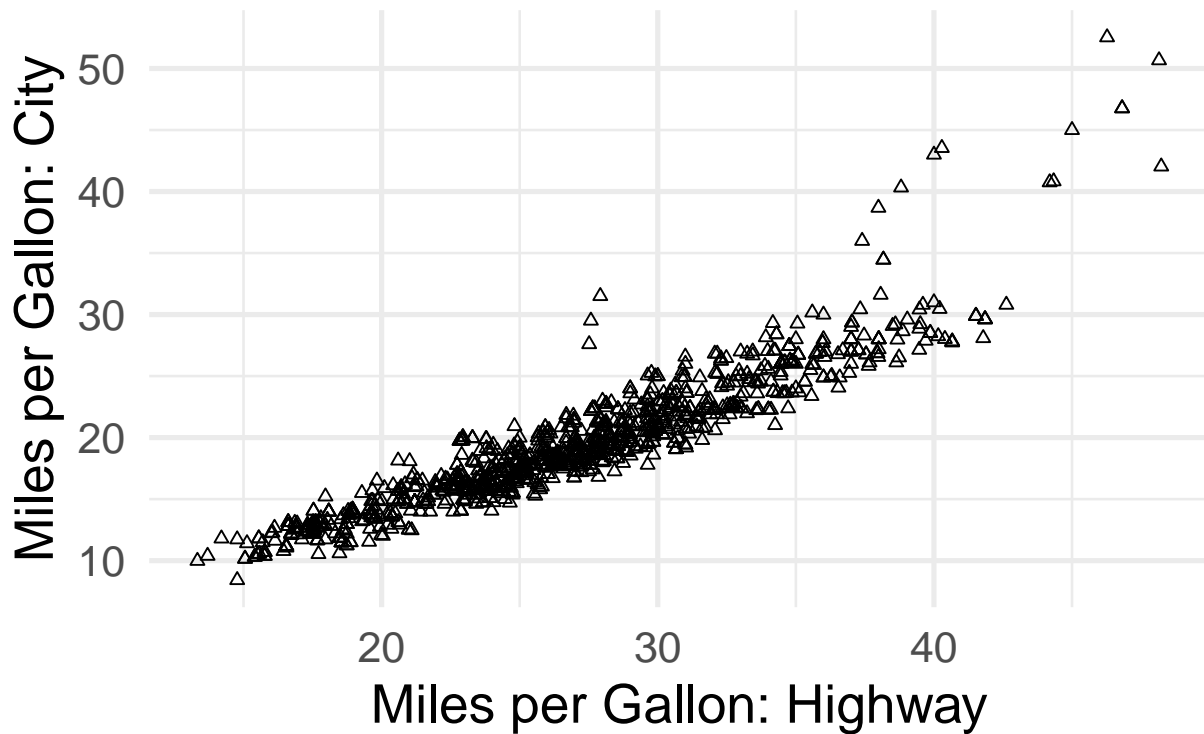
```
plot1 +  
  theme_gray(20)
```

# Fuel Economy



```
plot1 +  
  theme_minimal(20)
```

# Fuel Economy



For more information on shapes and colors that are available to `ggplot` see [http://www.cookbook-r.com/Graphs/Colors\\_\(ggplot2\)/](http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/) and [http://www.cookbook-r.com/Graphs/Colors\\_\(ggplot2\)/](http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/)

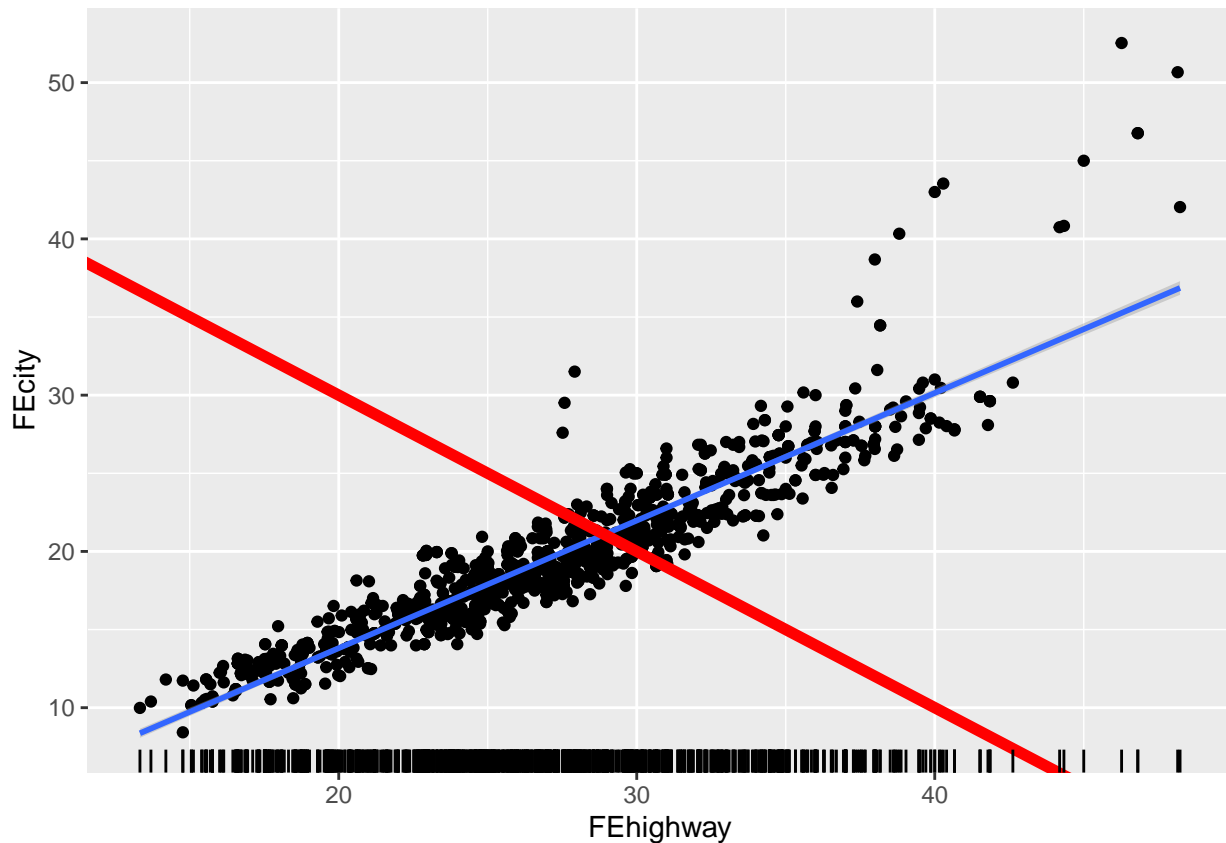
## 3.3 Adding addition geoms

We can easily add more things to our plot. In this example we'll add a best fit line, an arbitrary line, and a rug plot.

**Note** *In this plot we will specify `aes` in the the initialization step, this specifies it as a global option. In other words it's the same as entering into each geom individually.*

```
plot2 <- ggplot(FE2013, aes(x=FEhighway, y=FEcity))+  
  geom_point()+ ##Since we used aes globally we don't need it here  
  geom_smooth(method='lm', linewidth=1)+ ##best fit line, size 1  
  geom_abline(intercept=50, slope=-1, color='red',  
              linewidth=2)+ ##line  
  geom_rug(sides='b')##just across the bottom  
print(plot2)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



In theory we could keep adding on and on.

### 3.4 Other plots

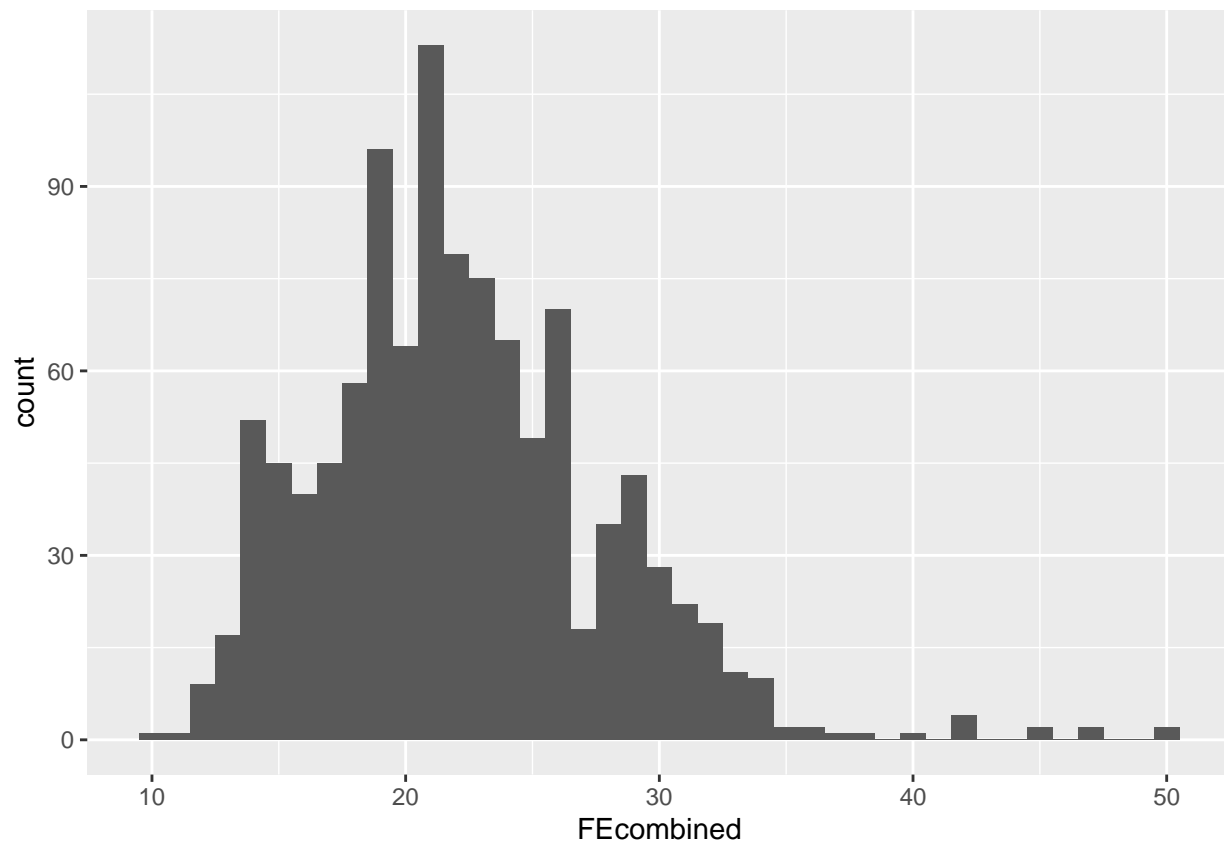
We'll now take a look at some other commonly used plots. If you have the need for other types of plots I'd recommend looking at <http://www.cookbook-r.com/Graphs/> first. They have many wonderful example with code.

#### 3.4.1 Histograms

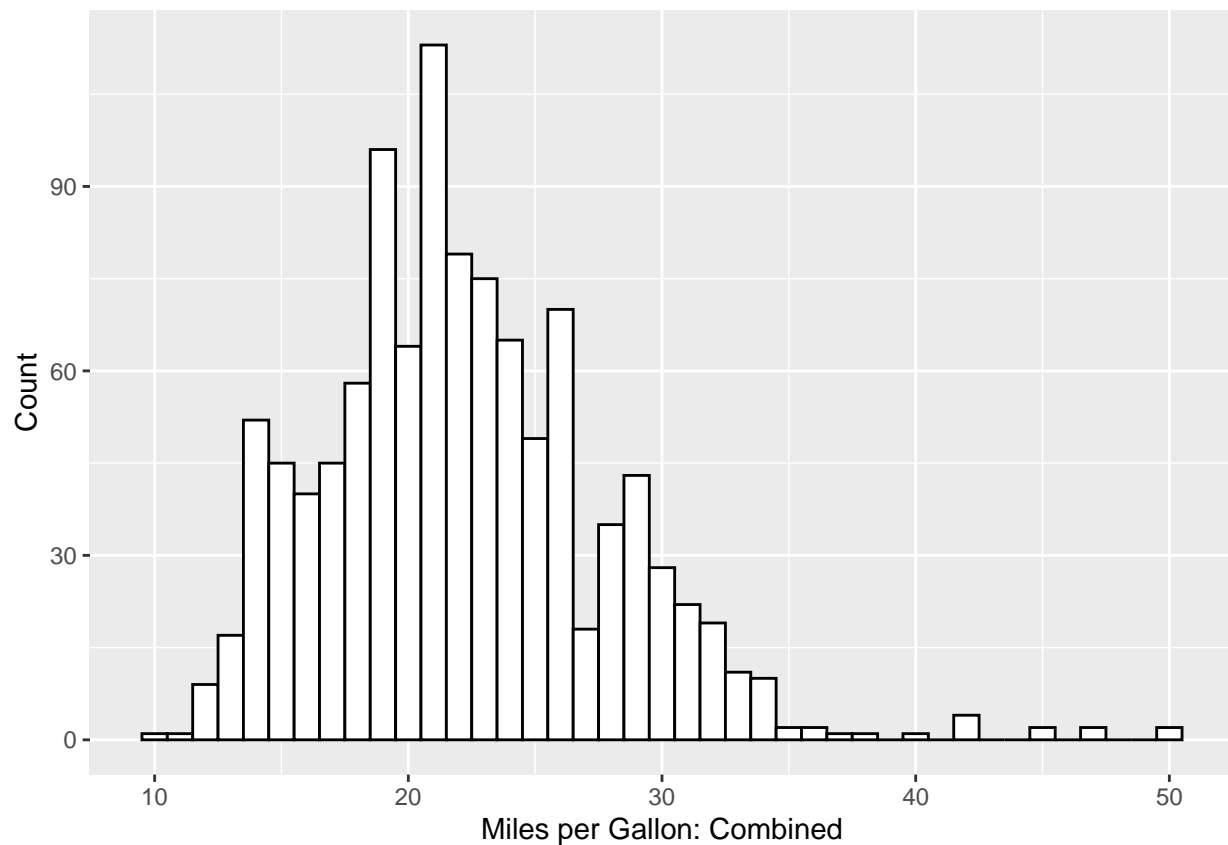
We'll start with histograms.

```
plot3 <- ggplot(FE2013, aes(x=FEcombined))+ ##only need x for hist
  geom_histogram(binwidth=1)

##if you don't specify binwidth it chooses something,
##I specified it so you can see how
print(plot3)
```



```
plot3 <- plot3 +  
  geom_histogram(binwidth=1,  
                 color='black', ##outline  
                 fill='white')+ ##Inside  
  ylab('Count')+  
  xlab('Miles per Gallon: Combined')  
print(plot3)
```

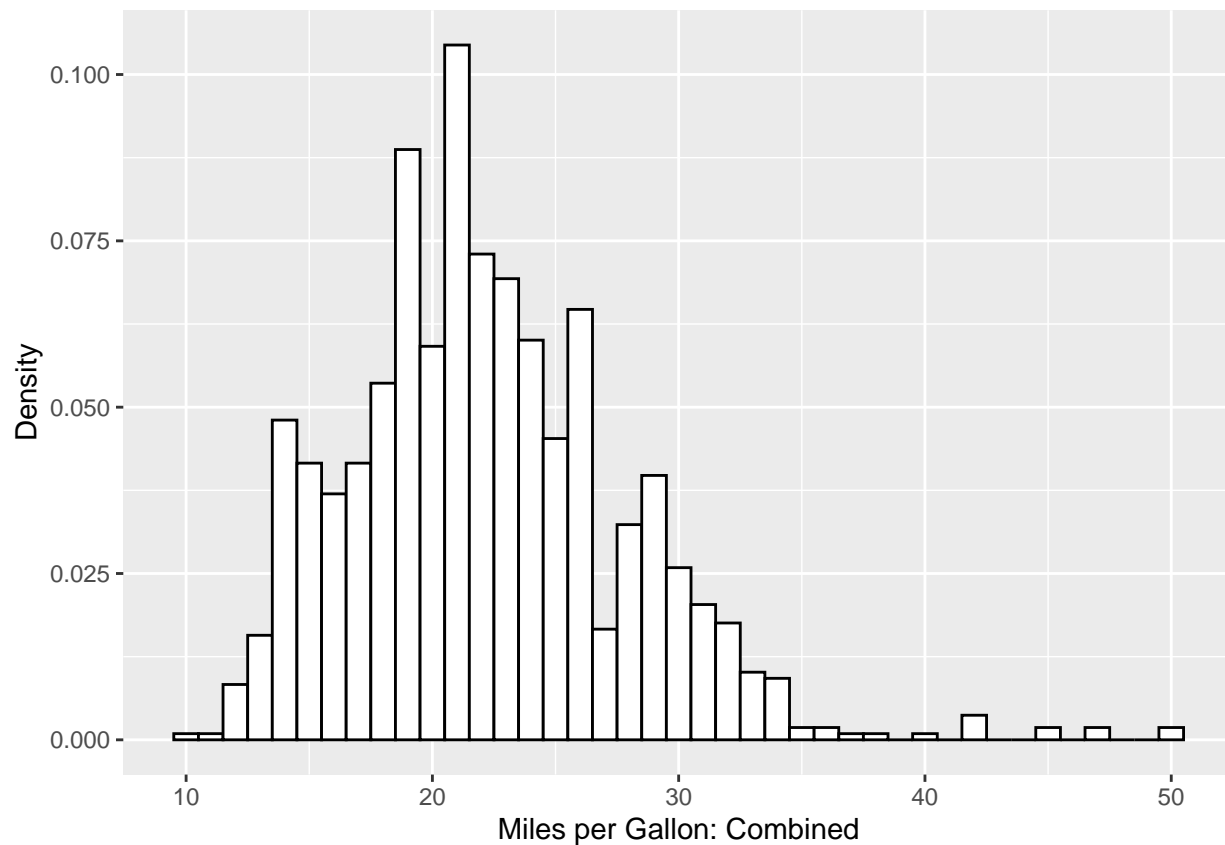


We can change counts in the histogram to density

```
plot3 <- ggplot(FE2013, aes(x=FEcombined))+ ##only need x for hist
  geom_histogram(binwidth=1,
                 color='black', ##outline
                 fill='white', ##Inside
                 aes(y=after_stat(density)))+##call aes again
  ylab('Density')+
  xlab('Miles per Gallon: Combined')

print(plot3)
```



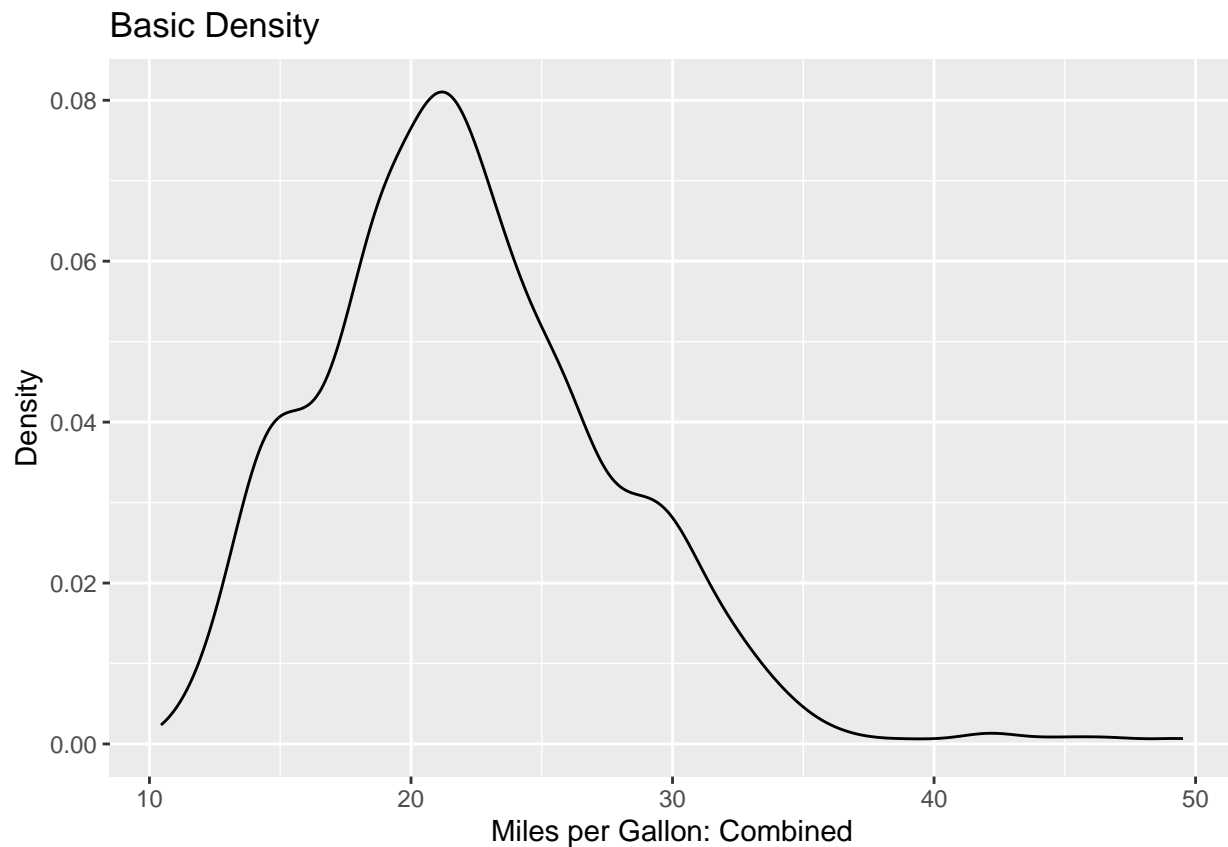


### 3.4.2 Density

We'll now look at density plots

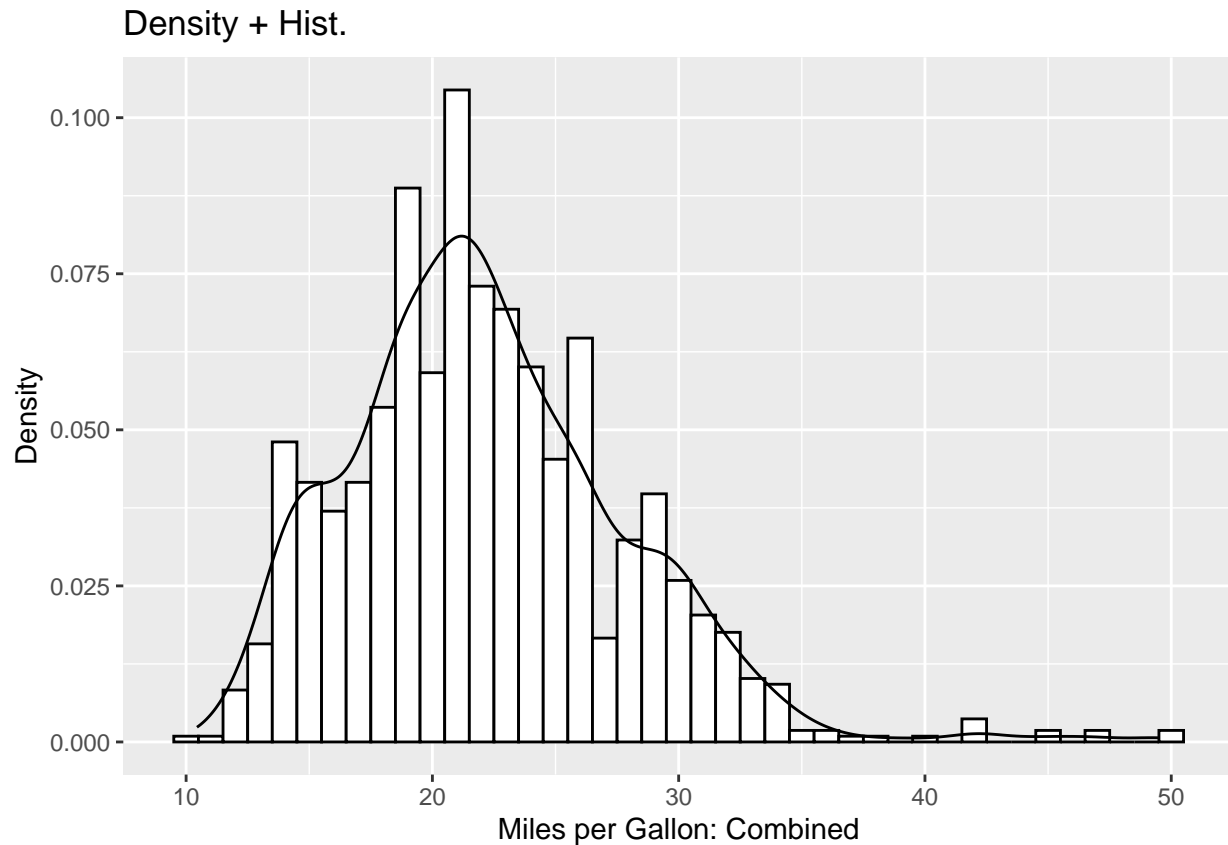
```
plot4 <- ggplot(FE2013, aes(x=FEcombined))+ ##only need x for hist
  geom_density()+
  ylab('Density')+
  xlab('Miles per Gallon: Combined')+
  ggtitle("Basic Density")

print(plot4)
```



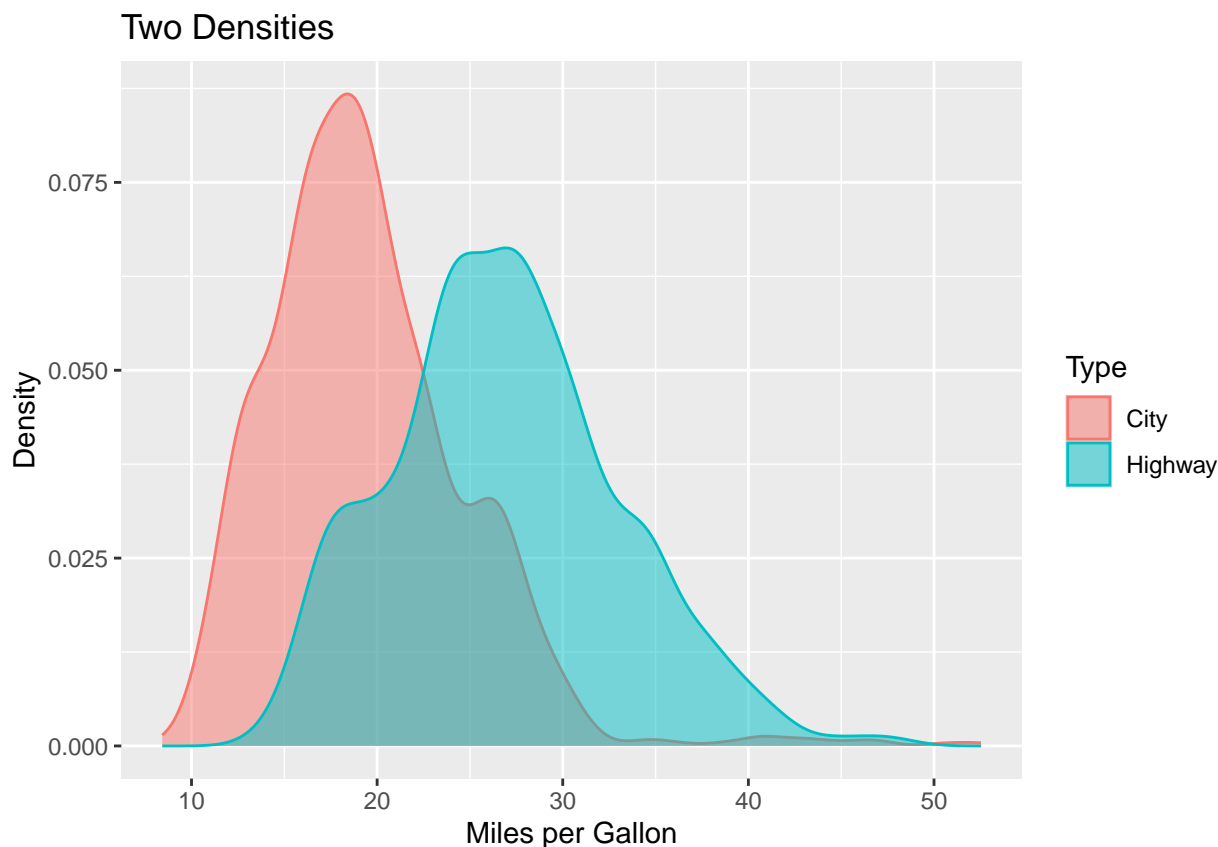
We can see that it matches by overlapping them

```
plot4 <- plot3 +  
  geom_density()+  
  ylab('Density')+  
  xlab('Miles per Gallon: Combined')+  
  ggtitle("Density + Hist.")  
  
print(plot4)
```



We can also do multiple densities at the same time

```
plot5 <- ggplot(FE2013) +
  geom_density(aes(x=FEcity ,
    fill = "City",
    color= "City"),
    alpha = 0.5)+
  geom_density(aes(x=FEhighway ,
    fill = "Highway",
    color= "Highway"),
    alpha = 0.5)+
  ylab('Density')+
  xlab('Miles per Gallon')+
  ggtitle("Two Densities")+
  guides(fill = guide_legend(title = 'Type'),
    color = guide_legend(title = 'Type'))##Change legend title
print(plot5)
```



**Note** In the last example we specified `fill` and `color` as strings, and `ggplot` made the legend for us. It is also possible to specify them as a variable (like we did with *Cylinder* above, and `ggplot` will still make the legend for us.)

### 3.5 Saving Plots

To save individual plots you can do the following

```
ggsave(plot4, file="Figures/plot4.pdf", height=4, width=6)
```

To save either individual or pages of plots you can use `pdf`

```
pdf("Figures/plot4.pdf", height=4, width=6)
```

```
plot4
```

```
dev.off()
```

```
## pdf
```

```
## 2
```

You don't have to save as .pdf, that's just what I always do because it's easy to use them for figures in other documents.