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

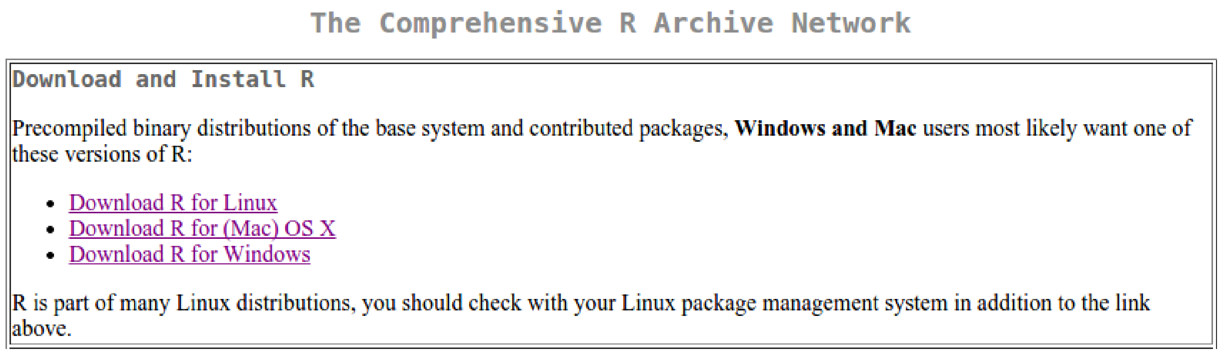
- Conduct basic statistical analysis with R (5)
- Create tables (5) and plots (4) that can be exported directly into LaTeX

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

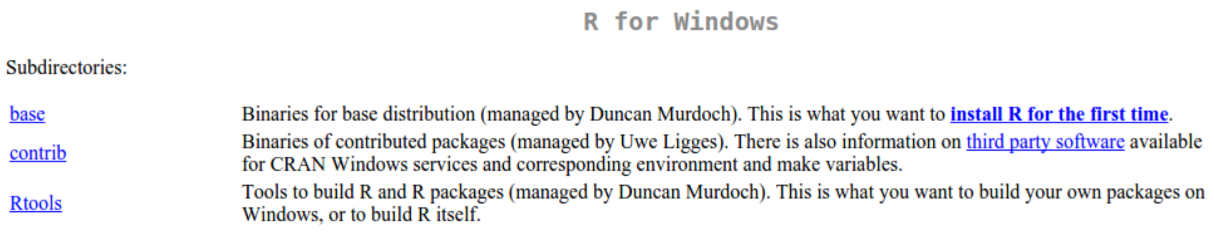
Download and Install R

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/`

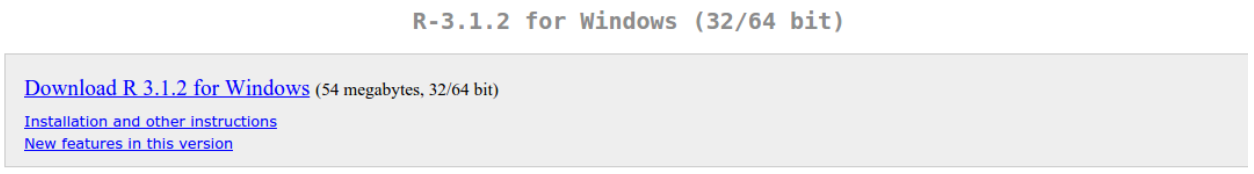

R for Windows

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

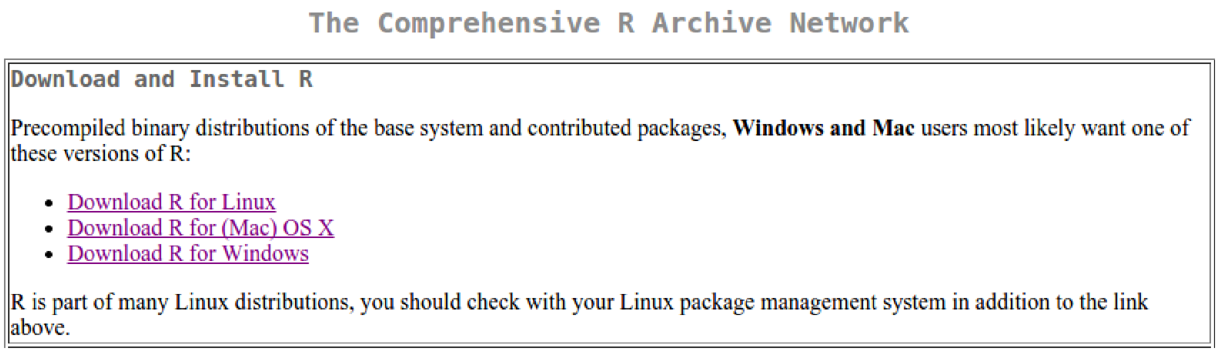
Figure 3: `http://cran.r-project.org/bin/windows/base`

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

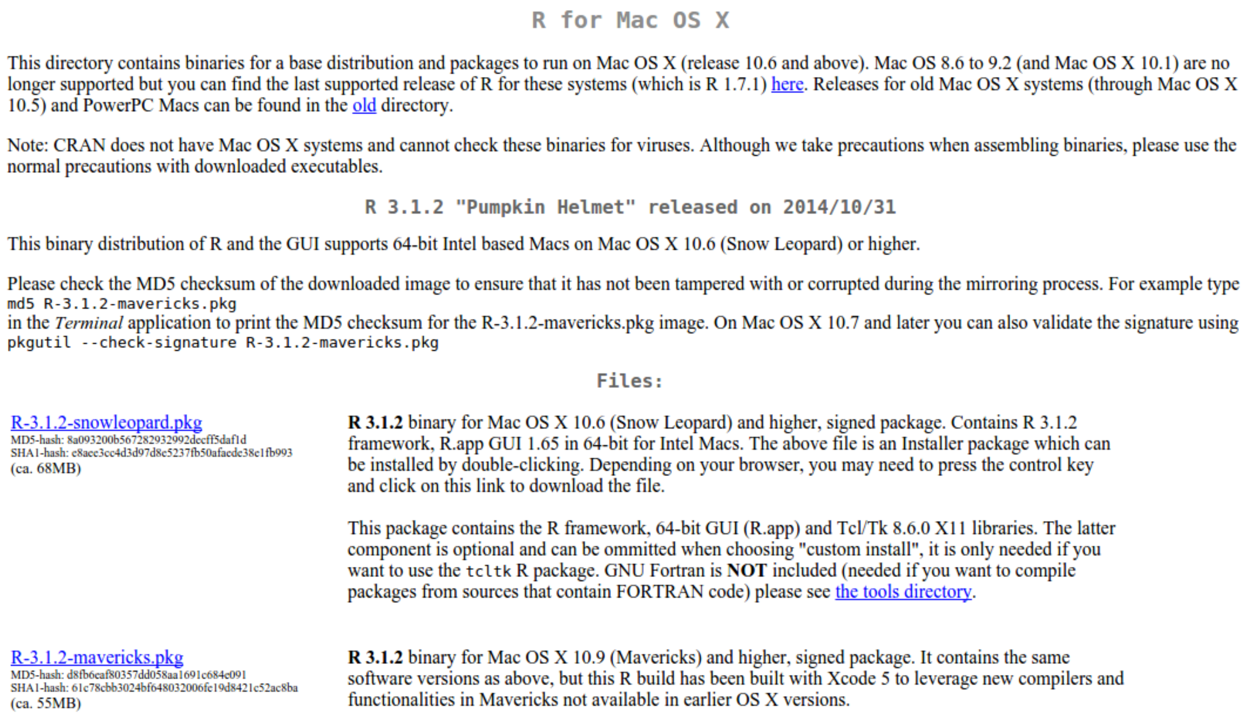
Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- 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:

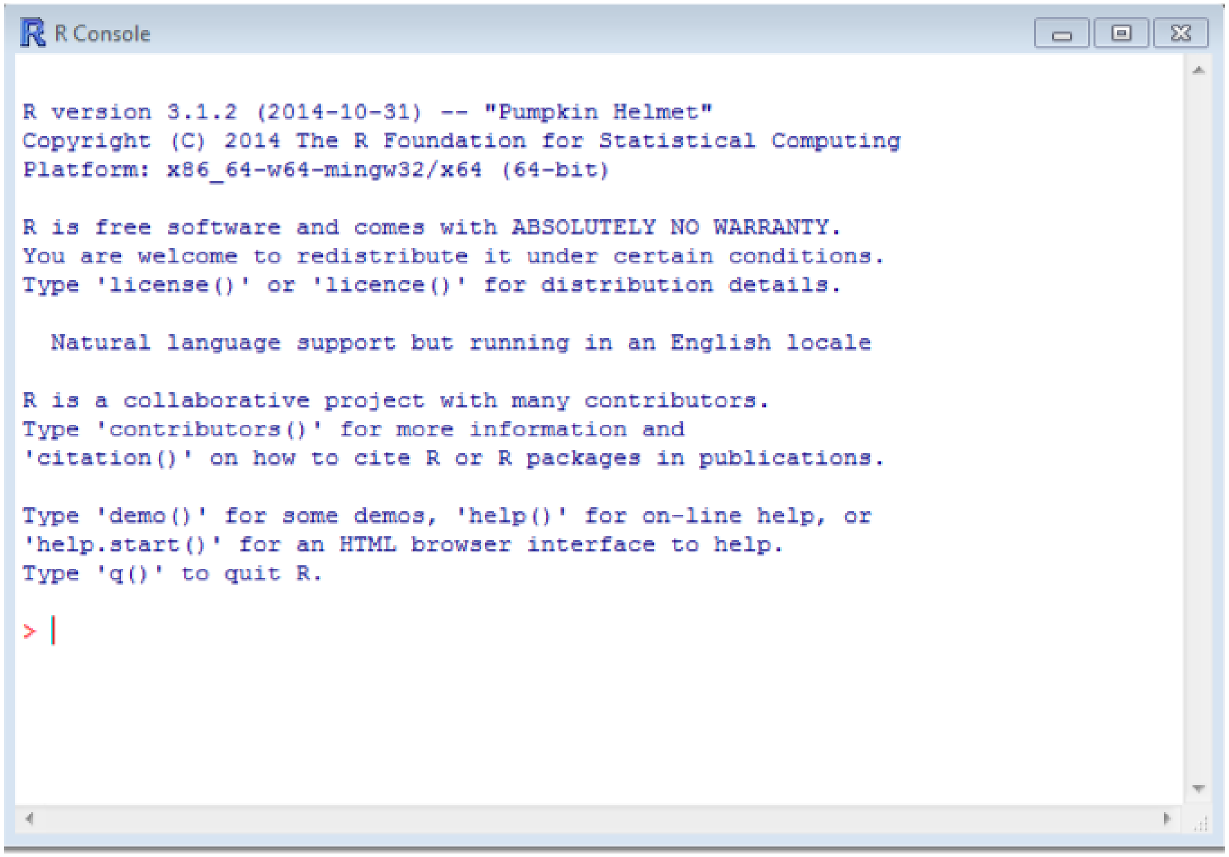
R-3.1.2-snowleopard.pkg MD5-hash: 8a093200b567282932992decff5daf1d SHA1-hash: e8aee3cc4d3d97d8e5237fb50afaede38e1fb993 (ca. 68MB)	**R 3.1.2** binary for Mac OS X 10.6 (Snow Leopard) and higher, signed package. Contains R 3.1.2 framework, R.app GUI 1.65 in 64-bit for Intel Macs. The above file is an Installer 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

```
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.4 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/liblapack.so.3
##
## 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
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.2.1  magrittr_2.0.3  fastmap_1.1.0   cli_3.3.0
##  [5] tools_4.2.1     htmltools_0.5.2 rstudioapi_0.13 yaml_2.3.5
##  [9] stringi_1.7.8   rmarkdown_2.14  knitr_1.39      stringr_1.4.0
## [13] xfun_0.30       digest_0.6.29   rlang_1.0.2     evaluate_0.15
```

Which shows us the version of R we're using, our operating system (actually 4.2.1, 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.*

R Studio 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 R Studio visit http://www.rstudio.com/products/rstudio/download/ and find the installer that matches your system. I strongly recommend the use of R Studio 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
```

```
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

```r
x
```

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

```r
x-2
```

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

```r
log(x)
```

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

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

```r
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.

```r
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 in number - of - states <- 50: 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

```r
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`.

```r
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
```

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

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

As before we can replace individual elements

```r
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

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

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

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

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

```r
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

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

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
```

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

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

As before we can use the logical operators

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

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

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

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

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

```
length(X)
```

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

```
mean(X)
```

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

```
sd(X)
```

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

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

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

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

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

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

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

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

But what if we wanted means by column? This takes us to our first introduction of the `for` loop and the `apply` function.

We will cover them in greater detail later but for now let's start with `for` loop.

```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.

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

```
## NULL
```

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

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

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

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

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

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

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

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

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

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

1.6.2 Matrix Operations

Matrix math in R includes standard operations including arithmetic.

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

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

```r
X-Y
```

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

Note that * performs *element-wise* multiplication. For standard matrix multiplication use %*%

```r
X*Y
```

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

```r
X %*% Y
```

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

If you use matrix multiplication on a vector R will guess whether it is a row or column vector. It typically does a good job of it, but be careful.

```r
c(1, 1) %*% X
```

```
##      [,1] [,2]
## [1,]    3    7
```

```r
X %*% c(1,1)
```

```
##      [,1]
## [1,]    4
## [2,]    6
```

We can transpose matrices (typically written as X' or X^T)

```
X
```

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

```
t(X)
```

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

Matrix multiplication

```
A = matrix(1:6, nrow=2)
B = matrix(1:6, nrow=3)
A %*% B  #Matrix multiplication
```

```
##      [,1] [,2]
## [1,]   22   49
## [2,]   28   64
```

Matrix inversion (typically written as X^{-1}) is done via the `solve` command.

```
solve(X)
```

```
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

```
solve(X) %*% X
```

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

Note that the `solve` command is done by numerical computation, not an analytic solution, so the results are only accurate up to something like the 16th decimal place. To illustrate we'll use `rnorm` to generate random numbers from the standard normal distribution. Note that we set a seed value here, that tells us which random numbers we want so we will get the same random numbers if we use the same seed. This allows for reproducible randomness.

```r
set.seed(1)
Z <- matrix(rnorm(16), nrow = 4)
solve(Z) %*% Z
```

```
##               [,1]         [,2]          [,3]          [,4]
## [1,]  1.000000e+00 0.000000e+00 -1.387779e-16 -6.591949e-17
## [2,] -5.551115e-17 1.000000e+00  2.914335e-16  2.463307e-16
## [3,]  1.665335e-16 0.000000e+00  1.000000e+00 -7.459311e-17
## [4,]  0.000000e+00 1.387779e-17 -1.387779e-17  1.000000e+00
```

Notice this is really close to an identity matrix, but not quite, we can use the `round` function to make this easier on the eyes.

```r
round(solve(Z) %*%Z, digits=12)
```

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

So close enough for almost anything we're interested in doing.

Additional functions that may come in handy include the determinant the Cholesky decomposition

```r
Y <- matrix(c(1, 0.5, 0.5, 1), nrow=2)
det(Y)
```

```
## [1] 0.75
```

```r
chol(Y)
```

```
##      [,1]      [,2]
## [1,]    1 0.5000000
## [2,]    0 0.8660254
```

```r
t(chol(Y)) %*% chol(Y)  #make sure it worked
```

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

There's no command for the trace, but it's easy to figure it out with what we know

```
sum(diag(Y)) #trace
```

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

We can also get eigenvalues and eigenvectors

```
eigen(Y)
```

```
## eigen() decomposition
## $values
## [1] 1.5 0.5
##
## $vectors
##           [,1]       [,2]
## [1,] 0.7071068 -0.7071068
## [2,] 0.7071068  0.7071068
```

Notice that the `eigen` output as two components designed with the `$` sign. The `$` means that we're dealing with a list which a new type of output, to which we now turn our attention

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. The case of eigenvalues and eigenvector this would like this.

```
names(eigen(Y))
```

```
## [1] "values"  "vectors"
```

```
eigen(Y)$values
```

```
## [1] 1.5 0.5
```

```
eigen(Y)$vectors
```

```
##           [,1]       [,2]
## [1,] 0.7071068 -0.7071068
## [2,] 0.7071068  0.7071068
```

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

```
eigen(Y)[[2]]
```

```
##           [,1]       [,2]
## [1,] 0.7071068 -0.7071068
## [2,] 0.7071068  0.7071068
```

```
class(eigen(Y)[[2]])
```

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

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. To create a list we just use the list command

```
matrixList <- list(matrix = diag(4), #Identity matrix
                   M2 = Y,
                   Eig = eigen(Y))
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
##
## $M2
##      [,1] [,2]
## [1,]  1.0  0.5
## [2,]  0.5  1.0
##
## $Eig
## eigen() decomposition
## $values
## [1] 1.5 0.5
##
## $vectors
##            [,1]       [,2]
## [1,] 0.7071068 -0.7071068
## [2,] 0.7071068  0.7071068
```

As you can see we can even nest lists within lists. If we wanted to extract the eigenvectors from this list we could use either the names or the square brackets.

```
matrixList$Eig$vectors
```

```
##            [,1]       [,2]
## [1,] 0.7071068 -0.7071068
## [2,] 0.7071068  0.7071068
```

```
matrixList[[3]][[2]] #Same thing
```

```
##            [,1]       [,2]
## [1,] 0.7071068 -0.7071068
## [2,] 0.7071068  0.7071068
```

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 class of each object in a list we could do the following.

```
lapply(matrixList, class)
```

```
## $matrix
## [1] "matrix" "array"
##
## $M2
## [1] "matrix" "array"
##
## $Eig
## [1] "eigen"
```

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

```
sapply(matrixList, class)
```

```
## $matrix
## [1] "matrix" "array"
##
## $M2
## [1] "matrix" "array"
##
## $Eig
## [1] "eigen"
```

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

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

```
## $matrix1
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## $matrix2
##      [,1] [,2] [,3]
## [1,]    0    2    4
## [2,]    1    3    5
##
## $matrix3
##             [,1] [,2]
## [1,] -0.01619026    1
## [2,]  0.94383621    1
## [3,]  0.82122120    1
```

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

```
##      matrix1  matrix2  matrix3
## [1,] "matrix" "matrix" "matrix"
## [2,] "array"  "array"  "array"
```

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

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

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

```
## $matrix1
## [1] 2 5 8
##
## $matrix2
## [1] 0.5 2.5 4.5
##
## $matrix3
## [1] 0.5829557 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' if you click this once every few months and select all you should be fine.

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,

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

Searches the help files for mentions of ridge regression. 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:** 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, and results from the R mailing list archives are also typically good.

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 $1,000 \times 3$ matrix, call it `X` where the first column is all 1s, the second column contains random draws from a normal with mean 1 and standard deviation 2 (hint: look at problem 1) and the last column contains random draws from the uniform distribution [0, 1] (use `??` or google to try and find the function for this). Use any of the methods discussed above to create the matrix. Look up and use

the function `colMeans` to print the column means for each column and use `apply` to print standard deviations of each column to make sure you that you did this correctly (the standard deviation for $U[0,1]$ will be between 0.27 and 0.30)

   b. Create a vector `b` equal to (-1, 2, 2). Then change the second value to -2.
   c. Use matrix multiplication to generate `y` such that $y = Xb + e$ where e is a vector of length 1,000 and is distributed normal (0, 1).

3. Download the following packages:
   - `readstata13`
   - `data.table`
   - `MASS`
   - `tidyr`
   - `dplyr`
   - `ggplot2`
   - `gridExtra`
   - `lmtest`
   - `car`
   - `sandwich`
   - `plm`
   - `stargazer`
   - `xtable`

2 Control Statements and Programming

This chapter really takes us into the meat of `R` programming. In particular we will cover the basics of `for` and `while` loops and if-else commands.

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.

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

```
## Goodbye
```

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

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

```
## [1] 0.7237109
```

```
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] 0.4112744 0.8209463 0.6470602 0.7829328 0.5530363 0.5297196 0.7893562
##  [8] 0.0233312 0.4772301 0.7323137
```

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

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

As with if-else constructs we can also nest them

```
print(test)
```

```
##  [1] 0.4112744 0.8209463 0.6470602 0.7829328 0.5530363 0.5297196 0.7893562
##  [8] 0.0233312 0.4772301 0.7323137
```

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

```
##  [1] "Liz"  "Bob"  "Bob"  "Bob"  "Bob"  "Bob"  "Bob"  "Mary" "Liz"  "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 can consider a couple different loops here. The most basic, which we briefly saw above, is the `for` loop.

```
y <- 1:10
for(i in 1:10){
  y[i] <- y[i]^2
}
y
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

We can combine it with with if statements

```
y <- 1:10
for(i in 1:10){
  if(y[i] %% 2){
    print("y is odd")
  }else{
    print("y is even")
  }
}
```

```
## [1] "y is odd"
## [1] "y is even"
## [1] "y is odd"
## [1] "y is even"
## [1] "y is odd"
## [1] "y is even"
## [1] "y is odd"
## [1] "y is even"
## [1] "y is odd"
## [1] "y is even"
```

Use the help functions from before to figure out what `%%` means and why we can use it to find odds and evens.

Let's say we didn't know how many times something needed done though, we just know when it's done. For that we can use 2 different structures. The first is the repeat structure:

```r
repeat{
  y <- runif(1)
  if(y< .05){
    break
  }
}
y
```

```
## [1] 0.01307758
```

We could do this OR we could do the much easier

```r
#Create initial value of y that satifies the condition
y <- 1
while(y>0.05){
  y <- runif(1)
}
y
```

```
## [1] 0.03554058
```

Typically we use `for` loops when we want to repeat an operation some set number of times and there is no breaking condition. On the other side of things, `while` loops are useful for situations where you want something to converge to within some tolerance (such as trying to maximize/minimize a function).

2.3 *ply Functions

The ply family of functions is a set of functions that are designed to make more readable. They typically are used in place of loops because they are less cumbersome to write (once you understand them). The first function we'll look at is `apply`, which is used on matrices

```r
X <- replicate(3, rnorm(10))
apply(X, 2, sd) #take the standard deviation of each column
```

```
## [1] 0.4638124 0.8784235 0.7808487
```

```r
apply(X, 1, max) #max of each row
```

```
##  [1]  0.3659411  1.2560188  0.6466744  1.2993123  1.2540831  0.7721422
##  [7] -0.1191688  0.6641357  1.1009691  0.9969869
```

```
apply(X, 1, function(x){ifelse(all(x>0), # can you explain this?
                                return(max(x)),
                                return(min(x)))})
```

```
##  [1] -0.2757780 -0.9120684 -1.4375862 -0.7970895 -0.8732621 -0.6490101
##  [7] -0.8808717 -0.4248103 -0.4189801 -0.2821739
```

If you're dealing with lists you may want to use `lapply`.

```
X <- list(A = diag(1:4),
          B = matrix(1:4, nrow=2))
lapply(X, solve)
```

```
## $A
##      [,1] [,2]      [,3] [,4]
## [1,]    1  0.0 0.0000000 0.00
## [2,]    0  0.5 0.0000000 0.00
## [3,]    0  0.0 0.3333333 0.00
## [4,]    0  0.0 0.0000000 0.25
##
## $B
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

```
lapply(X, t)
```

```
## $A
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    2    0    0
## [3,]    0    0    3    0
## [4,]    0    0    0    4
##
## $B
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

If you're dealing with lists, but you want to return a vector we have `sapply`.

```
X <- list(A = diag(1:4),
          B = matrix(1:4, nrow=2))
#Look at the difference between
sapply(X, max)
```

```
## A B
## 4 4
```

```
lapply(X, max)
```

```
## $A
## [1] 4
##
## $B
## [1] 4
```

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.

2.4 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. 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 list with summary stats
  ans <- list(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] -0.03044184
##
## $StDev
## [1] 1.01675
##
## $Min
## [1] -3.236386
##
## $Median
## [1] -0.0642373
##
## $Max
## [1] 3.266415
```

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] "A"           "B"           "dat"         "i"           "matrixList"
##  [6] "mean.x"      "summarize"   "test"        "x"           "X"
## [11] "x2"          "y"           "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] -0.007817983
```

```
##
## [[1]]$StDev
## [1] 1.031619
##
## [[1]]$Min
## [1] -3.045364
##
## [[1]]$Median
## [1] -0.00701718
##
## [[1]]$Max
## [1] 3.039033
##
##
## [[2]]
## [[2]]$Mean
## [1] 500.5
##
## [[2]]$StDev
## [1] 288.8194
##
## [[2]]$Min
## [1] 1
##
## [[2]]$Median
## [1] 500.5
##
## [[2]]$Max
## [1] 1000
```

Which we may think is too cumbersome of a result so we can collapse some of that by using the `unlist` command to collapse a list into a vector

```
lapply( apply(X, 2, summarize), unlist)
```

```
## [[1]]
##         Mean       StDev          Min       Median          Max
## -0.007817983  1.031618601 -3.045363930 -0.007017180  3.039033406
```

```
##
## [[2]]
##      Mean     StDev       Min    Median       Max
##  500.5000  288.8194    1.0000  500.5000 1000.0000
```

But we really don't want to write too many functions which is why we let other people do that and then use their packages.

2.5 APPLICATION: Solving a Nonlinear System of Equations

Consider the following battle of the sexes with Irving and Claire.

		C	
		M	B
I	M	2, 3	0, 0
	B	0, 0	3, 2

Further suppose each player has some action-specific private information (this induces nonlinearity and makes it a little more tricky than just a system of linear equations). We will denote this as an action specific shock for each player and action such that $\varepsilon_i(a_i)$ for $i \in \{I, C\}$. Let this information be iid normal with mean 0 and variance 1/2. We can think of this private information as being something like Claire discovers it's free hot dog day at the monster truck rally and so she may want to go more than is known to both players or the analyst. Games with private information will be covered more in Game Theory, for now just take it as a condition of the exercise. We want to find a mixed strategy equilibrium.

The conditional choice probabilities for this game are given by

$$\begin{aligned}
\Psi_I(a_I = B) &= \Pr\left[3\Pr(a_C = B) + 0(1 - \Pr(a_C = B)) + \varepsilon_I(a_I = B)\right.\\
&\qquad \left. > 2(1 - \Pr(a_C = B)) + 0(\Pr(a_C = B)) + \varepsilon_I(a_I = M)\right]\\
&= \Pr[5\Pr(a_C = B) - 2 > \varepsilon_I(a_I = M) - \varepsilon_I(a_I = B)]\\
&= \Phi(5\Pr(a_C = B) - 2)\\
\Psi_C(a_C = B) &= \Pr\left[2\Pr(a_I = B) + 0(1 - \Pr(a_I = B)) + \varepsilon_C(a_C = B)\right.\\
&\qquad \left. > 3(1 - \Pr(a_I = B)) + 0(\Pr(a_I = B)) + \varepsilon_C(a_C = M) + \varepsilon_M\right]\\
&= \Pr[5\Pr(a_I = B) - 3 > \varepsilon_C(a_C = M) - \varepsilon_C(a_C = B)]\\
&= \Phi(5\Pr(a_I = B) - 3)
\end{aligned}$$

Where Φ is the standard normal PDF. Let's combine those into a single Ψ

$$\Psi(p) = \begin{bmatrix} \Psi_I(a_I = B) \\ \Psi_C(a_C = B) \end{bmatrix}$$

An equilibrium can be described as a vector, $p = (\Pr(a_I = B), \Pr(a_C = B))$, such that

$$\Psi(p) - p = 0.$$

This is a nonlinear system of equations which we will solve using an iterative procedure, but before we get to the procedure let's lay the ground work and write down our Ψ function.

```
Psi <- function(p){
  pI <- pnorm(5*p[2]-2)
  pC <- pnorm(5*p[1]-3)
  return(c(pI,pC) -p)
}
```

Newton's method for nonlinear equations requires that we know the Jacobian (first derivatives) of this function $\Psi(p) - p$.

$$\begin{aligned}
J_p\Psi(p) - p = J_p &\begin{bmatrix} \Phi(5\Pr(a_C = B) - 2) - \Pr(a_I = B) \\ \Phi(5\Pr(a_I = B) - 3) - \Pr(a_C = B) \end{bmatrix}\\
&= \begin{bmatrix} -1 & 5\phi(5\Pr(a_C = B) - 2) \\ 5\phi(5\Pr(a_I = B) - 3) & -1 \end{bmatrix}
\end{aligned}$$

```r
jac <- function(p){
  DpI <- c(-1, 5*dnorm(5*p[1]-3))
  DpC <- c(5*dnorm(5*p[1]-2), -1)
  return(cbind(DpI, DpC))
}
```

Armed with these tools we can now solve the problem. Newton's method for solving non-linear equations starts with an initial guess at the solution, call this x_0, and does the following for iteration $k = 1, 2, \ldots$

$$x_k = x_{k-1} - \Psi(x_{k-1})\left(D_x\Psi(x_{k-1})\right)^{-1}.$$

This procedure is iterated until $\max(|x_k - x_{k-1}|) < \varepsilon$ for some pre-specified tolerance.

```r
Newton <- function(func, jac, x0, tol=1e-5){
  xold <- x0
  diff <- 1
  while(diff > tol){
    xnew <- xold - func(xold)%*% solve(jac(xold))
    diff <- max(abs(xnew-xold))
    xold <- xnew
  }
  return(xnew)
}
x0 <- c(.5,.5)

p.eq <- Newton(func=Psi, jac=jac, x0=x0)
p.eq
```

```
##           [,1]      [,2]
## [1,] 0.5665073 0.4334978
```

Note that there are actually three equilibira to this game, but Newton will only ever find 1 (usually one that's near the starting values). In this equilibrium, Irving goes to the ballgame with probability 0.57 and to the monster truck rally with probability 0.43. For Claire these numbers are switched. Play with the starting values and see if you can find another equilibrium.

2.6 APPLICATION: Least Squares by Maximum Likelihood

As you may or may not have learned by now, OLS is also the maximum likelihood estimator $\hat{\beta}$ when y is distributed normally with mean $X\beta$. This means that solving OLS by either maximum likelihood or by minimizing the sum of squared error should give us the estimates of β. To satisfy us that this is the case we will use R to maximize the logged likelihood function and compare it to the traditional OLS estimates.

First, let's generate some data

```
set.seed(1)
N <- 2000
X <- cbind(1, replicate(2,rnorm(N)))
beta <- c(-1, 2, -2)
sigma2 <- 1
y <- X %*% beta + rnorm(N, 0, sqrt(sigma2))
```

Since y is distributed i.i.d. normal the joint pdf of the sample is

$$f(y|X, \beta, \sigma^2) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(y_i - X_i\beta)^2}{2\sigma^2}\right)$$

As you should remember from math camp, this joint pdf is proportional to the likelihood function so we will just switch the order of the conditionals. To make things easy we often take the log of it. The likelihood function is thus:

$$L(\beta, \sigma^2|X, y) = \sum_{i=1}^{N} -\frac{1}{2}\log(\sigma^2) - \frac{1}{2}\log(2\pi) - \frac{1}{2\sigma^2}(y_i - X_i\beta)^2.$$

As we know the 2π term is constant in β and σ^2 so we can drop it from our likelihood routine. It is advisable that we reparameterize this function so that it is a function of β and $\theta = \log(\sigma^2)$. The reason why we reparameterize the model was so that we can let R take guesses at $\theta \in \mathbb{R}$ rather than $\sigma^2 \in \mathbb{R}_+$. Numerical optimizers are much easier to work with if you can find a way to let them take guesses that are not constrained. Our new likelihood is now

$$L^*(\beta, \theta|X, y) = \sum_{i=1}^{N} -\frac{1}{2}\theta - \frac{1}{2\exp(\theta)}(y_i - X_i\beta)^2. \tag{1}$$

Now when our optimizer takes guesses at θ it can guess any real number and our estimate for σ^2 is $\widehat{\sigma^2} = \exp(\hat{\theta})$. Let $\theta = (\beta, \theta)$ we can now create the likelihood function.

```r
NormalMLE <- function(theta, X, y){
  eta <- theta[length(theta)] #extract eta from parameter vector
  beta <- theta[-length(theta)] #beta coefficients

  Lik <- - 1/2 * eta - 1/(2*exp(eta)) * (y - X%*%beta)^2 #L* from above
  Lik <- -sum(Lik)
  return(Lik)
}
```

Note that we waited until the end to sum up the observations. Also note that we took the negative of the sum. This is because most (but not all) numerical optimizers look for a minimum rather than a maximum, minimizing the negative likelihood is the same as maximizing the likelihood.

It is advisable in cases of numerical optimization that we also include first derivative information on the parameters. The gradient takes the form

$$\begin{aligned}\frac{\partial L}{\partial \beta} &= \sum_{i=1}^{N} \frac{X_i(y_i - X_i\beta)}{\exp(\theta)} \\ \frac{\partial L}{\partial \theta} &= \sum_{i=1}^{N} \left(\frac{(y_i - X_i\beta)^2}{2\exp(\theta)} - \frac{1}{2} \right)\end{aligned}$$

and returns a vector of length equal to to the length of θ. The actual programming will be left as an exercise for the reader but here's what the output should look like

```r
#Should look something like this, but your results may
#differ based on the seed value.
grNormalMLE(rep(0,4), X=X, y=y)
```

```
## [1]  2151.647 -4178.802  4342.570 -9735.758
```

We don't really need the gradient to optimize the function it's just useful for improving accuracy, speed, and reliability. However, in a lot of problems it's unnecessary. To actually optimize the function we will use the `optim` function.

```r
##optim is a nonlinear optimizer that takes the following inputs
#par = starting values, in our case draws from the uniform.
#       These correspond to theta above
#fn = function to optimize
#gr = gradient (first derivatives)
```

```
#method = Method to use for optimization BFGS is a quasi-Newton method
#          that works really well on most problems
#X, y are the extra arguements that we included in the  NormalMLE
#     and grNormalMLE functions.
optim(par=runif(4), fn=NormalMLE, gr=grNormalMLE, method="BFGS", X=X, y=y)
```

```
## $par
## [1] -1.01468245  1.99629945 -2.07802388 -0.03969555
##
## $value
## [1] 960.3087
##
## $counts
## function gradient
##       48       12
##
## $convergence
## [1] 0
##
## $message
## NULL
```

This is nice what if we wanted to standard errors though. `optim` doesn't have an option for that directly but it can return the Hessian. Do you remember how to get standard errors from the Hessian?

```
mod1 <- optim(par=runif(4), fn=NormalMLE, method="BFGS", X=X, y=y,
              gr=grNormalMLE, hessian=TRUE)

vcov1 <- solve(mod1$hessian)
sqrt(diag(vcov1))
```

```
## [1] 0.02192632 0.02115178 0.02120424 0.03162315
```

Comparing these results to OLS estimates is left as an exercise to the reader.

2.7 APPLICATION: Monte Carlo on Omitted Variable Bias

This application will use a Monte Carlo experiment to explore the effect of omitted variable bias in a linear model. A Monte Carlo experiment is a simulation experiment wherein we set the true values of data to see how models perform in particular circumstances. In this application we will be seeing how the linear model performs in cases where a relevant explanatory variable is not included. For each iteration of the Monte Carlo we will do the following:

1. Generate data using the following data generating process: Draw X_1 and X_2 from the multivariate normal with mean 0, correlation ρ, and $\sigma_1^2 = \sigma_2^2 = 1$. Use $\rho = (-0.5, 0, 0.5)$.

$$y = 1 - 2(X_1) + 2(X_2) + \varepsilon$$

   Where $\varepsilon \sim N(0, 1)$. Create 2,000 observations in each sample.
2. Estimate $\hat{\beta}$ using OLS of only y on X_1.
3. Calculate the bias by subtracting the true values of β, $(1, -2)$ from the estimated values that you get from `lm`
4. Store this bias
5. Repeat 1,000 times
6. Create a list of length 3 (one for each value of ρ). Within that list create a 2×3 matrix where row 1 is the mean and 95% Confidence Interval of the bias the of $\hat{\beta}_0$, and row 2 is the same for the bias of $\hat{\beta}_1$.

So what does this look like?

```r
library(MASS) #for the multivariate normal
N <- 2000 #Sample size
rho <- c(-0.5, 0, 0.5) #Values of rho
beta <- c(1, -2, 2) #True betas
MCresults <- list() #empty list

for(r in 1:3){ #loop over values of rho
  results <- matrix(0, nrow=1000, ncol=2)
  for(i in 1:1000){
    Sigma <- matrix(c(1, rho[r], rho[r],1), nrow=2)
    X <- mvrnorm(N, c(0,0), Sigma)
    y <- cbind(1, X) %*% beta + rnorm(N)
    X1 <- cbind(1, X[,1])
    bhat <- solve(t(X1)%*%X1)%*%t(X1)%*%y
    bias <- beta[-3] - bhat #what's the -3 do?
    results[i,] <- bias
  }
  biasOut <- cbind(colMeans(results),
                   t(apply(results, 2, quantile,  #explain this?
                           c(0.025, 0.97)))) #and this?
  MCresults[[r]] <- biasOut
}
names(MCresults) <- paste("rho:", rho)
MCresults <- lapply(MCresults,
                    function(x){
                      rownames(x) <- c("bias in hat(beta)[0]", "bias in hat(beta)[1]")
                      return(x)
                      }
                    )
MCresults
```

```
## $`rho: -0.5`
##                                            2.5%        97%
## bias in hat(beta)[0] 0.0003684895 -0.0865589 0.08026098
## bias in hat(beta)[1] 0.9997653586  0.9066455 1.08259700
##
## $`rho: 0`
##                                             2.5%        97%
## bias in hat(beta)[0] -0.002249316 -0.09458069 0.09493118
## bias in hat(beta)[1] -0.001499449 -0.10094983 0.09784147
##
## $`rho: 0.5`
##                                             2.5%         97%
## bias in hat(beta)[0]  0.002419757 -0.08562747  0.08762999
## bias in hat(beta)[1] -1.000319606 -1.09064125 -0.91485670
```

What we can see from this particular Monte Carlo is that the constant term remains roughly unbiased, but there can be noticeable bias on the coefficient on X_1. More importantly the size and direction of the bias varies depending on how the omitted variable is related to the variable included. You'll cover this problem in more detail in 602

2.8 Exercises

1. Create a function that takes a vector of numbers and returns the maximum.

2. In this exercise you will use Gibbs sampling to estimate a Bayesian linear regression model. First, we will continue to assume that

$$y|X, \beta, \sigma^2 \sim N(X\beta, \sigma^2 I).$$

Since this is Bayesian we need to assume a prior distribution on β and σ^2. The standard priors are from a diffuse uniform.

As you may recall we need to identify the conditionals of our parameters. I'll spare you the details, but the distributions we want are (as you might guess)

$$\begin{aligned}\beta|\sigma^2, y, X &\sim N\left((X'X)^{-1}X'y, \sigma^2(X'X)^{-1}\right)\\ \sigma^2|\beta, y, X &\sim \text{Inv-}\chi^2\left(N-k, \frac{1}{N-k}(\hat{e}'\hat{e})\right),\end{aligned}$$

where $\hat{e} = y - X[(X'X)^{-1}X'y]$.

We will do this in steps:

a. Generate data where $y = X\beta + \varepsilon$, where $\varepsilon \sim N(0, \sigma^2)$. Set $\beta = (-1, 2, -2)$ and $\sigma^2 = 4$. Have X be a matrix of a constant term and two random normal variables. Set $N = 2,000$.
b. Initialize a matrix of dimension $10,000 \times 4$, fill it with 0s.
c. Draw initial values for $\hat{\beta}$ from a uniform from $-100,000$ to $100,000$
d. Construct a for loop that runs for 10,000 iterations. Each iteration i should
   - Take the last draw of β and set it as the current value of β.
   - Draw σ_i^2 from the inverse χ^2 (you'll need `geoR::rinvchisq`).
   - Draw β_i from the multivariate normal (you'll need `MASS::mvrnorm`).
   - Store the vector (β_i, σ_i^2) in the matrix you previously initialized as row i.
e. When this is done running, take the resulting matrix and discard the first 5,000 rows.
f. Take the mean of each column, it should be about equal to the true value of (β, σ^2).

3. Recall that the OLS estimator is

$$\hat{\beta} = (X'X)^{-1}X'y$$

with variance estimator

$$s^2(X'X)^{-1}$$

where $s^2 = \dfrac{\hat{e}'\hat{e}}{(N-k)}$, and $\hat{e} = y - X\hat{\beta}$. Now do the following

a. Your task is to create a function that takes 2 inputs `X` and `y` (`function(X, y)`) as inputs and returns a list containing the OLS estimates, the variance matrix and the standard errors (Recall the standard errors are equal to the square root of the diagonal of the variance matrix). Note, you may have to use `as.vector` on `s` to avoid an error.
b. test this function on the data you generated in exercise 2.
c. Look up the function `pt` and edit your function from the last part to conduct a t test to see if each coefficient is statistically significant from 0.

$$t = \frac{(\hat{\beta} - 0)}{S.E.}$$

(Hint: In order to get the right p value use the absolute value of the t statistic, the upper tail of the t distribution, and multiply your final answer by 2)

4. Code the gradient for the normal MLE regression problem above.

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
```

```
#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
```

```
##
```

```
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
```

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

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

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

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

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

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

```
## 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

```
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
```

```
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

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

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

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

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

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

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 -2.5354081
## 2     2  0.9869172
## 3     3  0.5297845
## 4     4 -0.5695382
## 5     5  0.3474034
```

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

```
##   ccode       Var2
## 1     1 0.48546473
## 2     2 0.30397046
## 3     3 0.93599756
## 4     4 0.07793285
## 5     5 0.96221627
```

```
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 -2.5354081 0.48546473
## 2     2  0.9869172 0.30397046
## 3     3  0.5297845 0.93599756
## 4      4 -0.5695382 0.07793285
## 5      5  0.3474034 0.96221627
```

A slightly more complicated example might be

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


temp.df
```

```
##   cow.code       Var1
## 1        1 -2.1791863
## 2        2 -0.9578550
## 3        3  1.2485817
## 4        4  0.2200321
## 5        5  0.6140073
```

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

```
##   ccode       Var2
## 1     1 0.60791216
## 2     2 0.37712201
## 3     3 0.08888156
## 4     4 0.91867842
## 5     5 0.17887154
```

```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 -2.1791863 0.60791216
## 2        2 -0.9578550 0.37712201
## 3         3  1.2485817 0.08888156
## 4         4  0.2200321 0.91867842
## 5         5  0.6140073 0.17887154
```

An even more complex example

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


temp.df
```

```
##    cow.code       Var1
## 1         1 -1.1131257
## 2         2  0.3048678
## 3         3  0.5627107
## 4         4  0.3030037
## 5         5  1.0670407
## 6         6 -0.1440281
## 7         7  0.8358535
## 8         8 -0.3234967
## 9         9 -0.9183713
## 10       10  0.6525688
```

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

```
##    ccode       Var2
## 1      1 0.05925756
## 2      2 0.95287853
## 3      3 0.93846697
## 4      4 0.16446574
## 5      5 0.32256539
## 6     11 0.05925756
## 7     12 0.95287853
## 8     13 0.93846697
## 9     14 0.16446574
## 10    15 0.32256539
```

```r
##We want to merge of country codes, but they have different countries
temp.df3 <- merge(temp.df,
                  temp.df2,
                  by.x='cow.code',
                  by.y='ccode')

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

```
##   cow.code       Var1       Var2
## 1        1 -1.1131257 0.05925756
## 2        2  0.3048678 0.95287853
## 3        3  0.5627107 0.93846697
## 4        4  0.3030037 0.16446574
## 5        5  1.0670407 0.32256539
```

```r
##All the countries from just the first data.frame
merge(temp.df,
      temp.df2,
      by.x='cow.code',
      by.y='ccode',
      all.x=TRUE)
```

```
##    cow.code       Var1       Var2
## 1         1 -1.1131257 0.05925756
## 2         2  0.3048678 0.95287853
## 3         3  0.5627107 0.93846697
## 4         4  0.3030037 0.16446574
## 5         5  1.0670407 0.32256539
## 6         6 -0.1440281         NA
## 7         7  0.8358535         NA
## 8         8 -0.3234967         NA
## 9         9 -0.9183713         NA
## 10       10  0.6525688         NA
```

```r
##Same for the 2nd
merge(temp.df,
      temp.df2,
      by.x='cow.code',
      by.y='ccode',
      all.y=TRUE)
```

```
##    cow.code       Var1       Var2
## 1          1 -1.1131257 0.05925756
## 2          2  0.3048678 0.95287853
## 3          3  0.5627107 0.93846697
## 4          4  0.3030037 0.16446574
## 5          5  1.0670407 0.32256539
## 6         11         NA 0.05925756
## 7         12         NA 0.95287853
## 8         13         NA 0.93846697
## 9         14         NA 0.16446574
## 10        15         NA 0.32256539
```

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

```
##    cow.code       Var1       Var2
## 1          1 -1.1131257 0.05925756
## 2          2  0.3048678 0.95287853
## 3          3  0.5627107 0.93846697
## 4          4  0.3030037 0.16446574
## 5          5  1.0670407 0.32256539
## 6          6 -0.1440281         NA
## 7          7  0.8358535         NA
## 8          8 -0.3234967         NA
## 9          9 -0.9183713         NA
## 10        10  0.6525688         NA
## 11        11         NA 0.05925756
## 12        12         NA 0.95287853
## 13        13         NA 0.93846697
## 14        14         NA 0.16446574
## 15       15          NA 0.32256539
```

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

```r
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

```r
##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"
```

```r
# 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)
```

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

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

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

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

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

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

```r
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
```

```
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
```

```
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).

3.5.3 APPLICATION: Generating a Lagged DV

Another common task is creating lagged variables. We'll now take a look at a couple of ways to create a lagged variable. There's not a good way base R way that I know of to do this. We will wrap things within the `system.time({...})` command to show us how slow/fast these approaches are.

```
# Tidy way
system.time({
FL2003 <- FL2003 %>%
  arrange(ccode, year) %>% #tell it this is a panel of country-years
  group_by(ccode) %>%  #work within countries
  mutate(laggedOnset = lag(onset))%>%#lag
  ungroup() #weird things can happen otherwise
summary(FL2003$laggedOnset)
})
```

```
##    user  system elapsed
##   0.021   0.001   0.022
```

```
# Did it work?
FL2003 %>%
  filter(ccode==200 & year > 1965) %>%
   select(cname, year, onset, laggedOnset) %>%
  head()
```

```
## # A tibble: 6 x 4
##   cname  year onset laggedOnset
##   <chr> <dbl> <dbl>       <dbl>
## 1 UK     1966     0           0
## 2 UK     1967     0           0
## 3 UK     1968     0           0
## 4 UK     1969     1           0
## 5 UK     1970     0           1
## 6 UK     1971     0           0
```

```
# Preferred way 2: Data Table (my most preferred)
system.time({
  FL2003 <- data.table(FL2003) #change type
  setkey(FL2003, ccode, year) #arange the data
  FL2003[,laggedOnset2:=shift(onset, n=1, type="lag"), by=ccode]
})
```

```
##    user  system elapsed
##   0.013   0.000   0.008
```

```
with(FL2003, table(laggedOnset, laggedOnset2))
```

```
##            laggedOnset2
## laggedOnset    0    1
##           0 6334    0
##           1    0  109
```

In the tidy case, we start with a data frame `FL2003` then we arrange it by `ccode` and `year`, then we group by `ccode`, then we mutate the onset variable using the `lag` function to generate a new variable with our groups. This can be a little weird to look at at first, but lots of people find it intuitive with just a little practice.

In the second (and faster) approach, we use a data table. This requires us to convert the data frame to a data table and then set the "keys" to arrange the data. The syntax of the final line is beyond our scope today, but if you want to know more, I can help.

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.

```
###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
##   0.037   0.000   0.237
```

```
output
```

```
## # A tibble: 25 x 3
##    ccode  year  SumVar
##    <int> <int>   <dbl>
##  1     1     1 -0.0425
##  2     1     2  0.383
##  3     1     3  0.793
##  4     1     4  4.20
##  5     1     5 -0.505
##  6     2     1 -0.618
##  7     2     2 -1.82
##  8     2     3  0.256
##  9     2     4 -1.10
## 10     2     5 -1.91
## # ... with 15 more rows
```

```
 #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
##   0.002   0.000   0.002
```

```
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.

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

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

```
ls() #Everything
```

```
##  [1] "A"             "B"             "beta"          "bhat"          "bias"
##  [6] "biasOut"     "cDummies"    "dat"         "FL.dt"       "FL2003"
## [11] "FullData"    "grNormalMLE" "i"           "jac"         "matrixList"
## [16] "MCresults"   "mean.x"      "mergedData"  "mod1"        "N"
## [21] "newDat"      "newDt"       "Newton"      "NMC"         "NormalMLE"
## [26] "out.dt"      "output"      "p.eq"        "pressData"   "Psi"
## [31] "r"           "results"     "rho"         "Sigma"       "sigma2"
## [36] "summarize"   "temp.dat"    "temp.df"     "temp.df2"    "temp.df3"
## [41] "test"        "vcov1"       "x"           "X"           "x0"
## [46] "X1"          "x2"          "y"           "Y"           "z"
## [51] "Z"
```

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

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

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

```
##  [1] "A"           "B"           "beta"        "bhat"        "bias"
##  [6] "biasOut"     "cDummies"    "dat"         "FL.dt"       "FL2003"
## [11] "FullData"    "grNormalMLE" "i"           "jac"         "matrixList"
## [16] "MCresults"   "mean.x"      "mergedData"  "mod1"        "N"
## [21] "newDat"      "newDt"       "Newton"      "NMC"         "NormalMLE"
## [26] "out.dt"      "output"      "p.eq"        "pressData"   "Psi"
## [31] "r"           "results"     "rho"         "Sigma"       "sigma2"
## [36] "summarize"   "temp.dat"    "temp.df"     "temp.df2"    "temp.df3"
## [41] "test"        "vcov1"       "x"           "X"           "x0"
## [46] "X1"          "x2"          "y"           "Y"           "z"
## [51] "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.

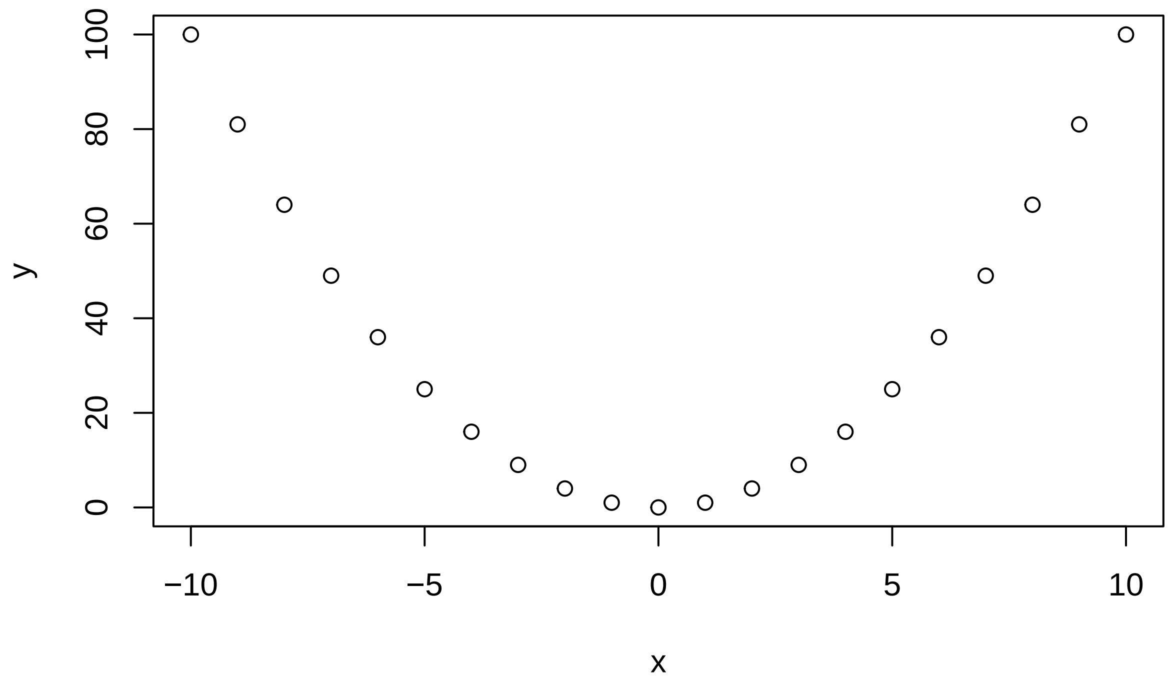
4 Plotting

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

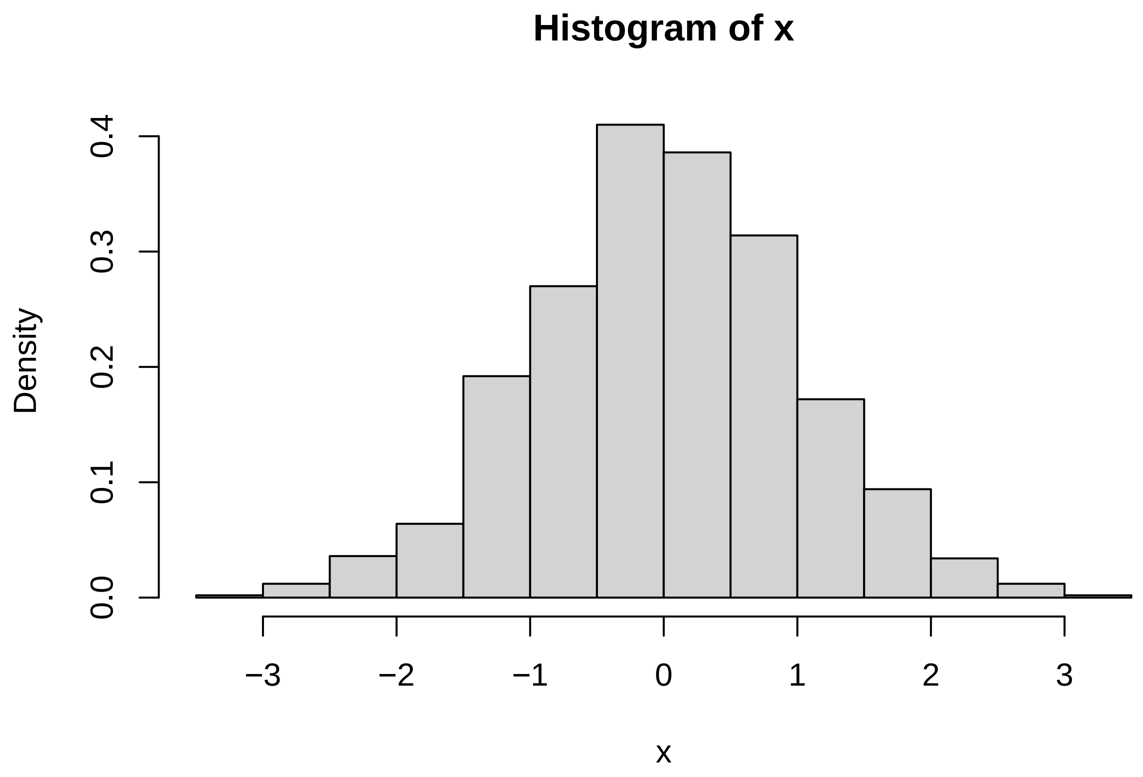
Despite the good things about ggplot sometimes is nice to do some some basic, exploratory plots with base graphics.

```r
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.

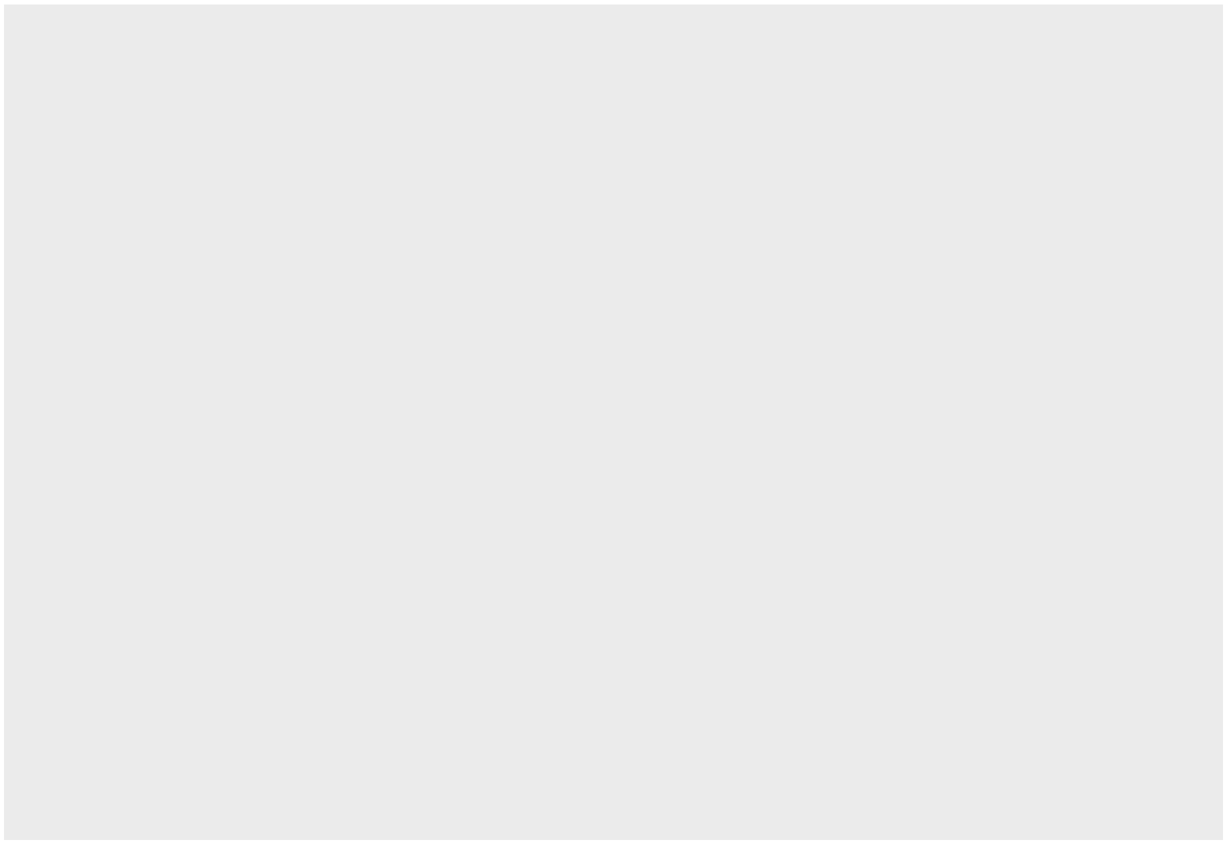
```
library(ggplot2)

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

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

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

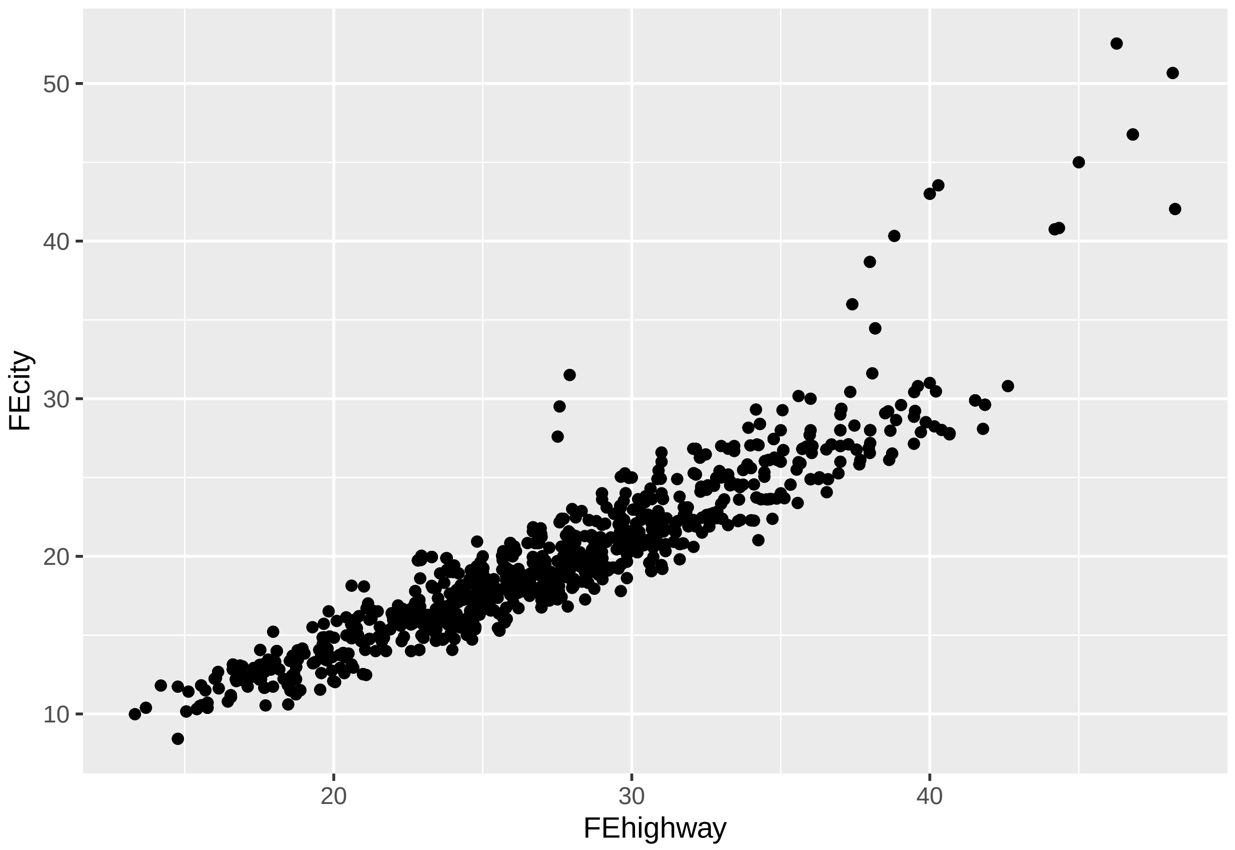
Note *`ggplot` works best with data frame and data table objects.*

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



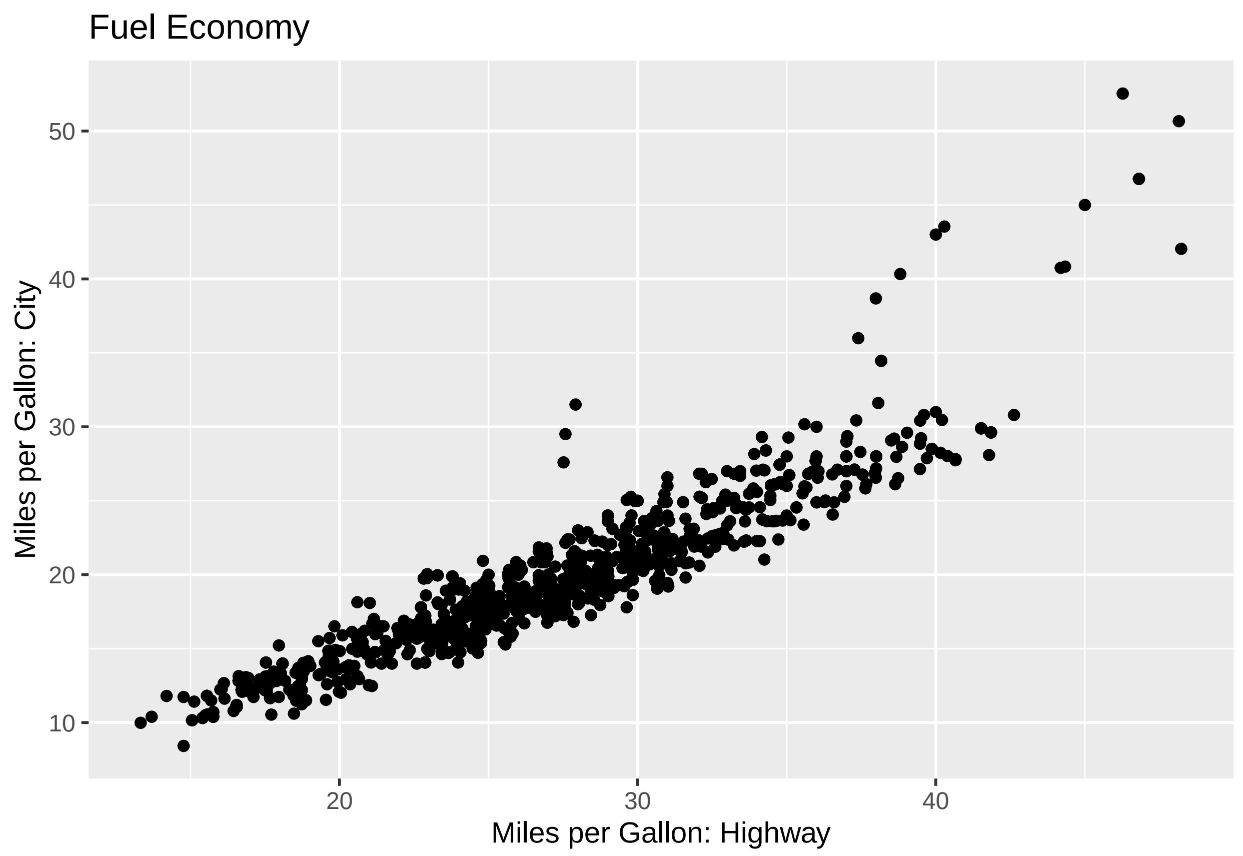
To create the scatterplot we need to add that layer to the plot

```
plot1 <- ggplot(FE2013) +  ##Initial layer
            geom_point(aes(x = FEhighway, y=FEcity))

## geom_point is used to specify that we want a scatter plot
## aes is used to specify the variables used in the plot

print(plot1)
```



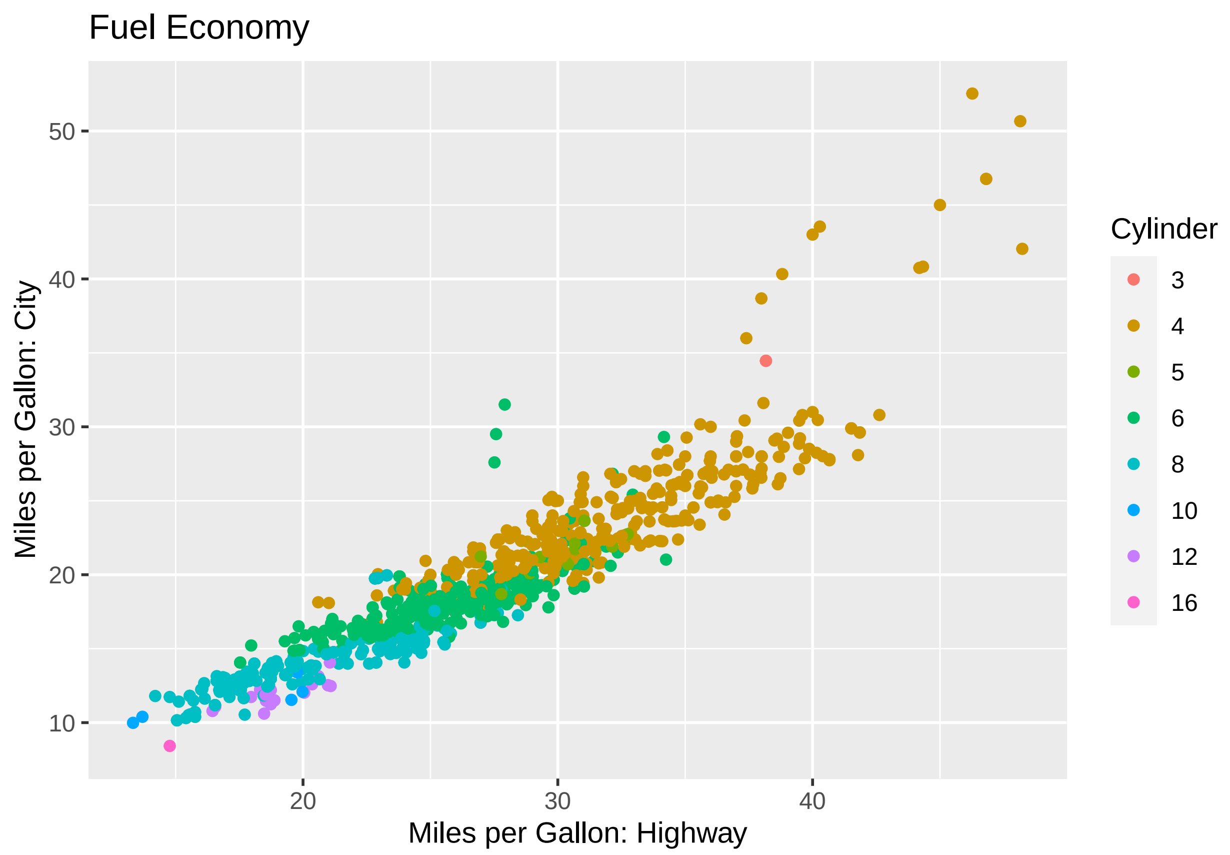
It's pretty straight forward to make changes to plot once it's created. Say we wanted to add a title and change the axis labels.

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

print(plot1)
```



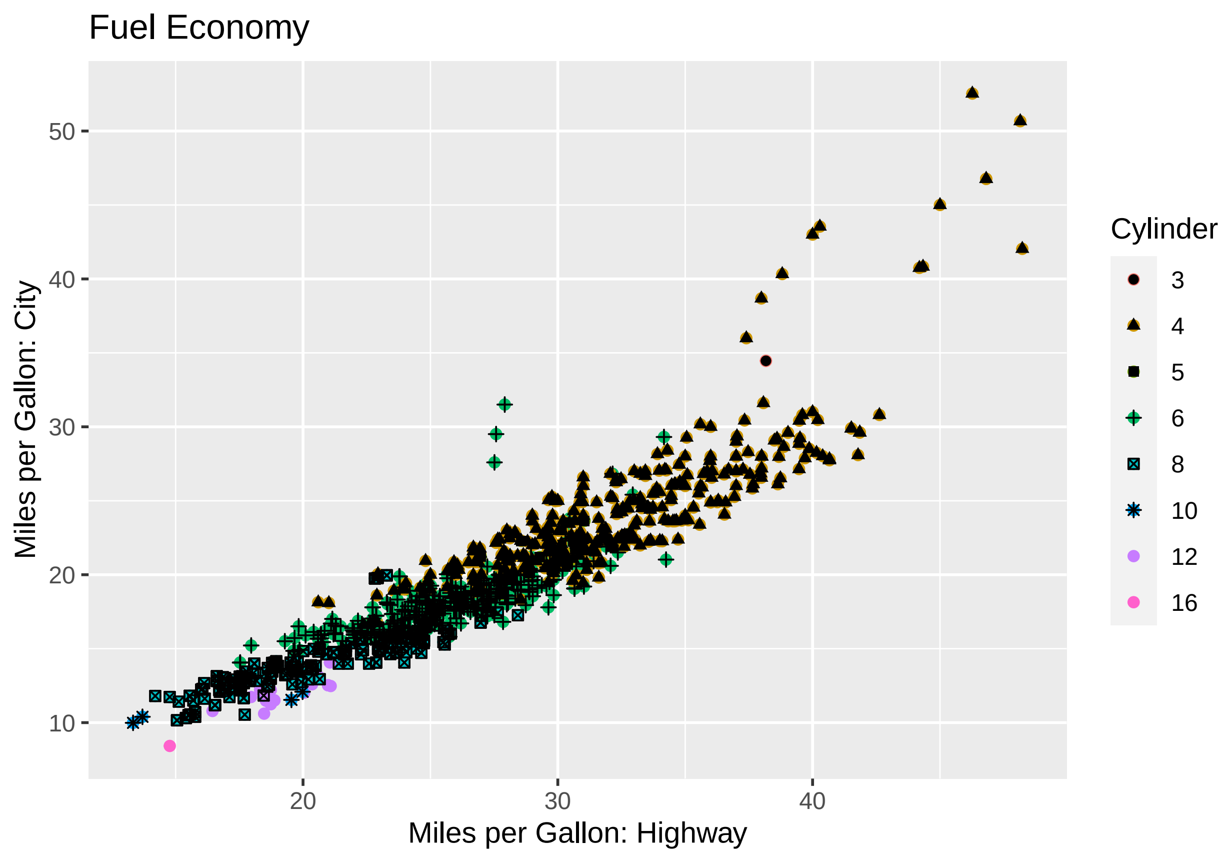
We can add color to the plot by including a factor variable. In this case, let's color the observations by number of cylinders.

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

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



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

print(plot1)
```

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

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

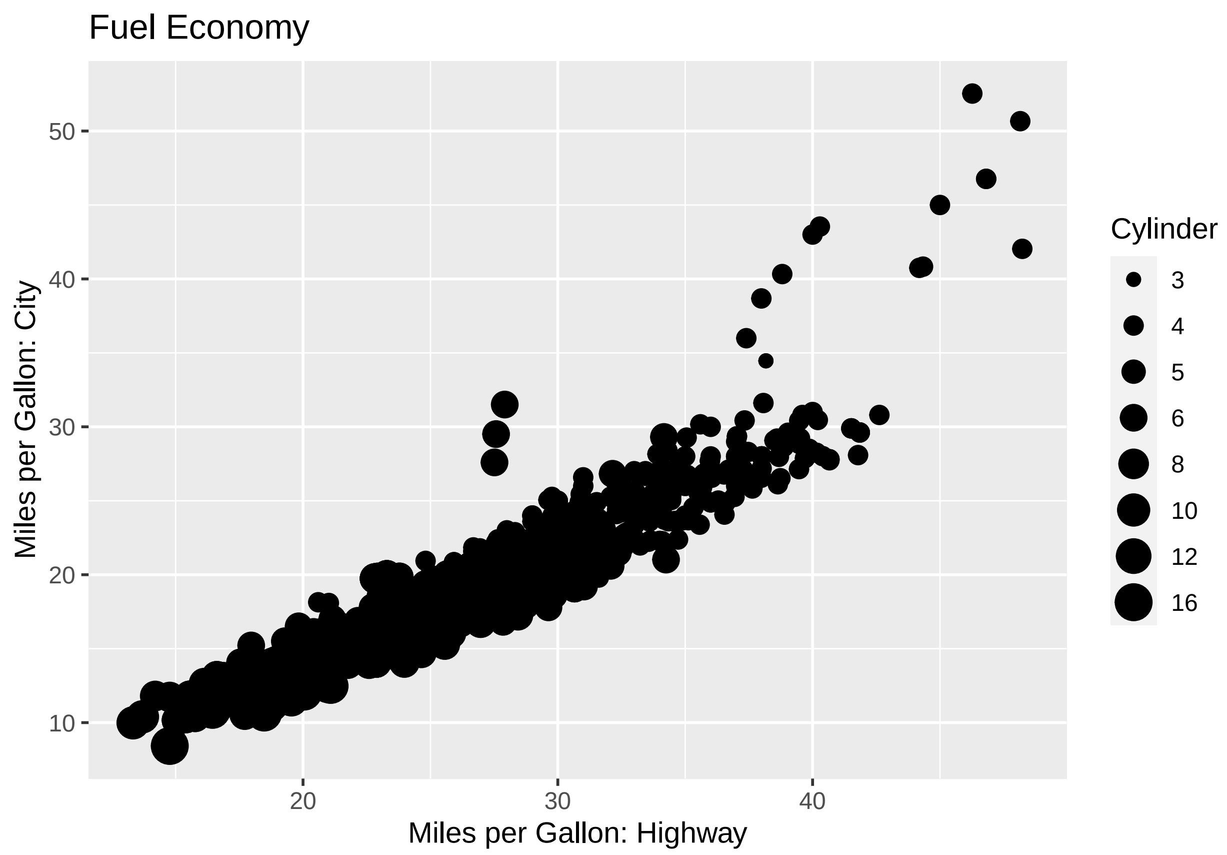


```r
##Or sizes
plot1 <- plot1 +
          geom_point(aes(x=FEhighway, y  = FEcity, size= Cylinder))
print(plot1)
```

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

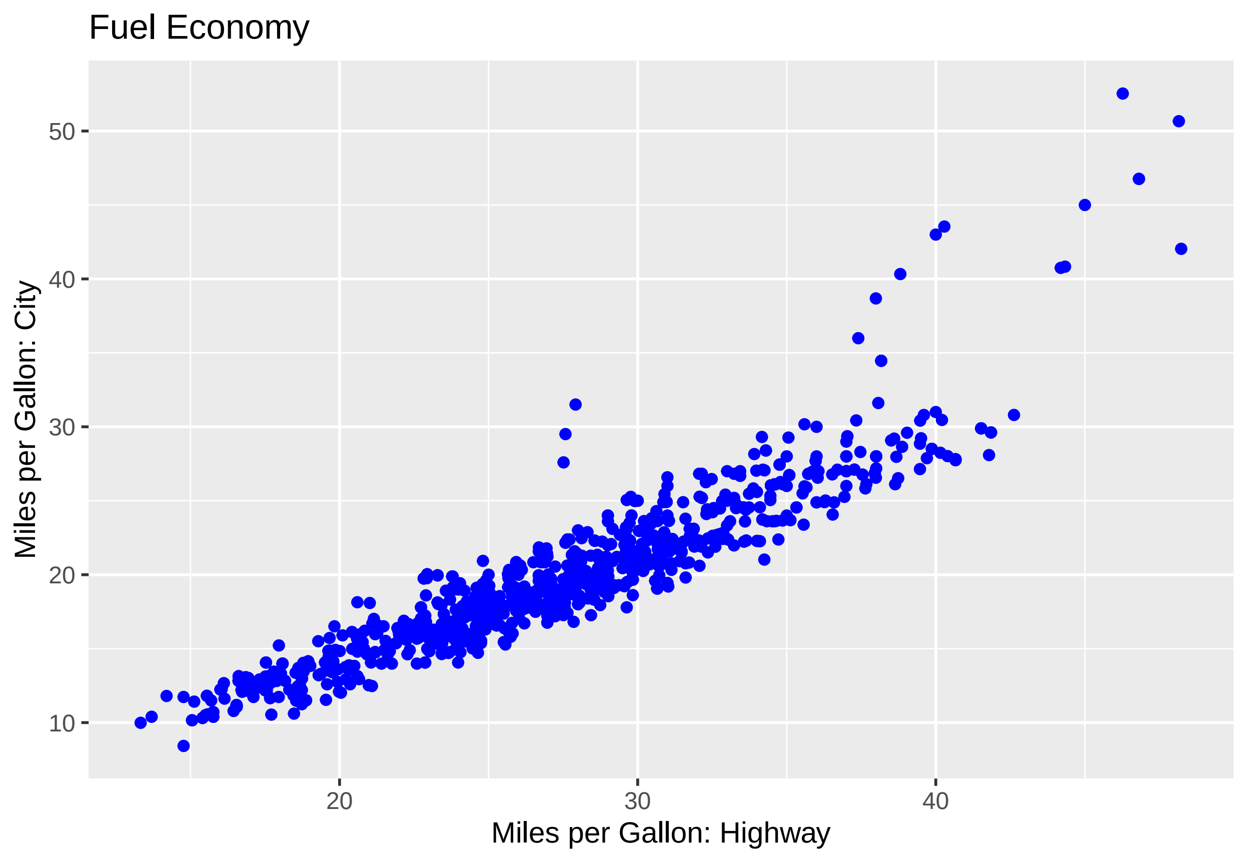
```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 8. Consider
## specifying shapes manually if you must have them.
```

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

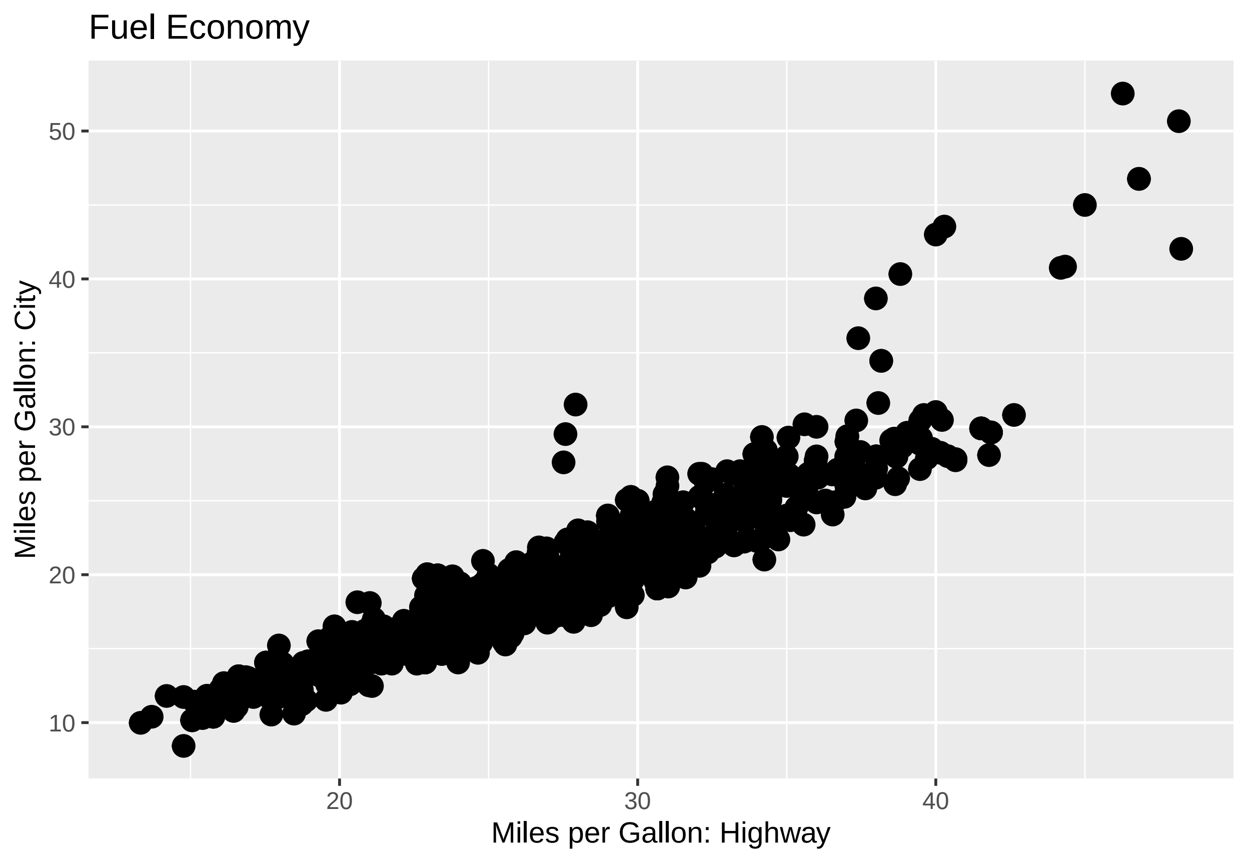


Alternatively we can adjust the color, shape, and size all the points if we do it within `geom_point` and outside 'aes"'

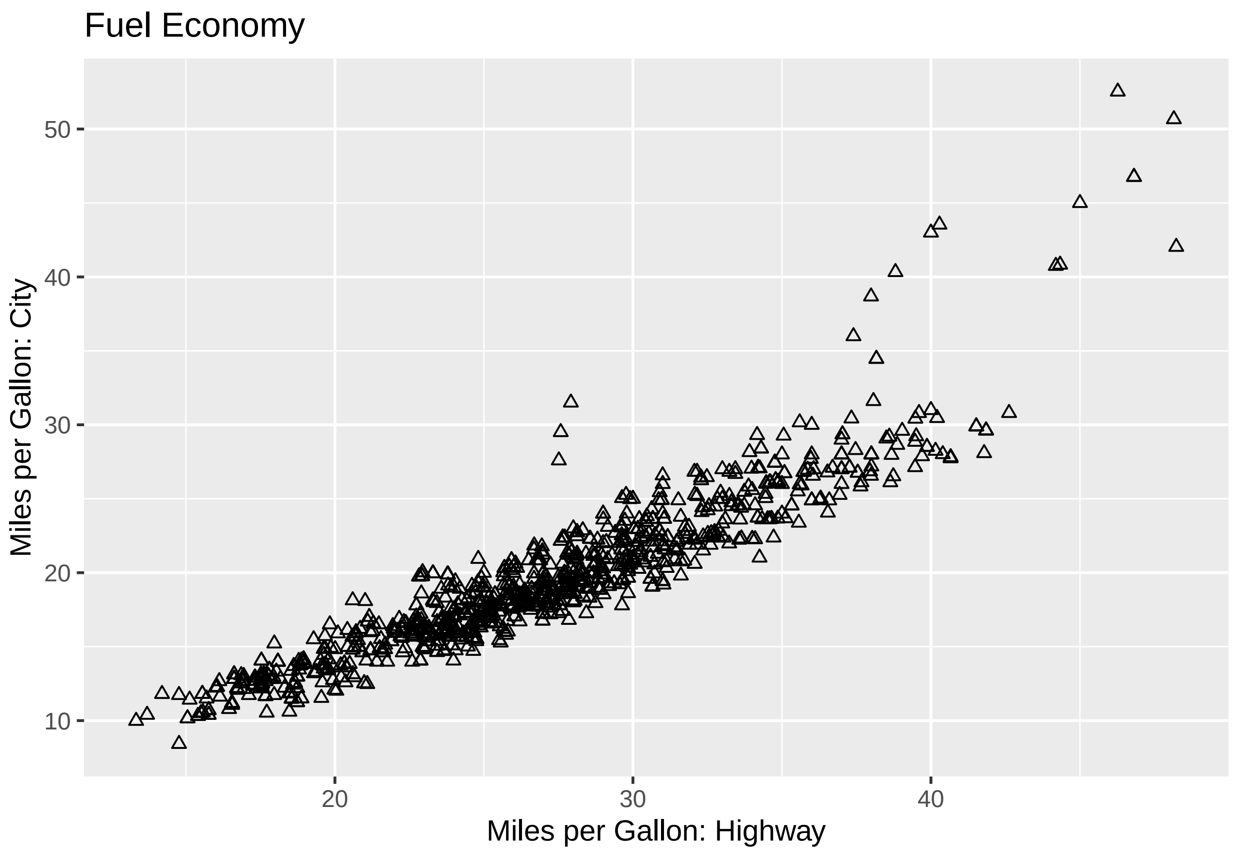
```
plot1 <- ggplot(FE2013) +
          geom_point(aes(x=FEhighway, y  = FEcity), color="blue")+
          xlab('Miles per Gallon: Highway')+
          ylab('Miles per Gallon: City')+
          ggtitle('Fuel Economy')
print(plot1)
```



```
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)
```

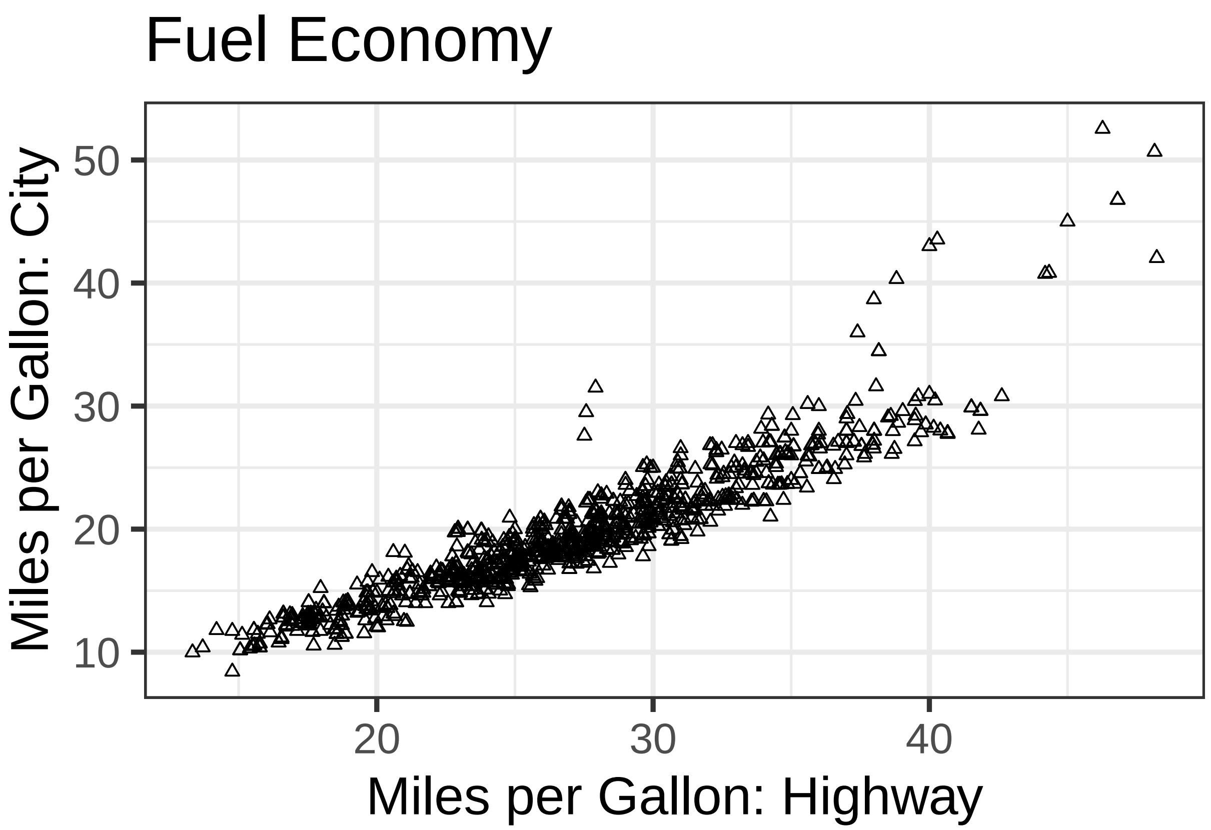


```
plot1 <- ggplot(FE2013) +
         geom_point(aes(x=FEhighway, y  = FEcity), pch=24)+
         xlab('Miles per Gallon: Highway')+
         ylab('Miles per Gallon: City')+
         ggtitle('Fuel Economy')
print(plot1)
```



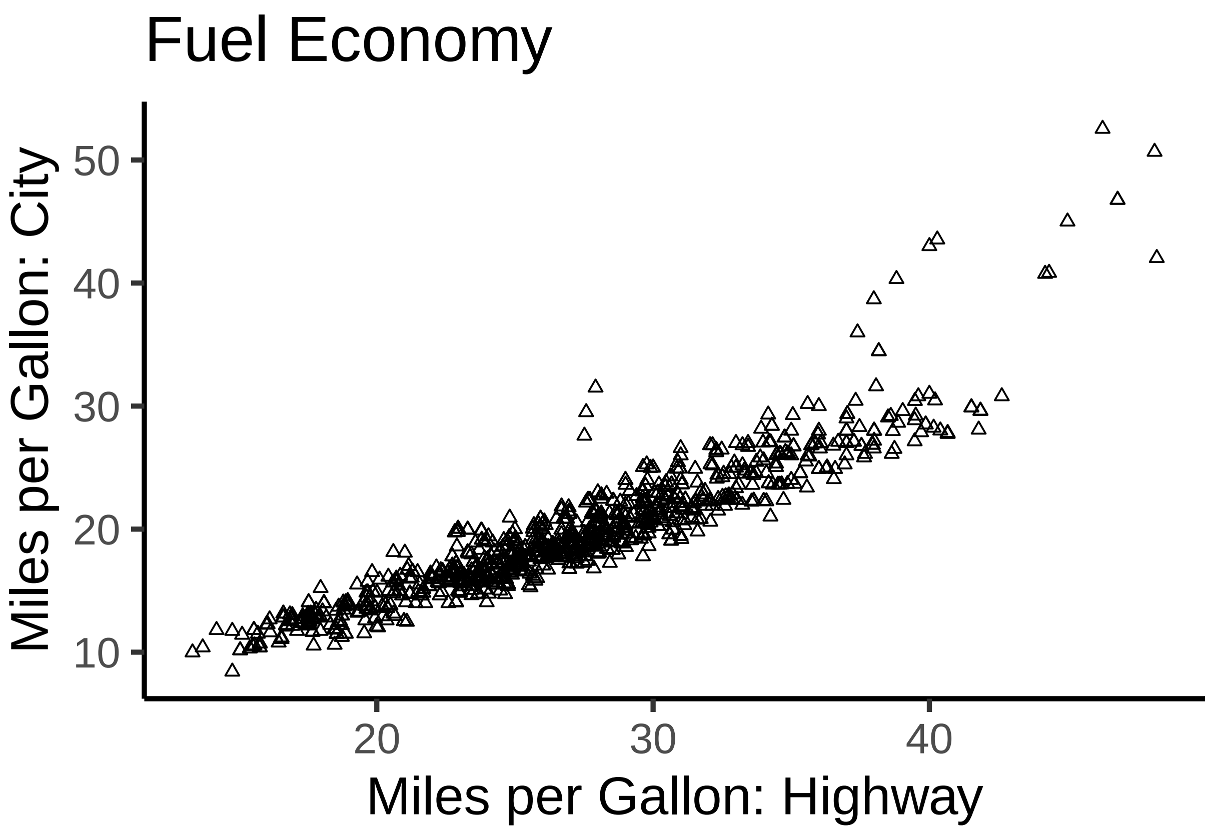
We can also change the background theme and the font side.

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

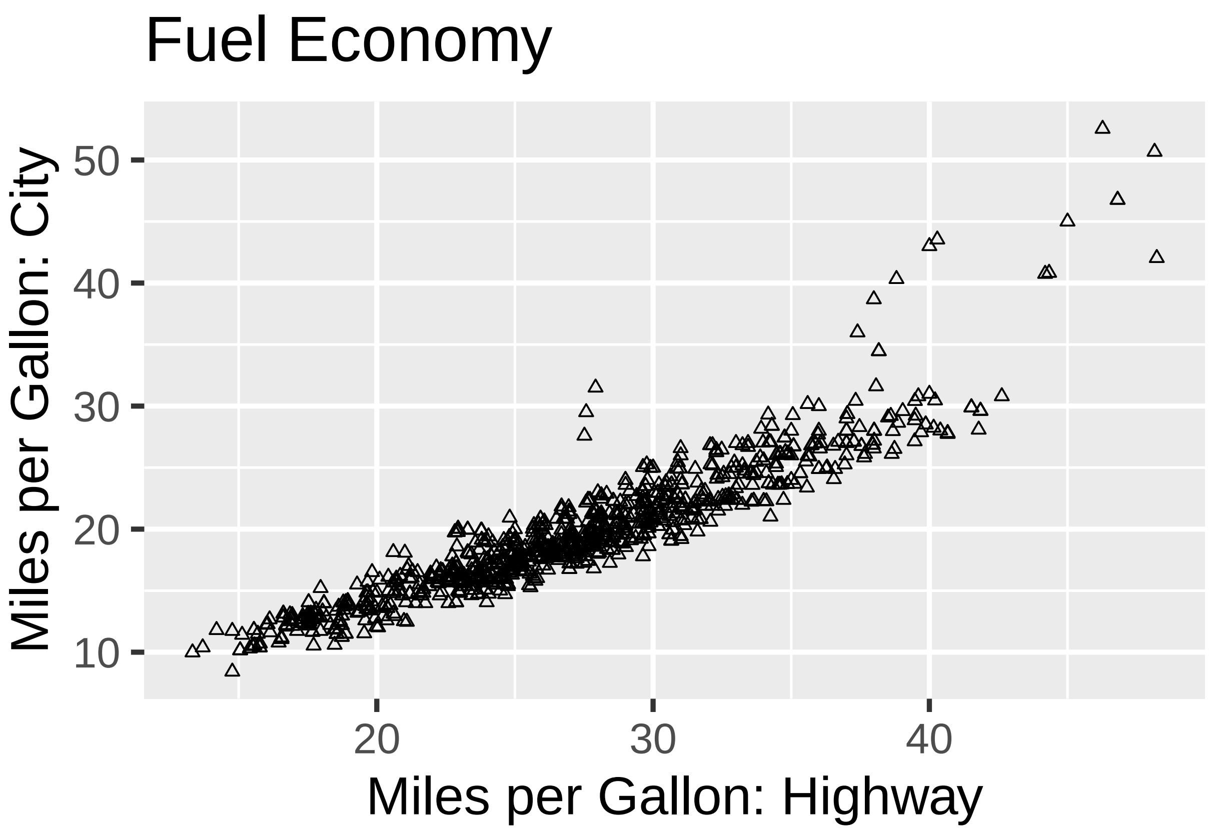


```
##theme_bw changes the color theme,
##20 means 20pt font
```

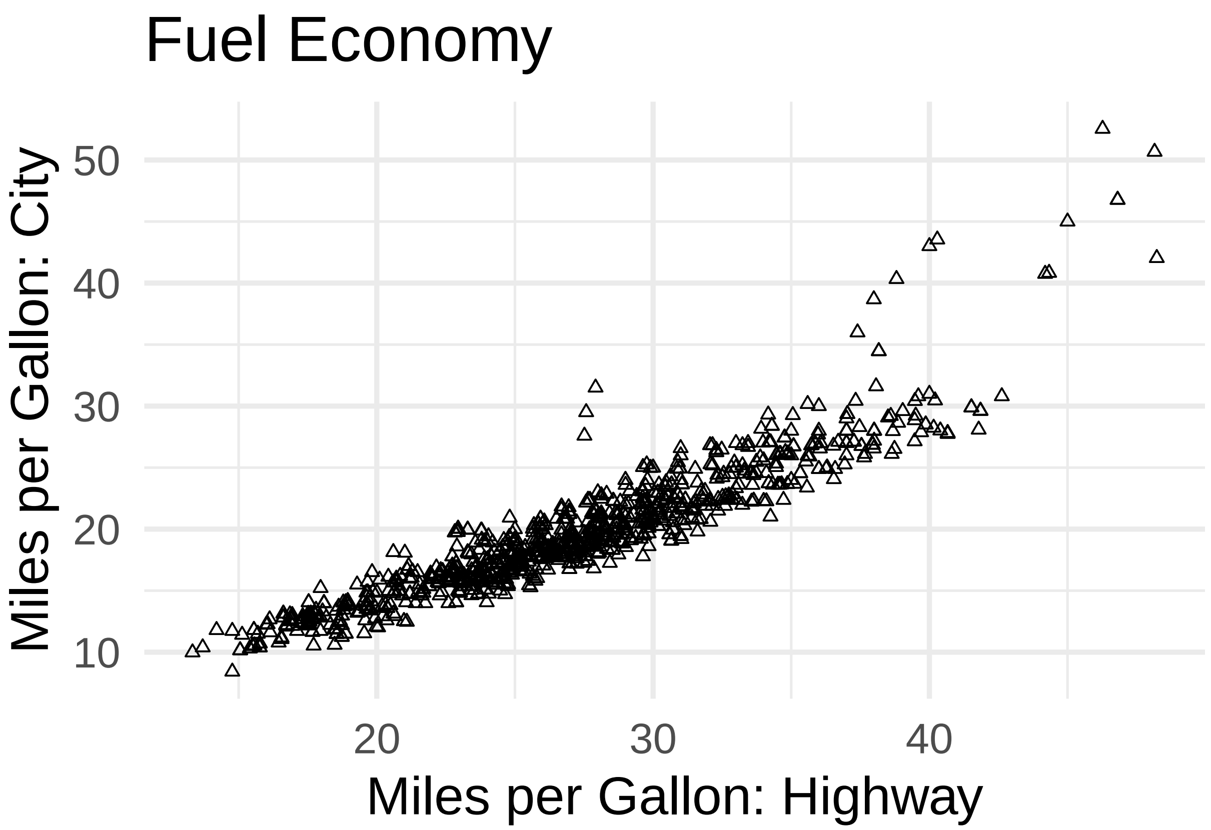
```
##Also
plot1 +
  theme_classic(20)
```



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



```r
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)/`

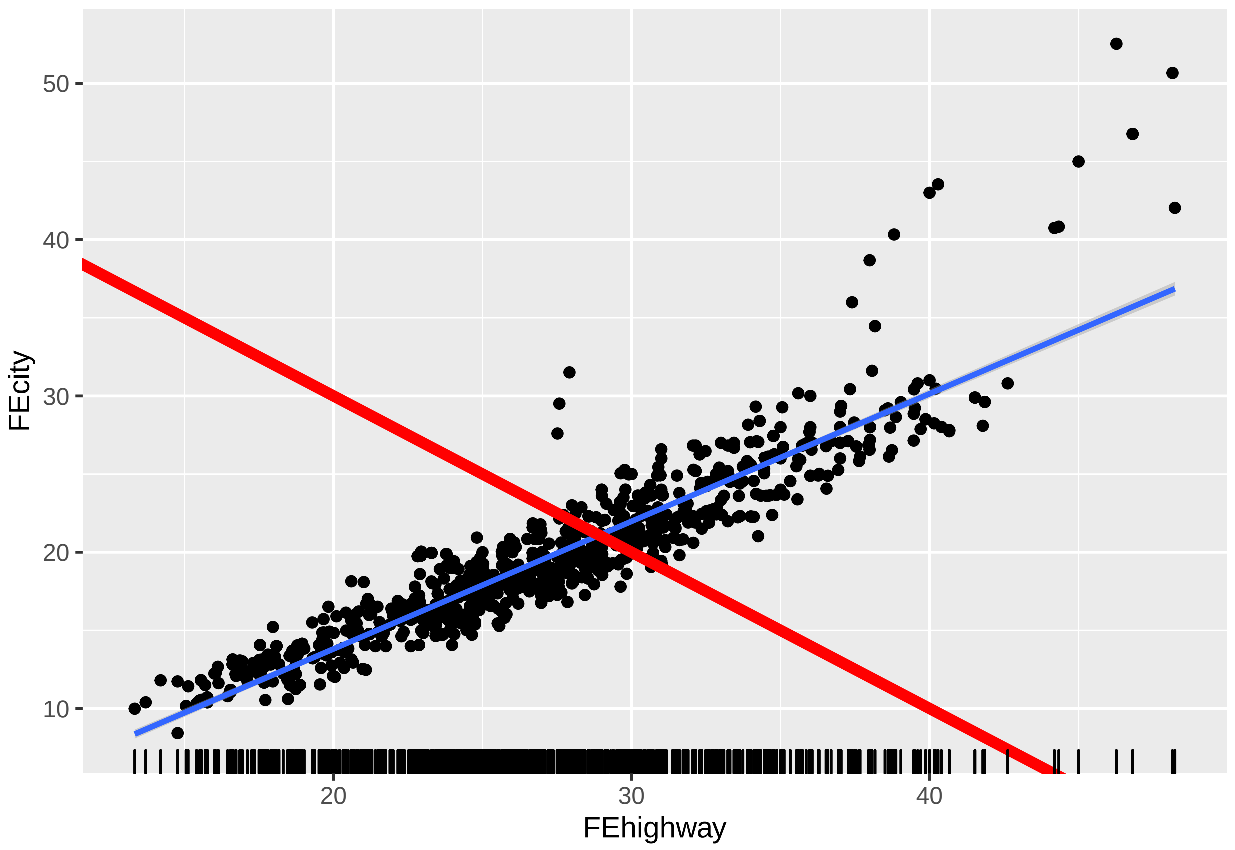
4.3 Adding addition geoms

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
print(plot2)
```

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



In theory we could keep adding on and on.

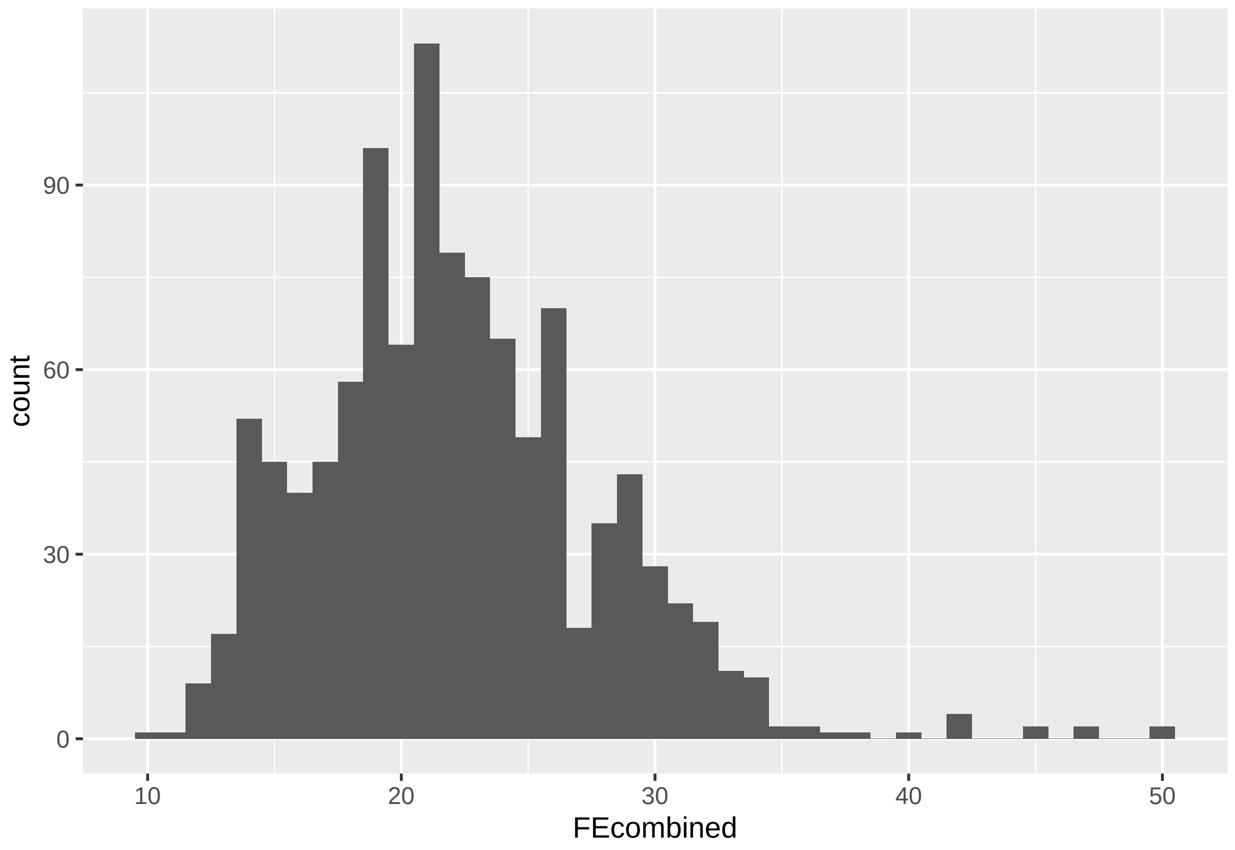
4.4 Special plots

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.

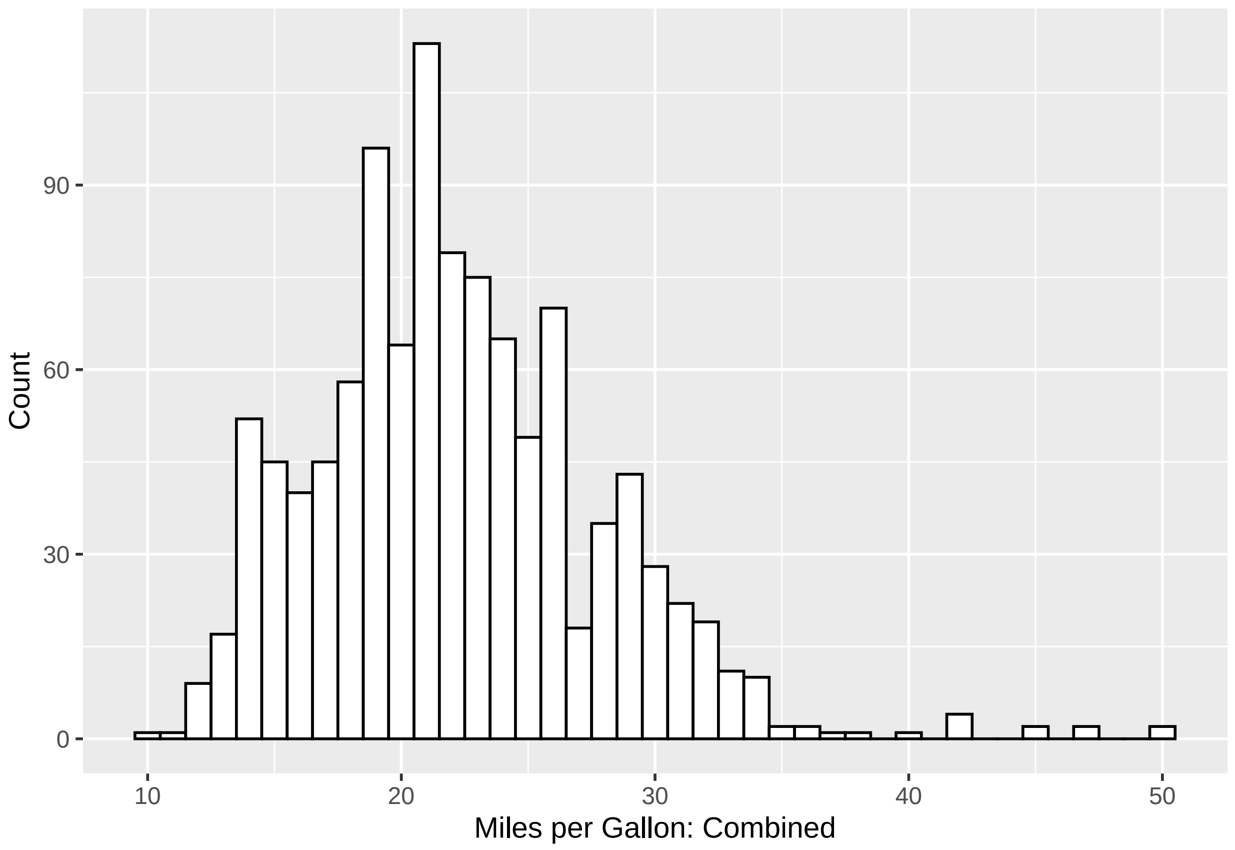
4.4.1 Histogram

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)
```

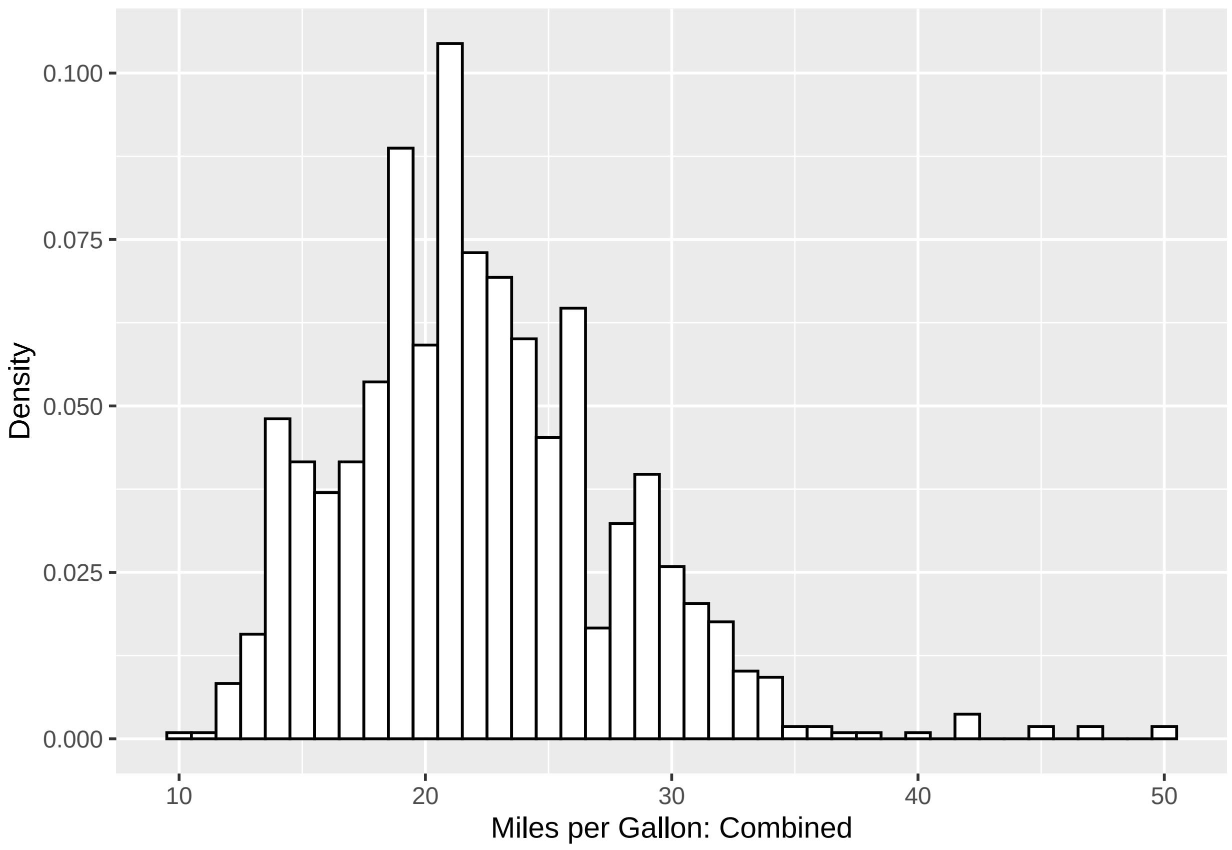


```
plot3 <- plot3 +
            geom_histogram(binwidth=1,
                           color='black', ##outline
                           fill='white')+ ##Inside
            ylab('Count')+
            xlab('Miles per Gallon: Combined')
print(plot3)
```



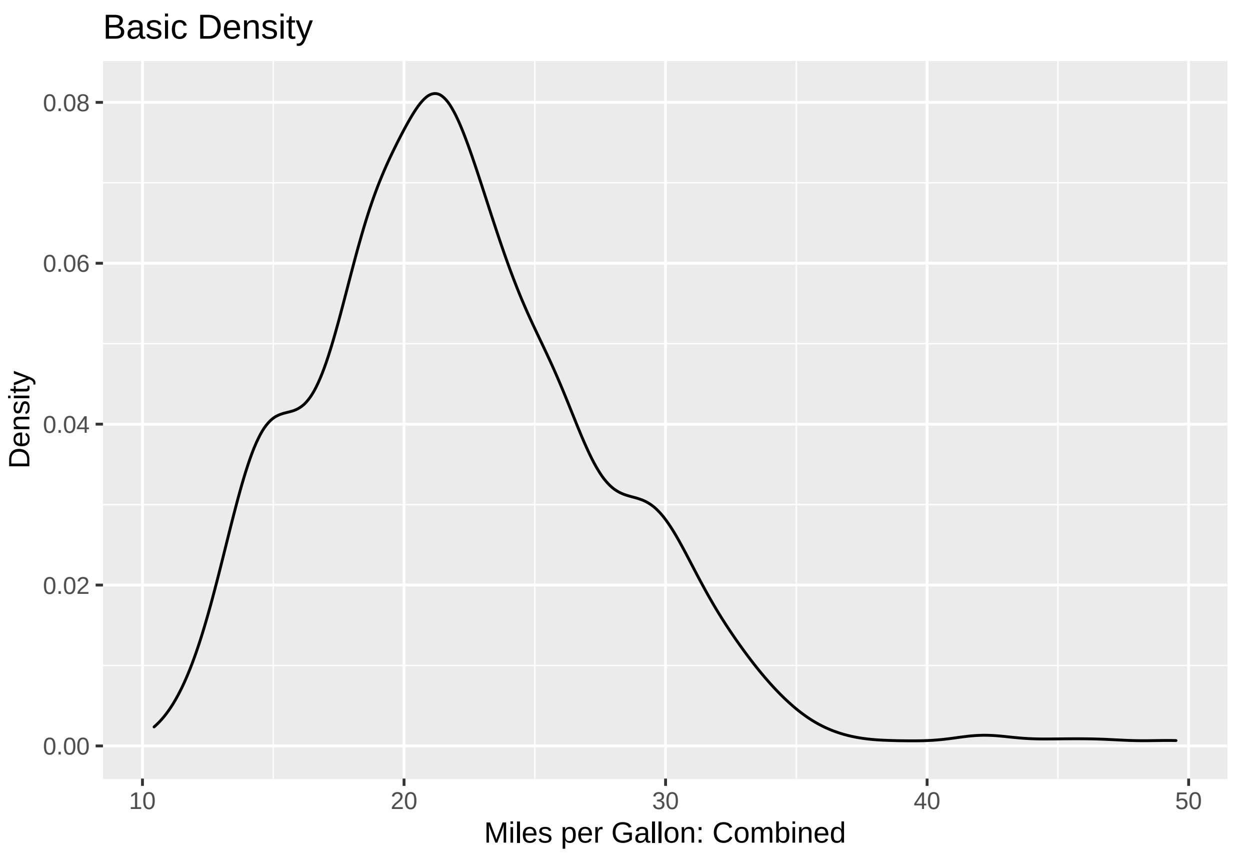
We can change counts in the histogram to density

```
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)
```



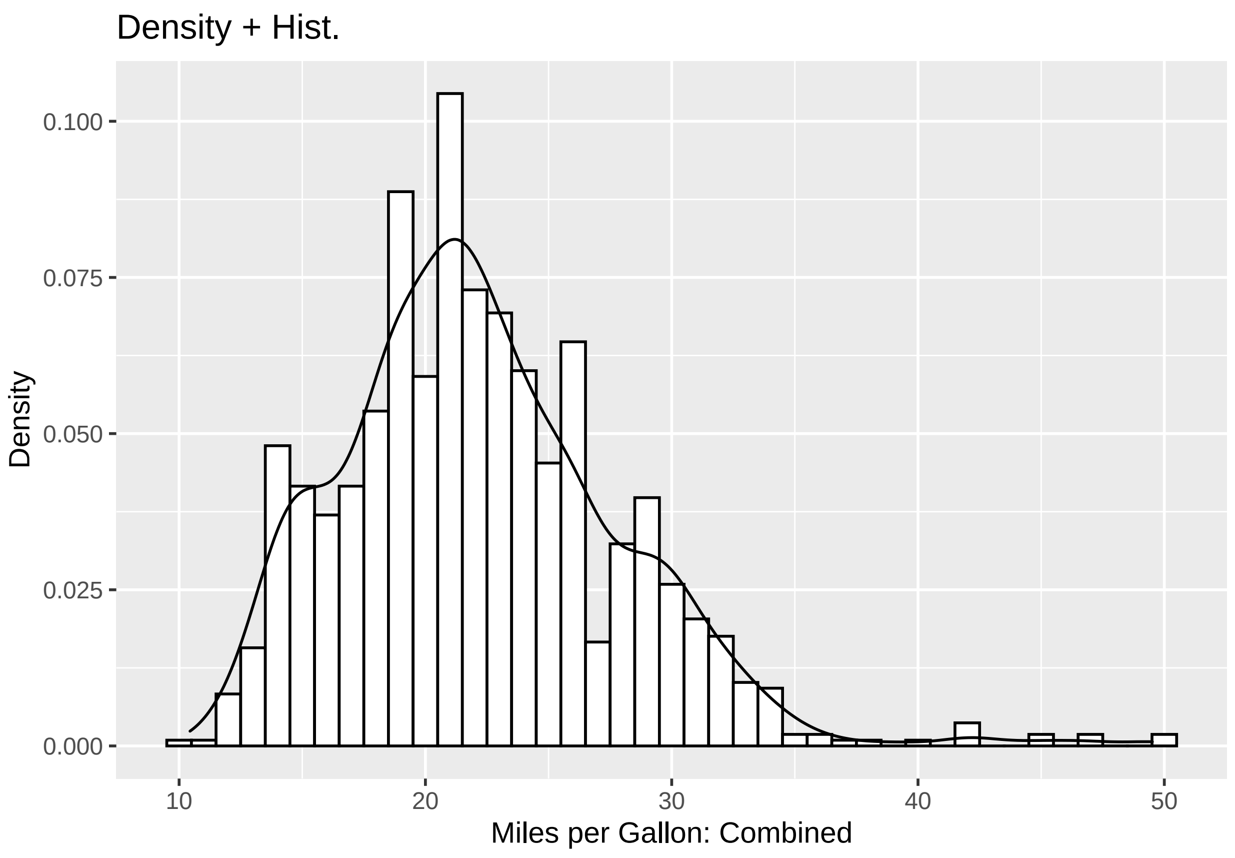
### Density We'll now look at density plots

```
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)
```



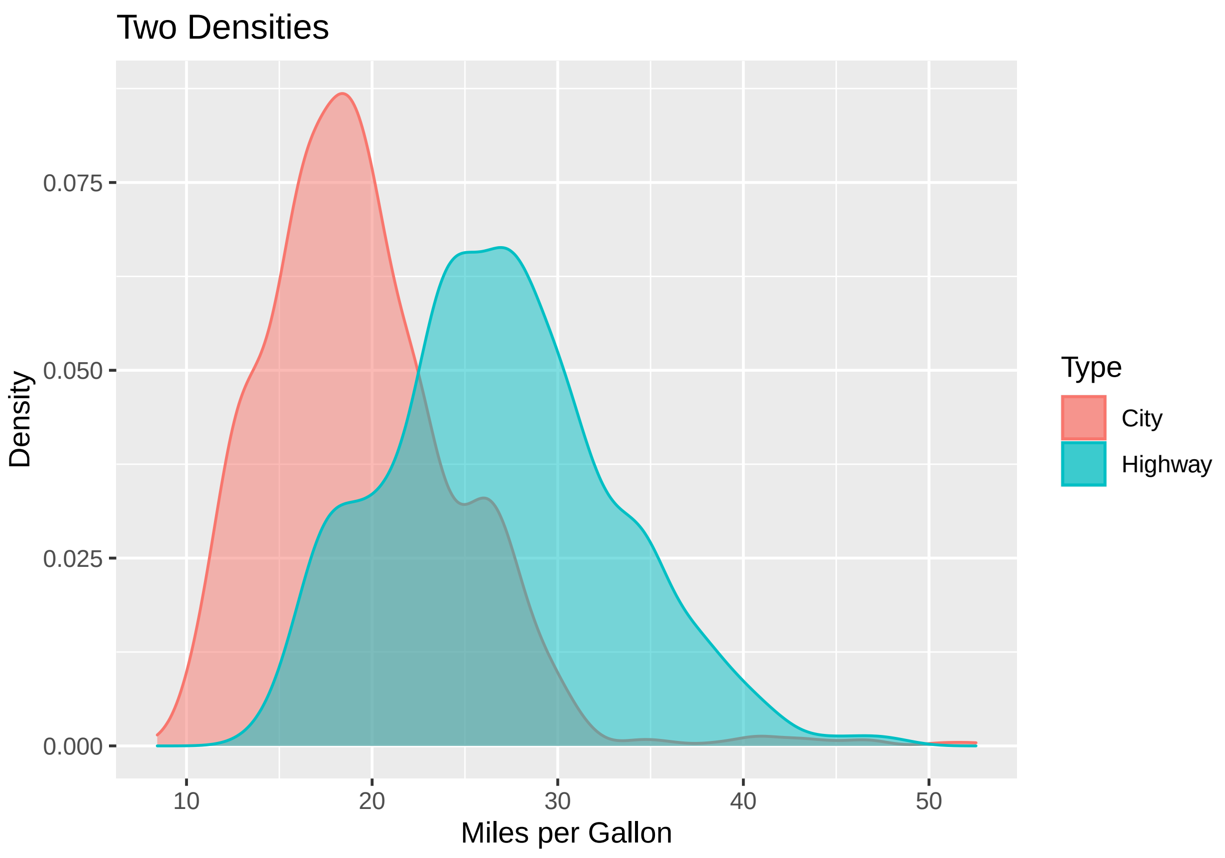
We can see that it matches by overlapping them

```
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.)*

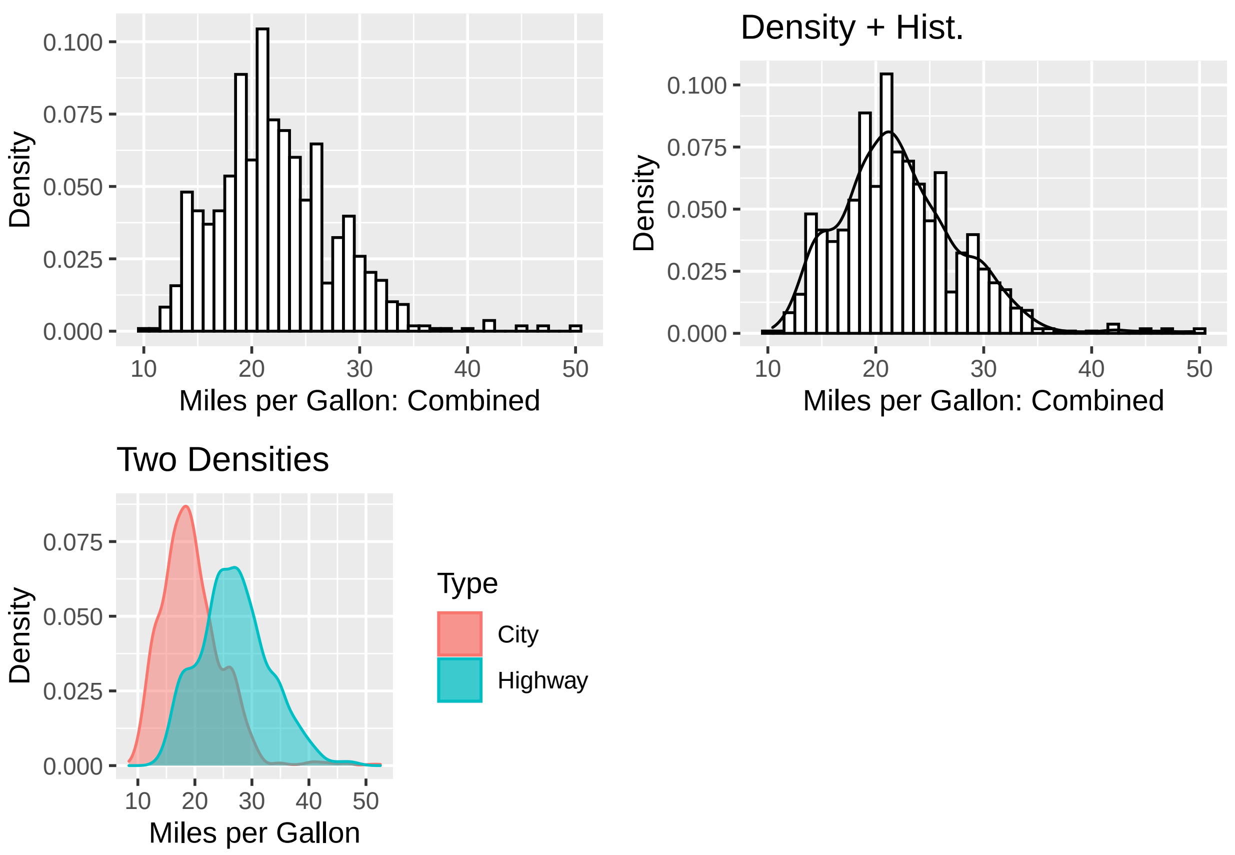
4.5 Stacking Plots

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

```
library(gridExtra)
```

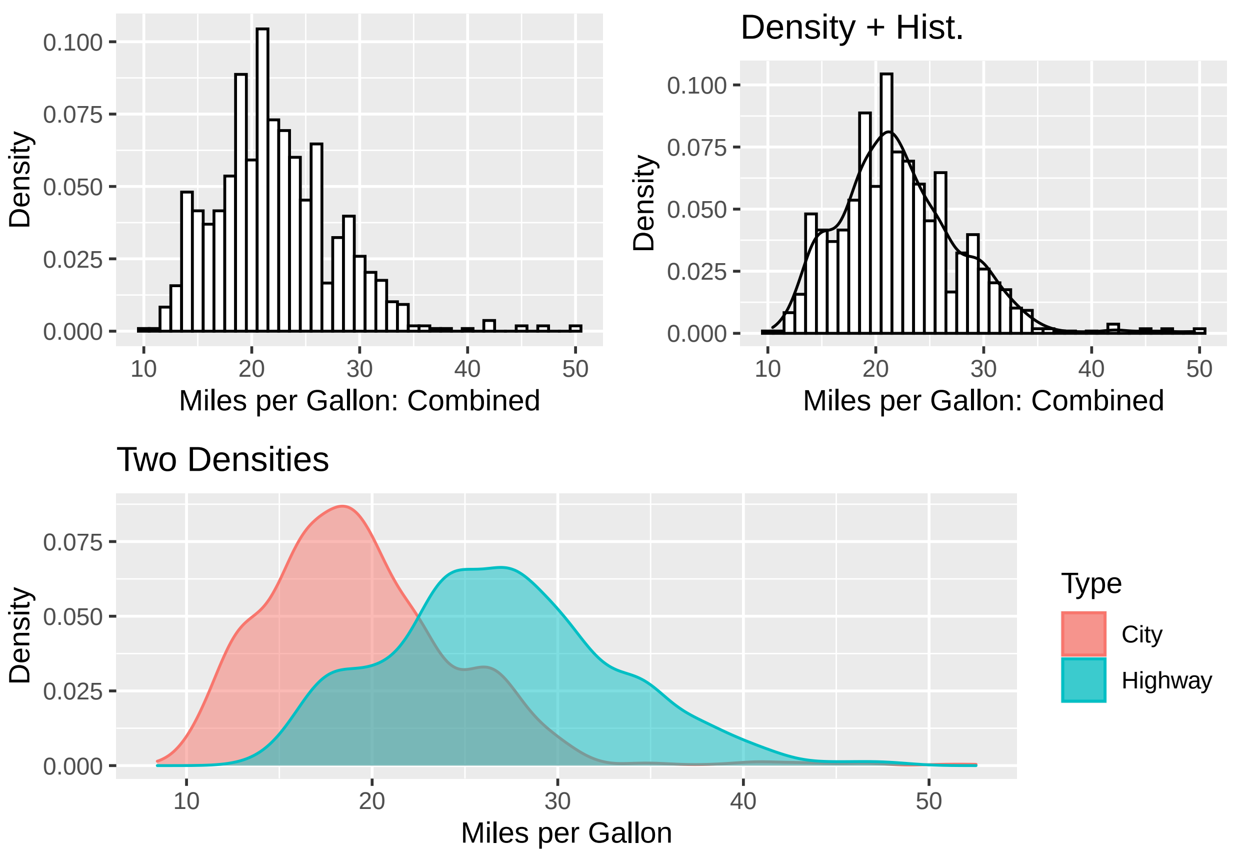
```
##
## Attaching package: 'gridExtra'
## The following object is masked from 'package:dplyr':
##
##     combine
```

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



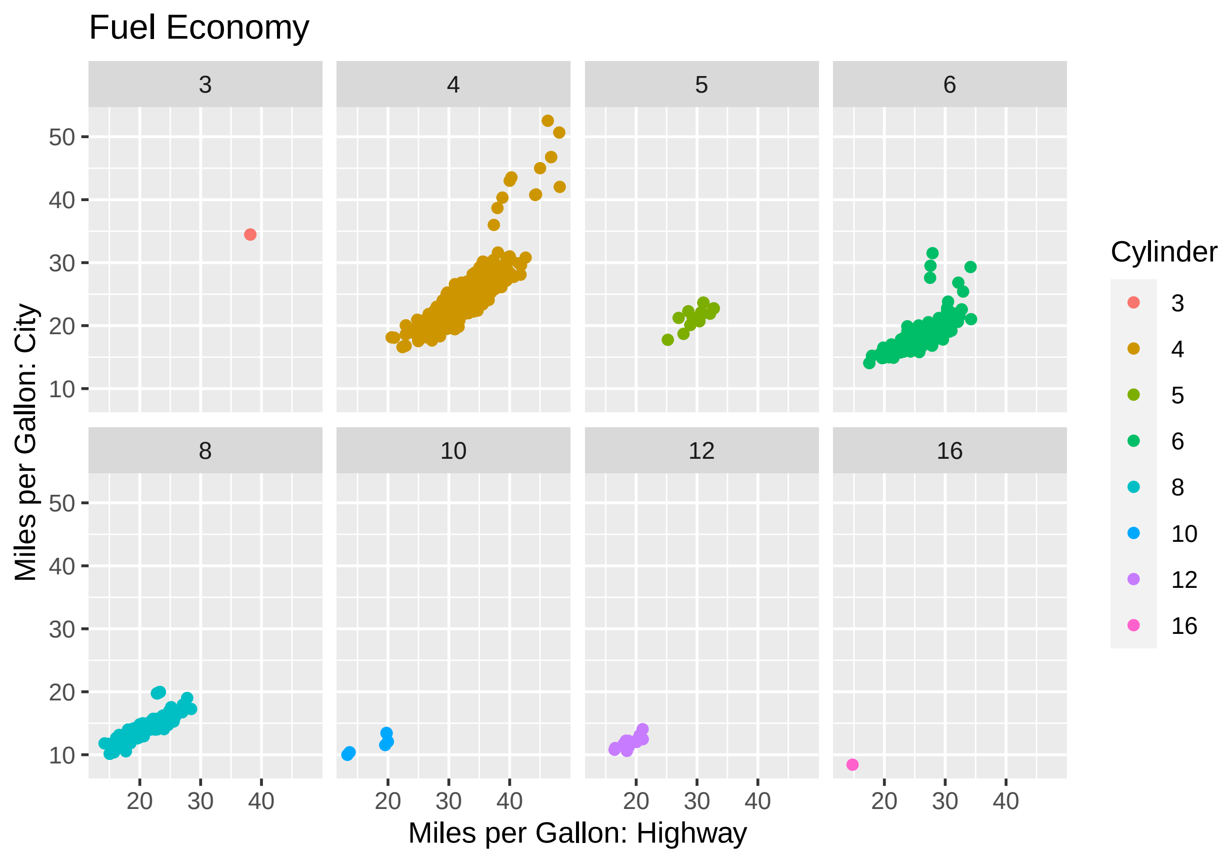
You may not like the look of this so we can adjust that bottom plot

```r
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.

5 Fitting models and other analysis

Today we'll look at running statistical models in R. Today we'll cover everything from cross-tabs to MLE estimation in R. To start we'll take a look at 2 variable contingency tables and correlation coefficients.

5.1 Bivariate relationships

We'll start by generating some related data. Let's generate a 2 column matrix composed of draws from the Multivariate Normal, i.e.

$$x \sim N\left((0,0), \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}\right)$$

To do that we need to use the function `MASS::mvrnorm`

```
library(MASS)
x <- mvrnorm(1000, ##number of draws
             mu=c(0, 0), ##Mean vector
             Sigma=matrix(c(1,  ##Var matrix
                            0.4,
                            0.4,
                            1),
                          nrow=2))


head(x)
```

```
##             [,1]        [,2]
## [1,] -0.84841814 -2.21914357
## [2,] -0.08601934 -0.84411610
## [3,] -1.41459896 -0.14974514
## [4,] -0.92456680 -0.55398579
## [5,]  0.72984619 -0.04190288
## [6,]  1.61089948  0.84633969
```

```
##Should be about the same by construction
var(x) ##Variance-Covariance matrix
```

```
##           [,1]      [,2]
## [1,] 1.0252678 0.4260349
## [2,] 0.4260349 0.9801171
```

```
cor(x) ##Correlation matrix
```

```
##           [,1]      [,2]
## [1,] 1.0000000 0.4249986
## [2,] 0.4249986 1.0000000
```

```
##If you just want the correlation coef.
cor(x[,1], x[,2]) ##Pearson's rho
```

```
## [1] 0.4249986
```

To do cross tabs we'll need to convert to categorical data.

```
x[x>0] <- TRUE
x[x<0] <- FALSE
tab1 <- table(x[,1], x[,2])  ##Work great with factors
tab1
```

```
##
##       0   1
##   0 314 176
##   1 202 308
```

```
chisq.test(tab1)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  tab1
## X-squared = 58.958, df = 1, p-value = 1.611e-14
```

```r
##Both methods can handle more than two variables
x <- runif(1000)
y <- rnorm(1000)
z <- rpois(1000, lambda=1)

cor(x, y)
```

```
## [1] -0.01072792
```

```r
cor(cbind(x,y,z))
```

```
##              x           y           z
## x  1.000000000 -0.01072792 0.006397617
## y -0.010727922  1.00000000 0.015007692
## z  0.006397617  0.01500769 1.000000000
```

```r
x[x<1/3] <-0
x[x>1/3 & x < 2/3] <- 0.5
x[x>2/3] <- 1
x <- factor(x, labels=c('Left',
                        'Middle',
                        'Right'))
table(x)
```

```
## x
##   Left Middle  Right
##    313    356    331
```

```r
y <- ifelse(y>0, "Up", "down")
table(y, x)
```

```
##       x
## y      Left Middle Right
##   down  162    183   166
##   Up    151    173   165
```

```r
chisq.test(table(y,x))
```

```
##
##  Pearson's Chi-squared test
```

```
##
## data:  table(y, x)
## X-squared = 0.18659, df = 2, p-value = 0.9109
```

But setting that aside for now let's get into estimating models

5.2 OLS: lm

For this section we'll read in two data sets.

```
library(readstata13)
FearonLaitin <- read.dta13("Datasets/FearonLaitin_CivilWar2003.dta",
                           nonint.factors = TRUE,
                           convert.dates = FALSE)
Wages <- read.dta13("Datasets/wages_full_time.dta",
                    nonint.factors = TRUE,
                    convert.dates = FALSE)
UNSC <- read.dta13("Datasets/KW_bare.dta",
                   nonint.factors = TRUE,
                   convert.dates = FALSE)
##Check the var names
colnames(FearonLaitin)
```

```
##  [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"
```

```
colnames(Wages)
```

```
## [1] "WeeklyHours" "SchoolYears" "Male"        "Age"         "Earnings"
## [6] "Wage"
```

```
colnames(UNSC)
```

```
## [1] "year"          "unmem"         "scmem"         "ccode"         "insample"
## [6] "ln_totaid96"
```

```
##and for demonstration purposes
X <- cbind(1, rnorm(1000), rnorm(1000, 1, 2))
b <- c(1, -2, 2)
y <- X %*% b + rnorm(1000)
```

OLS in R is done with the `lm` command. The `lm` command has the following options

- **formula:** The formulas takes the following form: `y ~ X1 + X2`. Where `Y` is the dependent variable and the `X`s are whatever independent variables we want to include in the model. The tilde is used to separate them. We can also include a `-1` if we want to drop the constant term.
- **data:** An argument that tells R what data frame we want to use.
- **subset"** An argument that takes logical statements. It is used to restrict the model to a certain subset of the data.
- **weights:** If you want to specify weights (i.e. Weighted Least Squares) you can put the vector of weights here
- **model, x, y:** These are arguments that tell R you want it to also return the data used to fit the model. Specifying these can be useful for knowing which observations are used to fit the model.

To fit a model, we:

```
##Ordinary model
model1 <- lm(Wage~Male+Age, data=Wages)

##Run only on Males
model2 <- lm(Wage~Age, data=Wages, subset=Male==1)
summary(model1)
```

```
##
## Call:
## lm(formula = Wage ~ Male + Age, data = Wages)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -17.918  -5.445  -1.882   2.433 143.155
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 10.04550    1.25673   7.993 3.71e-15 ***
## Male         4.50116    0.83748   5.375 9.61e-08 ***
## Age          0.06761    0.02934   2.304   0.0214 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.34 on 971 degrees of freedom
## Multiple R-squared:  0.03953,    Adjusted R-squared:  0.03755
## F-statistic: 19.98 on 2 and 971 DF,  p-value: 3.13e-09
```

```
summary(model2)
```

```
##
## Call:
## lm(formula = Wage ~ Age, data = Wages, subset = Male == 1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -17.798  -6.054  -2.289   2.428 143.129
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 14.70656    1.61993   9.079   <2e-16 ***
## Age          0.06367    0.03819   1.667    0.096 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.61 on 714 degrees of freedom
## Multiple R-squared:  0.003876,   Adjusted R-squared:  0.002481
## F-statistic: 2.779 on 1 and 714 DF,  p-value: 0.09597
```

```r
##We can also make adjustments in the formula
##Use I() to make most adjustments
model3 <- lm(log(Wage)~Male + Age +I(Age^2), data=Wages)
summary(model3)
```

```
##
## Call:
## lm(formula = log(Wage) ~ Male + Age + I(Age^2), data = Wages)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.31812 -0.24971  0.02659  0.28952  2.37650
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.522e+00  1.590e-01   9.575  < 2e-16 ***
## Male         2.358e-01  4.144e-02   5.690 1.68e-08 ***
## Age          4.414e-02  8.007e-03   5.513 4.53e-08 ***
## I(Age^2)    -4.845e-04  9.386e-05  -5.162 2.96e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5583 on 970 degrees of freedom
## Multiple R-squared:  0.07512,    Adjusted R-squared:  0.07226
## F-statistic: 26.26 on 3 and 970 DF,  p-value: 2.459e-16
```

```r
##And create interactions
model4 <- lm(log(Wage)~Male*Age, data=Wages)
summary(model4)
```

```
##
## Call:
## lm(formula = log(Wage) ~ Male * Age, data = Wages)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.39288 -0.26408  0.02981  0.30451  2.41313
```

```
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.171650   0.105530  20.578  < 2e-16 ***
## Male         0.409987   0.128129   3.200  0.00142 **
## Age          0.006493   0.002808   2.312  0.02096 *
## Male:Age    -0.004128   0.003289  -1.255  0.20978
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5655 on 970 degrees of freedom
## Multiple R-squared:  0.05125,    Adjusted R-squared:  0.04832
## F-statistic: 17.47 on 3 and 970 DF,  p-value: 4.752e-11
```

```
## Interactions with no constituents
model5 <- lm(log(Wage)~Male:Age, data=Wages)
summary(model5)
```

```
##
## Call:
## lm(formula = log(Wage) ~ Male:Age, data = Wages)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.48055 -0.25948  0.03651  0.30546  2.44708
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.435911   0.031819   76.56  < 2e-16 ***
## Male:Age    0.005652   0.000875    6.46 1.65e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5679 on 972 degrees of freedom
## Multiple R-squared:  0.04116,    Adjusted R-squared:  0.04018
## F-statistic: 41.73 on 1 and 972 DF,  p-value: 1.655e-10
```

An `lm` object contains a bunch of information, use the `names` command to see what all it contains

```
names(model1)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```
model1$coefficients ##coefs
```

```
## (Intercept)        Male         Age
## 10.04550160  4.50115599  0.06760639
```

```
head(model1$residuals)  ##residuals
```

```
##          1          2          3          4          5          6
## -6.8300230 -3.3075233 13.3796010  1.9784487 -3.2487854  0.8854556
```

```
head(model1$fitted.values) ##XB
```

```
##        1        2        3        4        5        6
## 11.33002 14.50752 17.45373 14.03428 15.89879 12.61454
```

```
head(model1$model) ##data used to fit the model
```

```
##       Wage Male Age
## 1  4.50000    0  19
## 2 11.20000    0  66
## 3 30.83333    1  43
## 4 16.01273    0  59
## 5 12.65000    1  20
## 6 13.50000    0  38
```

```
model1$call ##Returns the command used to create it
```

```
## lm(formula = Wage ~ Male + Age, data = Wages)
```

```
vcov(model1) ##returns Variance matrix of model
```

```
##             (Intercept)         Male          Age
## (Intercept)  1.57936226 -0.341770012 -0.030497885
## Male        -0.34177001  0.701380433 -0.004432516
## Age         -0.03049788 -0.004432516  0.000860787
```

```
model1 <- lm(formula = Wage ~ Male + Age, data = Wages, x=TRUE, y=TRUE)
head(model1$x) ##X values used
```

```
##   (Intercept) Male Age
## 1           1    0  19
## 2           1    0  66
## 3           1    1  43
## 4           1    0  59
## 5           1    1  20
## 6           1    0  38
```

```
head(model1$y) ##y values used
```

```
##        1        2        3        4        5        6
##  4.50000 11.20000 30.83333 16.01273 12.65000 13.50000
```

```
##Check results
##notice we can abbreviate elements of the lm
summary(cbind(model1$x %*% model1$coef,
              model1$fitted))
```

```
##        V1              V2
##  Min.   :11.26   Min.   :11.26
##  1st Qu.:13.92   1st Qu.:13.92
##  Median :16.85   Median :16.85
##  Mean   :16.01   Mean   :16.01
##  3rd Qu.:17.66   3rd Qu.:17.66
##  Max.   :20.36   Max.   :20.36
```

```
summary(cbind(model1$y-model1$fitted,
              model1$resid))
```

```
##        V1                 V2
##  Min.   :-17.918   Min.   :-17.918
##  1st Qu.: -5.445   1st Qu.: -5.445
##  Median : -1.882   Median : -1.882
##  Mean   :  0.000   Mean   :  0.000
##  3rd Qu.:  2.433   3rd Qu.:  2.433
##  Max.   :143.155   Max.   :143.155
```

```r
##If the data is in matrix form then we still use the formula
##Just not the data argument

##We need -1 because we have our own constant
summary(lm(y~X -1 ))
```

```
##
## Call:
## lm(formula = y ~ X - 1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.5814 -0.6718 -0.0077  0.6799  3.0912
##
## Coefficients:
##    Estimate Std. Error t value Pr(>|t|)
## X1  0.97533    0.03461   28.18   <2e-16 ***
## X2 -1.95144    0.03129  -62.36   <2e-16 ***
## X3  1.99859    0.01592  125.51   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9909 on 997 degrees of freedom
## Multiple R-squared:  0.9647, Adjusted R-squared:  0.9646
## F-statistic:  9084 on 3 and 997 DF,  p-value: < 2.2e-16
```

To get fitted values with standard errors we can use the predict command.

```r
modelFit <- predict(model1, se.fit=TRUE)
#This returns a list \hat{y} and s.e.(\hat{yat})

#Create a profile of data that is of interest to us.
profile <- data.frame(Male = 1,
                      Age = seq(18, 80, length.out=15))
fitted <- predict(model1, se.fit=TRUE, newdata=profile)
fitted
```

```
## $fit
##        1        2        3        4        5        6        7        8
## 15.76357 16.06297 16.36237 16.66177 16.96117 17.26057 17.55997 17.85937
##        9       10       11       12       13       14       15
## 18.15877 18.45817 18.75757 19.05697 19.35637 19.65577 19.95517
##
## $se.fit
##         1         2         3         4         5         6         7         8
## 0.7865133 0.6806876 0.5846641 0.5040758 0.4473440 0.4241519 0.4398374 0.4906863
##         9        10        11        12        13        14        15
## 0.5673209 0.6608305 0.7650521 0.8761715 0.9918731 1.1107258 1.2318180
##
## $df
## [1] 971
##
## $residual.scale
## [1] 11.34435
```

The `predict` function can also return confidence or prediction intervals:

```
predict(model1, interval = "confidence", newdata=profile)
```

```
##         fit      lwr      upr
## 1  15.76357 14.22011 17.30703
## 2  16.06297 14.72718 17.39876
## 3  16.36237 15.21502 17.50972
## 4  16.66177 15.67257 17.65098
## 5  16.96117 16.08330 17.83904
## 6  17.26057 16.42821 18.09293
## 7  17.55997 16.69683 18.42311
## 8  17.85937 16.89644 18.82230
## 9  18.15877 17.04545 19.27209
## 10 18.45817 17.16135 19.75499
## 11 18.75757 17.25622 20.25892
## 12 19.05697 17.33756 20.77638
## 13 19.35637 17.40991 21.30283
## 14 19.65577 17.47607 21.83547
## 15 19.95517 17.53784 22.37250
```

```
predict(model1, interval = "prediction", newdata=profile)
```

```
##         fit       lwr      upr
## 1  15.76357 -6.552137 38.07928
## 2  16.06297 -6.239336 38.36528
## 3  16.36237 -5.929443 38.65419
## 4  16.66177 -5.622463 38.94601
## 5  16.96117 -5.318399 39.24074
## 6  17.26057 -5.017253 39.53839
## 7  17.55997 -4.719024 39.83897
## 8  17.85937 -4.423713 40.14245
## 9  18.15877 -4.131319 40.44886
## 10 18.45817 -3.841838 40.75818
## 11 18.75757 -3.555266 41.07041
## 12 19.05697 -3.271599 41.38554
## 13 19.35637 -2.990830 41.70357
## 14 19.65577 -2.712952 42.02449
## 15 19.95517 -2.437957 42.34829
```

5.2.1 Robust Standard Errors and Hypothesis Testing

Suppose we wanted robust standard errors, there are actually a few ways to do this. Most common is the following

```
library(sandwich)
library(lmtest)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
###sandwich::vcovHC returns the robust covariance matrix
##use lmtest::coeftest to get the t test
coeftest(model1, vcovHC)
```

```
##
## t test of coefficients:
##
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 10.045502   1.093048  9.1904  < 2e-16 ***
## Male         4.501156   0.643660  6.9931    5e-12 ***
## Age          0.067606   0.029183  2.3166  0.02073 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
## Also can just give it a matrix (like from a bootstrap)
coeftest(model1, vcovHC(model3))
```

```
##
## t test of coefficients:
##
##              Estimate Std. Error  t value  Pr(>|t|)
## (Intercept) 10.045502   0.242885  41.3590 < 2.2e-16 ***
## Male         4.501156   0.038620 116.5512 < 2.2e-16 ***
## Age          0.067606   0.013457   5.0239 6.024e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

`coeftest` is a great function where you can use new covariance matrices on your models.

The car package also offers a function for joint hypothesis testing

```
library(car)
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

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

```
summary(model3)
```

```
##
```

```
## Call:
## lm(formula = log(Wage) ~ Male + Age + I(Age^2), data = Wages)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.31812 -0.24971  0.02659  0.28952  2.37650
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.522e+00  1.590e-01   9.575  < 2e-16 ***
## Male         2.358e-01  4.144e-02   5.690 1.68e-08 ***
## Age          4.414e-02  8.007e-03   5.513 4.53e-08 ***
## I(Age^2)    -4.845e-04  9.386e-05  -5.162 2.96e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5583 on 970 degrees of freedom
## Multiple R-squared:  0.07512,    Adjusted R-squared:  0.07226
## F-statistic: 26.26 on 3 and 970 DF,  p-value: 2.459e-16
```

```
##Same as the basic t-test
linearHypothesis(model3, c("Age=0"))
```

```
## Linear hypothesis test
##
## Hypothesis:
## Age = 0
##
## Model 1: restricted model
## Model 2: log(Wage) ~ Male + Age + I(Age^2)
##
##   Res.Df    RSS Df Sum of Sq     F   Pr(>F)
## 1    971 311.87
## 2    970 302.40  1     9.474 30.39 4.53e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##Special test
linearHypothesis(model3, c("Age=2"))
```

```
## Linear hypothesis test
##
## Hypothesis:
## Age = 2
##
## Model 1: restricted model
## Model 2: log(Wage) ~ Male + Age + I(Age^2)
##
##   Res.Df     RSS Df Sum of Sq     F    Pr(>F)
## 1    971 18903.6
## 2    970   302.4  1     18601 59667 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##Test if they have the same coefficient
linearHypothesis(model3, c("Age=I(Age^2)"))
```

```
## Linear hypothesis test
##
## Hypothesis:
## Age - I(Age^2) = 0
##
## Model 1: restricted model
## Model 2: log(Wage) ~ Male + Age + I(Age^2)
##
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    971 311.86
## 2    970 302.40  1    9.4636 30.356 4.607e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##Joint significance of Age
linearHypothesis(model3, c("Age=0", "I(Age^2)=0"))
```

```
## Linear hypothesis test
```

```
##
## Hypothesis:
## Age = 0
## I(Age^2) = 0
##
## Model 1: restricted model
## Model 2: log(Wage) ~ Male + Age + I(Age^2)
##
##   Res.Df    RSS Df Sum of Sq      F   Pr(>F)
## 1    972 312.52
## 2    970 302.40  2    10.123 16.236 1.16e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##Full F test
linearHypothesis(model3, c("Age=0", "I(Age^2)=0", "Male=0"))
```

```
## Linear hypothesis test
##
## Hypothesis:
## Age = 0
## I(Age^2) = 0
## Male = 0
##
## Model 1: restricted model
## Model 2: log(Wage) ~ Male + Age + I(Age^2)
##
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    973 326.96
## 2    970 302.40  3    24.561 26.261 2.459e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
###Can also test with robust errors
linearHypothesis(model3,
                 c("Age=0", "I(Age^2)=0", "Male=0"),
                 vcov=vcovHC)
```

```
## Linear hypothesis test
##
## Hypothesis:
## Age = 0
## I(Age^2) = 0
## Male = 0
##
## Model 1: restricted model
## Model 2: log(Wage) ~ Male + Age + I(Age^2)
##
## Note: Coefficient covariance matrix supplied.
##
##   Res.Df Df     F    Pr(>F)
## 1    973
## 2    970  3 26.46 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
## Also can just give it a matrix (like from a bootstrap)
linearHypothesis(model3,
                 c("Age=0", "I(Age^2)=0", "Male=0"),
                 vcov=vcovHC(model3))
```

```
## Linear hypothesis test
##
## Hypothesis:
## Age = 0
## I(Age^2) = 0
## Male = 0
##
## Model 1: restricted model
## Model 2: log(Wage) ~ Male + Age + I(Age^2)
##
## Note: Coefficient covariance matrix supplied.
##
##   Res.Df Df     F    Pr(>F)
## 1    973
## 2    970  3 26.46 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
###In the same vein there's also a function for nested model testing###
model6 <- lm(log(Wage)~Male , data=Wages)

##Same as joint Significance test on Age above
##This is an example of nested model testing
waldtest(model3, model6)
```

```
## Wald test
##
## Model 1: log(Wage) ~ Male + Age + I(Age^2)
## Model 2: log(Wage) ~ Male
##   Res.Df Df      F   Pr(>F)
## 1    970
## 2    972 -2 16.236 1.16e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

5.2.2 Fixed Effects and Clustered Errors

To estimate fixed effects we have some options. The first is to use dummies

```
FEmodel1 <- lm(ln_totaid96~scmem+factor(ccode), data=UNSC)
summary(FEmodel1)
```

```
##
## Call:
## lm(formula = ln_totaid96 ~ scmem + factor(ccode), data = UNSC)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.4800  -3.1966  -0.2903   3.9922  17.8827
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       1.236e+01  8.004e-01  15.436  < 2e-16 ***
## scmem             1.489e+00  2.953e-01   5.042 4.70e-07 ***
## factor(ccode)AGO -5.374e+00  1.132e+00  -4.747 2.09e-06 ***
## factor(ccode)ALB -8.818e+00  1.132e+00  -7.790 7.41e-15 ***
## factor(ccode)ARG -2.583e+00  1.134e+00  -2.277 0.022817 *
## factor(ccode)ARM -9.159e+00  1.132e+00  -8.091 6.64e-16 ***
## factor(ccode)AUS -1.002e+01  1.133e+00  -8.849  < 2e-16 ***
## factor(ccode)AUT -5.707e+00  1.132e+00  -5.041 4.72e-07 ***
## factor(ccode)AZE -9.987e+00  1.132e+00  -8.822  < 2e-16 ***
## factor(ccode)BDI -2.404e+00  1.132e+00  -2.124 0.033719 *
## factor(ccode)BEL -7.131e+00  1.133e+00  -6.295 3.20e-10 ***
## factor(ccode)BEN -1.691e+00  1.132e+00  -1.494 0.135282
## factor(ccode)BFA -1.354e+00  1.132e+00  -1.196 0.231782
## factor(ccode)BGD -2.610e+00  1.132e+00  -2.305 0.021185 *
## factor(ccode)BGR -9.461e+00  1.132e+00  -8.357  < 2e-16 ***
## factor(ccode)BHR -9.205e+00  1.132e+00  -8.131 4.78e-16 ***
## factor(ccode)BHS -8.469e+00  1.132e+00  -7.482 7.98e-14 ***
## factor(ccode)BIH -9.425e+00  1.132e+00  -8.326  < 2e-16 ***
## factor(ccode)BLR -9.540e+00  1.132e+00  -8.427  < 2e-16 ***
## factor(ccode)BLZ -1.583e+00  1.132e+00  -1.399 0.161979
## factor(ccode)BOL  3.610e+00  1.132e+00   3.188 0.001435 **
## factor(ccode)BRA  2.350e+00  1.135e+00   2.070 0.038460 *
## factor(ccode)BRB -9.025e+00  1.132e+00  -7.973 1.73e-15 ***
## factor(ccode)BRN -1.236e+01  1.132e+00 -10.915  < 2e-16 ***
## factor(ccode)BTN -8.936e+00  1.132e+00  -7.894 3.25e-15 ***
## factor(ccode)BWA -2.440e+00  1.132e+00  -2.155 0.031186 *
## factor(ccode)CAF -2.875e+00  1.132e+00  -2.540 0.011103 *
## factor(ccode)CAN -1.144e+01  1.134e+00 -10.093  < 2e-16 ***
## factor(ccode)CHE -1.236e+01  1.132e+00 -10.915  < 2e-16 ***
## factor(ccode)CHL  2.357e+00  1.132e+00   2.082 0.037391 *
## factor(ccode)CHN -9.447e+00  1.132e+00  -8.345  < 2e-16 ***
## factor(ccode)CIV -2.588e+00  1.132e+00  -2.286 0.022303 *
## factor(ccode)CMR -1.367e+00  1.132e+00  -1.208 0.227148
## factor(ccode)COG -3.992e+00  1.132e+00  -3.526 0.000423 ***
## factor(ccode)COL  3.222e+00  1.133e+00   2.842 0.004486 **
## factor(ccode)COM -7.864e+00  1.132e+00  -6.947 3.96e-12 ***
## factor(ccode)CPV -5.078e+00  1.132e+00  -4.486 7.36e-06 ***
## factor(ccode)CRI  2.373e+00  1.132e+00   2.096 0.036139 *
## factor(ccode)CUB -9.253e+00  1.132e+00  -8.170 3.46e-16 ***
## factor(ccode)CYP -2.589e+00  1.132e+00  -2.287 0.022217 *
## factor(ccode)CZE -9.893e+00  1.132e+00  -8.739  < 2e-16 ***
## factor(ccode)DEU -6.719e+00  1.132e+00  -5.933 3.07e-09 ***
## factor(ccode)DJI -6.252e+00  1.132e+00  -5.523 3.43e-08 ***
## factor(ccode)DMA -1.207e+01  1.132e+00 -10.658  < 2e-16 ***
## factor(ccode)DNK -6.975e+00  1.132e+00  -6.160 7.59e-10 ***
## factor(ccode)DOM  2.373e+00  1.132e+00   2.096 0.036090 *
## factor(ccode)DZA -4.232e+00  1.132e+00  -3.738 0.000187 ***
## factor(ccode)ECU  2.783e+00  1.132e+00   2.458 0.013992 *
## factor(ccode)EGY  3.223e+00  1.133e+00   2.845 0.004450 **
## factor(ccode)ERI -9.665e+00  1.132e+00  -8.538  < 2e-16 ***
## factor(ccode)ESP  2.338e-01  1.132e+00   0.206 0.836463
## factor(ccode)EST -9.776e+00  1.132e+00  -8.636  < 2e-16 ***
## factor(ccode)ETH  2.559e+00  1.132e+00   2.260 0.023824 *
## factor(ccode)FIN -9.341e+00  1.132e+00  -8.251  < 2e-16 ***
## factor(ccode)FJI -1.215e+01  1.132e+00 -10.732  < 2e-16 ***
## factor(ccode)FRA -6.564e+00  1.132e+00  -5.798 6.90e-09 ***
## factor(ccode)FSM -1.158e+01  1.132e+00 -10.234  < 2e-16 ***
## factor(ccode)GAB -3.110e+00  1.132e+00  -2.747 0.006034 **
## factor(ccode)GBR -7.045e+00  1.132e+00  -6.224 5.06e-10 ***
## factor(ccode)GEO -9.234e+00  1.132e+00  -8.157 3.86e-16 ***
## factor(ccode)GHA  4.573e-01  1.132e+00   0.404 0.686304
## factor(ccode)GIN -6.559e-01  1.132e+00  -0.579 0.562358
## factor(ccode)GMB -2.505e+00  1.132e+00  -2.213 0.026954 *
## factor(ccode)GNB -5.430e+00  1.132e+00  -4.797 1.64e-06 ***
## factor(ccode)GNQ -8.940e+00  1.132e+00  -7.897 3.16e-15 ***
## factor(ccode)GRC  4.218e+00  1.132e+00   3.726 0.000195 ***
## factor(ccode)GRD -1.094e+01  1.132e+00  -9.667  < 2e-16 ***
## factor(ccode)GTM  3.273e+00  1.132e+00   2.891 0.003847 **
## factor(ccode)GUY -1.146e+00  1.132e+00  -1.012 0.311339
## factor(ccode)HND  2.977e+00  1.132e+00   2.630 0.008561 **
## factor(ccode)HRV -9.735e+00  1.132e+00  -8.600  < 2e-16 ***
## factor(ccode)HTI  2.892e+00  1.132e+00   2.555 0.010641 *
## factor(ccode)HUN -7.461e+00  1.132e+00  -6.590 4.64e-11 ***
## factor(ccode)IDN  3.920e+00  1.132e+00   3.462 0.000539 ***
## factor(ccode)IND  4.636e+00  1.134e+00   4.089 4.37e-05 ***
## factor(ccode)IRL -8.828e+00  1.132e+00  -7.797 7.00e-15 ***
## factor(ccode)IRN -4.078e+00  1.132e+00  -3.602 0.000317 ***
## factor(ccode)IRQ -6.355e+00  1.132e+00  -5.613 2.04e-08 ***
## factor(ccode)ISL -6.883e+00  1.132e+00  -6.080 1.24e-09 ***
## factor(ccode)ISR  5.006e+00  1.132e+00   4.422 9.87e-06 ***
## factor(ccode)ITA -2.812e+00  1.133e+00  -2.482 0.013093 *
## factor(ccode)JAM  3.898e-01  1.132e+00   0.344 0.730626
## factor(ccode)JOR  2.966e+00  1.132e+00   2.620 0.008819 **
## factor(ccode)JPN -6.014e+00  1.135e+00  -5.298 1.19e-07 ***
## factor(ccode)KAZ -9.581e+00  1.132e+00  -8.464  < 2e-16 ***
## factor(ccode)KEN  8.844e-01  1.132e+00   0.781 0.434765
## factor(ccode)KGZ -9.319e+00  1.132e+00  -8.232  < 2e-16 ***
## factor(ccode)KHM -2.346e+00  1.132e+00  -2.073 0.038240 *
## factor(ccode)KIR -1.161e+01  1.132e+00 -10.255  < 2e-16 ***
## factor(ccode)KOR  3.377e+00  1.132e+00   2.983 0.002864 **
## factor(ccode)KWT -1.241e+01  1.132e+00 -10.961  < 2e-16 ***
## factor(ccode)LAO -1.638e+00  1.132e+00  -1.447 0.147901
## factor(ccode)LBN  1.292e+00  1.132e+00   1.141 0.253926
## factor(ccode)LBR  2.615e+00  1.132e+00   2.310 0.020892 *
## factor(ccode)LBY -7.436e+00  1.132e+00  -6.569 5.34e-11 ***
## factor(ccode)LKA  1.006e+00  1.132e+00   0.888 0.374374
## factor(ccode)LSO -2.161e+00  1.132e+00  -1.909 0.056316 .
## factor(ccode)LTU -9.426e+00  1.132e+00  -8.327  < 2e-16 ***
## factor(ccode)LVA -9.716e+00  1.132e+00  -8.583  < 2e-16 ***
## factor(ccode)MAR  2.006e+00  1.132e+00   1.772 0.076453 .
## factor(ccode)MDA -9.291e+00  1.132e+00  -8.207 2.56e-16 ***
## factor(ccode)MDG -2.013e+00  1.132e+00  -1.778 0.075390 .
## factor(ccode)MDV -9.587e+00  1.132e+00  -8.469  < 2e-16 ***
## factor(ccode)MEX  2.115e+00  1.132e+00   1.868 0.061816 .
## factor(ccode)MKD -9.684e+00  1.132e+00  -8.554  < 2e-16 ***
## factor(ccode)MLI -1.015e+00  1.132e+00  -0.897 0.369992
## factor(ccode)MLT -5.133e+00  1.132e+00  -4.534 5.85e-06 ***
## factor(ccode)MMR -1.974e+00  1.132e+00  -1.744 0.081213 .
## factor(ccode)MNG -9.126e+00  1.132e+00  -8.062 8.42e-16 ***
## factor(ccode)MOZ -4.515e+00  1.132e+00  -3.988 6.70e-05 ***
## factor(ccode)MRT -2.757e+00  1.132e+00  -2.436 0.014886 *
## factor(ccode)MUS -3.490e+00  1.132e+00  -3.083 0.002055 **
## factor(ccode)MWI -1.032e+00  1.132e+00  -0.912 0.362035
## factor(ccode)MYS -1.516e+00  1.132e+00  -1.338 0.180787
## factor(ccode)NAM -9.203e+00  1.132e+00  -8.129 4.86e-16 ***
## factor(ccode)NER -1.248e+00  1.132e+00  -1.103 0.270238
## factor(ccode)NGA -1.935e+00  1.132e+00  -1.708 0.087621 .
## factor(ccode)NIC  1.401e+00  1.132e+00   1.237 0.216006
## factor(ccode)NLD -6.879e+00  1.133e+00  -6.072 1.31e-09 ***
## factor(ccode)NOR -6.666e+00  1.133e+00  -5.886 4.09e-09 ***
## factor(ccode)NPL  1.142e+00  1.132e+00   1.008 0.313307
## factor(ccode)NZL -1.178e+01  1.132e+00 -10.408  < 2e-16 ***
## factor(ccode)OMN -5.300e+00  1.132e+00  -4.682 2.88e-06 ***
## factor(ccode)PAK  3.671e+00  1.133e+00   3.240 0.001201 **
## factor(ccode)PAN  2.235e+00  1.133e+00   1.973 0.048478 *
## factor(ccode)PER  3.741e+00  1.132e+00   3.304 0.000958 ***
## factor(ccode)PHL  4.952e+00  1.132e+00   4.374 1.23e-05 ***
## factor(ccode)PNG -6.646e+00  1.132e+00  -5.871 4.47e-09 ***
## factor(ccode)POL -2.230e+00  1.133e+00  -1.968 0.049081 *
## factor(ccode)PRK -1.105e+01  1.132e+00  -9.758  < 2e-16 ***
## factor(ccode)PRT  2.123e+00  1.132e+00   1.875 0.060803 .
## factor(ccode)PRY  1.839e+00  1.132e+00   1.625 0.104224
## factor(ccode)ROM -7.631e+00  1.132e+00  -6.739 1.68e-11 ***
## factor(ccode)RUS -9.228e+00  1.132e+00  -8.152 4.04e-16 ***
## factor(ccode)RWA -2.052e+00  1.132e+00  -1.813 0.069856 .
## factor(ccode)SAU -6.708e+00  1.132e+00  -5.926 3.22e-09 ***
## factor(ccode)SDN -1.685e-01  1.132e+00  -0.149 0.881702
## factor(ccode)SEN -9.187e-01  1.132e+00  -0.811 0.417164
## factor(ccode)SGP -7.638e+00  1.132e+00  -6.748 1.59e-11 ***
## factor(ccode)SLB -1.163e+01  1.132e+00 -10.272  < 2e-16 ***
## factor(ccode)SLE -8.293e-01  1.132e+00  -0.733 0.463842
## factor(ccode)SLV  3.052e+00  1.132e+00   2.696 0.007032 **
## factor(ccode)SOM -2.500e-01  1.132e+00  -0.221 0.825210
## factor(ccode)STP -6.893e+00  1.132e+00  -6.089 1.18e-09 ***
## factor(ccode)SUR -6.398e+00  1.132e+00  -5.652 1.63e-08 ***
## factor(ccode)SVK -9.851e+00  1.132e+00  -8.702  < 2e-16 ***
## factor(ccode)SVN -1.010e+01  1.132e+00  -8.926  < 2e-16 ***
## factor(ccode)SWE -1.148e+01  1.132e+00 -10.140  < 2e-16 ***
## factor(ccode)SWZ -3.729e+00  1.132e+00  -3.294 0.000991 ***
## factor(ccode)SYC -5.679e+00  1.132e+00  -5.017 5.34e-07 ***
## factor(ccode)SYR -5.963e+00  1.132e+00  -5.267 1.42e-07 ***
## factor(ccode)TCD -2.035e+00  1.132e+00  -1.798 0.072271 .
## factor(ccode)TGO -1.525e+00  1.132e+00  -1.347 0.177984
## factor(ccode)THA  3.361e+00  1.132e+00   2.969 0.002996 **
## factor(ccode)TJK -9.408e+00  1.132e+00  -8.311  < 2e-16 ***
## factor(ccode)TKM -9.741e+00  1.132e+00  -8.605  < 2e-16 ***
## factor(ccode)TON -1.160e+01  1.132e+00 -10.248  < 2e-16 ***
## factor(ccode)TTO -5.420e+00  1.132e+00  -4.788 1.71e-06 ***
## factor(ccode)TUN  1.346e+00  1.132e+00   1.189 0.234557
## factor(ccode)TUR  5.278e+00  1.132e+00   4.661 3.18e-06 ***
## factor(ccode)TZA -4.805e-01  1.132e+00  -0.424 0.671264
## factor(ccode)UGA -6.875e-01  1.132e+00  -0.607 0.543713
## factor(ccode)UKR -9.520e+00  1.132e+00  -8.406  < 2e-16 ***
## factor(ccode)URY  8.177e-01  1.132e+00   0.722 0.470120
## factor(ccode)UZB -9.710e+00  1.132e+00  -8.578  < 2e-16 ***
## factor(ccode)VEN  6.677e-04  1.133e+00   0.001 0.999530
## factor(ccode)VNM -9.965e+00  1.132e+00  -8.803  < 2e-16 ***
## factor(ccode)VUT -1.159e+01  1.132e+00 -10.242  < 2e-16 ***
## factor(ccode)WBG -8.142e+00  1.132e+00  -7.193 6.82e-13 ***
## factor(ccode)WSM -4.708e+00  1.132e+00  -4.159 3.22e-05 ***
## factor(ccode)YEM -1.809e+00  1.132e+00  -1.598 0.110082
## factor(ccode)ZAF -7.085e+00  1.132e+00  -6.259 4.04e-10 ***
## factor(ccode)ZAR -1.369e+00  1.132e+00  -1.209 0.226700
## factor(ccode)ZMB -1.339e+00  1.132e+00  -1.183 0.237003
## factor(ccode)ZWE -4.239e+00  1.132e+00  -3.744 0.000182 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.99 on 9569 degrees of freedom
##   (2916 observations deleted due to missingness)
## Multiple R-squared:  0.4165, Adjusted R-squared:  0.4058
## F-statistic: 39.25 on 174 and 9569 DF,  p-value: < 2.2e-16
```

```
##This is an eyesore.  We can use plm
library(plm)
```

```
##
## Attaching package: 'plm'

## The following object is masked from 'package:data.table':
##
##     between

## The following objects are masked from 'package:dplyr':
##
##     between, lag, lead
```

```
FEmodel2 <- plm(ln_totaid96~scmem,
                model="within", ##FE command
                index=c("ccode", "year"), ##panel vars
                data=UNSC)
summary(FEmodel2)
```

```
## Oneway (individual) effect Within Model
##
## Call:
## plm(formula = ln_totaid96 ~ scmem, data = UNSC, model = "within",
##     index = c("ccode", "year"))
##
## Balanced Panel: n = 174, T = 56, N = 9744
##
## Residuals:
##      Min.   1st Qu.    Median   3rd Qu.      Max.
## -18.47996  -3.19657  -0.29027   3.99217  17.88267
##
## Coefficients:
##       Estimate Std. Error t-value  Pr(>|t|)
## scmem  1.48864    0.29527  5.0416 4.702e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Total Sum of Squares:    344250
## Residual Sum of Squares: 343340
## R-Squared:      0.0026492
## Adj. R-Squared: -0.015486
## F-statistic: 25.4175 on 1 and 9569 DF, p-value: 4.7018e-07
```

```
##Random effects
REmodel1 <- plm(ln_totaid96~scmem,
                model="random", ##RE command
                index=c("ccode", "year"), ##panel vars
                data=UNSC)

summary(REmodel1)
```

```
## Oneway (individual) effect Random Effect Model
##    (Swamy-Arora's transformation)
##
## Call:
## plm(formula = ln_totaid96 ~ scmem, data = UNSC, model = "random",
##     index = c("ccode", "year"))
##
## Balanced Panel: n = 174, T = 56, N = 9744
##
## Effects:
##                  var std.dev share
## idiosyncratic 35.880   5.990 0.622
## individual    21.827   4.672 0.378
## theta: 0.8311
##
## Residuals:
##      Min.   1st Qu.    Median   3rd Qu.      Max.
## -17.03358  -3.97170  -0.16341   4.50659  17.03889
##
## Coefficients:
##             Estimate Std. Error z-value  Pr(>|z|)
## (Intercept)  8.12455    0.35997 22.5703 < 2.2e-16 ***
## scmem        1.55061    0.29523  5.2522 1.503e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Total Sum of Squares:    351210
## Residual Sum of Squares: 350220
## R-Squared:      0.0028237
## Adj. R-Squared: 0.0027213
## Chisq: 27.5859 on 1 DF, p-value: 1.5027e-07
```

```
##Clustered errors

#Fixed effects model SE clustered on country
clust <- vcovHC(FEmodel2, cluster="group")
coeftest(FEmodel2, clust)
```

```
##
## t test of coefficients:
##
##       Estimate Std. Error t value  Pr(>|t|)
## scmem  1.48864    0.35206  4.2284 2.376e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#Or time
clust2 <- vcovHC(FEmodel2, cluster="time")
coeftest(FEmodel2, clust)
```

```
##
## t test of coefficients:
##
##       Estimate Std. Error t value  Pr(>|t|)
## scmem  1.48864    0.35206  4.2284 2.376e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#OR if you have an lm fit you can use sandwich::vcovCL
pooled.model1 <- lm(ln_totaid96~scmem, data=UNSC)
clust.pooled <- vcovCL(pooled.model1, cluster=UNSC$ccode)
```

```
coeftest(pooled.model1, clust)
```

```
##
## t test of coefficients:
##
##       Estimate Std. Error t value  Pr(>|t|)
## scmem  3.50864    0.35206  9.9661 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##Can use LinearHypothesis with this new Variance matrix
linearHypothesis(FEmodel2, "scmem=0", vcov=clust)
```

```
## Linear hypothesis test
##
## Hypothesis:
## scmem = 0
##
## Model 1: restricted model
## Model 2: ln_totaid96 ~ scmem
##
## Note: Coefficient covariance matrix supplied.
##
##   Res.Df Df  Chisq Pr(>Chisq)
## 1   9570
## 2   9569  1 17.879  2.354e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
linearHypothesis(pooled.model1, "scmem=0", vcov=clust.pooled)
```

```
## Linear hypothesis test
##
## Hypothesis:
## scmem = 0
##
## Model 1: restricted model
## Model 2: ln_totaid96 ~ scmem
```

```
##
## Note: Coefficient covariance matrix supplied.
##
##   Res.Df Df    F    Pr(>F)
## 1   9743
## 2   9742  1 33.1 9.018e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

5.2.3 A smorgasbord of Regression Tests and Statistics

Additional regression tests and statistics are listed here as a reference

Table 3: Additional Tools and Tests

Function	Use	Package
`t.test`	Test if two samples drawn from normals with the same mean	`stats`
`wilcox.test`	Test if two samples drawn from distribution with same mean	`stats`
`chisq.test`	Pearson's chi square test for independence or goodness-of-fit	`stats`
`fisher.test`	Fisher's exact test for independence in 2x2 tables	`stats`
`ks.test`	Kolmogorov-Smirnov test for comparing distributions	`stats`
`aic`	Returns the AIC of a model	`stats`
`bic`	Returns the BIC of a model	`stats`
`vuongtest`	Voung's test for (non-nested) model comparison	`nonnest2`
`lrtest`	Likelihood ratio test for (nested) model comparison	`lmtest`
`linearHypothesis`	Wald test for (nested) model comparison	`car`
`Box.test`	Box-Pierce & Ljung-Box tests for autocorrelation	`stats`
`bptest`	Breusch-Pagan test for heteroskedasticity	`lmtest`
`dwtest`	Durbin-Watson test for autocorrelation in Errors	`lmtest`
`shapiro.test`	Shapiro-Wilk test for normality	`stats`

5.3 GLMs

GLMs in R are pretty straight forward and can be estimated using the `glm` function. These are models that you'll get to in 603, so for now just note that they're here if you need them.

```
model11 <- glm(onset ~ ethfrac + relfrac,
              family=binomial, ##specify logit
              data=FearonLaitin)
summary(model11)
```

```
##
## Call:
## glm(formula = onset ~ ethfrac + relfrac, family = binomial, data = FearonLaitin)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.2399  -0.2087  -0.1732  -0.1526   3.0337
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -4.5050     0.2214 -20.343  < 2e-16 ***
## ethfrac       1.1608     0.3571   3.251  0.00115 **
## relfrac      -0.1703     0.4677  -0.364  0.71587
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1127.4  on 6609  degrees of freedom
## Residual deviance: 1116.0  on 6607  degrees of freedom
## AIC: 1122
##
## Number of Fisher Scoring iterations: 7
```

```
names(model11)
```

```
##  [1] "coefficients"      "residuals"         "fitted.values"
##  [4] "effects"           "R"                 "rank"
##  [7] "qr"                "family"            "linear.predictors"
## [10] "deviance"          "aic"               "null.deviance"
## [13] "iter"              "weights"           "prior.weights"
## [16] "df.residual"       "df.null"           "y"
## [19] "converged"         "boundary"          "model"
## [22] "call"              "formula"           "terms"
## [25] "data"                "offset"            "control"
## [28] "method"              "contrasts"         "xlevels"
```

```
##Everything from lm carries over

model12 <- glm(onset ~ ethfrac + relfrac + log(gdpen),
              family=binomial,
              data=FearonLaitin)
model13 <- glm(onset ~ ethfrac + relfrac+log(gdpen) + pop + I(pop^2),
              data=FearonLaitin,
              family=binomial, ##specify logit
              x=TRUE, y=TRUE)

##Subsetting
model14 <- glm(onset ~ ethfrac + relfrac+log(gdpen),
              data=FearonLaitin,
              family=binomial, ##specify logit
              x=TRUE, y=TRUE,
              subset=!is.na(pop))

##Check the fitted values
##plogis is the CDF of the logistic distribution
summary(cbind(plogis(model13$x %*% model13$coef),
              model13$fitted.values))
```

```
##          V1                  V2
##  Min.   :0.001009   Min.   :0.001009
##  1st Qu.:0.007351   1st Qu.:0.007351
##  Median :0.013030   Median :0.013030
##  Mean   :0.016792   Mean   :0.016792
##  3rd Qu.:0.022045   3rd Qu.:0.022045
##  Max.   :0.152631   Max.   :0.152631
```

```
##Note that glms also have an option for convergence
model14$converged
```

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

GLMs can take any of the following models\[12pt]

Model	family=
logit	`binomial`
probit	`binomial(link="probit")`
cloglog	`binomial(link="cloglog")`
OLS	`gaussian`
Poisson	`poisson`
gamma	`Gamma`

5.4 Tabling Results

Now that we have some models we may want to put them in tables so that we can put them in a paper. All of the functions that I know for this are convert R output into either HTML or LaTeX. I don't know of an easy way to convert either to Word. We'll look at 2 packages to convert R to LaTeX, although many others exist.

5.4.1 `xtable`

The first tabling package we'll look at is `xtable`.

```
library(xtable)
xtable(model1)
```

```
## % latex table generated in R 4.2.1 by xtable 1.8-4 package
## % Tue Aug 16 09:37:34 2022
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrr}
##   \hline
##  & Estimate & Std. Error & t value & Pr($>$$|$t$|$) \\
##   \hline
## (Intercept) & 10.0455 & 1.2567 & 7.99 & 0.0000 \\
##   Male & 4.5012 & 0.8375 & 5.37 & 0.0000 \\
##   Age & 0.0676 & 0.0293 & 2.30 & 0.0214 \\
##    \hline
## \end{tabular}
## \end{table}
```

Which in LaTeX looks like

```
xtable(model1)
```

% latex table generated in R 4.2.1 by xtable 1.8-4 package % Tue Aug 16 09:37:34 2022

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	10.0455	1.2567	7.99	0.0000
Male	4.5012	0.8375	5.37	0.0000
Age	0.0676	0.0293	2.30	0.0214

Now we can customize this using built in arguments and using the `print` command.

```
##Give the variables nice names
names(model1$coefficients) <- c("Intercept",
                                "Ethnic Frac.",
                                "Religous Frac.")


print(xtable(model1,
             caption="Fearon and Laitin Logit",
             label="tab:FL1",
             align=c("rcccc"),
             digits=2),
      caption.placement="top",
      table.placement="htb")
```

```
## % latex table generated in R 4.2.1 by xtable 1.8-4 package
## % Tue Aug 16 09:37:34 2022
## \begin{table}[htb]
## \centering
## \caption{Fearon and Laitin Logit}
## \label{tab:FL1}
## \begin{tabular}{rcccc}
##   \hline
##  & Estimate & Std. Error & t value & Pr($>$$|$$t$$|$) \\
##   \hline
## Intercept & 10.05 & 1.26 & 7.99 & 0.00 \\
##   Ethnic Frac. & 4.50 & 0.84 & 5.37 & 0.00 \\
##   Religous Frac. & 0.07 & 0.03 & 2.30 & 0.02 \\
##    \hline
## \end{tabular}
## \end{table}
```

```
####Print model 1 using robust standard errors
robustModel1 <- coeftest(model1, vcovHC)
robustModel1 <- coeftest(model1, vcovHC)[]

##For quirky reasons only the command with [] can be
##given to xtable
##Note that using [] extracts just the matrix part of it

xtable(robustModel1)
```

```
## % latex table generated in R 4.2.1 by xtable 1.8-4 package
## % Tue Aug 16 09:37:34 2022
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrr}
##   \hline
##  & Estimate & Std. Error & t value & Pr($>$$|$t$|$) \\
##   \hline
## Intercept & 10.05 & 1.09 & 9.19 & 0.00 \\
##   Ethnic Frac. & 4.50 & 0.64 & 6.99 & 0.00 \\
##   Religous Frac. & 0.07 & 0.03 & 2.32 & 0.02 \\
##    \hline
## \end{tabular}
## \end{table}
```

MANY options exist to customize `xtable` output, and we could spend hours going over just different commands within `xtable`. Take a look at help file by running `?print.xtable` and go to http://cran.r-project.org/web/packages/xtable/vignettes/xtableGallery.pdf to see examples (with code) of customized `xtable` output.

5.4.2 stargazer

There are a lot of times when we would prefer to view models side by side. For this I like `stargazer::stargazer` although `modelsummary` is becoming more popular. Unlike `xtable` it works on a lot of canned models directly (to varying degrees of effectiveness so watch out).

```r
library(stargazer)
```

```
##
## Please cite as:
##  Hlavac, Marek (2022). stargazer: Well-Formatted Regression and Summary Statistics Ta
##  R package version 5.2.3. https://CRAN.R-project.org/package=stargazer
```

```r
stargazer(model2, model3, FEmodel2,
          title="Trying Stargazer",
          label="tab:Star",
          ##It allows you to put nice names
          ##directly into the call
          covariate.labels=c("Ethnic Frac.",
                             "Religious Frac.",
                             "log(GDP per capita)",
                             "Population",
                             "Population$^2$",
                             "UNSC Member"),
          ##nice names for the Dependent
          ##Variables
          dep.var.labels=c("Onset", "Foreign Aid"),
          digits=2)
```

```
##
## % Table created by stargazer v.5.2.3 by Marek Hlavac, Social Policy Institute. E-mail
## % Date and time: Tue, Aug 16, 2022 - 09:37:34 AM
## \begin{table}[!htbp] \centering
##   \caption{Trying Stargazer}
##   \label{tab:Star}
## \begin{tabular}{@{\extracolsep{5pt}}lccc}
## \\[-1.8ex]\hline
## \hline \\[-1.8ex]
##  & \multicolumn{3}{c}{\textit{Dependent variable:}} \\
## \cline{2-4}
## \\[-1.8ex] & Onset & Foreign Aid & ln\_totaid96 \\
## \\[-1.8ex] & \textit{OLS} & \textit{OLS} & \textit{panel} \\
##  & \textit{} & \textit{} & \textit{linear} \\
## \\[-1.8ex] & (1) & (2) & (3)\\
## \hline \\[-1.8ex]
##  Ethnic Frac. &  & 0.24$^{***}$ &  \\
##   &  & (0.04) &  \\
##   & & & \\
##  Religious Frac. & 0.06$^{*}$ & 0.04$^{***}$ &  \\
##   & (0.04) & (0.01) &  \\
##   & & & \\
##  log(GDP per capita) &  & $-$0.0005$^{***}$ &  \\
##   &  & (0.0001) &  \\
##   & & & \\
##  Population &  &  & 1.49$^{***}$ \\
##   &  &  & (0.30) \\
##   & & & \\
##  Population$^2$ & 14.71$^{***}$ & 1.52$^{***}$ &  \\
##   & (1.62) & (0.16) &  \\
##   & & & \\
## \hline \\[-1.8ex]
## Observations & 716 & 974 & 9,744 \\
## R$^{2}$ & 0.004 & 0.08 & 0.003 \\
## Adjusted R$^{2}$ & 0.002 & 0.07 & $-$0.02 \\
## Residual Std. Error & 12.61 (df = 714) & 0.56 (df = 970) &  \\
## F Statistic & 2.78$^{*}$ (df = 1; 714) & 26.26$^{***}$ (df = 3; 970) & 25.42$^{***}$
## \hline
## \hline \\[-1.8ex]
## \textit{Note:}  & \multicolumn{3}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05; $^{***}$p$<$0.01
## \end{tabular}
## \end{table}
```

There is lots of customization available for `stargazer` output as well, which can all be found in the help file for the function.

5.5 Other Models and Where to Find Them

Table 4: Other Common Models and their Packages

Model	Package
Conditional Logit	`survival`
Weibull, Exponential, Cox PH	`survival`
Ordered Probit	`MASS`
Negative Binomial	`MASS`
Multinomial Logit	`VGAM`
Heckman Selection	`sampleSelection`
GAMS	`mgcv`
ARIMA models	`stats`
Instrumental Regression	`ivreg`
tobit	`AER`
Bayesian GLM	`brm`

Other models can be estimated by maximizing a likelihood (using `optim`) or by Bayesian methods using something like Stan.

5.6 Plotting Results from Regression Models

While these summary plots from the last chapter are interesting on their own, the real reason you want to learn to plot is to plot regression results. So we'll look at some data on GRE scores and admission into grad school to look at some interesting ways to represent regression results graphically.

```
##Data
GREdat <- read.csv("https://stats.idre.ucla.edu/stat/data/binary.csv")
colnames(GREdat)
```

```
## [1] "admit" "gre"   "gpa"   "rank"
```

Suppose then we want to predict admission to grad school using GRE scores and the ranking of the candidate's undergraduate institution (4 pt scale). As I'm sure you can guess, that's exactly what's in the data we just read in. Our model then is

```
admitMod <- glm(admit ~ gre  + factor(rank),
                data=GREdat,##data
                family=binomial, ##
                x=TRUE) ##We'll want the X matrix

summary(admitMod)
```

```
##
## Call:
## glm(formula = admit ~ gre + factor(rank), family = binomial,
##     data = GREdat, x = TRUE)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5199  -0.8715  -0.6588   1.1775   2.1113
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -1.802365   0.672982  -2.678 0.007402 **
## gre            0.003224   0.001019   3.163 0.001562 **
## factor(rank)2 -0.721737   0.313033  -2.306 0.021132 *
## factor(rank)3 -1.291305   0.340775  -3.789 0.000151 ***
## factor(rank)4 -1.602054   0.414932  -3.861 0.000113 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 499.98  on 399  degrees of freedom
## Residual deviance: 464.53  on 395  degrees of freedom
## AIC: 474.53
##
## Number of Fisher Scoring iterations: 4
```

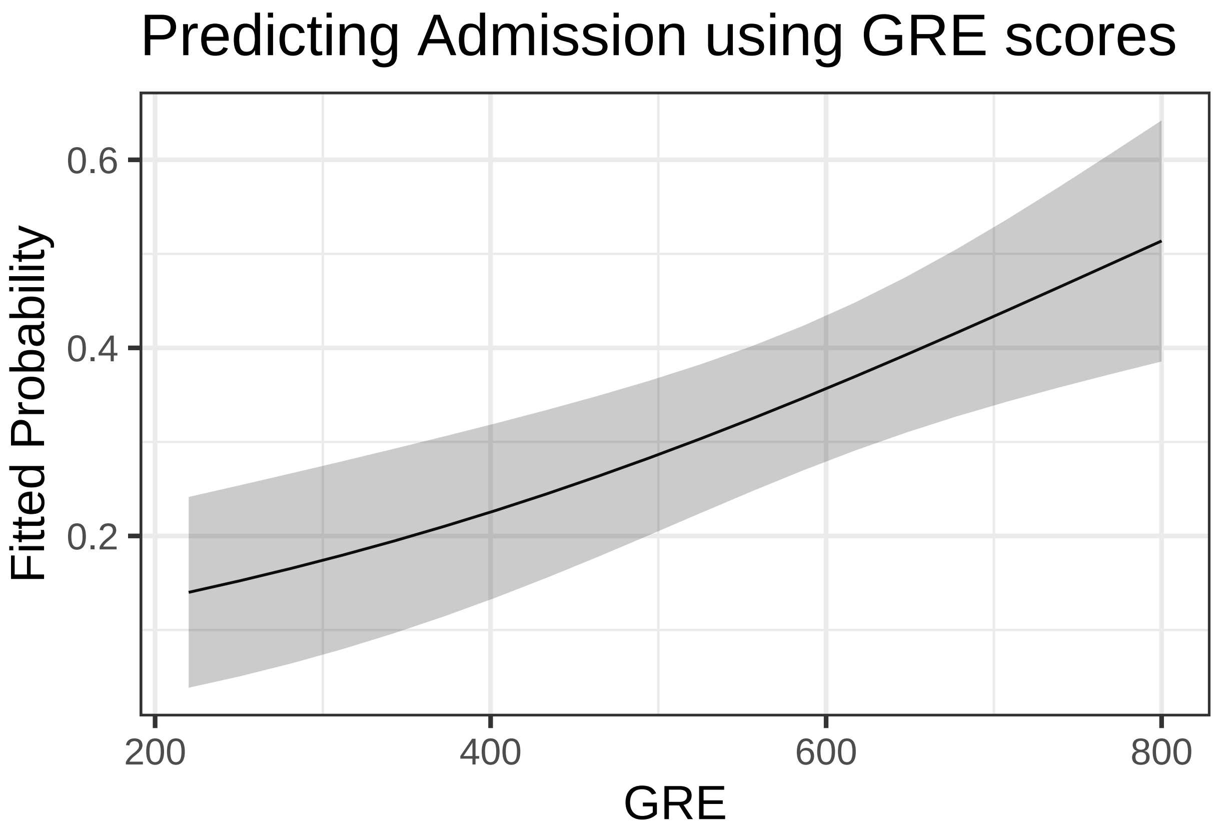
5.6.1 Continuous Variables

Let's say we're first interesting in the effect that GRE score has on admission. We first want to hold all the other variables constant.

```
library(ggplot2)
## A median student is from a rank 2 school
# Vary gre score for predictions
newdata <- data.frame(gre= seq(min(GREdat$gre),
                               max(GREdat$gre),
                               length.out=20),
                      rank=median(GREdat$rank))
predictions <- predict(admitMod, newdata=newdata, type="response", se.fit=TRUE)

plottingData <- data.frame(fit = predictions$fit,
                           hi = predictions$fit+1.96*predictions$se.fit,
                           lo = predictions$fit-1.96*predictions$se.fit,
                           GRE = newdata$gre)
plottingData
```

```
##          fit        hi         lo      GRE
## 1  0.1400540 0.2414980 0.03861010 220.0000
## 2  0.1523319 0.2539260 0.05073785 250.5263
## 3  0.1654791 0.2665845 0.06437358 281.0526
## 4  0.1795206 0.2794716 0.07956954 311.5789
## 5  0.1944759 0.2926035 0.09634824 342.1053
## 6  0.2103576 0.3060239 0.11469144 372.6316
## 7  0.2271706 0.3198148 0.13452647 403.1579
## 8  0.2449106 0.3341120 0.15570925 433.6842
## 9  0.2635635 0.3491224 0.17800458 464.2105
## 10 0.2831044 0.3651419 0.20106677 494.7368
## 11 0.3034969 0.3825644 0.22442947 525.2632
## 12 0.3246931 0.4018632 0.24752306 555.7895
## 13 0.3466330 0.4235228 0.26974312 586.3158
## 14 0.3692447 0.4479149 0.29057446 616.8421
## 15 0.3924454 0.4751676 0.30972326 647.3684
## 16 0.4161422 0.5051098 0.32717464 677.8947
## 17 0.4402330 0.5373239 0.34314217 708.4211
## 18 0.4646087 0.5712570 0.35796048 738.9474
## 19 0.4891547 0.6063188 0.37199054 769.4737
## 20 0.5137530 0.6419405 0.38556547 800.0000
```

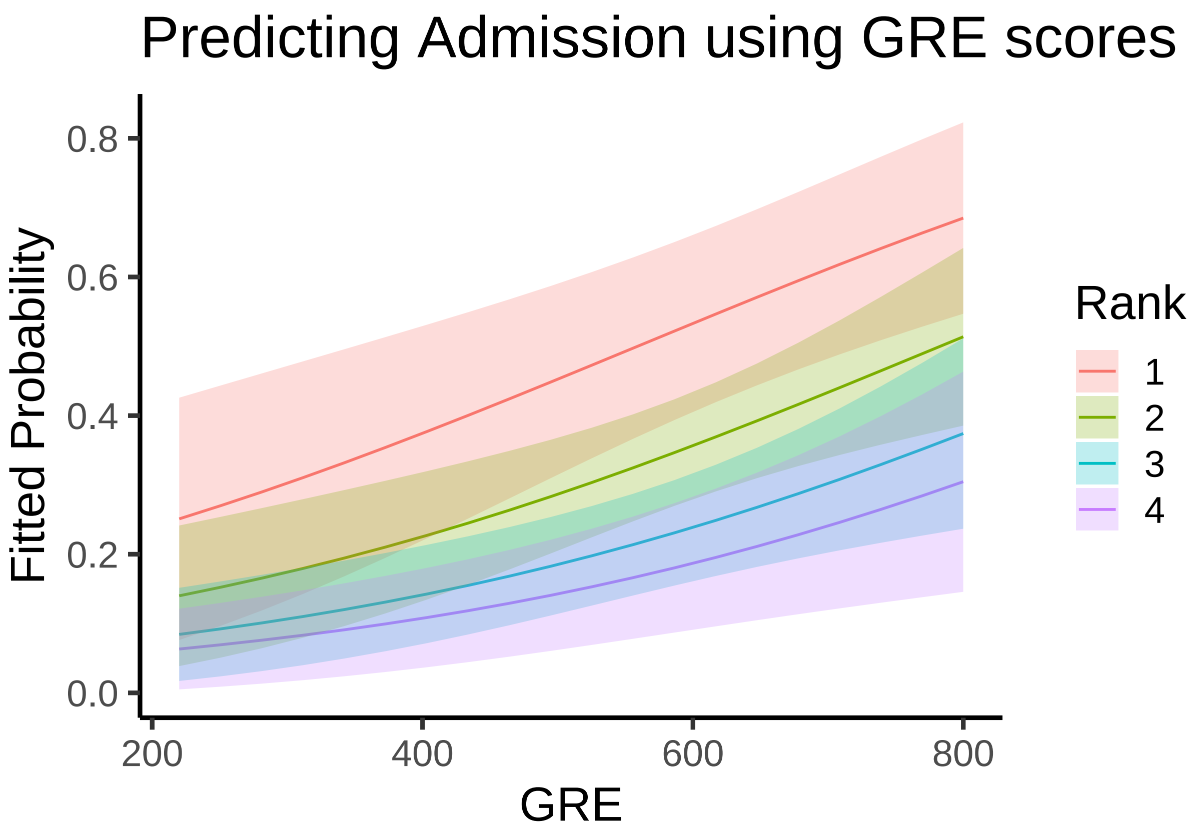
```
########plot it ########
grePlot <- ggplot(plottingData)+
              geom_line(aes(x=GRE, y=fit))+
              geom_ribbon(aes(x=GRE, ymin=lo, max=hi), alpha=0.25)+
              ylab("Fitted Probability")+
              xlab("GRE")+
              ggtitle("Predicting Admission using GRE scores")+
              theme_bw(18)
print(grePlot)
```



We can also examine how our GRE predictions change with rank

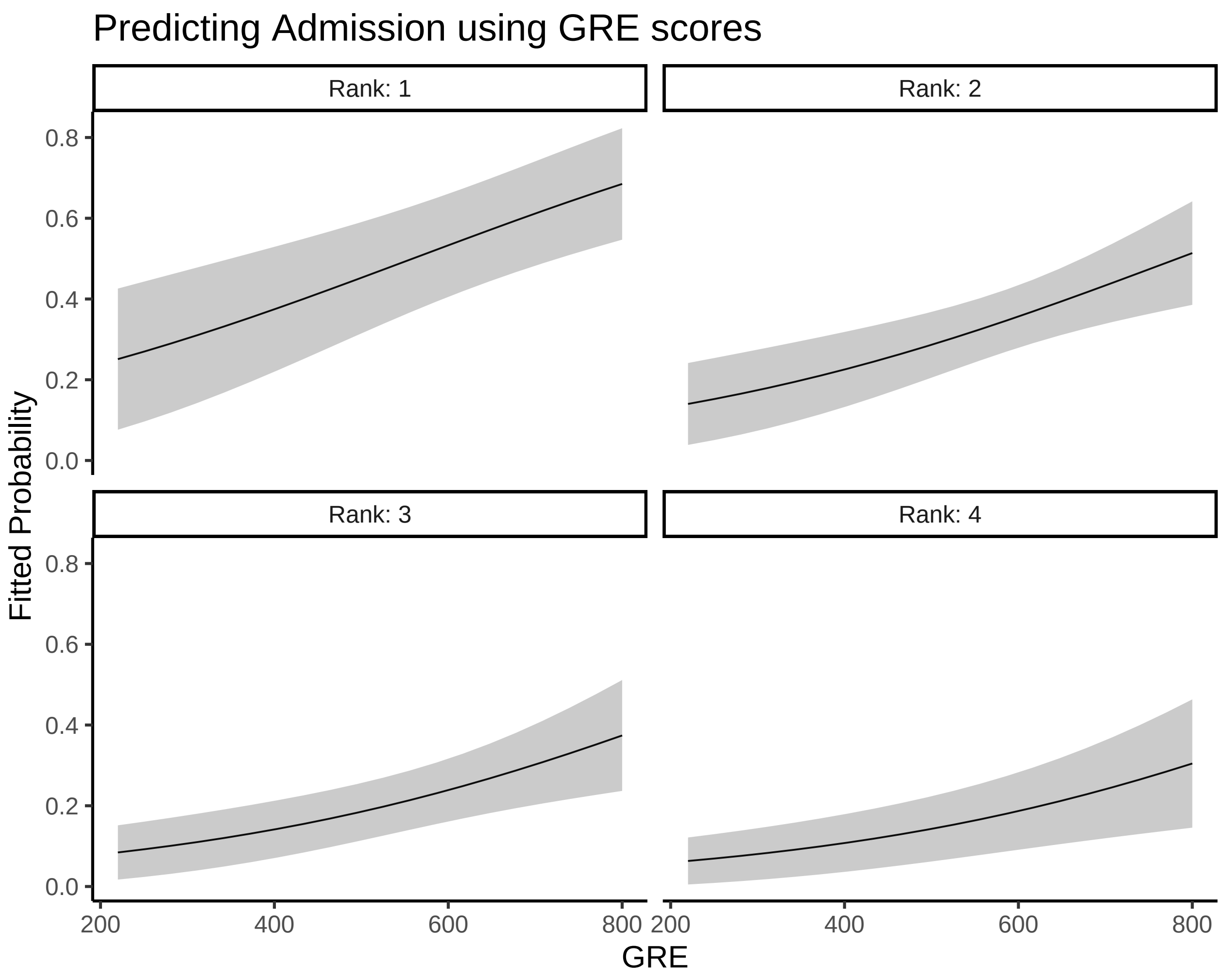
```
newdata <- data.frame(gre= seq(min(GREdat$gre),
                               max(GREdat$gre),
                               length.out=20),
                      rank=rep(1:4, each=20))
predictions <- predict(admitMod, newdata=newdata, type="response", se.fit=TRUE)

plottingData <- data.frame(fit = predictions$fit,
```

```r
                           hi = predictions$fit+1.96*predictions$se.fit,
                           lo = predictions$fit-1.96*predictions$se.fit,
                           GRE = newdata$gre,
                           Rank = factor(newdata$rank))




grePlot2 <- ggplot(plottingData)+
              geom_line(aes(x=GRE,
                            y=fit,
                            color=Rank))+
              geom_ribbon(aes(x=GRE,
                              ymin=lo,
                              max=hi,
                              fill=Rank), alpha=0.25)+
              ylab("Fitted Probability")+
              xlab("GRE")+
              ggtitle("Predicting Admission using GRE scores")+
              theme_classic(18)
print(grePlot2)
```



But we may say that there's too much overlap and we could split it into separate plots. Either by creating four different plots or by using the facet option

```r
##Give better labels
levels(plottingData$Rank) <- c("Rank: 1",
                               "Rank: 2",
                               "Rank: 3",
                               "Rank: 4")


ggplot(plottingData)+
              geom_line(aes(x=GRE,
                            y=fit))+
              geom_ribbon(aes(x=GRE,
                              ymin=lo,
                              max=hi), alpha=0.25)+
              facet_wrap(~Rank, ncol=2)+ ##Can specify columns
              ylab("Fitted Probability")+
              xlab("GRE")+
              ggtitle("Predicting Admission using GRE scores")+
              theme_classic(18)
```



5.6.2 Categorical Variables

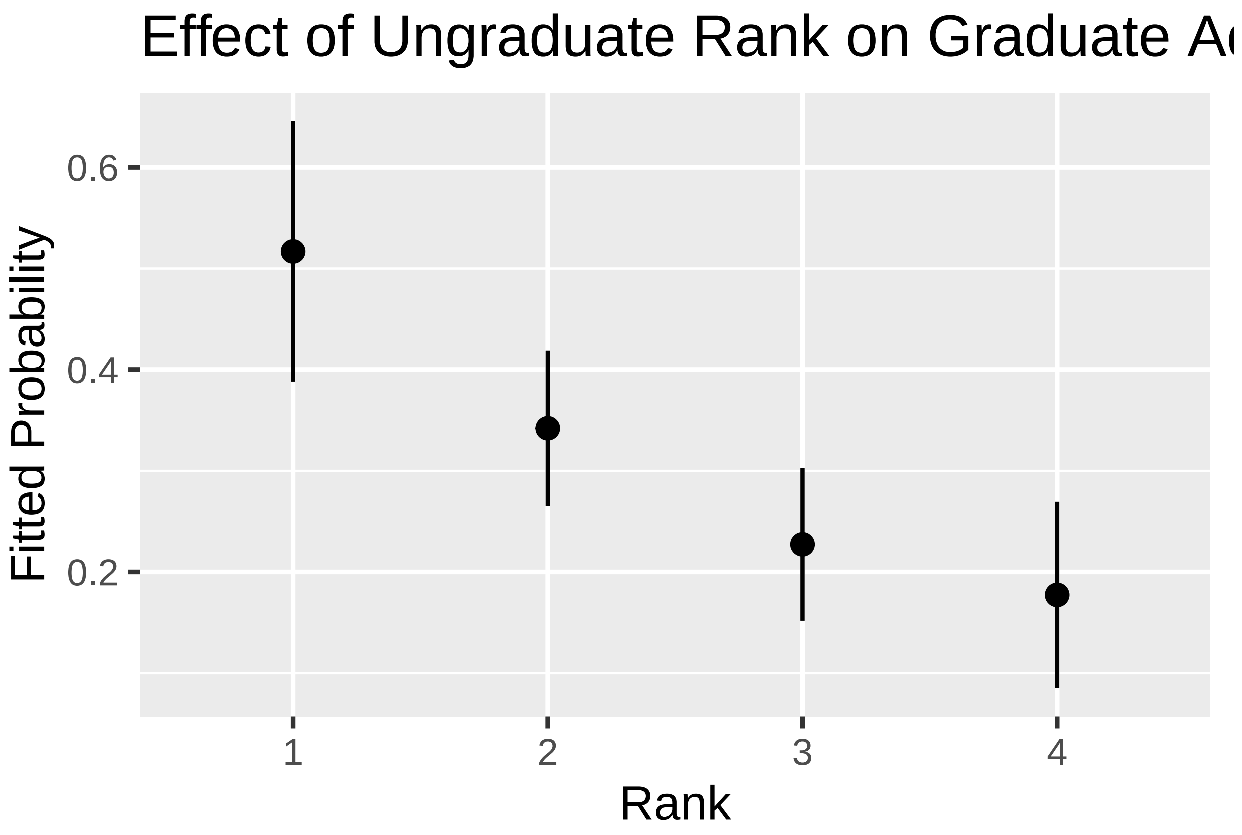
However suppose we wanted to predict based on rank, holding GRE constant.

```
newdata <- data.frame(gre= median(GREdat$gre),
                      rank=rep(1:4))
predictions <- predict(admitMod, newdata=newdata, type="response", se.fit=TRUE)

plottingData <- data.frame(fit = predictions$fit,
                           hi = predictions$fit+1.96*predictions$se.fit,
                           lo = predictions$fit-1.96*predictions$se.fit,
                           GRE = newdata$gre,
                           Rank = factor(newdata$rank))
```

```
plottingData
```

```
##         fit        hi         lo GRE Rank
## 1 0.5168693 0.6456540 0.38808454 580    1
## 2 0.3420360 0.4188267 0.26524533 580    2
## 3 0.2272680 0.3026981 0.15183787 580    3
## 4 0.1773288 0.2695111 0.08514646 580    4
```

```
rankPlot <- ggplot(plottingData)+
              geom_pointrange(aes(x=Rank,
                                  y=fit,
                                  ymin=lo,
                                  ymax=hi),
                              size=0.75)+
              theme_grey(18)+
              ylab('Fitted Probability')+
              ggtitle('Effect of Ungraduate Rank on Graduate Admission')
print(rankPlot)
```



A Answers to exeriences

A.1 Answers to chapter 1 exercises

```
## Solution to problem 1.1
?rnorm
```

```
## Solution to problem 1.2
X <- cbind(1, rnorm(1000, mean=1, sd=2), runif(1000))
colMeans(X)
```

```
## [1] 1.0000000 0.9873285 0.4888034
```

```
apply(X, 2, sd)
```

```
## [1] 0.0000000 1.9563893 0.2879309
```

```
b <- c(-1,2,2)
b[2] <- -2

y <- X%*% b + rnorm(1000)
```

```
## Solutions to problem 1.3
install.packages(c("readstata13", "data.table", "MASS",
                   "tidyr", "dplyr", "ggplot2", "gridExtra",
                   "lmtest", "car", "sandwich", "plm",
                   "xtable", "stargazer"))
```

A.2 Answers to chapter 2 exercises

```
## Solution to problem 2.1
my.max <- function(x){
  maximum <- x[1]
  for(i in 2:length(x)){
    if(x[i] > maximum){
      maximum <- x[i]
    }
  }
  return(maximum)
}
```

```
## Solution to problem 2.2
# install.packages("geoR")
library(geoR)
```

```
## --------------------------------------------------------------
##  Analysis of Geostatistical Data
##  For an Introduction to geoR go to http://www.leg.ufpr.br/geoR
##  geoR version 1.9-2 (built on 2022-08-09) is now loaded
## --------------------------------------------------------------
```

```
library(MASS)
N <- 2000
X <- cbind(1, rnorm(N), rnorm(N))
b <- c(-1,2,-2)
e <- rnorm(N, mean=0, sd=2)

y <- X %*% b + e

MCout <- matrix(0, 10000, 4)
beta.i <- runif(3, -100000, 100000)
mu <- solve(t(X) %*% X) %*% t(X) %*% y
for(i in 1:10000){
  b.input <- beta.i
  s2 <- sum( (y-X%*%b.input)^2)/(2000-3)
  sigma2.new <- rinvchisq(1,df=2000-3, scale= s2)
  V <- sigma2.new*solve(t(X) %*% X)
  beta.i <- mvrnorm(1, mu, V)
  MCout[i,] <- c(beta.i, sigma2.new)
}

MCout <- MCout[5001:10000,]
colMeans(MCout) #estimates
```

```
## [1] -0.9899791  1.9313443 -2.0016070  4.0383458
```

```
# Solution to problem 2.3
ols <- function(y,X){
  N <- nrow(X)
  k <- ncol(X)
  b.hat <- solve(crossprod(X)) %*% crossprod(X,y)
  s2 <- sum( (y-X%*%b.hat)^2)/(N-k)
  var.beta <- s2*solve(crossprod(X))
  output <- list(est = b.hat,
                 se = sqrt(diag(var.beta)),
                 Var = var.beta)
  return(output)
}
ols(y, X)
```

```
## $est
##            [,1]
## [1,] -0.9900163
## [2,]  1.9303936
## [3,] -2.0016281
##
## $se
## [1] 0.04489838 0.04498747 0.04409603
##
## $Var
##              [,1]          [,2]          [,3]
## [1,] 2.015865e-03  1.174113e-05  6.783560e-05
## [2,] 1.174113e-05  2.023872e-03 -3.459786e-06
## [3,] 6.783560e-05 -3.459786e-06  1.944460e-03
```

```
ols <- function(y,X){
  N <- nrow(X)
  k <- ncol(X)
  b.hat <- solve(crossprod(X)) %*% crossprod(X,y)
  s2 <- sum( (y-X%*%b.hat)^2)/(N-k)
  var.beta <- s2*solve(crossprod(X))
  se.bhat <- sqrt(diag(var.beta))
  t <- b.hat/se.bhat
  p <- pt(abs(t), lower.tail = FALSE, df=N-k)*2
  output <- list(est = b.hat,
                 se = se.bhat,
                 Var = var.beta,
                 p.values  = p)
  return(output)
}
ols(y, X)
```

```
## $est
##            [,1]
## [1,] -0.9900163
## [2,]  1.9303936
## [3,] -2.0016281
##
## $se
## [1] 0.04489838 0.04498747 0.04409603
##
## $Var
##              [,1]          [,2]          [,3]
## [1,] 2.015865e-03  1.174113e-05  6.783560e-05
## [2,] 1.174113e-05  2.023872e-03 -3.459786e-06
## [3,] 6.783560e-05 -3.459786e-06  1.944460e-03
##
## $p.values
##               [,1]
## [1,]  1.292985e-96
## [2,] 1.211680e-285
## [3,] 9.603416e-310
```

```
# Solution to problem 2.4
grNormalMLE <- function(theta, X,y){
  lnsigma2 <- theta[length(theta)]
  sigma2 <- exp(lnsigma2)
  beta <- theta[-length(theta)]

  dBeta <- (X * as.numeric(y-X%*%beta))/sigma2
  dln.sigma2 <- (y- X%*%beta)^2 / (2*sigma2) - 1/2
  D <- colSums(cbind(dBeta, dln.sigma2))
  return(-D)
}
```

A.3 Answer to chapter 3 exercises

```
# Solution to problem 3.1
library(tidyr)
library(readstata13)
library(stringr)
library(countrycode)

pressData <- read.csv('Datasets/Press_FH.csv')
FL <- read.dta13("Datasets/FearonLaitin_CivilWar2003.dta",
                 nonint.factors = TRUE,
                 convert.dates = FALSE)

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
  mutate(year=as.numeric(year),
         ccode = countrycode(country, origin="country.name", destination="cown"),
         ccode = ifelse(str_detect(country, "Serbia"), 345, ccode ))
```

```
## Warning in countrycode_convert(sourcevar = sourcevar, origin = origin, destination =
```

```
newData <- merge(FL, pressData, by=c("ccode", "year"), all.x=TRUE, all.y=FALSE)
dim(newData)
```

```
## [1] 6651   55
```

```
dim(FL)
```

```
## [1] 6610   53
```

```
newData %>%
  select(ccode, year) %>%
  filter(duplicated(.))
```

```
##    ccode year
## 1    255 1991
## 2    255 1992
## 3    255 1993
## 4    255 1994
## 5    255 1995
## 6    255 1996
## 7    255 1997
## 8    255 1998
## 9    255 1999
## 10   345 1979
## 11   345 1979
## 12   345 1980
## 13   345 1980
## 14   345 1981
## 15   345 1981
## 16   345 1982
## 17   345 1982
## 18   345 1983
## 19   345 1983
## 20   345 1984
## 21   345 1984
## 22   345 1985
## 23   345 1985
## 24   345 1986
## 25   345 1986
## 26   345 1987
## 27   345 1987
## 28   345 1988
## 29   345 1988
## 30   345 1989
## 31   345 1989
## 32   345 1990
## 33   345 1990
## 34   365 1992
## 35   365 1993
## 36   365 1994
## 37   365 1995
## 38   365 1996
## 39   365 1997
## 40   365 1998
## 41   365 1999
```

```
# issues with West Germany (255 here should be 260),
# Serbia/Yugoslavia/Serbia & Montenegro triple counting
# Russia/Soviet Union double counting
# Looking at the data, it becomes clear that if we just trash the NAs and fix West Ger
# We'll be good
pressData <- pressData %>%
  mutate(ccode = ifelse(country=="Germany, West", 260, ccode),
         press= ifelse(press=="N/A", NA, press)) %>%
  na.omit()

newData <- merge(FL, pressData, by=c("ccode", "year"), all.x=TRUE, all.y=FALSE)
dim(newData)
```

```
## [1] 6610   55
```

```
dim(FL)
```

```
## [1] 6610   53
```

```
FL %>%
  group_by(region) %>%
  summarise(Polity=mean(polity2, na.rm=TRUE))
```

```
## # A tibble: 6 x 2
##   region                                Polity
##   <fct>                                  <dbl>
## 1 western democracies and japan           8.67
## 2 e. europe and the former soviet union -3.40
## 3 asia                                -1.77
## 4 n. africa and the middle east       -4.25
## 5 sub-saharan africa                  -3.73
## 6 latin america and the caribbean      0.836
```