Module I

Introduction

* R is an interpreted computer programming language. [There are two types of programming languages - Interpreted and compiled. In interpreted language the code is executed line by line & is compiled the code is compiled to ~~executable and then~~ executable, is whole and then gets executed).

* R is used to analyse statical information, graphical representations, reporting and data modeling.
  - If we want get the statical information such as mean, median, mode of the data, it can be implemented easily in R.
  - To do the graphical representation of data or to visualize the data, there are a lot of packages available in R which are very user friendly.
  - We use R program in Data analytics, Data Science
  - To implement the machine learning algorithms, R is a very useful programming language

History of R language.

- R was developed by Ross Ihaka & Robert Gentleman in the University of Auckland. New Zealand.
- The programming language's name is taken from the name of both the developers.
- The language 1st project was introduced in 1992.
- The initial version of R was released in 1995. and in 2000, a stable beta version was released.
- Latest version of R version 4.3.2 has been released on ~~2023-10-31~~. 2023-10-31.

- And you'll be prompted for your password, and if you have sudo privileges, R should be installed on your sms.
- To update the versions use
  $ sudo yum update R.x86_64

Installing R from downloaded files.

- Currently there are pre-compiled R packages for several flavors of Linux
- CRAN Red Hat-style sms.
- Once you download the file to the directory ~/Downloads, you could install it with a command like
  $ rpm -i ~/Downloads/R-2.15.1-fc10.i386.rpms.

The R Graphical User Interface

→ When you open the R application on Windows or Mac OS X, you'll see a command window and some menu bars.

→ On most Linux sms, R will simply start on the command line.

Windows

* By default, R is installed into %Program Files%R and installed into the Start menu under the group R.
* Inside the R GUI window, there is a menu bar, a toolbar, and the R console.

Mac OS X

* The default R installer will add an application called R to your applications folder that you can run like any other applications on your Mac.
* The Application has a menu bar, ~~a tool~~ a toolbar with common functions, and an R Console window.

Linux and Unix

* on Linux sms, you can start R from the command line by typing:
  $ R
  - notice that its a capital "R", filenames on Linux are case sensitive.
  - $ symbol is just a Unix prompt. No need to type it.

R Installation

→ Visit the official R website (http://www.r-project.org/).
→ Go to the download link
→ Find the right binary for your platform & run the installer.

Windows.

→ According to file permissions there are two approaches to install R in Windows.

1. Install R as a standard user in your own file space. This is the simplest approach.
2. Install R as the default administrator account (if it is enabled and you have access to it).

Mac OSX.

→ The current version of R runs on both PowerPC and Intel-based Mac systems running Mac OSX 10.5 (Leopard) & higher.
→ Older versions are also available on the website if the OS is having older version.

→ There are three ~~ways to install R~~ on different R installers for Mac OS X:
- A three-way universal binary for Mac OS X 10.5 (Leopard) and higher
- A legacy universal binary for Mac OS X 10.4 and higher without supplement tools.
- A legacy universal binary for Mac OS X 10.4 and higher without supplement tools.

Linux and Unix Systems

~~Rmac~~

Installation using package management systems.

- It is the easiest way to install R in Linux or Unix.
- open a terminal window and type;

```
$ sudo yum install R.x86_64.
```

- And you'll be prompted to your password, and if your
  Sudo privileges it should be installed on your Mac.
- To update the version use
  $ sudo port selfupdate

Installing R from downloaded files:

- Currently there are pre-compiled R packages for several flavours
  Linux
- When you download the file to the directory
  you could install it with a command like
  $ open -i ~/Downloads/R-2.15.1.pkg (tab)

The R Graphical User Interface

- When you open the R application on Windows or Mac OS X
  you'll see a command window and some menus here.
- On most Linux sites, R will simply start on the command line.

Windows

- By default, R is installed in C:\Program Files\R and installs into the
  Start menu under the group R.
- Inside the R GUI window, there is a menu bar, a toolbar
  and the R Console.

Mac OS X

- The default R installer will put an application called R in
  your applications folder. And you can run like any other
  application on your Mac.
- The Application has a menu bar, a toolbar with common
  functions, and an R Console window.

Linux and Unix

- On Linux first you can start R from the command line by
  typing:
  $R
  - note that it is a capital "R". Humans on Linux are case
    sensitive.
  - $ symbol is just a shell prompt. Do not need to type it.

Module I

Introduction

* R is an interpreted computer programming language. [There are two types of programming languages - Interpreted and compiled. In interpreted language the code is executed line by line & in compiled the code is compiled ~~for entire code and then~~ as whole and then gets executed).

* R is used to analyze statical information, graphical representations, reporting and data modeling.
  - If we want get the statical information such as mean, median, mode of the data, it can be implemented easily in R.
  - To do the graphical representation of data or to visualize the data, there are a lot of packages available in R which are very user friendly.
  - We use R programs in Data analytics, Data Science
  - To implement the machine learning algorithms, R is a very useful programming language

History of R language.

- R was developed by Ross Ihaka & Robert Gentleman in the University of Auckland. New Zealand.
- The Programming language is name is taken from the name of both the developers.
- The language 1st project was introduced in 1992.
- The initial version of R was released in 1995. and in 2000, a stable beta version was released.
- Latest version of R version 4.3.2 has been released on ~~2023-10-31~~. 2023-10-31.

Features of R Programming Language

- R is a comprehensive programming language that provides support for procedural programming involving functions as well as object-oriented programming.
- R can be easily easily integrated with many other technologies and frameworks like Hadoop & HDFS. It can also integrate with other programming languages like C, C++, Python & Java, FORTRAN, & JavaScript.
- It is an open source language. It is completely free for anybody to use.
- There are more than 15000 packages for R on online repositories like CRAN, Bioconductor, & GitHub.
- Powerful visualisation & graphics. It can produce publication-quality graphs & plots of any kind with its base packages. With added package like ggplot2 & Plotly the possibilities are endless.
- No need for a compiler. The R language is an interpreted language. It does not need a compiler to convert the code into a program.
- Cross-platform support - R is cross-platform supportive that is it can run on any OS & on any software environment without any hastle.
- Performs Fast Calculations. Through R, you can perform a wide variety of complex operations of vectors, arrays, data frames and other data objects of varying sizes. Furthermore, all these operations operate at a Lightning Lightning speed.

R Installation

→ Visit the official R website (http://www.r-project.org/).
→ Go to the download link
→ Find the right binary for your platform & run the installer.

Windows.

→ According to file permissions there are two approaches to install R in Windows.

1. Install R as a standard user in your own file space. This is the simplest approach.
2. Install R as the default administrator account (if it is enabled and you have access to it).

Mac OS X.

→ The current version of R runs on both PowerPC and Intel-based Mac systems running Mac OS X 10.5 (Leopard) & higher.
→ Older versions are also available on the website if the user is having older version.
→ There are three ~~ways to install R~~ different R installers for Mac OS X:
- A three-way universal binary for Mac OS X 10.5 (Leopard) and higher
- A legacy universal binary for Mac OS X 10.4 and higher without supplement tools.
- A legacy universal binary for Mac OS X 10.4 and higher without supplement tools.

Linux and Unix Systems

Installation using package management systems.
- It is the easiest way to install R in Linux or Unix.
- open a terminal window and type;

```
$ sudo yum install R.x86_64.
```

- And you'll be prompted for your password, and if you have sudo privileges, R should be installed on your sys.
- To update the versions use
  $ sudo yum update R.x86_64

Installing R from downloaded files.

- Currently there are pre-compiled R packages for several flavors of Linux
- Red Hat-style sys.
- Once you download the file to the directory ~/Downloads you could install it with a command like
  $ rpm -i ~/Downloads/R-2.15.1-fc10.i386.rpms.

The R Graphical User Interface

→ When you open the R application on Windows or Mac OS X, you'll see a command window and some menu bars.

→ On most Linux sys, R will simply start on the command line.

Windows

* By default, R is installed into %Program Files%R and installed into the Start menu under the group R.
* Inside the R GUI window, there is a menu bar, a toolbar, and the R console.

Mac OS X

* The default R installer will add an application called R to your applications folder that you can run like any other application on your Mac.
* The Application has a menu bar, ~~a tool~~ a toolbar with common functions, and an R console window.

Linux and Unix

* On Linux sys, you can start R from the command line by typing:
  $R
  - notice that its a capital "R", filenames on Linux are case sensitive.
  - $ symbol is just a Unix prompt. No need to type it.

* If you prefer a user interface similar to other platforms, you can use the command:

$ R -g Tk &

- This will launch R in the background running in its own window.
- Like other platforms, there is a menu bar with some common functions, but unlike the other platforms, there is no toolbar.
- The main window acts as the R console.

The R Console

* The R console is the most important tool for using R.
* The R console is a tool that allows you to type commands into R and see how the R sln responds.
* The commands that you type into the console are called expressions
* A part of the R sln called the interpreter will read the expression and respond with a result or an error message.
* When you launch R, you'll see a window with the R console.
  - Inside the console there are some messages or some basic informations about R such as the version of R you're running, some license information, quick reminders about how to get help, and a command prompt.
* By default, R will display a greater-than sign (">") in the console (at the beginning of a line, when nothing else is shown)
* R is prompting you to type something, so this is called a "prompt".

eg:- > 17+3
[1] 20

* ~~If you type "17+3" in the command prompt & press the enter key.~~
* Sometimes, an R command doesn't fit on a single line. If you enter an incomplete command on one line, the R prompt will change to a plus sign ("+").

eg:- > 1*2*3*4*5*
+ 6*7*8*9*10
[1] 3628800

* To avoid confusions, on most platforms, command prompts, user-entered text, and R responses are displayed in different colors to help clarify the differences.

Platform	Command Prompt	User Input	R output
Mac OS X	Purple	Blue	Black
Microsoft Windows	Red	Red	Blue
Linux	Black	Black	Black

Command Line Interface

* The command line interface is what makes R so powerful
* To run a command on R, type it into the console next to the ">" symbol and press the Enter key.
* The entries can be as simple as the number 2 or complex functions ~~such as th~~
* You can use up & down arrow keys to edit the previously executed commands
* You can also ~~use~~ type history() to get a list of previously typed commands.
* R also includes automatic completion for function names and filenames.
* To do so, use the tab key to see the possible completions for a function or a filename.

Batch Processing

* Since R is an interpretable programming language, it executes line by line.
* Once you type in a line of code and press enter, the line of code will get executed. But if we want to execute a block of code, then we need batch mode.

* R provides a way to run a larger set of commands in sequence and save the results to a file. This is called batch mode.
  - one way to run R in batch mode is from the OS command line (not the R console).
  - By running R from the OS command line, its possible to run a set of commands without starting R.
  - eg:- to load a set of commands from the file 'generate-graph.R' use command
    o/s R CMD BATCH generate-graph.R.

R Studio

* R Studio is the best IDE for R programming language for the time being.
* It is available for Windows, Mac & Linux and looks identical in all of them.
* To run one line, place the cursor at the desired line & press Ctrl + Enter.
* To insert a selection, simply highlight the selection & press Ctrl + Enter
* To run an entire file of code, press Ctrl + Shift + S
* When typing code, such as an object name or function name, hitting Tab will ~~auto-complete~~ autocomplete the code.
  - If there are more than one options, there will be a dialogue box pop up with all possible options
  - Typing Ctrl + 1 moves the cursor to the next editor area and Ctrl + 2 moves it to the Console.
  - To move to the previous tab in the text editor

Ctrl + Pageup — Linux

Ctrl + options + Left — on Mac

* To move to the next tab in the text editor,
  - → Press Ctrl + Alt + Right in Windows
  - → Ctrl + PageDown — Linux
  - → Ctrl + Option + Right - Mac

RStudio Projects

- A primary feature of RStudio is projects
- A project is a collection of files - a possibly data, results and graphs - that are all related to each other
- To start a new project
  click File >> New Project

* There are three options when creating a new project.
  - → Starting a new project in a new directory.
  - → associating a project with an existing directory
  - → checking out a project from a version control repository such as Git or SVN.

→ In all cases a .Rproj file is put into the resulting directory and keeps track of the project.

RStudio Tools

→ RStudio is highly customizable.

→ Most options are contained in the options dialog accessed by clicking Tools >> Global options

- Code editing options, control the way code is entered and displayed in the text editor.
- Code display options, control visual cues in the txt editor & Console.

- Highlighting selected words makes it easy to spot multiple occurences

→ Code saving options, control how the text files containing code are saved

→ Code completion options, control how code is completed while programming.

→ Code diagnostic options, enable code checking. These can be very helpful in identifying mistyped object names, poor style & general mistakes.

→ Appearance options, change the way code looks, esthetically. The font, size and color of the background and text can all be customized.

→ The Pane option, rearrange the panes that make up RStudio

→ The Package options, set options regarding packages.
- The most important is the CRAN mirror.

→ The RMarkdown option, control settings for working with RMarkdown documents.

→ Sweave option - Used to generate pdf documents with knitr or Sweave.

Git Integration

→ We can use Git to control the versions of the objects.

→ The main functionality is committing changes, pushing them to the server and pulling changes made by other users.
- Clicking the Commit button brings up a dialog which displays files that have been modified, or new files.
- Clicking on one of these files displays the changes, deletions are colored pink and additions are colored green.
- There is also a space to write a message describing th

→ clicking commit will stage the changes, and clicking Push will send them to the server.

* R Packages

→ One of the greatest advantages of R programming language is the no. of packages available on R.

→ There are more than 10,000 packages available on CRAN as of early 2017.

→ A package is essentially a library of pre written codes designed to accomplish some task or a collection of tasks.

→ The survival package is used for survival analysis.

→ ggplot2 is used for plotting.

→ SP is used for dealing with spatial data.

→ The R language was basically written by statisticians, so the quality of the packages may differ from what a computer engineer would expect.

Installing Packages

→ There are many ways to install R packages

- The simplest is to install them using the GUI provided by RStudio
- Access the packages pane either by clicking its tab or by press Ctrl+7 on the keyboard.
- In the upper left corner, click the Install packages button and click install.
- An alternative is to type a simple command into the console

```
> install.packages("coefplot")
```

- To install packages directly from GitHub or BitBucket

> library (devtools)

> install_github (repo = "ecetplot/jared_lander")

Uninstalling Packages

- To uninstall a package, the easiest way is to click the white [X] inside a gray circle on the right of the package descriptions in RStudio's Packages pane.

- Or use ~~remov~~ remove.packages, where the first argument is a character vector naming the packages to be removed.

Loading Packages

→ To load a package, we can use two commands.

1. library.
2. require.

→ using require will return TRUE if it succeeds and FALSE with a warning if it cannot find the package.

→ Calling library on a package that is not installed will cause an error.

> library (ecetplot)

→ To unload a package can be done either by clearing the checkbox in Rstudio's packages pane or by using the detach function.

> detach ("package : ecetplot")

* Basics of R

→ R is a powerful tool for all manner of calculations, data manipulation and scientific computations

⇒ Basic Math

→ Simply test R by running

```
> 1+1
[1] 2
```

eg:-
```
> 1+2+3
[1] 6
> 3*7*2
[1] 42
> 4/2
[1] 2
> 4/3
[1] 1.3333
> 4*6+5
[1] 29
> (4*6)+5
[1] 29
> 4*(6+5)
[1] 44.
```

⇒ Variables

→ Variables are an integral part of any programming language and R offers a great deal of flexibility.

→ Unlike statically typed languages like C++, R does not require variable types to be declared.

→ A single variable can at one point hold a number, then later on hold a character and then later a number again.

⇒ Variable Assignment

→ The valid assignment operators in R are

1. <— — most preferred
2. =

eg:- > x <— 2

> x

[2]

> y = 5

> y

[1] 5

→ The arrow operator can also point in the other direction

> 3 → z

> z

[1] 3

→ We can also use the assignment operator to successively assign a value to multiple variables simultaneously.

> a <— b <— 7

> a

[1] 7

> b

[1] 7

→ We can also use assign function

> assign ("j", 4)

> j

[1] 4

→ Variable names can contain any combination of alphanumeric characters along with periods (.) and underscores (_).

→ But they cannot start with a number or underscore.

→ To remove a variable, we can use the keywords remove or its shortcut rm

> j

[1] 4

> rm(j)

> # now it is gone

→ R automatically does the garbage collection periodically.

→ The variable names are case sensitive.

eg:- > theVariable ← 17
> theVariable
[1] 17
> THEVARIABLE
error.

⇒ Data Types.

→ The four main types of data are
1. Numeric
2. character (string)
3. Date / POSIXct (time-based)
4. logical (TRUE/FALSE)

→ We can check the data types of variables can be checked ~~by~~ with the function class'

→ ~~To set an integer~~ > class(x)
> typeof(x) also works.

1. Numeric

→ The most commonly used numeric data is numeric.
- It is similar to float or double in other languages.

→ It can handle both integer & decimal, both +ve & -ve values.

→ To check whether a variable is numeric, use is.numeric(x) function.

→ is.integer(x) - To check a variable is integer or not.

→ To set an integer to a variable, ~~post~~ appended 'L' after the value. Otherwise it will be stored as a double value

```
#integer
x ← 2L
typeof(x)
#double
y ← 2.5
```

→ Complex data also can be stored in R
z ← 3+3i typeof(z) [complex]

2. Character Data

→ We use character type to store string or text data.

eg:- < a ← "h"

<

→ To find the length of a character (or numeric) use the nchar function.

```
>x ← 'hai!'
eg:- > nchar (x)
[1] 4
 > nchar ("hallo")
[1] 5
 > nchar(1)
 [1] 1
 > nchar (A52)
 [1] 3.
```

3. Dates

→ There are two ways to stores Dates in R.

1. Using Date function - we can store just a date
2. Using POSIXct - we can store a date and time.

```
eg:- > date1 ← as.Date ("2012-06-28")
 > date1.
 [1] "2012-06-28"
 > class (date1)
 [1] "Date"
 > date2 ← as. POSIXct ("2012-06-28 (7:42")
 > *date2
[1] "2012-06-28 17:42:00 EDT"
 > class(date2)
[1] "POSIXct" "POSIXt"
```

4. Logical

→ Logical data types stores value - either TRUE or FALSE

→ Numerically, TRUE is the same as 1 & FALSE is the same as 0.

eg:- > TRUE * 5
[1] 5
> FALSE * 5
> 0.

→ To check if the variable is logical or not, use is.logical function.

> k ← TRUE
> class(k)
[1] "logical"
> is.logical(k)
> TRUE

→ We can also use the letters T & F as shortcuts for TRUE & FALSE respectively, but it is best not to use them, as it is easy to over write.

⇒ Vectors

~~→ Vectors are~~

→ A vector is a collection of or sequence of data elements of the same basic type. (like an array in C programming)

eg:- > x ← c(1,3,2,5)
y ← c("R", "Excel", "SAS", "C++")

→ To create a vector, we type c(vector type).

→ "c" stands for combine because multiple elements are being combined into a vector.

Vector Operations

→ In R, the operations are applied to each element of the vector automatically, without the need to loop through the vector.

→ eg:- > x ← c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

```
> x + 3
 [1]  3  6  9 12 15 18 21 24 27 30
```

→ And also the indexing of the vector elements starts from 1. [Unlike the other languages like c, c++ where the indexing starts from 0.].

→ We can do any basic mathematical operation on vector like this.

→ we can use : operator to create a vector.

```
eg:- > 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
```

→ we can also extend the vector operations, suppose we have two vectors of same length.

```
eg:- > x ← 1:10
> y ← -5:4
> x+y
 [1] -4 -2  0  2  4  6 10 12 14

> length(x)
[1] 10
> length(y)
[1] 10
> length(x+y)
[1] 10
```

→ If the vectors are of unequal length, The shorter vector gets recycled - ie; its elements are repeated, in order, to

they have been matched up with every element of the longer vector.

→ If the longer vector is not a multiple of the shorter one, a warning is given.

eg:- > x + c(1,2)

[1] 2 4 4 6 6 8 8 10 10 12

> x + c(1,2,3)

Warning in x+c(1,2,3): longer object length is not a multiple of shorter object length.

→ We can also do comparisons on vectors.

— Here the result is a vector of the same length containing TRUE or FALSE for each element.

eg:- x <= 5

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

→ To test all the resulting elements are TRUE, use all function

→ Use any function to check whether any element is TRUE

eg:- > x ← 10:1

> y ← -4:5

> any(x<y)

[1] TRUE

> all(x<y)

[1] FALSE

→ which function also acts on each element of a vector.

→ To access individual elements of a vector, we can use square brackets ([]).

→ The first element of x is retrieved by typing x[1]

- The first two elements by x[1:2]
- To retrieve non consecutive elements use x[c(1,4)]

```
> x[1]
[1] 10
> x[1:2]
[1] 10 9
> x[c(1,4)]
[1] 10 7.
```

→ To name a vector, we can give names to a vector either during creation or after the fact.

> eg:-># provide a name for each element of an array using a name-value pair

```
> c(One = "a", Two = "y", Last = "r")
 One  Two Last
 "a"  "y"  "r"
># create a vector
> w <- 1:3
># name the elements
> names(w) <- c("a", "b", "c")
> w
 a  b  c
 1  2  3
```

Factor Vectors

→ If you want to represent the values in a vector as nominal values and you want to categorise them, you can use the data structure called factor

→ The factor stores the nominal values as a vector of integers in the range of [1, ... k] (where k is the no. of

unique values in the nominal variable), and an internal vector of character strings (the ~~nominal~~ original values) mapped to these integers.

→ eg:- Suppose we have a variable with 20 "male" entries and 30 "female" entries.

gender ← c(rep("male", 20), rep("female", 30))

gender ← factor(gender)

Stores gender as 20 1s & 30 2s & associates 1 = female, 2 = male internally (alphabetically)

→ To create a vector

eg:- g2 ← c("Hockey", "Hockey", "Waterpolo", "Lacrosse", "Hockey")

→ To convert a vector to factor use as.factor() function

> g2Factor ← as.factor(g2)

→

→ List

→ A list in R contain many different data types inside it, which means list can hold arbitrary objects of either the same type or varying types.

→ A list is a collection of data which is ordered and changeable.

→ To create a list, use list() function.

→ eg:- list1 ← list("apple", "orange", "banana")

→ > n ← c(2,3,5)

> s ← c("aa", "bb", "cc", "dd", "ee")

> b ← c(TRUE, FALSE, TRUE, FALSE, FALSE)

> x ← list (n, 5, 6, 3)

→ Each element in a list may be given a name and then be referred to by that name.

→ eg:- Suppose we have a parcel to be sent through the mail. And it is destined for New York, has dimensions of 2 inches deep by 6 inches wide by 9 inches long, and costs $12.9 to mail.

> parcel ← list (destination = "New York" dimensions = c(2,6,9)
price = 12.95)

→ We can access the list items by referring to its index number, inside the square brackets ([ ]).

→ eg :- list1[1]

< apple

→ To change the value of a specific item, refer to the index number

eg :- list2 ← list ("one", "two", "four")

list[3] ← "three"

→ To find out how many items in a list, use length() fun?.

→ To find out if a specified item is present in a list, use %in% operator.

→ eg :- "apple" %in% list1

→ To add an item to the end of the list, use append() fun?

→ eg :- list3 ← list ("Tomato", "potato", "okra")

append (list3, "onion")

→ To remove an item from the list from the last position do [-1] to the index.

→ eg :- list3 ← list ("apple", "banana", "cherry")

newlist ← list3[-1]

Data Frames

* Data frames are data displayed in a format as a table.
* Each table has many rows & columns.
  - Columns have same data type - ie; a single colu should have the same data type, but a table can have multiple columns
  - Each column may be a different type, but each row in the data frame must have the same length.
* Usually, each column is named, and sometimes rows are named as well. The columns in the data frames are called variables.

* To create a dataframe, use the data.frame() function.
* eg:- > Data_Frame <- data.frame (

```
Training = c ("Strength", "Stamina", "Othe
Pulse = c (100, 150, 120),
Duration = c (60, 30, 45)
)
```

Data_Frame.

* Use the summary() function to summarise the data from a dataframe
* To access columns from the dataframe, we can use single brackets [ ], or double brackets [[ ]] or $ Symbol.

eg:- > Data_Frame [1]

> Data_Frame [["Training"]]

* To add a row in a Dataframe use the rbind() function.

eg:- > New_row_DF ← rbind(Data_Frame, c("Strength", 110, 110))

* To add a column in a DataFrame, use the cbind() function.

eg:- New_col_DF ← cbind(Data_Frame, Steps = c(1000, 6000, 4000))

* To remove rows & columns, we can use the c() function.

eg:- > Data_Frame_New ← Data_Frame[-c(1), -c(1)].

* Use the dim() function to find the amount of rows & columns in a Data frame.

eg:- > dim(Data_Frame)

* we can also use ncol() function to find the no. of columns & nrow() function to find the no. of rows.

eg:- ncol(Data_Frame)
nrow(Data_Frame)

* we can use the rbind() function to combine two or more data frames in R vertically.

eg:- Data_Frame1 ← data.frame(
Training = c("Strength", "Stamina", "Other"),
Pulse = c(100, 150, 120),
Duration = c(60, 30, 45)
)

Data_Frame2 ← data.frame(
Training = c("Stamina", "Stamina", "Strength"),
Pulse = c(140, 150, 160),
Duration = c(30, 30, 20)
)

New_Data_Frame ← rbind(Data_Frame1, Data_Frame2)

* Use the cbind() function to combine two or more data frames in R horizontally.

eg:- > Data-Frame 3 <- data.frame (
Training = c("Strength", "Stamina", "Other")
Pulse = c(100, 150, 120),
Duration = c(60, 30, 45)
)

Data-Frame 4 <- data.frame (
Steps = c(3000, 6000, 2000),
Calories = c(300, 400, 300)
)

New-Data-frame 1 <- cbind (Data-Frame 3, Data-Frame 4)

➡ Matrices

* A matrix is a two dimensional data set with columns & rows.
* A column is a vertical representation of data, while a row is a horizontal representation of data.
* In R, a matrix can be created with the matrix() function. To get the no. of rows & columns, specify the nrow and ncol parameters.

eg:- > matrix 1 <- matrix (c(1,2,3,4,5,6), nrow=3, ncol=2)
># print the matrix
> matrix 1.

* To access the items, you can use [ ] brackets. The first number in the bracket specifies the row position and the second number specifies the column position.

eg:- > matrix 2 <- matrix (c("apple", "banana", "cherry", "orange"),
nrow=2, ncol=2)

> matrix [1,2]

* To access the whole row, specify a comma after the number of row in the brackets.
eg:> thismatrix[2,]
* To access the whole columns, specify just the comma before the no. of columns.
eg:-> thismatrix[,2]

* To find the transpose of a matrix, use t() function.

eg:-> Suppose, we have a matrix B.
transpose of B ⇒ t(B).

* We can deconstruct a matrix by applying the c function, which combines all column vectors into one.
eg:- c(B)

* If we want to access more than one row, we can use the c() function
eg:-> matrix4 ← matrix(c("apple", "banana", "cherry", "orange", "grape", "pineapple", "pear", "melon", "fig") nrow=3, ncol=3)

>matrix4[c(1,2),]

* To access more than one column, use c() function, and where the argument of the column number is specified, give the column numbers you need.
eg:- matrix4[,c(1,2)]

* To add a column to the matrix, we can use the cbind() function -
eg:- newmatrix ← cbind(matrix4, c("Strawberry", "blueberry", "raspberry"))

* To add a row to the matrix use the rbind() function
eg:- newmatrix ← rbind(matrix4, c("Strawberry", "blueberry"

* We can perform, matrix addition, subtraction, multiplication & division using R.

eg:- # create two 2x3 matrices.

```
matrix1 <- matrix (c(3,9,-1,4,2,6), nrow = 2)
print (matrix1)

matrix2 <- matrix (c(5,2,0,9,3,4), nrow = 2)
print (matrix2)

# Add the matrices
    result <- matrix1 + matrix2
    cat ("Result of Addition", "\n")
    print (result)
# Subtract the matrices
    result <- matrix1 - matrix2
    cat ("Result of subtraction", "\n")
    print (result)
# multiply the matrices
```

Arrays

* An array is essentially a multi dimensional vector.
* It must be of the same type, and individual elements are accessed in a similar fashion using square brackets.
* The first element is the row index, the second is the column index and the remaining elements are for outer dimensions.
* Eg:- >thearray <- array (1:12, dim = c(2,3,2))
  >thearray
* the main difference blw an array & a matrix is that matrices are restricted to two dimensions, while arrays can have an arbitrary number.
* we can also name the rows & columns in an array.

eg:- > r <- c ("ROW1", "ROW2", "ROW3")
> c <- c ("COL1", "COL2", "COL3")
> m <- c ("Matrix 1", "Matrix 2")
> array (1:9, c (3,3,2), list (r, c, m))

* To access the array elements we use the square brackets.

eg:- # print the third row of the second matrix of the array.
> print (result [3,,2])

Print the element in the 1st row & 3rd column of the 1st matrix
> print (result [1,3,1])

* Control Statements

R Operators

* R divides the operators into the following groups:

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. Logical operators
5. Miscellaneous operators.

R Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations

Operator	Name	Example
+	Addition	x+y
−	Subtraction	x-y
*	Multiplication	x*y
/	Division	x/y
^	Exponent	x^y
%%%	Modulus (Remainder from division)	x%%%y
%/%	Integer Division	x%/%y

R Assignment Operators: Used to assign values to the variables

Operator	Description
<-, <<-, =	Leftwards assignment
->, ->>	Rightwards assignment

Relational Operators (Comparison operators): Used to compare two values

Operator	Description
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	Equal to
!=	Not equal to

Logical Operators

: Used to combine conditional statements.

Operator	Description
&	Element-wise Logical AND operator. It returns TRUE if both elements are TRUE
&&	Logical AND operator - Returns TRUE if both statements are TRUE
|	Element-wise - Logical OR operator. It returns TRUE if any one of the element is TRUE
||	Logical OR operator. It returns TRUE if one of the Statement is TRUE
!	Logical NOT - returns FALSE if statement is TRUE

Miscellaneous Operators

Miscellaneous operators are used to manipulate data.

Operator	Description	Example
:	Creates a series of numbers in a sequence	x<- 1:10
%in%	Find out if an element belongs to a vector	x%in%y
%*%	Matrix multiplication	x<-Matrix1 %*% Matrix2

Control Statements

* Control statements allows us to control the flow of our programs and cause different things to happen, depending on the values of tests.
* Tests results in a logical, TRUE or FALSE, which is used like if-like statements.
* The main control statements are if, else, if else, & switch

1. If and else

* The most common test is the if command
* It means, if something is TRUE, then perform some action; otherwise, do not perform an action.
* We use the comparative operators to check the if statement.

Syntax:

```
if (condition)
  {
    Statement
  }
```

eg:-> tocheck <- 1

```
>if(tocheck==1)
  { print("hello!")
  }
```

if & else

Syntax

```
if (condition)
  {
    Statement
  }else
     {
       Statement
     }
```

* Here, Note that else is on the same line as its preceding closing curly brace (}). This is important, otherwise the code will fail.

eg:- > check.bool <- function(x)

```
{
  if (x==1)
   {
    # if the i/p is equal to 1, print hello
     print("hello")
   } else if(x==0)
      {
       # if the i/p is equal to 0, print goodbye
         print("goodbye")
       } else
          {
           # otherwise print confused
             print("confused")
          }
}
> check.bool(1)
```

2. Switch

* If we have multiple cases to check, writing else if repeatedly can be cumbersome & inefficient.
* Instead we use switch
* The first argument is the value we are testing.
* Subsequent arguments are a particular value and what should be the result.

eg:- > use.switch <- function(x)
{
switch(x,
"a" = "first",
"b" = "second",
"z" = "last",
"c" = "third",
"other")
}

3. ifelse

* ifelse is more like the if functions in Excel.
* The first argument is the condition to be tested, the second argument is the return value if the test is TRUE and the third argument is the return value if the test is FALSE.

eg:-ifelse(1==1, "yes", "no")

> toTest <- c(1,1,0,1,0,1)
ifelse(toTest==1, "yes", "no")

* Loops

for loop

Syntax :

```
for (val in sequence)
  {
    Statement
  }
```

eg:-
```
for ( i in 1:10)
    {
      print(i)
    }
```

while loop

Syntax:

```
while (test_expression)
  {
    Statement
  }
```

eg:-
```
> x <- 1
> while x(<=5)
    {
      print (x)
      x <- x+1
    }
```

Controlling Loops

* Sometimes we have to skip to the next iteration of the loop or completely break out of it. This is accomplished with **next** and **break**.

* A **break** statement is used inside a loop (repeat, for, while) to stop the iterations and flow of the control outside of the loop.

* A next statement is useful when we want to skip the current iteration of a loop without terminating it.

eg:- if (test_expression) {
        break
      }

```
x <- 1:15
for (val in x) {
    if (val == 3) {
        break
    }
    print(val)
}
```

```
* for (i in 1:20)
  {
    if (i==3)
      {next
      }
    print(i)
  }
```

Repeat Loop

* A repeat loop is used to iterate over a block of code multiple of times
* There is no condition check in repeat loop to exit the loop.
* Use the break statement to exit the loop. Failing to do so will result pato an infinite loop.

Syntax

```
repeat {
    Statement
}
```

Functions

* Functions are the R objects that evaluate a set of i/p arguments and returns an o/p value.
* In R, function objects are defined with this syntax.

function (arguments) body

* arguments is a set of symbol names (and, optionally, default values) that will be defined within the body of the function, and body is an R expression.

eg:- > f ← function (x, y) x+y
> f ← function (x, y) { x+y }

Arguments

* A function definition in R includes the names of arguments.
  - If you specify a default value for an argument, then the argument is considered optional
  - You can have as many as arguments, separated with comma.

eg:- > f ← function (x, y) { x+y }
> f(1,2)
[1] 3
> g ← function (x, y=10) { x+y }
> g(1)
[1] 11

* By default, a function must be called with the correct number of arguments, meaning that if your function expects 2 arguments, you have to call the function with 2 arguments - not more, and not less:

```
eg:- my_function ← function (fname, lname){
        paste (fname, lname)
     }
myfunction ("Peter", "Griffin")
```

Return Values

* To let a function return a result, use the return() function.

```
eg:- my_function <- function(x) {
                        return(5*x)
                    }
     print(my_function(3))
```

Functions As Arguments

* Many functions in R can take other functions as arguments
* eg:- sapply function
  - The sapply function iterates through each element in a vector, and returning the results.

```
eg:- > a <- 1:7
     > sapply(a, sqrt)
     [1] 1.0000 1.414214 1.732051 2.0000 2.236068 2.4494
     2.645751
```

Anonymous functions

* Functions are just objects in R. So it is possible to create functions that do not have names.
* These are called anonymous functions
* Anonymous functions are usually passed as arguments to another functions.

eg:- let create a function named apply_to_three

```
> apply_to_three <- function(f) { f(3)}
> apply_to_three (function(x) {x*7})
```

- when we call the function, R interpreter evaluate the expression, The interpreter first evaluates the expression f(3). The interpreter assign 3 to the argument x

the anonymous function. Finally, the interpreter evaluates the expression 3+7 & returns the result.

Named Arguments

* In R, functions can have named arguments, allowing you to specify values for specific arguments by name rather than their position.
* This feature provides flexibility and improves code readability, especially when functions have numerous arguments.

*eg: # Define a function with named arguments

```
calculate_area <- function (length=0, width=0) {
    area <- length * width
    return (area)
}
```

Call the calculate_area function with named arguments

```
rectangle_area <- calculate_area (length=5, width=3)
print (rectangle_area)
```

- In this example, the calculate_area fun" calculates the area of a rectangle. The function has two named arguments : length & breadth. These arguments have default values of 0, meaning they are optional. If no values are provided when calling the function, the default values will be used.

- Named arguments offer flexibility and clarity when working with functions that have multiple parameters, allowing you to specify values explicitely without relying on the order of arguments