

## Exercise 5

# Introduction to R

---

Part 1: R Overview . . . . .	68
Part 2: Installing CRAN . . . . .	69
Part 3: Installing RStudio IDE . . . . .	69
Part 4: Writing your first R script . . . . .	70
Part 5: R Language syntax . . . . .	71
Part 5.1: Comments . . . . .	71
Part 5.2: Variables . . . . .	71
Part 5.3: Data types . . . . .	72
Part 5.4: Strings . . . . .	72
Part 5.5: Operators . . . . .	73
Part 5.6: Math Library . . . . .	74
Part 5.7: Selection Structures . . . . .	75
Part 5.8: Loop Structures . . . . .	76
Part 5.9: Functions . . . . .	76
Part 5.10: Practice questions . . . . .	77
Part 6: Data structures in R . . . . .	77
Part 6.1: Vectors . . . . .	77
Part 6.2: Lists . . . . .	78
Part 6.3: Matrix . . . . .	78
Part 6.4: Arrays . . . . .	79
Part 6.5: Data Frames . . . . .	80
Part 6.6: Practice questions . . . . .	80
Part 7: Files in R . . . . .	81
Part 7.1: Sample files . . . . .	81
Part 7.2: Reading and writing files . . . . .	82
Part 7.3: Practice questions . . . . .	83
Part 8: Graphs in R . . . . .	84
Part 8.1: Plots . . . . .	84
Part 8.2: Scatter Plot . . . . .	85

---

Part 8.3: Pie Chart . . . . .	85
Part 8.4: Bar Chart . . . . .	86
Part 8.5: Histogram . . . . .	86
Part 8.6: Practice questions . . . . .	87
References . . . . .	<b>87</b>

---

## Estimated time

05:00 Hours

## Overview

It is assumed that you have basic programming skills and, you are familiar with high level programming languages (i.e. C++, Java). The exercise leverages on this familiarity in order to introduce you to R programming language.

In this exercise, you write simple R programs.

## Objectives

After completing this exercise, you should be:

- Install CRAN precompiled library distribution on a PC.
- Install RStudio IDE on a PC.
- Write and execute R scripts.

## Prerequisites

Basic understanding of high-level programming OR completed Exercise [1: Introduction to programming](#) or Exercise [2: Object-oriented programming](#).

## Requirements

This exercise requires a PC that has Internet access and, has a Web browser installed (i.e. Chrome).

## Exercise instructions

In this exercise, you complete the following tasks:

- Write and compile code excerpts as answers to exercise questions.
- Add a comment with the following information: **Name**, **Student Number**, and **Date**.
- Instructions for uploading your R scripts will be provided by the instructor.

## Part 1: R Overview

R is a system for statistical computation and graphics. It consists of a language plus a run-time environment with graphics, a debugger, access to certain system functions, and the ability to run programs stored in script files. R was initially created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team.

The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions. The R distribution contains functionality for a large number of statistical procedures. Among these are:

- linear and generalized linear models,
- nonlinear regression models,
- time series analysis,
- classical parametric and non-parametric tests,
- clustering and smoothing.

There is also a large set of functions which provide a flexible graphical environment for creating various kinds of data presentations/visualizations/graphs. Additional modules are available via “*add-on packages from CRAN*”.

R software has many uses, some popular ones include:

- computations on arrays, lists, vectors and matrices using a suite of operators;
- data analysis and generating graphs using a coherent and integrated collection of tools;
- writing statistical functions and machine learning models.

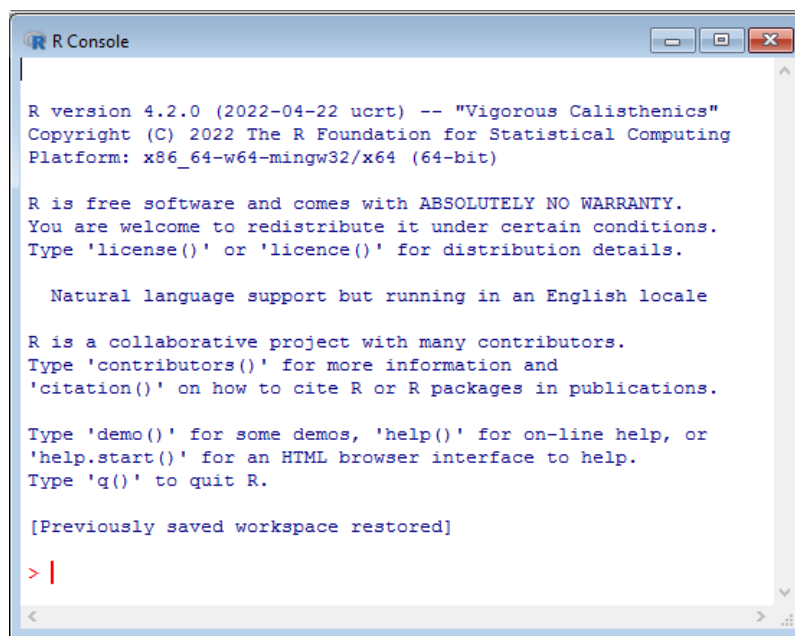
R has a well-developed, simple and effective programming language that is a great resource for data analysis, data visualization, data science and machine learning. It is easy to draw graphs in R, like pie charts, histograms, box plot, scatter plot, etc. R is open-source and it works on different platforms (Windows, Mac, Linux) and it provides many free libraries.

## Part 2: Installing CRAN

Before you begin writing R scripts, you need R software on your computer. The “*Comprehensive R Archive Network*” (CRAN) is R’s central software repository and it is supported by the R Foundation. It contains an archive (distributed on a collection of sites) consisting of the R distribution(s), the contributed extensions, documentation for R, and binaries.

CRAN includes source packages and pre-compiled binaries for Linux, Windows and MacOS. When you install CRAN, you install a pre-compiled R binary distribution (must be > v3.3 to work with RStudio) which comes with a console for using R interactively.

- Installing CRAN via Download, follow this link <https://cran.r-project.org/>.



```
R version 4.2.0 (2022-04-22 ucrt) -- "Vigorous Calisthenics"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

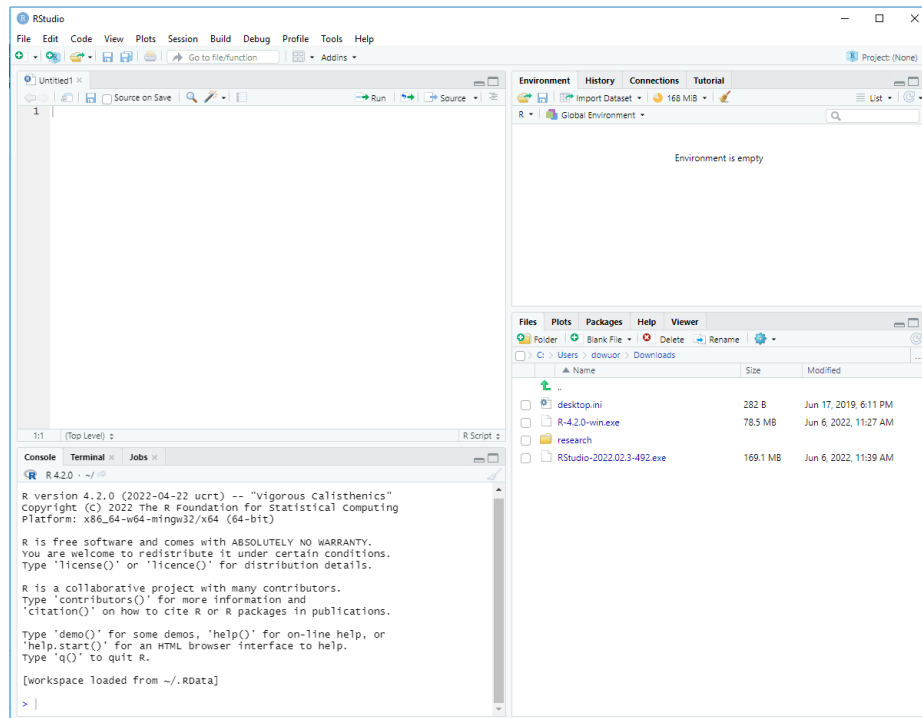
[Previously saved workspace restored]

> |
```

## Part 3: Installing RStudio IDE

Again before getting started, you may need an IDE or a text editor that has been tailored to make editing, interpreting and executing R scripts. This course recommends RStudio IDE for these tasks since it comes with a set of integrated tools designed to help you be more productive with R and Python. It includes a console, syntax-highlighting editor that supports direct code execution, and a variety of robust tools for plotting, viewing history, debugging and managing your work-space.

- Installing RStudio via Download, follow this link <https://www.rstudio.com/products/rstudio/download/>.



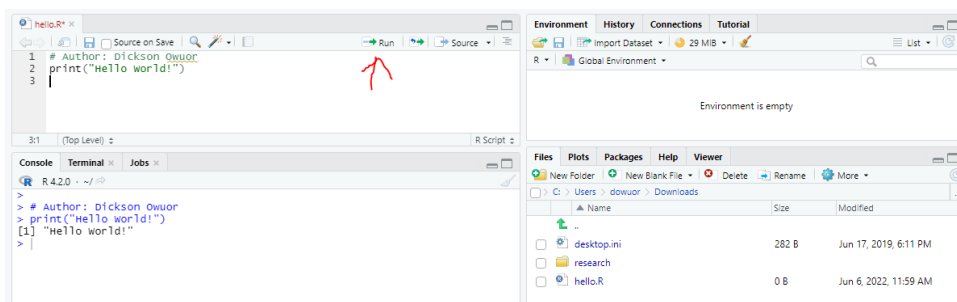
## Part 4: Writing your first R script

In this exercise, you write and execute your first R script. Follow the steps that follow:

1. Launch RStudio, create a 'New R Script' and you may save it as *'hi.R'*.
2. Write the code shown below.

```
# Author: Dickson Owuor  
print("Hello World!")
```

3. To interpret and execute your code, click on the 'run' button shown by the red arrow in the figure below. The results should be displayed in the Console window.



4. The same code can be written and executed interactively (line by line) using the R Console application.

## Part 5: R Language syntax

R language has a simple syntax and grammar. Below are the guidelines to adhere to when writing Python code:

- Designed for readability, and has some similarities to the English language with influence from mathematics.
- Case sensitive (i.e., *Print* is different from *print*)
- When executing an R script in RStudio, execution occurs from the line which is highlighted by the cursor.
- The `help()` function and `? help` operator in R provide access to the documentation pages for R functions, data sets, and other objects. (i.e., `help(solve)` or `?solve`).

### Part 5.1: Comments

`#` symbol is used to display comments in R. It is used for single-line and multi-line comments.

```
# Author: Dickson Owuor  
# Created: 06-06-2022
```

### Part 5.2: Variables

Variables are containers for storing data values. There is no precise command for declaring variables. We use the `<-` sign to assign a value to a variable.

```
# Author: Dickson Owuor  
# Created: 06-06-2022  
  
name <- "Jackline"  
yob <- 1992  
age = 2022 - yob  
print(name)  
print(paste("Age: ", age))
```

**Note:** If you try to combine a string (text) and a number, R will return an error.

### Part 5.3: Data types

In programming language, data types are used to determine the kind of data that a variable stores. In R language, variables types are not explicitly declared and can change after been set. Depending on the value stored by a variable R is able to assign the correct data type during execution. The basic data types can be divided into:

- Numeric
- Integer
- Complex
- Character
- Logical

```
# Author: Dickson Owuor  
# Created: 06-06-2022
```

```
# numeric type  
var <- 30  
class(var)
```

```
# integer type  
var <- 100L  
class(var)
```

```
# complex type  
var <- 2 + 40i  
class(var)
```

```
# character/string type  
var <- "R is exciting"  
class(var)
```

```
# logical/boolean  
var <- TRUE  
class(var)
```

**Note:** `Class()` function is used to check the data type of a variable.

### Part 5.4: Strings

Strings are special types of variables. A string is surrounded by either single or double quotation marks. String functions include: `nchar(x)`, `cat(x)`, `grepl()`, `toupper(x)`, `tolower(x)`, `substring()`, `format()` etc.



```
# Author: Dickson Owuor
# Created: 06-06-2022

# nchar to find character number
str <- "Welcome"
print(nchar(str))

# cat() to add a line break
str <- "Data science is an \"exciting\" discipline that allows you to turn raw
      data into knowledge."
str
print(cat(str))

# grepl() to check if a character appears in a string
str <- "Welcome John"
print(grepl("Jo", str))

# toupper(), tolower()
str <- "Good Job"
print(toupper(str))
print(tolower(str))

# substring(x, first, last)
str <- substring("Awesome", 1,3)
print(str)

# format(x, digits, nsmall, width,...)
str <- format("Awesome", width=20, justify="c")
print(str)
```

## Part 5.5: Operators

Operators are used to perform computations on variables and values. R divides the operators in the following groups:

- Arithmetic operators (+, -, \*, /, %%)
- Assignment operators (<-, «-, ->, -»)
- Relational operators (==, !=, >, <, >=, <=)
- Logical operators (&, &&, |, ||, !)
- Miscellaneous operators (:)

```
# Author: Dickson Owuor
# Created: 06-06-2022

# Arithmetic operators
2 + 25
25 - 25
3 %% 2 # Modulus (Remainder from division)
3 %/% 2 # Integer Division

# Assignment operator
x <- 20
x

# Comparison operators
22 == 25
21 != 25
1 >= 25

# Logical operators
!TRUE
TRUE | FALSE # Element-wise logical OR
TRUE || FALSE # Logical OR

# Miscellaneous operators
x <- 2:20
x
```

## Part 5.6: Math Library

R has many built-in math functions that allows you to perform mathematical operations. Basic maths functions are: `max()`, `min()`, `sqrt()`, `trunc()`, `ceiling()`, `floor()`, `log(x)` etc.

```
# Author: Dickson Owuor
# Created: 06-06-2022

x <- c(30, 20, 44)
# maximum and minimum number
print(max(x))
print(min(x))

# Square root of a number
x <- 25
print(sqrt(x))

# Truncate value of an input
x <- 2.5
print(trunc(x))

# Floor and ceiling values
# The ceiling() rounds a number upwards to its nearest integer, floor() rounds a
# number downwards to its nearest integer.
```

```
x <- 30.4
print(floor(x))
print(ceiling(x))

# logical/boolean
x <- 4
print(log(x))
```

**Note:** `c()` function is used to combine the elements into a vector.

## Part 5.7: Selection Structures

An “if statement” is written with the if keyword, and it is used to specify a block of code to be executed if a condition is TRUE.

```
# Author: Dickson Owuor
# Created: 06-06-2022

# If statement
x <- 20
y <- 10
if(x > y){
  print("x is greater than y")
}

# If else statement
if(x > y){
  print("x is greater than y")
} else {
  print("x is not greater than y")
}

# Else if statement
if (x > y) {
  print("x is greater than y")
} else if (x == y) {
  print("x and y are equal")
} else{
  print("x is not greater than y")
}
```

## Part 5.8: Loop Structures

Loop structures execute a set of statements as long as a condition is TRUE.

```
# Author: Dickson Owuor
# Created: 06-06-2022

# An example of a For Loop
for (x in 1:10) {
  print(x)
}

# Another example using a For Loop with a vector
dice <- c(1, 2, 3, 4, 5, 6)
for (x in dice) {
  print(x)
}

# An example of a While Loop
i <- 1
while (i < 6) {
  print(i)
  i <- i + 1
}
```

## Part 5.9: Functions

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. To call a function, use the function name followed by parenthesis, like `my_function()`, arguments are specified after the function name, inside the parentheses.

```
# Author: Dickson Owuor
# Created: 06-06-2022

# function() keyword
my_function <- function() {
  print("Hello World!")
}
my_function() # call the function named my_function

# Arguments passed inside a function
my_function <- function(fname) {
  paste(fname, "Steven")
}
my_function("Dickson")
my_function("Jackline")
```

## Part 5.10: Practice questions

Attempt all the questions that follow:

1. Write a program that displays all the *odd numbers* between 0 and 100. The program should also display the sum of the odd numbers.
2. Write a program that will calculate and display the *Factorial* of any number the user enters.
3. Write a program that will display the *Fibonacci sequence* of any number a user enters.

## Part 6: Data structures in R

R operates on named data structures. Examples of data structures include: vectors, lists, matrix, array, data-frames.

### Part 6.1: Vectors

A vector is simply a list of items that are of the same type. The `c()` function is used to combine the list of items (separated by commas) to a vector. To find out how many items a vector has, we use the `length()` function. To sort items in a vector alphabetically or numerically, we use the `sort()` function.

```
# Author: Dickson Owuor
# Created: 06-06-2022

# Vector of strings
animals <- c("Lion", "zebra", "tiger")
animals

# Vectors of numerical values
vals <- c(1, 2, 3, 4, 5)
vals

# OR use the : (colon) symbol
vals <- 1:5
vals

# Vector functions: length and sort
sort(vals) # Sort numbers
sort(animals) # Sort strings
length(vals)

# Access vector elements
animals[1]
animals[c(1,2)] # Access multiple elements
```

```
# Repeat vector uses the rep() function
repeat_each <- rep(c(1,2,3), each = 3)
repeat_each
```

## Part 6.2: Lists

A list in R can contain many different data types inside it. A list is a collection of data which is ordered and changeable. To create a list, we use the `list()` function. To add an item to the end of the list, we use the `append()` function. To find out how many items a list has, we use the `length()` function

```
# Author: Dickson Owuor
# Created: 06-06-2022

# list of strings
animals <- list("lion", "zebra", "tiger")
animals

# list of numerical values
vals <- list(1, 2, 3, 4, 5)
vals

# length()
length(animals)

# append
append(animals, "cobra")
append(animals, "bear", after = 2)

# check if item exists
"zebra" %in% animals

# Remove from list items
newlist <- vals[-1]
newlist
```

## Part 6.3: Matrix

A matrix can be created with the `matrix()` function. Specify the `nrow` and `ncol` parameters to get the count of rows and columns. We use the `cbind()` function to add additional columns in a Matrix. We use the `rbind()` function to add additional rows in a Matrix:

```
# Author: Dickson Owuor
# Created: 06-06-2022

# Create a matrix with numbers
thismatrix <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)
thismatrix # Print the matrix
```

```
# Create a matrix with strings
thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)
thismatrix

# Access Matrix Items
thismatrix[1, 2] # row 1, column 2 elements
thismatrix[2,] # all row 2 elements
thismatrix[,2] # all column 2 elements

# cbind function
newmatrix <- cbind(thismatrix, c("strawberry", "blueberry", "raspberry"))
newmatrix

newmatrix <- rbind(thismatrix, c("avocado", "mango"))
newmatrix# Print the new matrix

# dim() function to find column and row count
dim(thismatrix)
```

## Part 6.4: Arrays

Compared to matrices, arrays can have more than two dimensions. We use the `array()` function to create an array, and the `dim` parameter to specify the dimensions.

```
# Author: Dickson Owuor
# Created: 06-06-2022

# An array with 1 dimension
thisarray <- c(1:24)
thisarray

# An array with 3 dimension
multiarray <- array(thisarray, dim = c(4, 3, 2))
multiarray

#Access array items
multiarray[2, 3, 2] #[row pos, col pos, matrix level]

# Access whole row or columns using c()
# Access all the items from the first row from matrix one
multiarray[c(1),,1]

# Access all the items from the first column from matrix one
multiarray[,c(1),1]
```

## Part 6.5: Data Frames

Data Frames are data displayed in a format as a table. We use the `data.frame()` function to create a data frame. We use the `summary()` function to summarize the data from a Data Frame. We use the `rbind()` and `cbind()` functions to add new rows or columns in a Data Frame.

```
# Author: Dickson Owuor
# Created: 06-06-2022

# Create a data frame
df <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45))
df # Print the data frame

# Summarise the Data
summary(df) # Summarise the data

# Add a new row
New_row_DF <- rbind(df, c("Strength", 110, 110))
New_row_DF
```

## Part 6.6: Practice questions

Attempt all the questions that follow:

1. Write a program that creates a matrix of size  $100 \times 100$ , initialize the values one by one using a loop to any value. Display the matrix.
2. Write a program that will randomly generate 8 values. Store the values in an integer array. Calculate and output the sum, product and average of the values.



## Part 7: Files in R

In R, we can read data from files stored outside location. We can also write data into files which will be stored and accessed by the operating system. R can read and write into various file formats like `csv`, `excel`, `xml` etc.

### Part 7.1: Sample files

Create these files using a notepad and excel by copying and pasting this data. Save the file indicated.

```
# file.csv and file.xlsx
id,name,salary,start_date,dept
1,Rick,623.3,2012-01-01,IT
2,Dan,515.2,2013-09-23,Operations
3,Michelle,611,2014-11-15,IT
4,Ryan,729,2014-05-11,HR
5,Gary,843.25,2015-03-27,Finance
6,Nina,578,2013-05-21,IT
7,Simon,632.8,2013-07-30,Operations
8,Guru,722.5,2014-06-17,Finance

# input.json
[
  {
    "album": "The White Stripes",
    "year": 1999,
    "US_peak_chart_post": 65
  },
  {
    "album": "De Stijl",
    "year": 2000,
    "US_peak_chart_post": 78
  },
  {
    "album": "White Blood Cells",
    "year": 2001,
    "US_peak_chart_post": 61
  },
  {
    "album": "Elephant",
    "year": 2003,
    "US_peak_chart_post": 6
  },
  {
    "album": "Get Behind Me Satan",
    "year": 2005,
    "US_peak_chart_post": 3
  },
  {
```

```

    "album": "Icky Thump",
    "year": 2007,
    "US_peak_chart_post": 2
  },
  {
    "album": "Under Great White Northern Lights",
    "year": 2010,
    "US_peak_chart_post": 11
  },
  {
    "album": "Live in Mississippi",
    "year": 2011,
    "US_peak_chart_post": 54
  },
  {
    "album": "Live at the Gold Dollar",
    "year": 2012,
    "US_peak_chart_post": 100
  },
  {
    "album": "Nine Miles from the White City",
    "year": 2013,
    "US_peak_chart_post": 32
  }
]

```

## Part 7.2: Reading and writing files

```

# Author: Dickson Owuor
# Created: 06-06-2022

# Read a CSV file
data <- read.csv("file.csv")
print(data)

# Analysing the CSV File
print(is.data.frame(data))
print(ncol(data))
print(nrow(data))

# Get the maximum/minimum salary
maxsal <- max(data$salary)
minsal <- min(data$salary)
print(maxsal)
print(minsal)

# Write filtered data into a new CSV file
retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))
write.csv(retval, "output.csv", row.names = FALSE)
newdata <- read.csv("output.csv")
print(newdata)

```

```
# Install the xlsx package
install.packages("xlsx")

# Verify and load the xlsx package
any(grep1("xlsx", installed.packages())) # verify that the package is installed
correctly

# Load the library
library("xlsx")

# Read the Excel File
data <- read.xlsx("file.xlsx", sheetIndex = 1)
print(data)

# Install the rjson package
install.packages("rjson")

# Load the library
library("rjson")

# Give the input file name to the function
result <- fromJSON(file = "input.json")
print(result) # Print the result

# Convert JSON to a data frame and print
json_df <- as.data.frame(result)
print(json_df)
```

### Part 7.3: Practice questions

Attempt all the questions that follow:

1. Write a program that will read a matrix from a CSV file. Store the values in a Data Frame. Calculate and output the sum, product and average of the values.

## Part 8: Graphs in R

R programming language has numerous libraries to create charts and graphs. In the exercise that follow, we learn how to draw basic charts and graphs using R language.

### Part 8.1: Plots

The `plot()` function can be used to:

- draw points (markers) in a diagram, where parameter 1 specifies points on the x-axis and y-axis.
- draw a line to connect all the points in the diagram.

```
# Author: Dickson Owuor
# Created: 06-06-2022

# Documentation and Help
??plot

# Creating a plot
# Draw one point in the diagram, at position (1) and position (3):
plot(1, 3)

# Draw two points, one at position (1, 3) and another at (8, 10):
plot(c(1, 8), c(3, 10))

# Multiple points
plot(c(1, 2, 3, 4, 5), c(3, 7, 8, 9, 12))

# OR

x <- c(1, 2, 3, 4, 5)
y <- c(3, 7, 8, 9, 12)
plot(x, y)

# Sequence of points
plot(1:10)

# Draw a Line Graph
plot(1:10, type="l")

# Plot with Labels
plot(1:10, main="My Graph", xlab="The x-axis", ylab="The y axis")

# Add Colors
plot(1:10, col="red")
```

## Part 8.2: Scatter Plot

A “scatter plot” is plot that displays the relationship between two numerical variables, and plots one dot for each observation. It needs two vectors of same length, one for the x-axis (horizontal) and one for the y-axis (vertical):

```
# Author: Dickson Owuor
# Created: 06-06-2022

# Scatter Plot
x <- c(5,7,8,7,2,2,9,4,11,12,9,6)
y <- c(99,86,87,88,111,103,87,94,78,77,85,86)

plot(x, y)

# With an observation
plot(x, y, main="Observation of Cars", xlab="Car age", ylab="Car speed")

# Compare plots use the points() function
# Day one, the age and speed of 12 cars:
x1 <- x
y1 <- y

# Day two, the age and speed of 15 cars:
x2 <- c(2,2,8,1,15,8,12,9,7,3,11,4,7,14,12)
y2 <- c(100,105,84,105,90,99,90,95,94,100,79,112,91,80,85)

plot(x1, y1, main="Observation of Cars", xlab="Car age", ylab="Car speed",
     col="red", cex=2)
points(x2, y2, col="blue", cex=2)
```

## Part 8.3: Pie Chart

A pie chart provides a circular graphical view of data. We use the `pie()` function to draw pie charts.

```
# Author: Dickson Owuor
# Created: 06-06-2022

# Help
??pie

# Draw pie charts
x <- c(10,20,30,40) # a vector
pie(x)

# Start Angle, use init.angle parameter, default is zero.
pie(x, init.angle = 90) # start the first pie at 90 degrees

# Display the pie chart with labels
```

```
mylabel <- c("Apples", "Bananas", "Cherries", "Dates")
pie(x, label = mylabel, main = "Fruits")

# Display the pie chart with colors
colors <- c("blue", "yellow", "green", "black")
pie(x, label = mylabel, main = "Fruits", col = colors)
```

## Part 8.4: Bar Chart

A bar chart uses rectangular bars to visualize data. Bar charts can be displayed horizontally or vertically. The height or length of the bars are proportional to the values they represent. We use the `barplot()` function to draw a vertical bar chart.

```
# Author: Dickson Owuor
# Created: 06-06-2022

# Help
??barplot

# Draw bar charts
x <- c("A", "B", "C", "D") # x-axis values
y <- c(2, 4, 6, 8) # y-axis values
barplot(y, names.arg = x)

# The col parameter is used to change colors
barplot(y, names.arg = x, col = "red")

# Density parameter is used to change the bar texture
barplot(y, names.arg = x, density = 10)

# Width parameter is used to change the width of the bars
barplot(y, names.arg = x, width = c(1,2,3,4))

# To display Horizontal bars vertically, use horiz = TRUE
barplot(y, names.arg = x, horiz = TRUE)
```

## Part 8.5: Histogram

Histogram is similar to bar chat but the difference is that it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range. We use the `hist()` function to plot histograms.

```
# Author: Dickson Owuor
# Created: 06-06-2022

# Help
??hist
```

```
# Create a histogram
v <- c(9,13,21,8,36,22,12,41,31,33,19)
hist(v, xlab = "Weight", col = "yellow", border = "blue", main="Histogram of v")

# Range of X and Y values
hist(v, xlab = "Weight", col = "green", border = "red", xlim = c(0,40), ylim =
      c(0,5), breaks = 5, main="Histogram of v")
```

## Part 8.6: Practice questions

Attempt all the questions that follow:

1. Using the data-set shown below, draw a stacked bar-plot with a legend.

```
# Author: Dickson Owuor
# Created: 06-06-2022

#           A   B   C   D   E
# Group 1  0.2 0.3 0.7 0.1 0.3
# Group 2  0.4 0.1 0.1 0.2 0.3
```

## References

- [ 1 ] <https://cran.r-project.org/>
- [ 2 ] [https://en.wikipedia.org/wiki/R\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/R_(programming_language))
- [ 3 ] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2022. URL: <https://www.R-project.org>

## End of exercise