R Introduction[*]

Casey Crisman-Cox

Fall 2022 Update

[*]This document contains some material (including some chunks of code that are lifted verbatim) from Peter Haschke's *An Introduction to R* and Brenton Kenkel's *An Introduction to R*, under the terms of the Creative Commons Attribution license. Both of their works are available on the Star Lab website, `http://www.rochester.edu/college/psc/thestarlab/resources`. All errors are my own.

Contents

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.
- Methods like strategic estimators are, to my knowledge, not readily available in Stata, whereas they are straight forward 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 course sessions you should be able to

- Install/Update R and R packages (1)
- Know where to look for R help (1)
- Create simple programs and functions using R (2)
- Use control statements to program iterative procedures (2)
- Use R to read and save data (3)
- Effectively use matrices (1) and data frames (3) in R

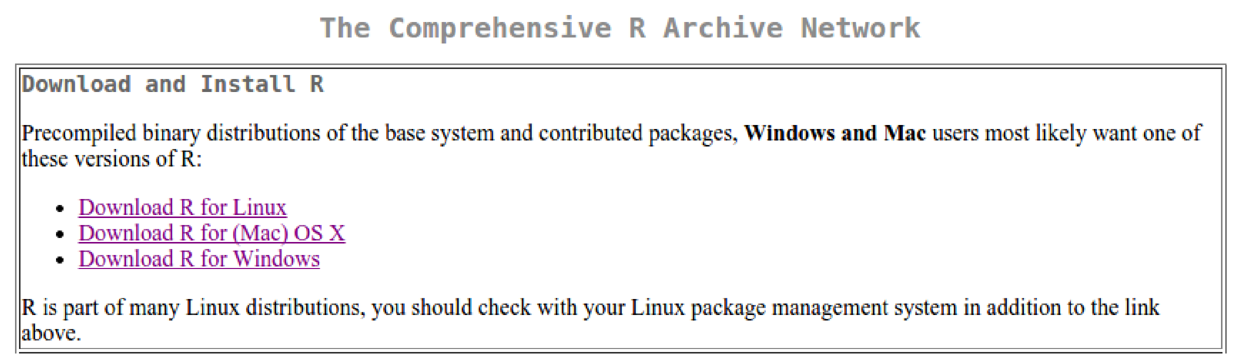
- Conduct basic statistical analysis with R (5)
- Create tables (5) and plots (4) that can be exported directly into LaTeX

We should be able to cover all this in 4 or 5 sessions, each one lasting no more than an hour. Today we'll just look at installing R and R packages, R help, and some basic operations with vectors and matrices.

1.3 Installing R

To install R for Windows

1. Go to `http://cran.r-project.org/`
2. Click on "Download R for Windows"
3. Click on "base"


The Comprehensive R Archive Network

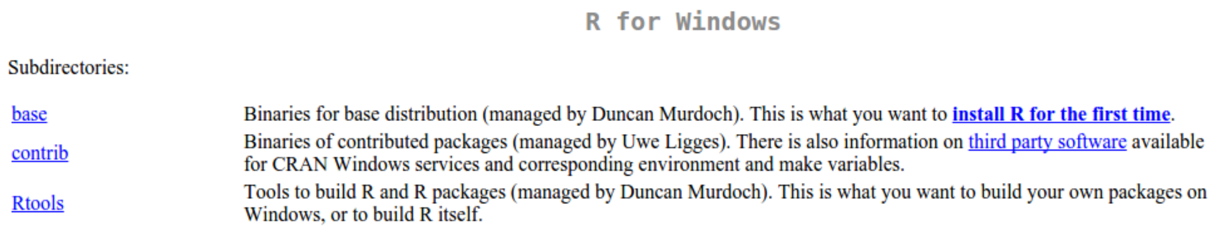
Download and Install R

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
- Download R for (Mac) OS X
- 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 1: `http://cran.r-project.org/`

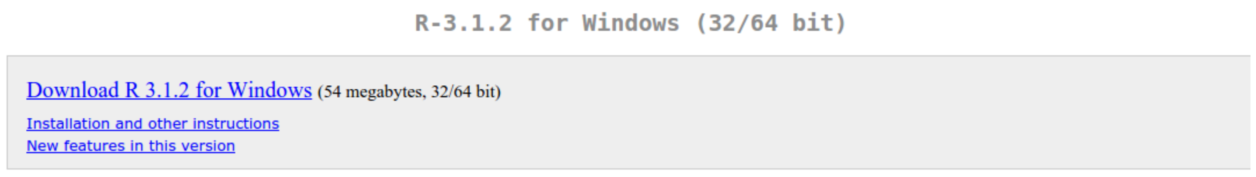

R for Windows

Subdirectories:

base	Binaries for base distribution (managed by Duncan Murdoch). This is what you want to **install R for the first time**.
contrib	Binaries of contributed packages (managed by Uwe Ligges). There is also information on third party software available for CRAN Windows services and corresponding environment and make variables.
Rtools	Tools to build R and R packages (managed by Duncan Murdoch). This is what you want to build your own packages on Windows, or to build R itself.

Figure 2: `http://cran.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


R-3.1.2 for Windows (32/64 bit)

Download R 3.1.2 for Windows (54 megabytes, 32/64 bit)

Installation and other instructions

New features in this version

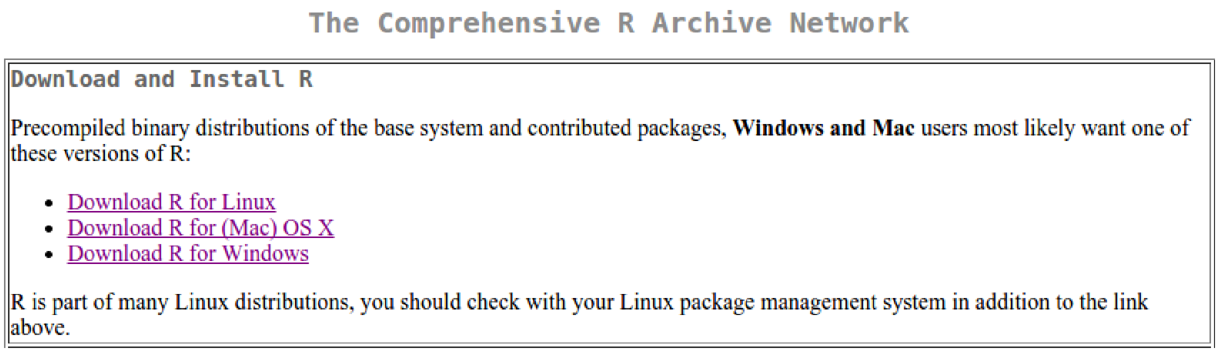
Figure 3: `http://cran.r-project.org/bin/windows/base`

You how 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://cran.r-project.org/`

2. Click on “Download R for (Mac OS X)”



The Comprehensive R Archive Network

Download and Install R

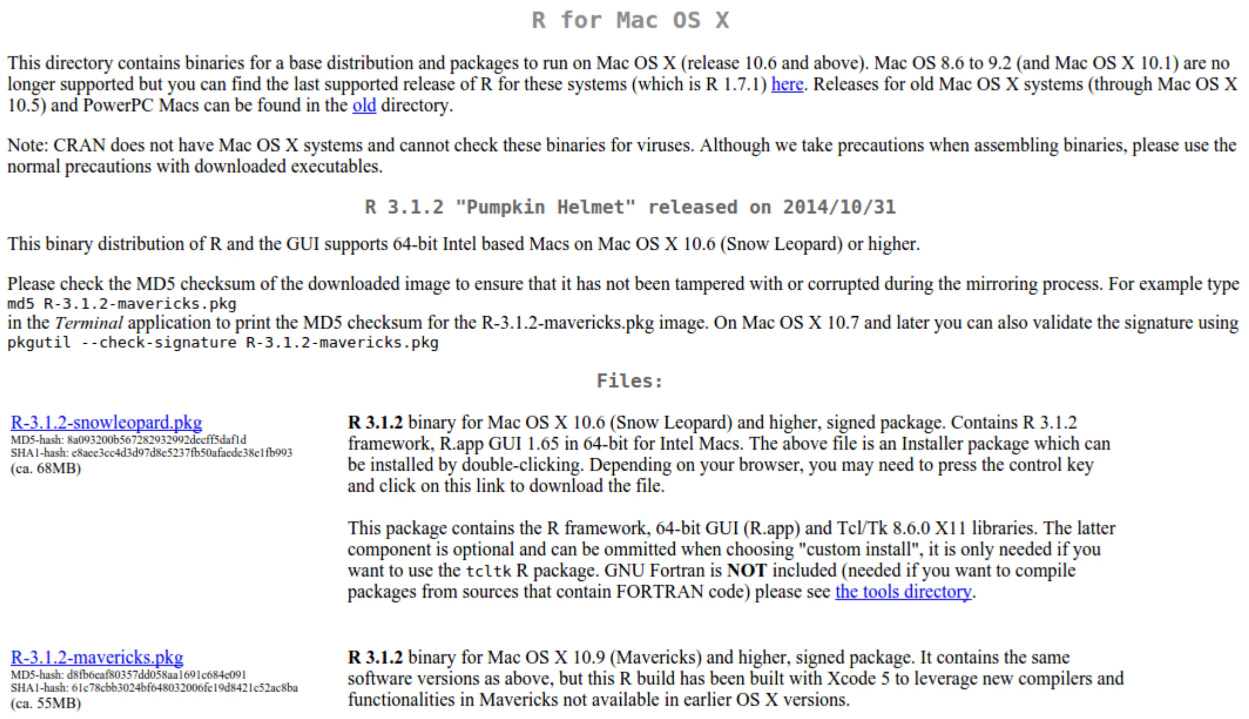
Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- 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://cran.r-project.org/`

3. Click on the version that matches your Mac



R for Mac OS X

This directory contains binaries for a base distribution and packages to run on Mac OS X (release 10.6 and above). Mac OS 8.6 to 9.2 (and Mac OS X 10.1) are no longer supported but you can find the last supported release of R for these systems (which is R 1.7.1) here. Releases for old Mac OS X systems (through Mac OS X 10.5) and PowerPC Macs can be found in the old directory.

Note: CRAN does not have Mac OS X systems and cannot check these binaries for viruses. Although we take precautions when assembling binaries, please use the normal precautions with downloaded executables.

R 3.1.2 "Pumpkin Helmet" released on 2014/10/31

This binary distribution of R and the GUI supports 64-bit Intel based Macs on Mac OS X 10.6 (Snow Leopard) or higher.

Please check the MD5 checksum of the downloaded image to ensure that it has not been tampered with or corrupted during the mirroring process. For example type
```
md5 R-3.1.2-mavericks.pkg
```
in the *Terminal* application to print the MD5 checksum for the R-3.1.2-mavericks.pkg image. On Mac OS X 10.7 and later you can also validate the signature using
```
pkgutil --check-signature R-3.1.2-mavericks.pkg
```

Files:

R-3.1.2-snowleopard.pkg
MD5-hash: 8a093200b567282932992decff5daf1d
SHA1-hash: e8aee3cc4d3d97d8e5237fb50afaede38e1fb993
(ca. 68MB)

R 3.1.2 binary for Mac OS X 10.6 (Snow Leopard) and higher, signed package. Contains R 3.1.2 framework, R.app GUI 1.65 in 64-bit for Intel Macs. The above file is an Installer package which can be installed by double-clicking. Depending on your browser, you may need to press the control key and click on this link to download the file.

This package contains the R framework, 64-bit GUI (R.app) and Tcl/Tk 8.6.0 X11 libraries. The latter component is optional and can be ommitted when choosing "custom install", it is only needed if you want to use the `tcltk` R package. GNU Fortran is **NOT** included (needed if you want to compile packages from sources that contain FORTRAN code) please see the tools directory.

R-3.1.2-mavericks.pkg
MD5-hash: d8fb6eaf80357dd058aa1691c684e091
SHA1-hash: 61c78cbb3024bf648032006fe19d8421c52ac8ba
(ca. 55MB)

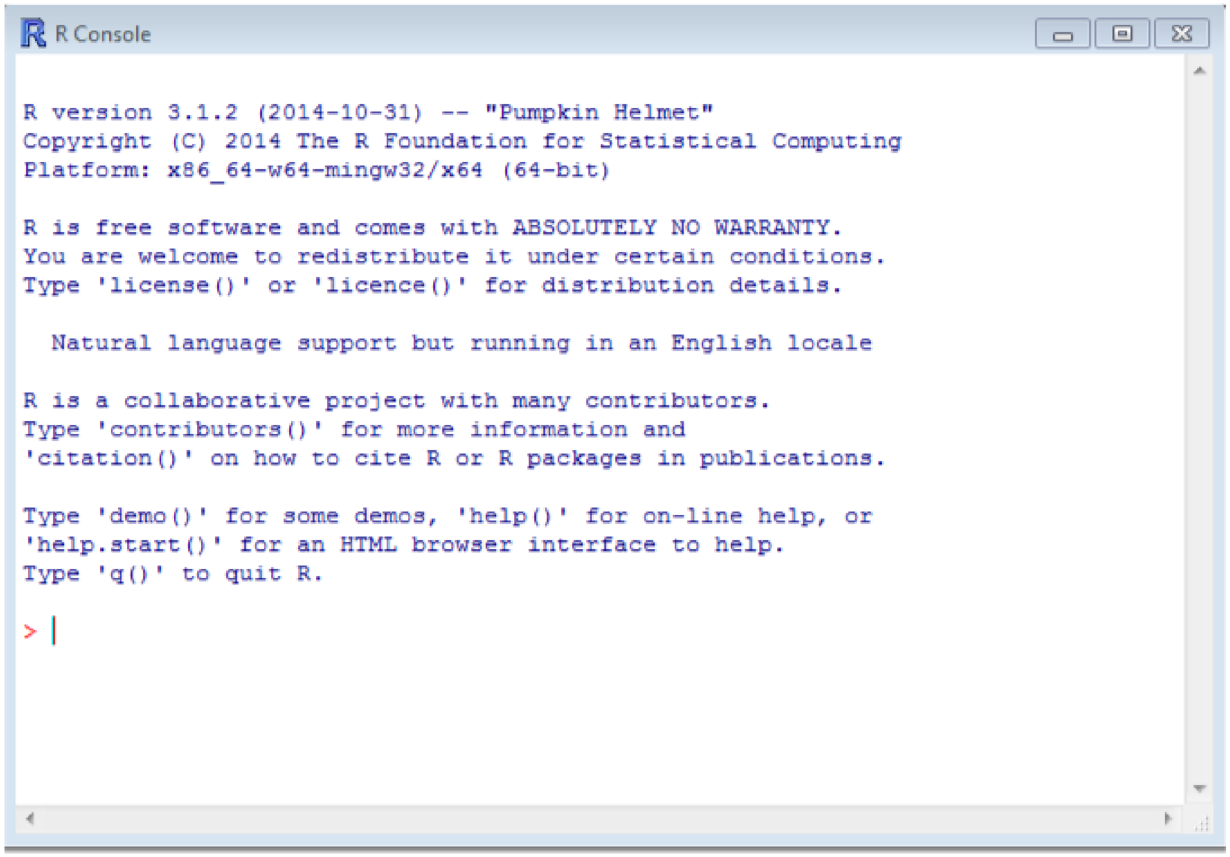
R 3.1.2 binary for Mac OS X 10.9 (Mavericks) and higher, signed package. It contains the same software versions as above, but this R build has been built with Xcode 5 to leverage new compilers and functionalities in Mavericks not available in earlier OS X versions.

Figure 5: `http://cran.r-project.org/bin/macosx/`

You how 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:



```
R version 3.1.2 (2014-10-31) -- "Pumpkin Helmet"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

Figure 6: R Console

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
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 22.04.3 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/libopenblasp-r0.3.20.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_4.3.1  fastmap_1.1.1   cli_3.6.1       tools_4.3.1
##  [5] htmltools_0.5.5 rstudioapi_0.14 yaml_2.3.7      rmarkdown_2.23
##  [9] knitr_1.43      xfun_0.39       digest_0.6.31   rlang_1.1.1
## [13] evaluate_0.20
```

Which shows us the version of R we're using, our operating system (actually 4.3.1, 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 `{r} 1+1` 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
```

```r
exp(1)
```

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

```r
sin(0)
```

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

```r
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

```r
log(0)
```

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

```r
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.

```r
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
```

As you can see the last one returned an error. 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 and the 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 vectors.

```
sum(x1)
```

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

```
prod(x1)
```

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

```r
mean(x1)
```

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

```r
median(x1)
```

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

```r
sd(x1)
```

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

We can also sort the values within a vector

```r
sort(x1)
```

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

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

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

```r
length(x1)
```

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

1.5.2 Easier 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.

```r
1:15
```

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

```r
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

```r
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
```

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

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

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

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

Finally the `rep` command allows you to repeat numbers

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

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

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

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

```r
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

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

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

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

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

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

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

```r
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

```r
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.

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

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

```r
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
```

```r
ls()
```

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

```r
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

```r
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

```r
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
```

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

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

As before we can use the logical operators

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

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

```r
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.

```r
length(X)
```

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

```r
mean(X)
```

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

```r
sd(X)
```

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

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

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

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

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

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

```r
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.

```r
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
```

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

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

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

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

Notice that both of the loop and `apply` do the same thing, but that apply is much easier to write. So let's break down what these things do. 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. The second thing we did was start the loop the line `for(i in 1:ncol(X))` just 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. The curly brackets tell R the extent of the loop.

The `apply` function on the other hand takes 3 arguments.
The first is a matrix, in this case `X`. The second is a direction, 2 means that we want R to apply the function over columns, 1 would mean we wanted to apply it over rows.
The last argument is a function, in example we just used means, 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.

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

```
## NULL
```

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

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

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

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

```r
row.names(X) <- c('up', 'down')
X
```

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

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

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

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

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

1.6.2 Matrix Operations

Matrix math in R includes standard operations including arithmetic.

```r
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,
                   nrowsY = nrow(Y))


names(matrixList)
```

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

```
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]
```

```
## $nrowsY
## [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.

```r
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.

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

```
## matrix      Y nrowsY  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.

```r
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,] -0.4651423    1
## [2,] -1.4117656    1
## [3,] -0.3429134    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] -0.7399404  1.0000000
```

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 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. I usually pick one from Pennsylvania, but it really doesn't matter which one you pick. Once it's installed you can load it.

```
library(MASS)
```

1.8.1 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.'

1.9 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,

```r
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 are usually very helpful and easy to follow.
2. ChatGPT and similar AI can also be very helpful at answering your questions.

1.10 Exercises

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. Do the following
   a. Create a 15×3 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.
3. Install the following packages:
   - `dplyr`
   - `xtable`
   - `ggplot2`

2 Control Statements and Programming

This chapter really takes us into the meat of R programming.

2.1 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.

```r
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.

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

```
## [1] 0.9832638
```

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

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

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] 0.23933694 0.25334430 0.04490898 0.78545784 0.39920010 0.21798006
##  [7] 0.07630470 0.95880996 0.65988211 0.95294198
```

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

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

As with if-else constructs we can also nest them

```
print(test)
```

```
##  [1] 0.23933694 0.25334430 0.04490898 0.78545784 0.39920010 0.21798006
##  [7] 0.07630470 0.95880996 0.65988211 0.95294198
```

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

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

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

2.2 Loops and breaks

Another commonly used control structure is the loop. We saw a `for` loop earlier, which iterates a fixed number of times, but what if we didn't know how many times something needed done though, we just know when it's done? This takes us to a second type of loop: `while`

```
#Create initial value of y that doesn't satisfy the condition
y <- 1
while(y>0.05){
  y <- runif(1)
}
y
```

```
## [1] 0.01195243
```

2.3 Scripting

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 <- list(Mean = mean(x), StDev = sd(x), Min = min(x),
## +         Median = median(x), Max = max(x))
## +     return( .... [TRUNCATED]
##
## > summarize(dat)
## $Mean
## [1] 0.005892798
##
## $StDev
## [1] 1.03719
##
## $Min
## [1] -4.509071
##
## $Median
## [1] 0.02661623
##
## $Max
## [1] 3.195255
```

Alternatively you can run individual lines by highlight them in the file editor and press ctrl+enter. RStudio also has a source button in 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]]
## [[1]]$Mean
## [1] 0.07991669
##
## [[1]]$StDev
## [1] 0.9917509
##
## [[1]]$Min
## [1] -2.703891
##
## [[1]]$Median
## [1] 0.1080072
##
## [[1]]$Max
## [1] 3.463734
##
##
## [[2]]
## [[2]]$Mean
## [1] 500.5
##
## [[2]]$StDev
## [1] 288.8194
##
## [[2]]$Min
## [1] 1
```

```
##
## [[2]]$Median
## [1] 500.5
##
## [[2]]$Max
## [1] 1000
```

3 Data Frames and tables

Today 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).

3.1 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 you can use either `readstata13` or `haven`. Haven is part of the "tidyverse" which is a set of packages that form an easy and increasingly popular way to do things in R. I also like `data.tables` as a faster alternative to the tidyverse. For each thing today we'll talk about the base and tidy way to do things. I will eventually include the data table ways as an appendix to this chapter.

The tidyverse has several packages for reading data `haven` for dta files, `readr` for most text data (csv, txt, tab), and `readxl` package for excel-style files (xls and xlsx). To see these in action, we'll read in the data files that I sent you this morning. 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

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

```r
# Tidy packages
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following object is masked _by_ '.GlobalEnv':
##
##     summarize

## The following object is masked from 'package:MASS':
##
##     select

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
#Alternative
library(data.table)
```

```
##
## Attaching package: 'data.table'
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
# for stata files (i like it better than haven)
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.

3.2 Commands to use on Data

3.2.1 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 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 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 ccode
## 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
##
```

```r
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.

3.2.2 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
```

3.2.3 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
```

```r
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
```

```r
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
```

```r
## 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
## DT approach
FL.dt <- data.table(FL2003) #need to convert first
temp.dat <- FL.dt[,.(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
```

In general, `subset`, `select`, or the `data.table` approaches are probably better than any of the other alternatives depending on whether you like base, tidy, or data table ecosystems. Take a second to look up `with`, it can be helpful with data frames when working in base.

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        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 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 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
```

```
#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 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 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
```

```
# data table
temp.dat <- FL.dt[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 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

```r
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
```

```
# Data table
temp.dat <- FL.dt[ccode==2, .(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
```

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

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

3.2.4 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

```r
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
```

```r
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.

3.2.4.1 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() #WHOOPS
```

```
##    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
`as.numeric`	Change a factor or character vector into numbers
`as.character`	Change a numeric or factor vector into a character string
`as.Date`	Change a character vector of dates in a Date object
`as.factor`	Change a character or numeric vector in factor
`as.matrix`	Change a vector or data frame into a matrix
`as.data.frame`	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

The only benefits of using the latter is that there are more options.

```
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
```

We'll do more with factors when we do analysis in the next session. They become more useful then.

3.3 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 -0.7177111
## 2     2  0.1187258
## 3     3  1.1998786
## 4     4 -0.1902675
## 5     5 -0.9428839
```

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

```
##   ccode      Var2
## 1     1 0.3886264
## 2     2 0.2761789
## 3     3 0.5406297
## 4     4 0.1911539
## 5     5 0.6625274
```

```
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 -0.7177111 0.3886264
## 2     2  0.1187258 0.2761789
## 3     3  1.1998786 0.5406297
## 4     4 -0.1902675 0.1911539
## 5     5 -0.9428839 0.6625274
```

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 -1.04887200
## 2        2 -0.04253714
## 3        3 -0.15480268
## 4        4 -0.12577959
## 5        5 -1.70851318
```

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

```
##   ccode       Var2
## 1     1 0.59558620
## 2     2 0.00272955
## 3     3 0.46254234
## 4     4 0.06399984
## 5      5 0.16387759
```

```r
##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 -1.04887200 0.59558620
## 2        2 -0.04253714 0.00272955
## 3        3 -0.15480268 0.46254234
## 4        4 -0.12577959 0.06399984
## 5        5 -1.70851318 0.16387759
```

An even more complex example

```r
###Data sets with different countries
temp.df <- data.frame(cow.code= 1:10,
                      Var1= rnorm(10))

temp.df
```

```
##    cow.code       Var1
## 1         1 -0.7927236
## 2         2  0.4151368
## 3         3  0.7639213
## 4         4  0.2218294
## 5         5 -0.6565050
## 6         6 -0.8168150
## 7         7 -0.5969682
## 8         8  1.1995010
## 9         9  0.4670254
## 10       10 -1.2323271
```

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

```
##    ccode      Var2
## 1      1 0.1955599
## 2      2 0.2416748
## 3      3 0.9704524
## 4      4 0.4493168
## 5      5 0.6832629
## 6     11 0.1955599
## 7     12 0.2416748
## 8     13 0.9704524
## 9     14 0.4493168
## 10    15 0.6832629
```

```
##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 -0.7927236 0.1955599
## 2        2  0.4151368 0.2416748
## 3        3  0.7639213 0.9704524
## 4        4  0.2218294 0.4493168
## 5        5 -0.6565050 0.6832629
```

```
##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 -0.7927236 0.1955599
## 2         2  0.4151368 0.2416748
## 3         3  0.7639213 0.9704524
## 4         4  0.2218294 0.4493168
## 5         5 -0.6565050 0.6832629
## 6         6 -0.8168150        NA
## 7         7 -0.5969682        NA
## 8         8  1.1995010        NA
## 9         9  0.4670254        NA
## 10       10 -1.2323271        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 -0.7927236 0.1955599
## 2         2  0.4151368 0.2416748
## 3         3  0.7639213 0.9704524
## 4         4  0.2218294 0.4493168
## 5         5 -0.6565050 0.6832629
## 6        11         NA 0.1955599
## 7        12         NA 0.2416748
## 8        13         NA 0.9704524
## 9        14         NA 0.4493168
## 10       15         NA 0.6832629
```

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

```
##    cow.code       Var1      Var2
## 1         1 -0.7927236 0.1955599
## 2         2  0.4151368 0.2416748
## 3         3  0.7639213 0.9704524
## 4         4  0.2218294 0.4493168
## 5         5 -0.6565050 0.6832629
## 6         6 -0.8168150        NA
## 7         7 -0.5969682        NA
## 8         8  1.1995010        NA
## 9         9  0.4670254        NA
## 10       10 -1.2323271        NA
## 11       11         NA 0.1955599
## 12       12         NA 0.2416748
## 13       13         NA 0.9704524
## 14       14         NA 0.4493168
## 15       15         NA 0.6832629
```

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 very straight forward. There is a tidy alternative, but I like merge and think it works just fine. Data tables have their own `merge` function so everything above should work fine on both data tables and the tidy data frames.

3.4 Reshaping Data

Sometimes we get data that need to be reshaped.
Some common examples are Freedom House or World Bank data which typically comes in a wide format. The tidy form of this task involves pivot functions

```
##Freedom House data on Freedom of the Press
pressData <- read.csv('Datasets/Press_FH.csv')

##It has  a column for country names and then a bunch of years
```

```
##We want to reshape it into a country year format
colnames(pressData)
```

```
##  [1] "country" "X1979"   "X1980"   "X1981"   "X1982"   "X1983"   "X1984"
##  [8] "X1985"   "X1986"   "X1987"   "X1988"   "X1989"   "X1990"   "X1991"
## [15] "X1992"   "X1993"   "X1994"   "X1995"   "X1996"   "X1997"   "X1998"
## [22] "X1999"   "X2000"   "X2001"   "X2002"   "X2003"   "X2004"   "X2005"
## [29] "X2006"   "X2007"   "X2008"   "X2009"   "X2010"   "X2011"
```

```
# Tidy approach
library(tidyr)
pressData <- read.csv('Datasets/Press_FH.csv')
pressData <- pressData %>%
  pivot_longer(cols = !country, #colnames to swing around (everything but country)
               names_to ='year', ##What to call column that is
               # now the old column names
               names_prefix = "X", #removing prefix
               values_to = 'press'  ) ##What to call column with the data
head(pressData)
```

```
## # A tibble: 6 x 3
##   country     year  press
##   <chr>       <chr> <chr>
## 1 Afghanistan 1979  NF
## 2 Afghanistan 1980  NF
## 3 Afghanistan 1981  NF
## 4 Afghanistan 1982  NF
## 5 Afghanistan 1983  NF
## 6 Afghanistan 1984  NF
```

```
#Melt is  the data table approach
pressData <- fread('Datasets/Press_FH.csv')  #read a csv directly to a data table
class(pressData)
```

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

```
pressData <- melt(pressData,
                  id.vars=c('country'), ##Variable to melt against
                  variable.name='year', ##What to call the column names
                  value.name = 'press' ##What to call the data
)
head(pressData)
```

```
##                country  year press
## 1:         Afghanistan X1979    NF
## 2:             Albania X1979    NF
## 3:             Algeria X1979    NF
## 4:             Andorra X1979   N/A
## 5:              Angola X1979    NF
## 6: Antigua and Barbuda X1979   N/A
```

3.5 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))

# data table
FL.dt[,log.gdpen := log(gdpen)]
```

3.5.1 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
```

```
# DT will use the null approach
c('politycode', 'casename') %in% colnames(FL.dt)
```

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

```
FL.dt[, `:=`(politycode=NULL,
             casename=NULL)]
c('politycode', 'casename') %in% colnames(FL.dt)
```

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

```r
# note with data tables the `:=`(...) syntax saves the results of
# what's inside the (...)
```

We'll now look at some applications of common data tasks.

3.5.2 APPLICATION: Generating Dummies

Generating dummy variables is a common task and there lots of ways to do it. First let's just look at a making a dummy for democracy

```r
##We can do it by indexing (not great)
FL2003$demDummy <- FL2003$polity2 ##initalize it
FL2003$demDummy[FL2003$polity2 < 7] <- 0
FL2003$demDummy[FL2003$polity2 >= 7] <- 1
summary(FL2003$demDummy)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  0.0000  0.0000  0.0000  0.3088  1.0000  1.0000      62
```

```r
FL2003$demDummy <- NULL #erase it



##There's a better way to do it
FL2003$demDummy <- ifelse(FL2003$polity2 < 7, ##if condition
                          0,  ##if TRUE, return 0
                          1)  ##else return 1
summary(FL2003$demDummy)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  0.0000  0.0000  0.0000  0.3088  1.0000  1.0000      62
```

```r
FL2003$demDummy <- NULL



##OR even
FL2003$demDummy <- as.numeric(FL2003$polity2 >= 7)
summary(FL2003$demDummy)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  0.0000  0.0000  0.0000  0.3088  1.0000  1.0000      62
```

```
FL2003$demDummy <- NULL
##This last one generates TRUE and FALSE values,
##as.numeric converts them 1 and 0 respectively.



# Tidy with ifelse
FL2003  <- FL2003 %>%
  mutate(demDummy = ifelse(polity2 < 7 ,0,1))
summary(FL2003$demDummy)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  0.0000  0.0000  0.0000  0.3088  1.0000  1.0000      62
```

```
# DT with ifelse
FL.dt[, demDummy:= ifelse(polity2 < 7 ,0,1)]
summary(FL.dt$demDummy)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  0.0000  0.0000  0.0000  0.3088  1.0000  1.0000      62
```

The `ifelse` command rolls an if-then-else statement into one command. It's nice because we don't have to initialize the variable because there was no indexing required, and we can do it one command.

We can also generate a whole set of dummies from a single variable (i.e. country or year dummies)

```
#base R
cDummies <- model.matrix(~factor(cname) - 1, data=FL2003)
FullData <- cbind(FL2003, cDummies)
colnames(FullData)[c(1:10, 55:80)] ##Take a look
```

```
##  [1] "year"                   "polity2"                "country"
##  [4] "cname"                  "cmark"                  "wars"
##  [7] "war"                    "warl"                   "onset"
## [10] "ethonset"               "factor(cname)ALGERIA"   "factor(cname)ANGOLA"
## [13] "factor(cname)ARGENTIN" "factor(cname)ARMENIA"   "factor(cname)AUSTRALI"
## [16] "factor(cname)AUSTRIA"  "factor(cname)AZERBAIJ" "factor(cname)BAHRAIN"
## [19] "factor(cname)BANGLADE" "factor(cname)BELARUS"   "factor(cname)BELGIUM"
## [22] "factor(cname)BENIN"    "factor(cname)BHUTAN"   "factor(cname)BOLIVIA"
## [25] "factor(cname)BOSNIA"   "factor(cname)BOTSWANA" "factor(cname)BRAZIL"
## [28] "factor(cname)BULGARIA" "factor(cname)BURKINA " "factor(cname)BURMA"
## [31] "factor(cname)BURUNDI"  "factor(cname)CAMBODIA" "factor(cname)CAMEROON"
## [34] "factor(cname)CANADA"   "factor(cname)CENTRAL " "factor(cname)CHAD"
```

```
# Tidy solution: hack pivot wider
FullData <- FL2003 %>%
  mutate(const=1) %>%
  pivot_wider(names_from = cname, values_from = const,
              names_prefix = "cname_", values_fill = 0)
colnames(FullData)[c(1:10, 55:80)] ##Take a look
```

```
##  [1] "year"           "polity2"        "country"        "cmark"
##  [5] "wars"           "war"            "warl"           "onset"
##  [9] "ethonset"       "durest"         "cname_HAITI"    "cname_DOMINICA"
## [13] "cname_JAMAICA"  "cname_TRINIDAD" "cname_MEXICO"   "cname_GUATEMAL"
## [17] "cname_HONDURAS" "cname_EL SALVA" "cname_NICARAGU" "cname_COSTARIC"
## [21] "cname_PANAMA"   "cname_COLOMBIA" "cname_VENEZUEL" "cname_GUYANA"
## [25] "cname_ECUADOR"  "cname_PERU"     "cname_BRAZIL"   "cname_BOLIVIA"
## [29] "cname_PARAGUAY" "cname_CHILE"    "cname_ARGENTIN" "cname_URUGUAY"
## [33] "cname_UK"       "cname_IRELAND"  "cname_NETHERLA" "cname_BELGIUM"
```

The command `model.matrix` uses what's called a formula in R. We'll go in to formulas more extensively when we start estimating models, but for now I'll note that the above command is an internal function that R uses when it's getting ready to create a matrix of variables whenever it runs a regression. We just borrowed it for making dummies. Formulas for regression take the form `y ~ X`. So the above formula has no dependent variable, country dummies as the only independent variables, and no constant (the -1 term). Including no constant means that it generated a dummy for all the countries (with a constant it would drop one).

3.6 APPLICATION: Aggregating or summarizing Data

As a final application, there may be a situation where you have data that you want to aggregate in different ways. Here the tidy verse and data tables will be your friend.

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.

```r
###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
system.time({
  output <- newDat %>%
    group_by(ccode, year) %>%
    summarise(SumVar = sum(Var1)) %>%
    ungroup()
})
```

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

```
##    user  system elapsed
##   0.036   0.000   0.137
```

```r
output
```

```
## # A tibble: 25 x 3
##    ccode  year  SumVar
##    <int> <int>   <dbl>
##  1     1     1 -1.49
##  2     1     2 -3.61
##  3     1     3 -0.703
##  4     1     4  0.331
##  5     1     5 -0.0586
##  6     2     1 -0.824
##  7     2     2  1.37
##  8     2     3 -0.590
##  9     2     4 -0.364
## 10     2     5 -1.40
## # i 15 more rows
```

```r
 #can you figure out the difference between mutate and summarize?

system.time({
  # data table approach
  newDt <- data.table(newDat)
  out.dt <- newDt[, .(SumVar = sum(Var1)), ##New variable with definition
                  by=list(ccode, year)] ##aggregate over these variables
})
```

```
##    user  system elapsed
##   0.001   0.000   0.001
```

```r
all(output$SumVar==out.dt$SumVar)
```

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

3.7 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.

```r
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.

```r
ls() #Everything
```

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

```r
rm(list=ls())
ls() #Nothing
```

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

```r
load('Datasets/DataFrames.Rdata')
ls() #It's all back
```

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

3.8 Exercises

This was probably the hardest section to create notes for because when it comes to manipulating data there are so many different ways to do the same thing, and there are so many possible tasks that could come up. The only way to really get the hang of data manipulation in R is to have a project where you do everything in R.

1. This exercise focuses on read data and manipulating it. In order to get the most out of it make sure that you're starting with an empty work space and no extra packages loaded. Try to load only the packages you need.

   a. Read in the Freedom House press data and the Fearon and Laitin data.
   b. Reshape the FH data into country-year format. Make the year variable a numeric value with no leading "X". (If the "X" is giving you trouble look up `gsub`)
   c. Install and load the package `countrycode`. Use the command `?` to figure out how to use the function `countrycode`. Make a variable called `ccode` in your press data that has COW country codes for each country. Look up any `NA`s that are returned and fill them in using the COW state list file (`states2016.csv`) in the datasets folder (HINT: only Serbia and Serbia and Montenergo need to be fixed).
   d. Once you have that merge it with the Fearon and Laitin data by ccode and year. Make sure that the number of rows in the merged data matches the number of rows in the original Fearon and Laitin data. Go back and see if you can solve any discrepancies or duplicates.
   e. Find the average polity2 score by region.

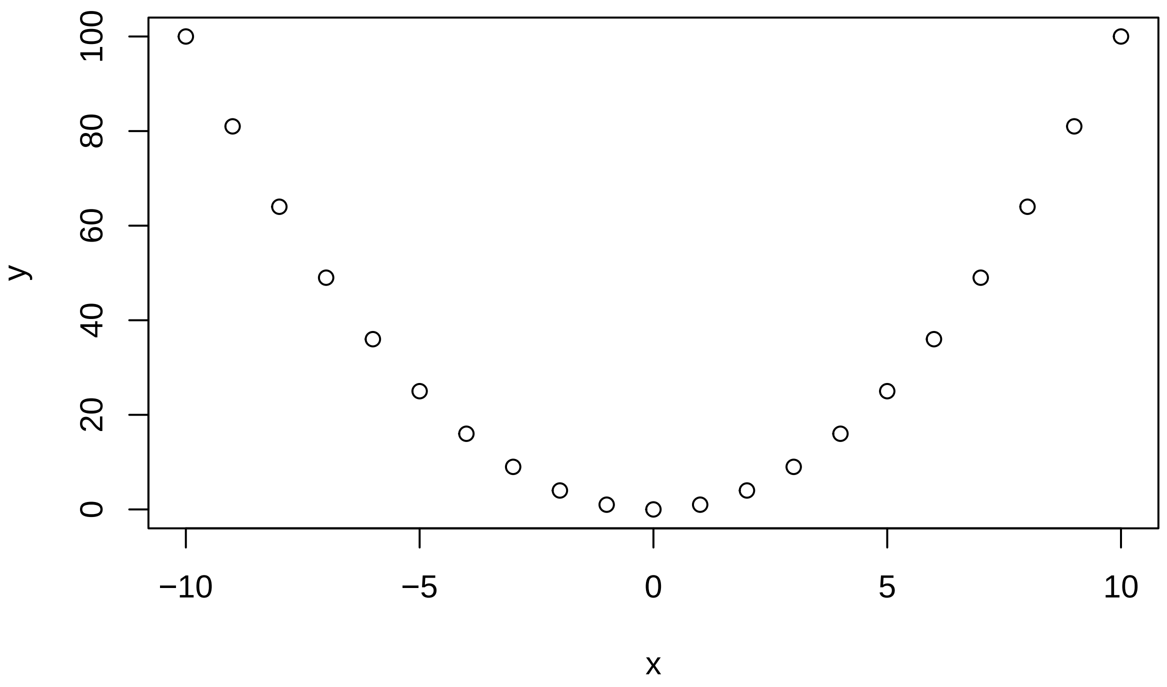
4 Plotting

Today we'll be looking at graphics in R. 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. Most of the grad students and other R enthusiasts like to use `ggplot` because it produces nice looking plots, it's more consistent in syntax across difference 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). We'll also use the `gridExtra` library to arrange multiple plots into a single figure.

4.1 Basic Plots

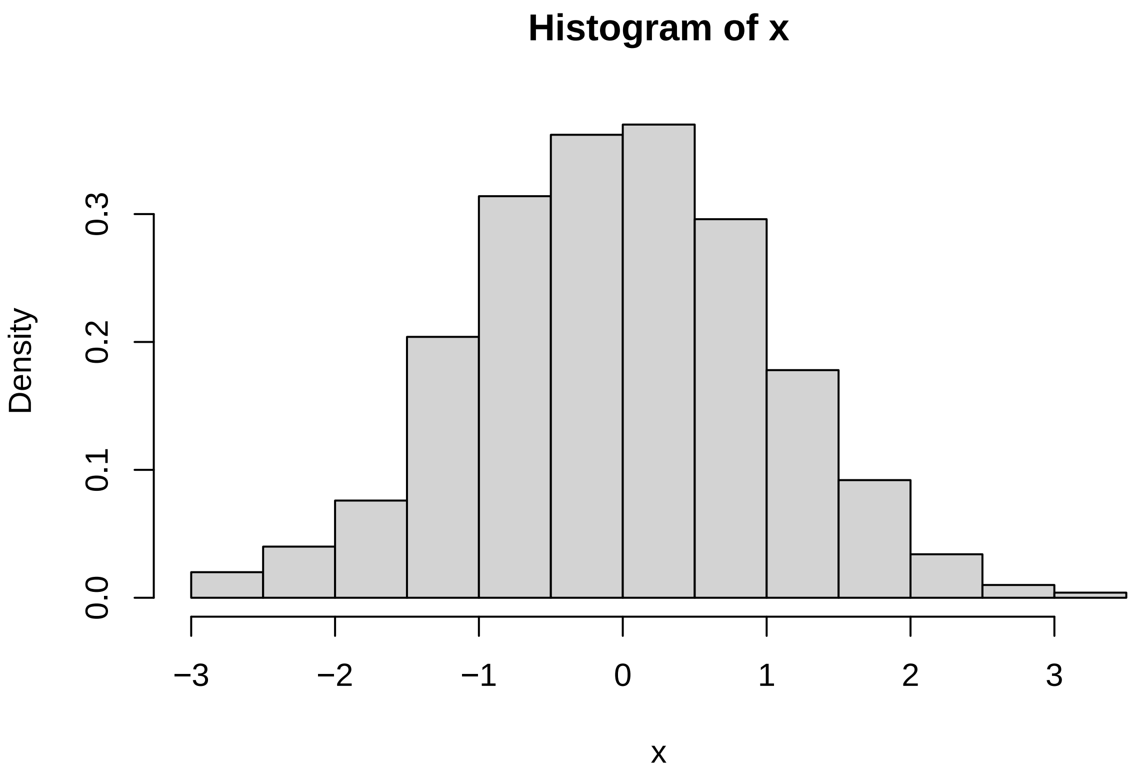
Despite the good things about ggplot sometimes is nice to do some some basic, exploratory plots with base graphics.

```
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:

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



4.2 Scatterplots and Layers

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

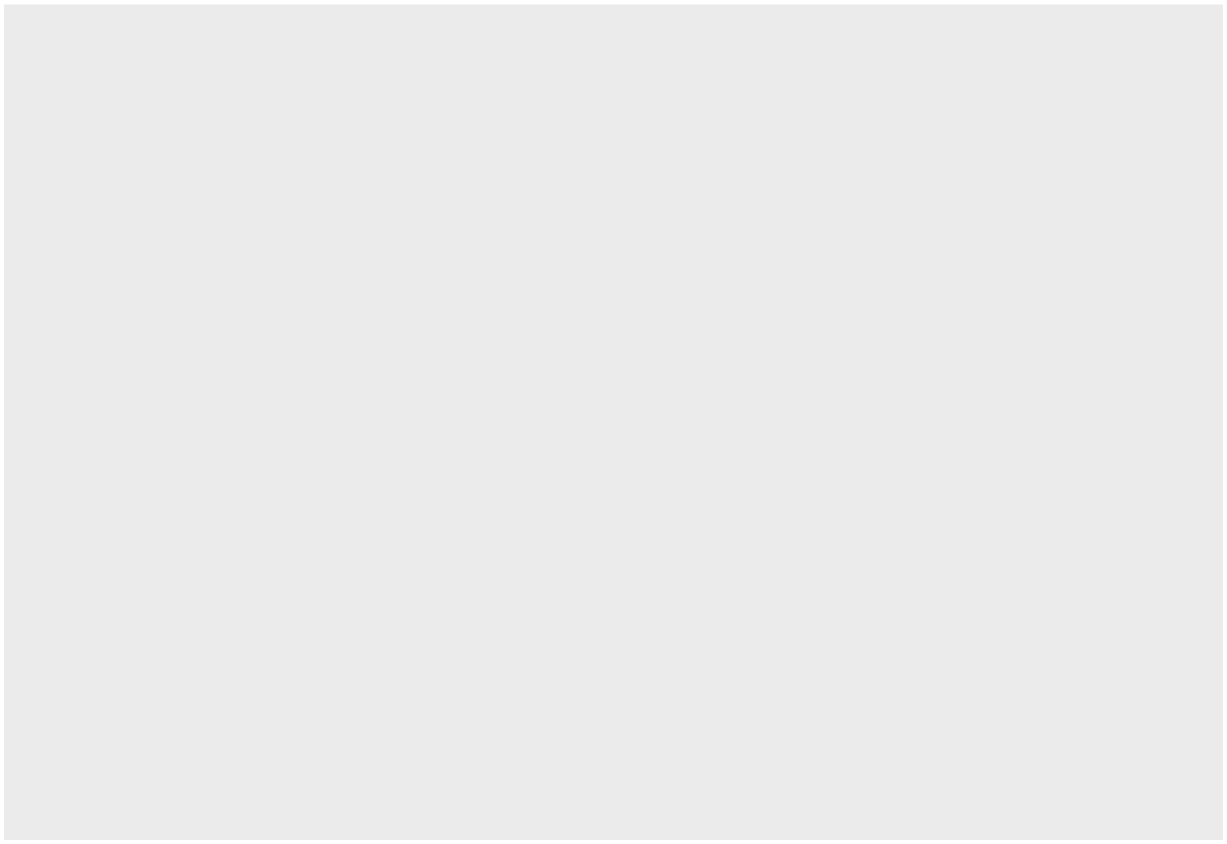
```r
library(ggplot2)

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.

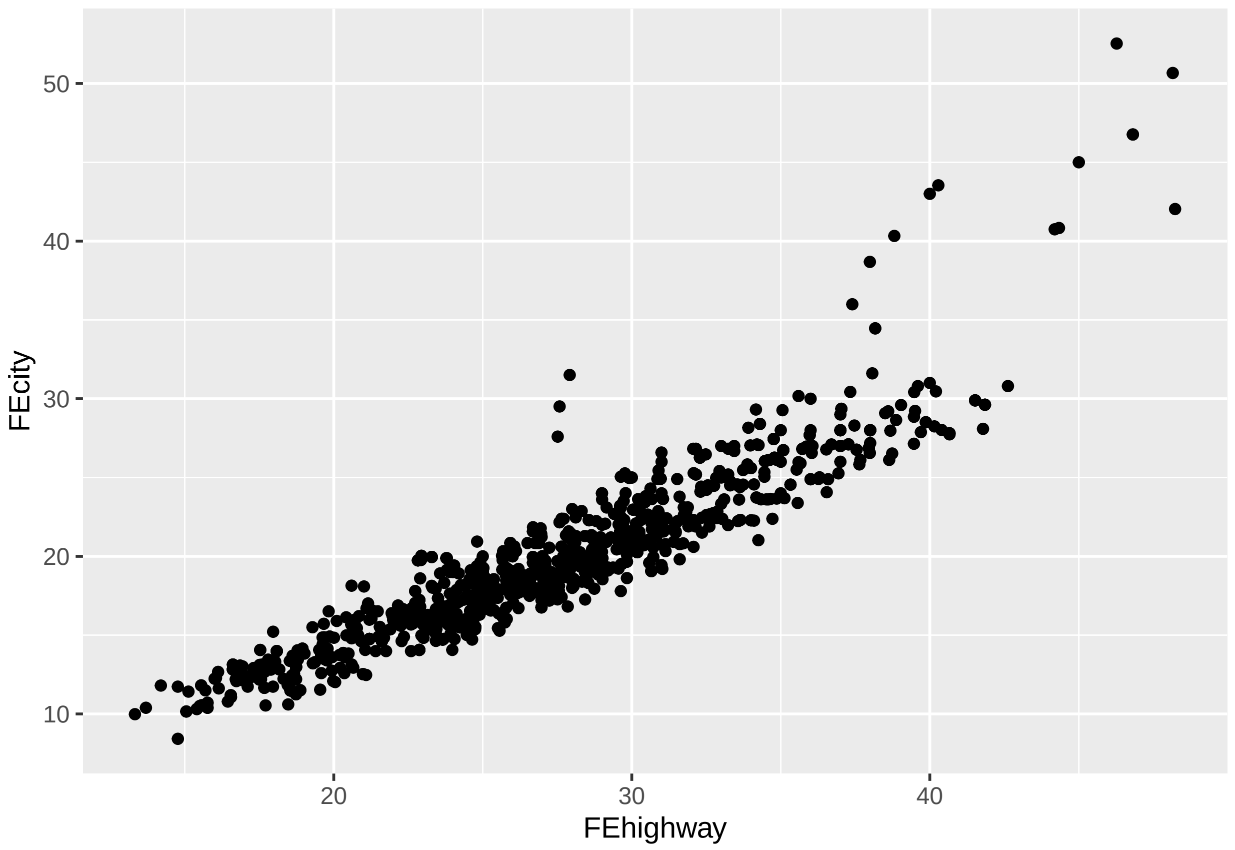
Note *`ggplot` works best with data frame and data table objects.*

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



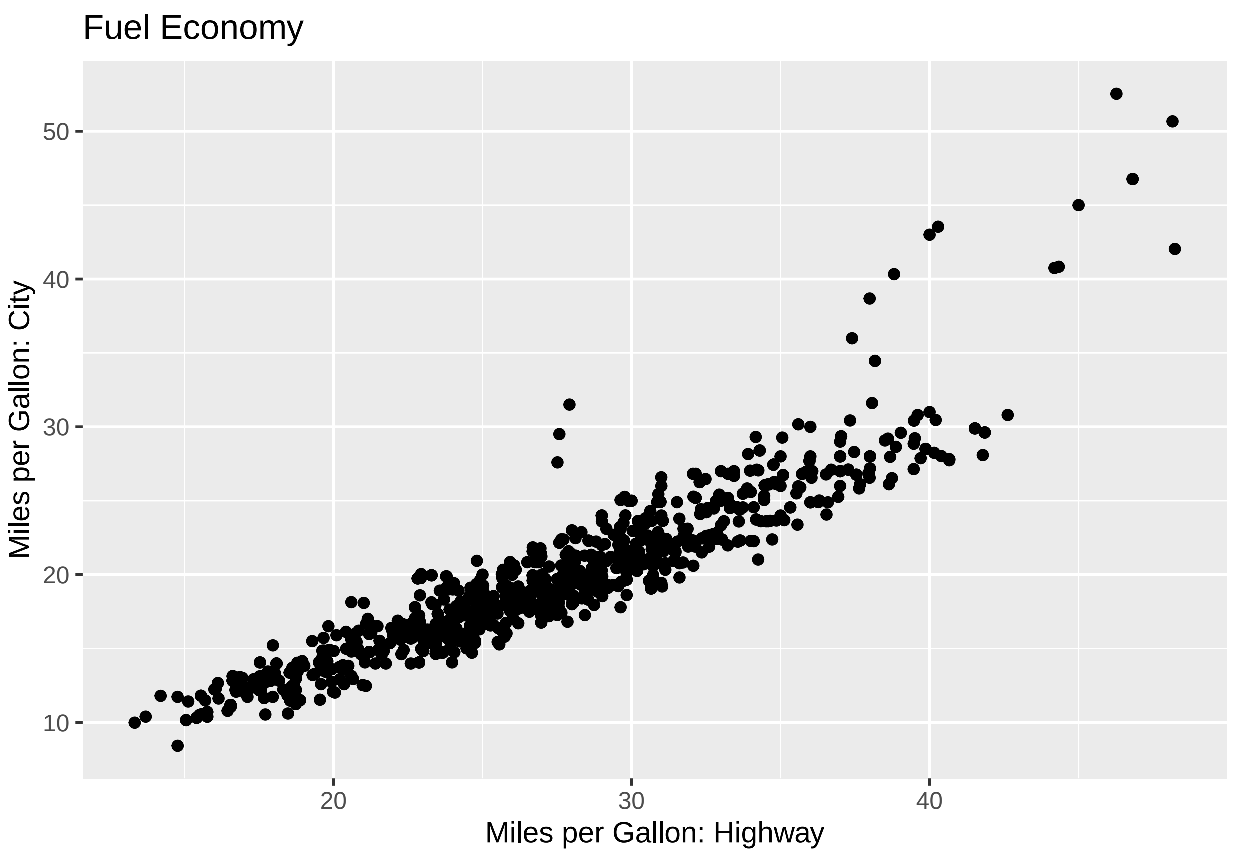
To create the scatterplot we need to add that layer to the plot

```r
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)
```



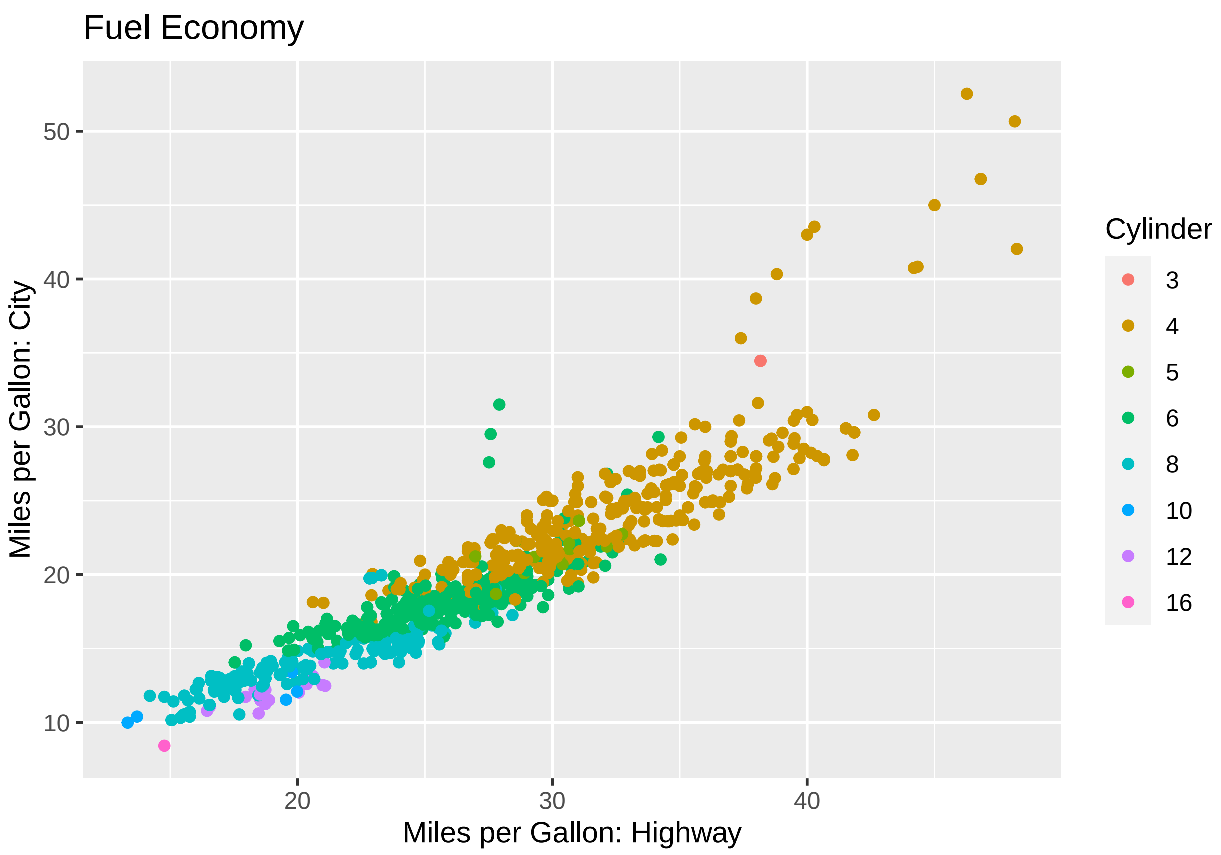
It's pretty straight forward to make changes to plot once it's created. Say we wanted to add a title and change the axis labels.

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

print(plot1)
```



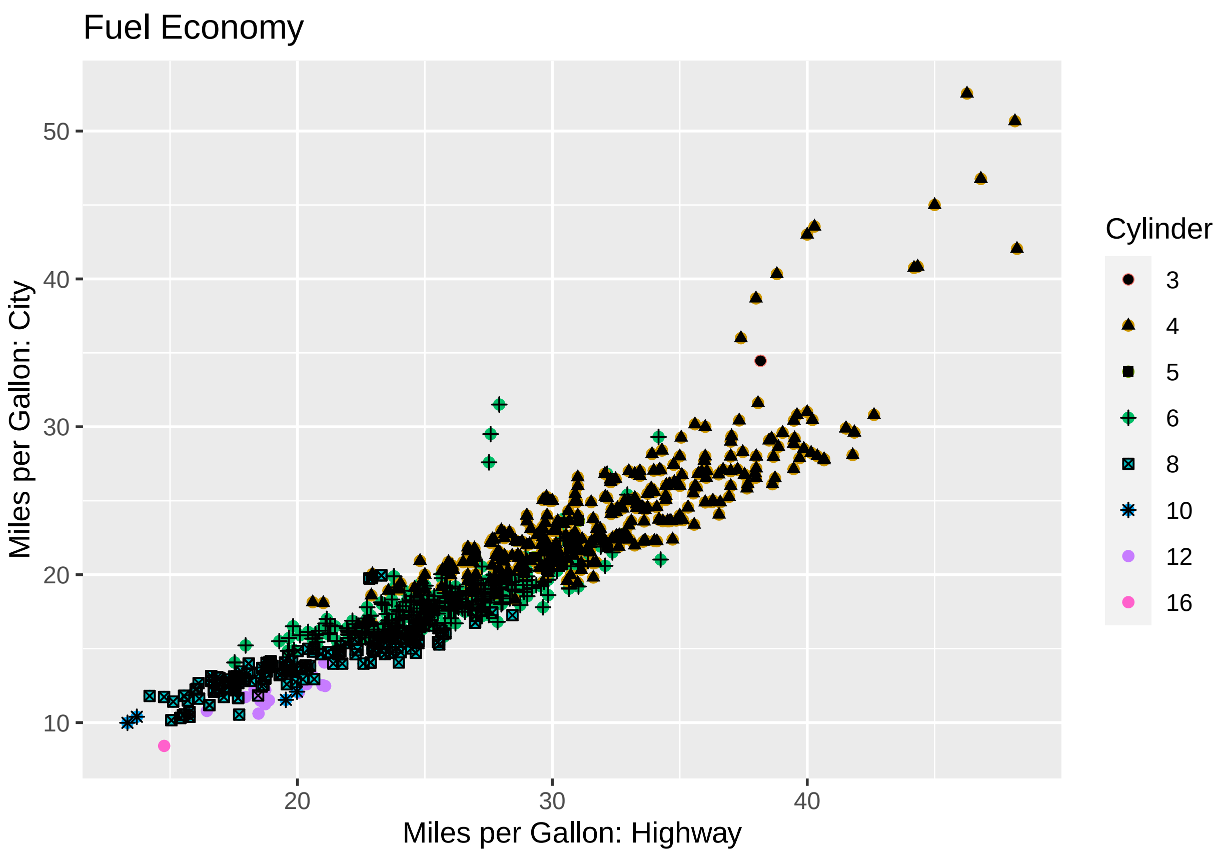
We can add color to the plot by including a factor variable. In this case, let's color the observations by number of cylinders.

```
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; you have 8. Consider
## specifying shapes manually if you must have them.

## Warning: Removed 23 rows containing missing values (`geom_point()`).
```

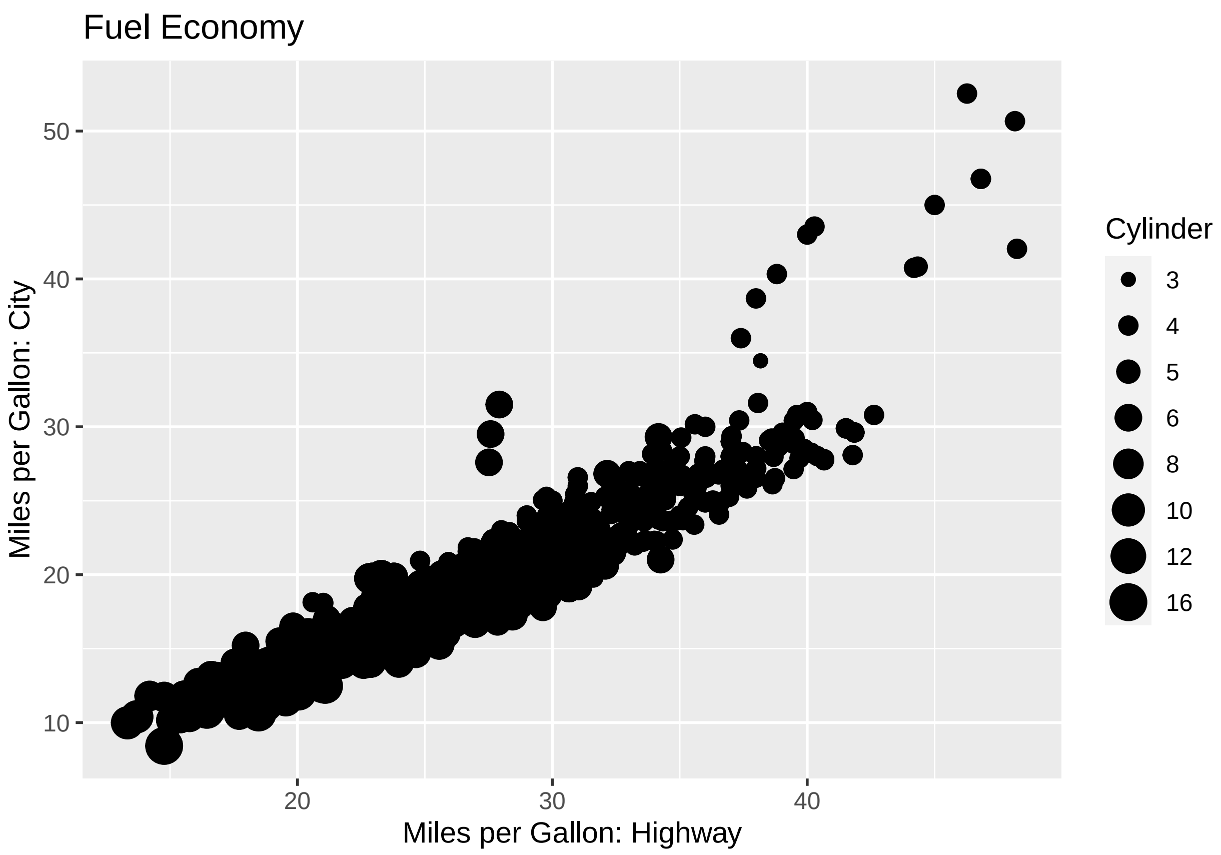


```r
##Or sizes
plot1 <- plot1 +
          geom_point(aes(x=FEhighway, y  = FEcity, size= Cylinder))
print(plot1)
```

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

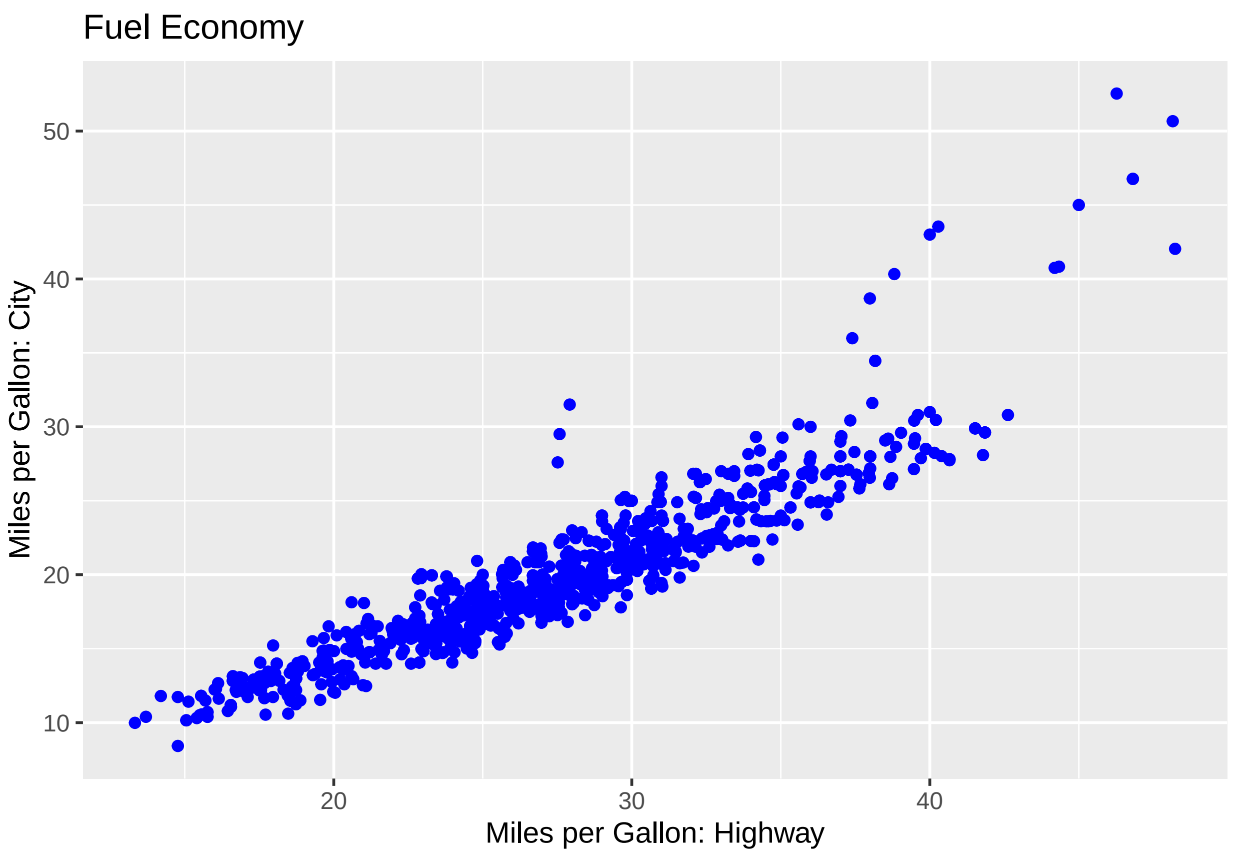
```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 8. Consider
## specifying shapes manually if you must have them.
```

```
## Warning: Removed 23 rows containing missing values (`geom_point()`).
```

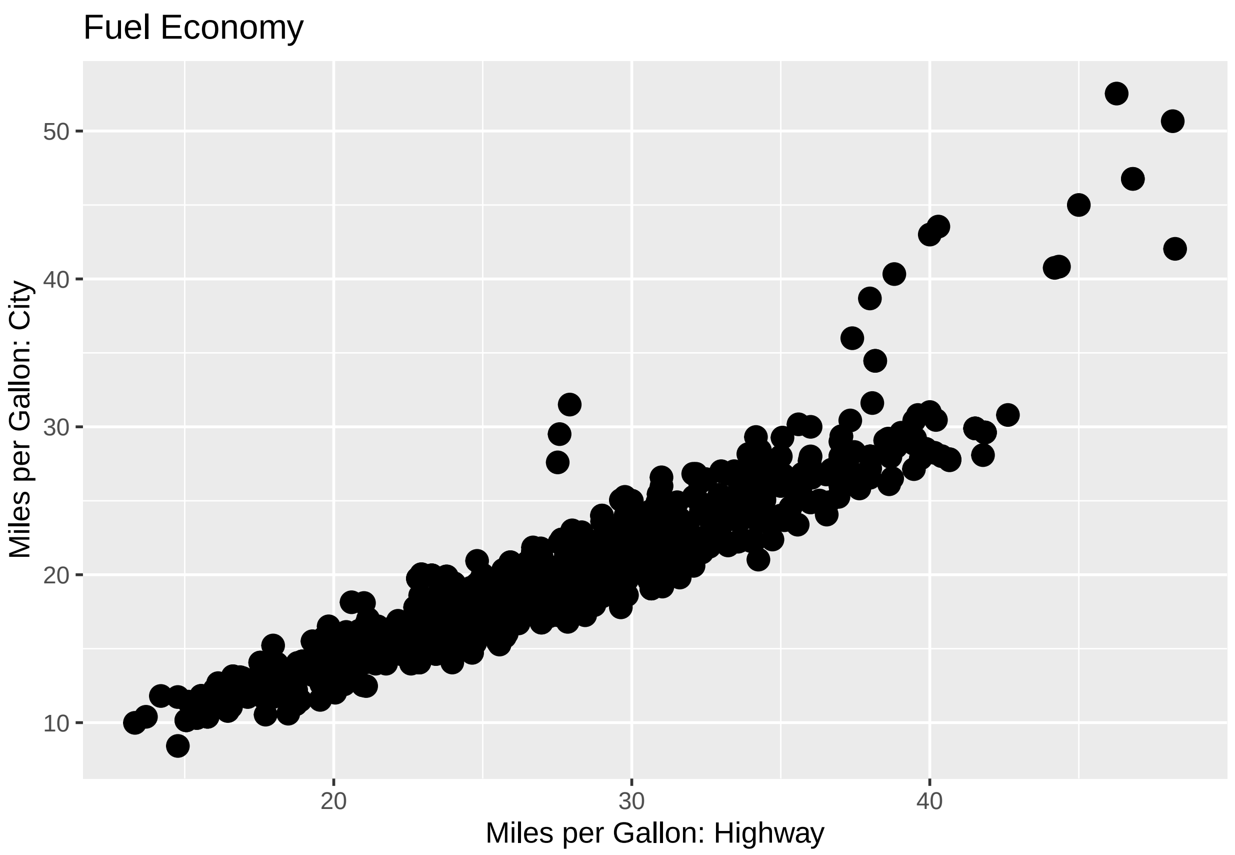


Alternatively we can adjust the color, shape, and size all the points if we do it within `geom_point` and outside ‘aes“‘

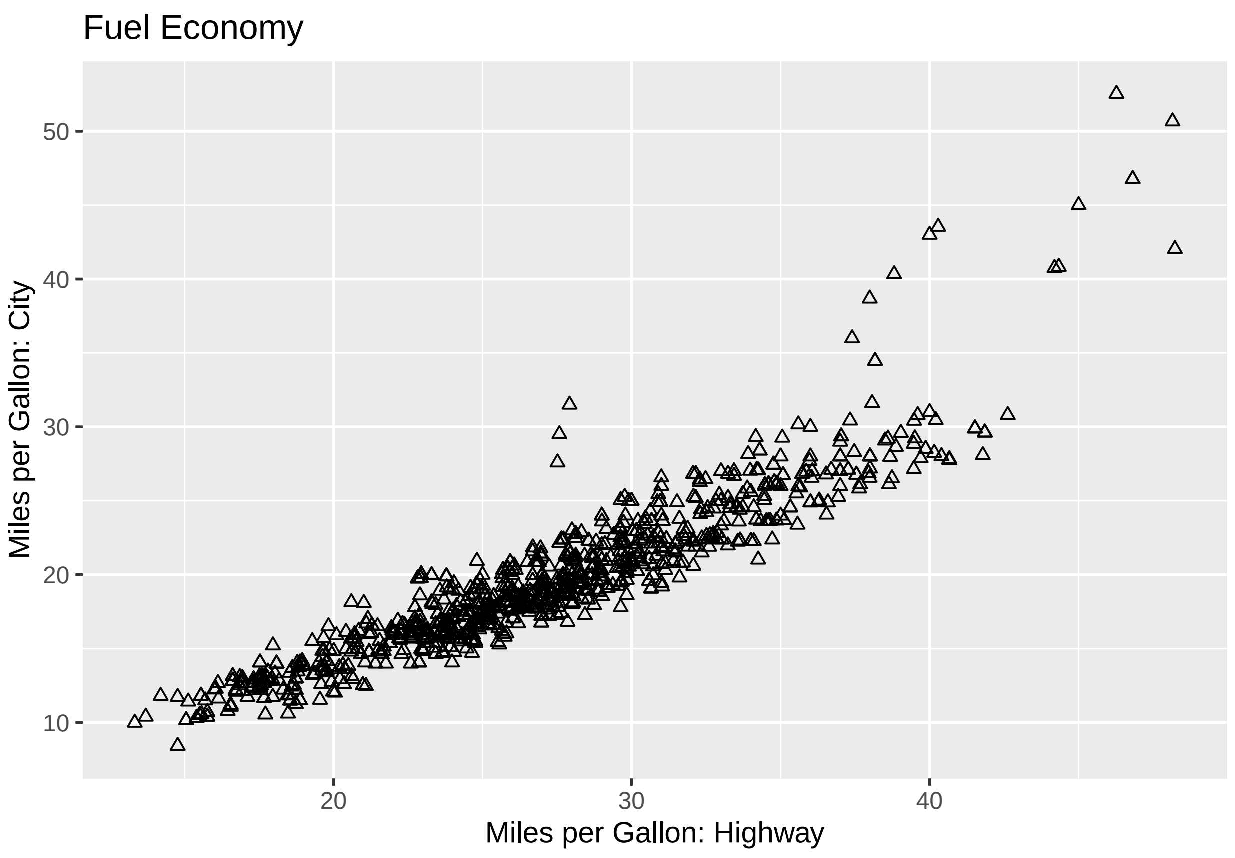
```
plot1 <- ggplot(FE2013) +
          geom_point(aes(x=FEhighway, y  = FEcity), color="blue")+
          xlab('Miles per Gallon: Highway')+
          ylab('Miles per Gallon: City')+
          ggtitle('Fuel Economy')
print(plot1)
```



```
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)
```

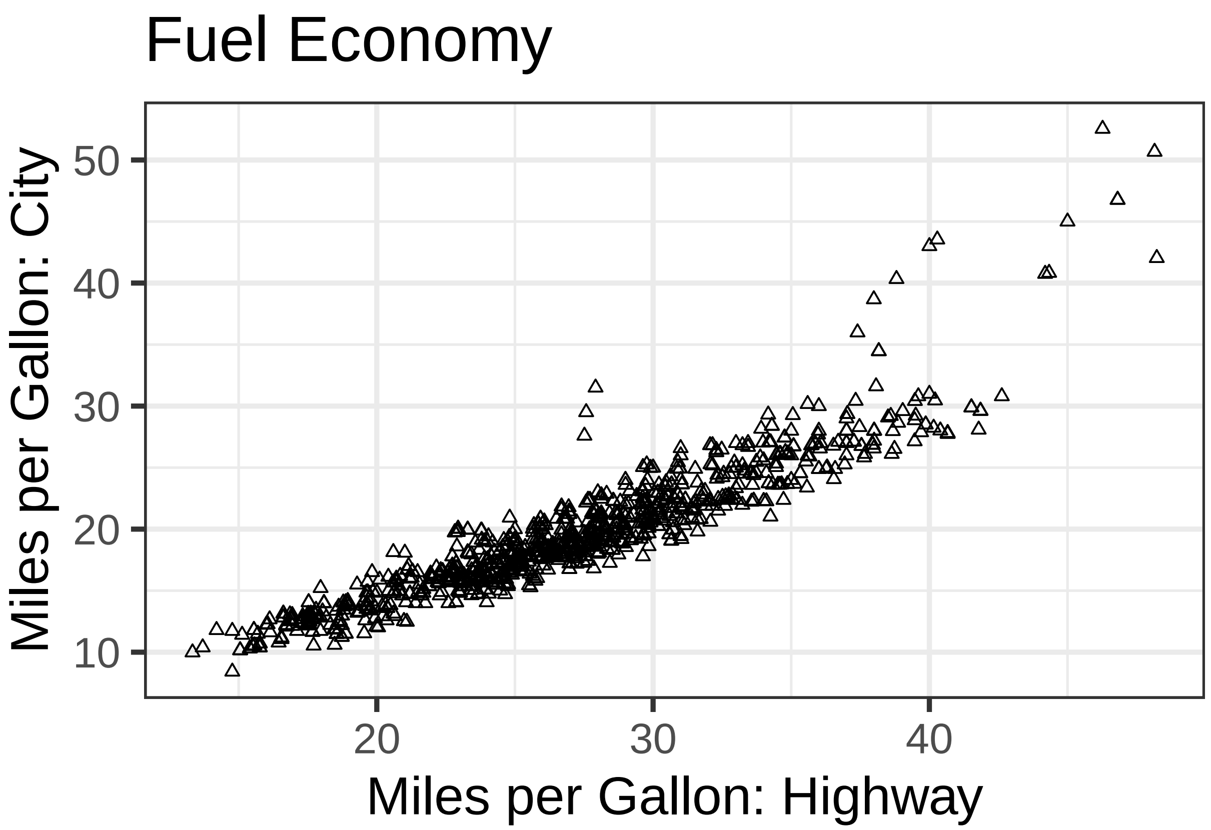


```
plot1 <- ggplot(FE2013) +
          geom_point(aes(x=FEhighway, y  = FEcity), pch=24)+
          xlab('Miles per Gallon: Highway')+
          ylab('Miles per Gallon: City')+
          ggtitle('Fuel Economy')
print(plot1)
```



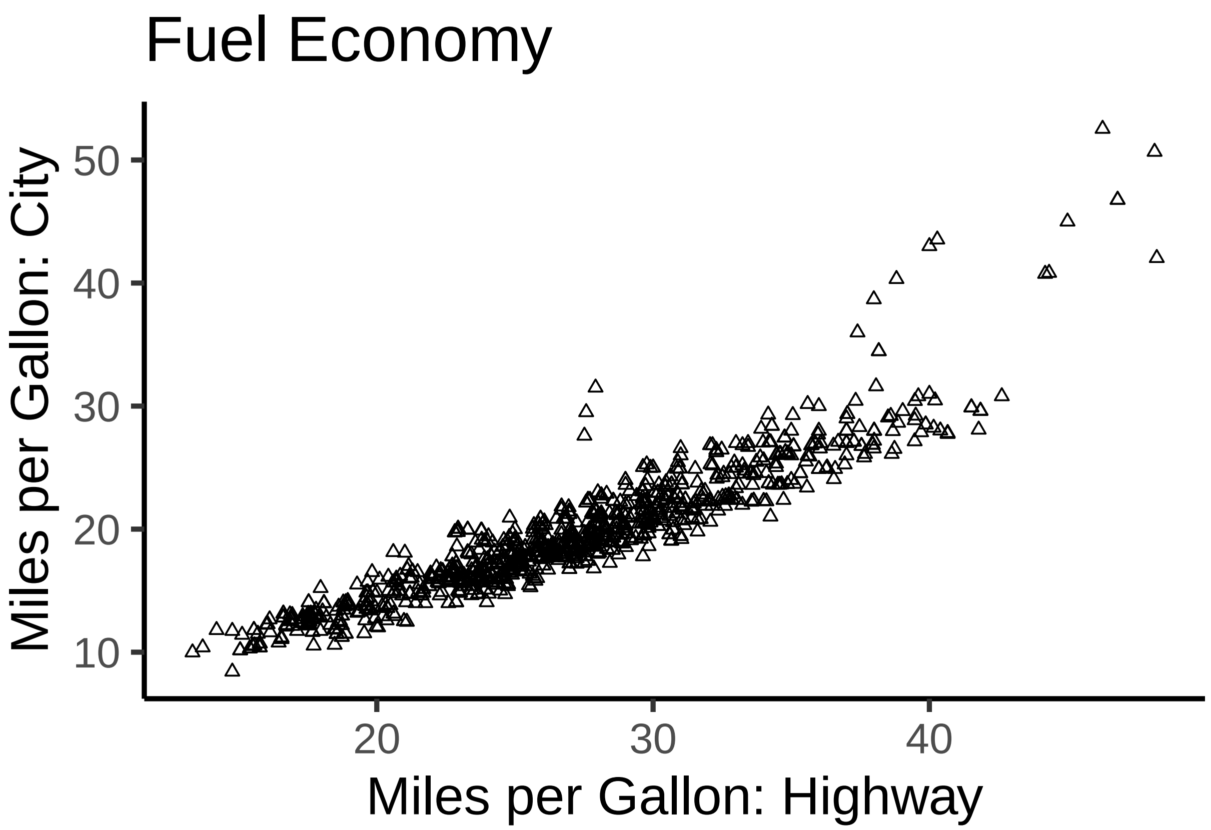
We can also change the background theme and the font side.

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

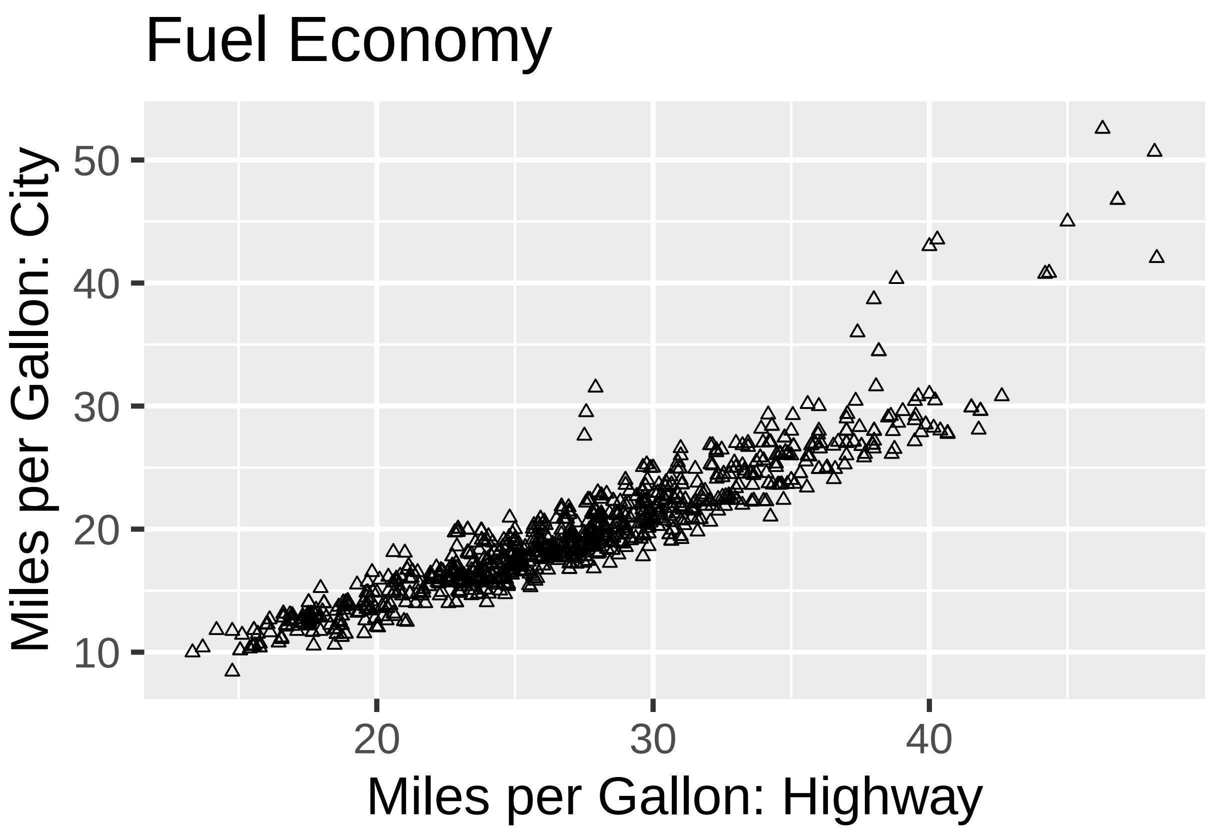


```r
##theme_bw changes the color theme,
##20 means 20pt font
```

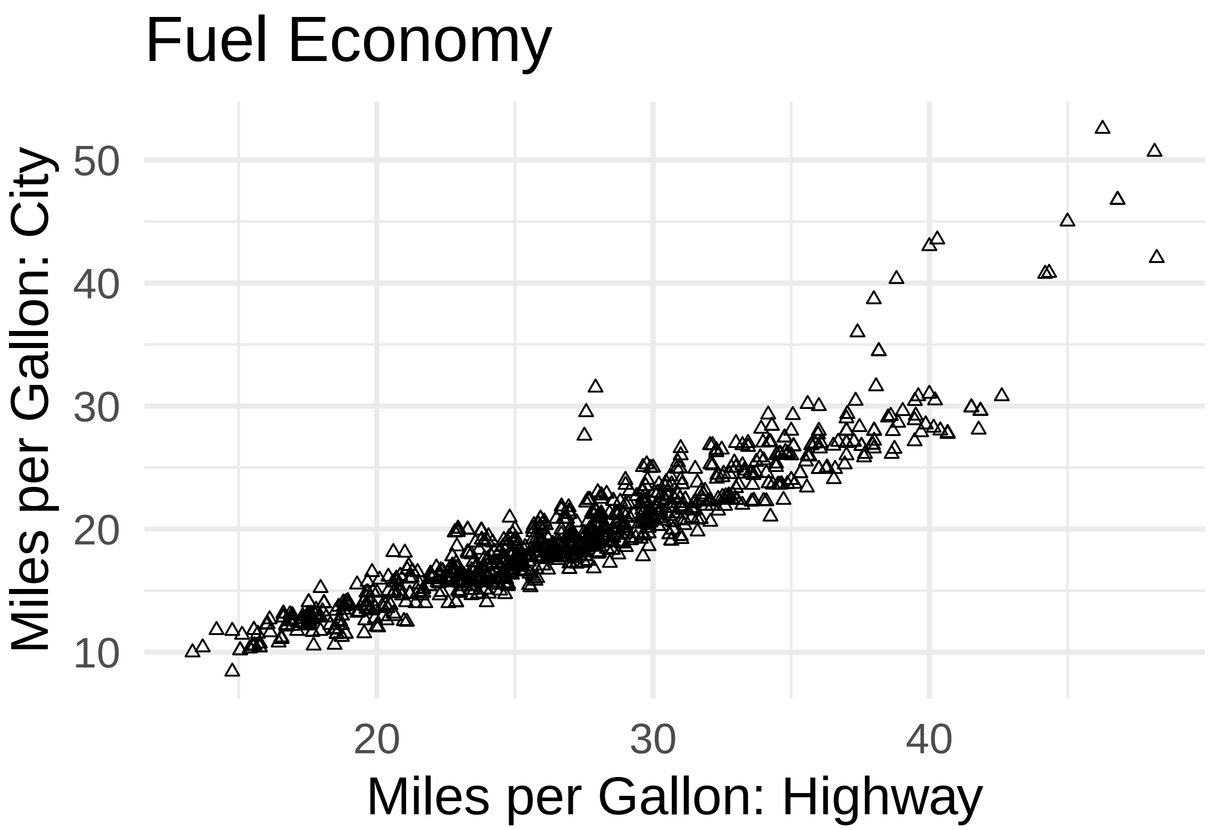
```r
##Also
plot1 +
  theme_classic(20)
```



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



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



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

4.3 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.

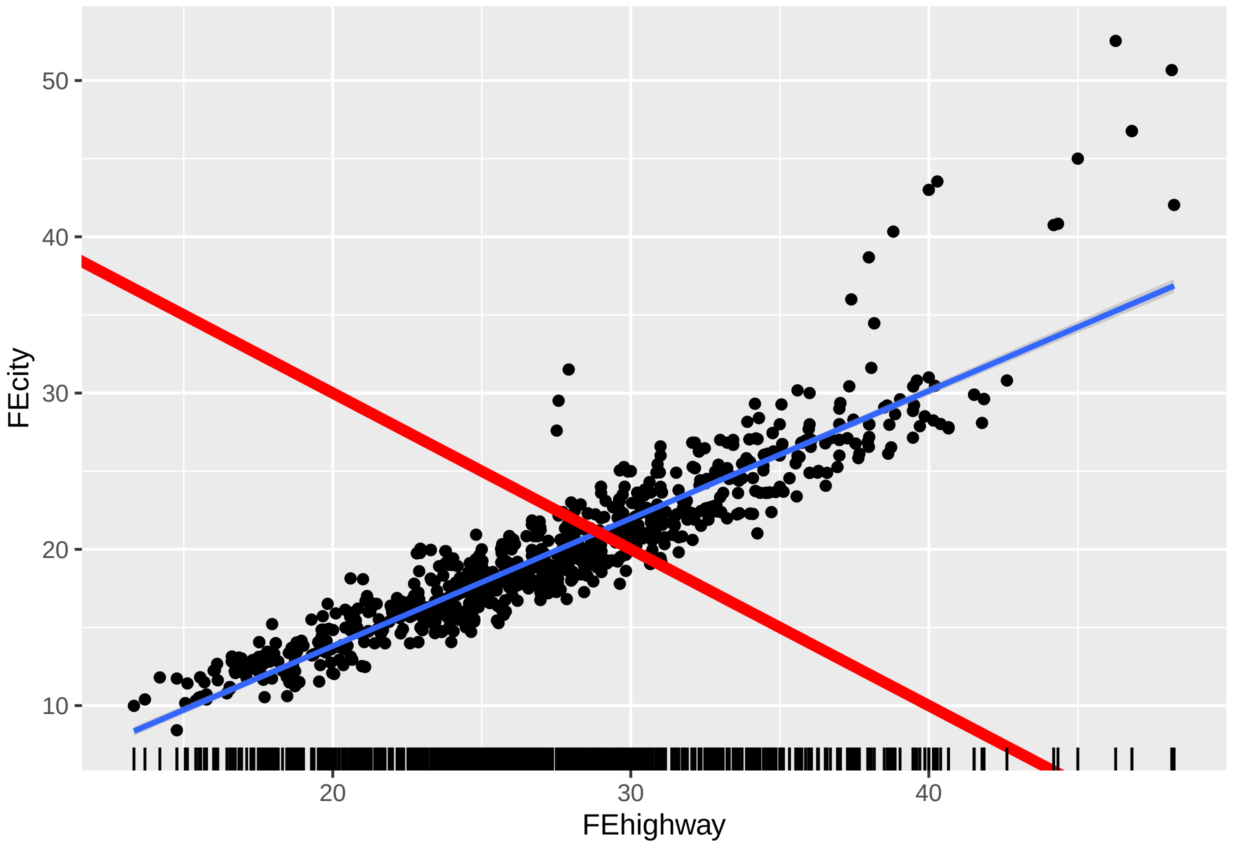
Note *In this plot we will specify* `aes` *in the the initialization step, this specifies it as a global option. In other words it's the same as entering into each geom individually.*

```
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', size=1)+ ##best fit line, size 1
            geom_abline(intercept=50, slope=-1, color='red',
                        size=2)+ ##line
            geom_rug(sides='b')##just across the bottom
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```r
print(plot2)
```

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



In theory we could keep adding on and on.

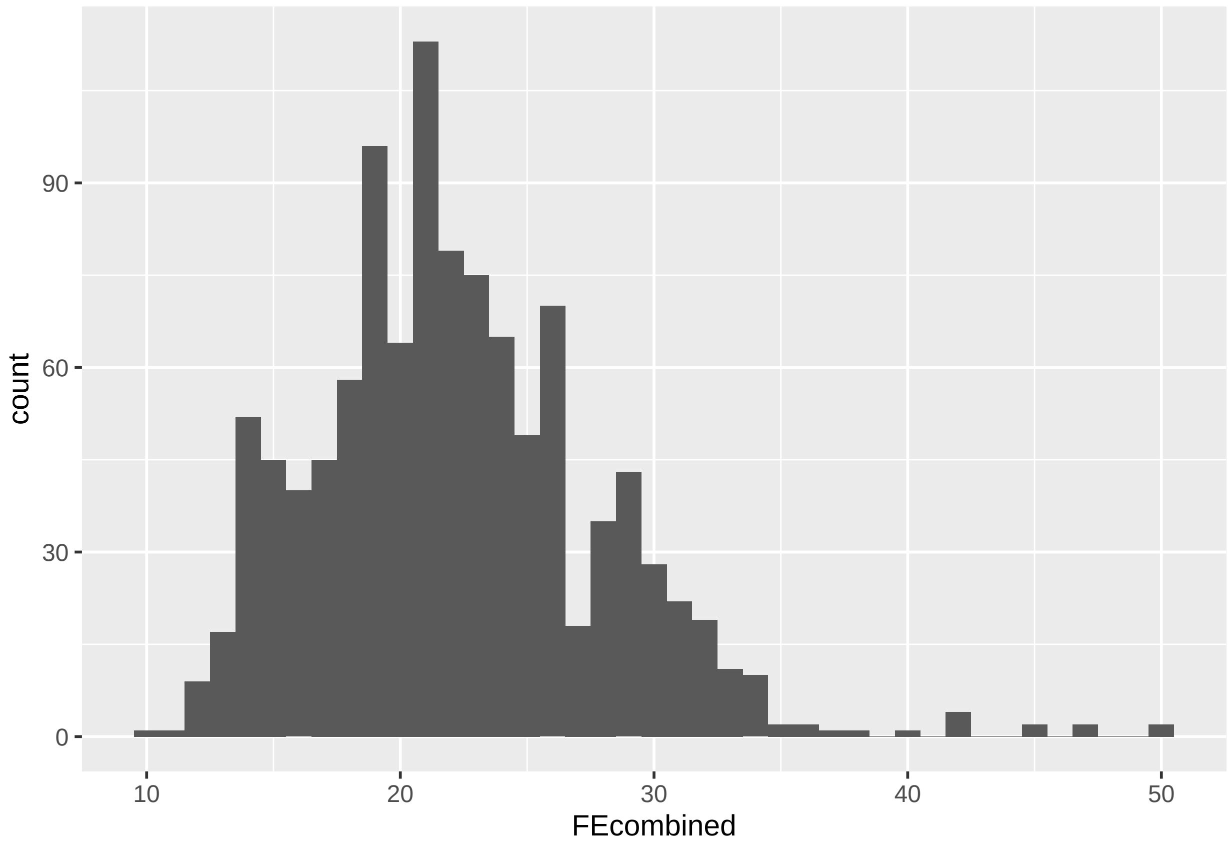
4.4 Special plots

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.

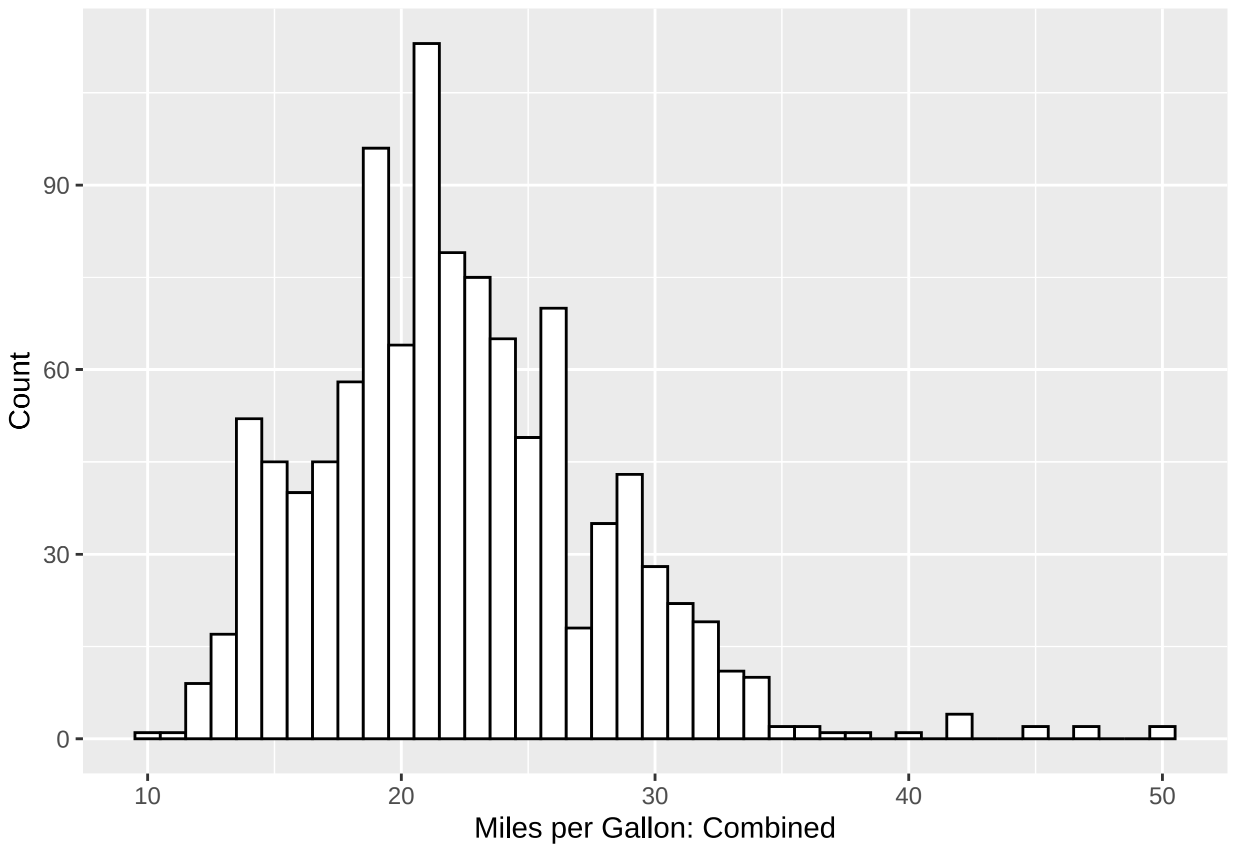
4.4.1 Histogram

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)
```



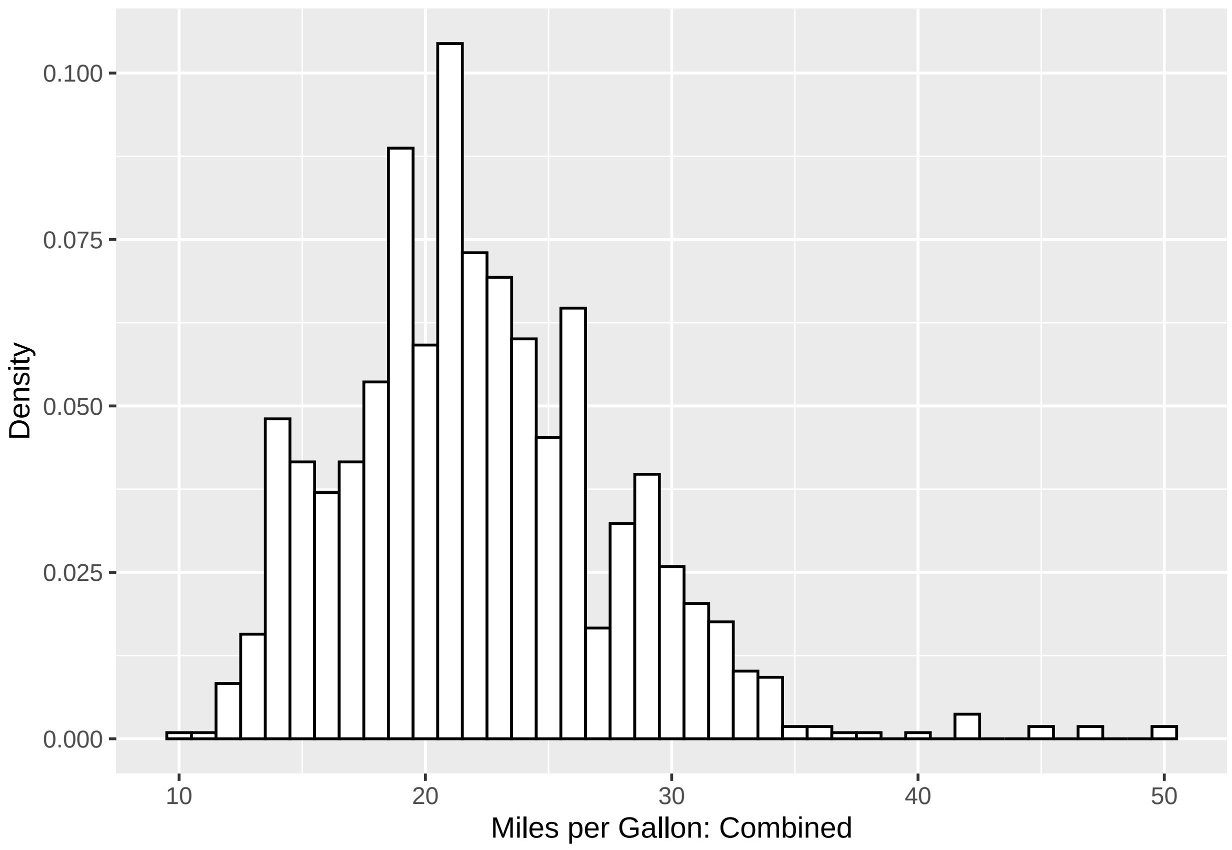
```
plot3 <- plot3 +
          geom_histogram(binwidth=1,
                         color='black', ##outline
                         fill='white')+ ##Inside
          ylab('Count')+
          xlab('Miles per Gallon: Combined')
print(plot3)
```



We can change counts in the histogram to density

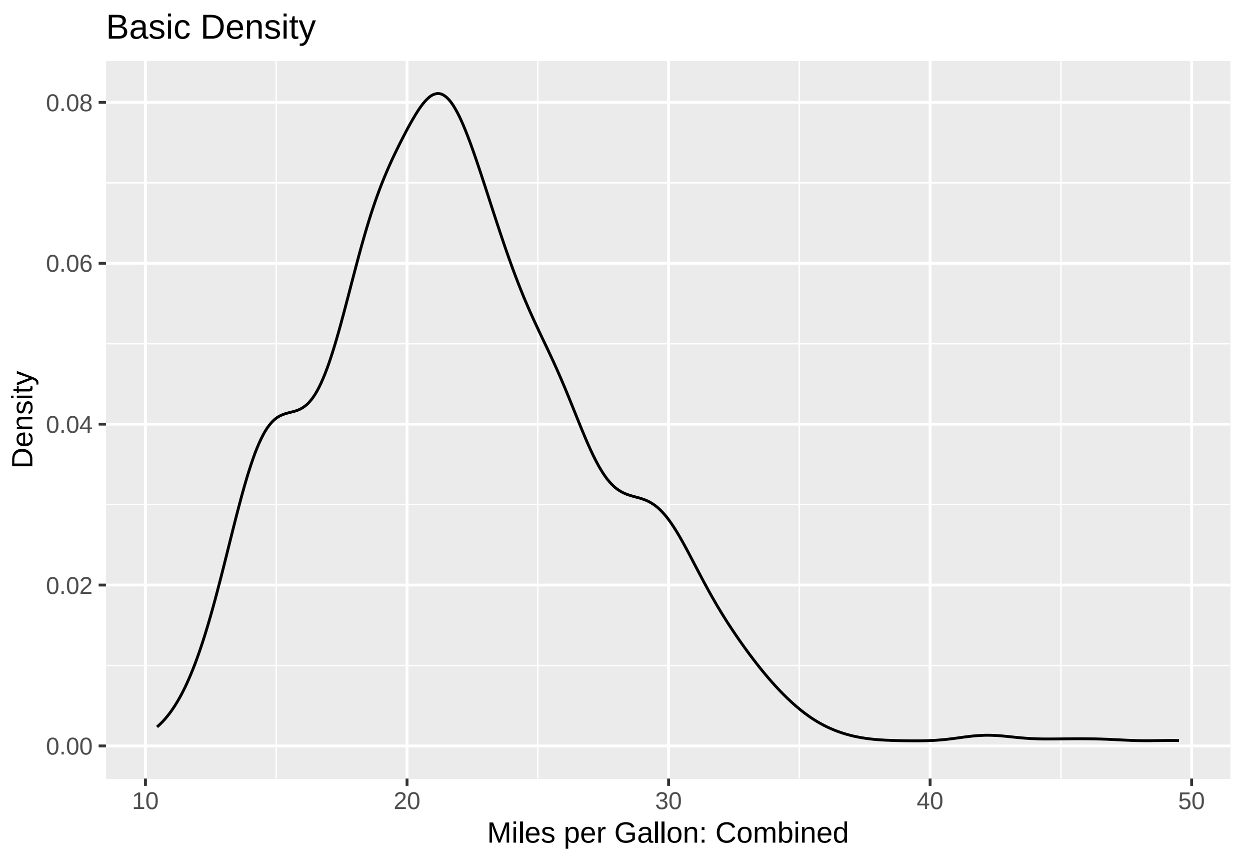
```
plot3 <- ggplot(FE2013, aes(x=FEcombined))+ ##only need x for hist
            geom_histogram(binwidth=1,
                           color='black', ##outline
                           fill='white', ##Inside
                           aes(y=..density..))+##call aes again
            ylab('Density')+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


print(plot3)
```

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



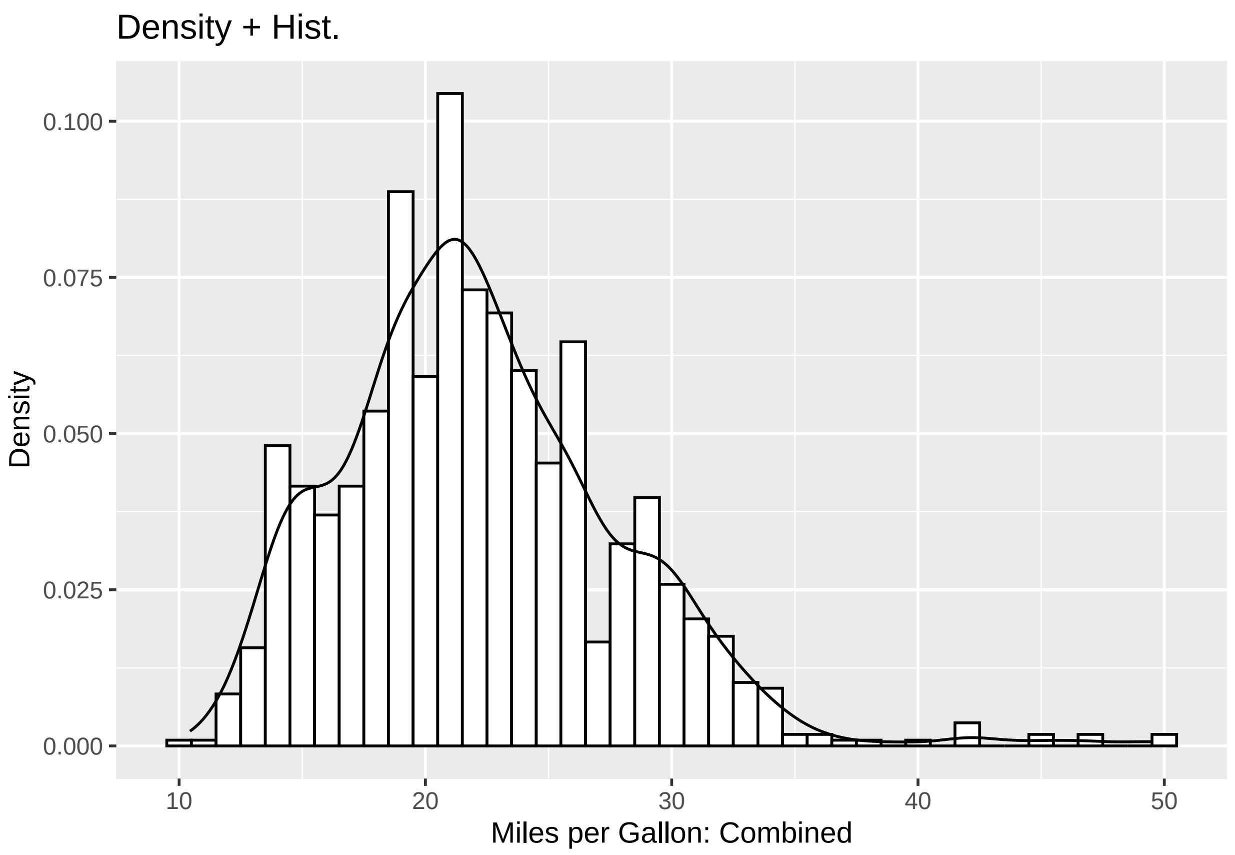
### Density We'll now look at density plots

```
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)
```



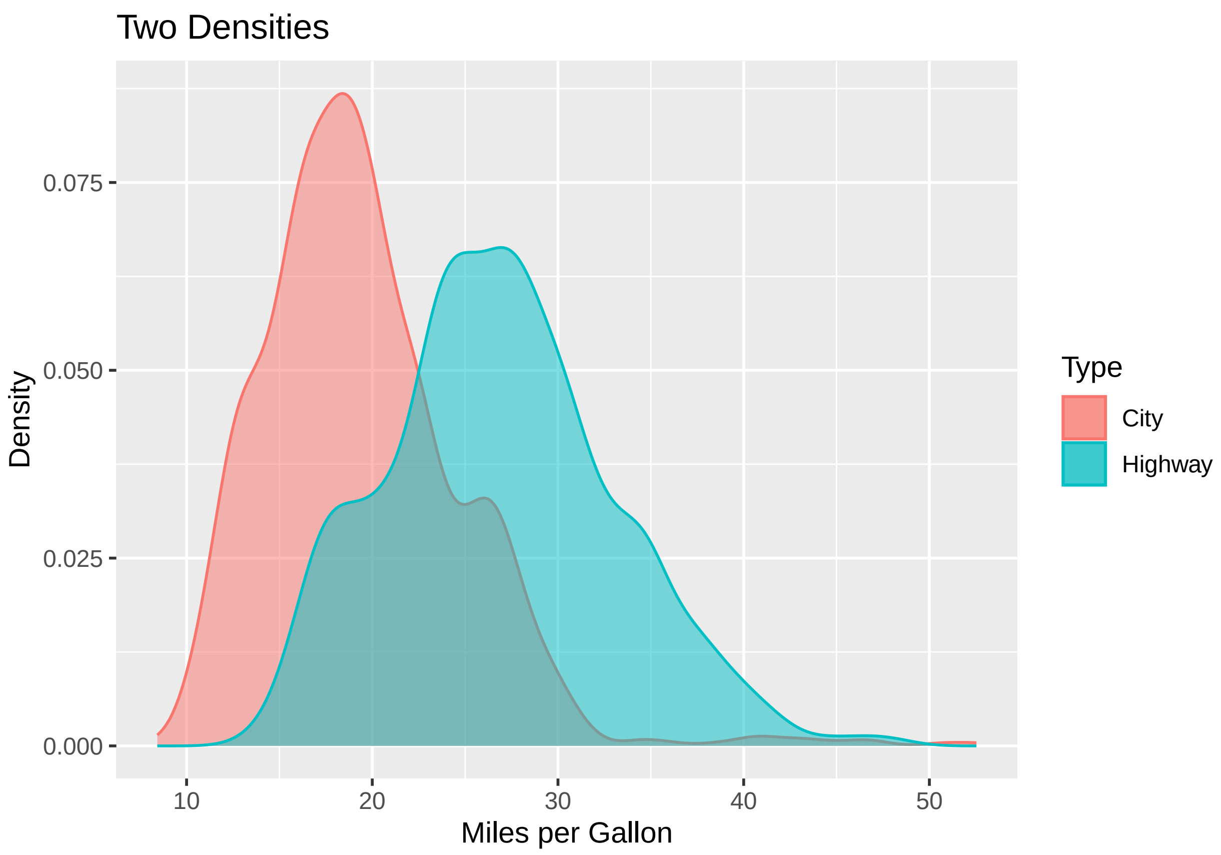
We can see that it matches by overlapping them

```
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 lenged for us.)*

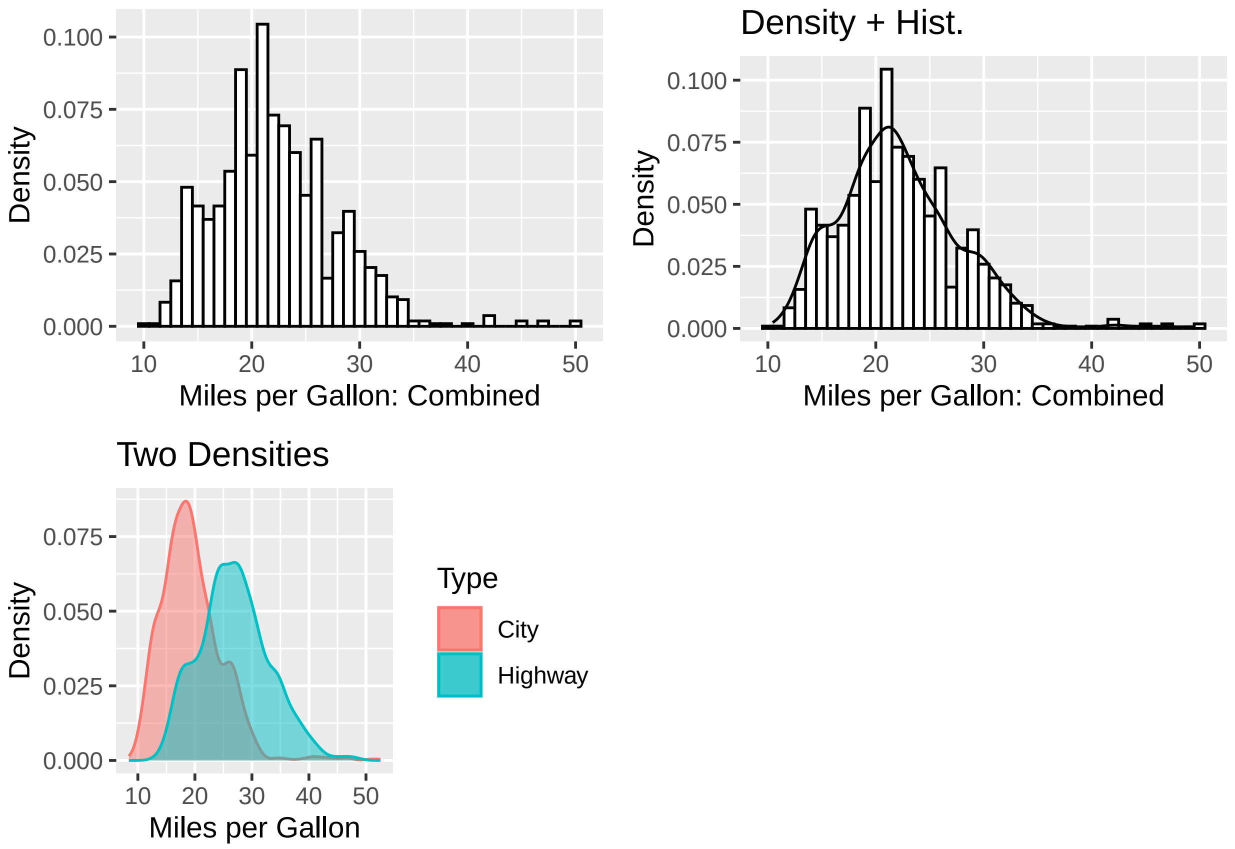
4.5 Stacking Plots

We can arrange multiple plots on a single page using the `gridExtra` package

```
library(gridExtra)
```

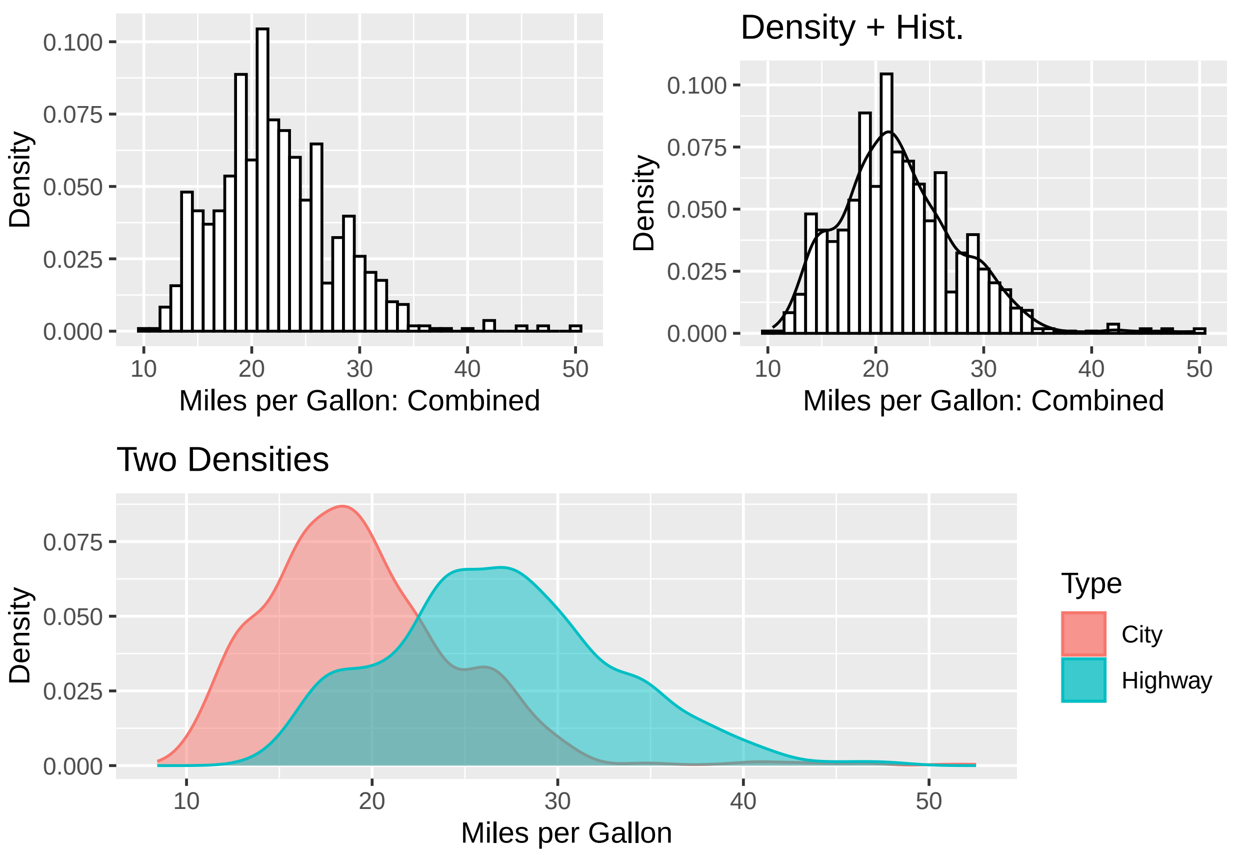
```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine
```

```
grid.arrange(plot3, plot4, plot5, ncol=2)
```



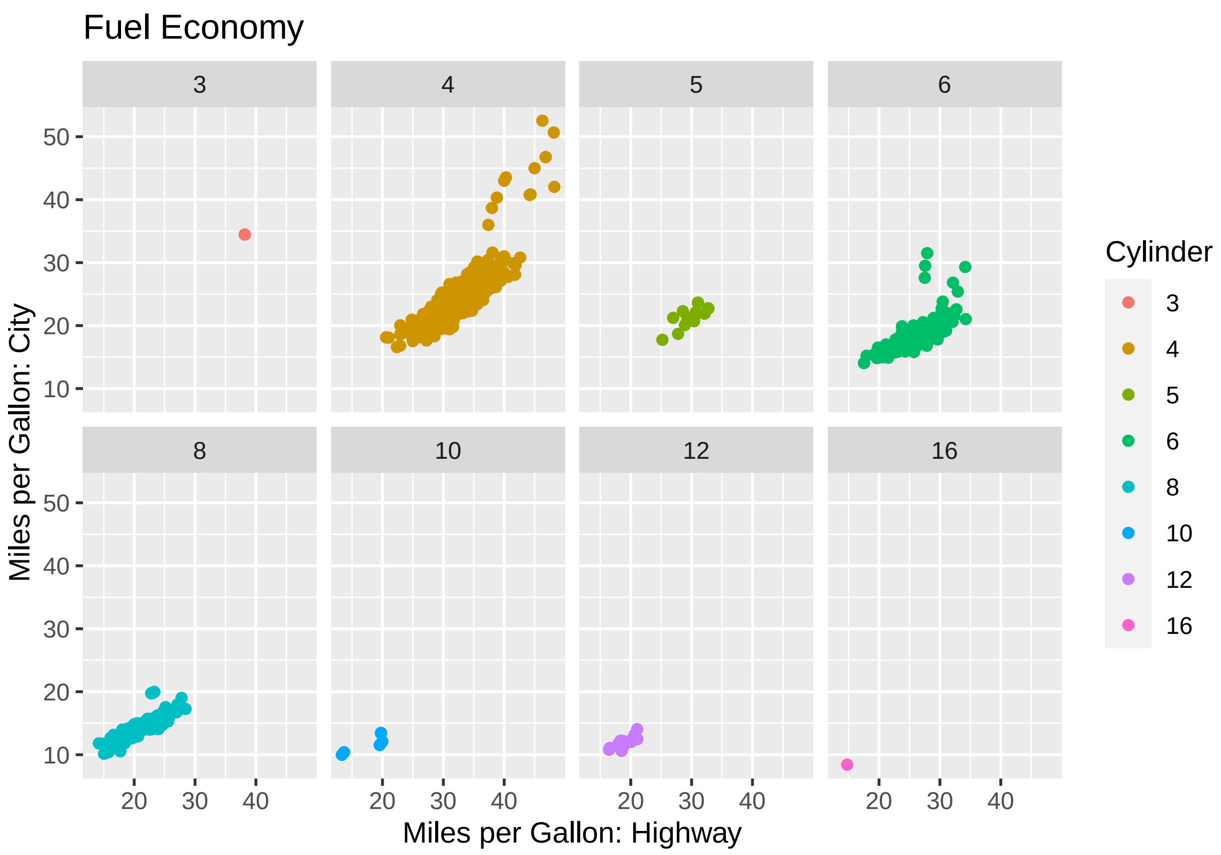
You may not like the look of this so we can adjust that bottom plot

```
grid.arrange(arrangeGrob(plot3, plot4, nrow=1),
             plot5,
             nrow=2)
```



Here the `arrangeGrob` is being used to combine the first two plots into a single graphic object (or Grob) that is then stacked on top of the other. If you want to split a plot across a specific variable we can do that with the `facet_wrap` command

```
plot1.faceted <- 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')+
  facet_wrap(~Cylinder,nrow = 2)
print(plot1.faceted)
```



4.6 Saving Plots

To save individual plots you can do the following

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

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

```
pdf("plot4.pdf", height=4, width=6)
plot4
dev.off()
```

```
## pdf
##   2
```

```
pdf("FullPlots.pdf", height=10, width=15)
grid.arrange(arrangeGrob(plot3, plot4, nrow=1),
             plot5,
             nrow=2)
dev.off()
```

```
## pdf
##   2
```

You don't have to save as .pdf, that's just want I always do because it's easy to use them for LaTeX figures. You could save as .bmp, .jpeg, .png, or .tiff using the same approach. We'll return to plotting in the next chapter when we discuss how to plot the substantive effects from regression models.