ECON 300 R Lab 1

Tyler Stoeger

Hello, and welcome to ECON 300! An important part of this course is applying the theory that you will learn in lectures, and to do so, we need to know how to program and work with large data sets. This semester, we will be using R and RStudio to complete these portions of the course.

R is a free, open-source software commonly used in statistical computing. Before we get started, however, we need to download and install both R and RStudio.

R can be downloaded by following this link: https://cran.rstudio.com/. Ensure you select the correct operating system before downloading.

RStudio can be downloaded by following this link: https://posit.co/download/rstudio-desktop/. Again, ensure that you select the correct operating system before downloading.

Both R and RStudio use installation wizards to set themselves up on your device. Following the wizard should result in everything downloading correctly. However, if any issues arise, there are a variety of tutorials online that will walk you through how to download and install each of these programs.

Learning Objectives

After completing this lab, you should be familiar with:
1. Downloading and installing R and RStudio
2. The basic layout and functions of RStudio
3. Creating and saving R Scripts
4. Basic programming in R
5. Downloading and calling packages
6. Importing data sets
7. Generating summary statistics
8. Basic conditional statements; boolean and logic operators
9. Basic graphing
10. Knitting documents and preparing assignments for submission

RStudio: A Brief Introduction

A typical RStudio window can roughly broken into four quadrants. When you open the application for the first time, only three will appear. We will briefly summarize the functions of each of these quadrants.

In the bottom left (or left half if this is your first time opening RStudio), we have the console. The console is where we can type commands, assign variables, and view outputs. If we type a line of code in the console and press Enter, that code will execute. Try typing 5 + 7 in the console and then press Enter.

In the bottom right, we have another panel with tabs across the top. In the Files tab, we can view files on our computer. Under the Plots tab, graphs that we create will appear. Under the Packages tab, packages that we have installed will be listed (more on this later in the lab). The Help tab allows us to search the documentation of all the different commands that we will use while programming. The Viewer and Presentation tabs also line the top of the bottom right panel, but we won't need to use them in this course.

In the top right, we have the Environment panel. This panel lists all the data sets, variables, and functions we have loaded into our application. When we import a data set into RStudio, we can use the Environment tab

to and see what our data looks like. We can also go the the History tab and look at the previous commands that we entered into our application.

Let us now look at the fourth quadrant. In the top left corner, click File -> New File -> R Script. A blank text editor should appear in the top left panel, pushing the console down.

We showed earlier that commands could be entered directly into the console. However, this has several disadvantages. Firstly, the console is cleared each time RStudio is closed, so if your task is more complicated and requires several lines of code, you would have to retype all these commands each time you entered RStudio. Secondly, the console is where the outputs of your code is printed, and so things can quickly become messy, especially when more complex commands are used. Using an R Script solves both these issues.

R Scripts allow for more complex codes to be written in the same file and ran without having to type into the console each time. They can be saved and worked on later, which is a huge benefit for complex projects. Additionally, they only display the code typed on them and not the output, so they are significantly easier to read. **When submitting assignments for this course, you will turn in a knitted R Script file as an HTML document** (more on this later).

How do we run code from an R Script? One way is to highlight the block of code that we want to execute and click Run in the top right of our R Script panel. Alternatively, we can move our cursor to the line that we want to run and then press CTRL + Enter (or Mac equivalent). This will execute this line of code and move your cursor down to the next line.

To save a R Script, you can click on the save icon in the top left just above the R Script panel, or you can click File -> Save. **It is highly recommended that you create a folder for this course that contains all the R-related files and data sets we will use.**

Basic R Programming

Now that we have a basic familiarity with RStudio, let's get to writing codes!

R Syntax

Different values in R have different formats. When we are dealing with text in R, very often this will be in the format of strings. When assigning strings, we can use double or single quotes.

```
"Hello World"
```

```
## [1] "Hello World"
```

To print numbers, we can simply type the number.

```
3
```

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

```
4.6
```

```
## [1] 4.6
```

We can also do math using R, which we showed earlier. In R, the mathematical operators +, -, *, and / all stand for addition, subtraction, multiplication, and division, respectively. Order of operations applies when doing these calculations.

```
2 * 5 / 4
```

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

Sometimes, we want to explain the purpose of our code, or make our code more readable. For this, we use comments. To start a comment, we but a # symbol, followed by whatever text we want to type. We can do this immediately after our code in the same line, or we can do this on a separate line

```r
2 * 5 / 4 # We can comment like this
# Or like this
```

We can also use comments to make sure that line of code is not executed, perhaps when troubleshooting bugs.

In this course, programming assignments will ask you to explain the results of your code. This should be done using comments.

Variables, Vectors, and Data Frames

Numbers, strings, and more complex data structures can be stored as variables. Variable names can contain numbers, but cannot start with them.

```r
var1 <- 3
var2 <- 4
var3 <- 2 * 5
var4 <- "Hello World"

var1
```

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

```r
var2
```

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

```r
var3
```

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

```r
var4
```

```
## [1] "Hello World"
```

```r
var1 + var2 # We can use variables in equations where applicable
```

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

There are many different data structures we can use in R. Above, we were dealing with singular data types, like numbers and strings. However, these can often be combined into more complicated data structures. One such structure is called a vector, which is a list of items of the same type.

```r
fruits <- c("banana", "apple", "orange") # vectors are characterized by a c(...)
numbers <- c(1, 2, 3)

fruits
```

```
## [1] "banana" "apple"  "orange"
```

```r
numbers
```

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

The most important data structure for this course will be data frames. When we load a data set into R, we turn it into a data frame. Let's learn how to do that now.

Today, we will learn how to load in CSV (Comma Separated Values) files. CSV files are a common way to store and distribute data, they are easily compressible, and they are easily handled by a variety of sofwares including R and Excel.

Before we do that, let's learn how to set our working directory. A working directory is a file on your computer that you tell RStudio to work out of. Setting a working directory isn't necessary when working in R, but it simplifies the file path that we need to specify so that R can find the file we are referencing.

For example, without a working directory and on a Windows computer, a file path to grab the CSV file we are trying to import might look like this:

> C:/Users/tstoe/OneDrive/Grad School/6 - Summer 2024/ECON 300 - Econometrics/Lab 1/card_95_sample.csv

As you can see, to find the file, I have to tell R to start at the C Drive, get into my OneDrive, go through my grad school files, and finally get to the folder I've set up for this lab. Now, to set up your working directory, go to the top left to Session -> Set Working Directory -> Choose Directory. You can now go through your files to and select the folder that you would like R to work out of. Now, instead of the long file path like above, I can just type:

> card_95_sample.csv

since R already knows where to look.

Now that we have set our working directory, let's import our data. We can read CSV files using the read.csv() command. The read.csv() command has a lot of arguments, which we can look at in the Help panel in the bottom right.

```
?read.csv
```

Let's import our data now.

```
data <- read.csv("card_95_sample.csv")
```

If you choose to import data without setting a working directory, a common error that can occur is using the wrong slashes. Make sure that you are using a forward slash (/) instead of a backslash (\).

Another way that we can import the data into RStudio is through the Import Dataset button at the top of the Environment panel.

Downloading and Calling Packages

Sometimes, we need commands beyond what our base R installation provides for us. We can get new commands by installing packages. There are many different packages for R, all doing a variety of things. I used the knitr and tinytex package for creating this document. There are packages that help with formatting, and packages that let you make more attractive graphs.

To install a package, we use the install.packages() command. Let's install the knitr package, which we will need to turn in assignments.

```
install.packages("knitr")
```

This command will automatically search the web for the knitr package and download it to your computer. Note that when using the install.packages() command, the package name should be in quotes.

Now we have to call the package so that we can use it. By default, R and RStudio do not have all your packages activated in order to save memory. So, we call the package to tell R that we are using that packages commands. We do this using the library() command.

```
library(knitr)
```

Note that when calling packages, we do not put the package in quotes.

The install.packages() command only needs to be run once. After the package downloads, it remains on your computer. Packages need to be called with the library command at the beginning of each new R script.

Basic Summary Statistics

Now that we have a data set loaded in, we can begin to do things with it. One of the first things that you might do when starting an econometric analysis is generate summary statistics. We can get a broad overview

of our data using the summary() command.

```
summary(data)
```

```
##        id            nearc4            ed76           age76
##  Min.   :   1   Min.   :0.0000   Min.   : 0.00   Min.   :24.00
##  1st Qu.: 904   1st Qu.:0.0000   1st Qu.:12.00   1st Qu.:25.00
##  Median :1807   Median :1.0000   Median :13.00   Median :28.00
##  Mean   :1807   Mean   :0.6781   Mean   :13.23   Mean   :28.18
##  3rd Qu.:2710   3rd Qu.:1.0000   3rd Qu.:16.00   3rd Qu.:31.00
##  Max.   :3613   Max.   :1.0000   Max.   :18.00   Max.   :34.00
##
##      daded           momed          momdad14         sinmom14
##  Min.   : 0.00   Min.   : 0.00   Min.   :0.0000   Min.   :0.0000
##  1st Qu.: 8.00   1st Qu.: 8.00   1st Qu.:1.0000   1st Qu.:0.0000
##  Median : 9.94   Median :11.00   Median :1.0000   Median :0.0000
##  Mean   :10.00   Mean   :10.34   Mean   :0.7921   Mean   :0.1002
##  3rd Qu.:12.00   3rd Qu.:12.00   3rd Qu.:1.0000   3rd Qu.:0.0000
##  Max.   :18.00   Max.   :18.00   Max.   :1.0000   Max.   :1.0000
##
##      step14            reg661            reg662          reg663
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.000   Min.   :0.000
##  1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.000   1st Qu.:0.000
##  Median :0.00000   Median :0.00000   Median :0.000   Median :0.000
##  Mean   :0.03847   Mean   :0.04456   Mean   :0.155   Mean   :0.194
##  3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.000   3rd Qu.:0.000
##  Max.   :1.00000   Max.   :1.00000   Max.   :1.000   Max.   :1.000
##
##      reg664            reg665           reg666          reg667
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.000   Min.   :0.0000
##  1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:0.0000
##  Median :0.00000   Median :0.0000   Median :0.000   Median :0.0000
##  Mean   :0.06919   Mean   :0.2095   Mean   :0.093   Mean   :0.1102
##  3rd Qu.:0.00000   3rd Qu.:0.0000   3rd Qu.:0.000   3rd Qu.:0.0000
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.000   Max.   :1.0000
##
##      reg668           reg669            south66          work76
##  Min.   :0.000   Min.   :0.00000   Min.   :0.0000   Min.   :0.000
##  1st Qu.:0.000   1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:1.000
##  Median :0.000   Median :0.00000   Median :0.0000   Median :1.000
##  Mean   :0.031   Mean   :0.09355   Mean   :0.4127   Mean   :0.835
##  3rd Qu.:0.000   3rd Qu.:0.00000   3rd Qu.:1.0000   3rd Qu.:1.000
##  Max.   :1.000   Max.   :1.00000   Max.   :1.0000   Max.   :1.000
##
##     lwage76          black          smsa66r           wage76
##  Min.   :4.605   Min.   :0.00   Min.   :0.0000   Min.   :  25.0
##  1st Qu.:5.977   1st Qu.:0.00   1st Qu.:0.0000   1st Qu.: 393.0
##  Median :6.287   Median :0.00   Median :1.0000   Median : 535.0
##  Mean   :6.262   Mean   :0.23   Mean   :0.6427   Mean   : 576.1
##  3rd Qu.:6.564   3rd Qu.:0.00   3rd Qu.:1.0000   3rd Qu.: 708.0
##  Max.   :7.785   Max.   :1.00   Max.   :1.0000   Max.   :2404.0
##  NA's   :603                                     NA's   :596
##       kww              iq           marsta76
##  Min.   : 0.00   Min.   : 50.0   Min.   :1.000
##  1st Qu.:28.00   1st Qu.: 93.0   1st Qu.:1.000
```

```
##  Median :34.00   Median :104.0   Median :1.000
##  Mean   :33.49   Mean   :102.6   Mean   :2.357
##  3rd Qu.:40.00   3rd Qu.:113.0   3rd Qu.:4.000
##  Max.   :56.00   Max.   :156.0   Max.   :6.000
##  NA's   :70      NA's   :1143    NA's   :9
```

The summary() command gives us the minimum, maximum, 1st quartile, median, 3rd quartile, and mean of each column, or variable, in our data set. It also gives us the number of NAs, or Not Applicables, when they are present. NAs mean that there are missing observations in our data.

We can also run this command on just a single variable in our data set.

```
summary(data$ed76)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   12.00   13.00   13.23   16.00   18.00
```

Note that these numbers are the same as before. We use the $ symbol the specify subset of an object. In this case, we are saying we only want the ed76 variable of our data set.

We can see the dimensions of our data set using the dim() command.

```
dim(data) # Output takes the form row x column
```

```
## [1] 3613   27
```

So, we now know that our data has 3,613 rows and 27 columns, or 3,613 observations of 27 variables. Of course, you can also look at the Environment panel, which has that information on display next to the object name.

Perhaps you would like to know what your data looks like. To do that, we can use the head() command.

```
head(data)
```

```
##   id nearc4 ed76 age76 daded momed momdad14 sinmom14 step14 reg661 reg662
## 1  1      0    7    29  9.94 10.25        1        0      0      1      0
## 2  2      0   12    27  8.00  8.00        1        0      0      1      0
## 3  3      0   12    34 14.00 12.00        1        0      0      1      0
## 4  4      1   11    27 11.00 12.00        1        0      0      0      1
## 5  5      1   12    34  8.00  7.00        1        0      0      0      1
## 6  6      1   12    26  9.00 12.00        1        0      0      0      1
##   reg663 reg664 reg665 reg666 reg667 reg668 reg669 south66 work76  lwage76
## 1      0      0      0      0      0      0      0       0      1 6.306275
## 2      0      0      0      0      0      0      0       0      1 6.175867
## 3      0      0      0      0      0      0      0       0      1 6.580639
## 4      0      0      0      0      0      0      0       0      1 5.521461
## 5      0      0      0      0      0      0      0       0      1 6.591674
## 6      0      0      0      0      0      0      0       0      1 6.214608
##   black smsa66r wage76 kww  iq marsta76
## 1     1       1    548  15  NA        1
## 2     0       1    481  35  93        1
## 3     0       1    721  42 103        1
## 4     0       1    250  25  88        1
## 5     0       1    729  34 108        1
## 6     0       1    500  38  85        1
```

The head() command will show us the first couple rows of our data set. We can also look at the entire data set in RStudio by using the View() command (note the capitalization).

```
View(data)
```

This will open a new tab in our top left panel that will let us view our entire data set. Alternatively, we can click on the spreadsheet icon in the Environment panel next to our data set to accomplish the same thing.

We might also want to know the frequency at which a variable occurs in our data set. For that, we can use the table() command.

```
table(data$ed76)
```

```
##
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##    3    2    2    4    6   13   22   40   90   91  146  193 1194  323  309  206
##   16   17   18
##  532  179  258
```

The ed76 variable is the number of years of education a person had in 1976. In our data set, values range between 0 and 18. The top line of output are the values that appear in the data, and the second line is the number of times that value occurred. The third and fourth lines are simply the continuation of the table, since it was too long to fit all together. So, using this table, we can see that most people have 12 years of education, which is the equivalent of a high school diploma.

To find the mean and standard deviation of a variable, we can use the mean() and sd() command, respectively.

```
mean(data$ed76)
```

```
## [1] 13.2253
```

```
sd(data$ed76)
```

```
## [1] 2.749741
```

You might also be asked to find the correlation between variables. We can do this with the cor() command. To find the correlation of two variables:

```
cor(data$ed76, data$momed)
```

```
## [1] 0.4366935
```

So, a persons education is positively correlated with their mother's education.

Conditional Statements, Booleans, and Logic

Oftentimes we want to filter our data set for observations that meet specific criteria. Say we were only interested in those people in our data set that had less than a high school diploma. How could we find, say, their average earnings?

We could do this in base R without packages by using indexing. Indexing is denoted by square brackets after an object

```
mean(data$wage76[data$ed76 < 12], na.rm = T)
```

```
## [1] 436.6373
```

```
# the na.rm parameter removes missing values from the calculation
```

Logical Operators and Booleans

Above, we use the logical operator $<$ to filter our data. Please see the table below for the complete list of operators.

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

Booleans are a data type in R that can take the value TRUE or FALSE. There are three boolean operators: the & which means AND, the | which means OR, and the ! which means NOT.

This code works, but it is somewhat difficult to read. It is not immediately obvious what we are doing nor what we are filtering by. One of the most popular packages, the dplyr package, addresses this fact. Let's look at another way to do this.

```
install.packages("dplyr")
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
data %>%
  filter(ed76 < 12) %>%
  summarize(mean = mean(wage76, na.rm = T))
```

```
##       mean
## 1 436.6373
```

Here, it is much easier to see how we are sorting and what we are trying to accomplish. The %>% symbol is called a pipe operator, and it helps make code more readable. You can take it as meaning THEN. So, in the code above, we start with our data frame, THEN we filter it, and THEN we take the average of one of our filtered variables.

We can do more complex filtering as well. Say we want to find the average earnings of those people who have exactly 12 years of schooling (ed76 == 12) OR lived with a single mother at age 14 (sinmom14 == 1). Then, our code would be.

```
data %>%
  filter(ed76 == 12 | sinmom14 == 1) %>%
  summarize(mean = mean(wage76, na.rm = T))
```

```
##       mean
## 1 553.6265
```

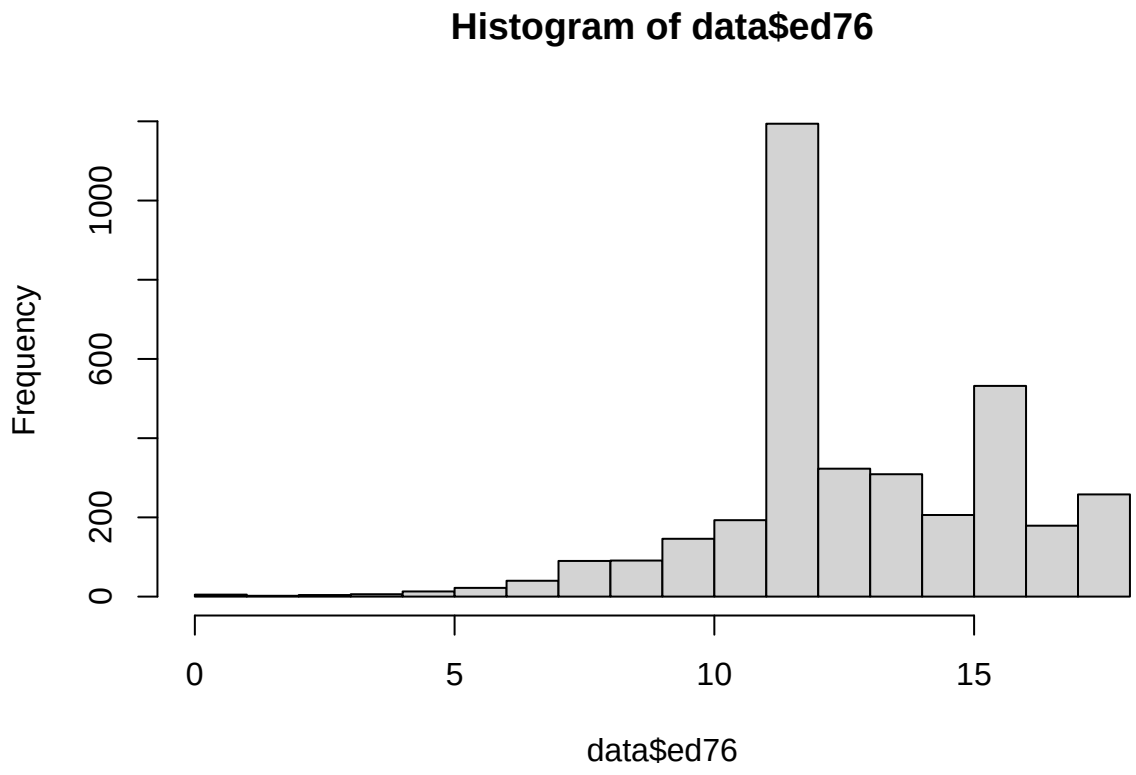
As you can imagine, this is just the tip of the iceberg when it comes to conditional statements and sorting. Using the dplyr package is optional, but extremely useful. A good resource to help with the dplyr package can be found by following this link: https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

Graphs

Sometimes, you'll be asked to create graphs of your data. While there exist packages specifically devoted to graphing, R has some native graphing commands.

Earlier, we created a frequency table using the table() command. We can also show this graphically using a histogram. To create a histogram, we can use the hist() function.

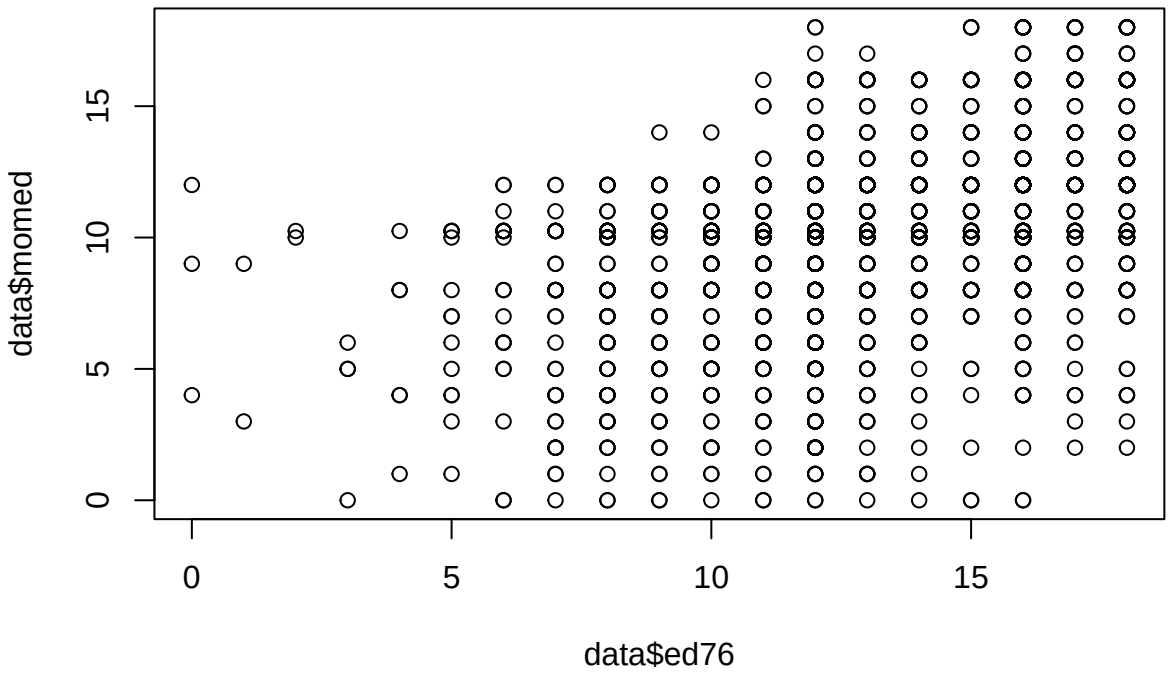
```
hist(data$ed76)
```



We can see that the most common value is again 12, verifying what we discovered earlier. Note that there are other commands that can format our histogram and make it more aesthetically pleasing, but we will skip this for now.

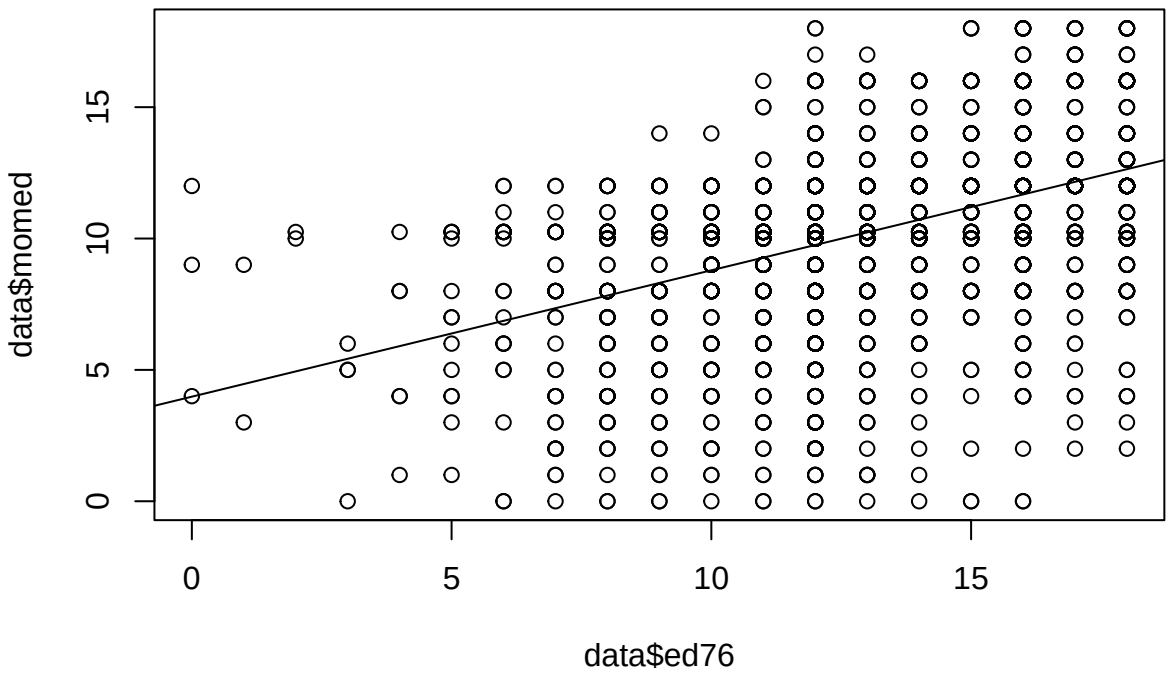
We can also plot scatterplots, which shows us the correlation between variables. Earlier, we used the cor() command to find the correlation between years of education and mother's education. Let's graph this using a scatterplot. To do so, we can use the plot() command.

```
plot(data$ed76, data$momed)
```



Sometimes, the trend is difficult to see, so we can add a regression line using the abline() command. This command uses a regression, which we will learn more about later in the course.

```
plot(data$ed76, data$momed)
abline(lm(data$momed ~ data$ed76))
```



From the trendline, we can see the positive relationship that we identified earlier.

If you are interested in upgrading your graphs, you can investigate the ggplot2 package.

Knitting Documents

When we are done with an assignment, we want to knit it into a document for submission. Make sure your R Script file is saved to your computer. Then, go to File -> Compile Report. The first time you click this button, there will be a prompt asking you to download some additional packages. Accept this, then wait for the packages to finish downloading. Next, a dialogue box will appear asking you to select a file format. Select HTML.

When you go to turn in your assignment, **turn in the HTML document that you just created!**

Conclusion

R has a rather steep learning curve, and this document is by no means comprehensive. An important part of coding is knowing how to look things up and apply those answers to your problem. Fortunately, there is an expansive set of resources available online for learning, troubleshooting, and debugging R. If you are interested in expanding your R knowledge or are running into difficulty on your assignments, I highly suggest you take advantage of those resources.