# 2. WRITING CODE

## Writing code in the console window

Single lines of code can be written in the console window (bottom left). However, this code can not be saved or edited after execution. The console window is best utilized for experimenting with short lines of code, accessing help files, installing packages, and other tasks that do not need to be included in a master code document.

**PRACTICE:** To see how the console can be used to access help files, type ?head in the console and hit return. You should see a help document for the head command appear in the bottom right panel of RStudio.

## Writing code inline with text.

R code *can* be mixed in with Markdown text using back ticks as demonstrated below. The code will be replaced by the output in the knitted document. To see the output of inline text in the R Markdown file (before knitting), place the cursor inside the tick marks below and press "command + enter".

**PRACTICE** In the .Rmd file, place the cursor inside 4 and press "command/control + enter. In the knitted document you should see a 4.

Inline code can be useful for referencing values and output saved in the R environment, but is best avoided for everything else.

## Writing code inside code chunks

Anytime you write multiple lines of code that work together, placing the code inside of code chunks is more useful than writing inline code. The commands inside the code chunk can be executed individually using the shortcut keys "command/control + enter" or collectively using "shift + command/control + enter"

## Executing code & code chunks

Below is a new code chunk that prints a message, creates data, a dataframe, and a scatterplot, and creates a histogram of horse power from the built in R dataset called mtcars. Here are several ways to execute the commands in code chunks:

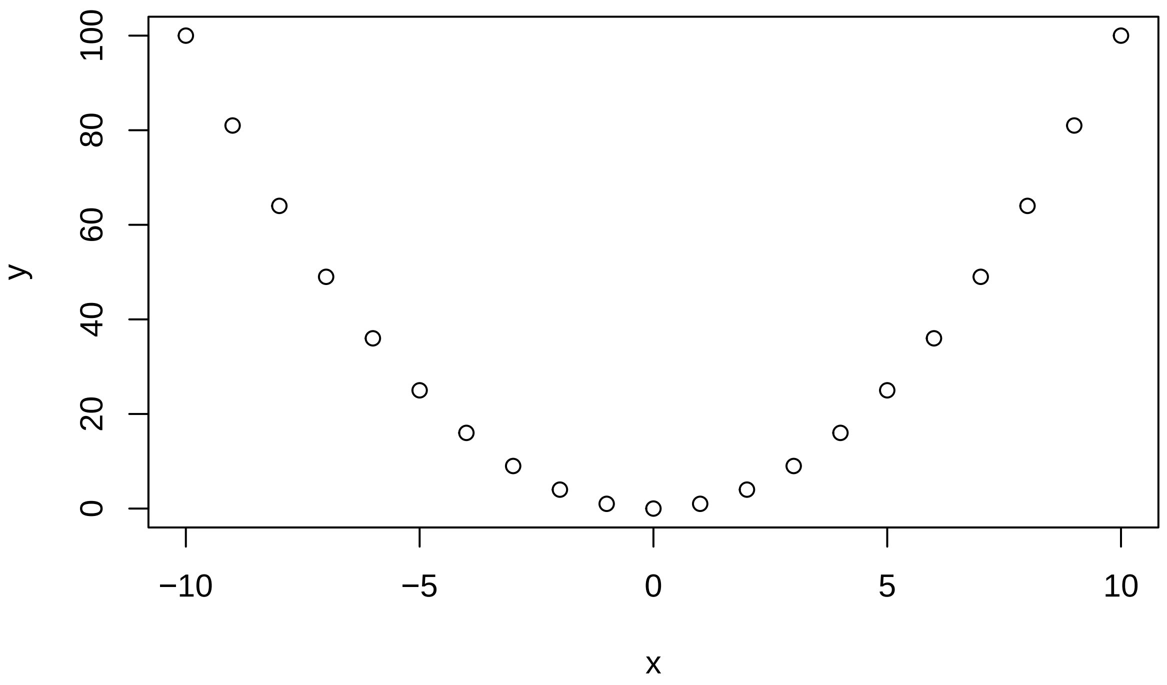
1. Execute the entire code chunk. Click the green arrow at the top right of the code chunk.
2. Execute the entire code chunk. With the cursor on any line of code, press "shift + Command + Enter".
3. Execute a single line. With the cursor anywhere on a line of code, press "command/control + Enter".
4. Execute all code chunks above a code chunk. Click the icon to the left of the green arrow at the right of a code chunk.
5. Execute all code chunks. Select this option from the "Run" pull down menu.

Experiment with each of the above approaches to run code in the chunk below.

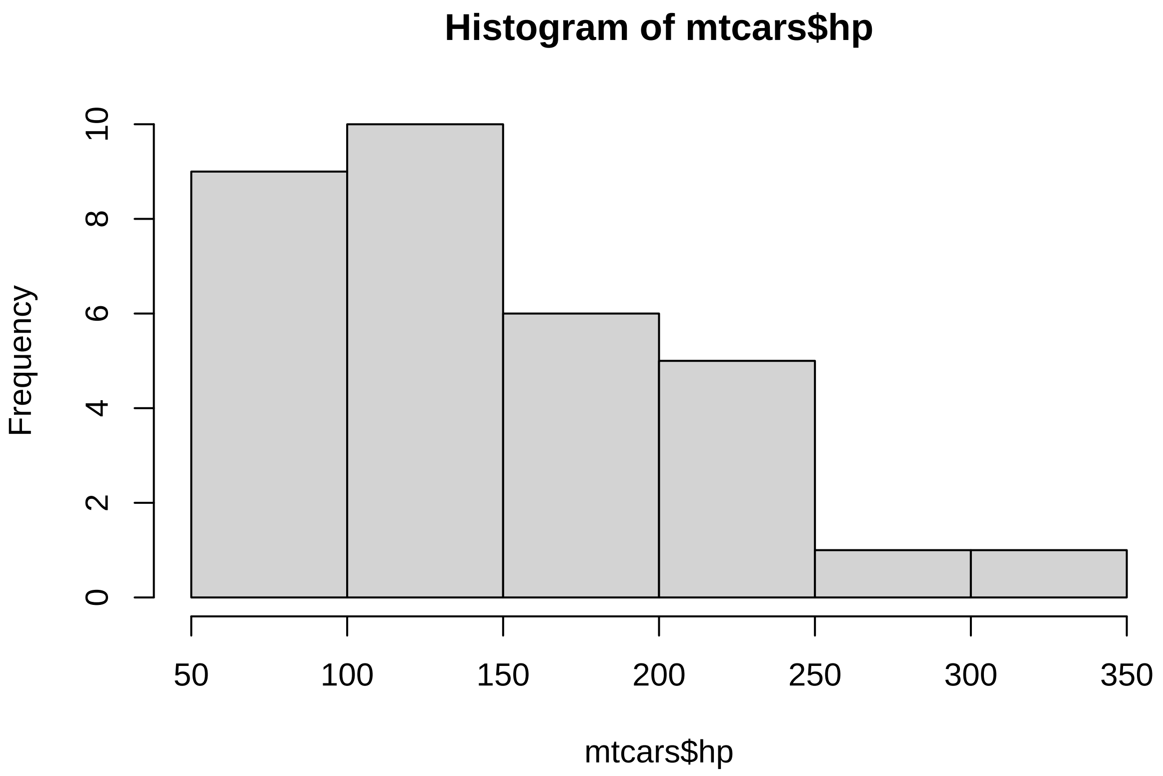
```r
print("Hello World")
```

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

```r
x <- -10:10
y <- x^2
df <- data.frame(x, y)
plot(x,y)
```



```r
hist(mtcars$hp)
```

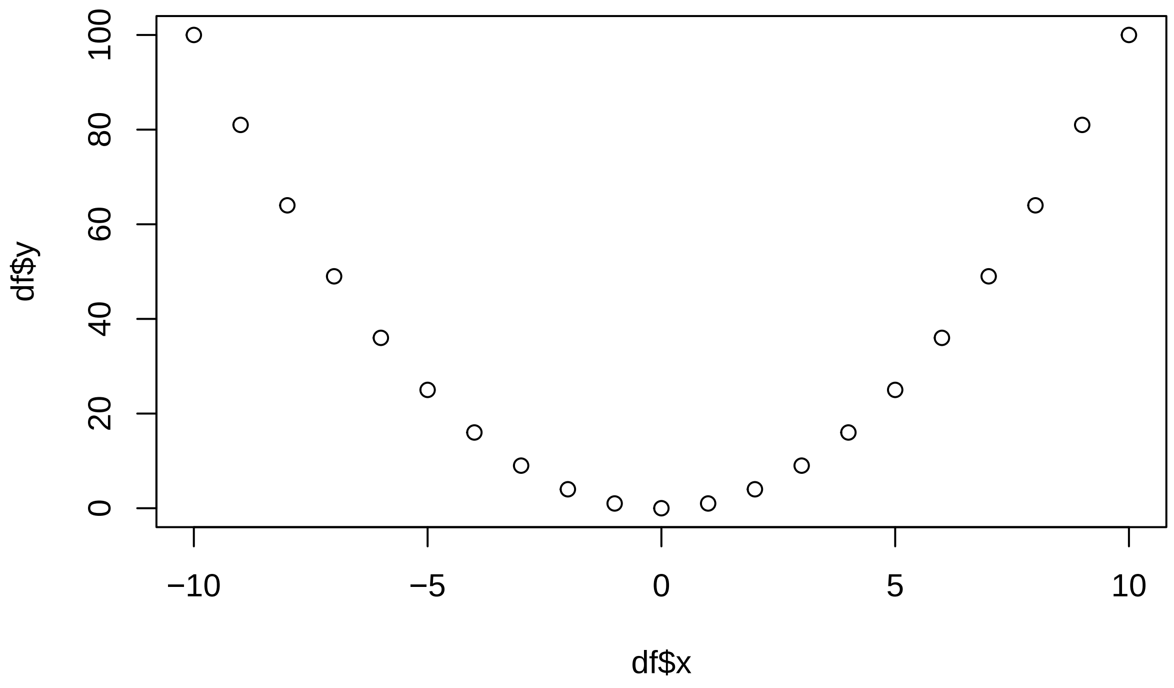


In RStudio, any visible output will appear in the console window and possibly below the code chunk depending on your settings. Also check the Environment panel (upper right) for objects that were created by executing the code chunk. The assignment operator, "<-", is used to create objects such as variables and dataframes. The visible output from code chunks will also appear in the knitted document.

## Comments vs Headers

Hashtags inside of code chunks function very differently than when used in Markdown space. Inside of code chunks, hashtags tell R to simply ignore the text that follows. These hashtags create comments within code chunks that are used to provide short explanatory messages as shown below:

```
# Hashtags inside of code chunks create comments that are not "executed".
# The code below creates a boring graph using base R
plot(x = df$x, y = df$y)
```



**PRACTICE** Use shortcut keys to execute individual commands in the code chunk below (hidden from knitted document) and avoid moving the cursor between executions. Pause after each execution to consider what each command accomplished.

## Commands and arguments

In general, commands perform actions on objects that are passed as arguments into those commands. The actions of commands can be modified by additional arguments, some of which are optional. Everything passed into the parentheses of a command is an argument, regardless of what that argument is or does. Arguments are separated by commas. The data command below has one argument, while the head command has two.

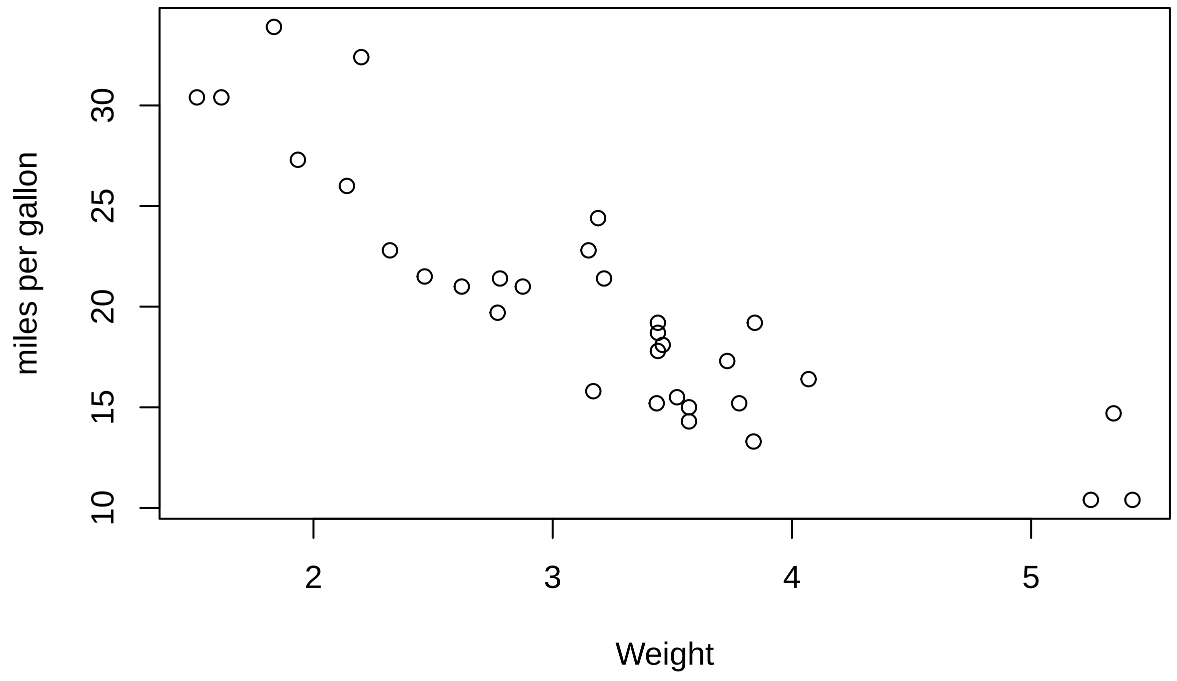
```
data(mtcars)
head(mtcars, 4)
```

```
##                 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
```

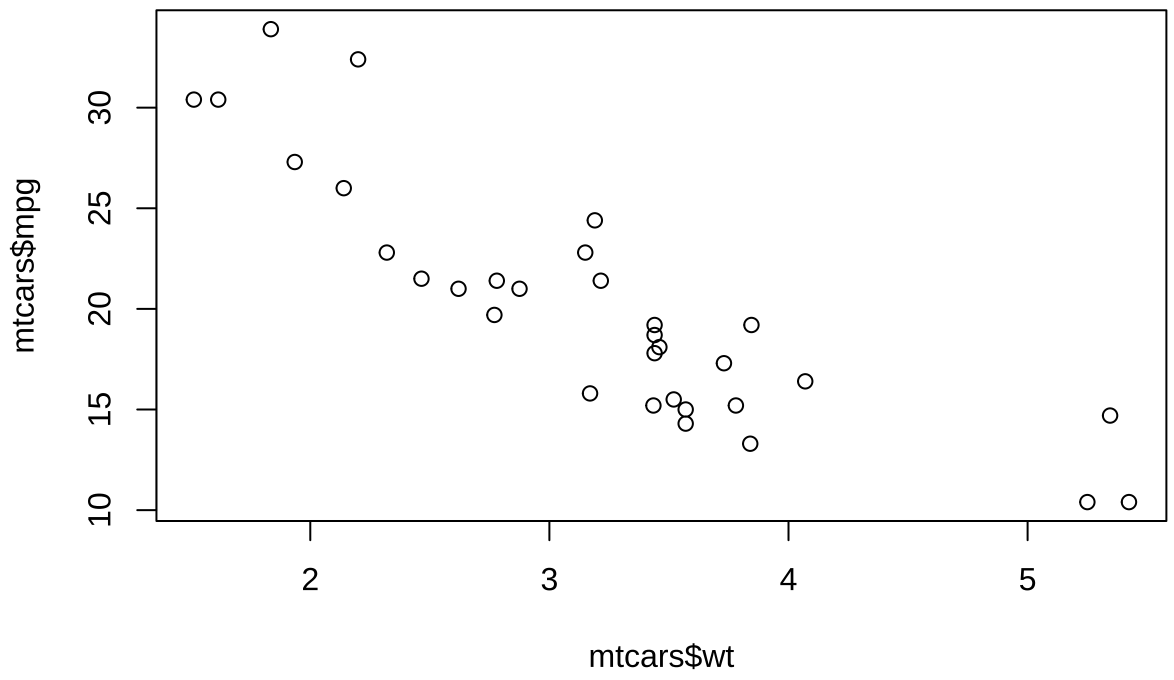
### Argument names

The plot command below contains 4 arguments that identify the data to be plotted on the x and y axis, and the optional arguments that customize the x and y axis labels. Arguments may have names (e.g. x =, y =, xlab = and ylab = below), but these names can be avoided if the arguments are presented in the exact order that r expects. However, I recommend using argument names, especially at first, to make your code more readable and intuitive.

```
plot(x = mtcars$wt, y = mtcars$mpg,
     xlab = "Weight", ylab = "miles per gallon")
```



```
# Avoiding argument names and optional arguments.
plot(mtcars$wt, mtcars$mpg)
```



## Piping commands with "%>%"

The Tidyverse pipe operator, %>%, allows code to follow a natural reading pattern, which is left to right and top to bottom. The %>% operator takes the output of code to the left of the operator and passes that into the first argument of a command on the right. This is called piping and is a feature of many programming languages. Below is an example of the pipe operator in use. The line of code would read something like "start with mtcars, select the column called hp, lastly summarize the values".

```
mtcars %>% select(hp) %>% summary
```

```
##        hp
##  Min.   : 52.0
##  1st Qu.: 96.5
##  Median :123.0
##  Mean   :146.7
##  3rd Qu.:180.0
##  Max.   :335.0
```

```
# or equivalently with white space added

mtcars %>%
    select(hp) %>%
    summary
```

```
##        hp
##  Min.   : 52.0
##  1st Qu.: 96.5
##  Median :123.0
##  Mean   :146.7
##  3rd Qu.:180.0
##  Max.   :335.0
```

## Nesting commands (not piping)

An equivalent line of code without piping is shown below. Notice that command execution begins with the inner most nested command and proceeds outwardly. This inside-out order makes complicated commands with multiple nested layers difficult to read and write.

```
summary(select(mtcars, hp))
```

```
##        hp
##  Min.   : 52.0
##  1st Qu.: 96.5
##  Median :123.0
##  Mean   :146.7
##  3rd Qu.:180.0
##  Max.   :335.0
```

## Videos, Tutorials, Cheatsheets & More Help

1) Installation Tutorials –> Install R, RStudio and tidyverse on your computer.
2) R Markdown Vimeo –> Short video on what R Markdown is and does.
3) R Markdown Tutorial –> A more detailed guide on R Markdown.
4) R language tutorials –> R programming tutorials hosted on RStudio Cloud. I recommend "The Basics", "Work with Data", "Visualize Data"
5) "Cheatsheets" –> 1 - 2 pages of densely packed introductory and higher level info. Don't try to absorb everything! They are helpful though.

6) Access help documentation on individual commands by typing a question mark before the command and then executing in the console window. Try this now with the command "version".