Learning R and R-Studio for Education and Social Science Research

Fernando Rodriguez and Hye Rin Lee

2021-07-26

Contents

Chapter 1

Introduction

We are looking forward to introducing you to the wonderful world of R and RStudio.

R is a very powerful statistical programming language that has several advantages over using Stata or SPSS. The most obvious is that R is completely free as opposed to paying a fee. Another great feature about R is that there are several add-on programs, such as the R-Studio, which helps us better manage multiple datasets at the same time and share reproducible code.

The downside of using RStudio is that it almost exclusively code-based, meaning that there are very limited point-and-click features. In order to get R to perform an analysis or plot a figure, you have to write several lines of code, which can feel like a big barrier if you are unfamiliar with programming concepts.

The goal of this book is to ease you into learning how to program in R and use the RStudio program to optimize your data workflow. What makes this book different from others is that we assume you have zero experience with using R and RStudio (and are also a bit intimidated by learning it!), so everything is explained as straightforward as possible. Moreover, the book will guide you through the entire process of analyzing educational data obtained from an online STEM course. This includes importing, inspecting, making decisions about your sample size, generating descriptive data, creating data visualizations, and using inferential statistics to draw conclusions from your sample. By the end of this book, you will have the necessary proficiency to use R and RStudio on your own research project from start to finish.

As you work your way through the chapters, you will find that programming in R is much easier than it looks. You will also appreciate how R-Studio helps us manage research projects. Even more exciting, once you get a good sense of some of the basics, you will soon begin tinkering with code and try things just for the sake of trying things out. That's where the real fun starts.

1.1 Who is this book for?

We wrote this book for people who do education and/or psychological research, who are at various levels in their careers, and who want a easy-to-follow book for learning R and RStudio. This includes undergraduate lab assistants who are conducting research for the first time, graduate students, faculty, and seasoned researchers who know how to use SPSS or Stata, but want to branch out to and further expand their skills.

While no experience with R or RStudio or coding is necessary, we do assume that you have a basic understanding of research methods and statistics.

1.2 Structure of the book

This book consists of three parts. `Part I` walks you through installing R, which is the actual program we need to have open in order to run code in RStudio, which is a graphical user interface (GUI) that helps us better manage our project files and datasets. We will then walk you through the most basic concepts surrounding the R programming language, as well as popular libraries. Libraries refer to a suite of features we can use in R, but are not part of the main R program. Finally, because we want you to see the immediate appeal of using R and RStudio, you will also write your first data visualization code.

`Part II` will guide you through importing, inspecting, and exploring your data. It is here you will learn all about the 'tidy' method for working with data. This 'tidy' method was developed by Hadley Wickham along with the RStudio team, and it refers to a principles for working with data.

`Part III` will help you understand how to conduct inferential statistics, and…..

1.3 Additional resources

While is this book provides a general introduction to using R and RStudio, we don't cover everything we think you should know about RStudio, so we recommend that you refer the following books:

1.4 About the authors

Fernando Rodriguez, Ph.D., is an assistant professor of teaching in the School of Education at the University of California, Irvine. He enjoys teaching various undergraduate-level courses and the graduate-level statistics course in the School of Education. His research focuses on learning analytics and higher-order thinking skills. Dr. Rodriguez earned his B.A. in Psychology from California State University, Northridge. He earned his Master's degree in Developmental Psychology and his Ph.D. in Educational Psychology from the University of Michigan, Ann Arbor.

Hye Rin Lee is a doctoral candidate in the School of Education at the University of California, Irvine, with a concentration in Human Development in Context. Hye Rin received her B.A. in Psychology and Sociology from Franklin and Marshall College. Hye Rin's research interests broadly focuses on academic motivation in school settings. Specifically, her research interests are situated in the intersections of STEM education, media technologies, and higher education. Outside of academia, she likes to watch movies and explore innovative places.

1.5 Acknowledgments

We would like to thank the National Science Foundation (Grant Number 1535300) and the Institute for Education Sciences (Grant Number R305B210006) for supporting the development of this book.

Chapter 2

Grounding Concepts

2.1 A Quick tour of R and R-Studio

Please watch the first 15 minutes of the Lesson 01 workshop video, as we give you a run down of these programs what all of these panes, tabs, and buttons do.

Click here to watch Lesson 01

2.1.0.1 *What's the difference between R and R-Studio?*

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.5     v dplyr   1.0.3
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Both programs are able to run R.

`R` is the original program. When we install R, we are installing two things: (1) the R programming language, and (2) a Graphical User Interface (GUI) that helps us work with the R-programming language, such as running code, opening and saving files. When you open R, you will notice that the GUI contains only a few buttons and icons. R looks very simple at first sight, but it does have all of the necessary tools you would need to work with data.

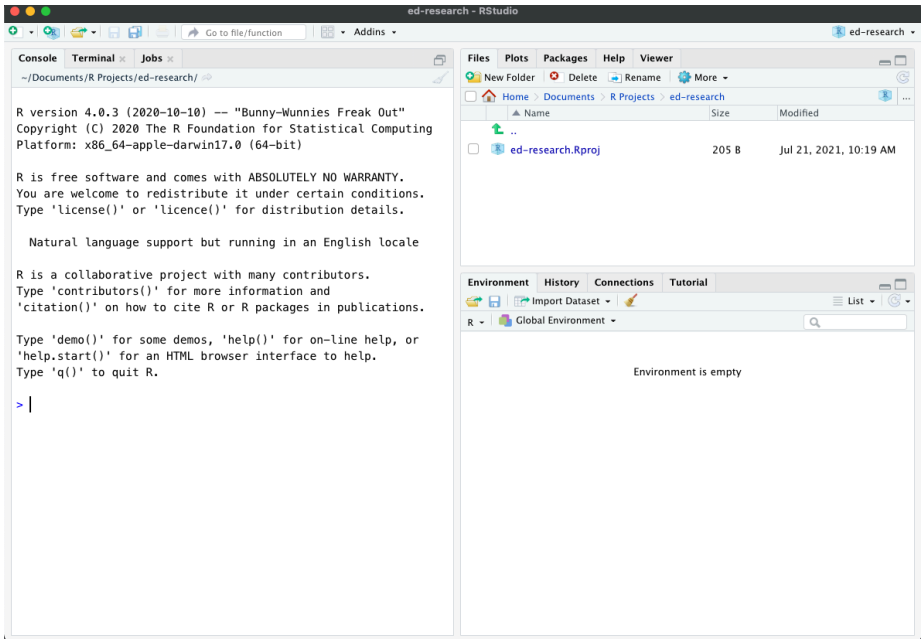
`R-Studio` is an add-on to R. It has many additional features that make working with the R-programming language much easier. As you can see from opening R-Studio, the GUI has several different panes, buttons, tabs, and icons.

R-Studio also has important tools that make it easy to implement open science practices in your work, like the ability to create notebooks that replicate your data analyses. It also has tools for uploading your work to code-sharing platforms, like Github.

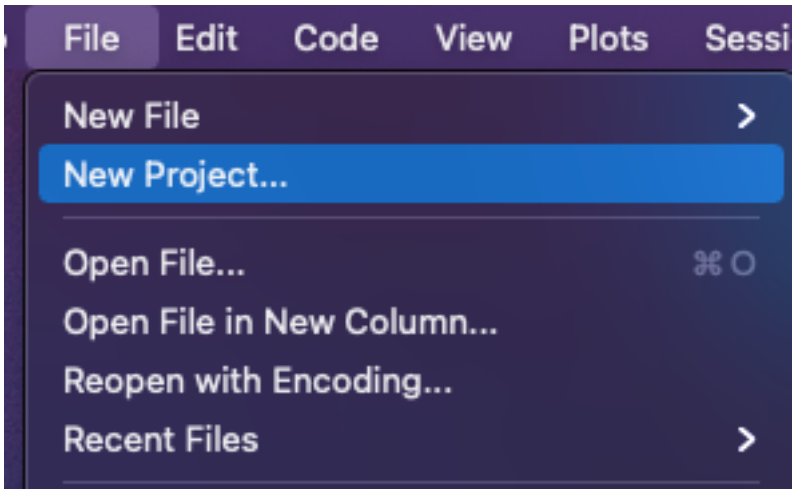
Because R-Studio is an add-on program, you need to ensure that you first install R on your computer. Note that when you open R-Studio, it already imports the R-programming language, so there is no need to open the R program.

We will use R-Studio exclusively for this book.

When you open R-Studio, you will see the following 3 panes



The `Console` pane is where you can type in lines of R code. This pane will also output the results from your code. For the sake of this book, we will primarily use the console to quickly check data, especially when we don't necessarily need to save the code. One important thing to understand about the Console pane is that it is temporary.



2.2 Running and saving code in R-Studio

The Console Pane, which by default appears on the bottom-left pane in R-Studio, is the interface for entering and running code.

For the purpose of this book, we will use the console to do quick data calculations, data checks, and experiment with code. We will primarily use R-Studio to write and execute our code, especially as it pertains to creating a data notebook.

2.2.1 R-Markdown Files

R-Markdown files is a document file that serves as a data notebook, where we can write text as well as lines of code. The benefit of using an R-Markdown file is that we can keep a record of everything we do, from importing our data, inspecting and cleaning variables, to analyzing and visualizing our data. This allows us to share our work with others in a completely transparent way. R-Markdown files do have some characteristics that look quite odd, but we'll address those in a bit.

2.2.2 Creating a New R-Markdown File

On the top-left corner in R-Studio, select File -> New File -> R-Markdown

2.2.3 R-Markdown Magic: Code Chunks

Let's get familiar with the concept of Code Chunks. Code chunks

This is specific feature of markdown (.Rmd) files, meaning that

2.3 Creating code chunks in R-Markdown

There are three ways to create a Code Chunk.

1. You can create a new chunk of code by typing the following:

   “‘`{r}`

   CODE HERE

   “‘

   The space in between the grave markers, “‘`{r}` `CODE HERE`“ is where we can write R code and calculations, such as 2 + 2.

2. You can insert a code chunk using the `+c` menu button, which appears directly above your R-Markdown document. To create a code chunk, click on the `+c` button, then select R.

3. You can also use the following keyboard shortcut

   - alt + command + i (mac)
   - control + alt + i (windows)

```
2 + 2
```

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

2.4 Running chunks of code

Now that you entered 2+ 2 in the code chunk, you can run this line of code by clicking on the green arrow to the right of the code chunk.

You can also use the following shortcuts to run the code within this code chunk:

command + enter (mac)

control + enter (windows)

2.5 The very basics

2.5.1 Simple Calculations

R works just like a calculator. You can do addition, subtraction, multiplication, etc. Here, we provide two examples, but you can experiment with calculations (+, -, *, /, ^, etc.) on the Console Pane.

Addition

```
2 + 2
```

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

Division

```
10/2
```

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

2.5.2 Objects & the Assignment Operator <-

Objects are the virtual space where we can temporarily store the data we load into R. When we want to load a .csv file into R, for example, we save it into an object. We can name these objects whatever we like, as long as it starts with a character string and does not contain special words or special characters that are exclusive to specific R commands or functions (more on this in later chapters).

Remember the simple calculations we just did? We can store those results into an object.

We do this by using the assignment operator `<-`

The assignment operator is an arrow `<-` (which is the `less than` sign and the `dash` sign). This is also what we mean by special characters—you cannot use `<-` for any other purpose in R.

Here's how it works.

Lets creating objects a, b, and c

```
a <- 2
b <- 10 + 2
c <- 2 + 2
```

2.5.3 Environment Pane in R-Studio

Notice that something happened to the environment pane. The environment name shows you the names of the objects we created. You will also see that the stored values are displayed to the right of the object name.

You may have also noticed that the results of a, b, and c, did not show up anywhere other than the environment pane. This is because when we use the assignment operator, we are telling R to save the results (and not displaying them).

2.6 Learning your first function: `print()`

```
print(a)
```

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

2.7 Objects & Functions

2.7.1 The Data Frame Object

here, I we are going to type `mtcars` in the code chunk below which is a dataframe that came pre-installed in R.

```
mtcars
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic       30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla    33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona     21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger  15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin       15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28        13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird  19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9         27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2     26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa      30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L    15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino      19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora     15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E        21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

2.7.2 Learning your first function in R: `str()`

If you want to see less rows you can use the `head()` function.

```
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

If you want to move the `mtcars` dataframe into the environment pane, you can duplicate it via the assignment command. Here, we'll save a copy of `mtcars` as `cars` and check the data using the `head()` function. Notice that I just added into the same chunk of code.

```
cars <- mtcars

head(cars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

2.8 View()

This allows us to view the actual raw data

```
View(mtcars)
```

2.9 A Note on Arguments

Notice that functions in R always have () beside them

head(mtcars)

In R, we put our arguments (which are things the function needs to run, and/or extra things we want the function to do) inside these parentheses.

2.10 Learning More About Libraries and Functions

If you want to see more information about what you can do with a library like ggplot2, you can put `?` in front of the name of the library.

If you want to know more about how to use a specific function put a ? in front of the function name

You can even do this with sub-functions, like `element_text()`

Chapter 3

Introducing Libraries + Building Your First Data Visualization Using *base-R* vs. A Library

3.1 About Libraries

As we mentioned in the first chapter, one of the largest advantage of RStudio is that it is free. How? One of the main reasons is that scientists from all over the world contribute to the software—making their code open access. In order to access, other researchers' packages, you have to install what is called a library. A library is like downloading various apps on your phone. Even though your phone comes with a camera app when you first purchase your iPhone (or any other smart phone), many of us who like to post on Instagram or just want good photos might download other camera or photo-related apps, such as Huji, Focos, Snow, etc. using the App Store (or Google Play Store) because these apps provide more functions compared to the original camera app. For example, although you can edit your photos using the basic camera app on your iPhone, such as crop or use certain filters, you cannot get the same vintage vibe as taking a picture using the Huji app. Translating this to RStudio language, *base R* is like the software of your smart phone with the basic apps installed from purchase, whereas *libraries* are the apps you can download from either the App Store or Google Play Store (depending on what phone you have).

To take advantage of these different "apps" that have been created by scholars across the world, we have to download the "app." Downloading the app, you will use the following code: install.packages(LIBRARY NAME)

There are so many different libraries to choose from. To see what libraries are out there, check out the RStudio website (https://www.rstudio.com/products/rpackages/) and we personally like to follow XXX on Twitter who often create different R-packages.

In this book, we will use XX libraries.

3.2 Introducing the *Tidyverse* Library

We will start with working with the *tidyverse* library. The *tidyverse* library consists of a few packages within the library, which is created by Hadley Wickham and his team. The current core *tidyverse* packages include: ggplot2, dplyr, tidyr, readr, purrr, tibble, stringr, and forcats. For more information about what each of these packages do, you can go onto their website: https://www.tidyverse.org/packages/

Another way to get more information about each of these libraries is through putting a `?` in front of library name. Lets try finding more information about the library, *ggplot2*, which is a library within *tidyverse*.

The first step involves making sure the library *ggplot2* is installed.

```
install.packages("ggplot2") # you can replace whatever is in the parentheses with anoth
```

If RStudio asks, "Do you want to install from sources the package which needs compilation? (Yes/no/cancel)," you should write yes on the console. This step of downloading libraries only needs to be done once—just like how you only need to download the app once on your smart phone. You can also add a number/hashtag sign to comment inside your code chunk. This number/hashtag sign tells RStudio to not run whatever is after the number/hashtag sign.

The second step is to open your library.

```
library(ggplot2) # you can replace whatever is in the parentheses with another library
```

This step should be done every time you make a new R-markdown file. You are asking R-Studio to retrieve the librar(ies) that are already installed on your RStudio. We recommend starting with opening all the librar(ies) you need.

The last step is to add a `?` in front of library name.

```
?ggplot2 # you can replace ggplot2 with any other library name
```

This step should be completed only after completing steps 1 and 2. However, you do not have to necessarily complete this step to use the library, in this case, ggplot2. The third step just allows you to see more information about the package.

[INSERT SCREENSHOT HERE?]

Because *tidyverse* is a collection of packages, one cool information is that you do not need to necessarily install all of the libraries separately (i.e., ggplot2, dplyr, tidyr, readr, purrr, tibble, stringr, and forcats). For instance, you do not need to run:

```
# downloading libraries
install.packages("ggplot2")
install.packages("dplyr")
install.packages("tidyr")
install.packages("readr")
install.packages("purrr")
install.packages("tibble")
install.packages("stringr")
install.packages("forcats")

# opening libraries
library(ggplot2)
library(dplyr)
library(tidyr)
library(readr)
library(purrr)
library(tibble)
library(stringr)
library(forcats)
```

Instead, you can run:

```
# downloading library
install.packages("tidyverse")
```

```
# opening library
library(tidyverse)
```

This shortcut allows you to the same packages (or libraries) as the code chunk above.

3.3 Creating Your First Data Visualization Using base-R vs. ggplot2

As we discussed in Section 3.1, an advantage of using a library is that it provides you with more functions. Again, think about using the camera app that comes with your smart phone vs. an app you can download from either the App Store or Google Play Store. In this section, we will use base-r (i.e., like a camera app that comes with your smart phone) and ggplot2 (i.e., like a camera app that you can download from either the App Store or Google Play Store).

3.3.1 Creating a Scatterplot Using *base-r*

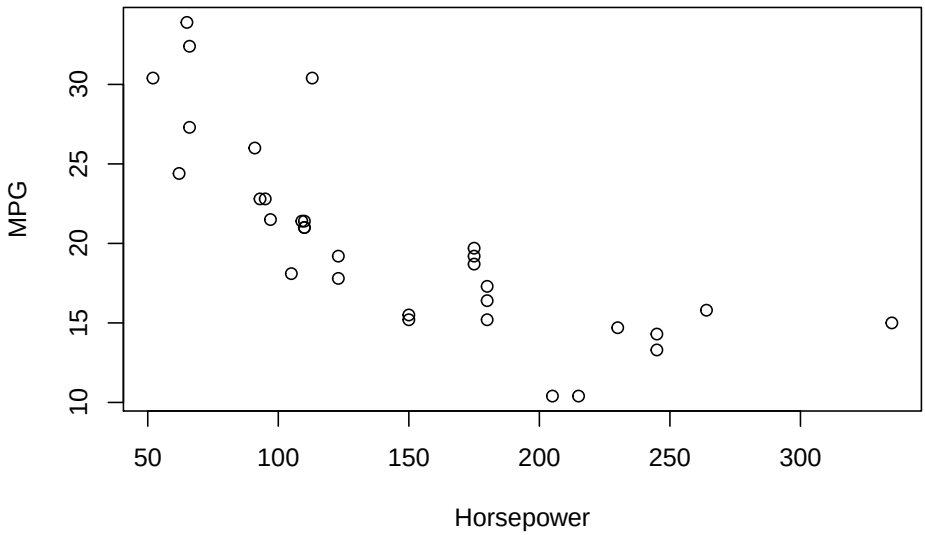
To create a scatterplot using *base-r*, we will use the `plot` function using the `mtcars` data. This dataset is an open access dataset within R-Studio. The dataset is built-in R and mtcars stands for the Motor Trend Car Road Tests.

Lets first open the dataset.

```
data("mtcars")
```

Then we can use the `plot` function to create the scatterplot. We have to add the dataset_name$variable_name.

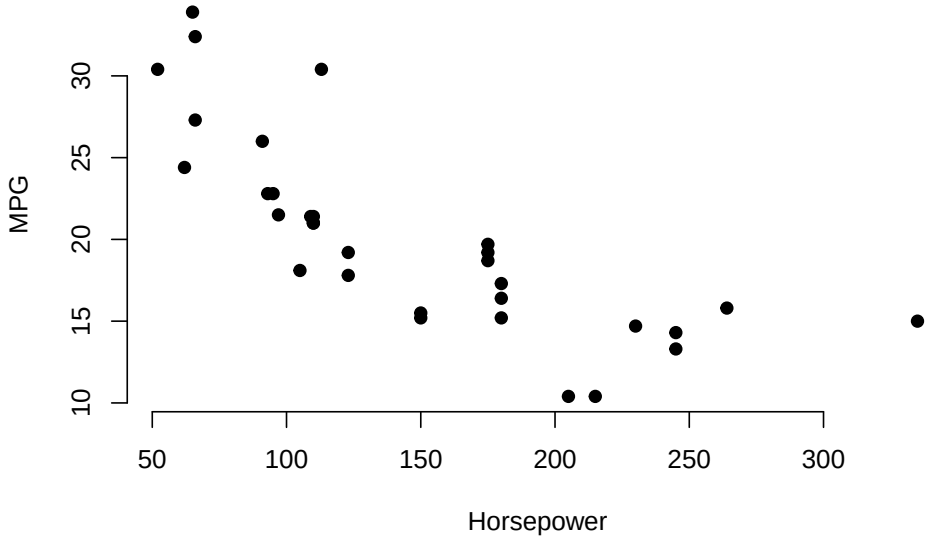
```
plot(mtcars$hp, mtcars$mpg,
     xlab = "Horsepower", ylab = "MPG")
```



This code allowed us to create a scatterplot where the x-axis is horsepower and the y-axis is MPG (i.e., miles per gallon). The format of the code is plot(variable name of x-axis, variable name of y-axis, xlab = "x-axis title label", ylab = "y-axis of title label").

Using the `plot` function, you can also change the plotting symbols. For example, you can make the plotting symbol a non-filled rectangle, a non-filled circle, an X, a medium-sized filled circle, etc. To receive a full-chart on all the possibilities in *base-r*, see the following resource: http://www.sthda.com/english/wiki/r-plot-pch-symbols-the-different-point-shapes-available-in-r. You can also include or exclude the boarder around the scatterplot. Lets try creating a scatterplot with a medium-sized filled circle with no boarders around the edges.

```
plot(mtcars$hp, mtcars$mpg,
     xlab = "Horsepower", ylab = "MPG",
     pch = 19, frame = FALSE)
```



The only difference between the code above and this code is the addition of pch = 19 and frame = FALSE. pch = 19 refers to creating a scatterplot with a medium-sized filled circle and frame = FALSE refers to creating a scatterplot with no boarders.

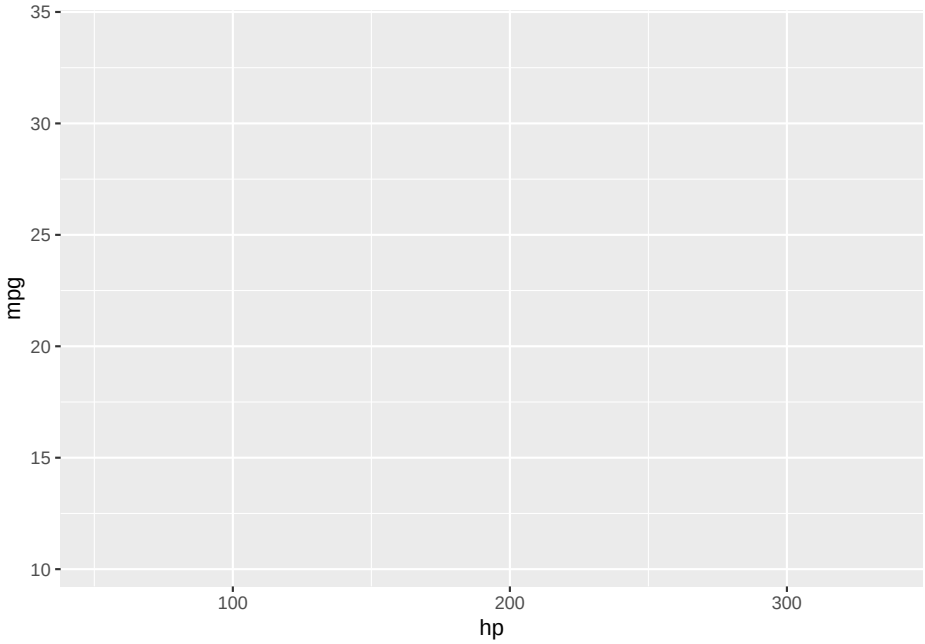
3.3.2 Creating a Scatterplot Using *ggplot2*

Using the same dataset, `mtcars`, we will use *ggplot2* to graph a scatterplot. As mentioned in the "Introducing the *Tidyverse* Library" section, we want to make sure that our library, *ggplot2* is downloaded and loaded.

```
library("ggplot2")
```

Then similar to the scatterplot we made using *base-r*, we will plot how miles per gallon (i.e., MPG) is related to horsepower (i.e., hp). We will set up the parameters by using the `aes()` function, which stands for aesthetic. The x-axis will be `hp` and the y-axis will be `mpg`.

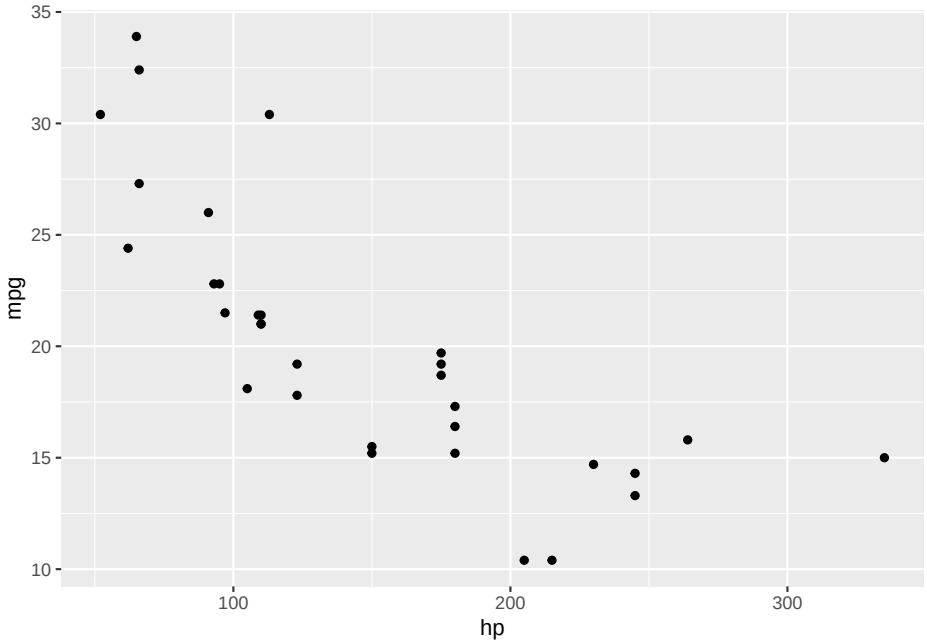
```
ggplot(mtcars,
       aes(x = hp, y = mpg))
```



You can also add new by using other functions that are part of the ggplot library. We can add functions by using the `+`.

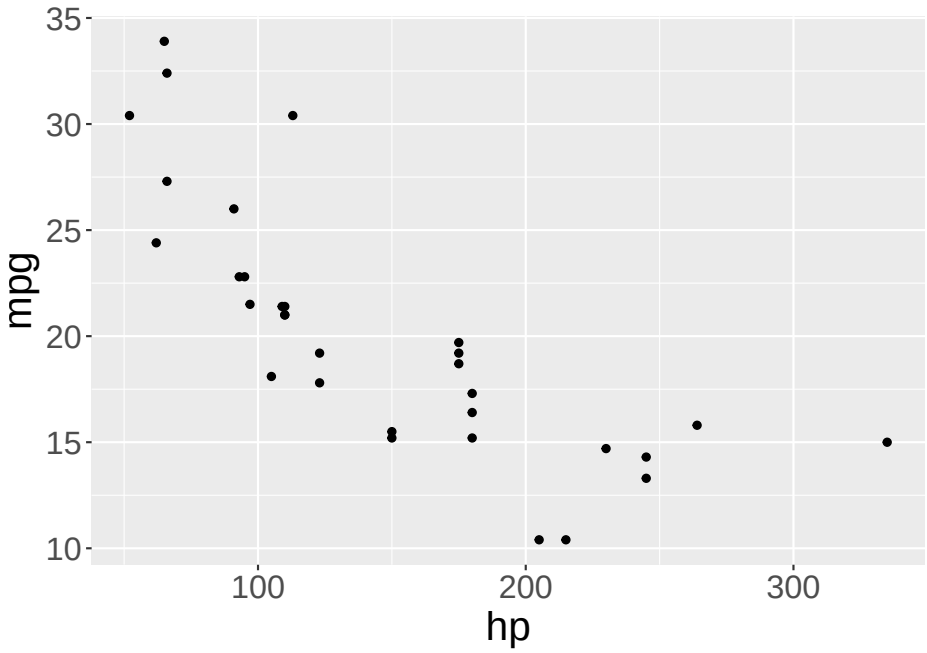
Lets try using the `geom()` function to state the kind of graph we want to create. Because we want to create a scatterplot, we are going to use the function `geom_point`. No arguments are required within the parentheses for `geom()`.

```
ggplot(mtcars, aes(x = hp, y = mpg)) +
      geom_point()
```



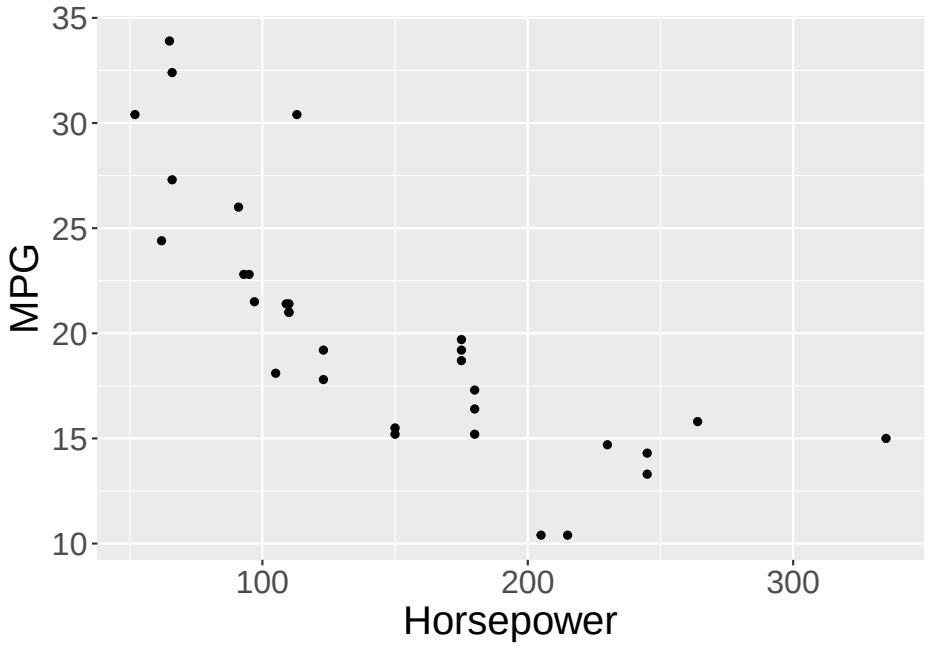
We can also change the size of the text. For example, we will use the `theme` function to change the size of the text to size 20.

```
ggplot(mtcars, aes(x = hp, y = mpg)) +
      geom_point() +
      theme(text = element_text(size = 20))
```



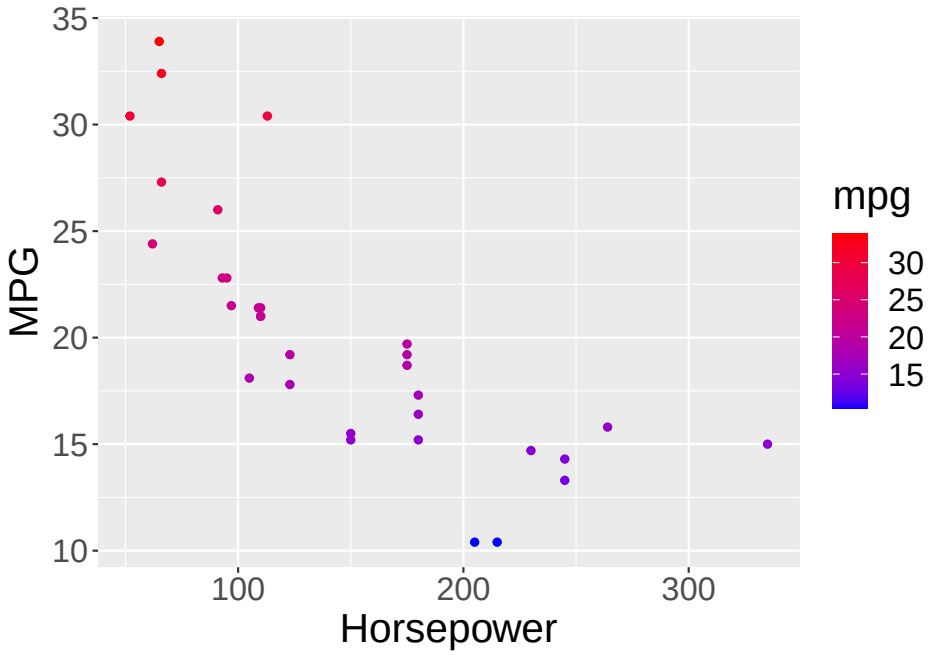
Moreover, we can use the `labs` function to label our x- and y-axis.

```
ggplot(mtcars, aes(x = hp, y = mpg)) +
      geom_point() +
      theme(text = element_text(size = 20)) +
      labs(x = "Horsepower", y = "MPG")
```



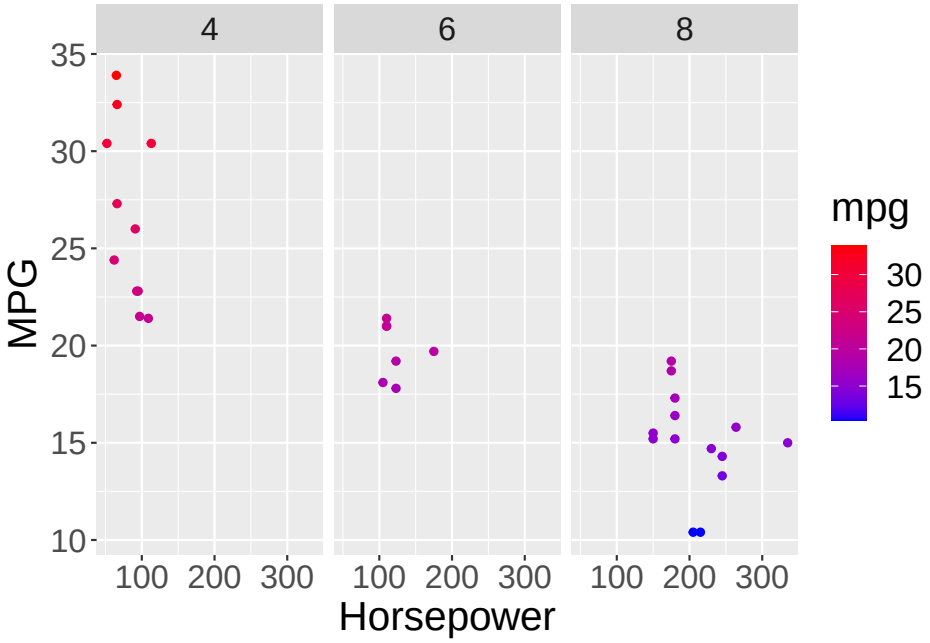
Another function that you can do with *ggplot2* is add a color gradient on a specific variable. For example, lets say that you want low MPG to be blue and high MPG to be red. In order to make this function work, we have to (1) state which variable you want to color, which in this case, is MPG and (2) go back to the `aes` function and write an additional argument. All arguments are separated by `,`.

```
ggplot(mtcars, aes(x = hp, y = mpg, color = mpg)) +
      geom_point() +
      theme(text = element_text(size = 20)) +
      labs(x = "Horsepower", y = "MPG") +
      scale_color_gradient(low = "blue", high = "red")
```



Finally, we will go over the function, `facet_grid`, which allows you to create graphs by a group. Lets try creating a scatterplot split by the variable, `cyl` (i.e., the number of cylinders).

```
ggplot(mtcars, aes(x = hp, y = mpg, color = mpg)) +
      geom_point() +
      theme(text = element_text(size = 20)) +
      labs(x = "Horsepower", y = "MPG") +
      scale_color_gradient(low = "blue", high = "red") +
      facet_grid(.~ cyl)
```


4
6
8
35
30
25
20
15
10
MPG
100
200
300
Horsepower
mpg
30
25
20
15

Chapter 4

Project Workflow

For this lesson, we'll understand how to develop an organized workflow.

4.0.1 Load Libraries

This is the first thing you want to do when authoring R-Markdown files. For this lesson, we'll use `ggplot2`.

```
library(ggplot2)
```

4.0.2 Creating Code Chunks

For mac: alt + command + i

For windows: control + alt + i

4.0.3 Creating Comments

use `#` to add comments within code chunks.

```
# I can write anything here about our object a.
a <- 2 + 2 # this is also allowed
```

4.0.4 Commenting out multiple lines of code

For Mac and Windows: control + shift + c For Mac: command + shift + c

```
a <- 2 + 2
a <- 2 + 2
a <- 2 + 2
a <- 2 + 2
a <- 2 + 2
```

Chapter 5

Data cleaning

Testing testing 1 2 3

```
objecta <- 2 + 2
print(objecta)
```

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

Chapter 6

More Data Cleaning

Chapter 7

Descriptive tables