Statistical Computing

Isaac Quintanilla Salinas

Table of contents

Welcome

Preface

This is a book created to be used for a statistical computing course at the undergraduate level.

Part I

R Programming

1 Basic R Programming

1.1 Introduction

This chapter focuses on the basics of R programming. While most of your statistical analysis will be done with R functions, it is important to at least have an idea of what is going on. Additionally, we will cover other topics that you may or may not need to know. The topics we will cover are:

1. Basic calculations in R
2. Types of Data
3. R Objects

There are many other topics that should be covered, but it may be unnecessary. If you are interested in those topics, I recommend using the 'swirl' package.

1.2 Basic Calculations

This section focuses the basic calculation that you can do in R. Essentially, we look at how R can be used as a calculator. This is done by using different operators in R. An operator is a symbol that tells R to do something. Some common operators are `+`,`-`, and `*` which corresponds to addition, subtraction, and division.

1.2.1 Calculator

1.2.1.1 Addition

To add numbers in R, all you need to use the `+` operator. For example 2+2=4. When you type it in R you have:

```
2+2
```

```
[1] 4
```

When you ask R to perform a task, it prints out the result of the task. As we can see above, R prints out the number 4.

To add more than 2 numbers, you can simply just type it in.

```
2+2+2
```

```
[1] 6
```

This provides the number 6.

1.2.1.2 Subtraction

To subtract numbers, you need to use the `-` operator. Try `4-2`:

```
4-2
```

```
[1] 2
```

Try 4-6-4

```
4-6-4
```

```
[1] -6
```

Notice that you get a negative number.

Now try `4+4-2+8`:

```
4+4-2+8
```

```
[1] 14
```

1.2.1.3 Multiplication

To multiply numbers, you will need to use the `*` operator. Try `4*4`:

```
4*4
```

```
[1] 16
```

1.2.1.4 Division

To divide numbers, you can use the `/` operator. Try `9/3`:

```
9/3
```

```
[1] 3
```

1.2.1.5 Exponents

To exponentiate a number to the power of another number, you can use the `^` operator. Try `2^5`:

```
2^5
```

```
[1] 32
```

If you want to take e to the power 2, you will use the `exp()` function. Try `exp(2)`:

```
exp(2)
```

```
[1] 7.389056
```

1.2.1.6 Roots

To take the n-th root of a value, use the `^` operator with the `/` operator to take the n-th root. For example, to take the 5th-root of 32, type `32^(1/5)`:

```
32^(1/5)
```

```
[1] 2
```

1.2.1.7 Logarithms

To take the natural logarithm of a value, you will use the `log()` function. Try `log(5)`:

```
log(5)
```

```
[1] 1.609438
```

If you want to take the logarithm of a different base, you will use the `log()` function with `base` argument. We will discuss this more in section 7 of this chapter.

1.2.2 Comparing Numbers

Another important part of R is comparing numbers. When you compare two numbers, R will tell you if that is true or false. We will talk about some of the basic comparisons and their operators.

1.2.2.1 Less than/Greater than

To check if one number is less than or greater than another number, you will use the `>` or `<` operators. Try `5>4`:

```
5 > 4
```

```
[1] TRUE
```

Notice that R states it's true. It evaluates the expression and tells you if it's true or not. Try `5<4`:

```r
5 < 4
```

```
[1] FALSE
```

Notice that R tells you it is false.

1.2.2.2 Less than or equal to/Greater than or equal to

To check if one number is less than or equal to/greater than or equal to another number, you will use the `>=` or `<=` operators. Try `5>=5`:

```r
5 >= 5
```

```
[1] TRUE
```

Try `5>=4`:

```r
5 >= 4
```

```
[1] TRUE
```

Try `5<=4`

```r
5 <= 4
```

```
[1] FALSE
```

1.2.2.3 Equals and Not Equals

To check if 2 numbers are equal to each other, you can use the `==` operator. Try `3==3`:

```r
3 == 3
```

```
[1] TRUE
```

Try `4==3`

```r
3 == 4
```

```
[1] FALSE
```

Another way to see if 2 numbers are not equal to each other, you can use the `!=`. Try `3!=4`:

```r
3 != 4
```

```
[1] TRUE
```

Try `3!=3`:

```r
3 != 3
```

```
[1] FALSE
```

You may be asking why use `!=` instead of `==`. They both provides similar results. Well the reason is that you may need the 'TRUE' output for analysis. One is only true when they are equal, while the other is true when they are not equal.

1.2.3 Help

The last operator we will discuss is the help operator `?`. If you want to know more about anything we talked about you can type `?` in front of a functiona and a help page will pop-up in your browser or in RStudio's 'Help' tab. For example you can type `?Arithmetic` or `?Comparison`, to review what we talked about. For other operators we didn't talk about use `?assignOps` and `?Logic`.

1.3 Types of Data

In R, the type of data, also known as class, that we are using dictates how the programming works. For the most part, users will use 'numeric','logical', 'POSIX' and 'character' data types. Other types of data you may encounter are 'integer', 'complex', and 'raw'. These types of data are rarely used. To obtain more information on them, use the `?` operator.

1.3.1 Numeric

The numeric class is the data that are numbers. Almost every analysis that you use will be based on the numeric class. To check if you have a numeric class, you just need to use the `is.numeric()` function. For example, try `is.numeric(5)`:

```
is.numeric(5)
```

```
[1] TRUE
```

Notice that when you input an number into R, it automatically changes it to a numeric class. R is changes data to the class that it most likely needs to be. Now this is great because you do not need to do anything on your end. Howerver, if you need a different class, you will need to change it.

1.3.2 Logical

A logical class are data where the only value is 'TRUE' or 'FALSE'. Sometimes the data is coded as 1 for 'TRUE' and 0 for 'FALSE'. The data may also be coded as 'T' or 'F'. To check if data belongs in the logical class, you will need the `is.logical()` function. Try `is.logical(3<4)`:

```
is.logical(3 < 4)
```

```
[1] TRUE
```

Remember when we ran `3<4` in the previous section. The output was 'TRUE'. Now R is checking whether the output is of a logical class. Since it it, R returns 'TRUE'. Now try `is.logical(3>4)`:

```
is.logical(3 > 4)
```

```
[1] TRUE
```

The output is 'TRUE' as well even though the condition `3>4` is 'FALSE'. Since the output is a logical data type, it is a logical variable.

1.3.3 POSIX

The POSIX class are date-time data. Where the data value is a time component. The POSIX class can be very complex in how it is formatted. IF you would like to learn more try `?POSIXct` or `?POSIClt`. First, lets run `Sys.time()` to check what is today's data and time:

```
Sys.time()
```

```
[1] "2022-12-24 00:20:24 PST"
```

Now lets check if its of POSIX class, you can use the `class()` function to figure out which class is it. Try `class(Sys.time())`:

```
class(Sys.time())
```

```
[1] "POSIXct" "POSIXt"
```

1.3.4 Character

A character value is where the data values follow a string format. Examples of characters values are letters, words and even numbers. A character value is any value surrounded by quotation marks. For example, the phrase "Hello World!" is considered as one character value. Another example if you data is coded with the actual words "yes" or "no". To check if you have character data, use the `is.character()` function. Try `is.character("Hello World!")`:

```
is.character("Hello World!")
```

```
[1] TRUE
```

Notice that the output says 'TRUE'. Character values can be created with single quotations. Try `is.character('Hello World!')`:

```
is.character('Hello World!')
```

```
[1] TRUE
```

1.3.5 Integers

Integers are just whole numbers for the most part. To create an integer, type the letter 'L' after a number. To check if you are using integer data, use the `is.integer()` function. Try `is.integer(5L)`:

```r
is.integer(5L)
```

```
[1] TRUE
```

1.3.6 Complex Numbers

Complex numbers are data values where there is a real component and an imaginary component. The imaginary component is a number multiplied by $i = \sqrt{-1}$. To create a complect number, use the `complex()` function. To check if a number is complex, use the `is.complex()` function. Try the following to create a complex number `complex(1,4,5)`:

```r
complex(1, 4, 5)
```

```
[1] 4+5i
```

Now try `is.complex(complex(1,4,5))`:

```r
is.complex(complex(1, 4, 5))
```

```
[1] TRUE
```

1.3.7 Raw

You will probably never use raw data. I have never used raw data in R. To create a raw value, use the `raw()` or `charToRaw()` functions. Try `charToRaw('Hello World!')`:

```r
charToRaw('Hello World!')
```

```
 [1] 48 65 6c 6c 6f 20 57 6f 72 6c 64 21
```

To check if you have raw data, use the `is.raw()` function. Try `is.raw(charToRaw('Hello World!'))`:

```r
is.raw(charToRaw('Hello World!'))
```

```
[1] TRUE
```

1.3.8 Missing

The last data class in R is missing data denoted as `NA`. Whenever you see `NA` in any of the analysis you see, it means that the data is missing. To check if you have missing data, use the `is.na()` function. Try `is.na(NA)`:

```r
is.na(NA)
```

```
[1] TRUE
```

1.4 R Objects

R objects are where most of the statistical analysis is conducted on. An R object can be thought of as a container of data. For the most part, you will only use a data frame (or tibble) for your data analysis. However, it is always a good idea to to have some basic understanding of the other R objects.

1.4.1 Assigning objects

To create an R object, all we need to do is assign data to a variable. The variable is the name of the R object. it can be called anything, but you can only use alphanumeric values, underscore, and periods. To assign a value to a variable, use the `<-` operator. This is known a left assignment. Kinda like an arrow pointing left. Try assigning 9 to 'x' (`x<-9`)':

```r
x <- 9
```

To see if `x` contains 9, type `x` in the console:

```r
x
```

```
[1] 9
```

Now `x` can be treated as data and we can perform data analysis on it. For example, try squaring it:

```
x^2
```

```
[1] 81
```

You can use any mathematical operation from the previous sections. Try some other operations and see what happens.

The output R prints out can be stored in a variable using the asign operator, `<-`. Try storing `x^3` in a variable called `x_cubed`:

```
x_cubed <- x^3
```

To see what is stored in `x_cubed` you can either type `x_cubed` in the console or use the `print()` function with 'x_cubed' inside the paranthesis.

```
x_cubed
```

```
[1] 729
```

```
print(x_cubed)
```

```
[1] 729
```

1.4.2 Vectors

A vector is a set data values of a certain length. The R object `x` is considered as a numerical vector (because it contains a number) with the length 1. To check, try `is.numeric(x)` and `is.vector(x)`:

```
is.numeric(x)
```

```
[1] TRUE
```

```
is.vector(x)
```

```
[1] TRUE
```

Now let's create a logical vector that contains 4 elements (have it follow this sequence: T,F,T,F) and assign it to `y`. To create a vector use the `c()` function and type all the values and seperating it with columns. Type `y<-c(T,F,T,F)`:

```
y <- c(T, F, T, F)
```

Now, lets see how `y` looks like. Type `y`:

```
y
```

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

Now lets see if it's a logical vector:

```
is.logical(y)
```

```
[1] TRUE
```

```
is.vector(y)
```

```
[1] TRUE
```

Fortunately, this vector is really small to count how many elements it has, but what if the vector is really large? To find out how many elements a vector has, use the `length()` function. Try `length(y)`:

```
length(y)
```

```
[1] 4
```

The `c()` function allows you to put any data type and as many values as you wish. The only condition of a vector is that it must be the same data type.

1.4.3 Matrices

A matrix can be thought as a square or rectangular grid of data values. This grid can be constructed in any shape. Similar to vectors they must contain the same data type. The size of a matrix is usually denoted as $n \times k$, where n represents the number of rows and k represents the number of columns. To get a rough idea of how a matrix may look like, type `matrix(rep(1,12),nrow=4,ncol=3)`[1]:

```
matrix(rep(1, 12), nrow = 4, ncol = 3)
```

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

Notice that this is a 4×3 matrix. Each element in the matrix has the value 1. Now try this `matrix(rbinom(12,1.5),nrow=4,ncol=3)`[2]:

```
matrix(rbinom(12, 1, .5), nrow = 4, ncol = 3)
```

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

Your matrix may look different, but that is to be expected. Notice that some elements in a matrix are 0's and some are 1's. Each element in a matrix can hold any value.

Constructing a matrix can be a bit difficult to do because the data values may need to be arranged in a certain way. Notice that I used the `matrix()` function to create the matrix. The examples above contain other components in the function that we will discuss later.

[1] The function `rep()` creates a vector by repeating a value for a certain length. `rep(1,12)` creates a vector of length 12 with each element being 1

[2] The `rbinom()` function generates binomial random variables and stores them in a vector. `rbinom(12,1,5)` This creates 12 random binomial numbers with parameter $n = 1$ and $p = 0.5$.

1.4.4 Arrays

Matrices can be considered as a 2-dimensional block of numbers. An array is an n-dimensional block of numbers. While you may never need to use an array for data analysis. It may come in handy when programming by hand. To create an array, use the `array()` function. Below is an example of a $3 \times 3 \times 3$ with the numbers 1, 2, and 3 representing the 3rd dimension stored in an R object called `first_array`[3].

```
(first_array <- array(c(rep(1, 9), rep(2, 9), rep(3, 9)),
                      dim=c(3,3,3)))
```

```
, , 1

     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    1    1    1

, , 2

     [,1] [,2] [,3]
[1,]    2    2    2
[2,]    2    2    2
[3,]    2    2    2

, , 3

     [,1] [,2] [,3]
[1,]    3    3    3
[2,]    3    3    3
[3,]    3    3    3
```

1.4.5 Data Frames

Data frames seems like a data set that you may encounter in an excel file. However, there are a couple of differences. First, each row represents an observation, and each column represents a characteristic of the observation. Additionally, each column in a data frame will be the same data type. To get an idea of what a data frame looks like, try `head(iris)`:

[3] Notice the code is surrounded by parenthesis. This tells R to store the array and print out the results. You can surround code with parenthesis evertime you create an object to also print what is stored.

```r
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

The `head()` function just tells R to only print the top few components of the data frame.

Now try `tail(iris)`:

```r
tail(iris)
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
145          6.7         3.3          5.7         2.5 virginica
146          6.7         3.0          5.2         2.3 virginica
147          6.3         2.5          5.0         1.9 virginica
148          6.5         3.0          5.2         2.0 virginica
149          6.2         3.4          5.4         2.3 virginica
150          5.9         3.0          5.1         1.8 virginica
```

The `tail()` function provides the last 6 rows of the data frame.

1.4.6 Lists

To me a list is just a container that you can store practically anything. It is compiled of elements, where each element contains an R object. For example, the first element of a list may contain a data frame, the second element may contain a vector, and the third element may contain another list. It is just a way to store things.

To create a list, use the `list()` function. Create a list compiled of first element with the mtcars data set, second element with a vector of zeros of size 4, and a matrix 3×3 identity matrix[4]. Store the list in an object called `list_one`:

[4] An identity matrix is a matrix where the diagonal elements are 1 and the non-diagonal elements are 0

```r
list_one <- list(mtcars, rep(0, 4),
                 diag(rep(1, 3)))
```

Type `list_one` to see what pops out:

```r
list_one
```

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

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

Each element in the list is labeled as a number. It is more useful to have the elements named. An element is named by typing the name in quotes followed by the `=` symbol before your object in the `list()` function (`mtcars=mtcars`).

```r
list_one <- list(mtcars = mtcars,
                 vector = rep(0, 4),
                 identity = diag(rep(1, 3)))
```

Here I am creating an object called `list_one`, where the first element is `mtcars` labeled `mtcars`, the second element is a vector of zeros labeled `vector` and the last element is the identity matrix labeled `identity`.'

Now create a new list called `list_two` and store `list_one` labeled as `list_one` and `first_array` labeled as `array`.

```r
(list_two <- list(list_one = list_one,
                  array = first_array))
```

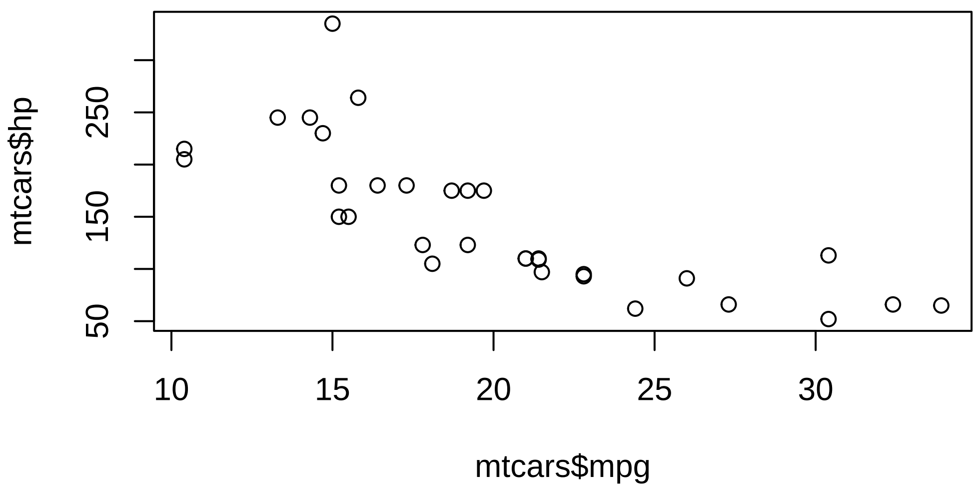
```
$list_one
$list_one$mtcars
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    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

$list_one$vector
[1] 0 0 0 0

$list_one$identity
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1


$array
, , 1

     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    1    1    1

, , 2
```

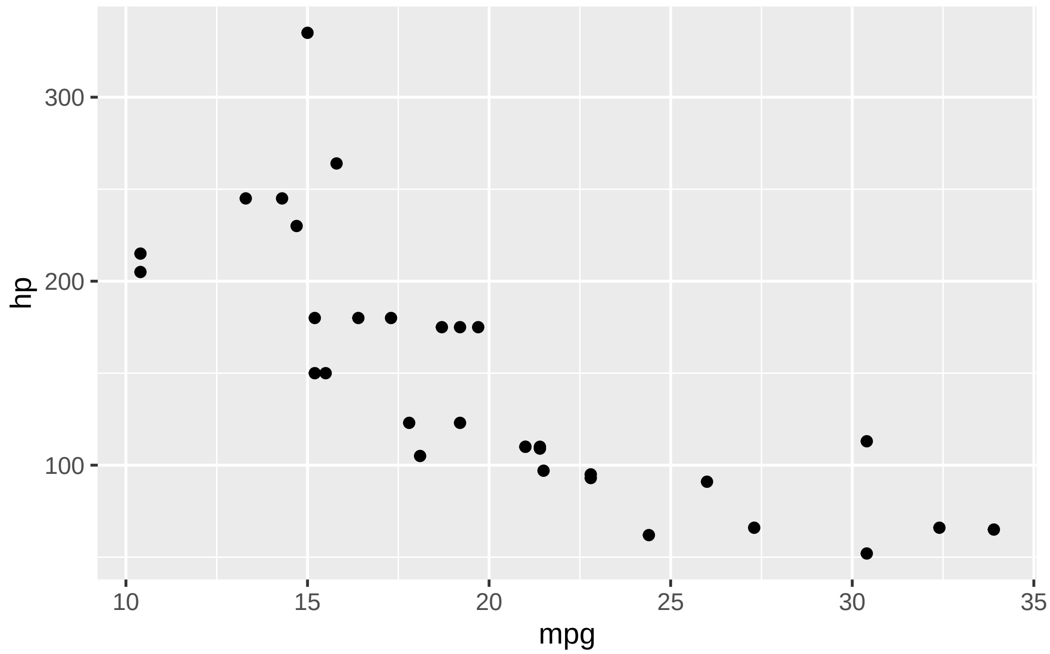
```
     [,1] [,2] [,3]
[1,]    2    2    2
[2,]    2    2    2
[3,]    2    2    2

, , 3

     [,1] [,2] [,3]
[1,]    3    3    3
[2,]    3    3    3
[3,]    3    3    3
```

1.5 R Packages

```r
plot(mtcars$mpg, mtcars$hp)

## Loading Packages
library(ggplot2)
```




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




```
## Installing Packages
#install.packages("rmarkdown")
```

2 Control Flow

In the Section 1.4, we discussed about different types of R objects. For example, a vector can be a certain data type with a set number of elements. Here we construct a vector called `x` increasing from -5 to 5 by one unit:

```
(x <- -5:5)
```

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

The vector `x` has 11 elements. If I want to know what the 6th element of `x`, I can index the 6th element from a vector. To do this, we use `[]` square brackets on `x` to index it. For example, we index the 6th element of `x`:

```
x[6]
```

```
[1] 0
```

When ever we use `[]` next to an R object, it will print out the data to a specific value inside the square brackets. We can index an R object with multiple values:

```
x[1:3]
```

```
[1] -5 -4 -3
```

```
x[c(3,9)]
```

```
[1] -3  3
```

Notice how the second line uses the `c()`. This is necessary when we want to specify non-contiguous elements. Now let's see how we can index a matrix

2.1 If/Else Statements

2.2 for Loops

2.3 while Loops

```
# Control Flow: if else statements
x <- rnorm(1)

## Logical Statements

x > 0
```

```
[1] TRUE
```

```
## if statements

if (x > 0) {
  print("Positive")
}
```

```
[1] "Positive"
```

```
## else statements

if (x > 0){
  print("Positive")
} else {
  print("Non-Positive")
}
```

```
[1] "Positive"
```

```
## Example 2
y <- rnorm(1)
```

```
if (y > 0){
  print("Positive")
  print(y)
  mean(y)
} else {
  print("Non-Positive")
  print(y)
  length(y)
}
```

```
[1] "Positive"
[1] 0.6449698
```

```
[1] 0.6449698
```

```
# Control Flow: else if statements
(x <- sample(-1:1,1))
```

```
[1] -1
```

```
## Logical Statements

x > 0
```

```
[1] FALSE
```

```
## if statements

if (x > 0) {
  print("Positive")
}

## else if statements

if (x > 0) {
  print("Positive")
```

```r
} else if (x < 0) {
  print("Negative")
} else {
  print("Zero")
}
```

```
[1] "Negative"
```

```r
## Example 2
y <- sample(-1:1,1)


if (y > 0){
  print("Positive")
} else if (y < 0) {
  print("Negative")
} else {
  print("Zero")
}
```

```
[1] "Negative"
```

```r
# Control Flow: break & next function -----

## Function
err_fx <- function(x){
  if (x>0){
    return(x)
  } else {
    stop("x is not positive")
  }
}

(y <- rnorm(1))
```

```
[1] -1.233007
```

```r
# err_fx(y)

## Loop Example
x <- rnorm(100)
loop <- c()
for (i in seq_along(x)) {
  try_err <- try(err_fx(x[i]), silent = T)
  if (inherits(try_err, "try-error")){
    loop[i] <- 0
  } else {
    loop[i] <- try_err
  }
}


## Break -----
x <- rnorm(100)
loop <- c()
for (i in seq_along(x)) {
  try_err <- try(err_fx(x[i]), silent = T)
  if (inherits(try_err, "try-error")){
    break
  } else {
    loop[i] <- try_err
  }
}

## Next -----
x <- rnorm(100)
loop <- c()
for (i in seq_along(x)) {
  try_err <- try(err_fx(x[i]), silent = T)
  if (inherits(try_err, "try-error")){
    next
  } else {
    loop[i] <- try_err
  }
}

x <- rnorm(100)
loop <- c()
```

```r
for (i in seq_along(x)) {
  try_err <- try(err_fx(x[i]), silent = T)
  if (inherits(try_err, "try-error")){
    next
  } else {
    loop <- c(loop, try_err)
  }
}
```

```r
# Control Flow: try function -----

## Function
err_fx <- function(x){
  if (x>0){
    return(x)
  } else {
    stop("x is not positive")
  }
}

## Example -----
(y <- rnorm(1))
```

```
[1] -0.005070777
```

```r
# err_fx(y)

## try function ----

y_err <- try(err_fx(y), silent = T)

## Example ----
# x <- rnorm(100)
# loop <- c()
# for (i in x){
#   loop[i] <- err_fx(i)
# }

## Using try
x <- rnorm(100)
```

```r
loop <- c()
for (i in seq_along(x)) {
  try_err <- try(err_fx(x[i]), silent = T)
  if (inherits(try_err, "try-error")){
    loop[i] <- 0
  } else {
    loop[i] <- try_err
  }
}
```

```r
# Control Flow: Loops ----

# Loops are used to conduct repetitive/iterative tasks
# Each iteration conducts a task given a set of values
# The values for each iteration change as the loop moves
# from one iteration to another

## for Anatomy -----

# for (i in vector) {
#   Perform Task
# }

## Printing Example ------

### print number 1 through 5, separately

# We want to do this:
print(1); print(2); print(3); print(4); print(5)
```

```
[1] 1

[1] 2

[1] 3

[1] 4

[1] 5
```

```r
# We don't want this:
print(1:5)
```

```
[1] 1 2 3 4 5
```

```r
### Using a loop
for (i in 1:5){
  print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

```r
## Printing Letters -----
### Print all the letters, seperately
print(letters)
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
```

```r
for (i in 1:26){
  print(letters[i])
}
```

```
[1] "a"
[1] "b"
[1] "c"
[1] "d"
[1] "e"
[1] "f"
[1] "g"
[1] "h"
[1] "i"
```

```
[1] "j"
[1] "k"
[1] "l"
[1] "m"
[1] "n"
[1] "o"
[1] "p"
[1] "q"
[1] "r"
[1] "s"
[1] "t"
[1] "u"
[1] "v"
[1] "w"
[1] "x"
[1] "y"
[1] "z"
```

```
## cleaner
for (i in seq_along(letters)){
  print(letters[i])
}
```

```
[1] "a"
[1] "b"
[1] "c"
[1] "d"
[1] "e"
[1] "f"
[1] "g"
[1] "h"
[1] "i"
[1] "j"
[1] "k"
[1] "l"
[1] "m"
[1] "n"
[1] "o"
[1] "p"
[1] "q"
[1] "r"
```

```
[1] "s"
[1] "t"
[1] "u"
[1] "v"
[1] "w"
[1] "x"
[1] "y"
[1] "z"
```

```r
## cleanest

for (i in letters){
  print(i)
}
```

```
[1] "a"
[1] "b"
[1] "c"
[1] "d"
[1] "e"
[1] "f"
[1] "g"
[1] "h"
[1] "i"
[1] "j"
[1] "k"
[1] "l"
[1] "m"
[1] "n"
[1] "o"
[1] "p"
[1] "q"
[1] "r"
[1] "s"
[1] "t"
[1] "u"
[1] "v"
[1] "w"
[1] "x"
[1] "y"
[1] "z"
```

```r
# Control Flow: Nested Loops ----
library(greekLetters)
letters_new <- letters[1:3]
greek_lower <- greek_vector[1:24]

paste(letters_new[1], greek_lower[1], sep = "")
```

```
[1] "a "
```

```r
paste(letters_new[1], greek_lower[2], sep = "")
```

```
[1] "a "
```

```r
paste(letters_new[2], greek_lower[1], sep = "")
```

```
[1] "b "
```

```r
paste(letters_new[2], greek_lower[2], sep = "")
```

```
[1] "b "
```

```r
paste(letters_new[3], greek_lower[1], sep = "")
```

```
[1] "c "
```

```r
paste(letters_new[3], greek_lower[2], sep = "")
```

```
[1] "c "
```

```r
## Inefficient way
for(i in greek_lower){
```

```
  print(paste(letters_new[1], i, sep = ""))
}
```

```
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
```

```
for(i in greek_lower){
  print(paste(letters_new[2], i, sep = ""))
}
```

```
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
```

```
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
```

```
for(i in greek_lower){
  print(paste(letters_new[3], i, sep = ""))
}
```

```
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
```

```
[1] "c "
[1] "c "
[1] "c "
[1] "c "
```

```r
for (i in 1:3){
  for (ii in greek_lower){
    print(paste(letters_new[i], ii, sep = ""))
  }
}
```

```
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
```

```
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
```

```r
for (i in letters_new){
  for (ii in greek_lower){
    print(paste(i, ii, sep = ""))
  }
}
```

```
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "a "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
```

```
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "b "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
[1] "c "
```

```
# for (i in vector) {
#   for (ii in vector) {
#     for (iii in vector) {
#       for (iiii in vector) {
```

```r
#
#       }
#     }
#   }
# }
```

```r
# Control Flow: Loops ----

# for (i in vector) {
#   Perform Task
# }

## Vector construction -----
# x²
### Method 1
x <- rnorm(1000)
x^2
```

```
  [1] 5.348584e-02 1.664092e+00 1.918243e+00 1.711389e+00 2.327740e-02
  [6] 1.012841e+00 2.936439e-01 1.519564e+00 3.676435e+00 1.152051e+00
 [11] 1.599377e+00 1.788807e-01 2.463992e+00 2.162355e+00 1.341476e+00
 [16] 3.990371e-02 1.198446e+00 4.204879e-02 4.249948e-02 4.752103e-01
 [21] 1.668430e+00 3.424167e-01 6.032967e-01 2.453492e+00 3.164387e+00
 [26] 1.628942e-01 5.248844e-03 6.650262e+00 6.685843e-02 6.478946e-01
 [31] 1.567603e-01 2.347303e+00 1.391420e-01 1.780851e+00 1.979508e-01
 [36] 9.499970e-02 3.691120e+00 2.293890e-01 2.010777e+00 2.256314e-01
 [41] 7.844216e-03 1.448226e+00 3.325593e-01 8.608092e-01 5.171390e-01
 [46] 1.268530e+00 5.232483e-01 4.732546e-01 1.602918e+00 1.509622e+00
 [51] 1.412346e+00 2.075882e+00 9.443494e-01 9.131136e-01 5.727308e-01
 [56] 2.723863e-01 2.847902e+00 4.280382e-11 5.074080e+00 2.446547e-01
 [61] 1.088274e+00 1.326914e-01 7.152752e-02 2.421226e+00 3.366379e+00
 [66] 4.072001e+00 4.111240e-02 2.624067e+00 1.055342e-01 2.332760e-02
 [71] 1.769168e+00 2.097070e+00 3.611866e-01 4.420847e-04 6.438789e-02
 [76] 2.088241e+00 1.535949e-01 1.200527e-01 4.420652e-02 1.185536e-02
 [81] 4.145782e-01 1.956516e-01 2.295051e-01 4.417025e-01 2.375374e-01
 [86] 6.023657e-01 4.571748e-01 8.199044e-01 1.059584e+00 4.456907e-01
 [91] 2.184405e-02 9.186077e-02 4.984223e-01 4.579482e-01 2.042197e+00
 [96] 4.719997e+00 6.054469e-04 1.856671e+00 1.444367e+00 1.719817e-02
[101] 2.040794e+00 1.165736e-01 2.595194e-02 5.345346e-02 3.164467e-02
[106] 5.165132e-01 6.897944e-02 1.953418e-01 5.025109e-01 8.360873e-01
[111] 2.677655e-01 4.493440e-01 1.667324e-01 9.316070e-02 3.637218e-01
```

```
[116] 8.193431e-01 2.385972e+00 1.717045e+00 2.696923e+00 3.921301e-01
[121] 6.544492e-01 4.085132e-01 4.443777e-01 3.119752e+00 6.623509e-01
[126] 8.657543e-02 2.896596e-01 2.741212e+00 9.456066e-06 3.592535e-01
[131] 9.363643e-02 1.437054e-01 6.572965e-01 1.747509e-01 1.077599e+00
[136] 3.066345e+00 1.235425e-01 2.577850e-01 1.979220e-02 1.834852e+00
[141] 1.438215e-01 9.419345e-03 3.854839e-02 2.926818e-01 3.638872e-01
[146] 1.430603e-01 3.849095e-03 6.823832e-04 3.546642e-01 1.278027e+00
[151] 3.382121e-01 1.229841e+00 1.307354e+00 9.631470e-01 9.262680e-01
[156] 4.259511e+00 4.896445e-02 1.148746e-03 8.873031e-01 2.034622e+00
[161] 1.429523e-02 3.914358e-01 1.075123e+00 7.399476e-01 2.412474e+00
[166] 4.033123e+00 3.412295e+00 2.208257e-03 4.925450e-01 5.345127e-01
[171] 2.942146e-01 2.097706e-02 2.002769e-02 2.134920e-01 3.116046e-02
[176] 2.652011e+00 9.633868e-01 7.467114e-02 6.823651e-03 7.270642e-02
[181] 4.491265e-01 4.593347e-01 1.479034e+00 9.085182e-01 4.660089e+00
[186] 6.571836e-03 2.236372e-01 2.965119e+00 1.165261e-01 3.956312e+00
[191] 3.151006e+00 1.124694e+00 6.258209e-02 1.922315e+00 1.568165e+00
[196] 5.450361e-01 1.183168e+00 5.096854e-01 5.264010e-01 7.820875e-01
[201] 3.400814e-01 4.783769e-02 1.625800e+00 1.649728e-02 4.605404e-01
[206] 7.764696e-03 1.006514e-01 1.828786e+00 1.258876e+00 1.031651e-01
[211] 2.315250e+00 2.654951e+00 9.048202e-02 6.640326e-01 2.344392e-01
[216] 1.560288e+00 3.248968e-01 2.361202e+00 3.403985e-02 1.409627e-02
[221] 3.727345e-01 5.097858e-01 2.828376e+00 2.929376e+00 8.781968e-01
[226] 1.002256e-05 1.633171e-01 2.741046e+00 2.045637e+00 6.329268e-03
[231] 1.251758e+00 6.491608e-03 1.674951e-02 2.708552e-01 1.886343e-01
[236] 5.957115e-01 4.814928e-02 7.131473e-01 6.988771e-01 1.867289e-01
[241] 3.495313e-01 2.449866e-01 3.932402e+00 1.220145e+00 3.479278e-02
[246] 4.220702e+00 1.215138e+00 7.524239e-01 4.703955e+00 6.191851e-01
[251] 2.583086e-02 3.812120e+00 8.165457e+00 3.237745e+00 2.115005e+00
[256] 1.516810e+00 5.082181e-01 1.418879e+00 1.184107e+00 1.807185e+00
[261] 1.498953e+00 1.660259e-01 7.830109e-03 1.132853e+00 7.823713e-01
[266] 4.231058e-01 1.356504e+00 1.333127e+00 1.487657e+00 1.753425e-02
[271] 1.337231e+00 4.104292e+00 3.014028e-01 3.777515e+00 4.204295e-01
[276] 9.325370e-03 8.353642e-02 3.773707e-01 3.417451e+00 4.090033e+00
[281] 1.264520e-01 1.423735e-01 1.197227e+00 9.004364e-03 2.341347e-01
[286] 3.193962e-03 2.384471e+00 1.862427e+00 2.017159e-01 5.639710e-01
[291] 1.034960e-01 5.596225e-04 2.901855e-01 4.541126e+00 1.331239e+00
[296] 7.144457e-03 2.274450e+00 2.486897e-01 1.541302e-01 1.908335e+00
[301] 3.024975e+00 6.728375e-02 6.164534e-01 4.435363e-01 1.002729e-02
[306] 1.074121e+00 6.253494e+00 4.647124e-02 1.699507e+00 1.357501e-01
[311] 1.475253e+00 2.531482e+00 4.558008e-03 7.421619e-01 7.990567e-01
[316] 2.099587e-01 3.681977e-02 9.714501e-01 9.888055e-02 1.105025e+00
[321] 5.719541e-01 4.540539e-01 1.938402e-01 2.850073e+00 5.106259e-01
[326] 8.373898e-01 8.576681e-03 2.676233e-01 5.286810e+00 2.823659e-02
```

```
[331] 2.862378e-01 1.453331e-01 1.277563e-03 2.554474e+00 7.058359e-02
[336] 3.716121e-02 1.039075e-01 2.992059e+00 8.318727e-01 1.443178e+00
[341] 2.774722e-01 9.438814e-01 5.378198e-01 5.395184e-03 1.167196e+00
[346] 2.789319e+00 2.171931e-03 3.861676e+00 5.628163e-01 2.256982e-02
[351] 2.560798e-01 3.979320e-01 4.157349e+00 1.107019e-01 3.288392e+00
[356] 5.804795e-02 1.533849e+00 1.140632e+00 1.082579e+00 1.732472e+00
[361] 9.794282e-01 2.679327e-02 1.606830e+00 9.698794e-01 1.265810e+00
[366] 2.728723e-01 3.572203e-03 7.982388e-01 6.475590e-02 2.429487e-01
[371] 2.656468e-01 5.633141e-01 9.260289e-03 6.208087e-02 4.084042e+00
[376] 4.103618e-01 3.738146e+00 9.600512e-02 2.966910e-01 4.798505e-01
[381] 2.069642e-02 5.591259e-02 1.709428e+00 4.677709e-01 1.574947e+00
[386] 4.542265e-02 3.331003e-01 1.600914e-01 4.251842e-01 6.939235e-02
[391] 9.737001e-01 3.900629e+00 7.005943e-01 1.500200e+00 3.744047e-01
[396] 1.379235e+00 6.390059e-01 5.853269e-02 7.770535e-01 1.834968e+00
[401] 2.618402e-01 2.083906e+00 1.272273e-01 3.850930e-01 9.504336e-02
[406] 1.014949e+00 2.858107e-02 2.038372e-01 9.633245e-03 1.080091e-01
[411] 6.706581e-02 4.943384e-01 3.871372e+00 1.042761e+00 2.341251e-01
[416] 1.721615e+00 7.281153e-01 1.071149e+00 6.374224e-02 8.011872e-01
[421] 2.934578e+00 3.203106e+00 8.155905e-01 3.623772e-01 2.154534e-01
[426] 6.412292e-01 3.232894e-01 1.811207e+00 2.755471e-01 9.437922e-03
[431] 1.875775e-01 5.947048e-01 2.029884e+00 8.170587e-05 9.368262e-01
[436] 4.040634e-01 1.263788e+00 2.105615e-01 3.477025e-04 2.987196e-01
[441] 5.213792e-01 1.415625e+00 6.790195e-02 1.302089e+00 4.714378e-01
[446] 9.894389e-02 2.034306e+00 3.090670e-01 2.923221e-01 1.228377e-01
[451] 4.469354e+00 8.859911e-01 1.217051e+01 2.767309e-01 2.207826e+00
[456] 3.342019e+00 3.156679e+00 7.730053e-01 2.989245e-02 5.171478e-01
[461] 3.275774e-02 1.876385e-01 5.353465e-01 6.503269e-01 7.260779e-03
[466] 1.522400e+00 1.444287e-02 2.927125e-04 2.666857e-01 4.527664e-02
[471] 6.762106e-01 3.227456e-02 1.387223e-01 2.190162e-02 2.078672e+00
[476] 2.573569e-02 2.393809e+00 9.574337e-04 1.671723e+00 4.274280e+00
[481] 2.118874e-01 1.216146e-04 2.238996e-02 2.826466e-01 2.493082e-01
[486] 9.895649e-02 1.996540e+00 4.218542e-01 3.343788e-03 2.961731e-01
[491] 5.300928e-02 6.043298e-01 3.301455e+00 1.424169e+00 3.048529e-01
[496] 2.816883e+00 4.214638e-03 4.796759e-01 1.194733e+00 5.097271e-01
[501] 2.603935e-02 2.362864e+00 6.968014e+00 2.890555e-01 4.082631e-01
[506] 4.214868e-01 6.836031e-01 3.577177e-02 6.962514e-03 5.378059e-01
[511] 9.620674e-03 2.890959e-02 1.003945e+00 1.472464e+00 2.343498e+00
[516] 3.919666e-01 4.703488e-02 8.252994e-02 1.172100e-02 1.252224e-02
[521] 6.086469e-04 6.516898e-04 9.948902e-01 6.964360e-04 7.866011e-02
[526] 2.034300e-02 2.743344e+00 7.307721e-01 4.963369e-01 6.087003e-01
[531] 1.303259e-01 6.473012e-01 3.083578e-01 1.022544e+00 5.552984e-09
[536] 4.091002e-03 4.110388e-02 1.104970e+00 1.654703e+00 2.125919e+00
[541] 9.916488e-02 3.118220e+00 8.920276e-01 7.361158e-02 1.370603e-01
```

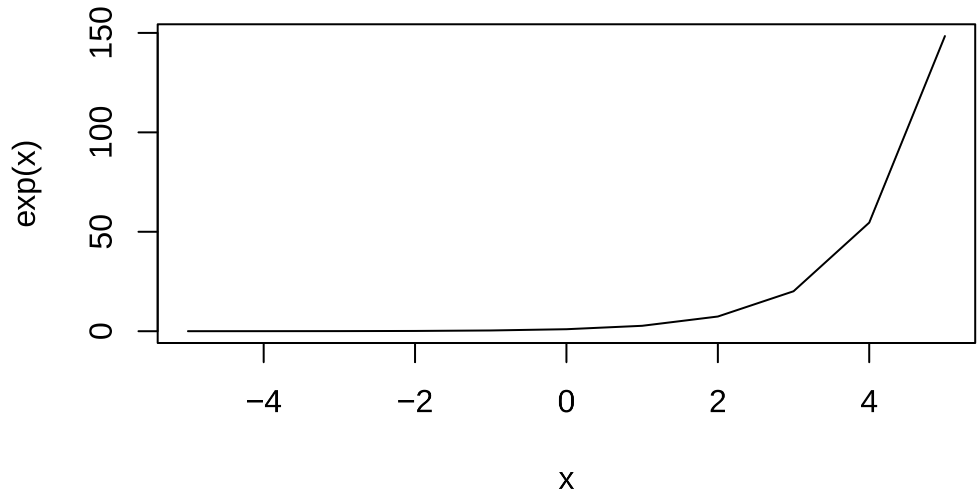
```
[546] 1.379626e-01 7.266824e-01 1.559830e-01 7.026175e-02 1.303477e-01
[551] 8.283332e-02 1.931669e-01 1.097759e+00 9.283096e-01 1.872770e-01
[556] 3.367129e-01 2.462173e+00 3.532358e-01 2.213799e-02 1.747013e+00
[561] 3.505523e-01 5.498238e-01 4.654462e-02 3.675808e+00 6.962178e-01
[566] 7.113374e+00 1.182878e+00 2.198274e-01 7.283321e-03 3.939403e-01
[571] 2.503053e-01 1.435070e+00 7.045933e-02 7.832965e-03 1.458602e+00
[576] 1.411160e-01 8.602029e-01 4.674467e-03 6.022733e-02 3.830312e-02
[581] 1.079911e-02 7.529295e-01 1.242153e-01 3.886801e+00 2.680432e-01
[586] 3.833496e+00 1.946219e+00 1.620593e+00 5.716322e-02 4.755986e+00
[591] 8.114526e-01 7.798150e-03 4.031550e-01 4.550249e-02 2.113373e+00
[596] 4.005098e+00 4.230529e-01 1.546895e+00 2.286572e+00 4.021833e-01
[601] 1.044909e-01 2.901475e-04 1.450410e-01 6.933484e-01 1.667234e+00
[606] 1.123188e-01 1.612457e-02 1.719700e+00 1.432326e+00 3.558226e-01
[611] 1.400427e-01 1.469675e+00 4.181872e-01 1.216060e+00 9.093315e-01
[616] 3.523033e-01 2.941246e-02 9.706899e-01 5.048933e-03 2.801646e-02
[621] 1.224550e-01 8.333162e+00 9.671693e-01 7.192539e-01 1.555991e+00
[626] 3.619749e-01 1.748793e+00 1.087240e+00 1.480133e-03 4.321849e-01
[631] 6.706861e-01 1.203086e+00 1.358825e-01 3.083906e-02 1.692266e-01
[636] 1.065802e-01 3.641177e-01 7.391391e-01 1.435364e-02 4.654386e-02
[641] 2.358896e+00 6.196982e-01 3.362774e-01 1.718071e-02 1.251488e-01
[646] 4.430480e-02 8.800042e-01 4.454879e-01 1.755579e-01 3.445031e-01
[651] 1.421253e+00 1.341943e+00 9.890641e-02 8.320266e-01 6.672051e-02
[656] 2.734812e-02 1.649422e-03 2.639952e-01 8.783530e-01 2.675785e-01
[661] 1.202287e-01 6.923662e-01 5.872748e+00 8.487320e-01 1.169131e+00
[666] 6.700352e-01 4.796204e-03 4.851608e-01 5.069662e+00 5.069409e+00
[671] 4.788046e-02 3.085236e+00 6.138251e-01 3.435093e+00 3.314998e+00
[676] 2.100677e-02 4.834096e-01 1.200267e-01 4.862483e-01 8.378728e-01
[681] 4.241098e-01 2.544738e-01 3.373085e+00 2.421032e-03 1.588961e+00
[686] 1.107610e+00 2.199404e-01 1.757799e-01 3.902992e-01 1.545481e-02
[691] 1.311708e+00 6.140748e-03 1.161900e+00 2.354405e-01 5.548847e-02
[696] 2.374687e-02 5.984754e-02 6.039751e-01 3.063944e+00 3.649804e-02
[701] 9.679915e-03 6.306001e-01 2.705214e+00 1.067983e+00 3.441013e+00
[706] 2.588194e+00 3.209789e-01 1.883517e-02 4.081701e+00 5.807980e-02
[711] 2.335523e+00 7.609288e-01 1.775268e+00 1.183408e+00 4.447900e-01
[716] 5.175619e-01 8.315621e-01 6.571961e-01 5.318665e-01 1.441965e+00
[721] 1.056155e+00 1.916837e+00 6.395106e-02 3.638635e-01 1.499919e-02
[726] 5.734469e-01 2.135486e+00 3.234100e+00 3.021325e-02 4.105810e+00
[731] 3.223397e-02 1.318964e-01 1.677759e-02 7.583979e-01 5.552906e-01
[736] 2.247961e-01 3.181469e-02 2.207623e-01 3.379442e+00 3.443565e-01
[741] 9.994246e-01 1.996820e-01 1.225982e+00 7.525038e-01 2.018350e+00
[746] 2.834088e+00 8.522858e-01 1.023255e+00 9.431686e-01 6.741017e-01
[751] 5.907913e-02 2.519993e-01 4.222497e-02 3.146148e-01 1.254522e-02
[756] 1.641874e+00 1.528381e-02 6.024758e-01 2.879514e-01 1.749842e-01
```

```
[761] 2.543419e+00 3.276944e+00 2.578163e-03 3.902083e+00 2.386888e-01
[766] 4.598627e-02 1.866320e+00 1.601901e-01 3.200328e-03 7.556022e-01
[771] 2.778338e-01 2.348392e-01 2.222203e-01 4.759682e-02 5.587343e-01
[776] 3.430552e-02 1.156228e-02 4.176113e-01 6.872766e-01 9.893936e-02
[781] 3.650845e-02 1.126020e+00 4.879825e+00 1.585324e-02 1.762196e-01
[786] 2.011064e-01 6.157998e-03 9.718579e-01 6.873609e-01 3.579477e-02
[791] 1.361617e+00 3.125172e+00 2.202478e-01 1.243209e+00 2.250305e+00
[796] 3.144248e-01 8.314536e-01 1.115631e+00 5.531508e-01 1.863332e-07
[801] 5.101036e-02 1.715845e-01 3.149104e-01 5.894088e-03 2.904219e-02
[806] 1.207176e+00 3.198951e-01 1.839850e-01 4.488831e-01 3.351428e+00
[811] 3.571128e-03 3.251807e+00 2.359167e+00 1.049669e-01 5.861239e-02
[816] 1.211202e+00 1.829236e-02 3.676408e-01 6.443920e-02 4.695523e+00
[821] 9.280515e-01 3.759045e+00 1.386166e+00 7.507506e-01 2.424933e-01
[826] 2.574604e-01 2.207476e-01 1.044186e-01 3.902219e-01 2.864956e+00
[831] 6.044741e-01 9.780417e-01 1.018703e-01 3.146869e-01 6.178665e+00
[836] 2.008586e+00 1.175970e+00 3.452850e-01 4.153765e-02 1.757825e-01
[841] 4.284514e+00 1.849729e+00 3.523655e-01 1.074456e+00 1.619745e+00
[846] 1.701628e-01 9.872797e-02 7.252695e-01 6.488126e-01 2.301995e-01
[851] 2.682796e-01 3.014409e+00 2.340518e+00 3.276803e-02 8.205941e-01
[856] 5.372449e+00 2.094396e-01 3.053779e+00 8.985709e-01 8.909134e-01
[861] 2.198155e+00 5.212965e-02 2.467327e+00 1.672381e-02 3.549565e-01
[866] 8.829009e-02 1.146069e-01 4.349927e-01 1.475090e-01 1.920677e-01
[871] 2.942201e+00 6.086110e-01 8.839149e-01 8.149333e-02 6.169637e+00
[876] 2.527502e+00 2.494115e-03 9.257340e-01 3.811972e-07 6.558382e-01
[881] 7.387166e-01 2.486733e+00 4.493887e+00 3.064065e-01 1.089303e+00
[886] 1.624365e+00 4.889055e-01 4.955480e-01 5.376320e-02 2.449047e-01
[891] 3.055337e+00 5.270435e-02 4.215367e-06 2.647343e-01 9.153207e-02
[896] 2.707282e-01 1.177480e+00 5.622839e-01 1.091704e+00 6.718690e-01
[901] 1.503171e+00 1.938837e+00 4.874873e+00 1.010658e+00 2.283674e-02
[906] 2.837947e+00 8.620938e-03 3.934213e-01 1.364180e+00 4.147407e+00
[911] 3.746468e+00 1.116446e+00 1.180436e+00 4.791686e-01 8.163386e-02
[916] 9.475983e-03 2.531469e+00 3.265622e-01 5.101334e-03 2.211865e+00
[921] 1.381063e-02 9.094544e-01 9.906998e-02 2.695558e-01 1.057293e-02
[926] 9.354072e-02 1.001128e+00 9.818381e-01 4.619040e-01 1.646822e-02
[931] 7.284779e-01 1.836977e+00 1.426604e+00 7.761052e-03 4.938741e+00
[936] 2.069402e-03 1.715127e+00 6.272324e-02 1.552440e+00 2.659720e-01
[941] 5.208616e-01 6.874459e+00 8.077839e-02 1.051706e-02 2.467462e-01
[946] 1.481947e+00 1.994890e+00 1.647618e+00 1.704152e+00 3.800891e-01
[951] 1.295386e+00 6.646892e-02 6.020226e-01 1.228086e+00 1.380950e-02
[956] 3.958217e-01 5.468486e-01 2.958823e-01 1.392606e+00 1.067636e+00
[961] 5.760814e-01 6.986888e+00 1.016910e+00 3.564605e+00 1.578958e-01
[966] 1.707840e-01 7.794471e-01 7.722081e-02 1.316900e-02 1.393515e+00
[971] 1.682330e-01 2.809506e+00 9.640901e-01 1.877643e-03 2.264277e-01
```

```
[976] 2.746613e+00 3.878983e-01 6.611567e-02 2.206388e+00 3.858181e+00
[981] 1.510164e+00 3.113806e-02 5.279598e-05 7.839137e-02 1.658041e-02
[986] 8.891252e-01 4.122520e-02 8.120739e-03 1.381895e+00 2.260731e+00
[991] 1.462446e-02 2.727826e+00 8.070967e-02 2.588105e+00 2.327858e+00
[996] 2.831578e+00 1.116272e+00 2.567017e-01 8.873182e-02 8.476827e-03
```

```
output <- c()
for (i in seq_along(x)){
  output <- c(output, x[i]^2)
}

### Method 2
x <- rnorm(1000)
output <- c()
for (i in seq_along(x)){
  output[i] <- x[i]^2
}

### Method 3
x <- rnorm(1000)
output <- vector(length = length(x))
for (i in seq_along(x)){
  output[i] <- x[i]^2
}
```

```
# Control Flow: While Loops -----
x <- -5:5
plot(x, exp(x), type = "l")
```




```r
## Asymptotic -----
abs(exp(-14)-exp(-13))
```

```
[1] 1.428801e-06
```

```r
## while statements -----
# while (condition) {
#
# }


## while loops ----
x <- 1
diff <- 1
while (diff > 1e-20) {
  old_x <- x
  x <- x - 1
  diff <- abs(exp(x) - exp(old_x))
}
print(x)
```

```
[1] -47
```

```r
print(diff)
```

```
[1] 6.65662e-21
```

```r
# Control Flow: Infinite While Loops -----
(y <- rnorm(1))
```

```
[1] 0.5548079
```

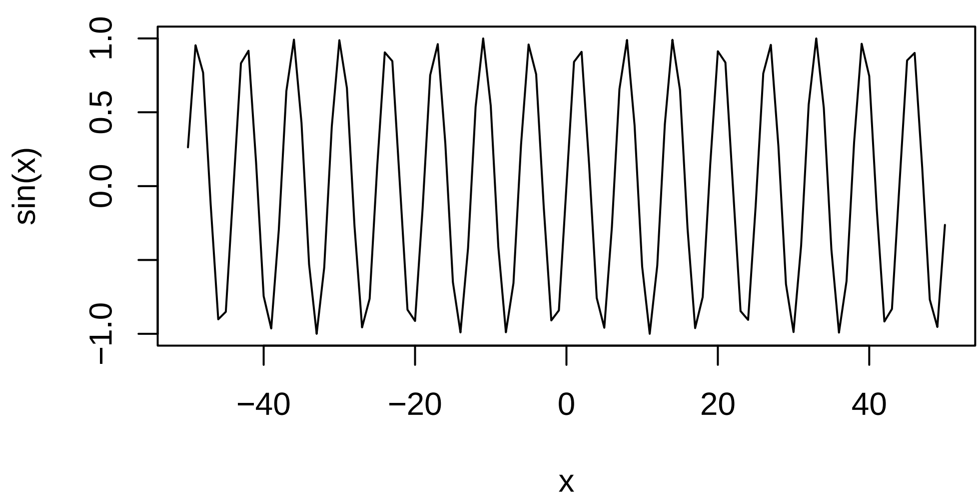
```r
z <- as.integer(2)

# logical operator &
is.integer(z) & z > 0
```

```
[1] TRUE
```

```r
is.integer(y) & y > 0
```

```
[1] FALSE
```

```r
x <- -50:50
plot(x, sin(x), type = "l")
```



```r
## Asymptotic -----
abs(exp(13)-exp(12))
```

```
[1] 279658.6
```

```r
## inf while loops ----
x <- 1
diff <- 1
# while (diff > 1e-20) {
#   old_x <- x
#   x <- x + 1
#   diff <- abs(exp(x) - exp(old_x))
# }
print(x)
```

```
[1] 1
```

```r
print(diff)
```

```
[1] 1
```

```r
# x <- 1
# diff <- 1
# while (diff > 1e-20) {
#   old_x <- x
#   x <- x + 1
#   diff <- abs(sin(x) - sin(old_x))
# }
# print(x)
# print(diff)




## while loops ----
x <- 1
counter <- 0
```

```
diff <- 1
while (diff > 1e-20 & counter < 30) {
  old_x <- x
  x <- x + 1
  diff <- abs(exp(x) - exp(old_x))
  counter <- counter + 1
}
print(x)
```

```
[1] 31
```

```
print(diff)
```

```
[1] 1.836238e+13
```

```
print(counter)
```

```
[1] 30
```

```
x <- 1
counter <- 0
diff <- 1
while (diff > 1e-20 & counter < 10^3) {
  old_x <- x
  x <- x + 1
  diff <- abs(sin(x) - sin(old_x))
  counter <- counter + 1
}
print(x)
```

```
[1] 1001
```

```
print(diff)
```

```
[1] 0.09311106
```

```r
print(counter)
```

```
[1] 1000
```

3 Functional Programming

3.1 *apply Functions

4 Scripting and Piping in R

Part II

Data Manipulation, Summarization, and Graphics

5 Importing Data

```
# Reading Data -----

## RData ----
load("~/x.RData")

## CSV ----
library(readr)
data_3_1_csv <- read_csv("student/stat_147/data/data_3_1.csv")
View(data_3_1_csv)

## Excel ----
library(readxl)
data_3_1 <- read_excel("student/stat_147/data/data_3_1.xlsx")
View(data_3_1)

## txt ----
library(readr)
data_3_1_s <- read_table2("student/stat_147/data/data_3_1_s.txt")
View(data_3_1_s)

## Semi-colon ----
library(readr)
data_3_1_sc <- read_delim("student/stat_147/data/data_3_1_sc.txt", ";", escape_double = FA
View(data_3_1_sc)
## SPSS ----
library(haven)
data_3_1 <- read_sav("student/stat_147/data/data_3_1.sav")
View(data_3_1)

## SAS -----
library(haven)
data_3_1 <- read_sas("student/stat_147/data/data_3_1.sas7bdat", NULL)
View(data_3_1)
```

```r
## Stata ----
library(haven)
data_3_1 <- read_dta("student/stat_147/data/data_3_1.dta")
View(data_3_1)

data_3_1 <- read.csv("~/student/stat_147/data/data_3_1.csv", header=FALSE)
View(data_3_1)


# Reading Data -----
setwd("~/Repos/s147/files/Week_2")

## Base R -----

# CSV
data.csv <- read.csv("data.csv")

# STATA File
library(foreign)
read.dta("data.dta")

## RStudio packages
library(readr)
read_csv("data.csv")

library(readxl)
read_excel("data.xlsx")

library(haven)
read_dta("data.dta")
```

6 Data Manipulation

7 Data Summarization

7.1 Descriptive Statistics

Here, we will go over some of the basic syntax to obtain basic statistics. We will use the variables `mpg` and `cyl` from the `mtcars` data set. To view the data set use the `head()`:

```
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
```

The variable `mpg` would be used as a continuous variable, and the variable `cyl` would be used as a categorical variable.

7.1.1 Point Estimates

The first basic statistic you can compute are point estimates. These are your means, medians, etc. Here we will calculate these estimates.

7.1.1.1 Mean

To obtain the mean, use the `mean()`, you only need to specify `x=` for the data to compute the mean:

```
mean(mtcars$mpg)
```

```
[1] 20.09062
```

7.1.1.2 Median

To obtain the median, use the `median()`, you only need to specify `x=` for the data to compute the median:

```
median(mtcars$mpg)
```

```
[1] 19.2
```

7.1.1.3 Frequency

To obtain a frequency table, use the `table()`, you only need to specify the data as the first argument to compute the frequency table:

```
table(mtcars$cyl)
```

```

 4  6  8 
11  7 14 
```

7.1.1.4 Proportion

To obtain a the proportions for the frequency table, use the `prop.table()`. However the first argument must be the results from the `table()`. Use the `table()` inside the `prop.table()` to get the proportions:

```
prop.table(table(mtcars$cyl))
```

```

      4       6       8 
0.34375 0.21875 0.43750 
```

7.1.2 Variability

In addition to point estimates, variability is an important statistic to report to let a user know about the spread of the data. Here we will calculate certain variability statistics.

7.1.2.1 Variance

To obtain the variance, use the `var()`, you only need to specify `x=` for the data to compute the variance:

```
var(mtcars$mpg)
```

```
[1] 36.3241
```

7.1.2.2 Standard deviation

To obtain the standard deviation, use the `sd()`, you only need to specify `x=` for the data to compute the standard deviation:

```
sd(mtcars$mpg)
```

```
[1] 6.026948
```

7.1.2.3 Max and Min

To obtain the max and min, use the `max()` and `min()`, respectively. You only need to specify the data as the first argument to compute the max and min:

```
max(mtcars$mpg)
```

```
[1] 33.9
```

```
min(mtcars$mpg)
```

```
[1] 10.4
```

7.1.2.4 Q1 and Q3

To obtain the Q1 and Q3, use the `quantile()` and specify the desired quantile with `probs=`. You only need to specify the data as the first argument and `probs=` (as a decimal) to compute the Q1 and Q3:

```r
quantile(mtcars$mpg, .25)
```

```
   25%
15.425
```

```r
quantile(mtcars$mpg, .75)
```

```
 75%
22.8
```

7.1.3 Associations

In statistics, we may be interested on how different variables are related to each other. These associations can be represented in a numerical value.

7.1.3.1 Continuous and Continuous

When we measure the association between to continuous variables, we tend to use a correlation statistic. This statistic tells us how linearly associated are the variables are to each other. Essentially, as one variable increases, what happens to the other variable? Does it increase (positive association) or does it decrease (negative association). To find the correlation in R, use the `cor()`. You will need to specify the `x=` and `y=` which represents vectors for each variable. Find the correlation between `mpg` and `hp` from the `mtcars` data set.

```r
cor(mtcars$mpg, mtcars$hp)
```

```
[1] -0.7761684
```

7.1.3.2 Categorical and Continuous

When comparing categorical variables, it becomes a bit more nuanced in how to report associations. Most of time you will discuss key differences in certain groups. Here, we will talk about how to get the means for different groups of data. Our continuous variable is the `mpg` variable, and our categorical variable is the `cyl` variable. Both are from the `mtcars` data set. The `tapply()` allows us to split the data into different groups and then calculate different statistics. We only need to specify `X=` of the R object to split, `INDEX=` which is a list of factors or categories indicating how to split the data set, and FUN= which is the function that needs to be computed. Use the tapply() and find the mean mpg for each cyl group: 4, 5, and 6.

```
tapply(mtcars$mpg, list(mtcars$cyl), mean)
```

```
       4        6        8 
26.66364 19.74286 15.10000 
```

7.1.3.3 Categorical and Categorical

Reporting the association between two categorical variables is may be challenging. If you have a 2×2 table, you can report a ratio of association. However, any other case may be challenging. You can report a hypothesis test to indicate an association, but it does not provide much information about the effect of each variable. You can also report row, column, or table proportions. Here we will talk about creating cross tables and report these proportions. To create a cross table, use the table() and use the first two arguments to specify the two categorical variables. Create a cross tabulation between cyl and carb from the mtcars data set.

```
table(mtcars$cyl, mtcars$carb)
```

```
   
    1 2 3 4 6 8
  4 5 6 0 0 0 0
  6 2 0 0 4 1 0
  8 0 4 3 6 0 1
```

Notice how the first argument is represented in the rows and the second argument is in the columns. Now create table proportions using both of the variables. You first need to create the table and store it in a variable and then use the prop.table().

```
prop.table(table(mtcars$cyl, mtcars$carb))
```

```
   
          1       2       3       4       6       8
  4 0.15625 0.18750 0.00000 0.00000 0.00000 0.00000
  6 0.06250 0.00000 0.00000 0.12500 0.03125 0.00000
  8 0.00000 0.12500 0.09375 0.18750 0.00000 0.03125
```

To get the row proportions, use the argument `margin = 1` within the `prop.table()`.

```
prop.table(table(mtcars$cyl, mtcars$carb),
           margin = 1)
```

```

             1          2          3          4          6          8
  4 0.45454545 0.54545455 0.00000000 0.00000000 0.00000000 0.00000000
  6 0.28571429 0.00000000 0.00000000 0.57142857 0.14285714 0.00000000
  8 0.00000000 0.28571429 0.21428571 0.42857143 0.00000000 0.07142857
```

To get the column proportions, use the argument `margin = 2` within the `prop.table()`.

```
prop.table(table(mtcars$cyl, mtcars$carb),
           margin = 2)
```

```

            1         2         3         4         6         8
  4 0.7142857 0.6000000 0.0000000 0.0000000 0.0000000 0.0000000
  6 0.2857143 0.0000000 0.0000000 0.4000000 1.0000000 0.0000000
  8 0.0000000 0.4000000 1.0000000 0.6000000 0.0000000 1.0000000
```

7.2 Summarizing with Tidyverse

```
library(magrittr)
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --
v ggplot2 3.4.0      v purrr   1.0.0
v tibble  3.1.8      v dplyr   1.0.10
v tidyr   1.2.1      v stringr 1.5.0
v readr   2.1.3      v forcats 0.5.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x tidyr::extract()   masks magrittr::extract()
x dplyr::filter()    masks stats::filter()
x dplyr::lag()       masks stats::lag()
x purrr::set_names() masks magrittr::set_names()
```

```r
f <- function(x){
  mtcars %>% split(~.$cyl) %>% map(~shapiro.test(.$mpg))
  return(1)}
g <- function(x){
  mtcars %>% group_by(cyl) %>% nest() %>% mutate(shapiro = map(data, ~shapiro.test(.$mpg))
  return(1)}
bench::mark(f(1),g(1))
```

```
# A tibble: 2 x 6
  expression      min   median `itr/sec` mem_alloc `gc/sec`
  <bch:expr> <bch:tm> <bch:tm>     <dbl> <bch:byt>    <dbl>
1 f(1)        405.3us  439.4us    2196.   134.23KB    16.9
2 g(1)         12.2ms   12.5ms      78.4    3.65MB     8.97
```

8 Graphics

Through out this chapter, we use certain notations for different components in R. To begin, when something is in a gray block, `_`, this indicates that R code is being used. When I am talking about an R Object, it will be displayed as a word. For example, we will be using the R object `mtcars`. When I am talking about an R function, it will be displayed as a word followed by an open and close parentheses. For example, we will use the mean function denoted as `mean()` (read this as "mean function"). When I am talking about an R argument for a function, it will be displayed as a word following by an equal sign. For example, we will use the data argument denoted as `data=` (read this as "data argument"). When I am referencing an R package, I will use `::` (two colons) after the name. For example, in this tutorial, I will use the `ggplot2::` (read this as "ggplot2 package") Lastly, if I am displaying R code for your reference or to run, it will be displayed on its own line. There are many components in R, and my hope is that this will help you understand what components am I talking about.

8.1 Base R Plotting

8.1.1 Introduction

This tutorial provides an introduction on how to create different graphics in R. For this tutorial, we will focus on plotting different components from the `mtcars` data set.

8.1.2 Contents

1. Basic
2. Grouping
3. Tweaking

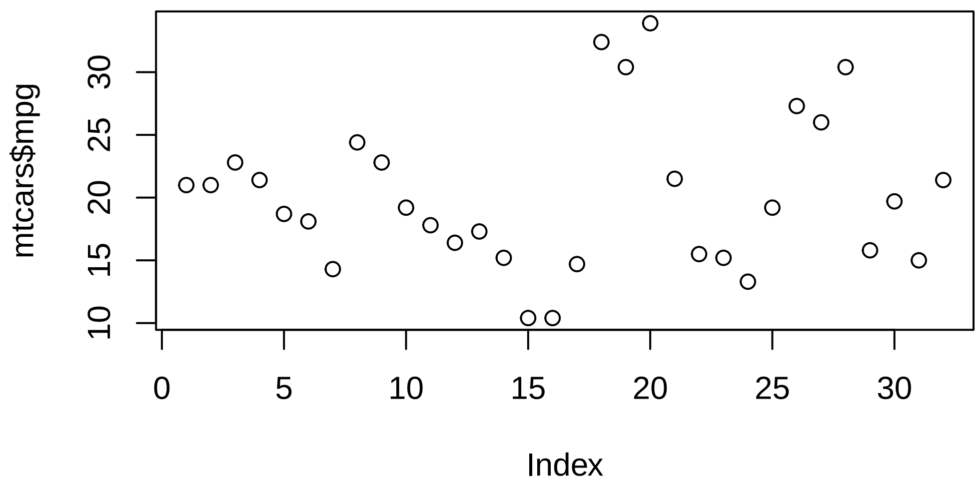
8.1.3 Basic Graphics

Here we will use the built-in R functions to create different graphics. The main function that you will use is the `plot()`. It contains much of the functionality to create many different plots in R. Additionally, it works well for different classes of R objects. It will provide many important plots that you will need for a certain statistical analysis.

8.1.4 Scatter Plot

Let's first create a scatter plot for one variable using the `mpg` variable. This is done using the `plot()` and setting the first argument `x=` to the vector.

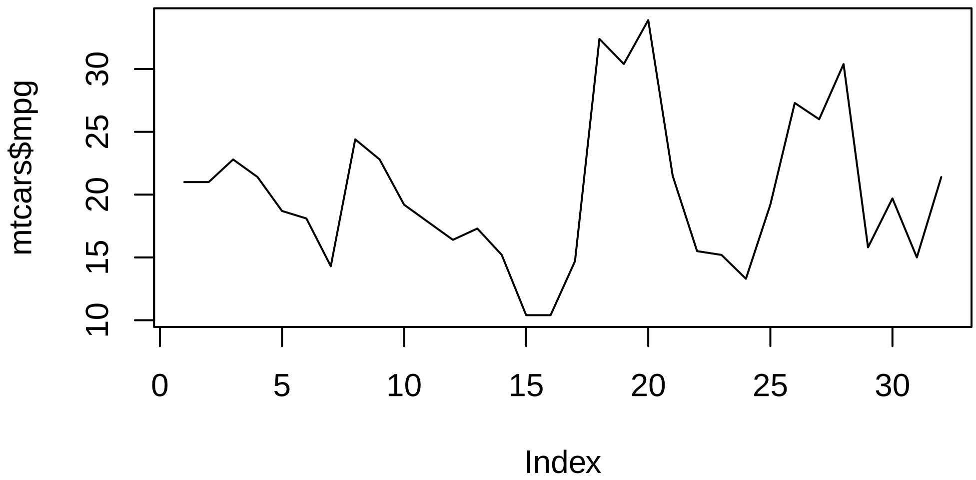
```
plot(mtcars$mpg)
```



Notice that the x-axis is the index (which is not informative) and the y-axis is the `mpg` values.

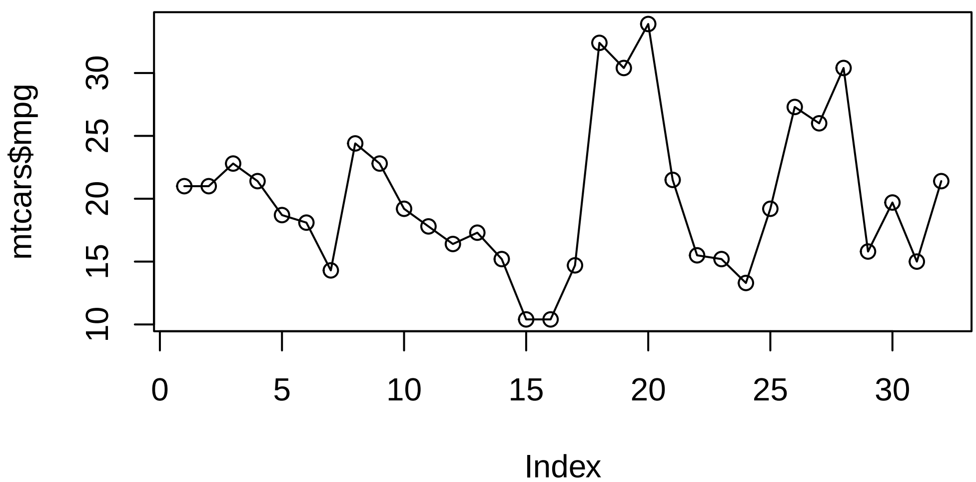
Let's connect the points with a line. This is done by setting the `type=` to `"l"`.

```
plot(mtcars$mpg, type = "l")
```



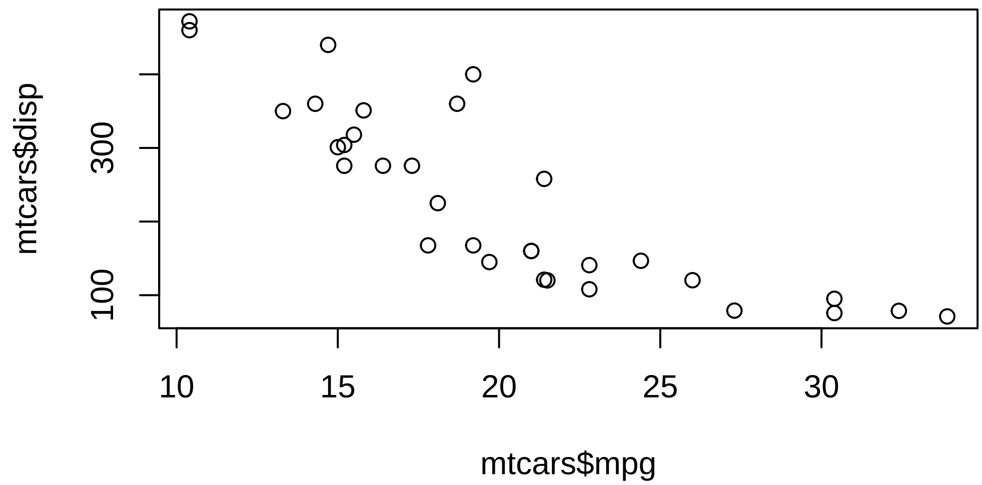
Let's add the points back to the plot and keep the lines. What we are going to do is first create the scatter plot as we did before, but we will also use the `lines()` to add the lines. The `lines()` needs the `x=` which is a vector of points (`mpg`). The two lines of code must run together.

```
plot(mtcars$mpg)
lines(mtcars$mpg)
```



Now, let's create a more realistic scatter plot with 2 variables. This is done by specifying the `y=` with another variable in addition to the `x=` in the `plot=`. Plot a scatter plot between `mpg` and `disp`.

```
plot(mtcars$mpg,mtcars$disp)
```

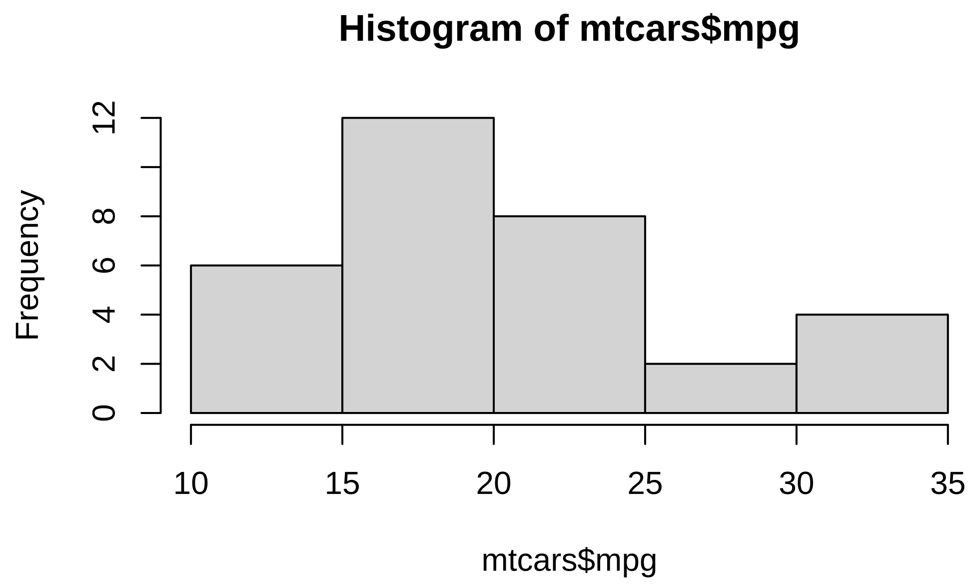


Now, let's change the the axis labels and plot title. This is done by using the arguments `main=`, `xlab=`, and `ylab`. The `main=` changes the title of the plot.

8.1.5 Histogram

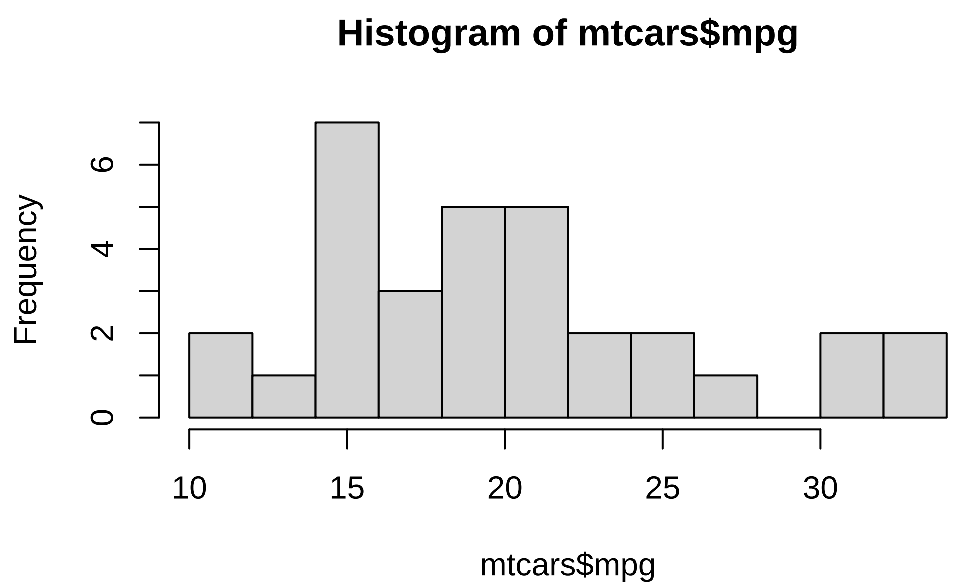
To create a histogram, use the `hist()`. The `hist()` only needs `x=` which is numerical vector. Create a histogram with the `mpg` variable.

```
hist(mtcars$mpg)
```



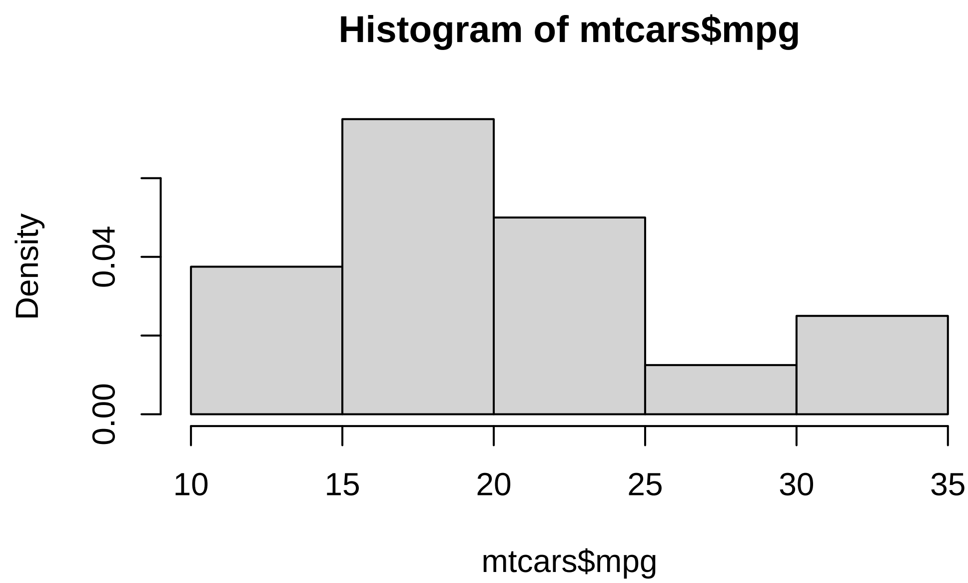
If you want to change the number of breaks in the histogram, use the `breaks=`. Create a new histogram of the `mpg` variable with ten breaks.

```
hist(mtcars$mpg, breaks = 10)
```



The above histograms provide frequencies instead of relative frequencies. If you want relative frequencies, use the `freq=` and set it equal to `FALSE` in the `hist()`.

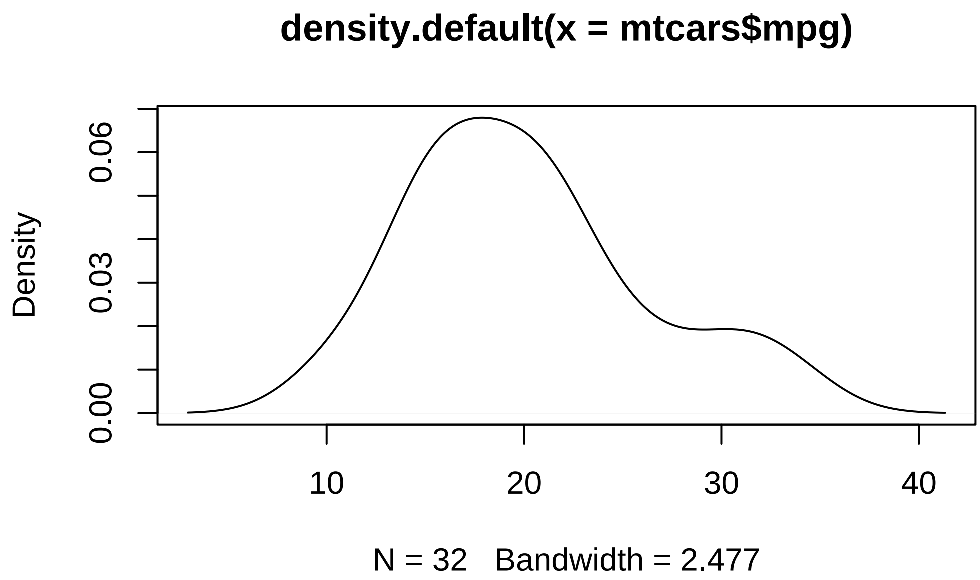
```r
hist(mtcars$mpg, freq = FALSE)
```



8.1.6 Density Plot

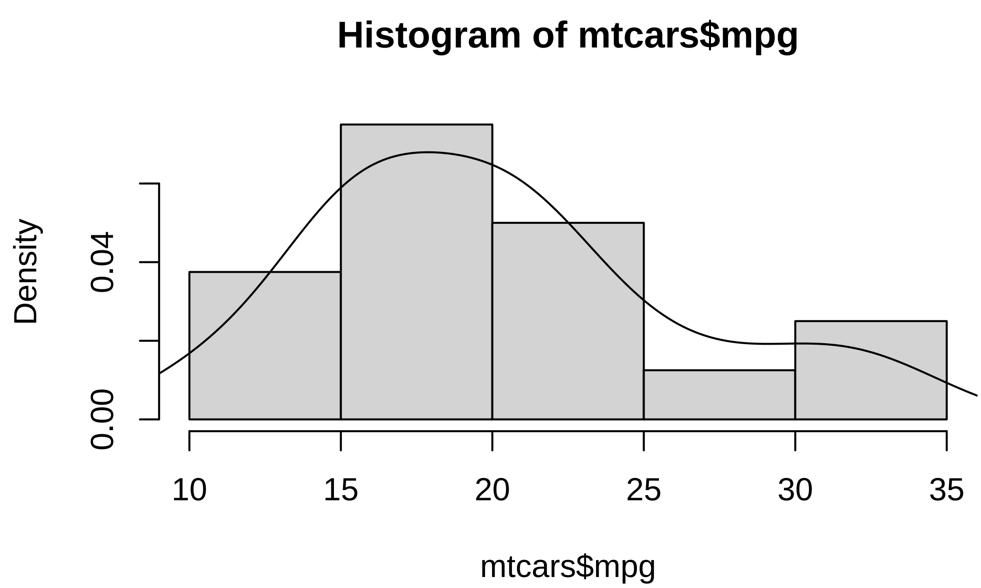
A density plot can be used instead of a histogram. This is done by using the `density()` to create an object containing the information to create density function. Then, use the `plot()` to display the plot. The only argument the `density()` needs is the `x=` which is the data to be used. Create a density plot the `mpg` variable.

```r
plot(density(mtcars$mpg))
```



Now, if we want to overlay the density function over a histogram, use the `lines()` with the output from the `density()` as its main input. First create the histogram using the `hist()` and setting the `freq=` to `FALSE`. Then use the `lines()` to overlay the density. Make sure to run both lines together.

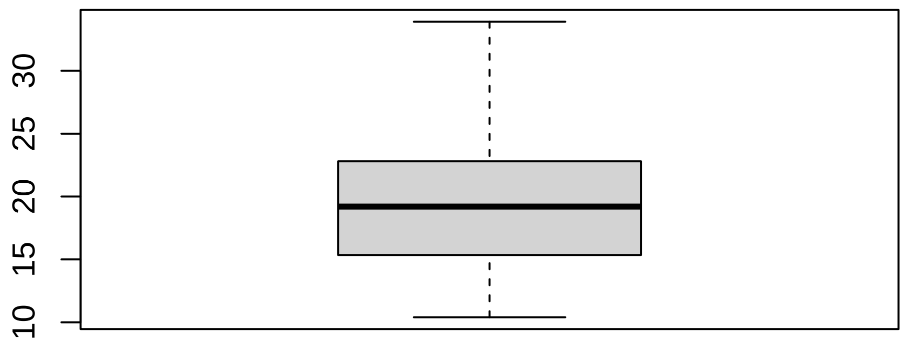
```
hist(mtcars$mpg, freq = FALSE)
lines(density(mtcars$mpg))
```



8.1.7 Box Plots

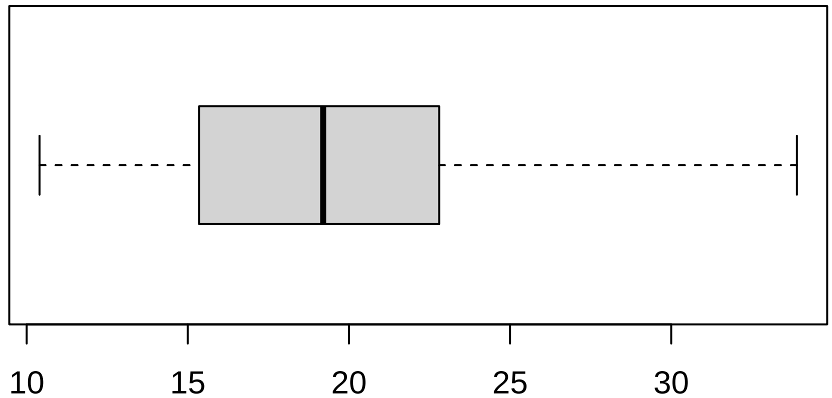
A commonly used plot to display relevant statistics is the box plot. To create a box plot use the `boxplot()`. The function only needs the `x=` which specifies the data to create the box plot. Use the box plot function to create a box plot on for the variable `mpg`.

```
boxplot(mtcars$mpg)
```



If you want to make the box plot horizontal, use `horizontal=` and set it equal to `TRUE`.

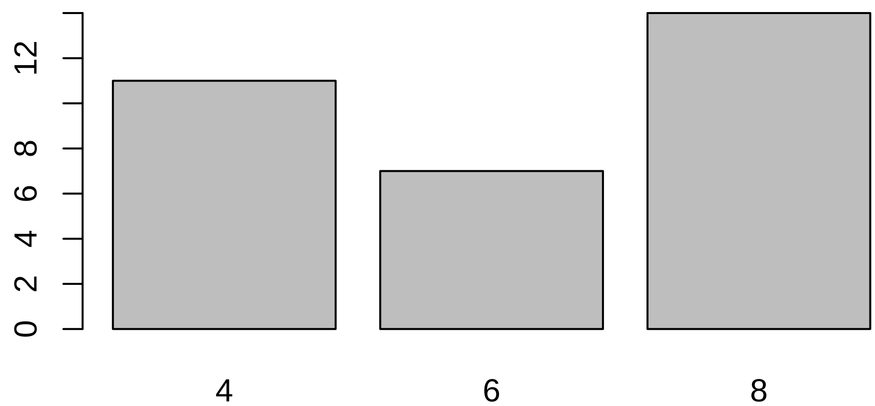
```
boxplot(mtcars$mpg, horizontal = TRUE)
```



8.1.8 Bar Chart

A histogram shows you the frequency for a continuous variable. A bar chart will show you the frequency of a categorical or discrete variable. To create a bar chart, use the `barplot()`. The main argument it needs is the `height=` which needs to an object from the `table()`. Create a bar chart for the `cyl` variable.

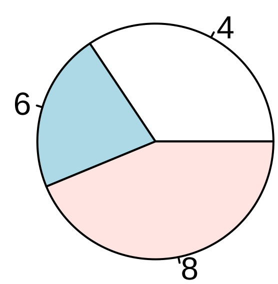
```r
barplot(table(mtcars$cyl))
```



8.1.9 Pie Chart

While I do not recommend using a pie chart, R is capable of creating one using the `pie()`. It only needs the `x=` which is a vector numerical quantities. This could be the output from the `table()`. Create a pie chart with the `cyl` variable.

```r
pie(table(mtcars$cyl))
```



8.1.10 Grouping

Similar to obtaining statistics for certain groups, plots can be grouped to reveal certain trends. We will look at a couple of methods to visualize different groups.

8.1.10.1 One Variable Grouping

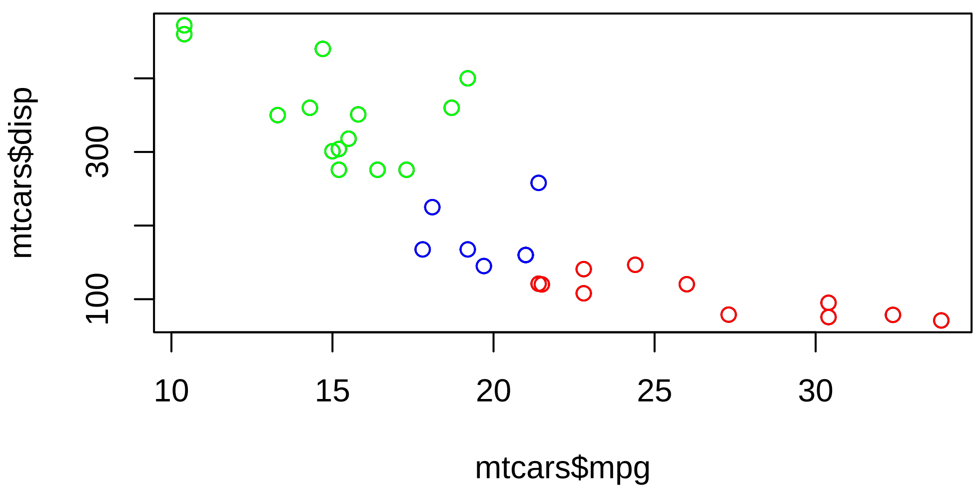
Two ways to display groups is by using color coding or panels. I will show you what I think is the best way to group variables. There may be better ways to do this, such as using the ggplot2 package. Before we begin, create three new R objects that are a subset of the `mtcars` data set into 3 different data sets with for the three different values of the `cyl` variable: "4", "6", and "8". use the `subset()` to create the different data sets. Name the new R objects `mtcars_4`, `mtcars_6`, and `mtcars_8`, respectively.

```
mtcars_4 <- subset(mtcars, cyl == 4)
mtcars_6 <- subset(mtcars, cyl == 6)
mtcars_8 <- subset(mtcars, cyl == 8)
```

8.1.10.1.1 Scatter Plot

To create different colors points for their respective label associated `cyl` variable. First create a base scatter plot using the `plot()` to set up the plot. Then one by one, overlay a set of new points on the base plot using the `points()`. The first two arguments should be the vectors of data from their respective R object subset. Also, use the `col=` to change the color of the points. The `col=` takes either a string or a number.

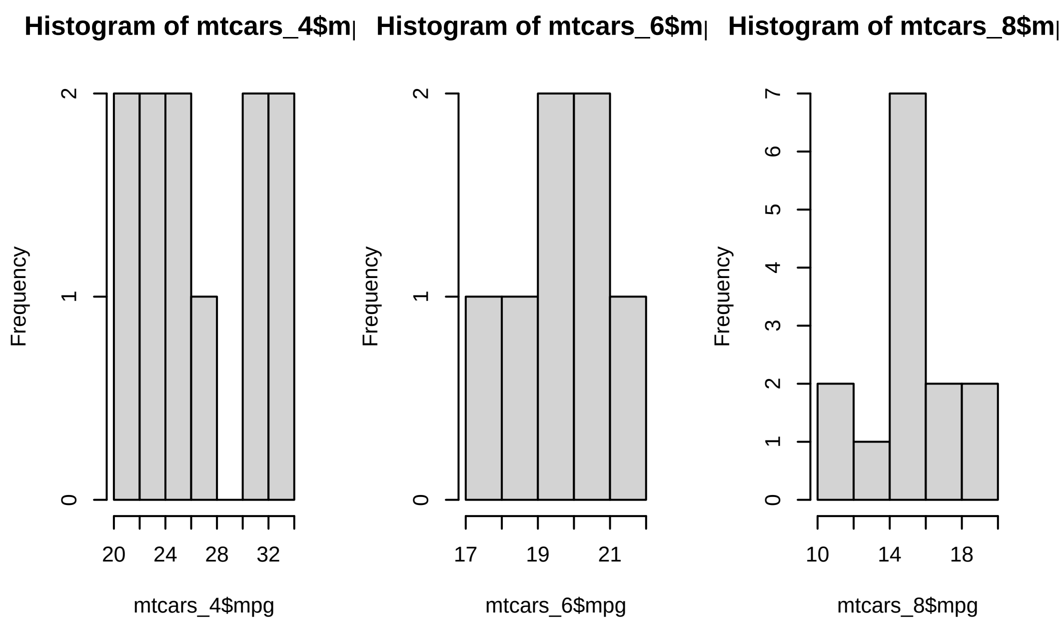
```
plot(mtcars$mpg, mtcars$disp)
points(mtcars_4$mpg, mtcars_4$disp, col = "red")
points(mtcars_6$mpg, mtcars_6$disp, col = "blue")
points(mtcars_8$mpg, mtcars_8$disp, col = "green")
```



8.1.10.1.2 Histogram

Now, it us more difficult to overlay histograms on a plot to different colors. Therefore, a panel approach may be more beneficial. This can be done by setting up R to plot a grid of plots. To do this, use the `par()` to tell R how to set up the grid. Then use the `mfrow=`, which is a vector of length two, to set up a grid. The `mfrow=` usually has an input of `c(ROWS,COLS)` which states the number of rows and the number of columns. Once this is done, the next plots you create will be used to populate the grid.

```
par(mfrow=c(1,3))
hist(mtcars_4$mpg)
hist(mtcars_6$mpg)
hist(mtcars_8$mpg)
```

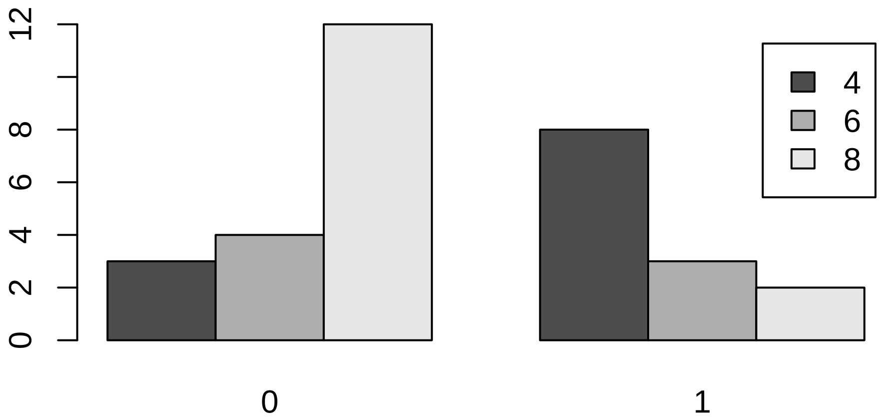


Every time you use the `par()`, it will change how graphics are created in an R session. Therefore, all your plots will follow the new graphic parameters. You will need to reset it by typing `dev.off()`.

8.1.10.1.3 Bar Chart

To visualize two categorical variables, we can use a color-coded bar chart to compare the frequencies of the categories. This is simple to do with the `barplot()`. First, use the `table()` to create a cross-tabulation of the frequencies for two variables. Then use the `boxplot()` to visualize both variables. Then use `legend=` to create a label when the bar chart is color-coded. Additionally, use the `beside=` argument to change how the plot looks. Use the code below to compare the variables `cyl` and `am` variable.

```
barplot(table(mtcars$cyl, mtcars$am), beside = TRUE, legend = rownames(table(mtcars$cyl, m
```



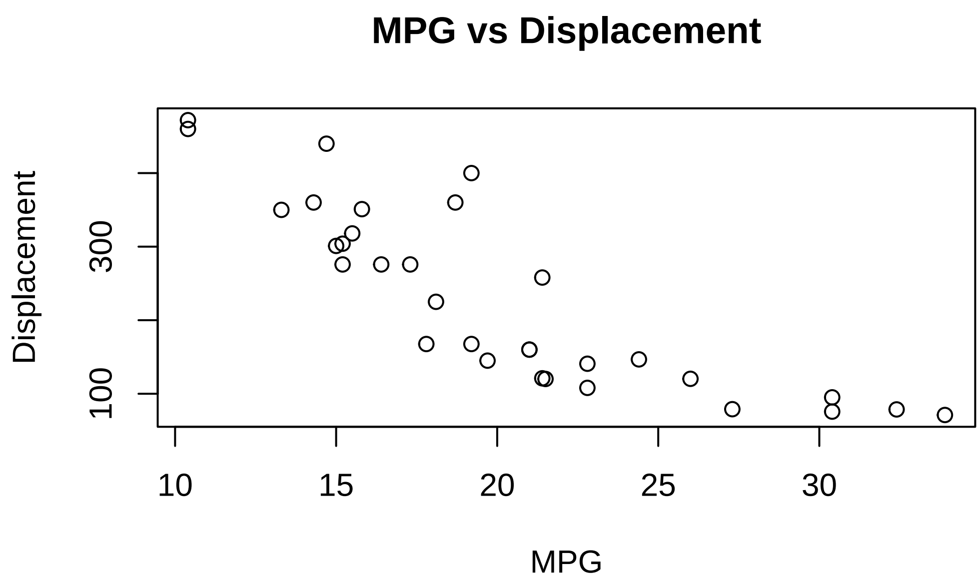
Notice that I use the `rownames()` to label the legend.

8.1.11 Tweaking

8.1.11.1 Labels

The main tweaking of plots I will talk about is changing the the axis label and titles. For the most part, each function allows you to use the `main=`, `xlab=`, and `ylab=`. The `main=` allows you to change the title. The `xlab=` and `ylab=` allow you to change the labels for the x-axis and y-axis, respectively. Create a scatter plot for the variables `mpg` and `disp` and change the labels.

```
plot(mtcars$mpg, mtcars$disp, main = "MPG vs Displacement", xlab = "MPG", ylab = "Displace
```



8.2 ggplot2

8.2.1 Introduction

The `ggplot2::` provides a set of functions to create different graphics. For more information on plotting in `ggplot2::`, please visit the this excellent [resource](). Here we will discuss some of the basics to the `ggplot2::``. To me,`ggplot2::‘`creates a plot by adding layers to a base plot. The syntax is designed for you to change different components of a plot in an intuitive manner. For this tutorial, we will focus on plotting different components from the`mpg‘ data set.

8.2.1.1 Contents

1. Basic
2. Grouping
3. Themes/Tweaking

8.2.2 Basics

To begin, the `ggplot2::` really works well when you are using data frames. If you have any output that you want to plot, convert into to a data frame. Once we have our data set, the first thing you would want to do is specify the main components of your base plot. This will be what will be plotted on your x-axis, and what will be plotted on your y-axis. Next, you will create the the type of plot. Lastly, you will add different layers to tweak the plot for your needs. This can be changing the layout or even overlaying another plot. The 'ggplot2::" provides you with tools to do almost everything you need to create a plot easily.

Before we begin plotting, load the `ggplot2::` in R.

```
library(ggplot2)
```

Now, when we create a base plot, we will use the `ggplot()`. This will initialize the data that we need to use with the `data=` and how to map it on the x and y axis with the `mapping=`. With the `mapping=`, you will need to use the `aes()` which constructs the mapping function for the base plot. The `aes()` requires the `x=` and optionally uses the `y=` to set which values represents the x and y axis. The `aes()` also accepts other arguments for grouping or other aesthetics.

Before we begin, create a new variable in `mtcars` called `ind` and place a numeric vector which contains integers from 1 to 32.

```
mtcars$ind <- c(1:32)
```

Now, let's create the base plot and assign it to `gg_1`. Use the `ggplot()` and set `mtcars` as its data and the variable `ind` as `x=` and `mpg` as the `y=`

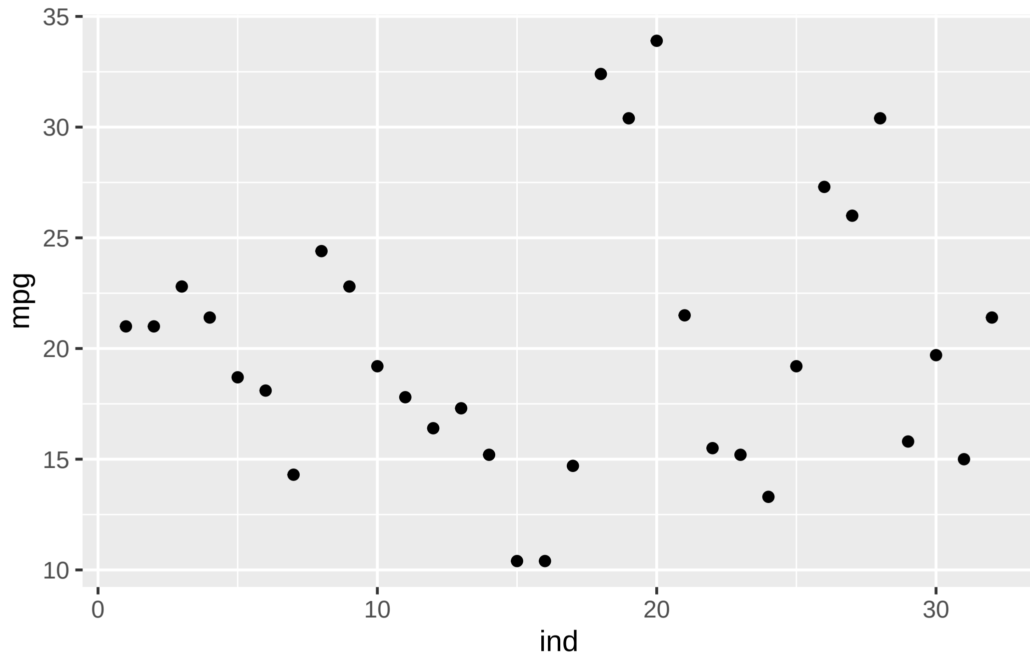
```
gg_1 <- ggplot(mtcars, aes(ind, mpg))
```

This base plot is now used to create certain plots. Plots are created by adding functions to the base plot. This is done by using the `+` operator and then a specific `ggplot2::` function. Below we will go over some of the functions necessary.

8.2.3 Scatter Plot

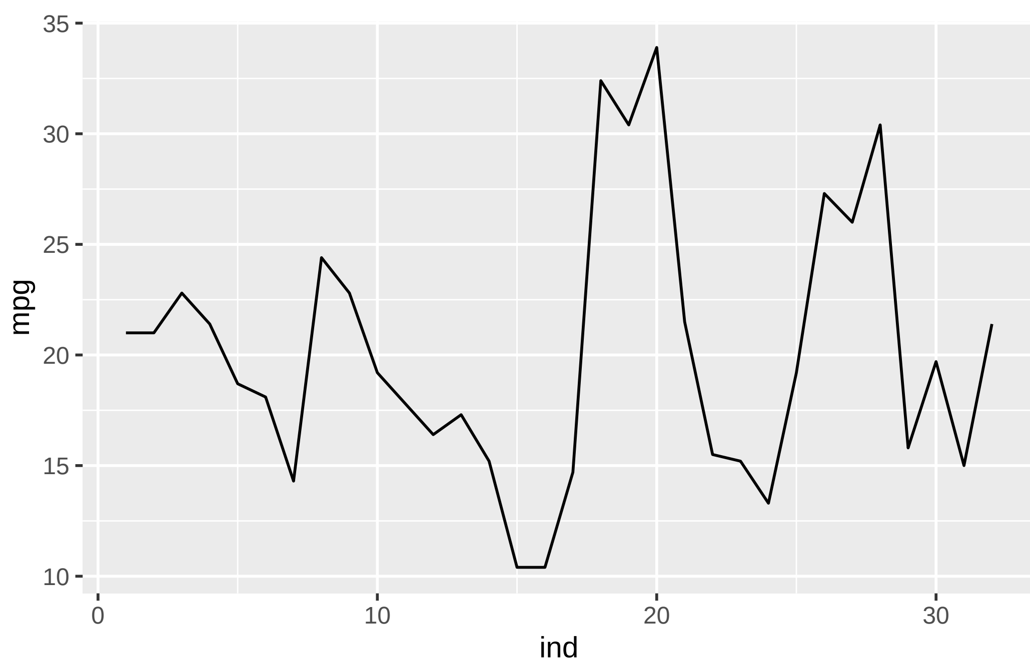
To create a scatter plot in `ggplot2::`, add the `geom_point()` to the base plot. You do not need to specify any arguments in the function. Create a scatter plot to `gg_1`

```
gg_1 + geom_point()
```



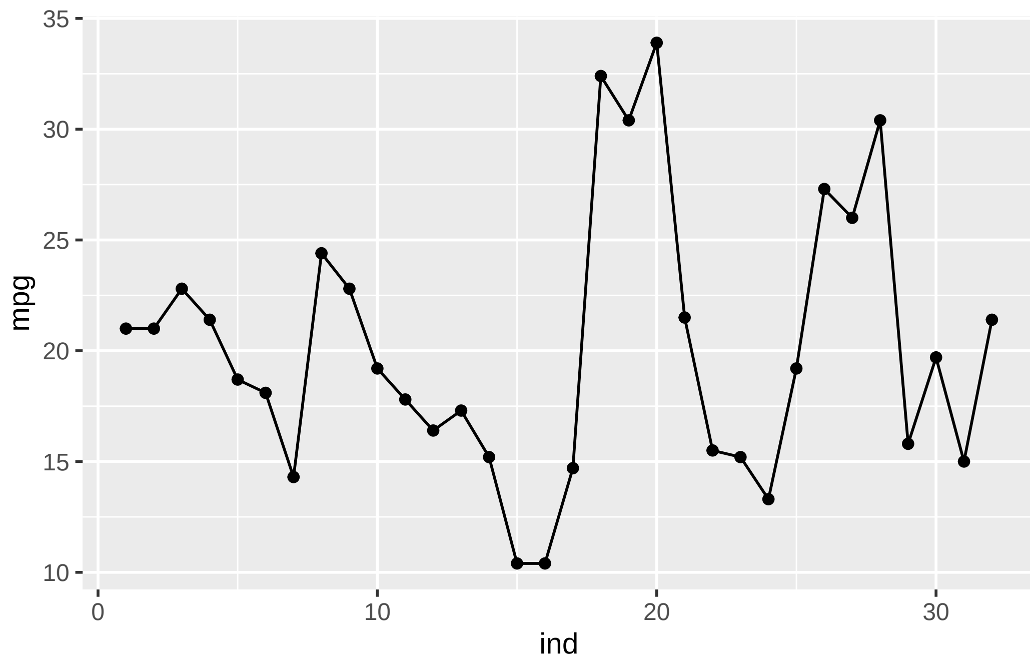
If we want to put lines instead of points, we will need to use the `geom_point()`. Change the points to a line.

```
gg_1 + geom_line()
```



To overlay points to the plot, add `geom_point()` as well as `geom_line()`. Add points to the plot above.

```
gg_1 + geom_point() + geom_line()
```

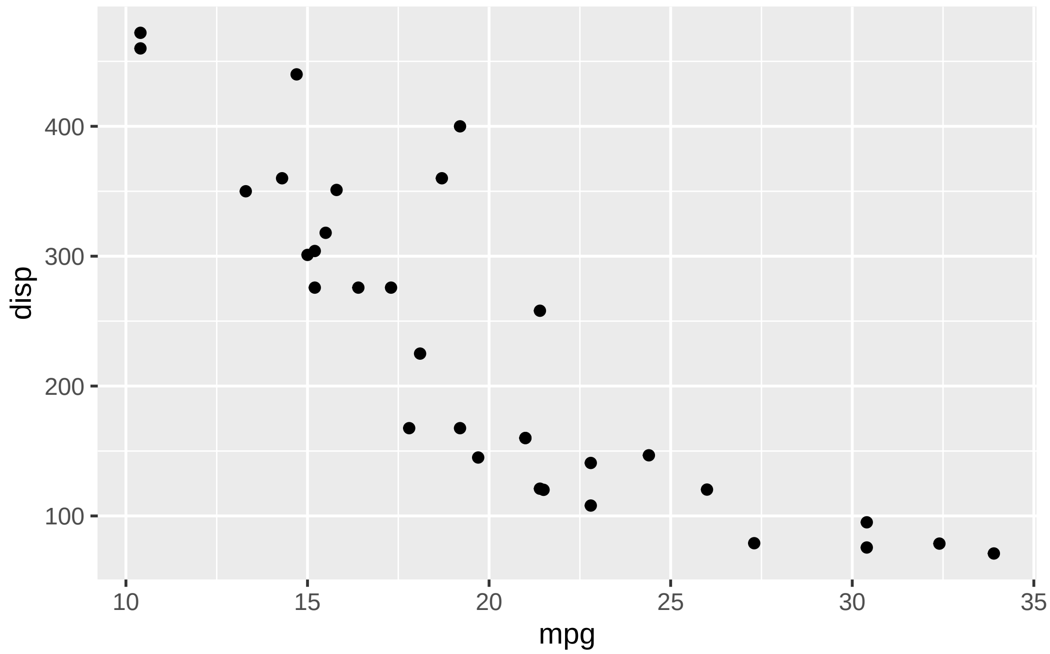


To create a 2 variable scatter plot. You will just need to specify the `x=` and `y=` in the `aes()`. Create a base plot using the `mtcars` data set and use the `mpg` and `disp` as the x and y variables, respectively, and assign in it to `gg_2`

```
gg_2 <- ggplot(mtcars, aes(mpg, disp))
```

Now create a scatter plot using `gg_2`.

```
gg_2 + geom_point()
```



8.2.4 Histogram and Density Plot

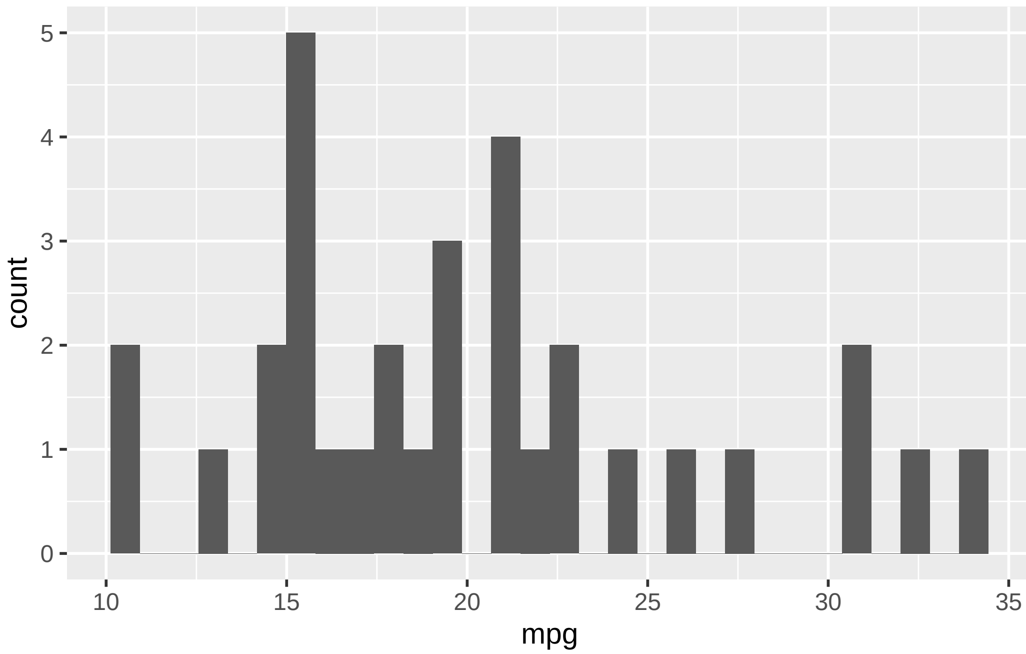
To create a histogram and density plots, create a base plot and specify the variable of interest in the `aes()`, only specify one variable. Create a base plot using the `mtcars` data set and the `mpg` variable. Assign it to `gg_3`.

```
gg_3 <- ggplot(mtcars, aes(mpg))
```

To create a histogram, use the `geom_histogram()`.

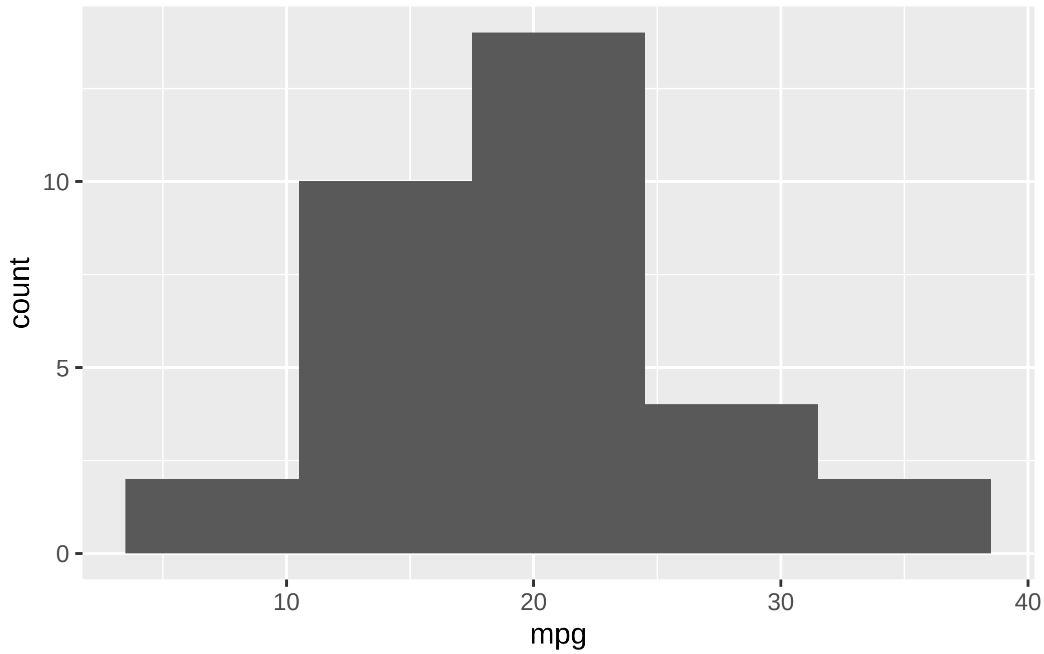
```
gg_3 + geom_histogram()
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The above plot shows a histogram, but the number of bins is quite large. We can change the bin width argument, `binwidth=`, the the `geom_histogram()`. Change the bin width to seven.

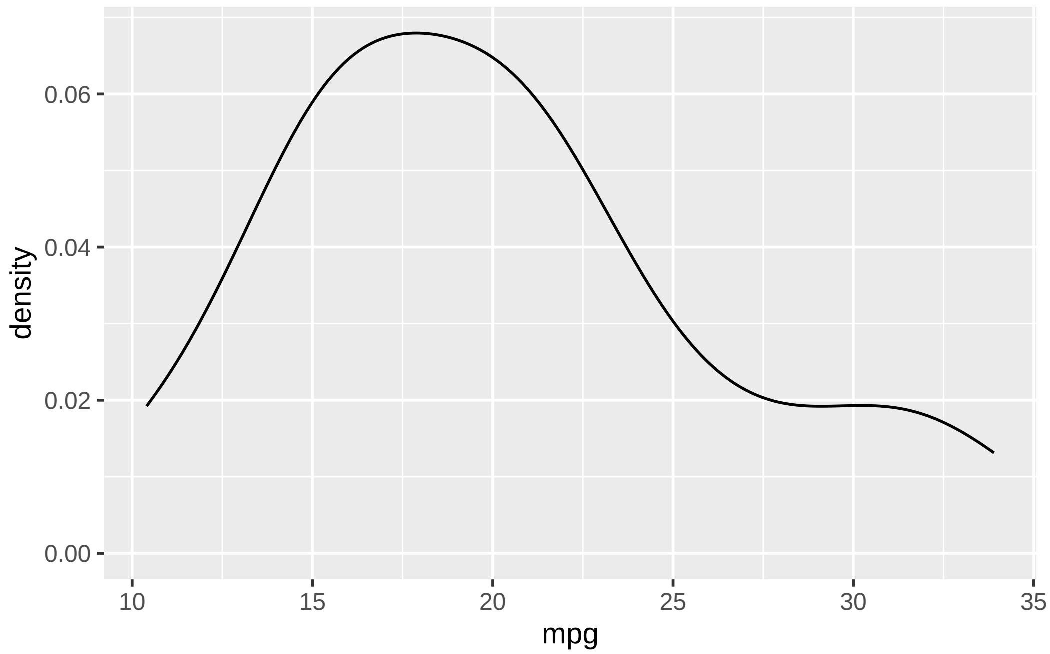
```
gg_3 + geom_histogram(binwidth = 7)
```



8.2.4.1 Density Plot

To create a density plot, use the `geom_density()`. Create a density plot for the `mpg` variable.

```
gg_3 + geom_density()
```

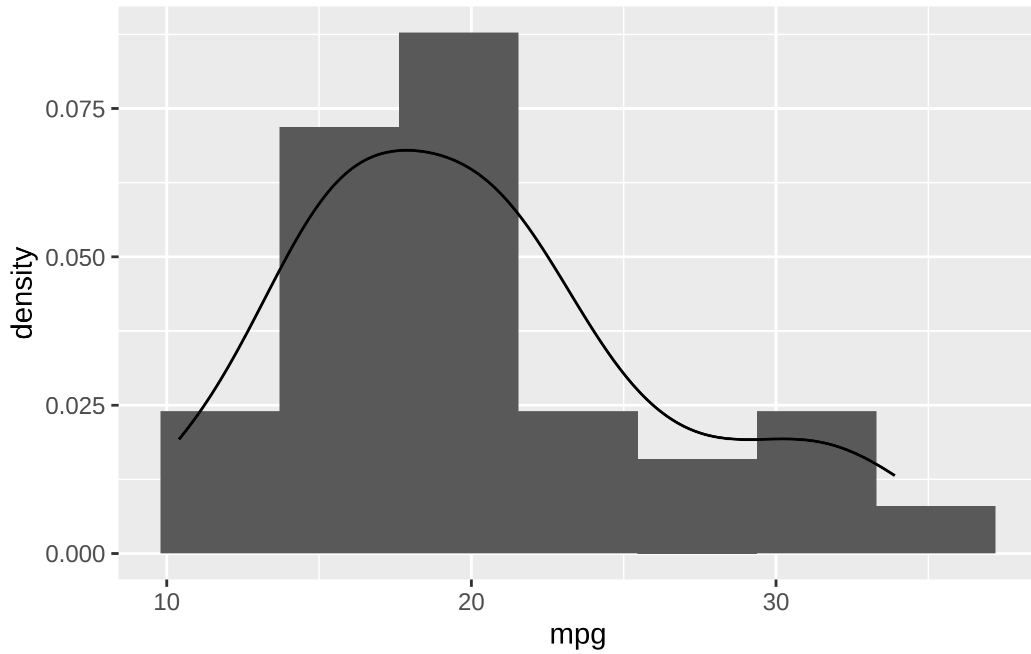


8.2.4.2 Both

Similar to adding lines and points in the same plot, you can add a histogram and a density plot by adding both the `geom_histogram()` and `geom_density()`. However, in the `geom_histogram()`, you must add `aes(y=..density..)` to create a frequency histogram. Create a plot with a histogram and a density plot.

```
gg_3 + geom_histogram(aes(y=..density..),bins=7) +
  geom_density()
```

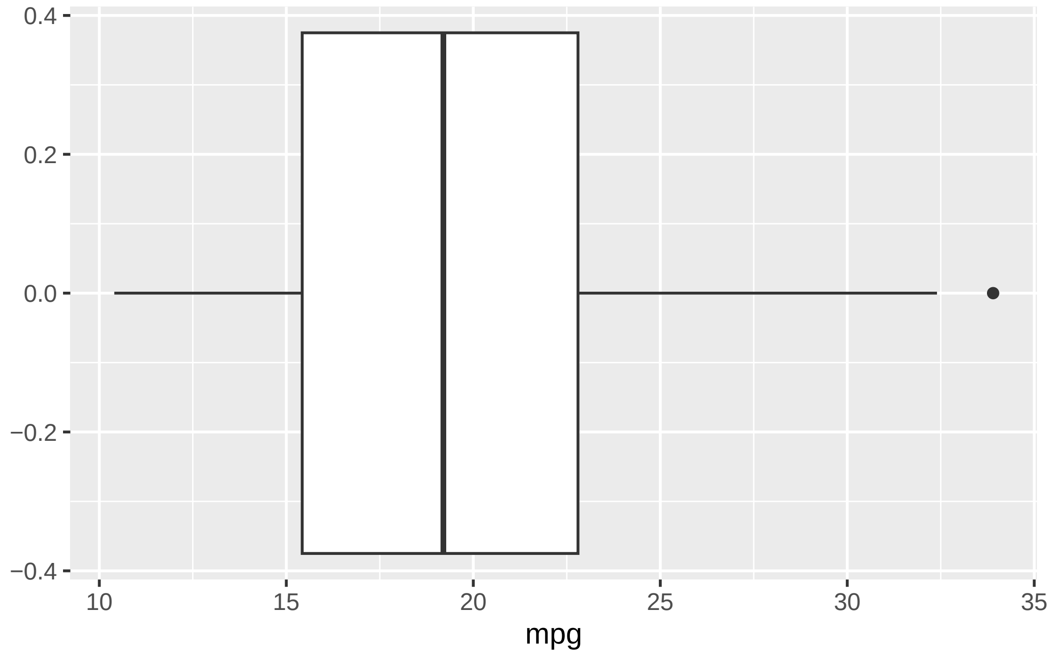
```
Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
i Please use `after_stat(density)` instead.
```



8.2.5 Box Plots

If you need to create a box plot, use the `stat_boxplot()`. Create a boxplot for the variable `mpg`. All you need to do is add `stat_boxplot()`.

```
gg_3 + stat_boxplot()
```



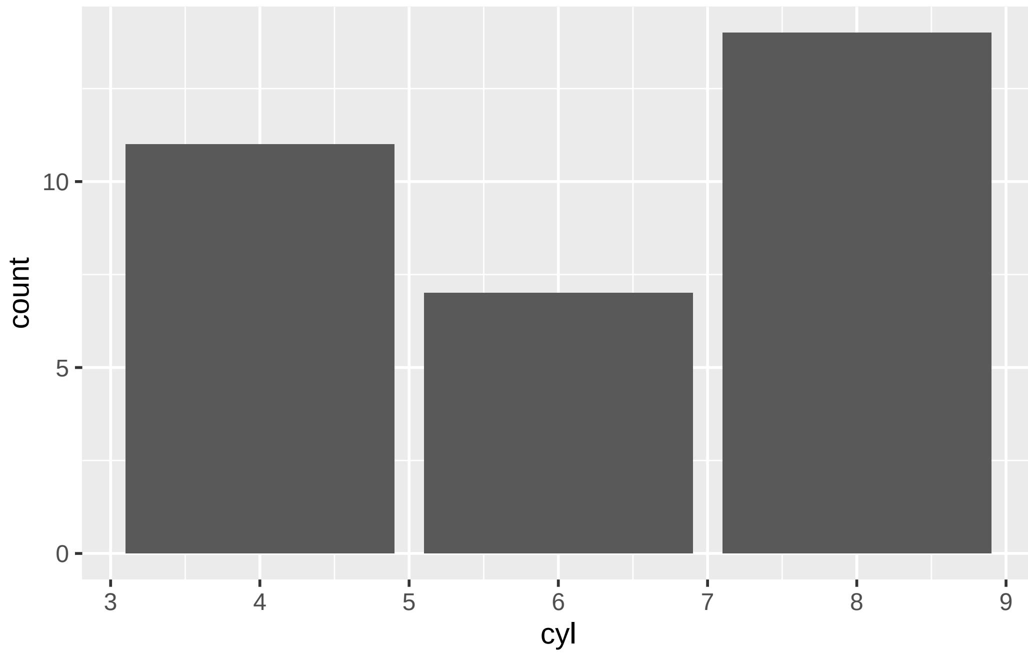
8.2.6 Bar Charts

Creating a bar chart is similar to create a box plot. All you need to do is use the `stat_count()`. First create a base plot using the `mtcars` data sets and the `cyl` variable for the mapping and assign it to `gg_4`.

```
gg_4 <- ggplot(mtcars, aes(cyl))
```

Now create the bar plot by adding the `stat_count()`.

```
gg_4 + stat_count()
```



8.2.7 Grouping

The 'ggplot2::" easily allows you to create plots from different groups. We will go over some of the arguments and functions to do this.

8.2.7.1 One Variable Grouping

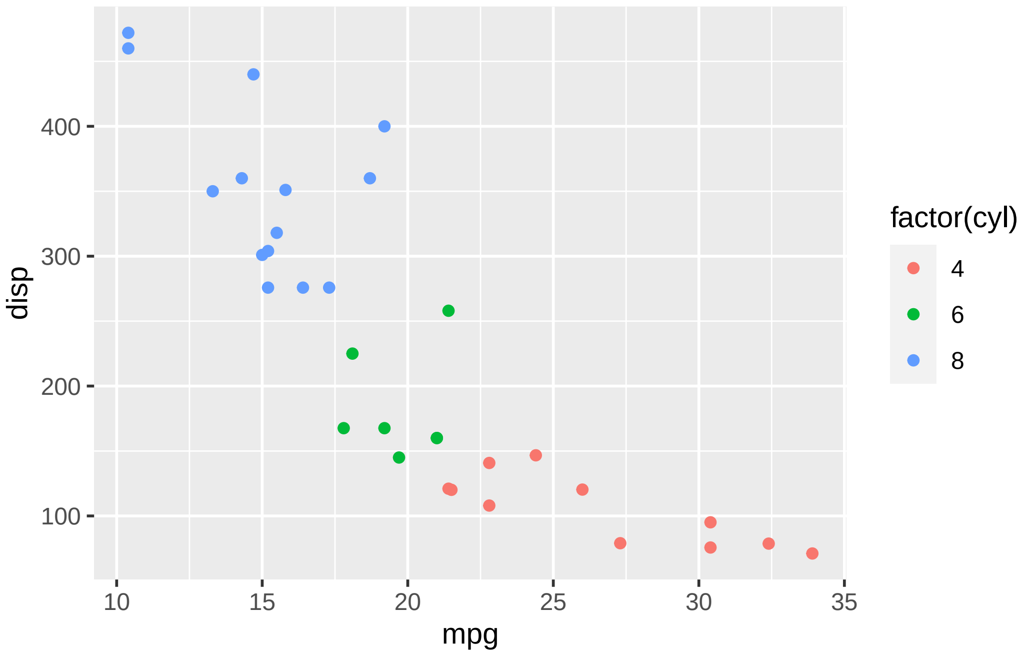
8.2.7.1.1 Scatter Plot

To begin, we want to specify the grouping variable within the `aes()` with the `color=`. Additionally, the argument works best with a factor variable, so use the `factor()` to create a factor variable. Create a base plot from the `mtcars` data set using `mpg` and `disp` for the x and y axis, respectively, and set the `color=` equal to the `factor(cyl)`. Assign it the R object `gg_5`.

```
gg_5 <- ggplot(mtcars, aes(mpg, disp, color=factor(cyl)))
```

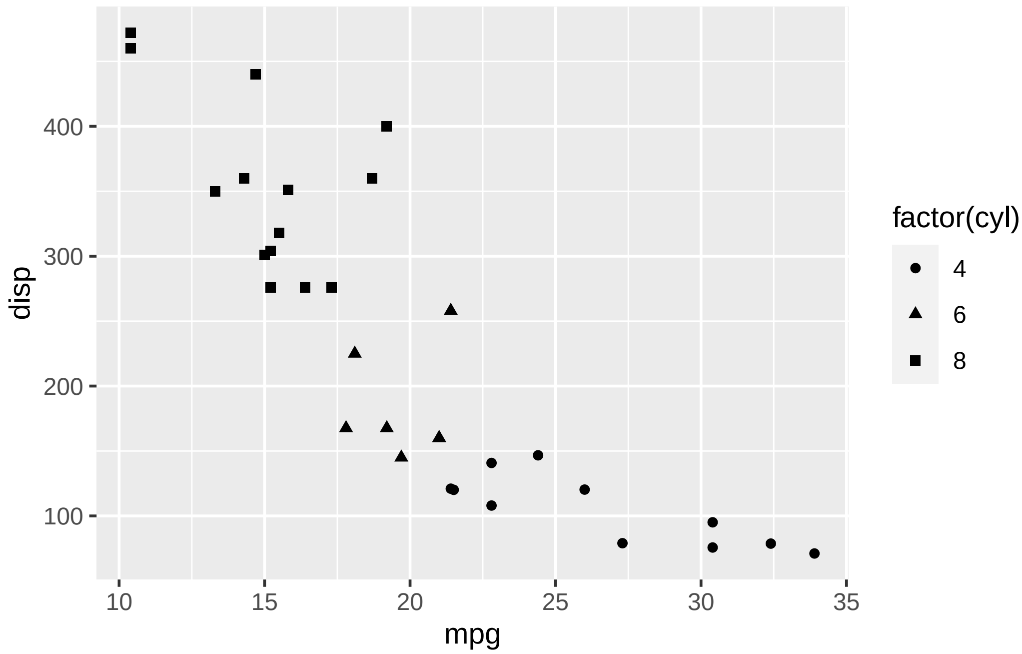
Once the base plot is created, 'ggplot2::" will automatically group the data in the plots. Create the scatter plot from the base plot.

```
gg_5 + geom_point()
```



If you want to change the shapes instead of the color, use the `shape=`. Create a base plot from the `mtcars` data set using `mpg`, and `disp` for the x and y axis, respectively, and group it by `cyl` with the `shape=`. Assign it the R object `gg_6`.

```
gg_6 <- ggplot(mtcars, aes(mpg, disp, shape=factor(cyl)))
gg_6 + geom_point()
```



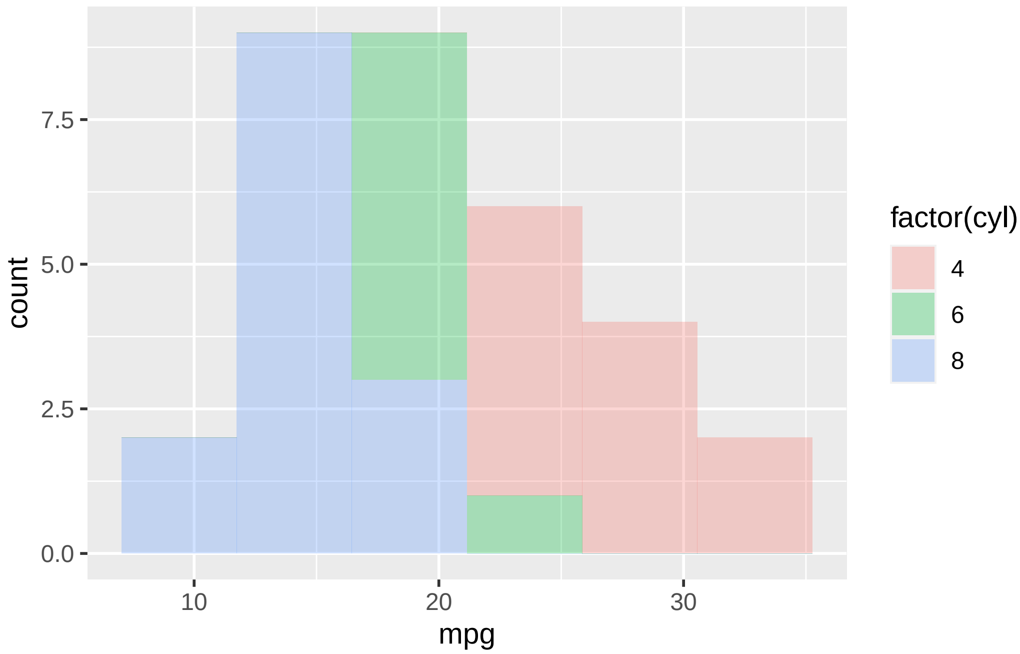
8.2.7.1.2 Histograms

Histograms can be grouped by different colors. This is done by using the `fill=` within the `aes()` in the base plot. Assign it the R object `gg_7`.

```
gg_7 <- ggplot(mtcars, aes(mpg, fill = factor(cyl)))
```

Now create a histogram from the base plot `gg_7`.

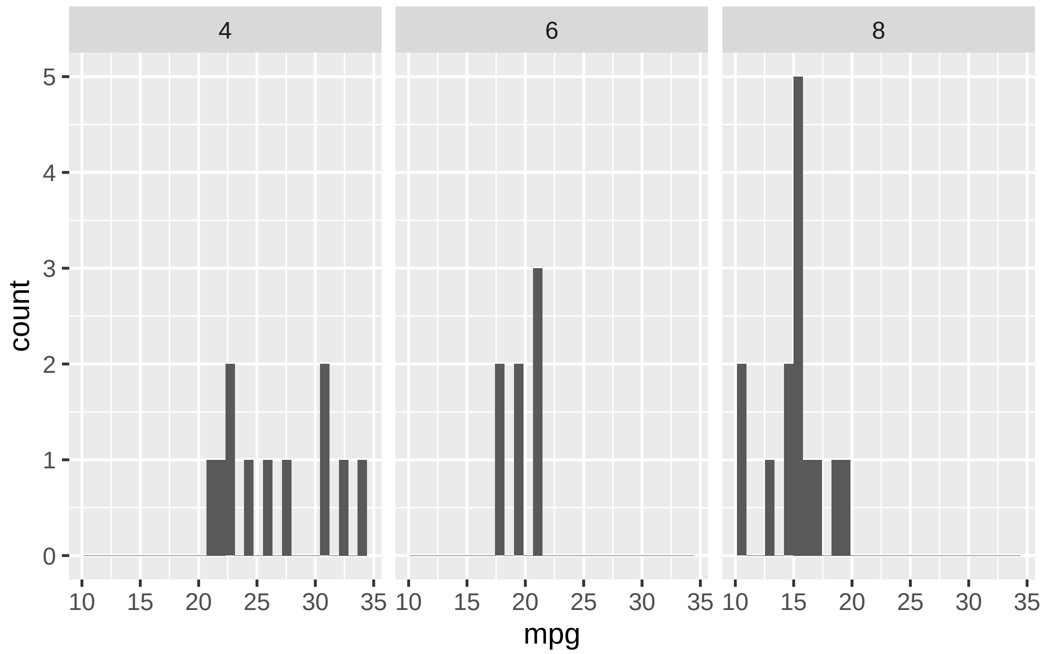
```
gg_7 + geom_histogram(bins = 6, alpha = 0.3)
```



Sometimes we would like to view the histogram on separate plots. The `facet_wrap()` and the `flact_grid()` allows this. Using either function, you do not need to specify the grouping factor in the `aes()`. You will add `facet_wrap()` to the plot. It needs a formula argument with the grouping variable. Using the R object `gg_3` create side by side plots using the `cyl` variable. Remember to add `geom_histogram()`.

```
gg_3+geom_histogram() + facet_wrap( ~ cyl)
```

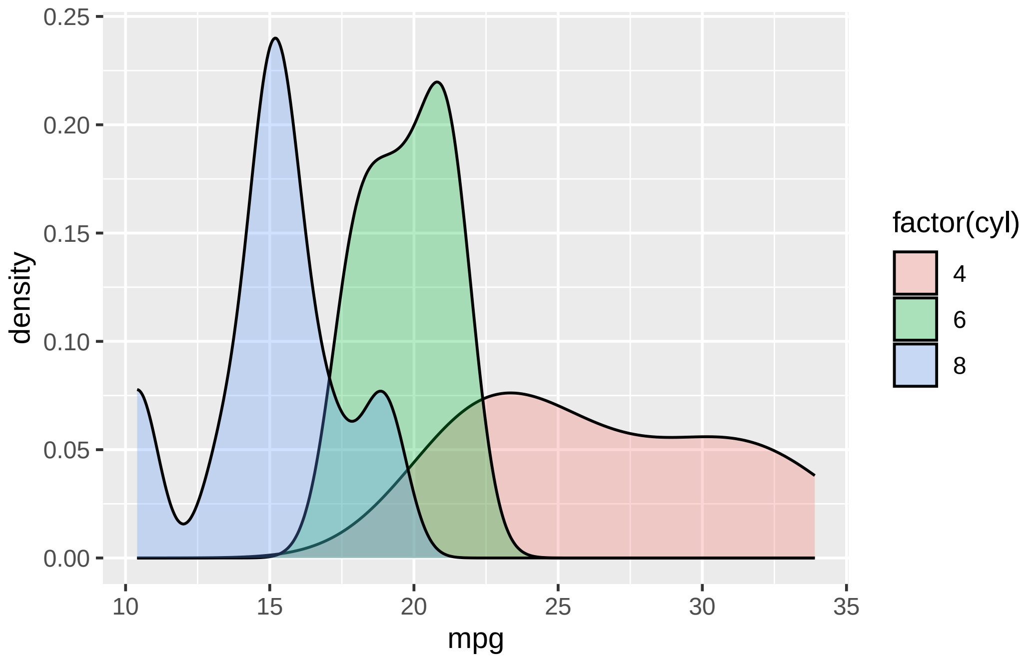
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



8.2.7.1.3 Density Plot

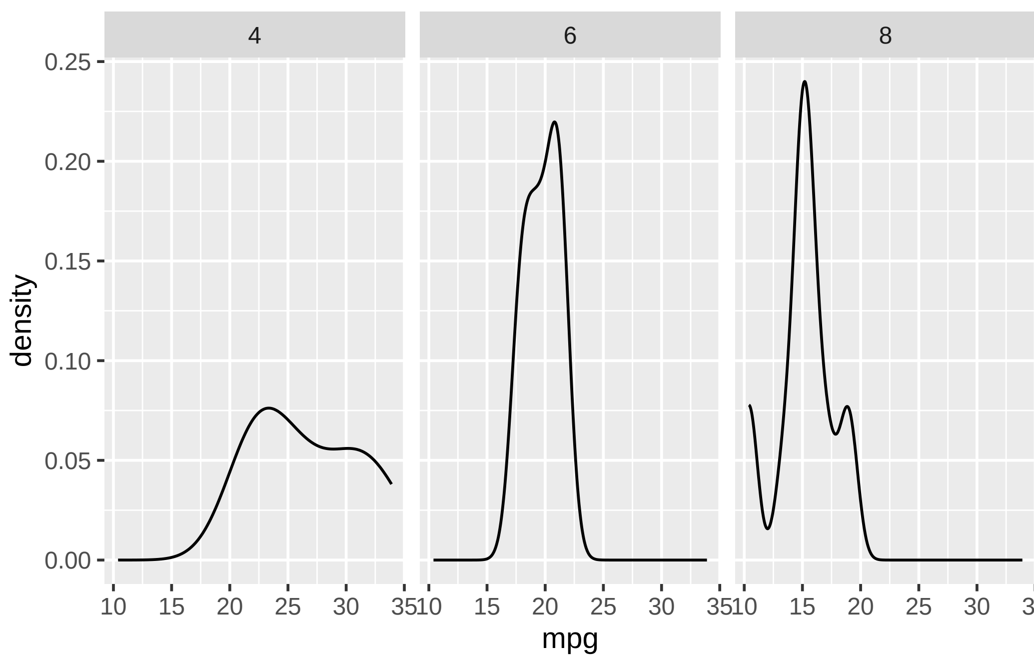
Similar to histograms, density plots can be grouped by variables the same way. Using `gg_7`, create color-coded density plots. All you need to do is add `geom_density()`.

```
gg_7 + geom_density(alpha=0.3)
```



Using `gg_3`, create side by side density plots. You need to do is add `geom_density()` and `facet_wrap()` to group with the `cyl` variable.

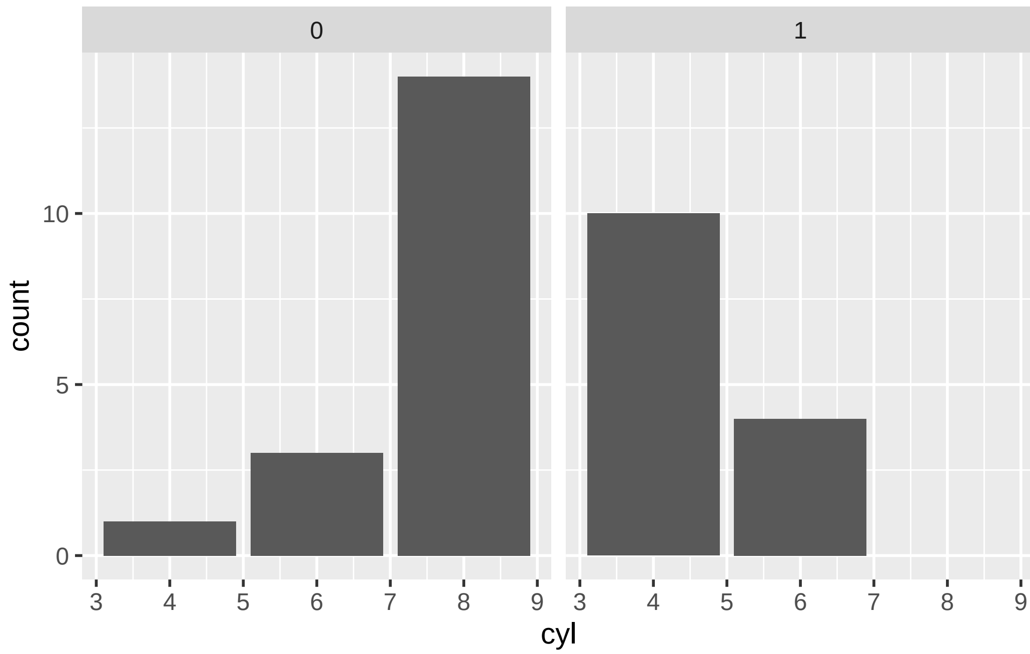
```
gg_3 + geom_density() + facet_wrap( ~ cyl)
```



8.2.7.1.4 Bar Chart

To create a side by side bar plot, you can use the `facet_wrap()` with a grouping variable. Using `gg_4`, create a side by side bar plot using `vs` as the grouping variable. Remember to add `stat_count()` as well.

```
gg_4 + stat_count() + facet_wrap(~vs)
```

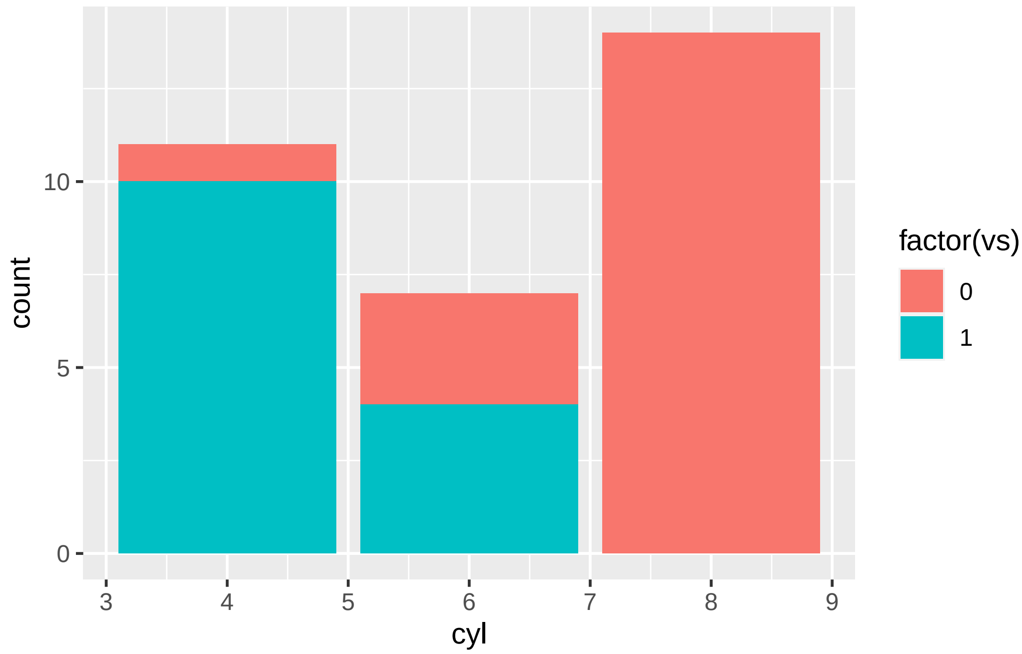


If you want to compare the bars from different group in one plot, you can use the `fill=` from the `aes()`. The `fill=` just needs a factor variable (use `factor()`). First create a base plot using the data `mtcars`, variable `cyl` and grouping variable `vs`. Assign it to `gg_8`.

```
gg_8 <- ggplot(mtcars, aes(cyl, fill = factor(vs)))
```

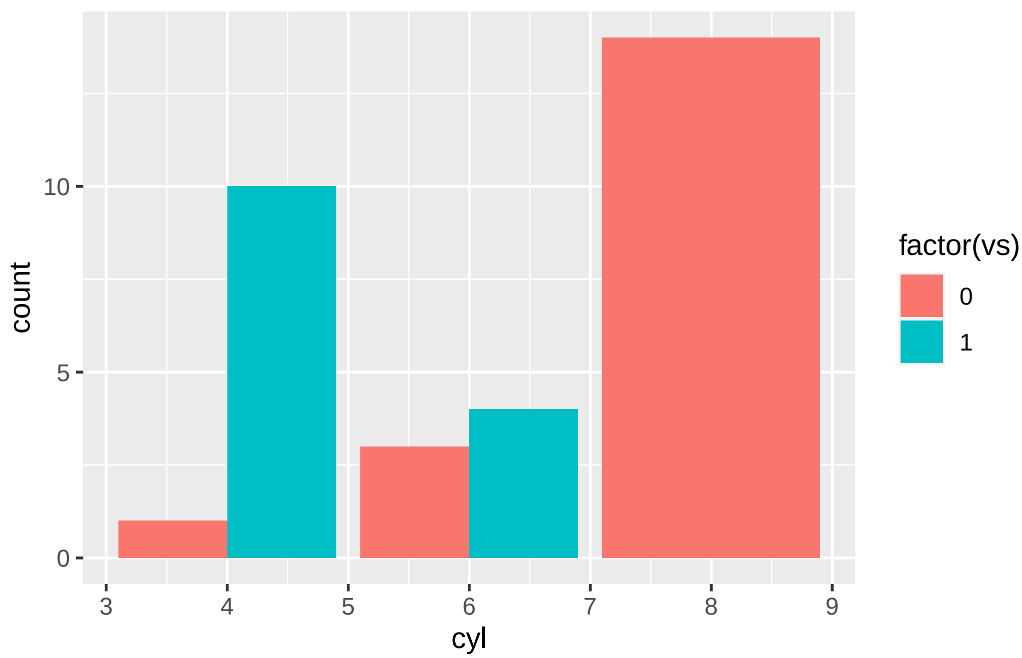
Now create a bar chart by adding `stat_count()`.

```
gg_8 + stat_count()
```



If you want to grouping bars to be side by side, use the `position=` in the `stat_count()` and set it equal to `"dodge"`. Create the bar plot using the `position = "dodge"`.

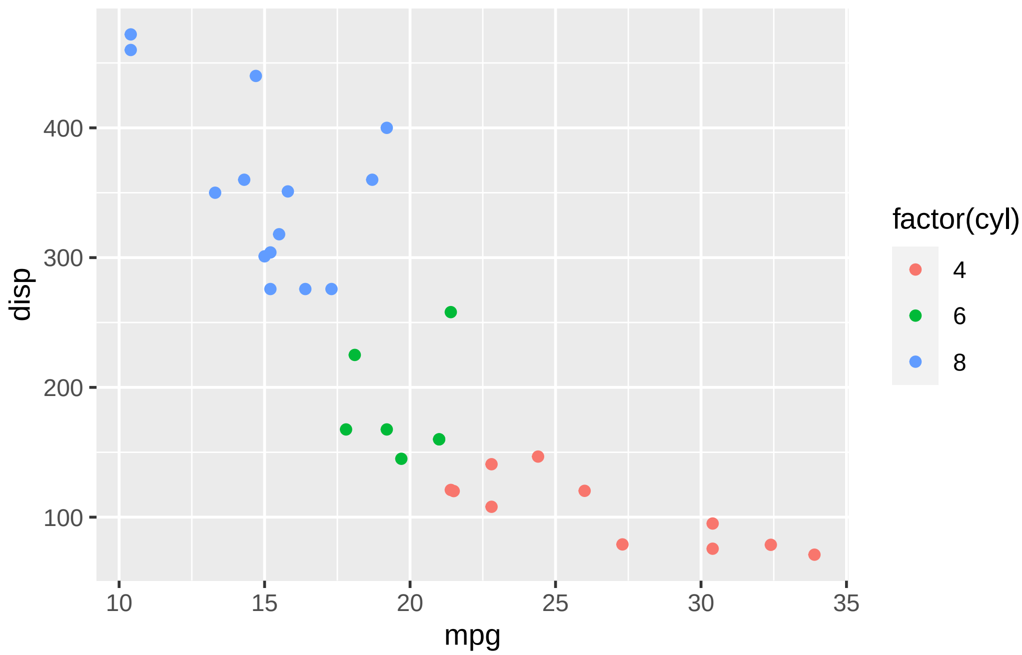
```
gg_8 + stat_count(position = "dodge")
```



8.2.8 Themes/Tweaking

In this section, we will talk about the basic tweaks and themes to `ggplot2::`. However. `ggplot2::` is much more powerful and can do much more. Before we begin, lets look at object `gg_9` to understand the plot. To view a plot, use the `plot()`.

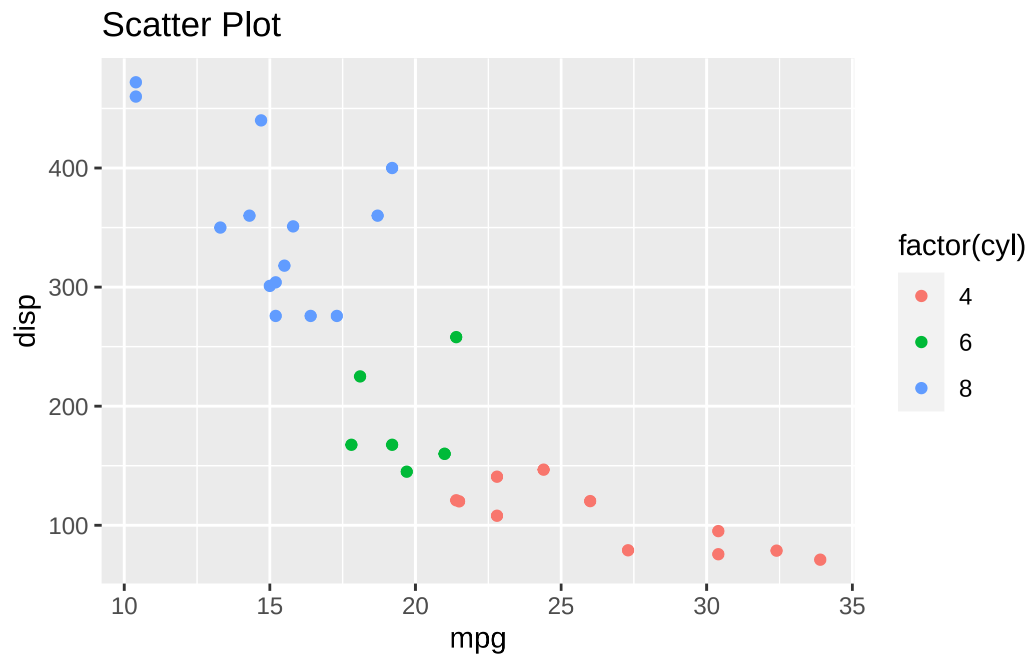
```
plot(gg_9)
```



8.2.8.1 Title

To change the title, add the `ggtitle()` to the plot. Put the new title in quotes as the first argument. Change the title for `gg_9`.

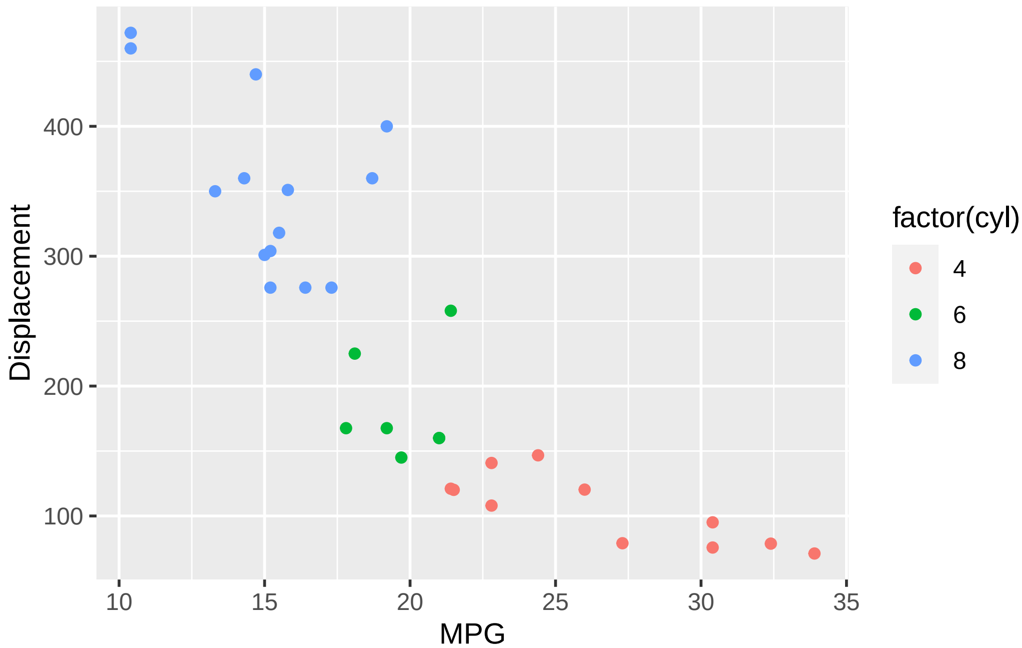
```
gg_9 + ggtitle("Scatter Plot")
```



8.2.8.2 Axis

Changing the labels for a plot, add the `xlab()` and `ylab()`, respectively. The first argument contains the phrase for the axis. Change the axis labels for `gg_9`.

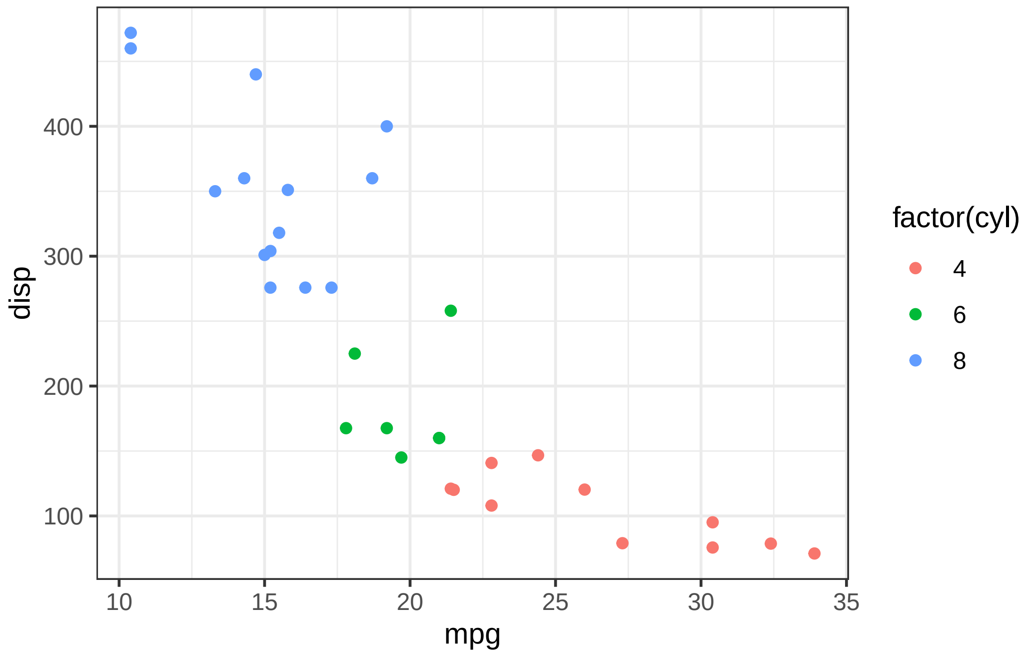
```
gg_9 + xlab("MPG") + ylab("Displacement")
```



8.2.8.3 Themes

If you don't like how the plot looks, `ggplot2::` has custom themes you can add to the plot to change it. These functions usually are formatted as `theme_*()`, where the `*` indicates different possibilities. I personally like how `theme_bw()` looks. Change the theme of `gg_9`.

```
gg_9 + theme_bw()
```



Additionally, you can change certain part of the theme using the `theme()`. I encourage you to look at what are other possibilities.

8.2.9 Saving plot

If you want to save the plot, use the `ggsave()`. Read the help documentation for the functions capabilities.

Part III

Reporting Data

9 Markdown Reports

10 Notebooks

11 Markdown Reports

Part IV

Debugging and Efficient Porgramming

12 Debugging Code

13 Efficient Programming and Profiling

14 Vectorizing Code

15 Incorporating C++ into R

Part V

Permutations

16 Permutation Tests

17 Permutation Regression

Part VI

Monte Carlo Methods

18 Monte Carlo Simulations

19 Monte Carlo Integration

20 Monte Carlo Hypothesis Testing

Part VII

Bootstrapping

21 Parametric Bootrapping

22 Nonparametric Boostrapping