Flash R

Minerva Statistical Consulting

David John Baker, Ph.D.

07/20/2019

Session I (< 45 Minutes)

Afternoon Goals

- Install R
- Install RStudio
- Run Basic Commands in Console
- Run Basic Tidyverse Commands

Why R?

1. The R community is fantastic, check out #rstats on Twitter as well as everyone affiliated with the tidyVerse
2. R will always be free because the people behind it believe in open source principles.
3. Time spent learning R is time spent learning how computers work. If you learn about R, you are also learning computer programming. Time spent in something like SPSS or SAS does not easily transfer to other programs.
4. On r-jobs.com the way they decide to split jobs is jobs that make above and below $100,000.
5. R is your ticket out of academia, if you need it. It's also insane to think people would learn so much about statistics, the hardest part about becoming a data scientist, without learning the software to get you in the door.
6. When you make analyses and graphs in R they are very easy to reproduce. You just press 'Run' again.
7. If you do your data cleaning in R, then each step is documented. There is less chance for human error.
8. It makes gorgeous graphs.
9. There are a lot of ways that R integrates into other software. This book is written in bookdown, my website is written in blogdown, you can also make interactive data applications.
10. It's kind of fun.

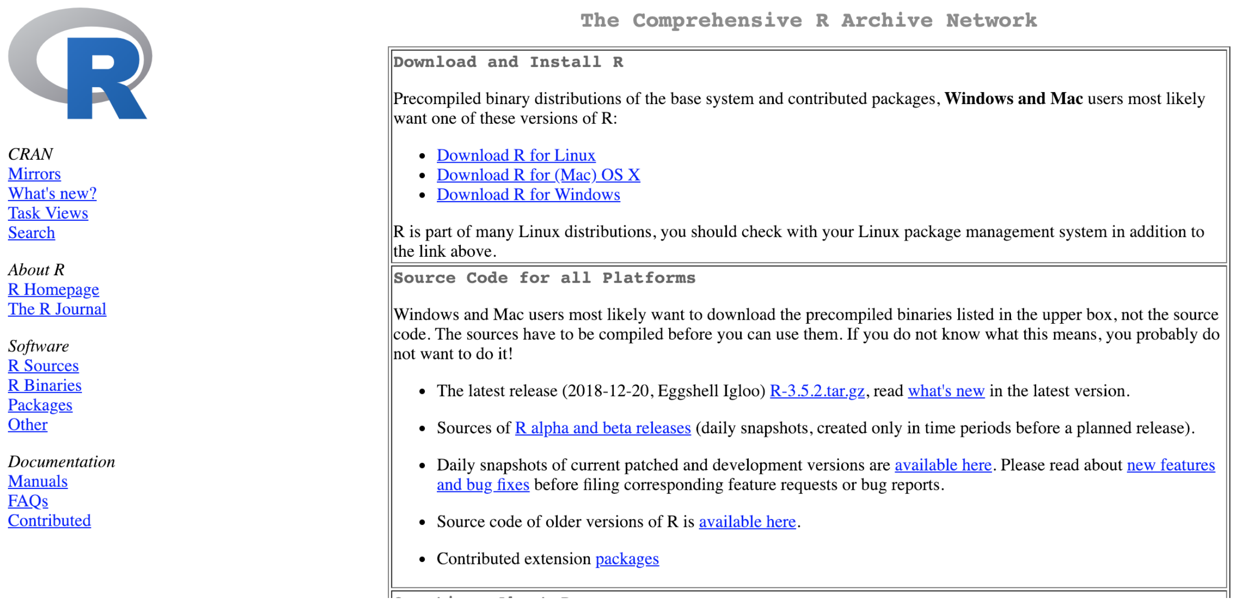


Figure 1: CRAN Homepage

R, RStudio, and Tidyverse

R

You can download R for your computer by going to CRAN and selecting the appropriate `Download and Install R` links. Make sure to install R first before installing RStudio.

RStudio

RStudio is an integrated development environment (IDE) for R[1]. RStudio is basically your workbench where you can access everything you need for managing your scripts, data, and project structure. By using RStudio, you also can use a host of other features ranging from Markdown documents (like this one!), interactive data dashboards like Shiny, and the tidyverse.

RStudio Environment

Once you now have R and RStudio installed, it's time to open up RStudio. By opening RStudio, you are also starting R. R will be running under the hood of RStudio. After installing R, you can run it on it's own by typing `R` into your terminal on a Unix machine (Mac, Linux). Though after seeing how RStudio works, you would realize why doing this is basically masochistic. (If you do this, you can quit out of the terminal R with `quit()` followed by `n`).

When you first open RStudio will see a few different panels. In it's default settings, the bottom left is the Console. The top right has your Environment, History, and version control commands. The bottom right has your Viewer, Library for your packages, and a system to navigate your files. The top left will be where you write your code.

[1] https://www.rstudio.com/products/RStudio/

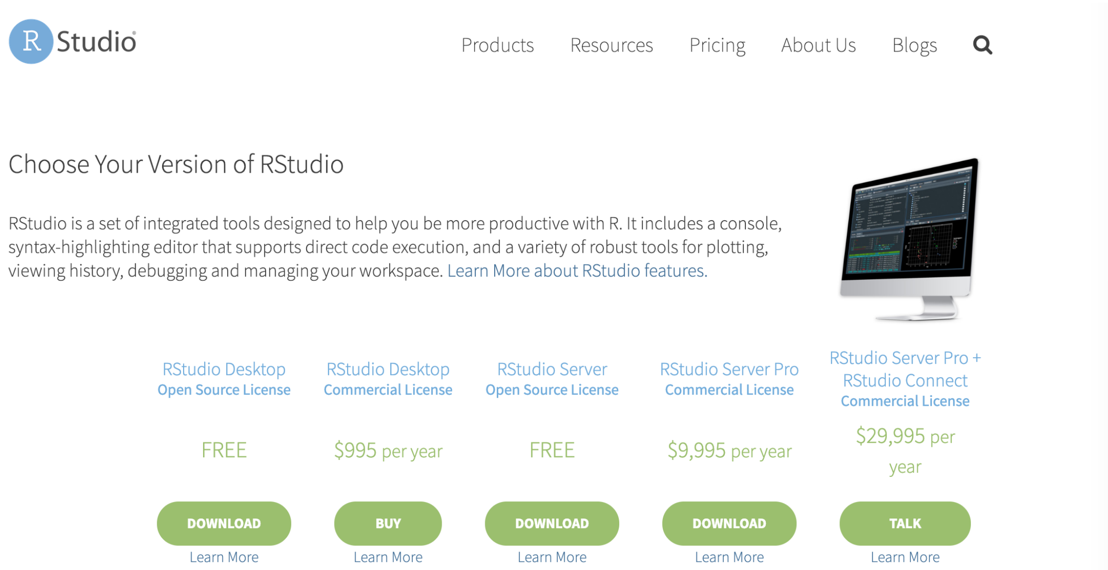


Figure 2: RStudio

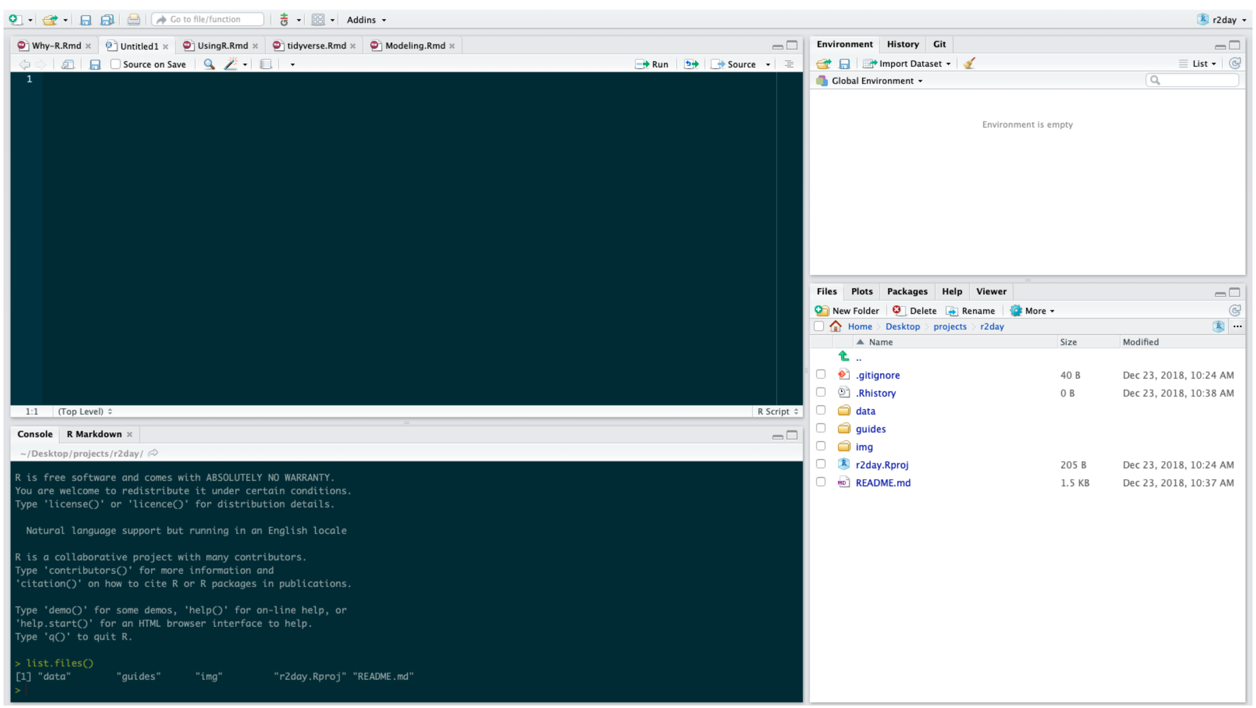


Figure 3: RStudio Environment

Environment

The top left has information about your current Environment. As you make new things in an R session you can track them here. There is also a History tab here that keeps track of code you wrote. Additionally there is a Git tab that will eventually allow you to do version control. You don't have to know what that is, but one day you might read about it.

Viewer

The bottom right is your File Explorer/Finder window. Try to click around on the **Files** tab. When you click **Plots** there should be nothing there as you have not made any plots yet. Your **Packages** tab will have a listing of software that you can load into R. Notice that if you click one of the package names, it will navigate you to the **Help** tab. Lastly, the Viewer tab will let you display any documents that you make while writing in R. This could be markdown documents or maybe a website that you are writing eventually.

> It is important to note that you will probably "break" R and RStudio many times when learning. Know that this is OK and the some of the best advice for learning how to program is by just seeing what happens when you change something and Googling your problems.

Console

The Console in R is where you can run one-off R commands. Try to type a few of to following commands into the Console.

```r
list.files()
```

```
## [1] "Flash_R_files" "Flash_R.html"  "Flash_R.Rmd"   "flash_R.Rproj"
## [5] "img"           "README.md"     "slides"
```

```r
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```r
2 + 2
```

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

Each of these will create a different kind of output. Now try to put something in your R console that will create an error message. Maybe some math that ends with an operation sign? Maybe some text? In the next session, we will go over what is legal and illegal input in the R Console.

Editor

The top left panel is where you edit your documents. RStudio allows you to handle many different types of documents. In this course we will mostly use RMarkdown files. These files end in `.Rmd` and allow for both text and R code. R scripts on the other hand only handle R code.

Using the editor, you should also familiarize yourself with the keyboard shortcuts in RStudio. For example, to run a line of code in the Editor, you can press `CMD + ENTER` on Mac or `CTRL + ENTER` on Windows. When the cursor is on a line that has runable R code, this will run that line in the console. You can also use your mouse highlight many lines of R code an run the same commands. We will get a lot of practice with this in the next session.

Session 2 - Using R (1 Hour)

Lesson Goals

- Run Basic Commands in R
- Understand Basic Data Structures
- Run commands over vectors
- Index data frames
- Learn basic data structures
- Understand base R vs Tidyverse
- Import and export data to/from R

RStudio Shortcuts and Markdown

In this session I will be using a lot of Keyboard Shortcuts when typing myself. In the past, people have always asked about these, so I'm anticipating that question with a link here to that page.

R as Calculator

The Console of R is where all the action happens. You can use it just like you would use a calculator. Try to do some basic math operations in it.

```r
2 + 2
```

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

```r
5 - 2
```

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

```r
10 / 4
```

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

```r
9 * 200
```

```
## [1] 1800
```

```r
sqrt(81)
```

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

```r
10 > 4
```

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

```r
2 < -1
```

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

Now from the output above, you'll notice that there are a few different types of responses that R will give. For the math responses, we get numbers, but we can also get TRUE and FALSE statements.

When working with data, we need to be aware not only of what the data represents, but what R thinks it represents. We won't go over the differences between things like ordinal, ratio, and categorical data, I'll assume you have a basic understanding of this. What we will focus on is the different data types that R thinks in.

For now, we are going to talk about R's basic data structures.

- Logical
- Integer
- Double (numeric)
- Character
- Factor

The first is logical. Logical is basically just TRUE or FALSE. We can try a few different expressions that show how this works.

```r
2 > 4
```

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

```r
1 > 0
```

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

```r
4 >= 7
```

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

```r
5 != 5
```

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

Eventually you will learn to take advantage of the complexities of this when we get to subsetting and combining them with other logical operators like `&`(and) and `|` (OR).

Next we have integers and double. Both integers and double are R's numeric forms of data. The `is.numeric()` command checks for if data is number-y.

```r
is.integer(7L)
```

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

```r
is.double(7)
```

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

```r
is.numeric(7)
```

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

Next we have characters. Characters are not *just* letters, but rather data that is text. Character data is always wrapped in quotes `" "`

```
is.character("hello, world!")
```

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

```
is.character("7")
```

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

```
is.character("I will drink 7 coffees by the end of today!")
```

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

```
is.character("NA")
```

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

Note that if a special character like `NA` is in quotes, R will still think it is a character. To change this, we need to coerce our data into a different type. We will cross that bridge later. For now, you just need to be aware of the different character types.

Lastly, there are factors which sometimes LOOK like characters, but are R's way of thinking about categorical data. We need to assign this to R. When you first import in data into R, it will sometimes guess it as being a factor which is very annoying! If R is being slow, or not responding to something you want it to do, a common rookie mistake is to have your data accidentally be a factor.

```
is.factor("doggo")
```

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

```
doggo <- as.factor("doggo")

is.factor(doggo)
```

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

```
is.character(doggo)
```

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

```
is.numeric(doggo)
```

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

Now that we're at least aware of the different types of data in R, we can move on to building up an intuitive understanding of how R thinks about data under the hood.

Being Lazy

You don't always want to print your output and retype it in. The idea of being a good programmer is to be very lazy (efficient).

One of the best ways to be efficient when programming is to save variables to objects. Below is some example code that uses the `<-` operator to assign some math to an object. After you assign it to an object, you can then manipulate it like you would any other number. Yes, you can use `=` as an assignment operator (for all you Pythonistas), but in R this is considered bad practice as R is primarily a statistical programming language and the `=` sign means something very different in a math context.

```
foo <- 2 * 3
foo * 6
```

```
## [1] 36
```

After running these two lines of code, notice what has popped up in your environment in RStudio! You should see that you now have any object in the Environment called `foo`.

In addition to saving single values to objects, you can also store a collection of values. Below we use an example that might have a bit more meaning, the below stores what could be some data into an object that represents what it might be.

```r
yearsSellingWidgets <- c(2,1,4,5,6,7,3,2,4,5,3)
```

The way that the line above works is that we use the `c()` function (c for combine) to group together a bunch of the same type of data (numbers) into a vector. Once we have everything combined and stored into an object, we can then manipulate all the numbers in the object just like we did above with a single number. A single dimensional object is called a **vector**. For example, we could multiply all the numbers by three.

```r
yearsSellingWidgets * 3
```

```
##  [1]  6  3 12 15 18 21  9  6 12 15  9
```

Or maybe we realized that our inputs were wrong and we need to shave off two years off of each of the entries.

```r
yearsSellingWidgets - 2
```

```
##  [1]  0 -1  2  3  4  5  1  0  2  3  1
```

Or perhaps we want to find out which of our pieces of data (and other data associated with that observation) are less than 2.

```r
yearsSellingWidgets < 2
```

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

Any sort of mathematical operation can be performed on a vector! In addition to treating it like a mathematical operation, we can also run functions on objects. By looking at the name of each function and it's output, take a guess at what each of the below functions does.

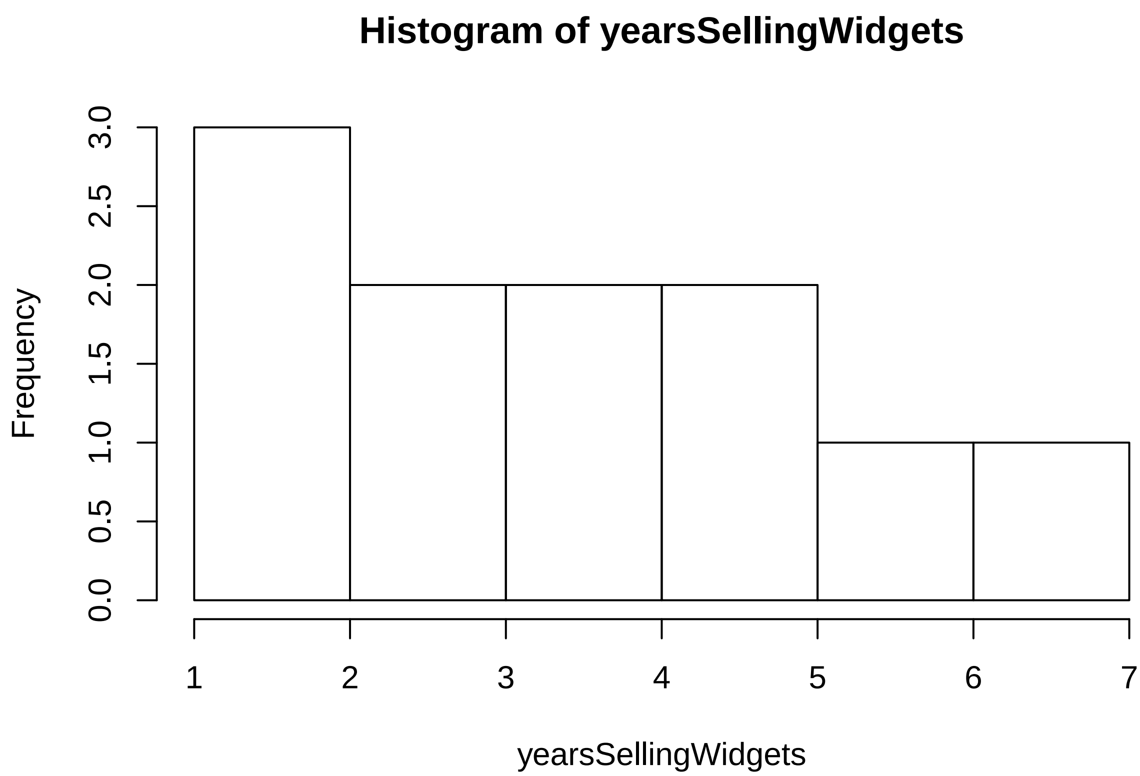
```r
mean(yearsSellingWidgets)
```

```
## [1] 3.818182
```

```r
sd(yearsSellingWidgets)
```

```
## [1] 1.834022
```

```r
hist(yearsSellingWidgets)
```




```r
scale(yearsSellingWidgets)
```

```
##              [,1]
##  [1,] -0.99136319
##  [2,] -1.53661295
##  [3,]  0.09913632
##  [4,]  0.64438608
##  [5,]  1.18963583
##  [6,]  1.73488559
##  [7,] -0.44611344
##  [8,] -0.99136319
##  [9,]  0.09913632
## [10,]  0.64438608
## [11,] -0.44611344
## attr(,"scaled:center")
## [1] 3.818182
## attr(,"scaled:scale")
## [1] 1.834022
```

```r
range(yearsSellingWidgets)
```

```
## [1] 1 7
```

```r
min(yearsSellingWidgets)
```

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

```r
class(yearsSellingWidgets)
```

```
## [1] "numeric"
```

```r
str(yearsSellingWidgets)
```

```
##  num [1:11] 2 1 4 5 6 7 3 2 4 5 ...
```

```r
summary(yearsSellingWidgets)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.500   4.000   3.818   5.000   7.000
```

Often working with data, we don't want to just play with one group of numbers. Most of the time we are trying to compare different observations in data science. If we then create two vectors (one of which we have already made!) and then combine them together into a data frame, we have something sort of looking like a spreadsheet. A two-dimensional object is called a **data frame**.

```r
yearsSellingWidgets <- c(2,1,4,5,6,7,3,2,4,5,3)
numberOfSales <- c(5,2,5,7,9,9,2,8,4,7,2)
salesData <- data.frame(yearsSellingWidgets,numberOfSales)
salesData
```

```
##    yearsSellingWidgets numberOfSales
## 1                    2             5
## 2                    1             2
## 3                    4             5
## 4                    5             7
## 5                    6             9
## 6                    7             9
## 7                    3             2
## 8                    2             8
## 9                    4             4
## 10                   5             7
## 11                   3             2
```

Now if we wanted to use something like R's correlation function we could just pass in the two objects that we have like this and get a correlation value.

```r
cor(yearsSellingWidgets,numberOfSales)
```

```
## [1] 0.6763509
```

But often our data will be saved in data frames and we need to be able to access one of our vectors inside our data frame. To access a piece of information in a data frame we use the `$` operator.

```r
salesData$yearsSellingWidgets
```

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

Running the above code will print out the vector called `yearsSellingWidgets` from the data frame `salesData`. Using this form, we can then use this with the correlation function.

```r
cor(salesData$yearsSellingWidgets,salesData$numberOfSales)
```

```
## [1] 0.6763509
```

In addition to just getting numeric output, we also want to be able to look at our data. Take a look at the code below and try to figure out what the function call is, as well as what each argument (or thing you pass to a function) does.

```r
plot(yearsSellingWidgets,numberOfSales,
     data = salesData,
     main = "My Plot",
```

```
     xlab = "Years at Company",
     ylab = "Number of Sales")
```

```
## Warning in plot.window(...): "data" is not a graphical parameter

## Warning in plot.xy(xy, type, ...): "data" is not a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "data" is not
## a graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "data" is not
## a graphical parameter

## Warning in box(...): "data" is not a graphical parameter

## Warning in title(...): "data" is not a graphical parameter
```

If you are having a hard time understanding arguments, one thing that might help to think about is that each argument is like a click in a software program like SPSS or Excel. Imagine you want to make the same plot with this data in SSPSS, what would you do? The first thing you would do is to go to the top of the bar and find the `Plot` function and click it. This is the same as typing out `plot()` in R. Then you would have to tell that new pop up screen what two variables you want to plot and click on the related variables. Dragging and dropping those variables into your plot builder is the same as just typing out the variables you want. Lastly you want to put names on your axes and a title on your plot. The same logic would follow. We'll explore these ideas a bit more in the next section

Packages and Help

One of best things to do is just open an R help page and play around with things (and break things) until you "get" how it works.

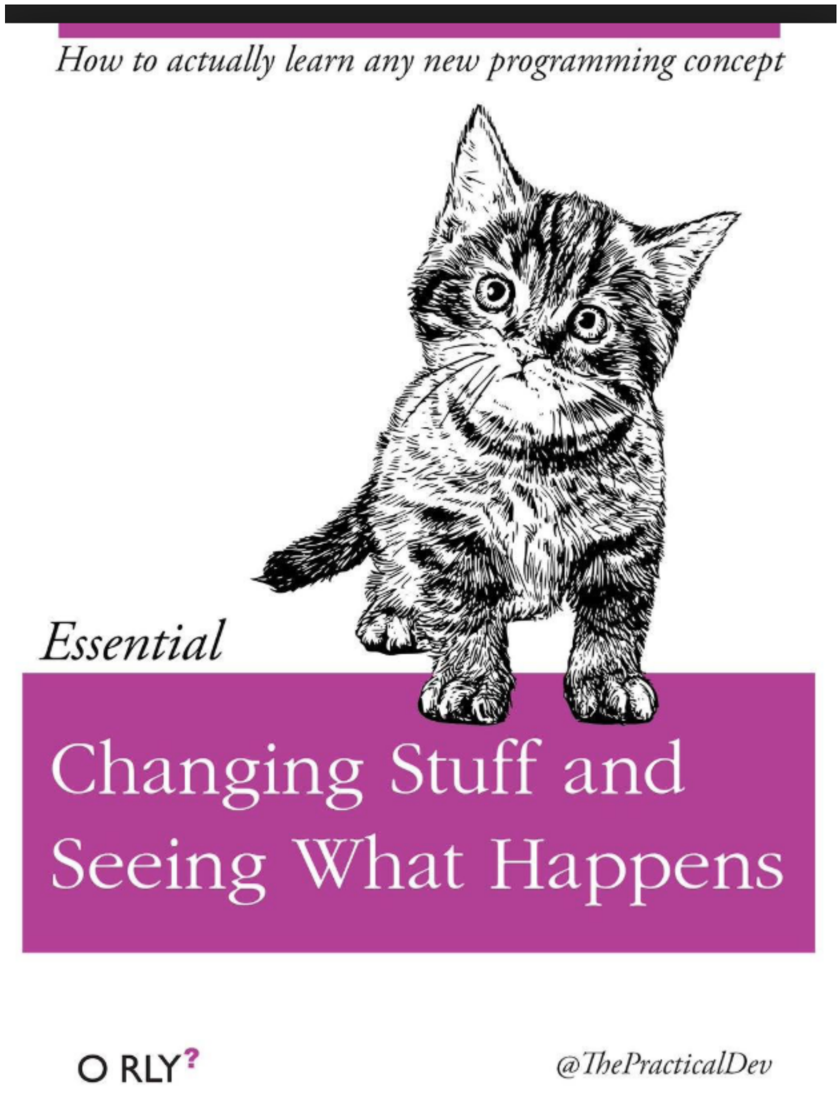


Figure 4: CRAN Homepage

To access R's in built help function you can easier use the Help viewer in R studio or type in a question mark before the command in the console. Using two `??` will search more generally

```
?scale()
??scale()
```

Data Exploration

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------- tidyverse 1.2.1 --
## v ggplot2 3.1.0       v purrr   0.3.2
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.3       v stringr 1.4.0
## v readr   1.3.1       v forcats 0.3.0
## -- Conflicts ------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
str(txhousing)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    8602 obs. of  9 variables:
##  $ city     : chr  "Abilene" "Abilene" "Abilene" "Abilene" ...
##  $ year     : int  2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 ...
##  $ month    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ sales    : num  72 98 130 98 141 156 152 131 104 101 ...
##  $ volume   : num  5380000 6505000 9285000 9730000 10590000 ...
##  $ median   : num  71400 58700 58100 68600 67300 66900 73500 75000 64500 59300 ...
##  $ listings : num  701 746 784 785 794 780 742 765 771 764 ...
##  $ inventory: num  6.3 6.6 6.8 6.9 6.8 6.6 6.2 6.4 6.5 6.6 ...
##  $ date     : num  2000 2000 2000 2000 2000 ...
```

```
class(txhousing)
```

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

```
summary(txhousing)
```

```
##      city                year          month            sales       
##  Length:8602        Min.   :2000   Min.   : 1.000   Min.   :   6.0  
##  Class :character   1st Qu.:2003   1st Qu.: 3.000   1st Qu.:  86.0  
##  Mode  :character   Median :2007   Median : 6.000   Median : 169.0  
##                     Mean   :2007   Mean   : 6.406   Mean   : 549.6  
##                     3rd Qu.:2011   3rd Qu.: 9.000   3rd Qu.: 467.0  
##                     Max.   :2015   Max.   :12.000   Max.   :8945.0  
##                                                     NA's   :568     
##      volume              median          listings       inventory     
##  Min.   :8.350e+05   Min.   : 50000   Min.   :    0   Min.   : 0.000  
##  1st Qu.:1.084e+07   1st Qu.:100000   1st Qu.:  682   1st Qu.: 4.900  
##  Median :2.299e+07   Median :123800   Median : 1283   Median : 6.200  
##  Mean   :1.069e+08   Mean   :128131   Mean   : 3217   Mean   : 7.175  
##  3rd Qu.:7.512e+07   3rd Qu.:150000   3rd Qu.: 2954   3rd Qu.: 8.150  
##  Max.   :2.568e+09   Max.   :304200   Max.   :43107   Max.   :55.900  
##  NA's   :568         NA's   :616      NA's   :1424    NA's   :1467    
```

```
##       date
##  Min.   :2000
##  1st Qu.:2004
##  Median :2008
##  Mean   :2008
##  3rd Qu.:2012
##  Max.   :2016
##
```

```r
# View(txhousing)
```

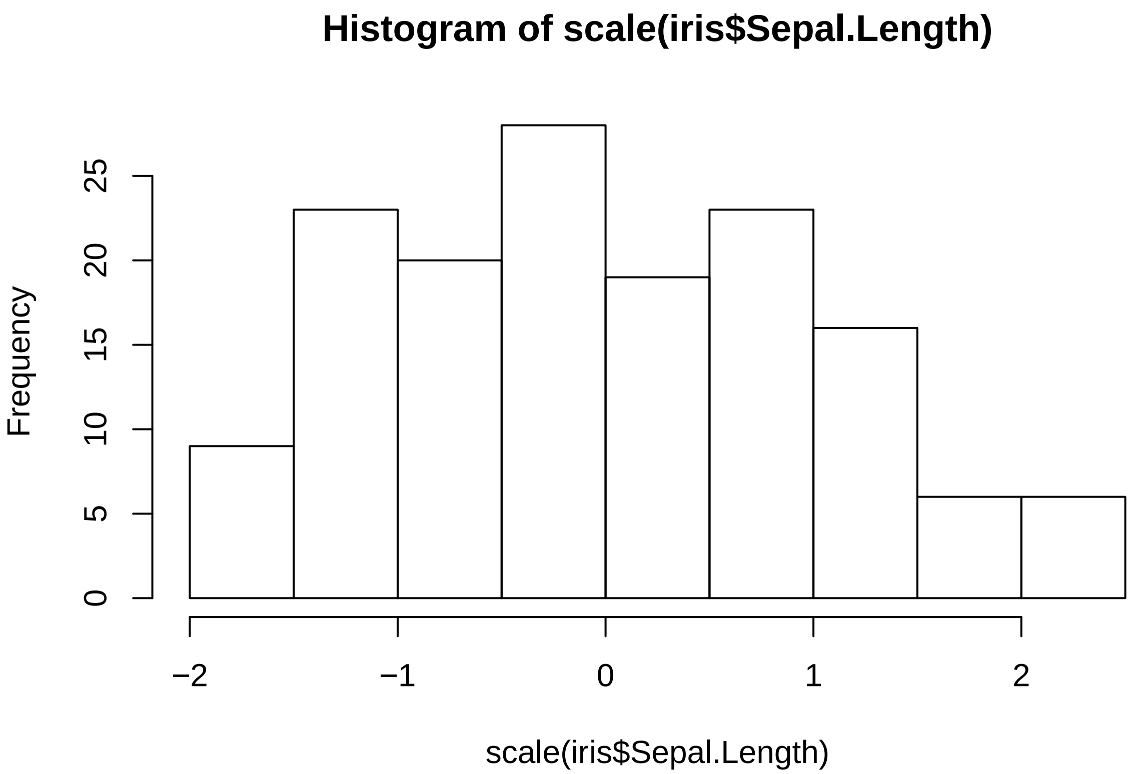
Accessing individual 'columns' is done with the $ operator

```r
txhousing$sales
```

Can you use this to plot the different numeric values against each other?

What would the follow commands do?

```r
hist(scale(iris$Sepal.Length))
```




```r
iris$Sepal.Length.scale <- scale(iris$Sepal.Length)
```

Indexing

Let's combine logical indexing with creating new objects.

What do the follow commands do? Why?

```r
txhousing[1,1]
```

```
## # A tibble: 1 x 1
```

```
##   city
##   <chr>
## 1 Abilene
```

```
txhousing[2,]
```

```
## # A tibble: 1 x 9
##   city     year month sales  volume median listings inventory  date
##   <chr>   <int> <int> <dbl>   <dbl>  <dbl>    <dbl>     <dbl> <dbl>
## 1 Abilene  2000     2    98 6505000  58700      746       6.6 2000.
```

```
txhousing[,5]
```

```
## # A tibble: 8,602 x 1
##      volume
##       <dbl>
##  1  5380000
##  2  6505000
##  3  9285000
##  4  9730000
##  5 10590000
##  6 13910000
##  7 12635000
##  8 10710000
##  9  7615000
## 10  7040000
## # ... with 8,592 more rows
```

```
txhousing[txhousing$year < 2003,]
```

```
## # A tibble: 1,656 x 9
##    city     year month sales   volume median listings inventory  date
##    <chr>   <int> <int> <dbl>    <dbl>  <dbl>    <dbl>     <dbl> <dbl>
##  1 Abilene  2000     1    72  5380000  71400      701       6.3 2000
##  2 Abilene  2000     2    98  6505000  58700      746       6.6 2000.
##  3 Abilene  2000     3   130  9285000  58100      784       6.8 2000.
##  4 Abilene  2000     4    98  9730000  68600      785       6.9 2000.
##  5 Abilene  2000     5   141 10590000  67300      794       6.8 2000.
##  6 Abilene  2000     6   156 13910000  66900      780       6.6 2000.
##  7 Abilene  2000     7   152 12635000  73500      742       6.2 2000.
##  8 Abilene  2000     8   131 10710000  75000      765       6.4 2001.
##  9 Abilene  2000     9   104  7615000  64500      771       6.5 2001.
## 10 Abilene  2000    10   101  7040000  59300      764       6.6 2001.
## # ... with 1,646 more rows
```

```
txhousing[,c(1:4)]
```

```
## # A tibble: 8,602 x 4
##    city     year month sales
##    <chr>   <int> <int> <dbl>
##  1 Abilene  2000     1    72
##  2 Abilene  2000     2    98
##  3 Abilene  2000     3   130
##  4 Abilene  2000     4    98
##  5 Abilene  2000     5   141
##  6 Abilene  2000     6   156
##  7 Abilene  2000     7   152
```

```
##  8 Abilene  2000     8   131
##  9 Abilene  2000     9   104
## 10 Abilene  2000    10   101
## # ... with 8,592 more rows
```

```
txhousing[txhousing$city=="San Antonio",c(1:6,8)]
```

```
## # A tibble: 187 x 7
##    city         year month sales    volume median inventory
##    <chr>       <int> <int> <dbl>     <dbl>  <dbl>     <dbl>
##  1 San Antonio  2000     1   820  98974924  90900       4.7
##  2 San Antonio  2000     2  1075 120851076  86000       4.7
##  3 San Antonio  2000     3  1433 167748201  87000       4.9
##  4 San Antonio  2000     4  1263 145280248  90200       5
##  5 San Antonio  2000     5  1574 183281564  91200       5
##  6 San Antonio  2000     6  1666 210779154 100100       5
##  7 San Antonio  2000     7  1508 185816640 100500       4.9
##  8 San Antonio  2000     8  1626 195515195  93400       5.2
##  9 San Antonio  2000     9  1300 156643797  94800       5.2
## 10 San Antonio  2000    10  1192 141630200  93500       5.2
## # ... with 177 more rows
```

```
AbilineData <- txhousing[txhousing$city == "Abilene",]
```

This could be an entire lecture by itself!!! It is important to know how R's indexing works, but in the year 2019 there is no need to be using base R command to index. We will talk more about the tidyverse tomorrow, but the following code does the exact same indexing as the base R code above, but is much more human readable.

Tidyverse

```
txhousing %>%
  filter(year < 2003)
```

```
## # A tibble: 1,656 x 9
##    city     year month sales   volume median listings inventory  date
##    <chr>   <int> <int> <dbl>    <dbl>  <dbl>    <dbl>     <dbl> <dbl>
##  1 Abilene  2000     1    72  5380000  71400      701       6.3 2000
##  2 Abilene  2000     2    98  6505000  58700      746       6.6 2000.
##  3 Abilene  2000     3   130  9285000  58100      784       6.8 2000.
##  4 Abilene  2000     4    98  9730000  68600      785       6.9 2000.
##  5 Abilene  2000     5   141 10590000  67300      794       6.8 2000.
##  6 Abilene  2000     6   156 13910000  66900      780       6.6 2000.
##  7 Abilene  2000     7   152 12635000  73500      742       6.2 2000.
##  8 Abilene  2000     8   131 10710000  75000      765       6.4 2001.
##  9 Abilene  2000     9   104  7615000  64500      771       6.5 2001.
## 10 Abilene  2000    10   101  7040000  59300      764       6.6 2001.
## # ... with 1,646 more rows
```

```
txhousing %>%
  select(city:volume)
```

```
## # A tibble: 8,602 x 5
##    city     year month sales   volume
##    <chr>   <int> <int> <dbl>    <dbl>
```

```
##  1 Abilene  2000     1    72  5380000
##  2 Abilene  2000     2    98  6505000
##  3 Abilene  2000     3   130  9285000
##  4 Abilene  2000     4    98  9730000
##  5 Abilene  2000     5   141 10590000
##  6 Abilene  2000     6   156 13910000
##  7 Abilene  2000     7   152 12635000
##  8 Abilene  2000     8   131 10710000
##  9 Abilene  2000     9   104  7615000
## 10 Abilene  2000    10   101  7040000
## # ... with 8,592 more rows
```

```
txhousing %>%
  select(1:6, inventory) %>%
  filter(city == "San Antonio")
```

```
## # A tibble: 187 x 7
##    city         year month sales    volume median inventory
##    <chr>       <int> <int> <dbl>     <dbl>  <dbl>     <dbl>
##  1 San Antonio  2000     1   820  98974924  90900       4.7
##  2 San Antonio  2000     2  1075 120851076  86000       4.7
##  3 San Antonio  2000     3  1433 167748201  87000       4.9
##  4 San Antonio  2000     4  1263 145280248  90200       5
##  5 San Antonio  2000     5  1574 183281564  91200       5
##  6 San Antonio  2000     6  1666 210779154 100100       5
##  7 San Antonio  2000     7  1508 185816640 100500       4.9
##  8 San Antonio  2000     8  1626 195515195  93400       5.2
##  9 San Antonio  2000     9  1300 156643797  94800       5.2
## 10 San Antonio  2000    10  1192 141630200  93500       5.2
## # ... with 177 more rows
```

```
AbilineData <- txhousing %>%
  filter(city == "Abiline")
```

As your code gets longer, the tidyverse becomes more readable. It is also more helpful for exploring data sets.

Saving and Importing

Finally, if we want to Import or Save other data, we can do that via the Console.

The Working Directory

Most of the work we have done this far is data that we do not want to save. Most of the work you will do after this workshop, you will want to save.

R works by pointing at a folder or directory on your computer. To see where R is pointing now, run the following code

```
getwd()
```

```
## [1] "/Users/davidjohnbaker/Desktop/projects/flash_R"
```

Whatever you **do** in your R session will happen here unless you tell it to otherwise. If you do not want R pointing in this location in your computer, you need to set your working directory elsewhere. To do this, use the `setwd()` command. This is also a good chance to use RStudio's auto complete feature.

```r
setwd()
```

Open a double quotation in the function then press TAB. This will allow you to navigate your computer. Going deeper into your directory structure can be done by just following the auto complete. Going higher in the directory requires you to type “‘`which will allow you to look up a level. Set your working directory to the`output“‘ directory.

The console should now read that it is pointed to the output directory.

You can write a dataset to your working directory with the `write_csv()` command.

```r
write.csv(x = AbilineData, file = "MyData.csv")
```

Importing Data

Data is imported using the same logic. You can use the `read.csv()` function to read in a csv file. At first, it might be easier to use the Import Dataset function in RStudio (Top right pane).

Session 3 - tidyverse (1 Hour)

Lesson Goals

- Be able to explain tidy data
- Explain the five tidyverse verbs
- Perform basic indexing
- Import and Export data from R

tidyverse + tidydata

One of the most important concepts data science and R is the idea of tidydata.

The idea behind tidy data is that…

1. Each variable forms a column
2. Each observation forms a row.
3. Each type of observation unit forms a table.

If your data is in this format, then you can do almost anything with the tidyverse.

In order to use the tidyverse, you first need to install it.

```
# install.packages("tidyverse") # Only need to do this once!
library(tidyverse)
```

Five Verbs

The five tidyverse verbs come from the dplyr package. More information on this package can be found here along with these descriptions.

- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.

We can think the verbs as happening in the logical order you would want to grab them. Each of the verbs is also going to be connected to one another with the pipe operator. The idea behind the pipe or '`%>%` is that the output of the last line is the first argument of the new function.

For example, if we wanted to make a small table that only had data from El Paso from 2011, then only get the first and fifth columns we would run the following code:

```
str(txhousing)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    8602 obs. of  9 variables:
##  $ city     : chr  "Abilene" "Abilene" "Abilene" "Abilene" ...
##  $ year     : int  2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 ...
##  $ month    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ sales    : num  72 98 130 98 141 156 152 131 104 101 ...
##  $ volume   : num  5380000 6505000 9285000 9730000 10590000 ...
##  $ median   : num  71400 58700 58100 68600 67300 66900 73500 75000 64500 59300 ...
##  $ listings : num  701 746 784 785 794 780 742 765 771 764 ...
##  $ inventory: num  6.3 6.6 6.8 6.9 6.8 6.6 6.2 6.4 6.5 6.6 ...
##  $ date     : num  2000 2000 2000 2000 2000 ...
```

```
txhousing_only_el_paso <- txhousing[txhousing$city == "El Paso",]
iris_only_only_el_paso_2005_2011 <- txhousing_only_el_paso[txhousing_only_el_paso$year >= 2011,]
iris_only_only_el_paso_2005_2011[,c(1,5)]
```

```
## # A tibble: 55 x 2
##    city      volume
##    <chr>      <dbl>
##  1 El Paso 66136913
##  2 El Paso 44840808
##  3 El Paso 63884923
##  4 El Paso 74429226
##  5 El Paso 61624856
##  6 El Paso 71212091
##  7 El Paso 72366107
##  8 El Paso 86547783
##  9 El Paso 64083406
## 10 El Paso 67015215
## # ... with 45 more rows
```

Which is a bit verbose.

In order to do this with the tidyverse, you would start with the dataset, the run two verbs over it, connected with the pipe.

```
library(tidyverse)
iris_tibble <- as_tibble(iris)

txhousing %>%
  filter(city == "El Paso") %>%
  filter(year >= 2011) %>%
  select(1,5)
```

```
## # A tibble: 55 x 2
##    city      volume
##    <chr>      <dbl>
##  1 El Paso 66136913
##  2 El Paso 44840808
##  3 El Paso 63884923
##  4 El Paso 74429226
##  5 El Paso 61624856
##  6 El Paso 71212091
##  7 El Paso 72366107
##  8 El Paso 86547783
##  9 El Paso 64083406
## 10 El Paso 67015215
## # ... with 45 more rows
```

Both create the same output, but one is much easier to read.

We will now explore a dataset using the five verbs in the dplyr package. You use each of the five verbs as you would in English to think about how you want to manipulate your data.

They key to using the tidyverse is the %>% operator (the pipe operator). It works by taking output from what is before it and piping it to the next command.

Economics Data

The dataset here comes from housing sales data in Texas provided by the TAMU real estate centre.

Variable	Description
city	Name of MLS area
year,month,date	Date
sales	Number of sales
volume	Total value of sales
median	Median sale price
listings	Total active listings
inventory	"Months inventory" : amount of time it would take to sell all current listings at current pace of sales.

Select

The select command works by "Selecting" the columns you wish to work with. It can either take the index of the column using numbers, or the text. There are other options like asking for columns that start or end with certain text.

```r
txhousing %>%
  select(1,2)
```

```
## # A tibble: 8,602 x 2
##    city     year
##    <chr>   <int>
##  1 Abilene  2000
##  2 Abilene  2000
##  3 Abilene  2000
##  4 Abilene  2000
##  5 Abilene  2000
##  6 Abilene  2000
##  7 Abilene  2000
##  8 Abilene  2000
##  9 Abilene  2000
## 10 Abilene  2000
## # ... with 8,592 more rows
```

```r
txhousing %>%
  select(1:3)
```

```
## # A tibble: 8,602 x 3
##    city     year month
##    <chr>   <int> <int>
##  1 Abilene  2000     1
##  2 Abilene  2000     2
##  3 Abilene  2000     3
##  4 Abilene  2000     4
##  5 Abilene  2000     5
##  6 Abilene  2000     6
##  7 Abilene  2000     7
##  8 Abilene  2000     8
##  9 Abilene  2000     9
## 10 Abilene  2000    10
## # ... with 8,592 more rows
```

```r
txhousing %>%
  select(city, sales:median)
```

```
## # A tibble: 8,602 x 4
##    city    sales   volume median
##    <chr>   <dbl>    <dbl>  <dbl>
##  1 Abilene    72  5380000  71400
##  2 Abilene    98  6505000  58700
##  3 Abilene   130  9285000  58100
##  4 Abilene    98  9730000  68600
##  5 Abilene   141 10590000  67300
##  6 Abilene   156 13910000  66900
##  7 Abilene   152 12635000  73500
##  8 Abilene   131 10710000  75000
##  9 Abilene   104  7615000  64500
## 10 Abilene   101  7040000  59300
## # ... with 8,592 more rows
```

```r
txhousing %>%
  select(starts_with("ci"))
```

```
## # A tibble: 8,602 x 1
##    city
##    <chr>
##  1 Abilene
##  2 Abilene
##  3 Abilene
##  4 Abilene
##  5 Abilene
##  6 Abilene
##  7 Abilene
##  8 Abilene
##  9 Abilene
## 10 Abilene
## # ... with 8,592 more rows
```

```r
txhousing %>%
  select(-city)
```

```
## # A tibble: 8,602 x 8
##     year month sales   volume median listings inventory  date
##    <int> <int> <dbl>    <dbl>  <dbl>    <dbl>     <dbl> <dbl>
##  1  2000     1    72  5380000  71400      701       6.3 2000
##  2  2000     2    98  6505000  58700      746       6.6 2000.
##  3  2000     3   130  9285000  58100      784       6.8 2000.
##  4  2000     4    98  9730000  68600      785       6.9 2000.
##  5  2000     5   141 10590000  67300      794       6.8 2000.
##  6  2000     6   156 13910000  66900      780       6.6 2000.
##  7  2000     7   152 12635000  73500      742       6.2 2000.
##  8  2000     8   131 10710000  75000      765       6.4 2001.
##  9  2000     9   104  7615000  64500      771       6.5 2001.
## 10  2000    10   101  7040000  59300      764       6.6 2001.
## # ... with 8,592 more rows
```

Filter

Once we have the columns we want to work with, we can then pick the rows that are of interest. We do this with the filter function. Here when asking for matches of character strings, you need to use the `==`. R will remind you if you forget. The filter command can be combined with the logical operators. Remember this includes negation operators.

```r
txhousing %>%
  filter(city == "El Paso")
```

```
## # A tibble: 187 x 9
##    city     year month sales   volume median listings inventory  date
##    <chr>   <int> <int> <dbl>    <dbl>  <dbl>    <dbl>     <dbl> <dbl>
##  1 El Paso  2000     1   306 31525000  82100     2512       5.8 2000
##  2 El Paso  2000     2   346 32300000  76600     2572       5.9 2000.
##  3 El Paso  2000     3   492 47505000  77100     2549       5.8 2000.
##  4 El Paso  2000     4   382 37915000  79400     2525       5.9 2000.
##  5 El Paso  2000     5   459 43335000  80100     2552       5.9 2000.
##  6 El Paso  2000     6   486 47880000  83200       NA        NA 2000.
```

```
##   7 El Paso   2000     7    422 42925000  82600     2685       6.4 2000.
##   8 El Paso   2000     8    538 53800000  81700     3396       8   2001.
##   9 El Paso   2000     9    382 36775000  78300     2661       6.3 2001.
##  10 El Paso   2000    10    392 40535000  81900     2704       6.5 2001.
## # ... with 177 more rows
```

```
txhousing %>%
  filter(city == "El Paso" | city == "San Antonio")
```

```
## # A tibble: 374 x 9
##     city      year month sales   volume median listings inventory  date
##     <chr>    <int> <int> <dbl>    <dbl>  <dbl>    <dbl>     <dbl> <dbl>
##   1 El Paso   2000     1    306 31525000  82100     2512       5.8 2000
##   2 El Paso   2000     2    346 32300000  76600     2572       5.9 2000.
##   3 El Paso   2000     3    492 47505000  77100     2549       5.8 2000.
##   4 El Paso   2000     4    382 37915000  79400     2525       5.9 2000.
##   5 El Paso   2000     5    459 43335000  80100     2552       5.9 2000.
##   6 El Paso   2000     6    486 47880000  83200       NA      NA   2000.
##   7 El Paso   2000     7    422 42925000  82600     2685       6.4 2000.
##   8 El Paso   2000     8    538 53800000  81700     3396       8   2001.
##   9 El Paso   2000     9    382 36775000  78300     2661       6.3 2001.
##  10 El Paso   2000    10    392 40535000  81900     2704       6.5 2001.
## # ... with 364 more rows
```

```
txhousing %>%
  filter(city == "El Paso" | city == "San Antonio") %>%
  filter(year >= 2004)
```

```
## # A tibble: 278 x 9
##     city      year month sales   volume median listings inventory  date
##     <chr>    <int> <int> <dbl>    <dbl>  <dbl>    <dbl>     <dbl> <dbl>
##   1 El Paso   2004     1    435 48330000  93500     3028       5.8 2004
##   2 El Paso   2004     2    441 48215000  89600     3162       6   2004.
##   3 El Paso   2004     3    551 60105000  89000     3288       6.2 2004.
##   4 El Paso   2004     4    579 64980000  89900     3320       6.2 2004.
##   5 El Paso   2004     5    576 68890000  97500     3271       6.1 2004.
##   6 El Paso   2004     6    586 72925000  97200       NA      NA   2004.
##   7 El Paso   2004     7    619 77745000  99900       NA      NA   2004.
##   8 El Paso   2004     8    462 54045000  95400       NA      NA   2005.
##   9 El Paso   2004     9    294 29910000  85800       NA      NA   2005.
##  10 El Paso   2004    10    484 56625000  94800       NA      NA   2005.
## # ... with 268 more rows
```

```
txhousing %>%
  filter(city == "El Paso" | city == "San Antonio") %>%
  filter(year >= 2004) %>%
  filter(month != 1)
```

```
## # A tibble: 254 x 9
##     city      year month sales   volume median listings inventory  date
##     <chr>    <int> <int> <dbl>    <dbl>  <dbl>    <dbl>     <dbl> <dbl>
##   1 El Paso   2004     2    441 48215000  89600     3162       6   2004.
##   2 El Paso   2004     3    551 60105000  89000     3288       6.2 2004.
##   3 El Paso   2004     4    579 64980000  89900     3320       6.2 2004.
##   4 El Paso   2004     5    576 68890000  97500     3271       6.1 2004.
##   5 El Paso   2004     6    586 72925000  97200       NA      NA   2004.
```

```
##  6 El Paso  2004     7   619 77745000  99900       NA      NA   2004.
##  7 El Paso  2004     8   462 54045000  95400       NA      NA   2005.
##  8 El Paso  2004     9   294 29910000  85800       NA      NA   2005.
##  9 El Paso  2004    10   484 56625000  94800       NA      NA   2005.
## 10 El Paso  2004    11   470 55690000  96200       NA      NA   2005.
## # ... with 244 more rows
```

Mutate

The mutate command will create new variables.

```
txhousing %>%
  mutate(zSales = scale(sales))
```

```
## # A tibble: 8,602 x 10
##    city   year month sales volume median listings inventory  date
##    <chr> <int> <int> <dbl>  <dbl>  <dbl>    <dbl>     <dbl> <dbl>
##  1 Abil~  2000     1    72 5.38e6  71400      701       6.3 2000
##  2 Abil~  2000     2    98 6.50e6  58700      746       6.6 2000.
##  3 Abil~  2000     3   130 9.28e6  58100      784       6.8 2000.
##  4 Abil~  2000     4    98 9.73e6  68600      785       6.9 2000.
##  5 Abil~  2000     5   141 1.06e7  67300      794       6.8 2000.
##  6 Abil~  2000     6   156 1.39e7  66900      780       6.6 2000.
##  7 Abil~  2000     7   152 1.26e7  73500      742       6.2 2000.
##  8 Abil~  2000     8   131 1.07e7  75000      765       6.4 2001.
##  9 Abil~  2000     9   104 7.62e6  64500      771       6.5 2001.
## 10 Abil~  2000    10   101 7.04e6  59300      764       6.6 2001.
## # ... with 8,592 more rows, and 1 more variable: zSales[,1] <dbl>
```

```
txhousing %>%
  filter(city == "El Paso" | city == "San Antonio") %>%
  filter(year >= 2004) %>%
  filter(month != 1) %>%
  mutate(zScale = scale(sales))
```

```
## # A tibble: 254 x 10
##    city   year month sales volume median listings inventory  date
##    <chr> <int> <int> <dbl>  <dbl>  <dbl>    <dbl>     <dbl> <dbl>
##  1 El P~  2004     2   441 4.82e7  89600     3162       6    2004.
##  2 El P~  2004     3   551 6.01e7  89000     3288       6.2  2004.
##  3 El P~  2004     4   579 6.50e7  89900     3320       6.2  2004.
##  4 El P~  2004     5   576 6.89e7  97500     3271       6.1  2004.
##  5 El P~  2004     6   586 7.29e7  97200       NA      NA    2004.
##  6 El P~  2004     7   619 7.77e7  99900       NA      NA    2004.
##  7 El P~  2004     8   462 5.40e7  95400       NA      NA    2005.
##  8 El P~  2004     9   294 2.99e7  85800       NA      NA    2005.
##  9 El P~  2004    10   484 5.66e7  94800       NA      NA    2005.
## 10 El P~  2004    11   470 5.57e7  96200       NA      NA    2005.
## # ... with 244 more rows, and 1 more variable: zScale[,1] <dbl>
```

Arrange

Arrange will sort our data.

```r
txhousing %>%
  filter(city == "El Paso" | city == "San Antonio") %>%
  filter(year >= 2004) %>%
  filter(month != 1) %>%
  mutate(zScale = scale(sales)) %>%
  arrange(sales)
```

```
## # A tibble: 254 x 10
##    city   year month sales volume median listings inventory  date
##    <chr> <int> <int> <dbl>  <dbl>  <dbl>    <dbl>     <dbl> <dbl>
##  1 El P~  2011     2   287 4.48e7 134500     3023       6.5 2011.
##  2 El P~  2008     4   292 4.45e7 126700     4840      10.9 2008.
##  3 El P~  2004     9   294 2.99e7  85800       NA      NA   2005.
##  4 El P~  2009     2   326 4.97e7 130900     4530      11.4 2009.
##  5 El P~  2008     2   328 5.34e7 133800     4374       9   2008.
##  6 El P~  2013     2   350 5.42e7 139000     3425       7.3 2013.
##  7 El P~  2005     9   356 4.23e7 111300       NA      NA   2006.
##  8 El P~  2007    12   362 5.56e7 133500     4625       8.9 2008.
##  9 El P~  2008    11   362 5.65e7 132800     4773      12   2009.
## 10 El P~  2008    12   363 5.80e7 136300     4454      11.2 2009.
## # ... with 244 more rows, and 1 more variable: zScale[,1] <dbl>
```

```r
txhousing %>%
  filter(city == "El Paso" | city == "San Antonio") %>%
  filter(year >= 2004) %>%
  filter(month != 1) %>%
  mutate(zScale = scale(sales)) %>%
  arrange(sales, -year)
```

```
## # A tibble: 254 x 10
##    city   year month sales volume median listings inventory  date
##    <chr> <int> <int> <dbl>  <dbl>  <dbl>    <dbl>     <dbl> <dbl>
##  1 El P~  2011     2   287 4.48e7 134500     3023       6.5 2011.
##  2 El P~  2008     4   292 4.45e7 126700     4840      10.9 2008.
##  3 El P~  2004     9   294 2.99e7  85800       NA      NA   2005.
##  4 El P~  2009     2   326 4.97e7 130900     4530      11.4 2009.
##  5 El P~  2008     2   328 5.34e7 133800     4374       9   2008.
##  6 El P~  2013     2   350 5.42e7 139000     3425       7.3 2013.
##  7 El P~  2005     9   356 4.23e7 111300       NA      NA   2006.
##  8 El P~  2008    11   362 5.65e7 132800     4773      12   2009.
##  9 El P~  2007    12   362 5.56e7 133500     4625       8.9 2008.
## 10 El P~  2010     2   363 5.16e7 126300     3321       7.4 2010.
## # ... with 244 more rows, and 1 more variable: zScale[,1] <dbl>
```

Group By and Sumemrise

Often we also want to perform the same type of calculation on a group in our dataset. For this we need to group our data, then use the summarize command. We can also use the `n()` function to count the number of observations in each group.

```r
txhousing %>%
  filter(city == "El Paso" | city == "San Antonio") %>%
  filter(year >= 2004) %>%
  group_by(year) %>%
```

```
  summarise(mean = mean(sales))
```

```
## # A tibble: 12 x 2
##     year  mean
##    <int> <dbl>
##  1  2004 1102.
##  2  2005 1224.
##  3  2006 1381.
##  4  2007 1257.
##  5  2008 1006.
##  6  2009 1004.
##  7  2010 1000.
##  8  2011  980.
##  9  2012 1090.
## 10  2013 1246.
## 11  2014 1323.
## 12  2015 1461.
```

```
txhousing %>%
  filter(city == "El Paso" | city == "San Antonio") %>%
  filter(year >= 2004) %>%
  group_by(year) %>%
  summarise(mean = mean(sales), n = n())
```

```
## # A tibble: 12 x 3
##     year  mean     n
##    <int> <dbl> <int>
##  1  2004 1102.    24
##  2  2005 1224.    24
##  3  2006 1381.    24
##  4  2007 1257.    24
##  5  2008 1006.    24
##  6  2009 1004.    24
##  7  2010 1000.    24
##  8  2011  980.    24
##  9  2012 1090.    24
## 10  2013 1246.    24
## 11  2014 1323.    24
## 12  2015 1461.    14
```

Work Time

We will now explore the dataset using some guided questions.

In a new script, create following:

- Create a table with only the the first four counties in the dataset.
- Next, run the same command and run that only using one argument that adds in counties that have the work "County" in the title
- Create any new table using a single logical operator
- Create a table with a two logical operators
- Create a table that has no observations from either Paris or Waco.
- Create a new variable based on two other variables
- Find the month with the highest average scales in Tyler county for the year 2015

- Create a table with data from Austin and Galveston, using only the last three years of the dataset. Group the sales by county and then calculate z scores for each county.
- Save your new table to a csv file