SHARDA UNIVERSITY

Beyond Boundaries

SCHOOL OF BASIC SCIENCE AND RESEARCH

Department of Mathematics

LAB REPORT FILE

Course Title: R Programming Lab

Course Code: BDA154

Program-Bachelor of Science (Hons.)-Data Science

Semester: II

Session 2021-2022

Student Name: - Jatin Verma

System ID: - 2021370138

Roll No.- 2107158011

Submitted to

Dr. Surya Kant Pal

(Assistant Professor)

INDEX

PRACTICAL-1

R Programming Tutorial

Student Name: Jatin Verma	**System ID:** 2021370138
Branch: BSc Data Science & Analytics	**Section/Group:** A
Semester: II	**Date of Performance:** 15/03/22
Subject Name: R-Programming	**Subject Code:** BDA 154

❖ **AIM:**

To understand R programming Tutorial.

❖ **THEORY:**

❖ **What is R Programming?**

R is an open-source programming language that is widely used as a statistical software and data analysis tool. R generally comes with the Command-line interface. R is available across widely used platforms like Windows, Linux, and macOS. Also, the R programming language is the latest cutting-edge tool.

It was designed by **Ross Ihaka and Robert Gentleman** at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team. R programming language is an implementation of the S programming language. It also combines with lexical scoping semantics inspired by Scheme. Moreover, the project conceives in 1992, with an initial version released in 1995 and a stable beta version in 2000.
R allows integration with the procedures written in the C, C++, .Net, Python, and FORTRAN languages to improve efficiency. In the present era, R is one of the most important tool which is used by researchers, data analyst, statisticians, and marketers for retrieving, cleaning, analyzing, visualizing, and presenting data.

❖ **Features Of R Programming**

1) R programming is used as a leading tool for machine learning, statistics, and data analysis. Objects, functions, and packages can easily be created by R.
2) It’s a platform-independent language. This means it can be applied to all operating system.

3) It’s an open-source free language. That means anyone can install it in any organization without purchasing a license.
4) R programming language is not only a statistic package but also allows us to integrate with other languages (C, C++). Thus, you can easily interact with many data sources and statistical packages.
5) The R programming language has a vast community of users and it’s growing day by day.
6) R is currently one of the most requested programming languages in the Data Science job market that makes it the hottest trend nowadays.

➢ Statistical Features Of R Programming

- **Basic Statistics:** The most common basic statistics terms are the mean, mode, and median. These are all known as “Measures of Central Tendency.” So using the R language we can measure central tendency very easily.
- **Static graphics:** R is rich with facilities for creating and developing interesting static graphics. R contains functionality for many plot types including graphic maps, mosaic plots, biplots, and the lst goes on.
- **Probability distributions:** Probability distributions play a vital role in statistics and by using R we can easily handle various types of probability distribution such as Binomial Distribution, Normal Distribution, Chi-squared Distribution and many more.
- **Data analysis:** It provides a large, coherent and integrated collection of tools for data analysis.

➢ Programming Features Of R Programming

- **R Packages:** One of the major features of R is it has a wide availability of libraries. R has CRAN(Comprehensive R Archive Network), which is a repository holding more than 10, 0000 packages.
- **Distributed Computing:** Distributed computing is a model in which components of a software system are shared among multiple computers to improve efficiency and performance. Two new packages **ddR and multidplyr** used for distributed programming in R were released in November 2015.

❖ Installation of R:

R programming is a very popular language and to work on that we have to install two things, i.e., R and RStudio. R and RStudio works together to create a project on R. Installing R to the local computer is very easy. First, we must know which operating system we are using so that we can download it accordingly.

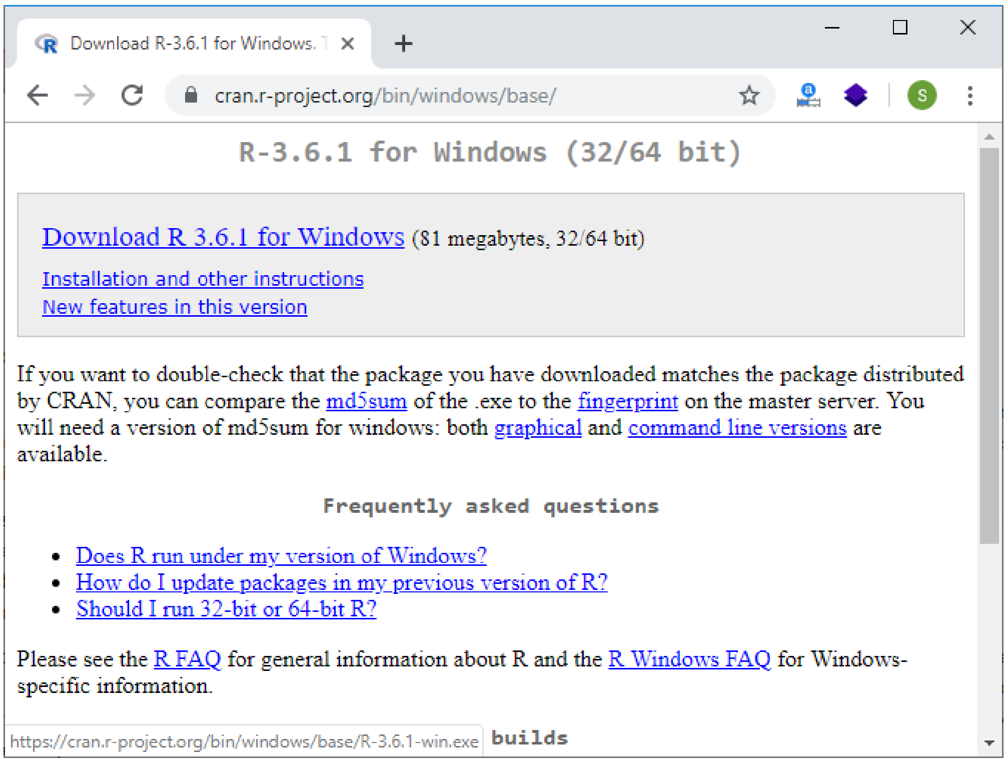
The official site https://cloud.r-project.org provides binary files for major operating systems including Windows, Linux, and Mac OS. In some Linux distributions, R is installed by default, which we can verify from the console by entering R.

To install R, either we can get it from the site https://cloud.r-project.org or can use commands from the terminal.

❖ Install R in Windows:

➢ There are following steps used to install the R in Windows:

♦ **Step 1:**First, we have to download the R setup from https://cloud.r-project.org/bin/windows/base/.


Download R-3.6.1 for Windows.

cran.r-project.org/bin/windows/base/

R-3.6.1 for Windows (32/64 bit)

Download R 3.6.1 for Windows (81 megabytes, 32/64 bit)

Installation and other instructions

New features in this version

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the md5sum of the .exe to the fingerprint on the master server. You will need a version of md5sum for windows: both graphical and command line versions are available.

Frequently asked questions

- Does R run under my version of Windows?
- How do I update packages in my previous version of R?
- Should I run 32-bit or 64-bit R?

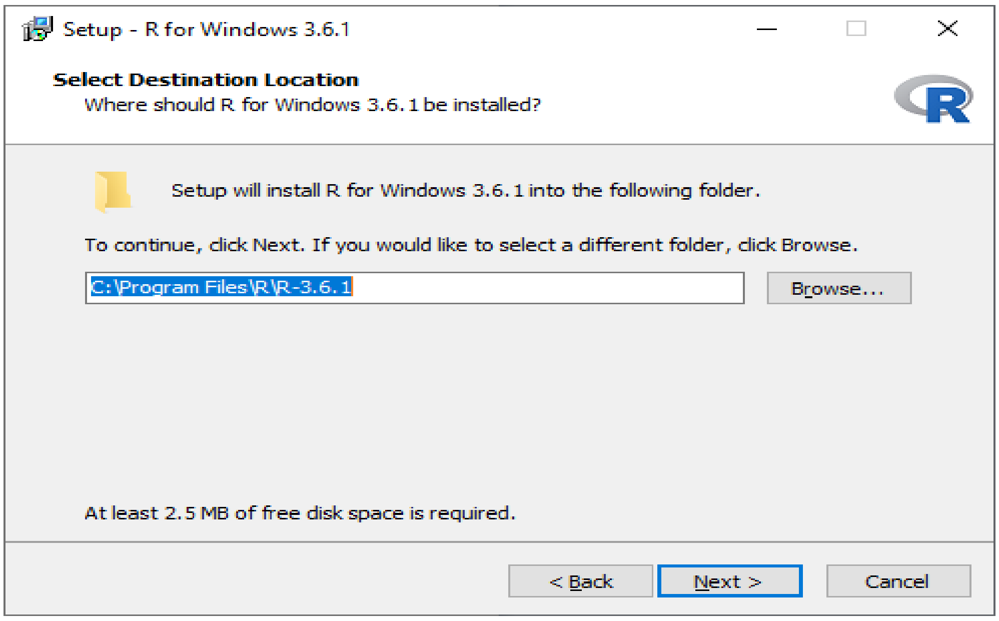
Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

https://cran.r-project.org/bin/windows/base/R-3.6.1-win.exe builds

♦ **Step 2:**

When we click on **Download R 3.6.1 for windows**, our downloading will be started of R setup. Once the downloading is finished, we have to run the setup of R in the following way:

1) Select the path where we want to download the R and proceed to Next.



Setup - R for Windows 3.6.1

Select Destination Location

Where should R for Windows 3.6.1 be installed?

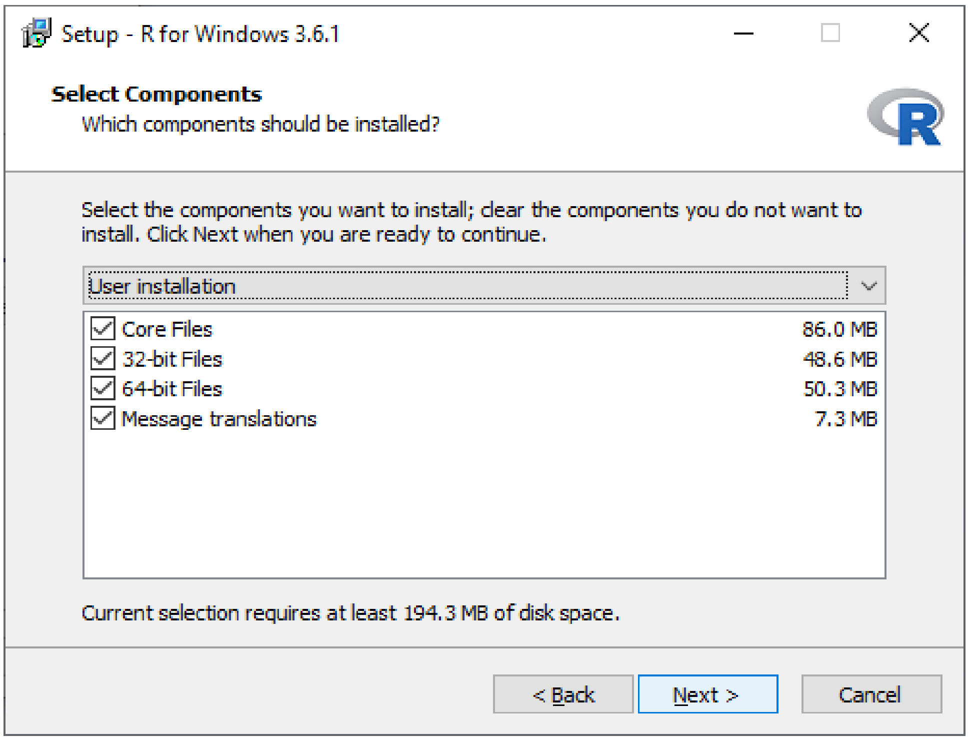
Setup will install R for Windows 3.6.1 into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files\R\R-3.6.1

At least 2.5 MB of free disk space is required.

2) Select all components which we want to install, and then we will proceed to **Next**.



Setup - R for Windows 3.6.1

Select Components

Which components should be installed?

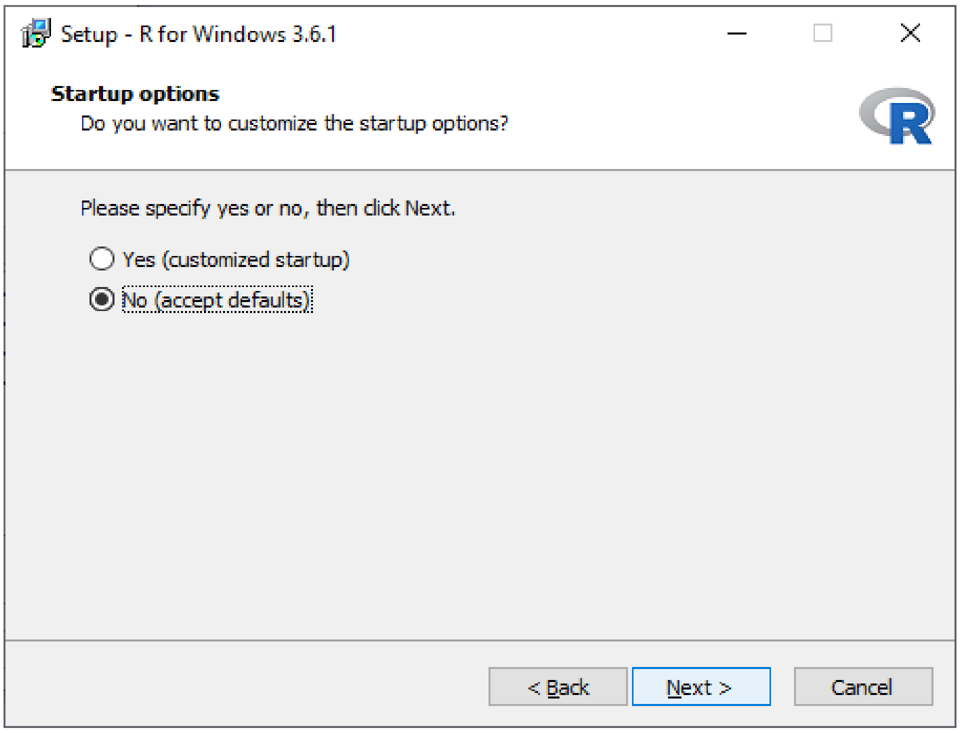
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

User installation

✓	Core Files	86.0 MB
✓	32-bit Files	48.6 MB
✓	64-bit Files	50.3 MB
✓	Message translations	7.3 MB

Current selection requires at least 194.3 MB of disk space.

3) In the next step, we have to select either customized startup or accept the default, and then we proceed to **Next**.


Setup - R for Windows 3.6.1

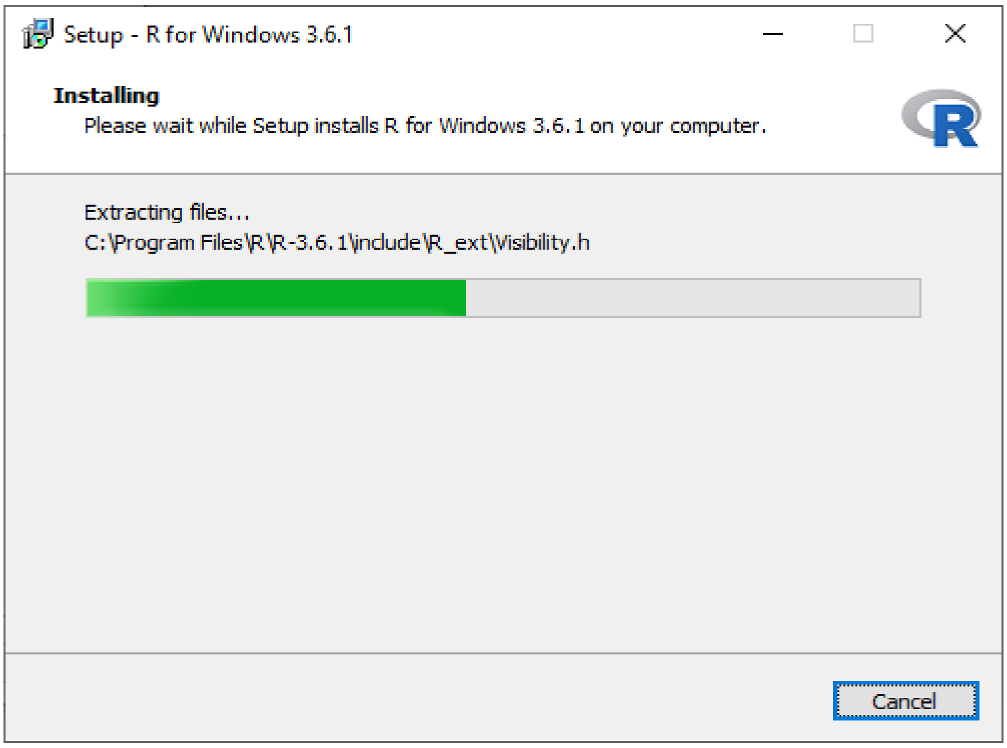
Startup options

Do you want to customize the startup options?

Please specify yes or no, then click Next.

Yes (customized startup) [✓ No (accept defaults)]

4) When we proceed to next, our installation of R in our system will get started:


Setup - R for Windows 3.6.1

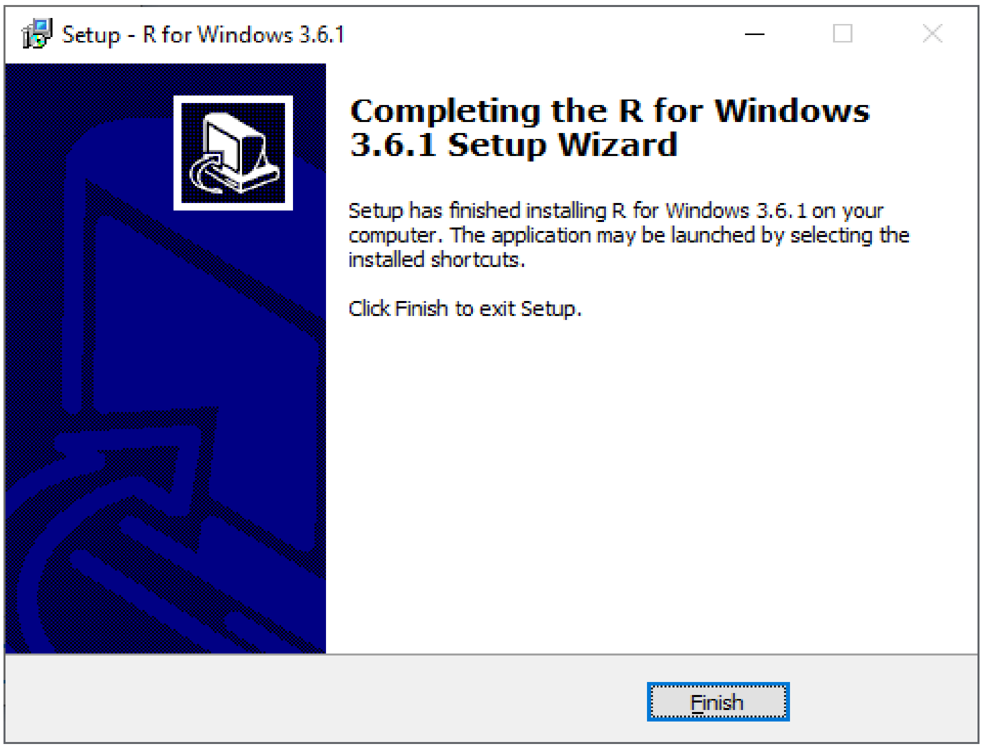
Installing

Please wait while Setup installs R for Windows 3.6.1 on your computer.

Extracting files...

C:\Program Files\R\R-3.6.1\include\R_ext\Visibility.h

5) In the last, we will click on finish to successfully install R in our system.


Setup - R for Windows 3.6.1

Completing the R for Windows 3.6.1 Setup Wizard

Setup has finished installing R for Windows 3.6.1 on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

❖ <u>R Studio IDE:</u>

RStudio is an integrated development environment which allows us to interact with R more readily. RStudio is similar to the standard RGUI, but it is considered more user-friendly. This IDE has various drop-down menus, Windows with multiple tabs, and so many customization processes. The first time when we open RStudio, we will see three Windows. The fourth Window will be hidden by default. We can open this hidden Window by clicking the **File** drop-down menu, then **New File** and then **R Script**.

RStudio Windows/Tabs	**Location**	**Description**
Console Window	Lower-left	The location where commands are entered and output is printed.
Source Tabs	Upper-left	Built-in test editor
Environment Tab	Upper-left	An interactive list of loaded R objects.
History Tab	Upper-left	List of keystrokes entered into the console.
Files Tab	Lower-right	File explorer to navigate C drive folders.

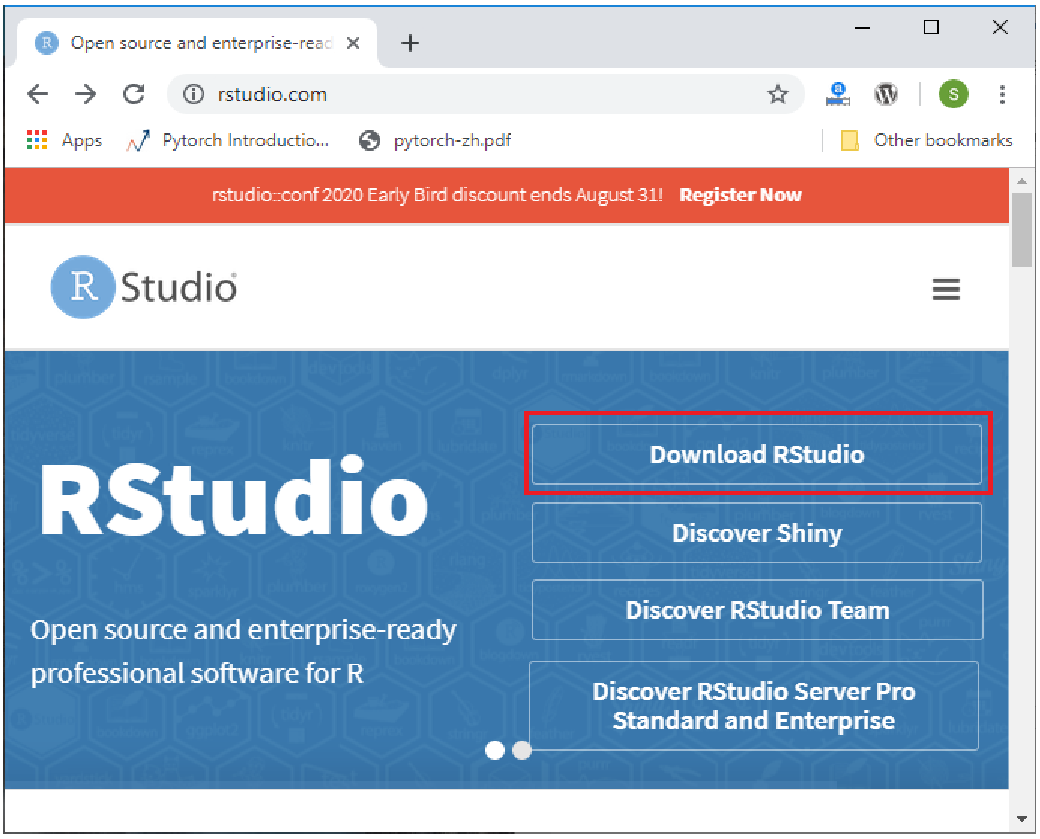
Plots Tab	Lower-right	Output location for plots.
Packages Tab	Lower-right	List of installed packages.
Help Tab	Lower-right	Output location for help commands and help search Window.
Viewer Tab	Lower-right	Advanced tab for local web content.

❖ <u>Installation Of R Studio in Windows:</u>

On Windows and Linux, it is quite simple to install RStudio. The process of installing RStudio in both the OS is the same. There are the following steps to install RStudio in our Windows/Linux:

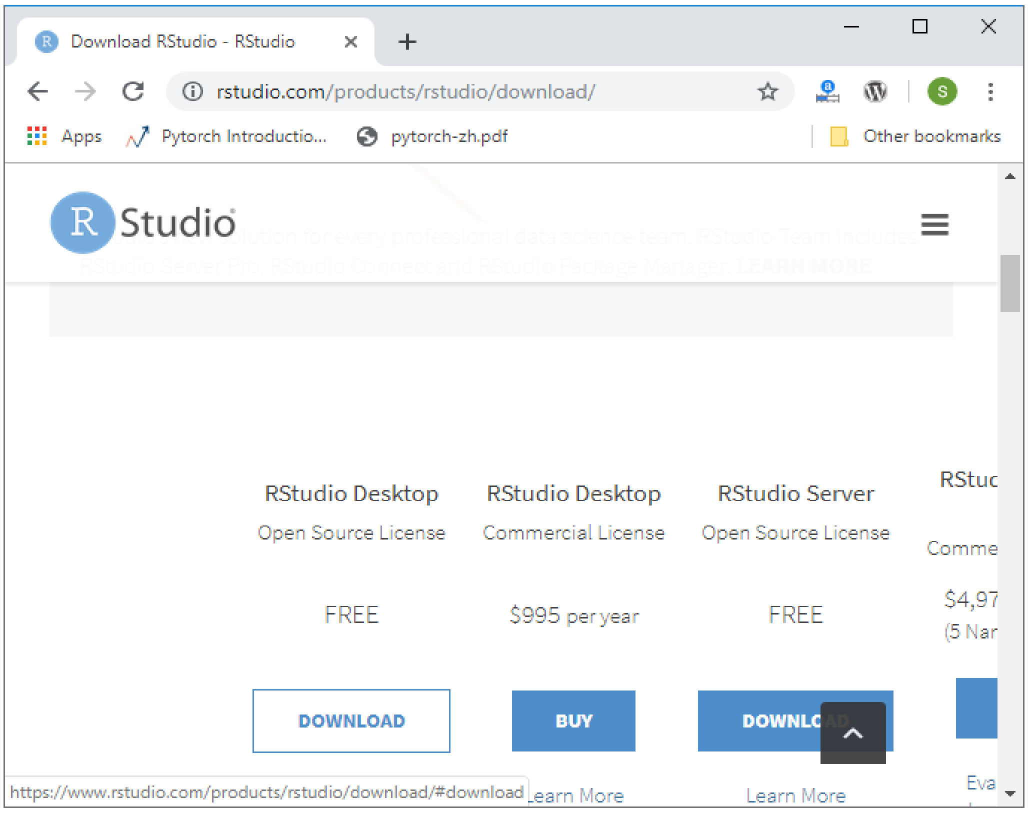
♦ <u>Step 1:</u>

In the first step, we visit the RStudio official site and click on **Download RStudio**.


Open source and enterprise-read
rstudio.com
Apps
Pytorch Introductio...
pytorch-zh.pdf
Other bookmarks
rstudio::conf 2020 Early Bird discount ends August 31! Register Now
RStudio
RStudio
Open source and enterprise-ready professional software for R
Download RStudio
Discover Shiny
Discover RStudio Team
Discover RStudio Server Pro Standard and Enterprise

♦ **<u>Step 2:</u>**

In the next step, we will select the RStudio desktop for open-source license and click on download.

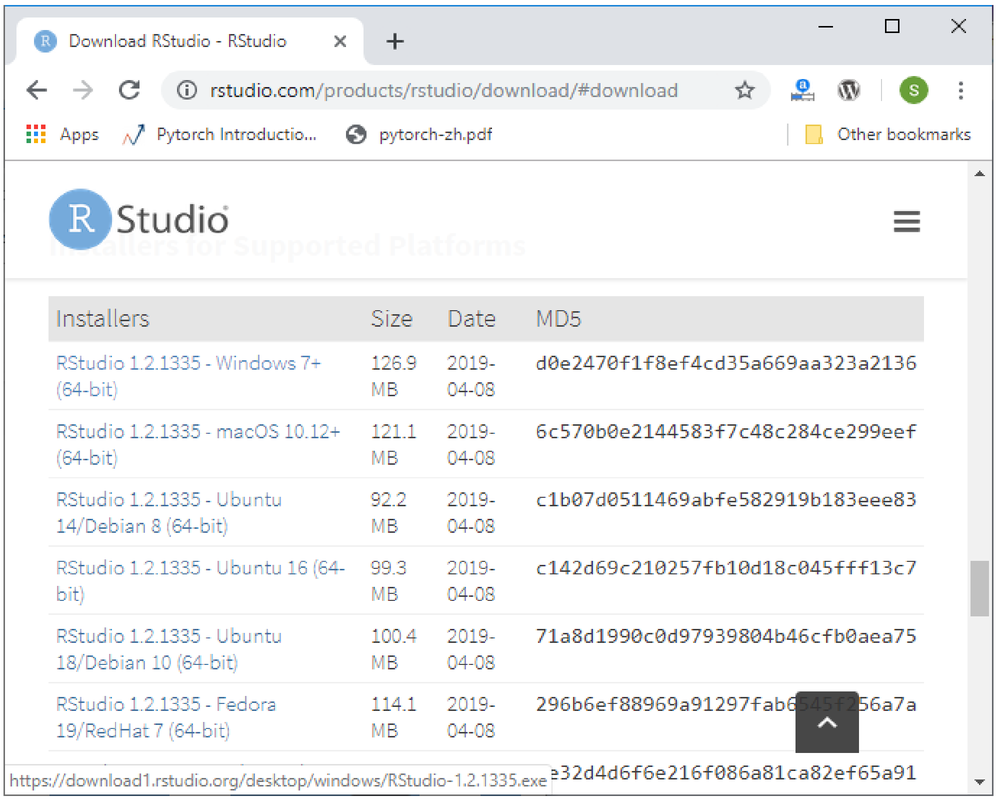

Download RStudio - RStudio
rstudio.com/products/rstudio/download/
Apps
Pytorch Introductio...
pytorch-zh.pdf
Other bookmarks
RStudio

RStudio Desktop	RStudio Desktop	RStudio Server	RStuc
Open Source License	Commercial License	Open Source License	Comme
FREE	$995 per year	FREE	$4,97 (5 Nar
DOWNLOAD	BUY	DOWNLOAD	
	Learn More	Learn More	Eva

https://www.rstudio.com/products/rstudio/download/#download

♦ **<u>Step 3:</u>**

In the next step, we will select the appropriate installer. When we select the installer, our downloading of RStudio setup will start.


Download RStudio - RStudio
rstudio.com/products/rstudio/download/#download
Apps
Pytorch Introductio...
pytorch-zh.pdf
Other bookmarks
RStudio

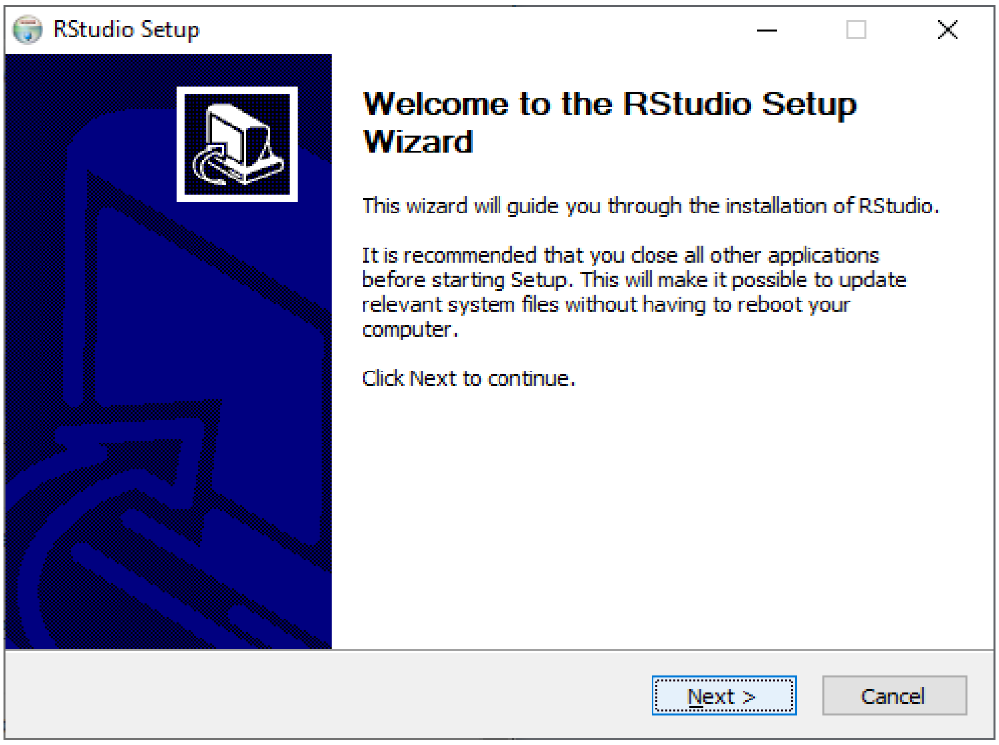
Installers	Size	Date	MD5
RStudio 1.2.1335 - Windows 7+ (64-bit)	126.9 MB	2019-04-08	d0e2470f1f8ef4cd35a669aa323a2136
RStudio 1.2.1335 - macOS 10.12+ (64-bit)	121.1 MB	2019-04-08	6c570b0e2144583f7c48c284ce299eef
RStudio 1.2.1335 - Ubuntu 14/Debian 8 (64-bit)	92.2 MB	2019-04-08	c1b07d0511469abfe582919b183eee83
RStudio 1.2.1335 - Ubuntu 16 (64-bit)	99.3 MB	2019-04-08	c142d69c210257fb10d18c045fff13c7
RStudio 1.2.1335 - Ubuntu 18/Debian 10 (64-bit)	100.4 MB	2019-04-08	71a8d1990c0d97939804b46cfb0aea75
RStudio 1.2.1335 - Fedora 19/RedHat 7 (64-bit)	114.1 MB	2019-04-08	296b6ef88969a91297fab6545f256a7a
			e32d4d6f6e216f086a81ca82ef65a91

https://download1.rstudio.org/desktop/windows/RStudio-1.2.1335.exe

♦ **Step 4:**

In the next step, we will run our setup in the following way:

1) Click on Next.



RStudio Setup

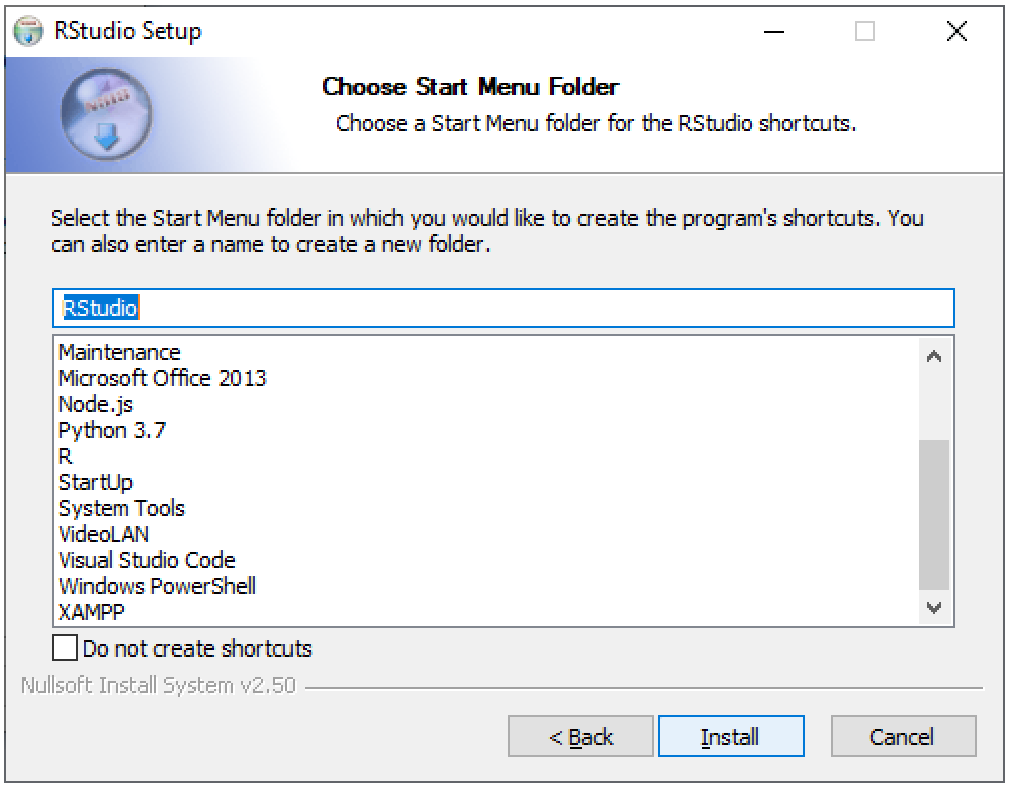
Welcome to the RStudio Setup Wizard

This wizard will guide you through the installation of RStudio.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

2) Click on Install.



RStudio Setup

Choose Start Menu Folder

Choose a Start Menu folder for the RStudio shortcuts.

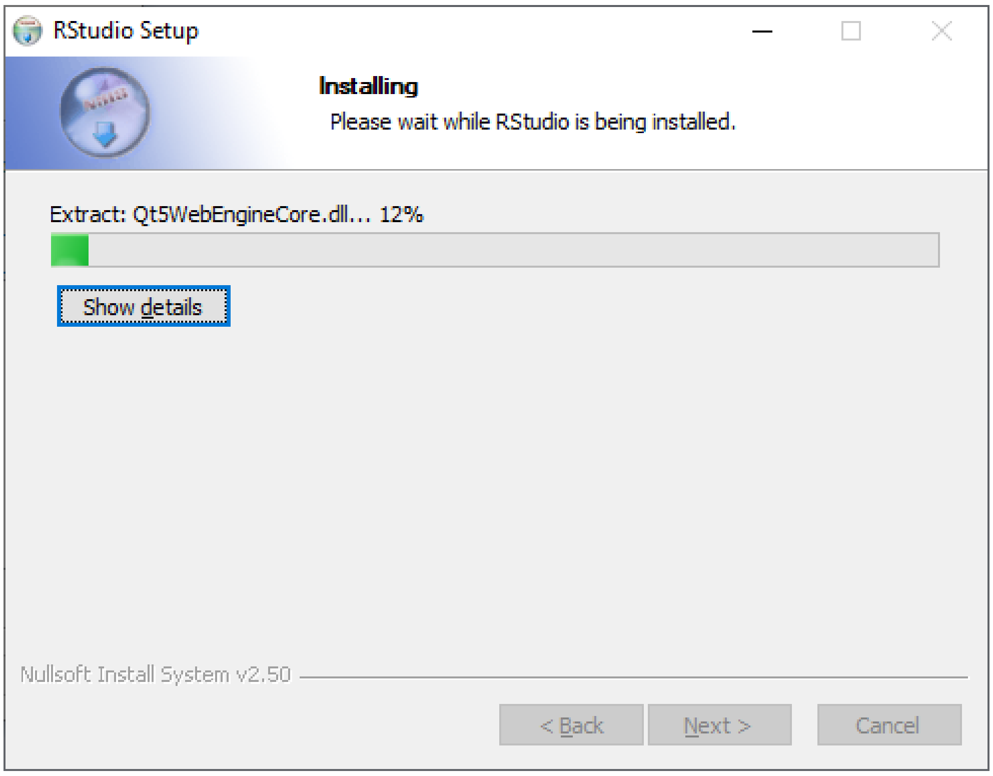
Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

RStudio

Maintenance
Microsoft Office 2013
Node.js
Python 3.7
R
StartUp
System Tools
VideoLAN
Visual Studio Code
Windows PowerShell
XAMPP

☐ Do not create shortcuts

Nullsoft Install System v2.50



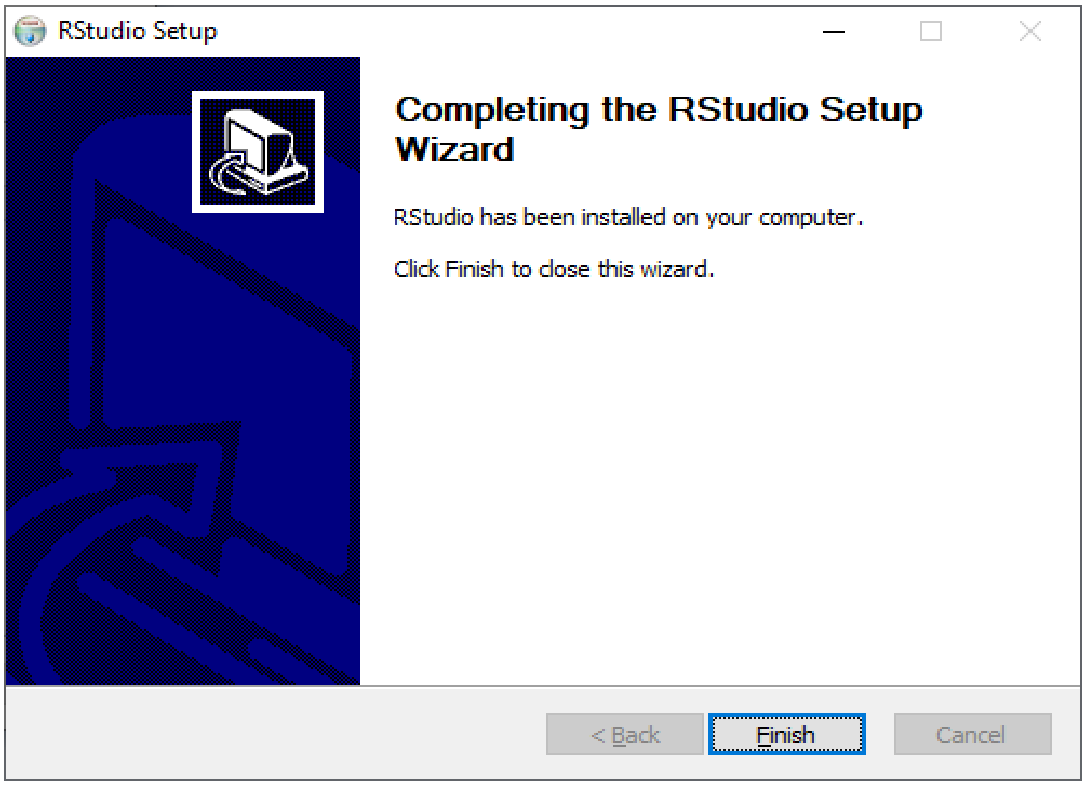
RStudio Setup

Installing

Please wait while RStudio is being installed.

Extract: Qt5WebEngineCore.dll... 12%

3) Click on finish.



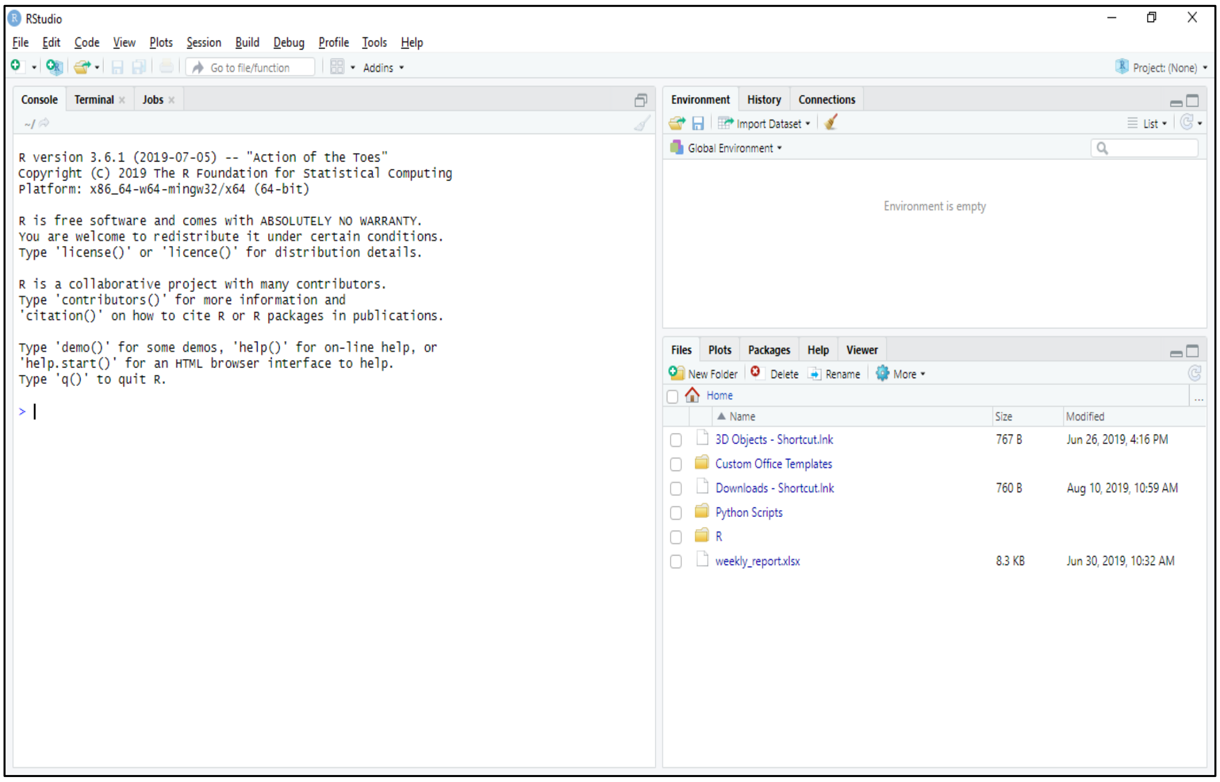
RStudio Setup

Completing the RStudio Setup Wizard

RStudio has been installed on your computer.

Click Finish to close this wizard.

4) RStudio is ready to work.



RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Console Terminal Jobs
Environment History Connections
Import Dataset
List
Global Environment
Environment is empty
Files Plots Packages Help Viewer
New Folder Delete Rename More
Home

```
R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 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.

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.

> |
```

Name	Size	Modified
3D Objects - Shortcut.lnk	767 B	Jun 26, 2019, 4:16 PM
Custom Office Templates		
Downloads - Shortcut.lnk	760 B	Aug 10, 2019, 10:59 AM
Python Scripts		
R		
weekly_report.xlsx	8.3 KB	Jun 30, 2019, 10:32 AM

❖ <u>OBSERVATIONS:</u>

We have installed R and R Studio for our Windows Operating System and installed different packages in R.

❖ <u>Learning outcomes (What I have learnt):</u>

1. Introduction to R programming.
2. Features of R programming.
3. Installation of R
4. Installation of R Studio

❖ **<u>Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):</u>**

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			

PRACTICAL-2

Datatypes in R Programming

Student Name: Jatin Verma	**System ID:** 2021370138
Branch: BSc Data Science & Analytics	**Section/Group:** A
Semester: II	**Date of Performance:** 22/03/22
Subject Name: R-Programming	**Subject Code:** BDA 154

❖ AIM:
To understand datatypes in R programming.

❖ THEORY:
In programming languages, we need to use various variables to store various information. Variables are the reserved memory location to store values. As we create a variable in our program, some space is reserved in memory. Each variable in R has an associated data type. Each data type requires different amounts of memory and has some specific operations which can be performed over it.
We can use the class() function to check the data type of a variable
the following data types which are used in R programming:

➢ Numeric:
The numeric data type is for numeric values. It is the default data type for numbers in R.
Examples of numeric values would be 1, 34.5, 3.145, -24, -45.003, etc.

➢ Integer:
The Integer data type is used for integer values. The integer data type is commonly used for discrete only values like unique IDs. To create an integer variable in R, we need to call the **as.integer()** function while assigning value to a variable. We can also convert a value into an integer type using the **as.integer()** function. We can also use the capital 'L' notation as a suffix to denote that a particular value is of the integer data type.
For example: 3L, 66L, 567L etc. Here, L tells R to store the value as an integer.

- **Complex:**

R supports complex data types that are set of all the complex numbers. The values containing the imaginary number ‘i’ (iota) are called complex values. The complex data type is to store numbers with an imaginary component.
Examples of complex values would be 1+2i, 3i, 4-5i, -12+6i, etc.

- **Character:**

R supports character data types where we have all the alphabets and special characters. The character data type stores character values or strings. Strings in R can contain the alphabet, numbers, and symbols. The easiest way to denote that a value is of character type in R is to wrap the value inside single or double inverted commas.
We can also use the **as.character()** function to convert objects into character values.
For example: "k", "R is exciting", "FALSE", "11.5".

- **Logical:**

A logical data type stores either of the two Boolean values: TRUE / FALSE. A logical value is often generated when two values are compared. Three standard logical operations i.e., AND(&), OR(|), and NOT(!) yield a variable of the logical data type.

❖ PROGRAM AND OUTPUT:

- **Numeric:**

- **Code:**

```
> num = 1
> class(num)
```

- **Output:**

```
[1] “numeric”
```

- **Integer:**

- **Code:**

```
> int = as.integer(16)
> class(int)
```

- **Output:**

```
[1] “numeric”
```

- **Complex:**

- **Code:**

```
> comp = 22-6i
> class(comp)
```

- **Output:**

```
[1] "complex"
```

- **Character:**

- **Code:**

```
> char = “Harsh”
> class(char)
```

- **Output:**

```
[1] "character"
```

- **Logical:**

- **Code:**

```
> log = FALSE
> class(log)
```

- **Output:**

```
[1] "logical"
```

- **Code:**

  > x=TRUE

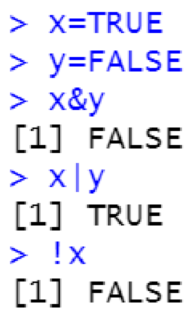
  > y=FALSE

  > x&y

  > x|y

  >! X

- **Output:**



```
> x=TRUE
> y=FALSE
> x&y
[1] FALSE
> x|y
[1] TRUE
> !x
[1] FALSE
```

❖ **OBSERVATIONS:**
We done Data Types in R programming– different R data types, their syntax and example R commands for R data types.

❖ **Learning outcomes (What I have learnt):**

1. How to use different data types in R Programming i.e. Numeric, Integer, Complex, Logical, Character.

❖ <u>**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**</u>

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			

PRACTICAL-3

Data Operators in R Programming

Student Name: Jatin Verma	**System ID:** 2021370138
Branch: BSc Data Science & Analytics	**Section/Group:** A
Semester: II	**Date of Performance:** 29/03/22
Subject Name: R-Programming	**Subject Code:** BDA 154

❖ **AIM:**

To understand data operators in R programming.

❖ **THEORY:**

Operators are the symbols which tells the compiler to perform various kinds of operations between the operands. Operators simulate the various mathematical, logical, and decision operations performed on a set of Complex Numbers, Integers, and Numerical as input operands.
In R programming, there are different types of operator, and each operator performs a different task.

➢ **There are the following types of operators used in R programming:**

➢ **Arithmetic Operators:**

Arithmetic operators simulate various math operations, like addition, subtraction, multiplication, division, and modulo using the specified operator between operands, which may be either scalar values, complex numbers, or vectors. The operations are performed element-wise at the corresponding positions of the vectors. There are various arithmetic operators which are supported by R:

Operator	Description	Usage
+	Addition of two operands	a + b
–	Subtraction of second operand from first	a – b

Operator	Description	Usage
*	Multiplication of two operands	a * b
/	Division of first operand with second	a / b
%%	Remainder from division of first operand with second	a %% b
%/%	Quotient from division of first operand with second	a %/% b
^	First operand raised to the power of second operand	a^b

- **Relational Operators:**
  The relational operators carry out comparison operations between the corresponding elements of the operands. A relational operator compares each element of the first vector with the corresponding element of the second vector. The result of the comparison will be a Boolean value (TRUE or FALSE). There are the following relational operators which are supported by R:

Operator	Description	Usage
<	Is first operand **less than** second operand	a < b
>	Is first operand **greater than** second operand	a > b
==	Is first operand **equal to** second operand	a == b

Operator	Description	Usage
<=	Is first operand **less than** or equal to second operand	a <= b
>=	Is first operand **greater than** or equal to second operand	a > = b
!=	Is first operand **not equal** to second operand	a!=b

- **Logical Operators:**
  Logical operations simulate element-wise decision operations, based on the specified operator between the operands, which are then evaluated to either a True or False Boolean value. Logical operators are applicable to those vectors whose type is logical, numeric, or complex. The logical operator compares each element of the first vector with the corresponding element of the second vector.

Operator	Description	Usage
&	This operator is known as the Logical AND operator. This operator takes the first element of both the vector and returns TRUE if both the elements are TRUE.	a & b
|	This operator is known as the Logical OR operator. This operator takes the first element of both the vector and returns TRUE if one of them is TRUE.	a | b
!	This operator is known as Logical NOT operator. This operator takes the first element of the vector and gives the opposite logical value as a result.	!a

Operator	Description	Usage
&&	This operator takes the first element of both the vector and gives TRUE as a result, only if both are TRUE.	a && b
||	This operator takes the first element of both the vector and gives the result TRUE, if one of them is true.	a || b

- **<u>Assignment Operators:</u>**
  An assignment operator is used to assign a new value to a variable. In R, these operators are used to assign values to vectors. There are two kinds of assignment operators: Left and Right.

S. No	Operator	Description	
1.	<- or = or <<-	These operators are known as left assignment operators.	
2.	-> or ->>	These operators are known as right assignment operators.	

- **<u>Miscellaneous Operators:</u>**
  Miscellaneous operators are used for a special and specific purpose. These operators are not used for general mathematical or logical computation.

Operator	Description	Usage
:	Creates series of numbers from left operand to right operand	a:b

Operator	Description	Usage
%in%	Identifies if an element(a) belongs to a vector(b)	a %in% b
%*%	Performs multiplication of a vector with its transpose	A %*% t(A)

❖ PROGRAM AND OUTPUT:

➢ Arithmetic Operators:

- **Code:**

```
a <- 7.5
b <- 2

print (a+b)
print (a-b)
print (a*b)
print (a/b)
print (a%%b)
print (a%/%b)
print (a^b)
```

- **Output:**

```
[1] 9.5

[1] 5.5

[1] 15

[1] 3.75

[1] 1.5

[1] 3

[1] 56.25
```

➢ **Relational Operators:**

- **Code:**

```
a <- 7.5
b <- 2

print (ab)
print (a==b)
print (a<=b)
print (a>=b)
print (a! =b)
```

- **Output:**

```
[1] FALSE

[1] TRUE

[1] FALSE

[1] FALSE

[1] TRUE

[1] TRUE
```

➢ **Logical Operators:**

- **Code:**

```
a <- 0
b <- 2

print (a & b)
print (a | b)
print (! a)
print (a && b)
print (a || b)
```

- **Output:**

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

➢ Assignment Operators:

- **Code:**

```
a = 2
print ( a )

a <- TRUE
print ( a )


454 -> a
print ( a )

a <<- 2.9
print ( a )

c(6, 8, 9) -> a
print ( a )
```

- **Output:**

```
[1] 2
[1] TRUE
[1] 454
[1] 2.9
[1] 6 8 9
```

➢ **Miscellaneous Operators:**

- **Code:**

```
a = 23:31
print ( a )

a = c(25, 27, 76)
b = 27
print ( b %in% a )

M = matrix(c(1,2,3,4), 2, 2, TRUE)
print ( M %*% t(M) )
```

- **Output:**

```
[1] 23 24 25 26 27 28 29 30 31

[1] TRUE

        [,1]   [,2]

[1,]    5      11

[2,]   11      25
```

❖ **OBSERVATIONS:**

We done Data Operators in R programming– different R data operators, their syntax and example of R commands for R data operators.

❖ **Learning outcomes (What I have learnt):**

1. How to use different data operators in R Programming i.e. Arithmetic, Relational, Logical, Assignment, Miscellaneous operators.

❖ **<u>Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):</u>**

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			

PRACTICAL-4

Data Structures in R Programming

Student Name: Jatin Verma	**System ID:** 2021370138
Branch: BSc Data Science & Analytics	**Section/Group:** A
Semester: II	**Date of Performance:** 05/04/22
Subject Name: R-Programming	**Subject Code:** BDA 154

❖ **AIM:**

To understand data structures in R programming.

❖ **THEORY:**

A data structure is a particular way of organizing data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks. Data structures are the objects that are manipulated regularly in R. They are used to store data in an organized fashion to make data manipulation and other data operations more efficient.

➢ **R has many data structures, which include:**

1) **Vectors:**

   Vectors are one-dimensional data structures. It is homogenous, which means that it only contains elements of the same data type. Data types can be numeric, integer, character, complex, or logical. Vectors are created by using the **c()** function. The **typeof()** function is used to check the data type of the vector, and the **class()** function is used to check the class of the vector.

2) **Lists:**

   A list is a non-homogeneous data structure, which implies that it can contain elements of different data types. It accepts numbers, characters, lists, and even matrices and functions inside it. It is created by using the list() function. These are also one-dimensional data structures.

3) **Matrices:**

Matrices are two-dimensional data structures that is homogeneous, meaning that it only accepts elements of the same data type. A matrix is a rectangular arrangement of numbers in rows and columns. In a matrix, a column is a vertical representation of data, while a row is a horizontal representation of data. A matrix is created with the help of the matrix() function in R. Specify the **nrow** and **ncol** parameters to get the amount of rows and columns.

4) **Arrays:**

Arrays are the R data objects which store the data in more than two dimensions. Arrays are n-dimensional data structures**.** An array is a collection of a similar data type with contiguous memory allocation. In R, an array is created with the help of **array()** function and the dim parameter to specify the dimensions.

5) **Factors:**

Factors are also data objects that are used to categorize the data and store it as levels. Factors can store both strings and integers. They are useful to categorize unique values in columns like "TRUE" or "FALSE", or "MALE" or "FEMALE", etc. They are useful in data analysis for statistical modelling. Factors are created with the help of **factor()** function by taking a vector as an input parameter.

6) **Data frames:**

Data frame is a two-dimensional array-like structure that also resembles a table, in which each column contains values of one variable and each row contains one set of values from each column. To create a data frame, we use the **data.frame()** function.

A data frame has the following characteristics:

- The column names of a data frame should not be empty.
- The row names of a data frame should be unique.
- The data stored in a data frame can be a numeric, factor, or character type.
- Each column should contain the same number of data items.

❖ **<u>PROGRAM AND OUTPUT</u>:**

➢ **<u>Vectors</u>:**

- **<u>Code:</u>**

```
> X = c (1, 3, 5, 7, 8)
> print(X)
```

- **<u>Output:</u>**

```
[1] 1 3 5 7 8
```

➢ **<u>Lists</u>:**

- **<u>Code:</u>**

```
list1= list (“Sam”, “Green”, c (8,2,67), TRUE, 67.56, FALSE)
```

- **<u>Output:</u>**

```
[[1]]
[1] “Sam”

[[2]]
[1] “Green”

[[3]]
[1] 8 2 67

[[4]]
[1] TRUE

[[5]]
[1] 67.56

[[6]]
[1] FALSE
```

➢ **Matrices:**

- **Code:**

```
> Matrix1 <- matrix (c (1,2,3,4,5,6), nrow = 3, ncol = 2)
> Print(Matrix1)
```

- **Output:**

```
      [,1]  [,2]
[1,]    1     4
[2,]    2     5
[3,]    3     6
```

➢ **Arrays:**

- **Code:**

```
> A = array (c (1, 2, 3, 4, 5, 6, 7, 8), dim = c (2, 2, 2))
> print(A)
```

- **Output:**

```
, , 1

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

, , 2

     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

➢ Factors:

- **Code:**

```
> fac = factor(c("Male", "Female", "Male",
              "Male", "Female", "Male", "Female"))

> print(fac)
```

- **Output:**

```
[1] Male   Female Male   Male   Female Male   Female
Levels: Female Male
```

➢ Dataframes:

- **Code:**

```
empid <- c(1:4)
empname <- c("Sam","Rob","Max","John")
empdept <- c("Sales","Marketing","HR","R & D")
emp.data <- data.frame(empid,empname,empdept)
print(emp.data)
```

- **Output:**

```
   empid   empname    empdept
1    1        Sam        Sales
2    2        Rob      Marketing
3    3        Max          HR
4    4        John       R & D
```

❖ **OBSERVATIONS:**

We observe Data Structures in R programming– different R data structures, their syntax and example of R commands for R data structures.

❖ **Learning outcomes (What I have learnt):**

1. How to use different data structures in R Programming i.e. Vectors, Matrices, Array, Factors, Dataframes.

❖ **<u>Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):</u>**

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			

PRACTICAL-5

Control Flow Statements in R Programming-

Decision Making and Loops

Student Name: Jatin Verma	**System ID:** 2021370138
Branch: BSc Data Science & Analytics	**Section/Group:** A
Semester: II	**Date of Performance:** 12/04/22
Subject Name: R-Programming	**Subject Code:** BDA 154

❖ **AIM:**

To understand Control Flow Statements in R programming- Decision Making and Loops.

❖ **THEORY:**

▪ **Decision Making**

Decision making structures are used by programmers to specify one or more conditions to be evaluated or tested by the program. It allows us to make a decision, based on the result of a condition. Decision making is involved in order to change the sequence of the execution of statements, depending upon certain conditions. A set of statements is provided to the program, along with a condition. Statements get executed only if the condition stands true, and, optionally, an alternate set of statements is executed if the condition becomes false.

♦ **The decision making statement in R are as followed:**

1) **if Statement:**

The if statement consists of the Boolean expressions followed by one or more statements. The if statement is the simplest decision-making statement which helps us to take a decision on the basis of the condition. If the Boolean expression evaluates to TRUE, the set of statements is executed. If the Boolean expression evaluates to FALSE, the statements after the end of the If statement are executed.

The basic syntax for the If statement is given below:

```
if(Boolean_expression) {
   This block of code will execute if the Boolean expression returns TRUE.
```

- **Example Program:**

```
x = 10
if (x> 0){
  cat(x, "is a positive number")
}
```

- **Output:**

```
10 is a positive number
```

2) if-else Statement:

An if-else statement is the if statement followed by an else statement. If-else, provides us with an optional else block which gets executed if the condition for if block is false. If the condition provided to if block is true then the statement within the if block gets executed, else the statement within the else block gets executed.

The basic syntax of If-else statement is as follows:

```
if(boolean_expression) {
   // statement(s) will be executed if the boolean expression is true.
} else {
   // statement(s) will be executed if the boolean expression is false.
}
```

- **Example Program:**

```
x =-5
if (x > 0){
   cat (x, "is a positive number\n")
}else {
   cat (x, "is a negative number\n")
}
```

- **Output:**

```
-5 is a negative number
```

3) Nested if-else Statement:

This statement is also known as nested if-else statement. The if statement is followed by an optional else if..... else statement. This statement is used to test various condition in a single if......else if statement. There are some key points which are necessary to keep in mind when we are using the if.....else if.....else statement. These points are as follows:

1. **if** statement can have either zero or one **else** statement and it must come after any **else if’s** statement.
2. **if** statement can have many **else if's** statement and they come before the else statement.
3. Once an **else if** statement succeeds, none of the remaining **else if's** or **else's** will be tested.

The basic syntax of If-else statement is as follows:

```
if(boolean_expression 1) {
  // This block executes when the boolean expression 1 is true.
} else if( boolean_expression 2) {
  // This block executes when the boolean expression 2 is true.
} else if( boolean_expression 3) {
  // This block executes when the boolean expression 3 is true.
} else {
  // This block executes when none of the above condition is true.
}
```

➢ **Example Program:**

```
x =19
if (x < 0){
  cat (x, "is a negative number")
}else if (x > 0){
  cat (x, "is a positive number")
}else{
  print ("Zero")
}
```

➢ **Output:**

```
19 is a positive number
```

4) **switch Statement:**

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case. The basic syntax for creating a switch statement in R is as follows.
switch (expression, case1, case2, case3. .. .)

- **Example Program:**

```
switch (2, "red", "green", "blue")

switch ("color", "color" = "red", "shape" = "square", "length" = 5)
```

- **Output:**

```
[1]  “green”
[1]   “red”
```

- **Loops**

The function of a looping statement is to execute a block of code, several times and to provide various control structures that allow for more complicated execution paths than a usual sequential execution.

- **The types of loops in R are as follows:**

1) **For Loop:**

For loop is one of the control statements in R programming that executes a set of statements in a loop for a specific number of times, as per the vector provided to it.

The basic syntax of a for loop is given below:

```
for (value in vector) {
statements
}
```

- **Example Program:**

```
x = c(2,5,3,9,8,11,6)
count=0
for (val in x){
  if (val %% 2 == 0)
```

```
    count = count+1
 }
cat("No. of even numbers in", x, "is", count)
```

- **Output:**

```
No. of even numbers in 2 5 3 9 8 11 6 is 3
```

2) **While Loop:**

A while loop is one of the control statements in R programming which executes a set of statements in a loop until the condition (the Boolean expression) evaluates to TRUE. In while loop, firstly the condition will be checked and then after the body of the statement will execute. In this statement, the condition will be checked n+1 time, rather than n times.

The basic syntax of while loop is as follows:

```
while (Boolean_expression) {
   statement
}
```

- **Example Program:**

```
num= 5
sum = 0
   while (num > 0){
    sum= sum + num
    num = num-1
 }
cat ("The sum is", sum)
```

- **Output:**

```
The sum is 15
```

- **Loop Control Statements**

Loop control statements are also known as jump statements. Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. The loop control statements in R are break statement and next statement.

1) **break Statement:**

A break statement is used inside a loop (repeat, for, while) to stop the iterations and flow the control outside of the loop. In a nested looping situation, where there is a loop inside another loop, this statement exits from the innermost loop that is being evaluated.

- **Example Program:**

```
x =1:10
for (val in x){
   if (val == 3) {
      break
   }
    print (val)
}
```

- **Output:**

```
[1]  1
[1]  2
```

2) **next Statement:**

A next statement is useful when we want to skip the current iteration of a loop without terminating .On encountering next, the R parser skips further evaluation and starts next iteration of the loop. This is equivalent to the continue statement in C.

- **Example Program:**

```
x =1:10
for (val in x){
   if (val == 3) {
      next
   }
    print (val)
}
```

➢ **Output:**

```
[1]  1
[1]  2
[1]  4
[1]  5
[1]  6
[1]  7
[1]  8
[1]  9
[1]  10
```

❖ OBSERVATIONS:

We observe control flow statements in R programming, their syntax, use and example of R commands for R decision making statements and loops.

❖ Learning outcomes (What I have learnt):

1. How to use different control flow statements in R Programming i.e. if, if-else, nested if-else, switch statements, for loop, while loop, break statement, next statement.

- **<u>Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):</u>**

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			

PRACTICAL-6

Mathematical Functions in R Programming

Student Name: Jatin Verma
Branch: BSc Data Science & Analytics
Semester: II
Subject Name: R-Programming

System ID: 2021370138
Section/Group: A
Date of Performance: 19/04/22
Subject Code: BDA 154

❖ AIM:

To understand Mathematical Functions in R programming.

❖ THEORY:

R provides the various mathematical functions to perform the mathematical calculation. These mathematical functions are very helpful to find absolute value, which are used:

1) abs ( )

This function computes the absolute value of numeric data. The syntax is abs(x), where x is any numeric value, array or vector.

➢ **Example Program**

```
x<- -4
print(abs(x))
```

➢ **Output**

```
[1]  4
```

2) sin ( ), cos ( ) and tan ( )

The function sin () computes the sin value, cos () computes the cosine value and tan () computes the tangent value of numeric data in radians. The syntax in sin (x) , cos (x) and tan (x) where x is any numeric value, array or vector.

➢ **Example Program**

```
> sin(10)
> cos(90)
> tan(50)
```

- **Output**

```
[1]  -0.5440211
[1]  -0.4480736
[1]  -0.2719006
```

3) **asin ( ), acos ( ) and atan ( )**

The function asin () inverse the sin value, acos () computes the cosine value and atan () computes the tangent value of numeric data in radians. The syntax in sin (x), cos (x) and tan (x) where x is any numeric value, array or vector.

- **Example Program**

```
> asin(1)
> acos(1)
> atan(50)
```

- **Output**

```
[1]  1.570796
[1]   0
[1]  1.550799
```

4) **exp (x)**

This function computes the exponential value of a number or number vector, e^x.

- **Example Program**

```
> x=5
> exp (x)
```

- **Output**

```
[1]  148.4132
```

5) **ceiling ( )**

This function returns the smallest integer larger than the parameter. The syntax is ceiling (x) where x is a numeric variable or vector.

- **Example Program**

```
> x=2.5
> ceiling (x)
```

- **Output**

```
[1]  3
```

6) **floor ( )**
   This function returns the largest integer not greater than the giving number. The syntax is floor (x) where x is a numeric variable or vector.

- **Example Program**

```
> x=2.5
> floor (x)
```

- **Output**

```
[1]  2
```

7) **round ( )**
   This function returns the integer rounded to the giving number. The syntax is round(x, digits=n) where x is a numeric variable or vector and digit specifies the number of digits to be rounded.

- **Example Program**

```
> x=2.5878888
> round (x,3)
```

- **Output**

```
[1]  2.588
```

8) **trunc ()**
   This function returns the integer truncated with decimal part. The syntax is trunc (x) where x is a numeric variable or vector.

- **Example Program**

```
> x=2.99
> trunc (x)
```

- **Output**

```
[1]  2
```

9) signif (x,digits=n)

This function rounds the value in its first argument to the specified number of significant digits. The syntax is signif (x, digits=n) where x is numeric variable or a vector and digit specifies the number of significant digits.

- **Example Program**

```
> x=2.5878888
> signif (x,3)
```

- **Output**

```
[1]  2.59
```

10) sqrt ()

This function computes the square root of a numeric vector. The syntax is sqrt (x) where x is numeric, complex vector or array.

- **Example Program**

```
> x=25
> sqrt (x)
```

- **Output**

```
[1]  5
```

11) log ( ), log10 ( ), log2 ( ), log(x,b)

log () function computes natural logarithms for a number or vector. log10 () computes common logarithms with base 10. 1og2 () computes binary logarithms or logarithms with base 2. log (x, b) computes logarithms with base b.

- **Example Program**

```
> log (5)
> log10 (5)
> log2 (5)
```

- **Output**

```
[1]  1.609438
[1]  0.69897
[1]  2.321928
```

12) **max ( ) and min ( )**

max () function computes the maximum value of a vector and min () function computes the minimum value of a vector.

- **Example Program**

```
> x= c(10,289,-100, 8000)
> max (x)
> min (x)
```

- **Output**

```
[1]  8000
[1]  -100
```

13) **beta ( ) and l beta ( )**

beta () function returns the beta value and lbeta () function returns the natural logarithm of the beta function. Beta function is calculated as follows.

B (a,b) = ɼ(a) ɼ(b)/ɼ(a+b)

The syntax is beta (a, b) and 1beta (a,b) where non-negative a, b are numeric vectors.

- **Example Program**

```
> x= c(10,289,-100, 8000)
> max (x)
> min (x)
```

- **Output**

```
[1]  8000
[1]  -100
```

14) **gamma ( )**

This function returns the gamma function ɼx. The syntax is gamma (x) where x is a numeric vector.

- **Example Program**

```
> x= 5
> gamma (x)
```

- **Output**

```
[1]  24
```

15) **factorial ( )**
This function computes factorial of a number or a numeric vector. The syntax is factorial (x) where x is a number or numeric vector.

- **Example Program**

```
> x= 5
> factorial (x)
```

- **Output**

```
[1]  120
```

❖ OBSERVATIONS:

We observe different statistical functions, their syntax and example of R commands for Statistical functions.

❖ Learning outcomes (What I have learnt):

1. How to use different mathematical functions in R Programming i.e. abs(), max(), min(), sqrt(), sin(), cos(), tan(), round(), floor(), ceiling().

❖ **<u>Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):</u>**

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			

PRACTICAL-7

Statistical Functions in R Programming

Student Name: Jatin Verma	**System ID:** 2021370138
Branch: BSc Data Science & Analytics	**Section/Group:** A
Semester: II	**Date of Performance:** 26/04/22
Subject Name: R-Programming	**Subject Code:** BDA 154

❖ **AIM:**

To understand Statistical Functions in R programming.

❖ **THEORY:**

Statistical analysis in R is performed by using many in-built functions. Most of these functions are part of the R base package: These functions take R vector as an input along with the arguments and give the result. Some of the basic and frequently used statistical functions are:

1) **mean ()**

It is calculated by taking the sum of the values and dividing with the number of values in a data series. The function mean () is used to calculate average or mean in R. The syntax tor calculating mean is mean (x, trim=0, na.rm=FALSE) where x is the input vector, trim is used to drop some observations from both end of the sorted vector and na.rm is used to remove the missing values from the input vector.

➢ **Example Program**

```
x<-c(10,2,30,4, 5,6,70, 8,9,10)
y<-c(1,2,3,4,5, NA)
mean (x)
mean (x, trim=0.2)
mean (y, na.rm=TRUE)
```

➢ **Output**

```
[1]  15.4
[1]  8
[1]  3
```

- In the above example, when the trim attribute is given as 0.2, it will remove 2 elements from both ends and mean will be calculated using the rest of the elements. If there are missing values, then the mean function returns NA. To drop the missing values from the calculation, use na.rm = TRUE which remove the NA values.

2) <u>median ()</u>

The middle most value in a data series is called the median. The median () function is used in R to calculate this value. The syntax for calculating median in R is median (x, na.rm= FALSE) , where x is the input vector, na.rm is used to remove the missing values from the input vector.

- **<u>Example Program</u>**

```
x<- c(10,2,30, 2,5,8, 70,8,9,2)
y<-c (3,2,3,4,5, NA)
median (x)
median (y, na.rm=TRUE)
```

- **<u>Output</u>**

```
[1]  8
[1]  3
```

3) <u>var ()</u>

The function var () returns the estimated variance of the population from which the numbers in vector x are sampled. The syntax tor calculating variance in R is var (x, na.rm = FALSE) , where x is the input vector, na.rm is used to remove the missing values from the input vector.

- **<u>Example Program</u>**

```
x<-c(10,2, 30, 2,5, 8, 70, 8,9,2)
y<-c (3,2,3,4,5,NA)
var (x)
var (y, na.rm=TRUE)
```

- **<u>Output</u>**

```
[1]  446.0444
[1]  1.3
```

4) <u>sd ()</u>

The function sd () returns the estimated standard deviation of the population from which the numbers in vector x are sampled. The syntax for calculating standard deviation in R is sd (x, na.rm= FALSE) , where x is the input vector, na.rm is used to remove the missing values from the input vector.

- **<u>Example Program</u>**

```
x<-c (10,2,30,2,5, 8,70,8,9,2)
y<-c(3, 2,3, 4,5, NA)
sd (x)
sd (y, na.rm=TRUE)
```

- **<u>Output</u>**

```
[1] 21.11976
[1] 1.140175
```

5) <u>scale ()</u>

The function scale (x, center=TRUE, scale=TRUE) returns the standard scores (z-scores) for the numbers in vector x. This is used for standardizing a matrix.

- **<u>Example Program</u>**

```
x<-matrix (1:9, 3,3)
x
scale (x)
```

- **<u>Output</u>**

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

# Scale

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

```
[3,]    1    1    1
attr(,"scaled:center")
[1] 2 5 8
attr(,"scaled:scale")
[1] 1 1 1
```

6) <u>sum ()</u>

This function adds up all elements of a vector. The syntax is sum (x), where x is a numeric vector.

- **<u>Example Program</u>**

```
sum (c (1:10))
sum (c (1,2,3,4, 5))
```

- **<u>Output</u>**

```
[1]  55
[1]  15
```

7) <u>diff ()</u>

This function returns suitably lagged and iterated differences. The syntax is diff (x, lag, differences) where x is a numeric vector or matrix containing the is diff (x, values to be differenced, lag is an integer indicating which lag to use and differences is a integer indicating the order of the difference. For example, if lag = 2, the differences between the third and the first value, between the fourth and the second value, between the fifth and the third value etc. are calculated. The attribute differences returns the differences of differences.

- **<u>Example Program</u>**

```
diff (c (5, 20,14, 4,2,10, 19) )
 diff (c (5,20,14, 4,2,10,19),lag=2)
 diff (c (5, 20,14, 4,2,10,19), differences=2)
```

- **<u>Output</u>**

```
[1]  15  -6  -10  -2  8  9
[1]  9  -16  -12  6  17
[1]  -21  -4  8   10  1
```

8) **range ()**

This function returns a vector of the minimum and maximum values. The syntax range (x, na.rm = FALSE, finite FALSE), where x is a numeric vector, na.rm is the attribute to decide whether NA should be removed and finite determines whether non-finite elements should be omitted.

- **Example Program**

```
x<-c (10,2,30,4,5,6, 70, 8, 9,10)
y<-c (1,2,3,4,5, NA)
range (x)
range (y, na.rm=FALSE)
```

- **Output**

```
[1]  2   70
[1]  NA  NA
```

9) **quantile ()**

A quantile, or percentile, tells how much of our data lies below a certain value. The 50 percent quantile, for example, is the same as the median. The syntax for quantile () function is quantile (x, probs) where x is the numeric vector whose quantiles are desired and probs is a numeric vector with probabilities in range
[0, 1].

- **Example Program**

```
x<-c (1,2,3,4,5,6,7,8,9,10)
quantile (x, c (0.03,0.5))
```

- **Output**

```
 3%   50%
1.27  5.50
```

- In the above example, the third quartile and 50th quartile is calculated.

10) rank ()

This function returns the ranks of the numbers (in increasing order) in vector x. The syntax of the function is rank (x, na.last = TRUE, ties.method = c(“average”, first", "last" random", "max", "min")), where x is a numeric, complex, character or logical vector, na.last is for controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed; if keep" they are kept with rank NA. The attribute ties.method is a character string specifying how ties are treated. If x is given as -x, the function returns the ranks of the numbers in decreasing order in vector x.

➢ **Example Program**

```
x <- c(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5)
x
#Rank in ascending order
rank (x)

#Rank in descending order
rank (-x)

#Tie is broken with average
rank (x, ties.method="average")

#Tie is broken with first element
rank (x, ties.method="first")

#Tie is broken with last element
rank (x, ties.method="last")

#Tie is broken with random element
rank (x, ties .method="random ")

#Tie is broken with largest element
rank (x, ties.method="max")

#Tie is broken with smallest element
rank (x, ties.method="min")
```

➢ **Output**

```
[1]  3 1 4 1 5 9 2 6 5 3 5
[1]  4.5 1.5 6.0 1.5 8.0 11.0 3.0 10.0 8.0 4.5 8.0
[1]  7.5 10.5 6.0 10.5 4.0 1.0 9.0 2.0 4.0 7.5 4.0
[1]  4.5 1.5 6.0 1.5 8.0 11.0 3.0 10.0 8.0 4.5 8.0
[1]  4 1 6 27 11 3 10 8 5 9
[1]  5 26 19 11 3 10 8 4 7
[1]  5 1 6 2 8 11 3 10 7 4 9
[1]  5 2 6 2 9 11 3 10 9 5 9
[1]  4 1 6 17 11 3 10 7 4 7
```

❖ OBSERVATIONS:

We observe different statistical functions, their syntax and example of R commands for Statistical functions.

❖ Learning outcomes (What I have learnt):

1. How to use different Statistical functions in R Programming i.e. to find mean, median, variance, standard deviation, rank, quantile, range, sum, difference.

- **<u>Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):</u>**

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			

PRACTICAL-8

Charts in R Programming

Student Name: Jatin Verma
Branch: BSc Data Science & Analytics
Semester: II
Subject Name: R-Programming

System ID: 2021370138
Section/Group: A
Date of Performance: 10/05/22
Subject Code: BDA 154

❖ AIM:

To understand Charts in R programming.

❖ THEORY:

➢ plot ( ) function:

The plot() function is used to plot R objects.

The basic syntax for the **plot()** function is given below:

plot(x, type, main, xlab, ylab, col)

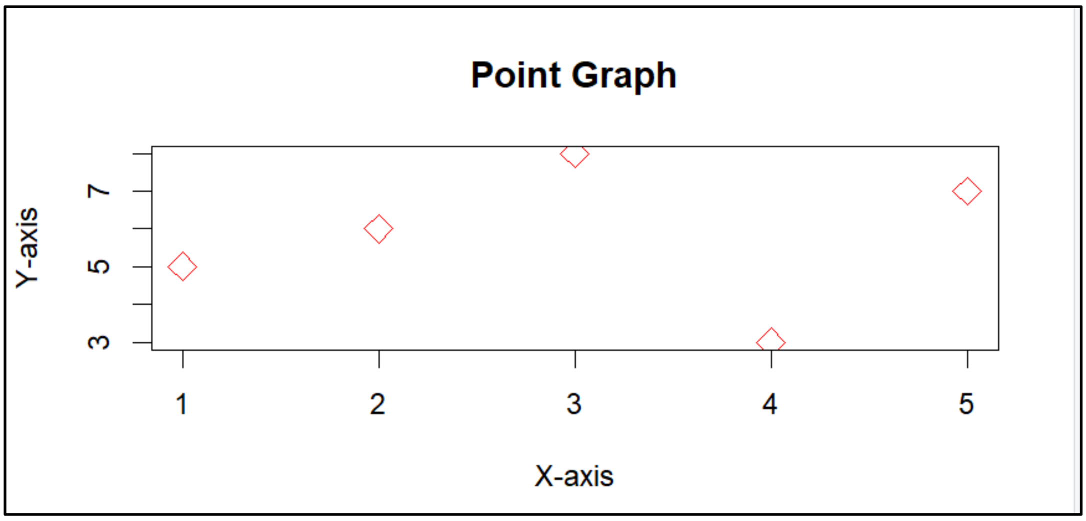
Parameters:

- **x:** This parameter is a contains only the numeric values.
- **xlab:** This parameter is the label for x axis in the chart.
- **ylab:** This parameter is the label for y axis in the chart.
- **main:** This parameter main is the title of the chart.
- **col:** This parameter is used to give colors to both the points and lines.
- **cex:** This parameter is used to change the size of the points.
- **pch:** Use pch with a value from 0 to 25 to change the point shape format.

➢ **Example Program:**

```
x=c(5,6,8,3,7)
plot(x, col="Red", main="Line Graph", xlab="X-axis", ylab="Y-axis"
,cex=1.5,pch=5)
```

➢ **Output:**



1) **Line Graph:**
A line graph is a chart that is used to display information in the form of a series of data points. Line graphs are drawn by plotting different points on their X coordinates and Y coordinates, then by joining them together through a line from beginning to end. Within a line graph, there are points connecting the data to show the continuous change. The lines in a line graph can move up and down based on the data. We can use a line graph to compare different events, information, and situations. Line charts are used in identifying the trends in data.

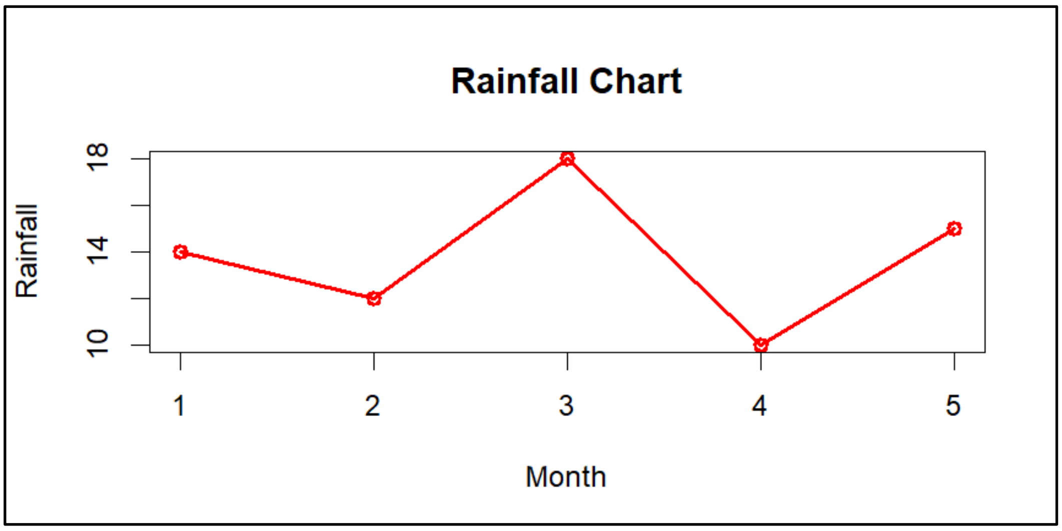
➢ **Simple Line Graph:**
The plot() function in R is used to create the line graph.
Syntax: *plot(x, type, col, xlab, ylab)*

S.No	Parameter	Description
1.	x and y	It is the x and y coordinates of points in the plot.
2.	type	This parameter takes the value "I" to draw only the lines or "p" to draw only the points and "o" to draw both lines and points.
3.	xlab	It is the label for the x-axis.
4.	ylab	It is the label for the y-axis.
5.	main	It is the title of the chart.
6.	col	It is used to give the color for both the points and lines
7.	lwd	This parameter is used to change line width.
8.	lty	This parameter is used with a value from 0 to 6 to specify the line format.

➢ **Example Program:**

```
x=c(1,2,3,4,5)
y=c(14,12,18,10,15)
plot(x, y, type="o", col="Red", main="Rainfall Chart", xlab="Month",
ylab="Rainfall", lwd=2)
```

➢ **Output:**



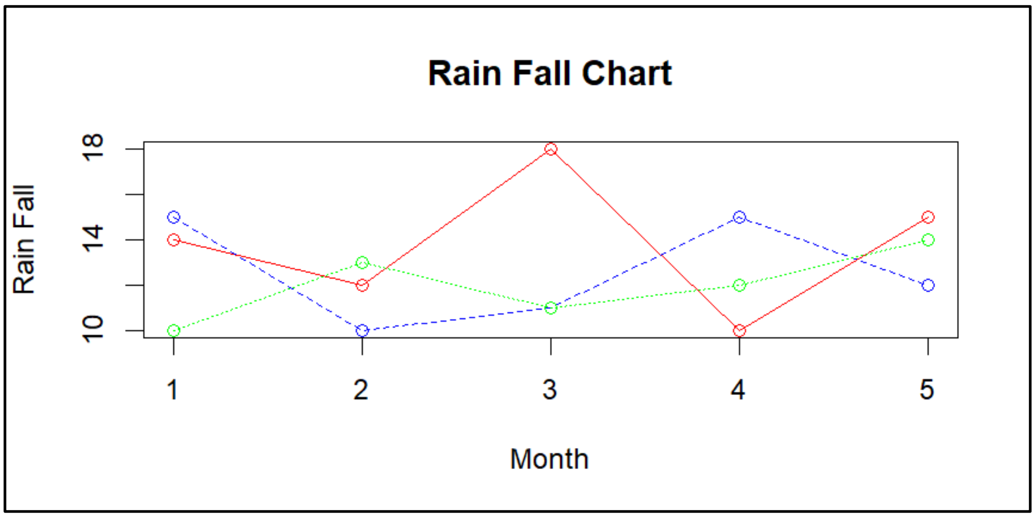
➢ **<u>Multiple Line Graph:</u>**

For drawing line graphs with multiple lines, we can use the function lines (x, y, type=) where x and y are numeric vectors of (x,y) points to connect.

➢ **<u>Example Program:</u>**

```
x=c(1,2,3,4,5)
y=c(14,12,18,10,15)
z=c(10,13,11,12,14)
k=c(15,10,11,15,12)
plot (x, y, type="o", col="red", xlab="Month", ylab="Rain Fall",
 main="Rain Fall Chart")
lines (x, k, type="o" , col="blue", lty=2)
lines (x, z, type="o", col ="green", lty=3)
```

➢ **<u>Output:</u>**



2) <u>Scatter Plot:</u>

A scatter plot is a set of dotted points to represent individual pieces of data in the horizontal and vertical axis. In a scatterplot, the data is represented as a collection of points. Each point on the scatterplot defines the values of the two variables. One variable is selected for the vertical axis and other for the horizontal axis. The pattern of the resulting points reveals a correlation between them.

A scatter plot can suggest various kinds of correlations between variables with a certain confidence interval. For example, weight and height, weight would be on y-axis and height would be on the x-axis, Correlations may be positive (rising), negative (falling), or null (uncorrelated). If the pattern of dots slopes from lower left to upper right, it indicates a positive correlation between the variables being studied. If the pattern of dots slopes from upper left to lower right, it indicates a negative correlation.

- We can create a scatter plot in R Programming Language using the **plot()** function.
  Syntax: *plot(x, y, main, xlab, ylab, xlim, ylim, axes)*

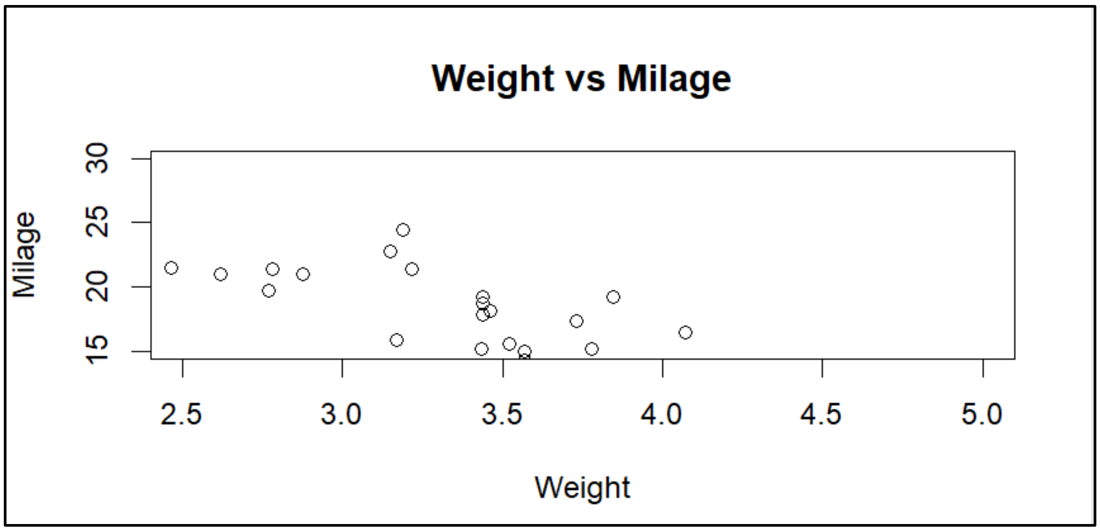
S.No	Parameters	Description
1.	x	It is the dataset whose values are the horizontal coordinates.
2.	y	It is the dataset whose values are the vertical coordinates.
3.	main	It is the title of the graph.
4.	xlab	It is the label on the horizontal axis.
5.	ylab	It is the label on the vertical axis.
6.	xlim	It is the limits of the x values which is used for plotting.
7.	ylim	It is the limits of the values of y, which is used for plotting.
8.	axes	It indicates whether both axes should be drawn on the plot

- **Simple Scatter Plot:**
  We use the data set “mtcars” available in the R environment to create a basic scatterplot. Let’s use the columns “wt” and “mpg” in mtcars. The below script will create a scatterplot graph for the relation between wt (weight) and mpg(miles per gallon).

- **Example Program:**

```
#Get the input values.
input = mtcars[c('wt', 'mpg')]
# Plot the chart for cars with weight between 2.5 to 5 and mileage between
15 and 30.
plot (x = input$wt,y = input$mpg,
     xlab = "Weight",
     ylab = "Mileage",
     xlim = c(2.5,5),
     ylim = c(15, 30),
     main = "Weight vs Mileage" )
```

- **Output:**

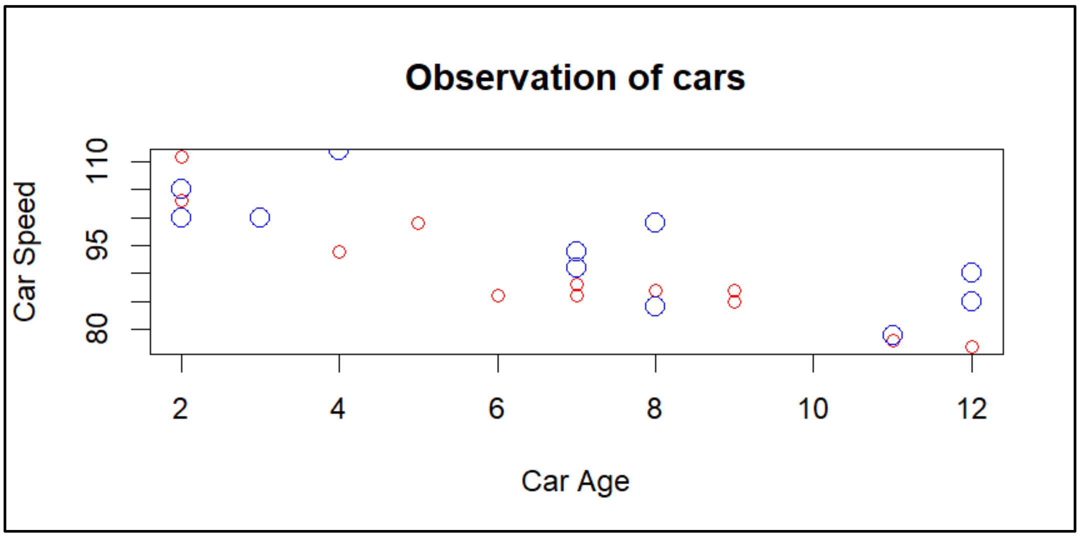


- **Multiple Scatter Plot:**
  We can also add more data to your original plot with the **points()** function, that will add the new points over the previous plot, respecting the original scale.

- **Example Program:**

```
# Day1,the age and speed of 12 cars:
x1=c(5,7,8,7,2,2,9,4,11,12,9,6)
y1=c(99,86,87,88,111,103,87,94,78,77,85,86)
# Day2,the age and speed of 15 cars:
x2=c(2,2,8,1,15,8,12,19,7,3,11,4,7,14,12)
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=1)
points(x2,y2,col="blue", cex=1.5)
```

➢ **Output:**



3) **Pie Charts:**

A pie chart is a representation of values as slices of a circle with different colors. The slices are labelled and the numbers corresponding to each slice is also represented in the chart. In R, the pie chart is created using the pie() function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc. The basic syntax for drawing a pie chart is as follows.

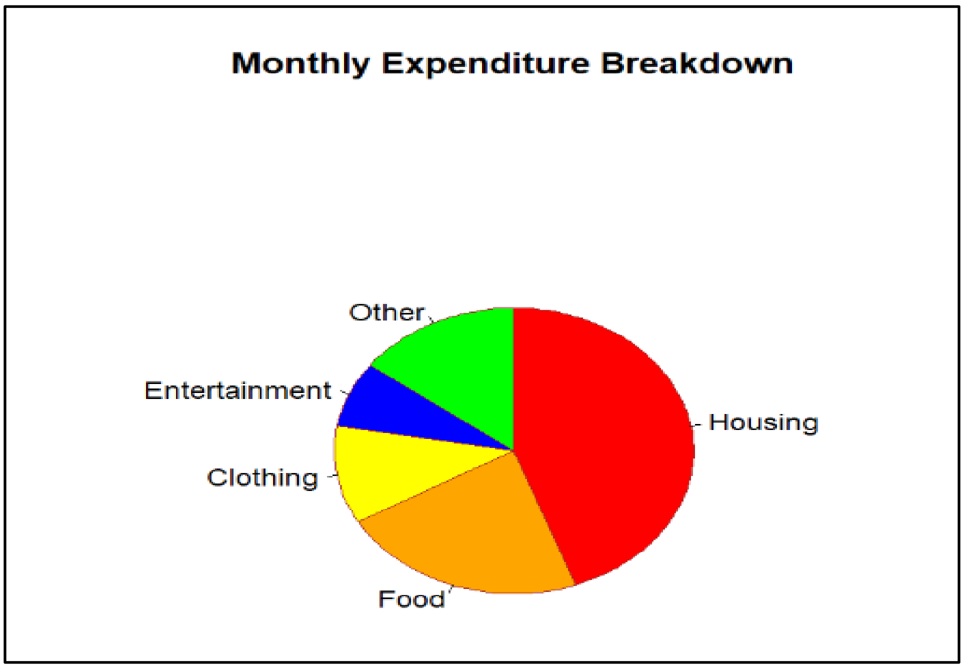
Syntax: *pie(x, labels, radius, main, col, clockwise)*

S.No	Parameters	Description
1.	x	It is a vector that contains the numeric values which are used in the pie chart.
2.	labels	It gives the description to the slices in pie chart.
3.	main	It represents title of the pie chart.
4.	radius	It is used to indicate the radius of the circle of the pie chart.(value between -1 and +1).
5.	col	It give colors to the pie in the graph.
6.	clockwise	It contains the logical value which indicates whether the slices are drawn clockwise or in anti-clockwise direction.

➢ **Example Program:**

```
expenditure=c (600, 300, 150, 100, 200)
pie (expenditure,
labels=c(“Housing”, " "Food” ,”Clothing”, ”Entertainment”, “Other”),
main="Monthly Expenditure Breakdown”,
col=c(“red”, “orange”,” yellow”, ”blue”, ”green”),
border="brown", clockwise=TRUE)
```

➢ **Output:**



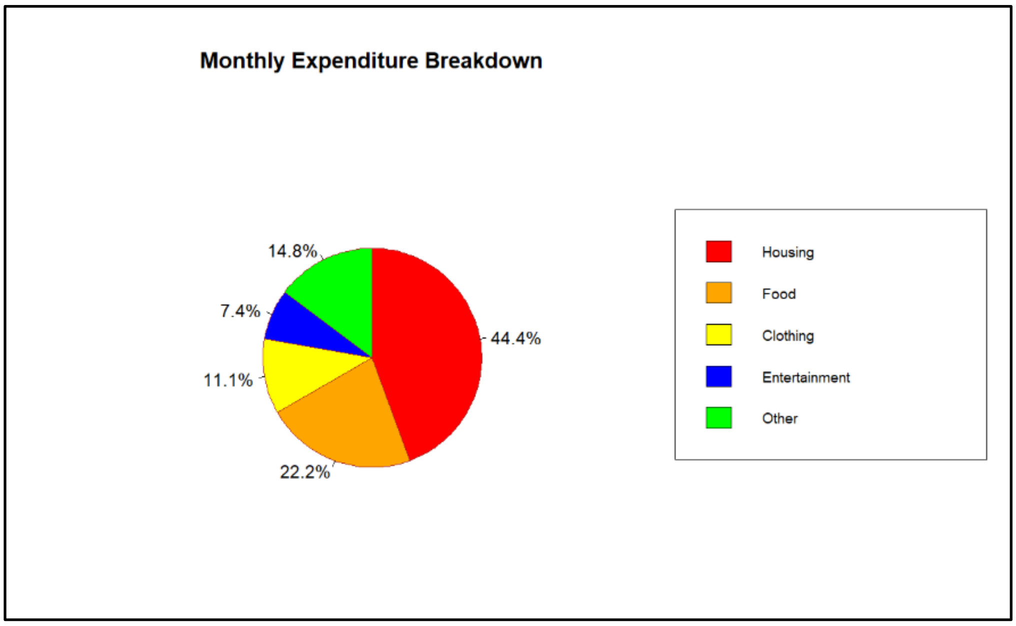
❖ Pie Charts with Slice Percentages and Chart Legend:

There are two additional properties of the pie chart, i.e., slice percentage and chart legend. We can show the data in the form of percentage as well as we can add legends to plots in R by using the legend() function.

- **Example Program:**

```
expenditure=c (600, 300, 150, 100, 200)
#Calculating the percentage
piepercent<- round (100*expenditure/sum (expenditure) , 1)
#Adding percentage Symbol to each value
piepercent <- paste (piepercent, "%", sep="")
#Plotting the Pie Graph
pie (expenditure, labels=piepercent, main="Monthly Expenditure Breakdown",
col=c("red", "orange", "yellow", "blue", "green"), border="brown",
clockwise=TRUE)
#Legend for the Pie Graph
legend ("topright", c("Housing", "Food", "Clothing", "Entertainment", "Other")
,cex = 0.8, fill=c ("red", "orange", "yellow", "blue", "green"))
```

- **Output:**



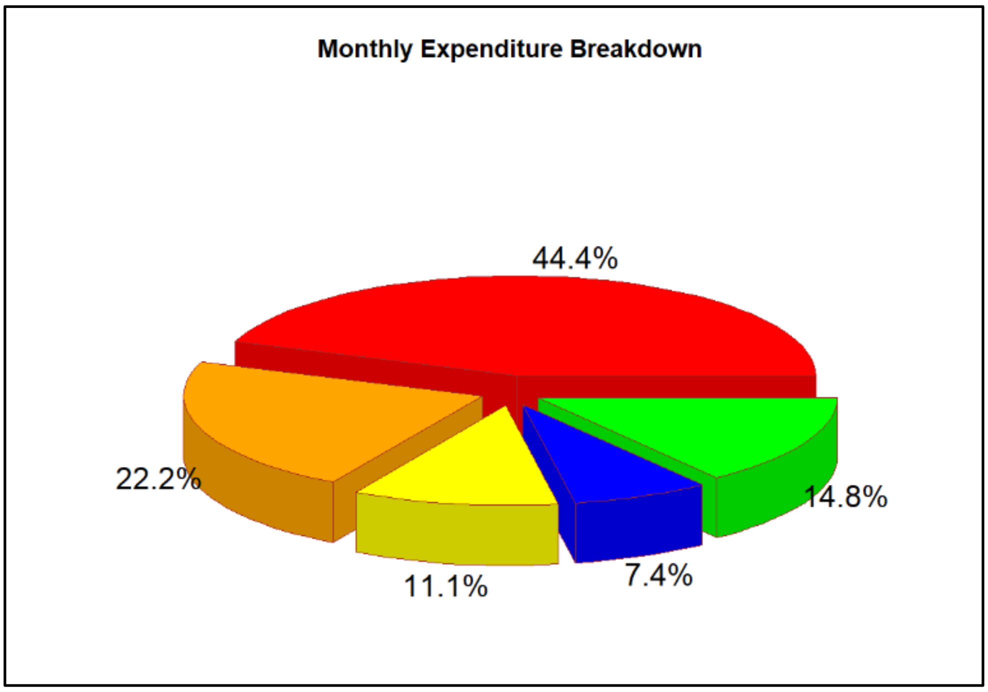
❖ 3D Pie Charts:

A pie chart with three dimensions is called a 3D pie chart. The pie3D( ) function in the plotrix package provides 3D exploded pie charts.

➢ Example Program:

```
#Include the library
library(plotrix)
expenditure=c (600, 300, 150, 100, 200)
#Calculating the percentage
piepercent<- round (100*expenditure/sum (expenditure) ,1)
#Adding percentage Symbol to each value
piepercent <- paste (piepercent, "%", sep="")
#Plotting the Pie Graph
pie3D(expenditure, labels=piepercent, explode=0.1, main="Monthly
Expenditure Breakdown", col=c("red", "orange", "yellow", "blue", "green"),
border="brown")
```

➢ Output:



4) Bar Charts:

A bar chart represents data in rectangular bars with length of the bar proportional to the value of the variable. R uses the function barplot () to create the bar charts. R can draw both vertical and horizontal bars in the bar chart. Here, both vertical and Horizontal bars can be drawn.

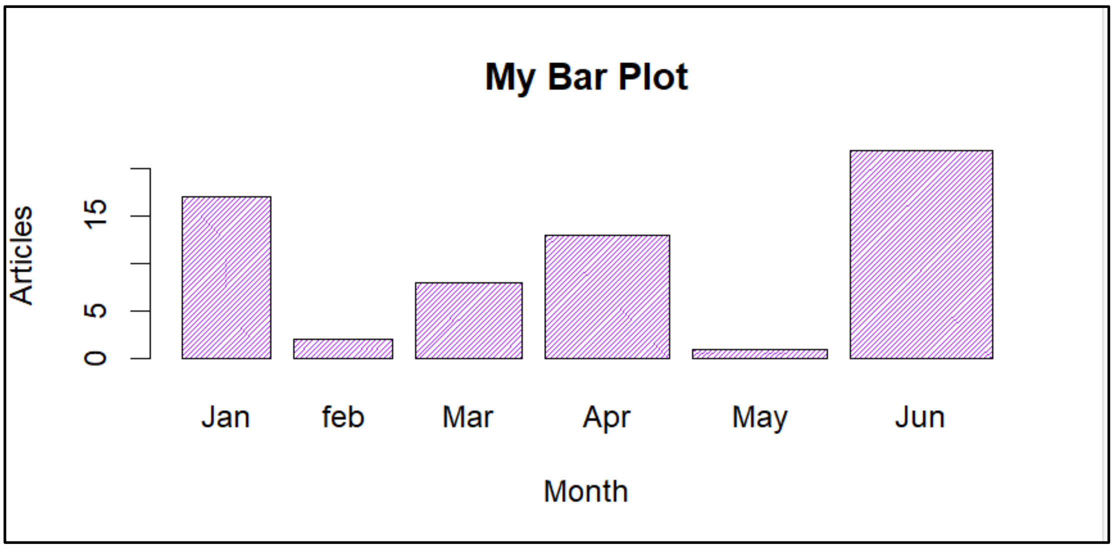
Syntax: *barplot(H, xlab, ylab, main, names.arg, col)*

S.No	Parameter	Description
1.	H	A vector or matrix which contains numeric values used in the bar chart.
2.	xlab	A label for the x-axis.
3.	ylab	A label for the y-axis.
4.	main	A title of the bar chart.
5.	names.arg	A vector of names that appear under each bar.
6.	col	It is used to give colors to the bars in the graph.
7.	width	It is used to change the width of the bars in the graph.
8.	density	This is a vector giving the density of shading lines, in lines per inch for the bars or bar components.

➢ **Example Program:**

```
A <- c(17, 2, 8, 13, 1, 22)
B <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun")
# Plot the bar chart
barplot(A, names.arg = B, xlab ="Month",
ylab ="Articles", col ="purple",
main ="My Bar Plot", density = 50, width=c(1,1.1,1.2,1.4,1.5,1.6))
```

➢ **Output:**



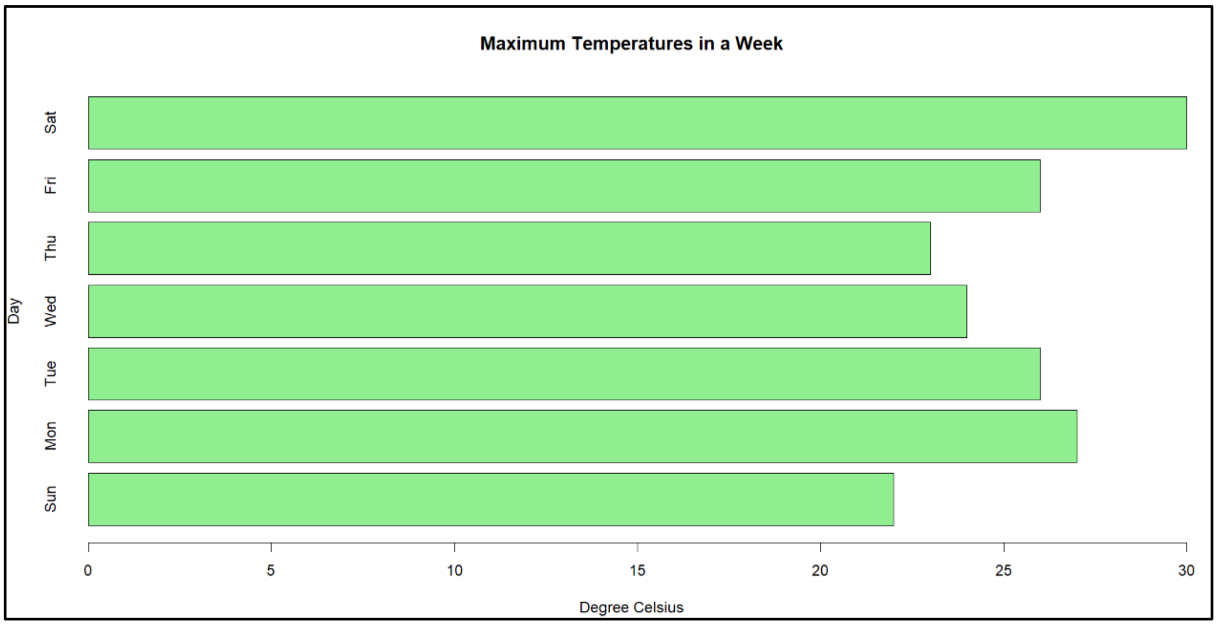
❖ Plotting Bars Horizontally:

We can also plot bars horizontally by providing the argument *horiz*=TRUE.

➢ **Example Program:**

```
max.temp <- c(22, 27, 26, 24, 23, 26, 30)
barplot (max.temp,
main = "Maximum Temperatures in a Week",
xlab = "Degree Celsius",
ylab = "Day",
names.arg = c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"),
col = "lightgreen",
horiz = TRUE)
```

➢ **Output:**



❖ **Grouped Bar Chart:**

Grouped bar charts are a way of showing information about different sub-groups of the main categories. A separate bar represents each of the sub-groups and these are usually colored or shared differently to distinguish between them. Grouped bar charts can be drawn as both horizontal and vertical charts depending upon the nature of the data to be presented.

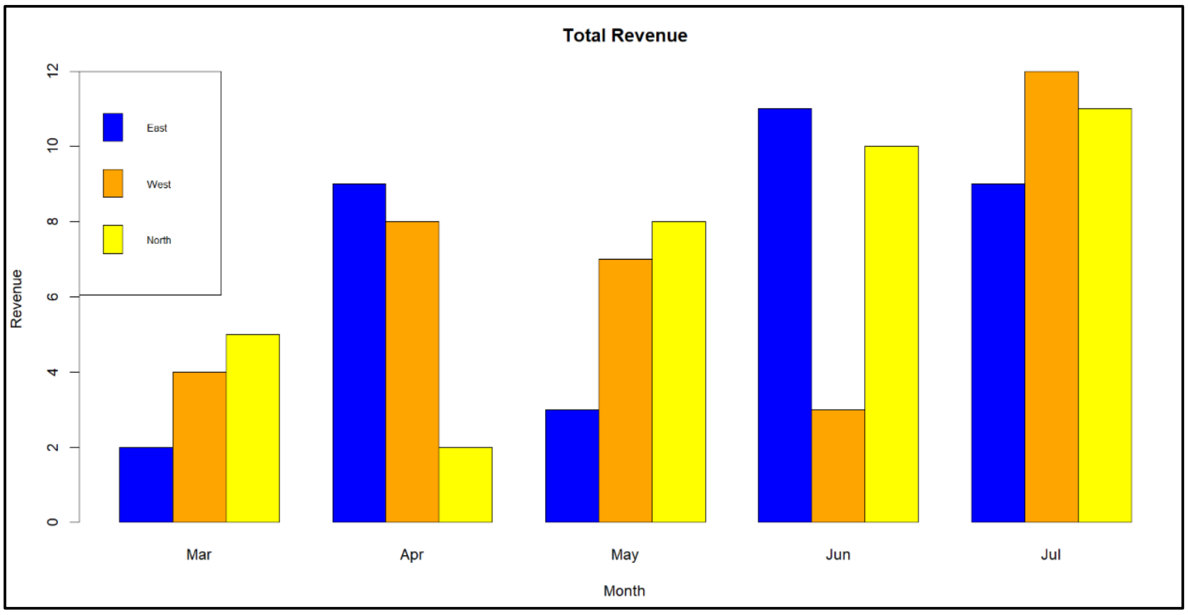
To display the bar explicitly we can use the **beside** parameter.

barplot(beside=TRUE)

➢ **Example Program:**

```
colors = c("blue", "orange", "yellow")
months = c("Mar", "Apr", "May", "Jun", "Jul")
regions =c("East", "West", "North")

# Create the matrix of the values.
Values = matrix(c(2, 9, 3, 11, 9, 4, 8, 7, 3, 12, 5, 2, 8, 10, 11),
                nrow = 3, ncol = 5, byrow = TRUE)

# Create the bar chart
barplot(Values, main = "Total Revenue", names.arg = months,
        xlab = "Month", ylab = "Revenue",
        col = colors, beside = TRUE)

# Add the legend to the chart
legend("topleft", regions, cex = 0.7, fill = colors)
```

➢ **Output:**



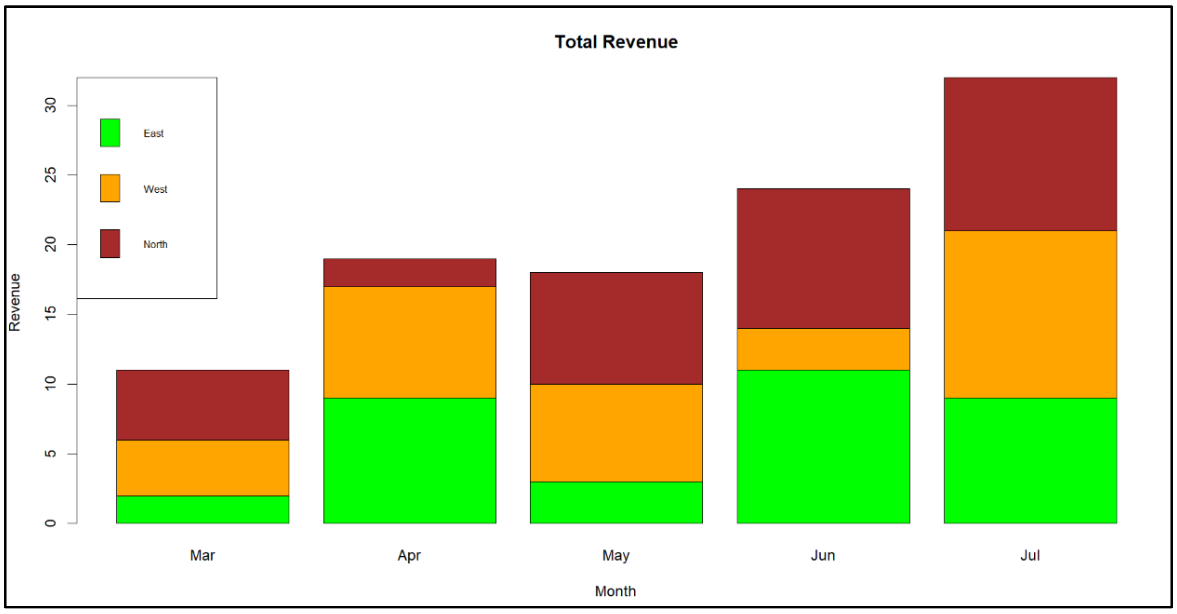
❖ **Stacked Bar Chart:**

Stacked bar charts are similar to grouped bar charts in that they are used to display information about the sub-groups that make up the different categories. In stacked bar charts the bars representing the sub- groups are placed on top of each other to make a single column, or side by side to make a single bar. The overall height or length of the bar shows the total size of the category whilst different colors or shadings are used to indicate the relative contribution of the different sub-groups.

➢ **Example Program:**

```
colors = c("green", "orange", "brown")
months <- c("Mar", "Apr", "May", "Jun", "Jul")
regions <- c("East", "West", "North")

# Create the matrix of the values.
Values <- matrix(c(2, 9, 3, 11, 9, 4, 8, 7, 3, 12, 5, 2, 8, 10, 11),
                nrow = 3, ncol = 5, byrow = TRUE)

# Create the bar chart
barplot(Values, main = "Total Revenue", names.arg = months,
        xlab = "Month", ylab = "Revenue", col = colors)

# Add the legend to the chart
legend("topleft", regions, cex = 0.7, fill = colors)
```

➢ **Output:**



5) **Boxplot:**

Boxplots are a measure of how well data is distributed across a data set. This divides the data set into three quartiles. This graph represents the minimum, maximum, average, first quartile, and the third quartile in the data set. Boxplot is also useful in comparing the distribution of data in a data set by drawing a boxplot for each of them. R provides a **boxplot()** function to create a boxplot. There is the following syntax of boxplot() function:

Syntax: *boxplot(x, data, notch, varwidth, names, main)*

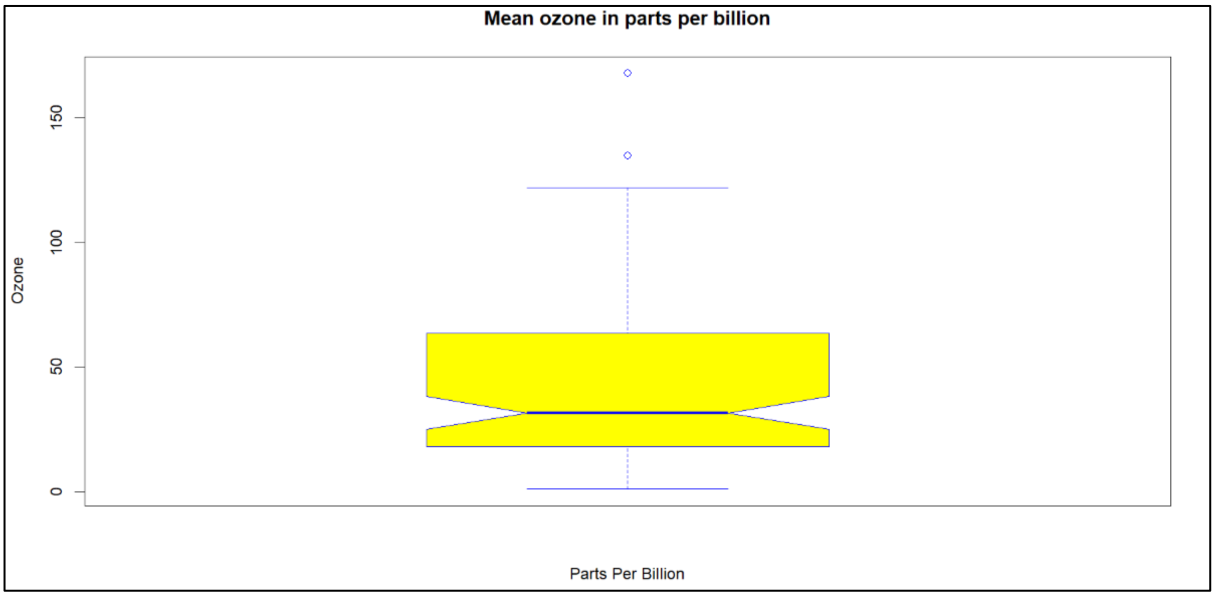
S.No	Parameter	Description
1.	x	It is a vector or a formula.
2.	data	It is the data frame.
3.	notch	It is a logical value set as true to draw a notch.
4.	varwidth	It is also a logical value set as true to draw the width of the box same as the sample size.

5.	names	It is the group of labels that will be printed under each boxplot.
6.	main	It is used to give a title to the graph.

➢ **Example Program:**

```
boxplot(airquality$Ozone,
        xlab = "Parts Per Billion",
        ylab = "Ozone",
        main = "Mean ozone in parts per billion",
        notch = TRUE,
        col="yellow",
        border="blue")
```

➢ **Output:**



❖ **<u>OBSERVATIONS:</u>**

We observe different charts, their syntax and example of R commands for charts and graphs.

❖ **<u>Learning outcomes (What I have learnt):</u>**

We learnt how to draw different charts by using different functions and installing various packages. For ex- Line graph, Pie chart, Scatter plot, Bar chart, Boxplot.

❖ **<u>Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):</u>**

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			

PRACTICAL-9

Graphs in R Programming

Student Name: Jatin Verma	**System ID:** 2021370138
Branch: BSc Data Science & Analytics	**Section/Group:** A
Semester: II	**Date of Performance:** 17/05/22
Subject Name: R-Programming	**Subject Code:** BDA 154

❖ AIM:

To understand Graphs in R programming.

❖ THEORY:

1) Histograms:

Histogram is a graphical representation used to create a graph with bars representing the frequency of grouped data in vector. In the histogram, each bar represents the height of the number of values present in the given range. Histogram is same as bar chart but only difference between them is histogram represents frequency of grouped data rather than data itself.

For creating a histogram, R provides **hist()** function, which takes a vector as an input and uses more parameters to add more functionality. There is the following syntax of hist() function:

Syntax: *hist(v, main ,xlab, ylab ,xlim ,ylim, breaks, col ,border)*

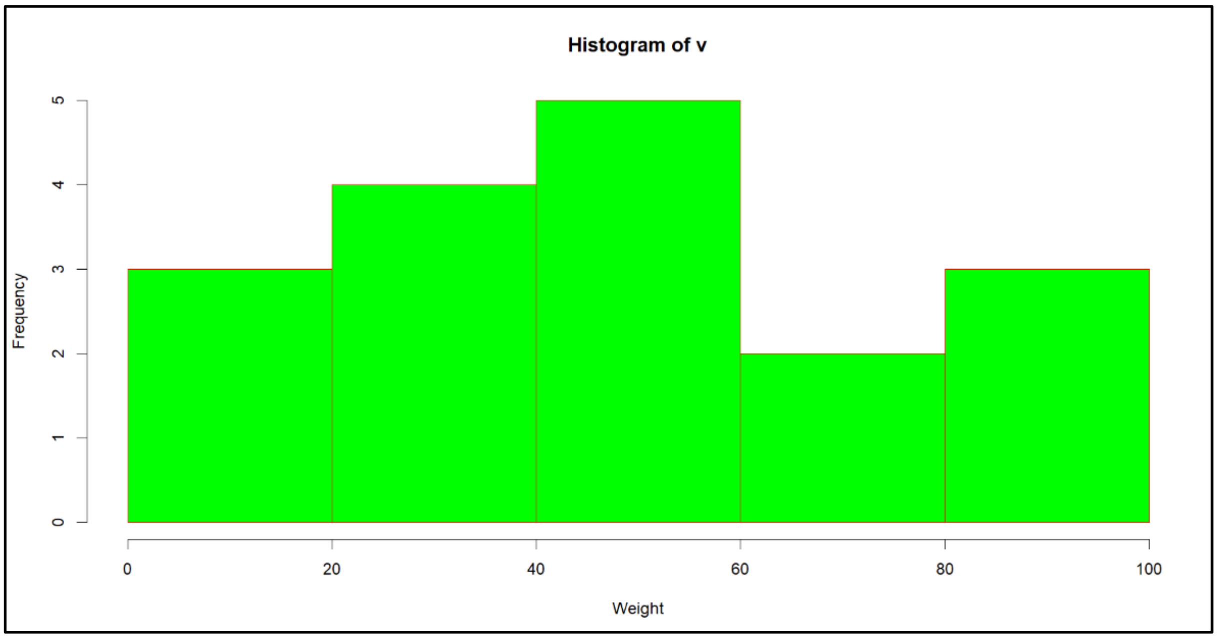
S.No	Parameter	Description
1.	v	It is a vector that contains numeric values.
2.	main	It indicates the title of the chart.
3.	col	It is used to set the color of the bars.
4.	border	It is used to set the border color of each bar.

5.	xlab	It is used to describe the x-axis.
6.	ylab	It is used to describe the y-axis.
7.	xlim	It is used to specify the range of values on the x-axis.
8.	ylim	It is used to specify the range of values on the y-axis.
9.	breaks	It is used to mention the width of each bar.

➢ **Example Program:**

```
v <- c(21, 23, 56, 90, 20, 7, 94, 12,
       57, 76, 69, 45, 34, 32, 49, 85, 57)
hist(v, xlab = "Weight", ylab="Frequency", col = "green", border = "red",
  xlim=c(0,100),breaks=5)
```

➢ **Output:**



❖ **<u>Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):</u>**

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			

PRACTICAL-10

Probability Distributions in R Programming

Student Name: Jatin Verma	**System ID:** 2021370138
Branch: BSc Data Science & Analytics	**Section/Group:** A
Semester: II	**Date of Performance:** 24/05/22
Subject Name: R-Programming	**Subject Code:** BDA 154

- ## AIM:

To understand Probability Distributions in R Programming.

- ## THEORY:

A Probability distribution describes how the values of a random variable are distributed. For example, the collection of all possible outcomes of a sequence of coin tossing is known to follow the binomial distribution. Whereas the means of sufficiently large samples of data. Population is known to resemble the normal distribution. Since the characteristics of these theoretical distributions are well understood, they can be used to make statistical inferences on the entire data population as a whole.

Types of probability distributions:

- Binomial Distribution
- Poisson Distribution
- Normal Distribution

Binomial Distribution

The binomial distribution model deals with finding the probability of success of an event which has only two possible outcomes in a series of experiments. For example, tossing of a coin always gives a head or a tail. The probability of finding exactly 3 heads in tossing a coin repeatedly for 10 times is estimated during the binomial distribution.

- R has four in-built functions to generate binomial distribution. They are described below:

  1) dbinom(x, size, prob)
  2) pbinom(x, size, prob)
  3) qbinom(p, size, prob)
  4) rbinom(n, size, prob)

- Following is the description of the parameters used :

  - **x** is a vector of numbers.
  - **p** is a vector of probabilities.
  - **n** is number of observations.
  - **size** is the number of trials.
  - **prob** is the probability of success of each trial.

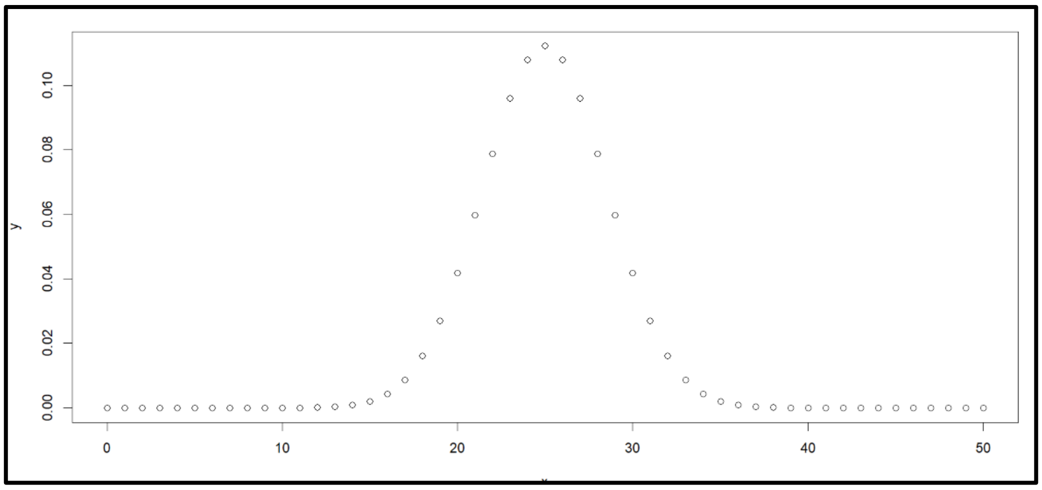
- **dbinom:**

  This function gives the probability density distribution at each point.

- **Code:**

```
# Create a sample of 50 numbers which are incremented by 1.
x <- seq(0,50,by = 1)

# Create the binomial distribution.
y <- dbinom(x,50,0.5)

# Plot the graph for this sample.
plot(x,y)
```

- **Output:**



- ♦ **pbinom:**

  This function gives the cumulative probability of an event. It is a single value representing the probability.

- **Code:**

```
# Probability of getting 26 or less heads from a 51 tosses of a coin.

x <- pbinom(26,51,0.5)

print(x)
```

- **Output:**

```
[1] 0.610116
```

- ♦ **qbinom:**

  This function takes the probability value and gives a number whose cumulative value matches the probability value.

- **Code:**

```
# How many heads will have a probability of 0.25 will come out when a coin
# is tossed 51 times.
x <- qbinom(0.25,51,1/2)
print(x)
```

- **Output:**

```
[1] 23
```

- ♦ **rbinom:**

  This function generates required number of random values of given probability from a given sample.

- **Code:**

```
# Find 8 random values from a sample of 150 with probability of 0.4.
x <- rbinom(8,150,.4)
print(x)
```

- **Output:**

```
[1]  60 67 74 58 55 71 65 61
```

❖ **Normal Distribution**

In a random collection of data from independent sources, it is generally observed that the distribution of data is normal. Which means, on plotting a graph with the value of the variable in the horizontal axis and the count of the values in the vertical axis we get a bell shape curve. The center of the curve represents the mean of the data set. In the graph, fifty percent of values lie to the left of the mean and the other fifty percent lie to the right of the graph. This is referred as normal distribution in statistics.

➢ R has four in-built functions to generate binomial distribution. They are described below:

1) dnorm(x, mean, sd)
2) pnorm(x, mean, sd)
3) qnorm(p, mean, sd)
4) rnorm(n, mean, sd)

➢ Following is the description of the parameters used :

- **x** is a vector of numbers.
- **p** is a vector of probabilities.
- **n** is number of observations.
- **mean** is the mean value of the sample data. It's default value is zero.
- **sd** is the standard deviation. It's default value is 1.

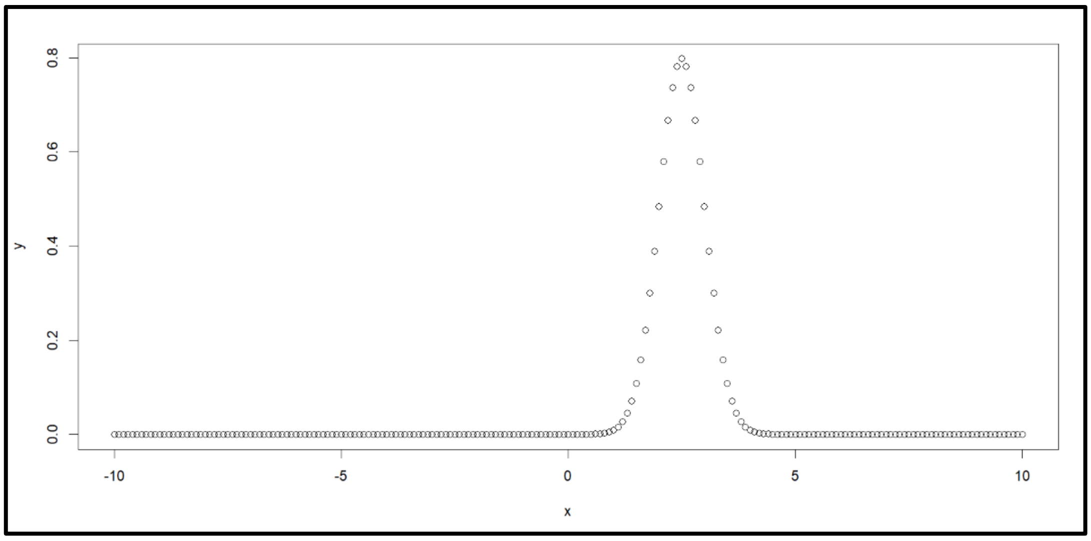
♦ **dnorm():**

This function gives height of the probability distribution at each point for a given mean and standard deviation.

• **Code:**

```
# Create a sequence of numbers between -10 and 10 incrementing by 0.1.

x <- seq(-10, 10, by = .1)

# Choose the mean as 2.5 and standard deviation as 0.5.

y <- dnorm(x, mean = 2.5, sd = 0.5)

plot(x,y)
```

- **Output:**



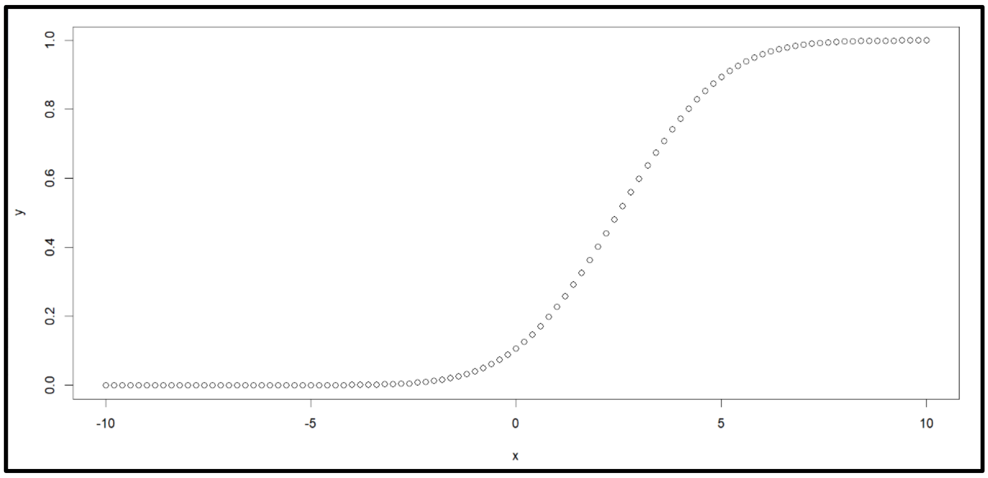
♦ **pnorm():**

This function gives the probability of a normally distributed random number to be less that the value of a given number. It is also called "Cumulative Distribution Function".

- **Code:**

```
# Create a sequence of numbers between -10 and 10 incrementing by 0.2.

x <- seq(-10,10,by = .2)

# Choose the mean as 2.5 and standard deviation as 2.

y <- pnorm(x, mean = 2.5, sd = 2)

# Plot the graph.

plot(x,y)
```

- **Output:**



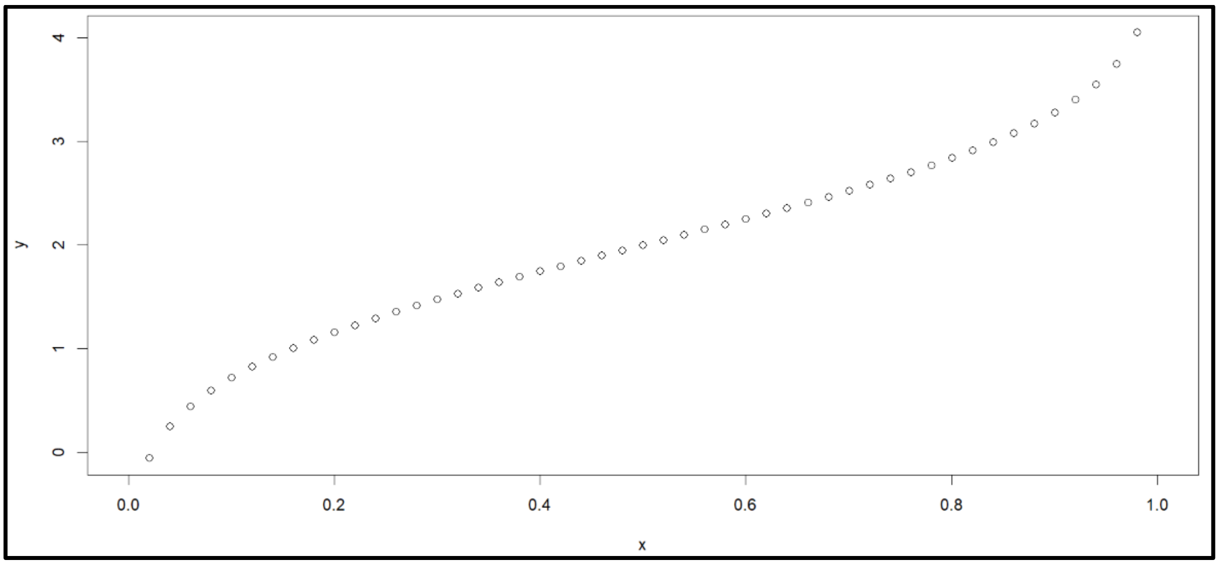
- **qnorm():**

  This function takes the probability value and gives a number whose cumulative value matches the probability value.

- **Code:**

```
# Create a sequence of probability values incrementing by 0.02.
x <- seq(0, 1, by = 0.02)

# Choose the mean as 2 and standard deviation as 3.
y <- qnorm(x, mean = 2, sd = 1)

# Plot the graph.
plot(x,y)
```

- **Output:**



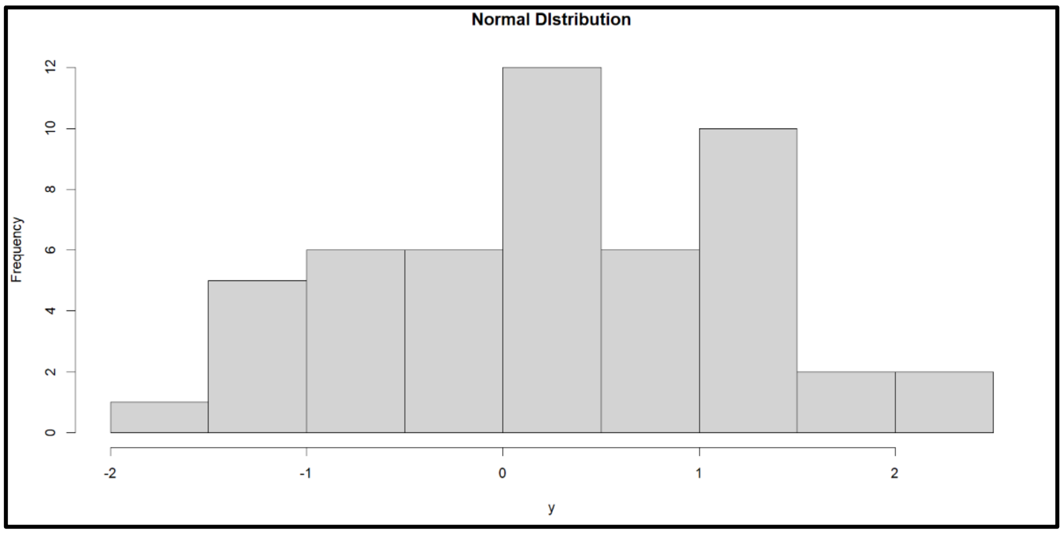
- **rnorm():**

  This function is used to generate random numbers whose distribution is normal. It takes the sample size as input and generates that many random numbers. We draw a histogram to show the distribution of the generated numbers.

- **Code:**

```
# Create a sample of 50 numbers which are normally distributed.
y <- rnorm(50)
# Plot the histogram for this sample.
hist(y, main = "Normal Distribution")
```

- **Output:**



❖ **Poisson Distribution**

The Poisson distribution represents the probability of a provided number of cases happening in a set period of space or time if these cases happen with an identified constant mean rate (free of the period since the ultimate event).

➢ There are four Poisson functions available in R:

- dpois()
- ppois()
- qpois()
- rpois()

♦ **dpois:**

This function is used for illustration of Poisson density in an R plot. The function dpois() calculates the probability of a random variable that is available within a certain range.

Syntax: dpois(k,λ,log)

where,

k: *number of successful events happened in an interval*
λ : *mean per interval*
log: *If TRUE then the function returns probability in form of log*

- **Code:**

```
dpois(2, 3)

dpois(6, 6)
```

- **Output:**

```
[1] 0.2240418

[1] 0.1606231
```

♦ **ppois:**

This function is used for the illustration of cumulative probability function in an R plot. The function ppois() calculates the probability of a random variable that will be equal to or less than a number.

Syntax: ppois(k, λ, lower.tail, log)

where,

k: *number of successful events happened in an interval*
λ : *mean per interval*
lower.tail: *If TRUE then left tail is considered otherwise if the FALSE right tail is considered*
log: *If TRUE then the function returns probability in form of log*

- **Code:**

```
ppois(2, 3)

ppois(6, 6)
```

- **Output:**

```
[1] 0.4231901

[1] 0.6063028
```

- **rpois:**

  The function rpois() is used for generating random numbers from a given Poisson's distribution.

  Syntax: rpois(q, λ)

  where,

  q: *number of random numbers needed*
  λ : *mean per interval*

- **Code:**

```
rpois(2, 3)

rpois(6, 6)
```

- **Output:**

```
[1] 2 3
[1]  6  7  6  10  9  4
```

- **qpois:**

  The function qpois() is used for generating quantile of a given Poisson's distribution.
  In probability, quantiles are marked points that divide the graph of a probability distribution into intervals (continuous) which have equal probabilities.

  Syntax: qpois(k, λ, lower.tail, log)

  where,

  k: *number of successful events happened in an interval*
  λ : *mean per interval*
  lower.tail: *If TRUE then left tail is considered otherwise if the FALSE right tail is considered*
  log: *If TRUE then the function returns probability in form of log*

- **<u>Code:</u>**

```
y <- c(.01, .05, .1, .2)

qpois(y, 2)

qpois(y, 6)
```

- **<u>Output:</u>**

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

❖ **<u>OBSERVATIONS:</u>**
We observe different types of probability distribution in R programming by plotting their graph.

❖ **<u>Learning outcomes (What I have learnt):</u>**

1. How to use different types of probability distribution in R programming i.e. Normal distribution, binomial distribution and poisson distribution.

❖ **<u>Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):</u>**

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			