# Introduction to R and RStudio Exercises

In this tutorial, you will learn how to execute basic functions using R.

## Getting Started with R

### Installing Packages

An R package is a collection of functions, data, and code that has been stored within a well-defined format. These packages often allow us to perform a specific task without having to write our own R functions.

RStudio comes with the most commonly used packages pre-installed; however, it is still useful to know how to install R packages if you decide to use a different code editor. To install an R package, use **install.packages()** and input the name of the package you would like to install as a string within double quotation marks.

Let’s say the ggplot2 package (used for data visualization) was not already installed. To install, this package, we would input **install.packages(“ggplot2”)** and run the cell to install the package, shown below.

install.packages(“ggplot2”)

### Simple Arithmetic and Logical Operators

To get used to working with the R interface, let’s start with performing simple math operations. R can perform addition, subtraction, multiplication, and division using a single line of code.

The arithmetic operators for R are as follows:

| Operator | Description |
| --- | --- |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division |
| \*\* **or** ^ | Exponents |

Using the code box below, input **2+2** and run the cell.

2+2

## [1] 4

In addition, we can use R to find the square root of a number using a function. In R, functions are contained lines of code that execute a specific task. In the code box below, use the **sqrt()** function to find the square root of 16.

sqrt(16)

## [1] 4

In addition to the arithmetic operators, R supports logical operators. Inputting a logical statement will return either TRUE or FALSE, depending on if the statement is correct.

| Logical Operator | Description |
| --- | --- |
| > | greater than |
| >= | greater than or equal to |
| < | less than |
| <= | less than or equal to |
| == | exactly equal to |
| != | not equal to |

5>6

## [1] FALSE

### Getting Help

RStudio comes with a built-in help system with information about various topics, including packages, installation, and importing and exporting data. To quickly access the help window, use the function **help.start().**

help.start()

## starting httpd help server ... done

## If nothing happens, you should open  
## 'http://127.0.0.1:27879/doc/html/index.html' yourself

### Clearing the Console

Sometimes, the R console can get messy and confusing if you have multiple outputs. To clear the console, use the shortcut **ctrl+L.**

Great! Now that you’re getting used to working within the RStudio interface, let’s move on to working with some data structures.

## Working with Variables and Data Structures

### Creating Variables

To create a variable in R, type the name of the variable you wish to create followed by **<-** . This will allow us to store data for later use.

Let’s say that we were looking at a bowl of fruit and wanted to record the amount of each type of fruit in the bowl. Using variables, we can use the names of the fruit and assign each fruit a value. Running the cell will assign the value to each variable.

**Important:** Make sure you give your variables meaningful names. Avoid ambiguous names such as num, values, x, etc.

apples <- 5  
oranges <- 6  
grapes <- 15

After creating variables, you can perform operations using the variable names. In our fruit example, let’s say that we wanted to create a variable with the value of the total number of fruit. Using the arithmetic operators, we can add the values of the variables apples, oranges, and grapes and store them within the variable fruit. We can also use the **sum()** function to add the values of the variables together.

fruit <- sum(apples, oranges, grapes)

### Vectors

The most simple data structure utilized in R is the vector. Essentially, it is a sequence of data elements of the same type.

Using the **c()** function, we can create a vector and assign it to a variable.

vec1 <- c(1,2,3,4,5)

We can also use operators and sequences to create vectors, both of which are useful for working with sequential data.

To create a vector using an operator, use a colon (**:**) in between the beginning and end of the numbers that you would like to use.

vec2 <-c(1:5)

vec3 <- seq(1,5, by=0.5)  
vec4 <- seq(1,5, length.out=3)

### 

### Simple Data Frames

Using R, we can create a simple data frame to house a small dataset.

First, create a variable in which you would like to store the dataframe. Next, using the **data.frame()** function, create columns with the values you would like to include in your data.

In the cell below, create a dataframe with one column containing the numbers 1-10 and the second column containing the first ten letters of the alphabet.

**Important:** Double-check your brackets! **letters** and other functions use square brackets instead of parentheses. Refer to the help guide if you’re unsure of which brackets to use.

df1 <- data.frame(Numbers=c(1:10),  
 Alphabet=letters[1:10])  
  
df1

## Numbers Alphabet  
## 1 1 a  
## 2 2 b  
## 3 3 c  
## 4 4 d  
## 5 5 e  
## 6 6 f  
## 7 7 g  
## 8 8 h  
## 9 9 i  
## 10 10 j

### Loading Data with R

Often, you will already have data that you will want to load into R.

Let’s say that we have a .csv file with the amount of CO2 emissions per person every year, organized by country (Gapminder). Using a variable and the **read.table()** function, we can input the file path and store the data in R.

To view the first few rows of your data, use the **head()** function. This will help check if your data is loaded correctly.

**carbon <- read.table(“filepath.csv”, header=TRUE, sep=“,”)’**

**head()**

## Visualizing Data

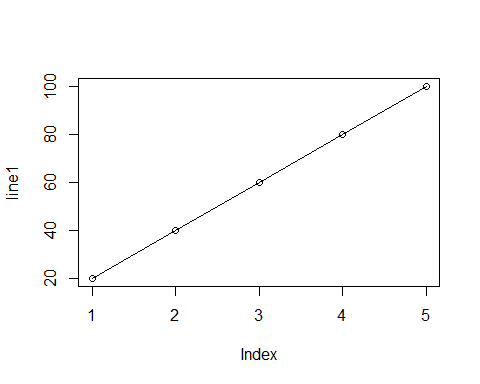
Finally, let’s learn some simple commands for engaging with data. While this can be a daunting and robust topic to cover, some basics will help you get started with working with R.

### Creating a Simple Line Graph

In R, we can visualize data in various ways.

Using a vector and the **plot()** function, we can plot data points within a line graph.

line1 <- c(20, 40, 60, 80, 100)  
  
plot(line1, type="o")

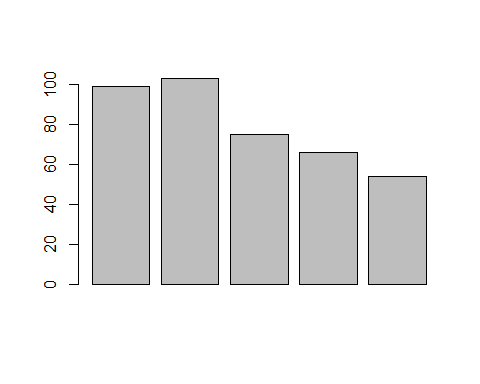


### Creating a Simple Bar Graph

Using the **barplot()** function, we can create a simple bar graph.

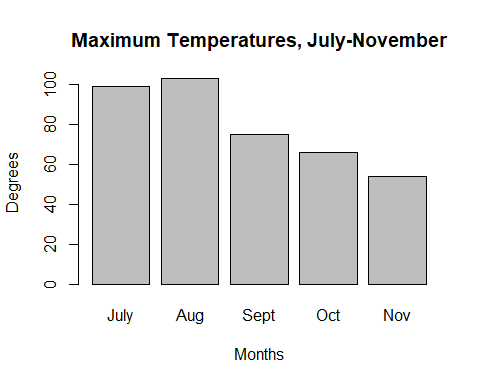
Let’s say we wanted to create a bar graph showing the maximum temperature over a monthly period. We can create a simple vector, store it in the variable **temps**, and use the barplot() function to create a bar graph.

temps <- c(99, 103, 75, 66, 54)  
  
barplot(temps)



In order to make the graph more meaningful, we can add a title and labels using arguments. Using **main,** we can add a title. Using **xlab** and **ylab**, we can add labels to the axises. Finally, **names.arg** allows us to utilize a vector in order to label the different bars in the graph.

barplot(temps,  
 main="Maximum Temperatures, July-November",   
 xlab= "Months",  
 ylab= "Degrees",  
 names.arg=c("July", "Aug", "Sept", "Oct", 'Nov'))



### 

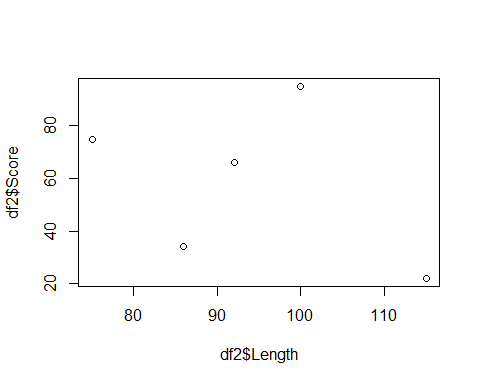
### Simple Scatter Plots

For this next visualization, we will use the dataframe below. Let’s say that we are interested in finding out if there is a correlation between a film’s length (minutes) and its Rotten Tomatoes score.

df2 <- data.frame(Length=c(75, 115, 100, 86, 92),  
 Score=c(75, 22, 95, 34, 66))

To create a scatter plot, we’re going to use the **plot()** function, adding two arguments: one for the x-axis and one for the y-axis. Using a **$**, we can use the columns from our dataframe within our scatter plot.

plot(df2$Length, df2$Score)



### Congrats, you’ve learned the basics of R!