R PROGRAMMING LAB

Week 1:

EXPERIMENT NO:1(a)
DATE:

Aim: Installing R and RStudio.

Description:

- **R is an open-source programming language and free environment that specializes in statistical computing and graphical representation.**
- **It is mainly used by statisticians and data miners for developing statistical software and performing data analysis.**
- **To use R language, you need the R environment to be installed on your machine, and an IDE (Integrated development environment) to run the language (can also be run using CMD on Windows or Terminal on Linux).**

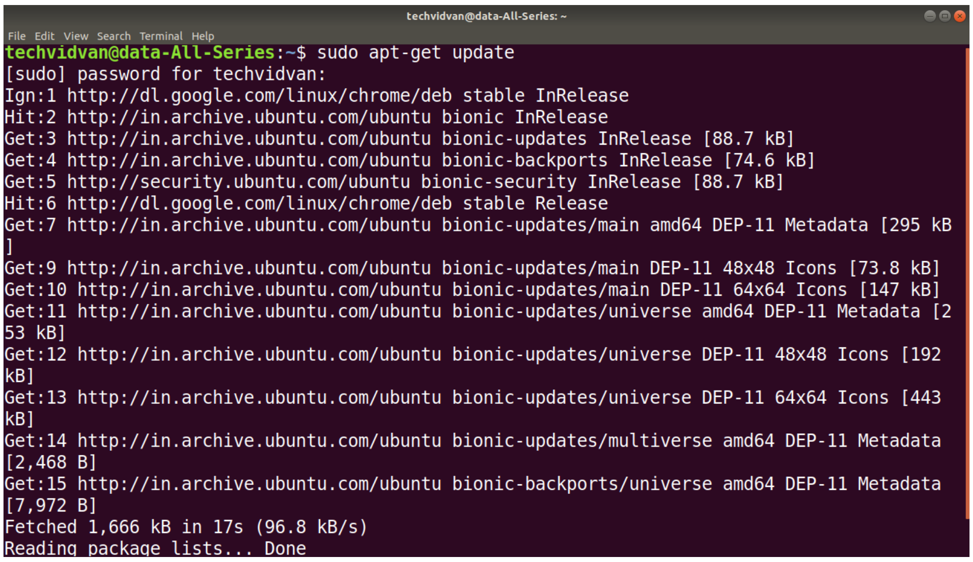
Installing R and RStudio on Linux:

Linux software is often distributed as source code and then compiled by package managers like apt or yum. To install R in Ubuntu, we will have to go through the following steps.

Install R on Linux

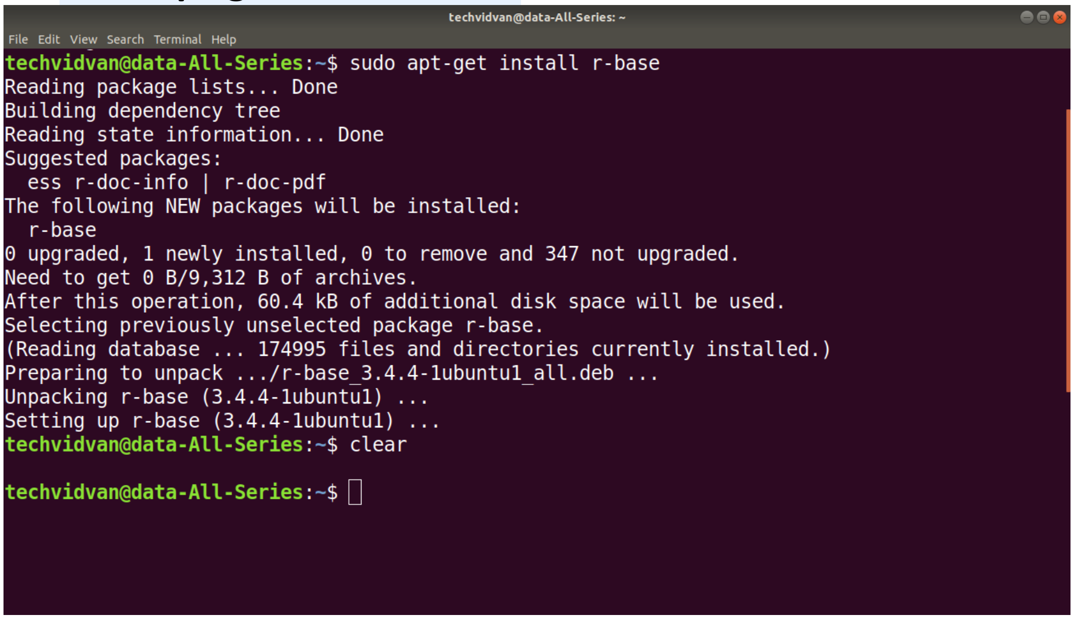
Install the R-base package using the following code

```
1. sudo apt-get update.
```



```
techvidvan@data-All-Series:~$ sudo apt-get update
[sudo] password for techvidvan:
Ign:1 http://dl.google.com/linux/chrome/deb stable InRelease
Hit:2 http://in.archive.ubuntu.com/ubuntu bionic InRelease
Get:3 http://in.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:4 http://in.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:5 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:6 http://dl.google.com/linux/chrome/deb stable Release
Get:7 http://in.archive.ubuntu.com/ubuntu bionic-updates/main amd64 DEP-11 Metadata [295 kB
]
Get:9 http://in.archive.ubuntu.com/ubuntu bionic-updates/main DEP-11 48x48 Icons [73.8 kB]
Get:10 http://in.archive.ubuntu.com/ubuntu bionic-updates/main DEP-11 64x64 Icons [147 kB]
Get:11 http://in.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 DEP-11 Metadata [2
53 kB]
Get:12 http://in.archive.ubuntu.com/ubuntu bionic-updates/universe DEP-11 48x48 Icons [192
kB]
Get:13 http://in.archive.ubuntu.com/ubuntu bionic-updates/universe DEP-11 64x64 Icons [443
kB]
Get:14 http://in.archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 DEP-11 Metadata
[2,468 B]
Get:15 http://in.archive.ubuntu.com/ubuntu bionic-backports/universe amd64 DEP-11 Metadata
[7,972 B]
Fetched 1,666 kB in 17s (96.8 kB/s)
Reading package lists... Done
```

- **sudo apt-get install r-base**



```
techvidvan@data-All-Series:~$ sudo apt-get install r-base
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  ess r-doc-info | r-doc-pdf
The following NEW packages will be installed:
  r-base
0 upgraded, 1 newly installed, 0 to remove and 347 not upgraded.
Need to get 0 B/9,312 B of archives.
After this operation, 60.4 kB of additional disk space will be used.
Selecting previously unselected package r-base.
(Reading database ... 174995 files and directories currently installed.)
Preparing to unpack .../r-base_3.4.4-1ubuntu1_all.deb ...
Unpacking r-base (3.4.4-1ubuntu1) ...
Setting up r-base (3.4.4-1ubuntu1) ...
techvidvan@data-All-Series:~$ clear

techvidvan@data-All-Series:~$
```

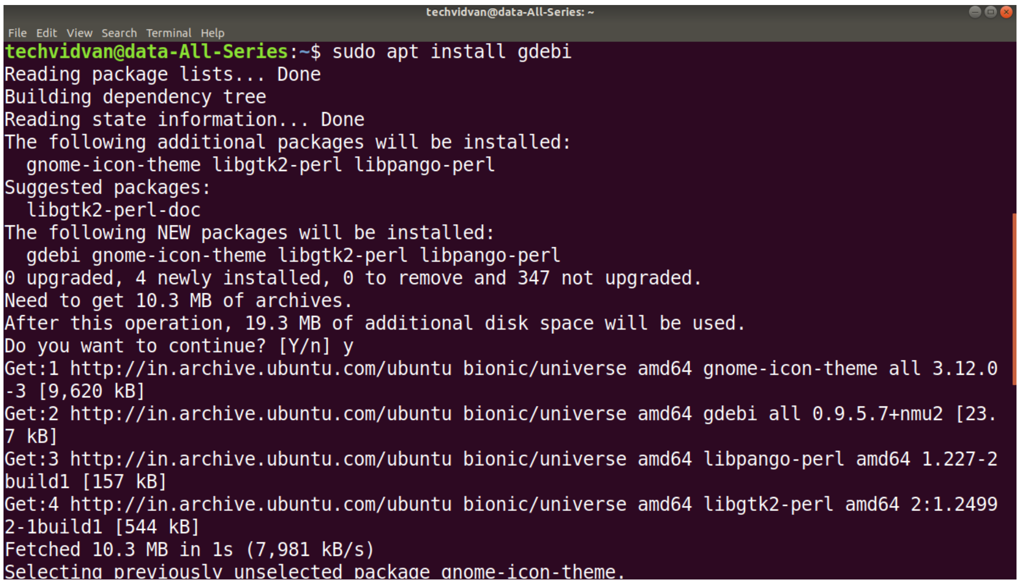
- **After running the command, a confirmation prompt will appear. Answer it with a 'Y' for yes.**

Install RStudio on Linux

Step – 1 Next comes installing RStudio. To install RStudio, go to download RStudio, click on the download button for RStudio desktop, click the link for the latest R version for your OS and save the .deb file.

Step – 2 Download and install the gdebi package using the following commands.

◆ sudo apt install gdebi

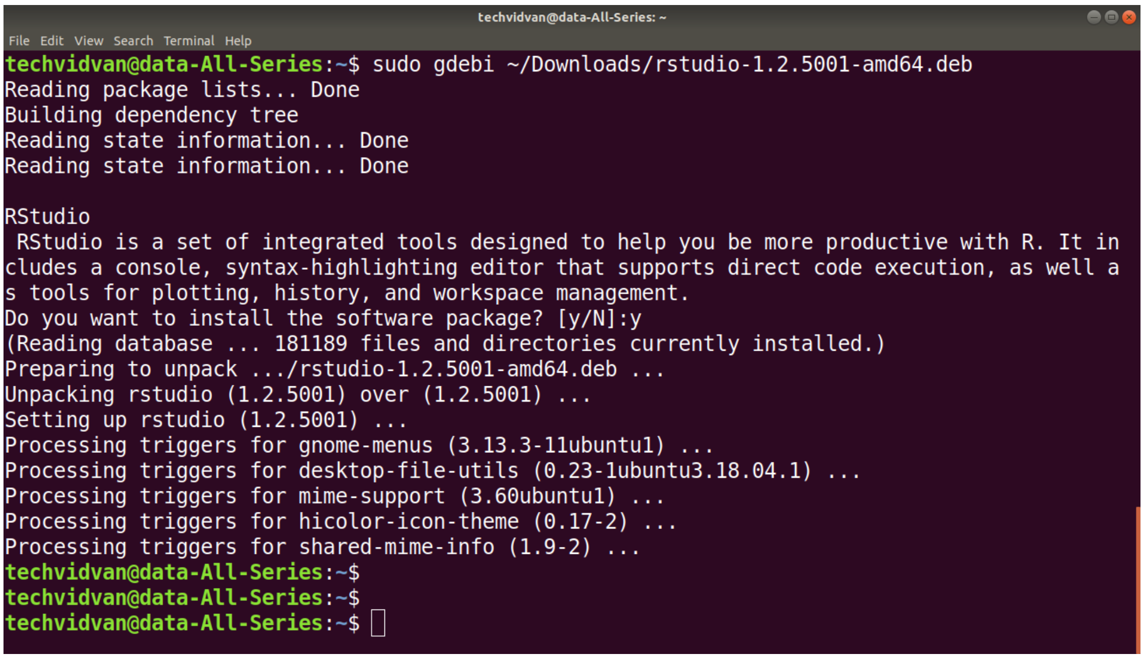


```
techvidvan@data-All-Series:~$ sudo apt install gdebi
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  gnome-icon-theme libgtk2-perl libpango-perl
Suggested packages:
  libgtk2-perl-doc
The following NEW packages will be installed:
  gdebi gnome-icon-theme libgtk2-perl libpango-perl
0 upgraded, 4 newly installed, 0 to remove and 347 not upgraded.
Need to get 10.3 MB of archives.
After this operation, 19.3 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://in.archive.ubuntu.com/ubuntu bionic/universe amd64 gnome-icon-theme all 3.12.0
-3 [9,620 kB]
Get:2 http://in.archive.ubuntu.com/ubuntu bionic/universe amd64 gdebi all 0.9.5.7+nmu2 [23.
7 kB]
Get:3 http://in.archive.ubuntu.com/ubuntu bionic/universe amd64 libpango-perl amd64 1.227-2
build1 [157 kB]
Get:4 http://in.archive.ubuntu.com/ubuntu bionic/universe amd64 libgtk2-perl amd64 2:1.2499
2-1build1 [544 kB]
Fetched 10.3 MB in 1s (7,981 kB/s)
Selecting previously unselected package gnome-icon-theme.
```

- **Answer with a ‘Y’ for yes to confirm when prompted.**

Step – 3: Use the following commands to install the .deb package

◆ **sudo gdebi /path/to/the/file/.deb**



```
techvidvan@data-All-Series:~$ sudo gdebi ~/Downloads/rstudio-1.2.5001-amd64.deb
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading state information... Done

RStudio
 RStudio is a set of integrated tools designed to help you be more productive with R. It in
cludes a console, syntax-highlighting editor that supports direct code execution, as well a
s tools for plotting, history, and workspace management.
Do you want to install the software package? [y/N]:y
(Reading database ... 181189 files and directories currently installed.)
Preparing to unpack .../rstudio-1.2.5001-amd64.deb ...
Unpacking rstudio (1.2.5001) over (1.2.5001) ...
Setting up rstudio (1.2.5001) ...
Processing triggers for gnome-menus (3.13.3-11ubuntu1) ...
Processing triggers for desktop-file-utils (0.23-1ubuntu3.18.04.1) ...
Processing triggers for mime-support (3.60ubuntu1) ...
Processing triggers for hicolor-icon-theme (0.17-2) ...
Processing triggers for shared-mime-info (1.9-2) ...
techvidvan@data-All-Series:~$
techvidvan@data-All-Series:~$
techvidvan@data-All-Series:~$
```

EXPERIMENT NO:1(b)
DATE:

Aim: Basic functionality of R, variable, data types in R.

Description:

Features of R Programming:

1. <u>Open-source:</u>

R is an open-source software environment(R is freely available).It is free of cost and can be adjusted and adapted according to the user's and the project's requirements.You can make improvements and add packages for additional functionalities.

2. <u>Strong Graphical Capabilities:</u>

R can produce static graphics with production quality visualizations and has extended libraries providing interactive graphic capabilities.3. Highly Active Community

3. <u>Highly Active Community:</u>

R has an open-source library which is supported by its growing number of users.The R environment is continuously growing. This growth is due to its large user-base.

4. <u>Distributed Computing:</u>

In distributed computing, tasks are split between multiple processing nodes to reduce processing time and increase efficiency.R has packages like ddR and multiDplyr that enable it to use distributed computing to process large data sets.

5. <u>Running Code Without a Compiler:</u>

R is an interpreted language which means that it does not need a compiler to make a program from the code. R directly interprets provided code into lower-level calls and pre-compiled code.

6. <u>Cross-platform Support:</u>

R is machine-independent. It supports the cross-platform operation. Therefore, it can be used on many different operating systems.

variables in R:

A variable is a memory allocated for the storage of specific data and the name associated with the variable is used to work around this reserved block. The name given to a variable is known as its variable name.

R Programming Language is a dynamically typed language, i.e. the R Language Variables variables are not declared with a data type rather they take the data type of the R-object assigned to them.

<u>Nomenclature of R Variables:</u>

The following rules need to be kept in mind while naming a variable:

- A valid variable name consists of a combination of alphabets, numbers, dot(.), and underscore(_) characters. Example: var.1_ is valid
- Apart from the dot and underscore operators, no other special character is allowed. Example: var$1 or var#1 both are invalid
- Variables can start with alphabets or dot characters. Example: .var or var is valid
- The variable should not start with numbers or underscore. Example: 2var or _var is invalid.
- If a variable starts with a dot the next thing after the dot cannot be a number. Example: .3var is invalid

- has to be started always with an alphabet.Other special characters like('!', '@', '#', '$') are not allowed in the variable name.

R supports three ways of variable assignment:

1. Using equal operator- data is copied from right to left.
2. Using leftward operator- data is copied from right to left.
3. Using rightward operator- data is copied from left to right.

Syntax:

```
#using equal to operator

variable_name = value

#using leftward operator

variable_name <- value

#using rightward operator

value -> variable_name
```

Data Types:

Each variable in R has an associated data type. R Programming language has the following basic data types and the following table shows the data type and the values that each data type can take.

1.Numeric Datatype: Decimal values are called numerics in R. It is the default data type for numbers in R.

2.Integer Datatype:R supports integer data types which are the set of all integers. You can create as well as convert a value into an integer type using the as.integer() function. You can also use the capital 'L' notation as a suffix to denote that a particular value is of the integer data type.

3.Logical Datatype: R has logical data types that take either a value of true or false. A logical value is often created via a comparison between variables.

4.Complex Datatype: R supports complex data types that are set of all the complex numbers. The complex data type is to store numbers with an imaginary component.
5.Character Datatype: R supports character data types where you have all the alphabets and special characters. It stores character values or strings. Strings in R can contain alphabets, 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.

- ◆ to find the data type of an object you have to use class() function

Syntax: class(object)

program:

```
# R program to illustrate Initialization of variables using equal to operator

var1 = "hello"

print(var1)

var2 < - "hello"# using leftward operator

print(var2)

"hello" -> var3# using rightward operator

print(var3)

# A simple R program to illustrate Numeric data type

x = 5.6# Assign a decimal value to x

print(class(x))# print the class name of variable

print(typeof(x)) # print the type of variable
```

```
# A simple R program to illustrate Numeric data type

# Assign a integer value to y

y = 5

print(is.integer(y))# is y an integer?

# A simple R program to illustrate integer data type

 x = as.integer(5)# Create an integer value

print(class(x))

print(typeof(x))

y = 5L# Declare an integer by appending an L suffix.

print(class(y))

print(typeof(y))
# A simple R program to illustrate logical data type

# Sample values

x = 4

y = 3

# Comparing two values

z = x > y

# print the logical value

print(z)

print(class(z))

print(typeof())

# A simple R program to illustrate complex data type

# Assign a complex value to x
```

```
x = 4 + 3i
print(class(x))
print(typeof(x))
#character A simple R program to illustrate character data type
 # Assign a character value to char
char = "Geeksforgeeks"
print(class(char))
print(typeof(char))
```

Output:

```
[1] "hello"
[1] "hello"
[1] "hello"
[1] "numeric"
[1] "double"
[1] FALSE
#integer
[1] "integer"
[1] "integer"
[1] "integer"
[1] "integer"
#logical
[1] TRUE
[1] "logical"
[1] "logical"
```

```
#complex
[1] "complex"
[1] "complex"
#character
[1] "character"
[1] "character"
```

EXPERIMENT NO:2(a)
DATE:

Aim:Implement R seript to show the usage of various operators available in R language.

Description:

Operators are the symbols directing the compiler to perform various kinds of operations between the operands.

R supports majorly four kinds of binary operators between a set of operands.

Types of the operator in R language:

- Arithmetic Operators
- Logical Operators
- Relational Operators
- Assignment Operators
- Miscellaneous Operator

➢ Arithmetic Operators:

Arithmetic operations 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.

1.Addition operator (+): The values at the corresponding positions of both the operands are added.
syntax:a+b

```
Input : a <- c (1, 0.1)
        b <- c (2.33, 4)
        print (a+b)
Output : 3.33
         4.10
```

<u>2.Subtraction Operator (-):</u>The second operand values are subtracted from the first.

syntax:a-b

Input : a <- 6
```
             b <- 8.4
             print (a-b)
```
Output : -2.4

<u>3.multiplication Operator (*):</u> The multiplication of corresponding elements of vectors and Integers are multiplied with the use of '*' operator.

Input : B= matrix(c(4,6i),nrow=1,ncol=2)
```
             C= matrix(c(2,2i ),nrow=1, ncol=2)
             print (B*C)
```
Output : 8+0i
```
               -12+0i
```

<u>4.Division Operator (/):</u> The first operand is divided by the second operand with the use of '/' operator.

syntax; a/b

Input : a <- 1
```
             b <- 0
             print (a/b)
```
Output : -Inf

<u>5.Power Operator (^):</u> The first operand is raised to the power of the second operand.

Input : list1 <- c(2, 3)
```
             list2 <- c(2,4)
          print(list1^list2)
```
Output : 4
```
                 81
```
<u>6.Modulo Operator (%%):</u> The remainder of the first operand divided by the second operand is returned.

Input : list1<- c(2, 3)
```
             list2<-c(2,4)
             print(list1%%list2)
```

Output : 0

3

- **<u>Logical Operators</u>: 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. Any non zero integer value is considered as a TRUE value, be it complex or real number.**

<u>1.Element-wise Logical AND operator (&):</u>Returns True if both the operands are True.

```
Input : list1 <- c(TRUE, 0.1)
           list2 <- c(0,4+3i)
             print(list1 & list2)
Output : FALSE
             TRUE
```

2. <u>Logical OR operator (|):</u>Returns True if either of the operands are True.

```
Input : list1 <- c(TRUE, 0.1)

            list2 <- c(0,4+3i)
            print(list1|list2)
Output : TRUE

  TRUE
```

<u>3.NOT operator (!):</u> A unary operator that negates the status of the elements of the operand.

```
Input : list1 <- c(0,FALSE)

          print(!list1)
```

Output : TRUE

TRUE

4.Logical AND operator (&&):Returns True if both the first elements of the operands are True.

Input : list1 <- c(TRUE, 0.1)

list2 <- c(0,4+3i)

print(list1 && list2)

Output : FALSE

5.Logical OR operator (||): Returns True if either of the first elements of the operands are True.

Input : list1 <- c(TRUE, 0.1)

list2 <- c(0,4+3i)

print(list1||list2)

Output : TRUE

- Relational Operators:The relational operators carry out comparison operations between the corresponding elements of the operands.

Returns a boolean TRUE value if the first operand satisfies the relation compared to the second.

- Assignment Operators:Assignment operators are used to assign values to various data objects in R. The objects may be integers, vectors, or functions.

There are two kinds of assignment operators:

Left and Right

1. Left Assignment (<- or <<- or =): Assigns a value to a vector.

Input : vec1 = c("ab", TRUE)

print (vec1)

Output : "ab"   "TRUE"

2. Right Assignment (-> or ->>): Assigns value to a vector.

Input : c("ab", TRUE) ->> vec1

    print (vec1)

Output : "ab"   "TRUE"

- Miscellaneous Operators:These are the mixed operators that simulate the printing of sequences and assignment of vectors, either left or right-handed.

1. %in% Operator: Checks if an element belongs to a list and returns a boolean value TRUE if the value is present else FALSE.

2.Colon Operator(:): Prints a list of elements starting with the element before the color to the element after it.

PROGRAM:

```
# R program to illustratethe use of Arithmetic operators
vec1 <- c(0, 2)
vec2 <- c(2, 3)
# Performing operations on Operands
cat ("Addition of vectors :", vec1 + vec2, "\n")
cat ("Subtraction of vectors :", vec1 - vec2, "\n")
cat ("Multiplication of vectors :", vec1 * vec2, "\n")
cat ("Division of vectors :", vec1 / vec2, "\n")
cat ("Modulo of vectors :", vec1 %% vec2, "\n")
cat ("Power operator :", vec1 ^ vec2)
# R program to illustrate the use of Logical operators
vec1 <- c(0,2)
vec2 <- c(TRUE,FALSE)
# Performing operations on Operands
cat ("Element wise AND :", vec1 & vec2, "\n")
cat ("Element wise OR :", vec1 | vec2, "\n")
```

```r
cat ("Logical AND :", vec1 && vec2, "\n")
cat ("Logical OR :", vec1 || vec2, "\n")
cat ("Negation :", !vec1)

# R program to illustrate the use of Relational operators
vec1 <- c(0, 2)
vec2 <- c(2, 3)
# Performing operations on Operands
cat ("Vector1 less than Vector2 :", vec1 < vec2, "\n")
cat ("Vector1 less than equal to Vector2 :", vec1 <= vec2, "\n")
cat ("Vector1 greater than Vector2 :", vec1 > vec2, "\n")
cat ("Vector1 greater than equal to Vector2 :", vec1 >= vec2, "\n")
cat ("Vector1 not equal to Vector2 :", vec1 != vec2, "\n")
# R program to illustrate the use of Assignment operators
vec1 <- c(2:5)
c(2:5) ->> vec2
vec3 <<- c(2:5)
vec4 = c(2:5)
c(2:5) -> vec5
# Performing operations on Operands
cat ("vector 1 :", vec1, "\n")
cat("vector 2 :", vec2, "\n")
cat ("vector 3 :", vec3, "\n")
cat("vector 4 :", vec4, "\n")
cat("vector 5 :", vec5)

# R program to illustrate the use of Miscellaneous operators
mat <- matrix (1:4, nrow = 1, ncol = 4)
print("Matrix elements using : ")
print(mat)
product = mat %*% t(mat)
print("Product of matrices")
print(product,)
cat ("does 1 exist in prod matrix :", "1" %in% product)
```

Output:

```
Addition of vectors : 2 5
Subtraction of vectors : -2 -1
Multiplication of vectors : 0 6
Division of vectors : 0 0.6666667
Modulo of vectors : 0 2
Power operator : 0 8

Element wise AND : FALSE FALSE
Element wise OR : TRUE TRUE
Logical AND : FALSE
Logical OR : TRUE
Negation : TRUE FALSE

Vector1 less than Vector2 : TRUE TRUE
Vector1 less than equal to Vector2 : TRUE TRUE
Vector1 greater than Vector2 : FALSE FALSE
Vector1 greater than equal to Vector2 : FALSE FALSE
Vector1 not equal to Vector2 : TRUE TRUE

vector 1 : 2 3 4 5
vector 2 : 2 3 4 5
vector 3 : 2 3 4 5
vector 4 : 2 3 4 5
vector 5 : 2 3 4 5
[1] "Matrix elements using : "
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[1] "Product of matrices"
     [,1]
[1,]   30
does 1 exist in prod matrix : FALSE
```

EXPERIMENT NO:2(B)
DATE:

Aim:Implement R script to read person's age from keyboard and display whether he is eligible for voting or not.

Description:
<u>Else If:</u>
The else if keyword is R's way of saying "if the previous conditions were not true, then try this condition".

syntax:

```
if( age >= 18)
 {
   //statement
 }
else
{
   //statement
}
```

PROGRAM:

```
{
   age <- as.integer(readline(prompt = "Enter your age :"))
   if (age >= 18)
   {
      print(paste("You are valid for voting :", age))
     }
     else
     {
        print(paste("You are not valid for voting :", age))
     }
}
```

OUTPUT:

```
Enter your age :48
 [1] "You are valid for voting : 48"
```

EXPERIMENT NO:2(c)
DATE:

Aim:Implement R script to find biggest number between two numbers.

Description:
readline(): reads a line from the terminal (in interactive use).
<u>Arguments:</u>
prompt: The string printed when prompting the user for input. Should usually end with a space " ".

program:

```
{
   x <- as.integer(readline(prompt = "Enter first number :"))
   y <- as.integer(readline(prompt = "Enter second number :"))
   z <- as.integer(readline(prompt = "Enter third number :"))
 if (x > y) {
     if (x > z)
       print(paste("Greatest is :", x))
     else
       print(paste("Greatest is :", z))
   } else  {
     if (y > z)
       print(paste("Greatest is :", y))
     else{
       print(paste("Greatest is :", z))
     }
   }
 }
```

output:

```
Enter first number :3
Enter second number :2
Enter third number :55
[1] "Greatest is : 55"
```

EXPERIMENT NO:2(d)
DATE:

Aim:Implement R script to check the given year is leap year or not.

Description:
A leap year is exactly divisible by 4 except for century years (years ending with 00). The century year is a leap year only if it is perfectly divisible by 400.

<u>if...else:</u>
The if-else statement helps to check the condition based on the condition the expression is performed. The syntax of if-else statement is like,

```
if (condition1) {
  expr1
  } else if (condition2) {
expr2
} else if (condition3) {
expr3
} else {
  expr4
}
```

<u>Modulo Operator (%%):</u>
The remainder of the first operand divided by the second operand is returned.

```
Input : list1<- c(2, 3)
             list2<-c(2,4)
           print(list1%%list2)
Output : 0    3
```

program:

```
# Program to check if the input year is a leap year or not
year = as.integer(readline(prompt="Enter a year: "))
if((year %% 4) == 0) {
if((year %% 100) == 0) {
```

```
if((year %% 400) == 0) {
print(paste(year,"is a leap year"))
} else {
print(paste(year,"is not a leap year"))
}
} else {
print(paste(year,"is a leap year"))
}
} else {
print(paste(year,"is not a leap year"))
}
```

output:

```
Enter a year: 1900
[1] "1900 is not a leap year"
```

Week 3:
EXPERIMENT NO:3(A)
DATA:
Aim: Implement R Script to create a list.
Description:
List is a data structure having components of mixed data types. A vector having all elements of the same type is called atomic vector but a vector having elements of different type is called list.

- **List can be created using the list() function.**

```
  x <- list("a" = 2.5, "b" = TRUE, "c" = 1:3)
```

- **The list elements can be given names and they can be accessed using these names.**

PROGRAM:

```
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow =
2),
      list("green",12.3))
# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
# Show the list.print(list_data)
```

OUTPUT:

```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"
$A_Matrix
[,1] [,2] [,3]
[1,] 3 5 -2
[2,] 9 1 8
$A_Inner_list
$A_Inner_list[[1]]
[1] "green"

$A_Inner_list[[2]]
[1] 12.3
```

EXPERIMENT NO:3(b)
DATA:
Aim:Implement R Seript to access elements in the list.

Description:Elements of the list can be accessed by the index of the element in the list. In case of named lists it can also be accessed using the names
Program:
```
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow =
2),
    list("green",12.3))
# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
# Access the first element of the list.
print(list_data[1])
# Access the thrid element. As it is also a list, all its elements will be
printed.
print(list_data[3])
# Access the list element using the name of the element
.print(list_data$A_Matrix)
```
Output:
```
$`1st_Quarter`
[1] "Jan" "Feb" "Mar"

$A_Inner_list
$A_Inner_list[[1]]
[1] "green"

$A_Inner_list[[2]]
[1] 12.3
[,1] [,2] [,3]
[1,] 3 5 -2
[2,] 9 1 8
```

EXPERIMENT NO:3(c)
DATA:
Aim: Implement R Script to merge two or more lists.
Description:
<u>Merging Lists:</u>

You can merge many lists into one list by placing all the lists inside one list() function.

<u>Syntax:</u>
merge(df1, df2, by.df1, by.df2, all.df1, all.df2, sort = TRUE)

<u>Parameters:</u>
df1: one dataframe
df2: another dataframe
by.df1, by.df2: The names of the columns that are common to both df1 and df2.
all, all.df1, all.df2: Logical values that actually specify the type of merging happens.

program:
```
df1 = data.frame(StudentId = c(101:106),
              Product = c("Hindi", "English",
                        "Maths", "Science",
                        "Political Science",
                        "Physics"))
df1
```

output:
```
      StudentId                Product
1           101                   Hindi
2           102                English
3           103                   Maths
4           104                Science
5           105 Political Science
6           106                Physics
```

EXPERIMENT NO:3(d) **date:**

Aim: Implement R Script to perform matrix operation.

DESCRIPTION:

Matrices in R are a bunch of values, either real or complex numbers, arranged in a group of fixed number of rows and columns.

Operations on Matrices

There are four basic operations i.e. DMAS (Division, Multiplication, Addition, Subtraction) that can be done with matrices. Both the matrices involved in the operation should have the same number of rows and columns.

Program:

```
#Adding Both Matrices
myMatrixCAfterAdding <- myMatrixA + myMatrixB
myMatrixCAfterAdding
#Subtracting Matrix
myMatrixCAfterSubtraction <- myMatrixA - myMatrixB
myMatrixCAfterSubtraction
# R program for matrix multiplication using '*' operator
# Creating 1st Matrix
B = matrix(c(1, 2 + 3i, 5.4), nrow = 1, ncol = 3)
# Creating 2nd Matrix
C = matrix(c(2, 1i, 0.1), nrow = 1, ncol = 3)
# Printing the resultant matrix
print (B * C)
# R program for matrix divisionusing '/' operator
# Creating 1st Matrix
B = matrix(c(4, 6i, -1), nrow = 1, ncol = 3)
# Creating 2nd Matrix
C = matrix(c(2, 2i, 0), nrow = 1, ncol = 3)
# Printing the resultant matrix
print (B / C)
```

Output

```
> myMatrixCAfterAdding
[,1] [,2] [,3]
[1,] 2 8 14
[2,] 4 10 16
[3,] 6 12 18

> myMatrixCAfterSubtraction
[,1] [,2] [,3]
[1,] 0 0 0
[2,] 0 0 0
[3,] 0 0 0

[,1]    [,2]        [,3]
[1,] 2+0i -3+2i 0.54+0i

[,1] [,2]            [,3]
[1,] 2+0i 3+0i -Inf+NaNi
```

EXPERIMENT NO:4(a) **date:**

Aim: Implement R script to perform various operations on vectors.

DESCRIPTION:

Vectors contain a sequence of homogeneous types of data. Vectors are nothing but arrays as defined in other languages. Even a single object created is also stored in the form of a vector. There are various operations that can be performed on vectors in R.

Creating a vector

Vectors can be created in many ways as shown in the following example. The most usual is the use of 'c' function to combine different elements together.

Accessing vector elements

The most basic is using the '[]', subscript operator.

Modifying a vector: Vectors can be modified using different indexing variations

Deleting a vector:

Vectors can be deleted by reassigning them as NULL. To delete a vector we use the NULL operator.

Arithmetic operations: We can perform arithmetic operations between 2 vectors. These operations are performed element-wise and hence the length of both the vectors should be the same.

Sorting of Vectors: For sorting we use the sort() function which sorts the vector in ascending order by default.

PROGRAM:

```
# Use of 'c' functionto combine the values as a vector. by default the type will be double
X <- c(1, 4, 5, 2, 6, 7)
print('using c function')
print(X)
# using the seq() function to generate a sequence of continuous values with different step-size and length.
#length.out defines the length of vector.
Y <- seq(1, 10, length.out = 5)
```

```
print('using seq() function')
print(Y)
# using ':' operator to create a vector of continuous values.
Z <- 5:10
print('using colon')
print(Y)

# Accessing elements using the position number.
X <- c(2, 5, 8, 1, 2)
print('using Subscript operator')
print(X[2])
# Accessing specific values by passing a vector inside
another vector.
Y <- c(4, 5, 2, 1, 7)
print('using c function')
print(Y[c(4, 1)])

# Logical indexing
Z <- c(5, 2, 1, 4, 4, 3)
print('Logical indexing')
print(Z[Z>3])

# Creating a vector
X <- c(2, 5, 1, 7, 8, 2)
# modify a specific element
X[3] <- 11
print('Using subscript operator')
print(X)
# Modify using different logics.
X[X>9] <- 0
print('Logical indexing')
print(X)
# Modify by specifying the position or elements.
X <- X[c(5, 2, 1)]
print('using c function')
print(X)

# Creating a vector
X <- c(5, 2, 1, 6)
# Deleting a vector
X <- NULL
print('Deleted vector')
print(X)


 # Creating Vectors
 X <- c(5, 2, 5, 1, 51, 2)
 Y <- c(7, 9, 1, 5, 2, 1)
 # Addition
 Z <- X + Y
 print('Addition')
 print(Z)
 # Subtraction
 S <- X - Y
 print('Subtraction')
 print(S)
 # Multiplication
 M <- X * Y
 print('Multiplication')
 print(M)
 # Division
 D <- X / Y
 print('Division')
 print(D)

# Creating a Vector
X <- c(5, 2, 5, 1, 51, 2)
# Sort in ascending order
A <- sort(X)
print('sorting done in ascending order')
print(A)
# sort in descending order.
B <- sort(Xdtable.ing     = TRUE)
print('sorting done in descending order')
print(B)
```

OUTPUT:

```
using c function 1 4 5 2 6 7
using seq function 1.00     3.25     5.50     7.75 10.00
using colon 5     6     7     8     9 10

using Subscript operator 5
using c function 1 4
Logical indexing 5 4 4

Using subscript operator 2     5 11     7     8     2
Logical indexing 2 5 0 7 8 2
using c function 8 5 2

Deleted vector NULL

Addition 12 11     6     6 53     3
Subtraction -2 -7     4 -4 49     1
Multiplication 35     18       5       5 102       2
Division 0.7142857     0.2222222     5.0000000     0.2000000
25.5000000     2.0000000

sorting done in ascending order 1     2     2     5     5 51
sorting done in descending order 51     5     5     2     2     1
```

EXPERIMENT NO:4(b) **date:**

Aim: Implement R script to find the sum and average of given numbers using arrays.

DESCRIPTION:

- **Arrays are the R data objects which can store data in more than two dimensions. For**
- **An array is created using the array() function. It takes vectors as input and uses the values in the dim parameter to create an array.**

The sum() is a built-in R function that calculates the sum of a numeric input vector. It accepts a numeric vector as an argument and returns the sum of the vector elements. To calculate the sum of vectors in R, use the sum() function.

Syntax:

```
sum(x, na.rm = FALSE, …)
```

PROGRAM:

```
rv <- c(11, 19, 21, 18, 46)
#calculates the sum of the values
sum(rv)
```

OUTPUT:

EXPERIMENT NO:4(c) **DATE:**

Aim: Implement R script to display elements of list in reverse order.

DESCRIPTION: Lists are the R objects which contain elements of different types like − numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using list() function.

To reverse the elements of a list we use the rev() function and pass given list as argument to it.

Rev() -function returns a new list with the contents of given list in reverse order.

syntax:

rev(x)

PROGRAM:

```
# create vector with names
vec = c("sravan", "mohan", "sudheer", "radha", "vani", "mohan")
print("Original vector-1:")
print(vec)
rv = rev(vec)
print("The said vector in reverse order:")
print(rv)
```

OUTPUT:

```
[1] “Original vector-1:”
[1] “sravan”   “mohan”     “sudheer”
“radha”     “vani”      “mohan”
[1] “The said vector in reverse order:”
[1] “mohan”     “vani”      “radha”     “sudheer” “mohan”     “sravan”
```

EXPERIMENT NO:4(d) **DATE:**

Aim: Implement R to find the minimum and maximum elements in the array.

DESCRIPTION:

- **The R max function returns the maximum value of a vector or column.**
  The R min function returns the minimum value of a vector or column.
- **A function called range() is also available which returns the minimum and maximum in a two element vector.**

<u>Syntax:</u>

```
max(x)
min(x)
range(x)
```

PROGRAM:

```
w<-( 5 , 8, 3 , 9 , 2, 7, 4, 6 ,10)
min(w)
max(w)
range(w)
```

OUTPUT:

```
[1]   2
[1]   10
[1]   2   10
```

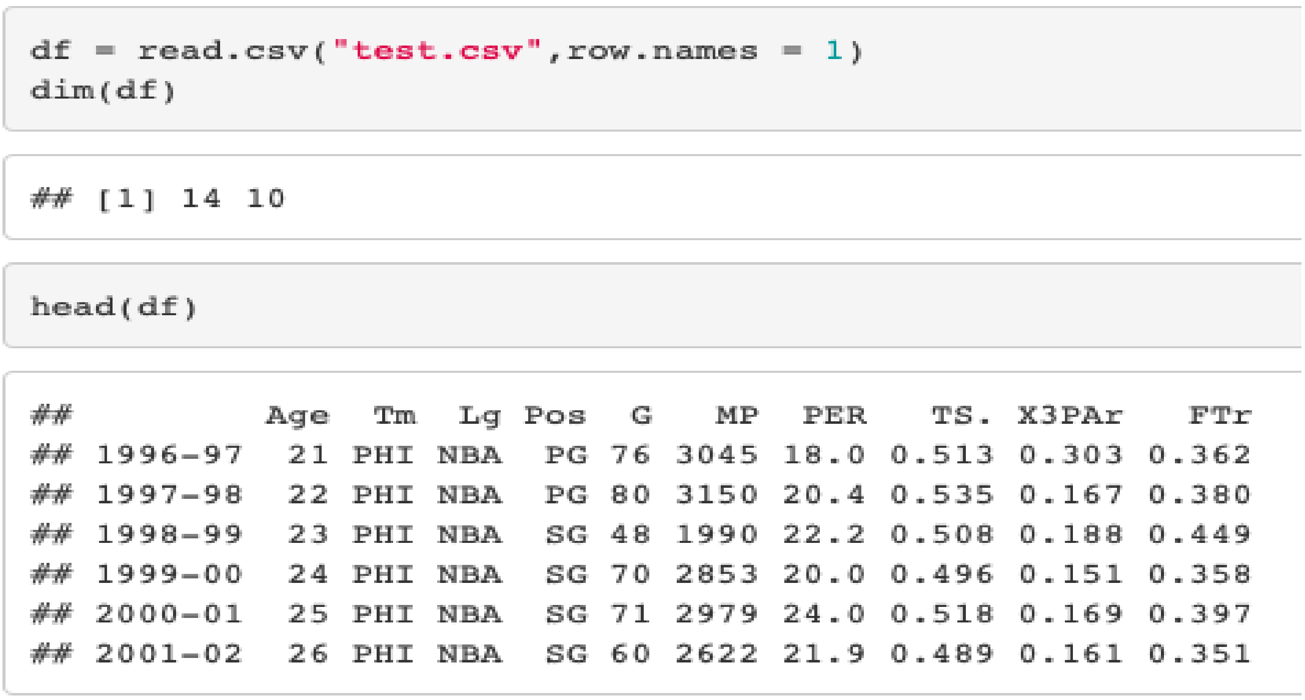
Week 5:
EXPERIMENT NO:5(b) **DATE:**
AIM: Implement R Script to extract the data from data frames.
DESCRIPTION: Data Frames are generic data objects of R which are used to store the tabular data. Data frames are considered to be the most popular data objects in R programming because it is more comfortable to analyze the data in the tabular form. Data frames can also be taught as matrices where each column of a matrix can be of the different data types. Data Frame are made up of three principal components, the data, rows, and columns.

Creating a data frame using Vectors: To create a data frame we use the data.frame() function in R. To create a data frame use data.frame() command and then pass each of the vectors you have created as arguments to the function.

- **The dimension and head of the data frame are extracted using dim() and head() functions.**
  Syntax:
  ```
  dim(df)
  head(df)
  ```
- **Rows can be extracted by their locations as :**
  ```
  # Extract the nth row
  df[n,]
  # Extract the first n rows
  df[1:n,]
  ### or ###
  df[c(1,2,…..,n),]
  ```
- **Columns can be extracted by their location as:**
  ```
  # Extract the nth column
  df[,n]
  ## Extract the first n columns
  df[,1:n]
  ```

- **Extract rows/columns by index or conditions:**

# R
## Extract 1999-2000 season.
df["1999-00",]
## Extract 1999-2000 and 2001-2002 seasons.
df[c("1999-00","2000-01"),]
df[df$varname, condition]
PROGRAM &OUTPUT:

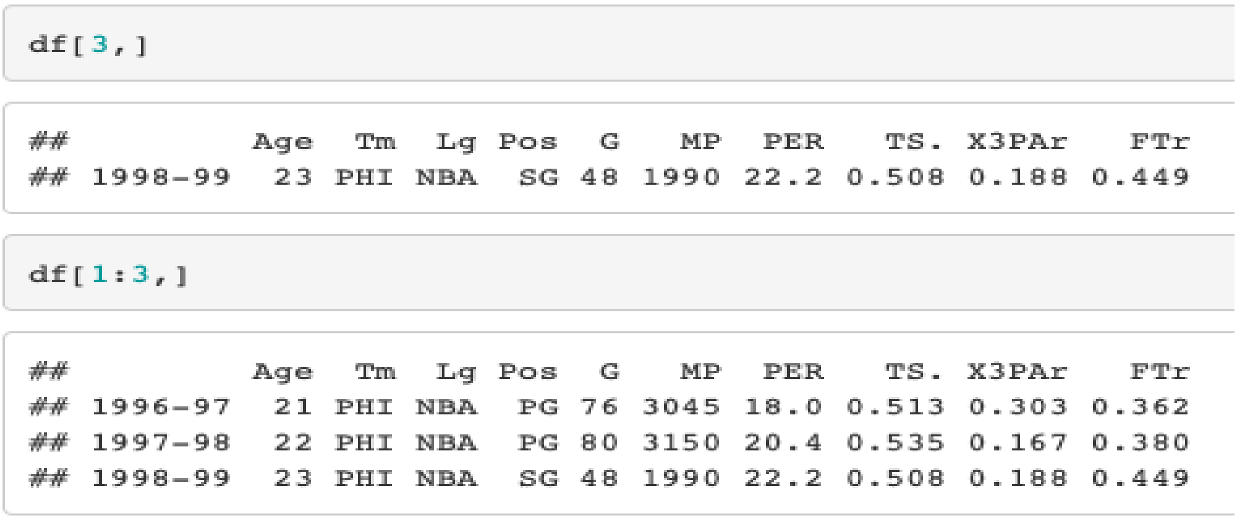


```
df = read.csv("test.csv",row.names = 1)
dim(df)
```

```
## [1] 14 10
```

```
head(df)
```

```
##         Age  Tm  Lg Pos  G   MP  PER   TS. X3PAr   FTr
## 1996-97  21 PHI NBA  PG 76 3045 18.0 0.513 0.303 0.362
## 1997-98  22 PHI NBA  PG 80 3150 20.4 0.535 0.167 0.380
## 1998-99  23 PHI NBA  SG 48 1990 22.2 0.508 0.188 0.449
## 1999-00  24 PHI NBA  SG 70 2853 20.0 0.496 0.151 0.358
## 2000-01  25 PHI NBA  SG 71 2979 24.0 0.518 0.169 0.397
## 2001-02  26 PHI NBA  SG 60 2622 21.9 0.489 0.161 0.351
```



```
df[3,]
```

```
##         Age  Tm  Lg Pos  G   MP  PER   TS. X3PAr   FTr
## 1998-99  23 PHI NBA  SG 48 1990 22.2 0.508 0.188 0.449
```

```
df[1:3,]
```

```
##         Age  Tm  Lg Pos  G   MP  PER   TS. X3PAr   FTr
## 1996-97  21 PHI NBA  PG 76 3045 18.0 0.513 0.303 0.362
## 1997-98  22 PHI NBA  PG 80 3150 20.4 0.535 0.167 0.380
## 1998-99  23 PHI NBA  SG 48 1990 22.2 0.508 0.188 0.449
```

```
df[,5]
```

```
##  [1] 76 80 48 70 71 60 82 48 75 72 65 82 57 28
```

```
df[,1:5]
```

```
##         Age  Tm  Lg   Pos  G
## 1996-97  21 PHI NBA    PG 76
## 1997-98  22 PHI NBA    PG 80
## 1998-99  23 PHI NBA    SG 48
## 1999-00  24 PHI NBA    SG 70
## 2000-01  25 PHI NBA    SG 71
## 2001-02  26 PHI NBA    SG 60
## 2002-03  27 PHI NBA    SG 82
## 2003-04  28 PHI NBA    SG 48
## 2004-05  29 PHI NBA    PG 75
## 2005-06  30 PHI NBA    PG 72
## 2006-07  31 TOT NBA SG-PG 65
## 2007-08  32 DEN NBA    SG 82
## 2008-09  33 TOT NBA PG-SG 57
## 2009-10  34 TOT NBA SG-PG 28
```

```r
df["1999-00",]
```

```
##         Age  Tm  Lg Pos  G   MP PER   TS. X3PAr   FTr
## 1999-00  24 PHI NBA  SG 70 2853  20 0.496 0.151 0.358
```

```r
df[c("1999-00","2000-01"),]
```

```
##         Age  Tm  Lg Pos  G   MP PER   TS. X3PAr   FTr
## 1999-00  24 PHI NBA  SG 70 2853  20 0.496 0.151 0.358
## 2000-01  25 PHI NBA  SG 71 2979  24 0.518 0.169 0.397
```

```r
df[df$MP > 3000,]
```

```
##         Age  Tm  Lg Pos  G   MP  PER   TS. X3PAr   FTr
## 1996-97  21 PHI NBA  PG 76 3045 18.0 0.513 0.303 0.362
## 1997-98  22 PHI NBA  PG 80 3150 20.4 0.535 0.167 0.380
## 2002-03  27 PHI NBA  SG 82 3485 21.2 0.500 0.156 0.379
## 2004-05  29 PHI NBA  PG 75 3174 23.2 0.532 0.186 0.432
## 2005-06  30 PHI NBA  PG 72 3103 25.9 0.543 0.122 0.455
## 2007-08  32 DEN NBA  SG 82 3424 20.9 0.567 0.177 0.512
```

EXPERIMENT NO:5(C) **DATE:**

AIM: Write R script to display file contents.

DESCRIPTION:

The two most common operations that can be performed on a file are:

- **Importing/Reading Files in R**
- **Exporting/Writing Files in R**

File reading in R

One of the important formats to store a file is in a text file. R provides various methods that one can read data from a text file.

- **read.delim(): This method is used for reading "tab-separated value" files (".txt"). By default, point (".") is used as decimal points.**
  <u>Syntax</u>: read.delim(file, header = TRUE, sep = "\t", dec = ".", …)
  <u>Parameters:</u>
  file: the path to the file containing the data to be read into R.
  header: a logical value. If TRUE, read.delim() assumes that your file has a header row, so row 1 is the name of each column. If that's not the case, you can add the argument header = FALSE.
  sep: the field separator character. "\t" is used for a tab-delimited file.
  dec: the character used in the file for decimal points

- **read.delim2(): This method is used for reading "tab-separated value" files (".txt"). By default, point (",") is used as decimal points.**
  <u>Syntax</u>: read.delim2(file, header = TRUE, sep = "\t", dec = ",", …)
  <u>Parameters:</u>
  file: the path to the file containing the data to be read into R.

header: a logical value. If TRUE, read.delim2() assumes that your file has a header row, so row 1 is the name of each column. If that's not the case, you can add the argument header = FALSE.
sep: the field separator character. "\t" is used for a tab-delimited file.
dec: the character used in the file for decimal points.

- file.choose(): In R it's also possible to choose a file interactively using the function file.choose(), and if you're a beginner in R programming then this method is very useful for you.

Reading one line at a time:
read_lines(): This method is used for the reading line of your own choice whether it's one or two or ten lines at a time. To use this method we have to import reader package.
Syntax: read_lines(file, skip = 0, n_max = -1L)
Parameters:
file: file path
skip: Number of lines to skip before reading data
n_max: Numbers of lines to read. If n is -1, all lines in the file will be read.

Reading a file in a table format:
Another popular format to store a file is in a tabular format. R provides various methods that one can read data from a tabular formatted data file.
read.table(): read.table() is a general function that can be used to read a file in table format. The data will be imported as a data frame.
Syntax: read.table(file, header = FALSE, sep = "", dec = ".")
Parameters:
file: the path to the file containing the data to be imported into R.
header: logical value. If TRUE, read.table() assumes that your file has a header row, so row 1 is the name of each column. If that's not the case, you can add the argument header = FALSE.

sep: the field separator character
dec: the character used in the file for decimal points.
PROGRAM:

```
myData = read.delim("geeksforgeeks.txt", header = FALSE)
print(myData)
myData = read.delim2("geeksforgeeks.txt", header = FALSE)
print(myData)
myFile = read.delim(file.choose(), header = FALSE)
print(myFile)
# R program to read one line at a time .Import the readr library
library(readr)
 # read_lines() to read one line at a time
myData = read_lines("geeksforgeeks.txt", n_max = 1)
print(myData)
# read_lines() to read two line at a time
myData = read_lines("geeksforgeeks.txt", n_max = 2)
print(myData)
myData = read.table("basic.csv")
print(myData)
```

output:

```
This is the output of the read.delim() function guys.

This is the output of the read.delim2() function guys.

This is the output of the file.choose() function guys.

"This is the output of the read lines one at a time guys."

"Hey this is the output of the read lines two at a time as per the
condition."

"do you understand? this is the next line ."

1 Name,Age,Qualification,Address

2 Amiya,18,MCA,BBS        4 Debi,23,BCA,SBP

3 Niru,23,Msc,BLS              5 Biku,56,ISC,JJP
```

EXPERIMENT NO:5(d) **DATE:**

AIM: Write R script to copy file contents from one file to another.

DESCRIPTION:

R provides many built-in functions that we can use to perform the file operations. Copy a file from one folder to another programmatically is one of the most important operations, and R provides a built-in function for that.

To copy a file in R, use the file.copy() method. The file.copy() function works in the same way as a file.append() function but with the arguments in the natural order for copying.

Syntax:

file.copy(from, to, overwrite = recursive, recursive = FALSE,copy.mode = TRUE, copy.date = FALSE)

parameters:

Copying to existing destination files is skipped unless overwrite = TRUE.

from, to: They are character vectors containing file names or paths.

overwrite: It is logical; should existing destination files be overwritten?

recursive: It is logical. If to is a directory, should directories in from being copied (and their contents)?

copy.mode: It is logical: should file permission bits be copied where possible?

copy.date: It is logical: should file dates be preserved where possible?

PROGRAM:

```
dir.create("newdir")
newDirPath <- "newdir"
files <- c("a.txt")
file.create(files)
newFilePath <- "a.txt"
file.copy(newFilePath, newDirPath)
```

OUTPUT:

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

WEEK 6:

EXPERIMENT NO:6(a) DATE:

AIM: Write an R script to find basic descriptive statistics using summary, str, quartile function on mtcars & cars datasets.

DESCRIPTION:

Descriptive statistics is the branch of statistics that focuses on describing and gaining more insight into the data in its present state. It makes the data easier to understand and also gives us knowledge about the data which is necessary to perform further analysis. Average measures like mean, median, mode, etc. are a good example of descriptive statistics.

R programming language provides us with lots of simple yet effective functions to perform descriptive statistics and gain more knowledge about our data. Summarizing the data, calculating average measures, finding out cumulative measures, summarizing rows/columns of data structures, etc. everything is possible with trivial commands. Let’s start simple with the summarizing functions `str()` and `summary()`.

- The str() function takes a single object as an argument and compactly shows us the structure of the input object. It shows us details like length, data type, names and other specifics about the components of the object.
- The summary() function also takes a single object as an argument. It then returns the averages measures like mean, median, minimum, maximum, 1st quantile, 3rd quantile, etc. for each component or variable in the object.
- the quantile() function: As the median, the first and third quartiles can be computed thanks to the quantile() function. quantile can also be computed with the quantile() function.

  Syntax: quantile(dat$Sepal.Length, quartilenumner/100)

PROGRAM:

```
  str(mtcars)

  str(cars)

summary(mtcars)

summary(cars)

quantile(mtcars$dist,0.45)

quantile(cars$speed,0.94)
```

OUTPUT:

#str(mtcars)

```
'data.frame':   32 obs. of   11 variables:
$ mpg : num   21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...

  $ cyl : num   6 6 4 6 8 6 8 4 4 6 ...

  $ disp: num   160 160 108 258 360 ...

  $ hp   : num   110 110 93 110 175 105 245 62 95 123 ...

  $ drat: num   3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92
3.92 ...

  $ wt   : num   2.62 2.88 2.32 3.21 3.44 ...

  $ qsec: num   16.5 17 18.6 19.4 17 ...

  $ vs   : num   0 0 1 1 0 1 0 1 1 1 ...

  $ am   : num   1 1 1 0 0 0 0 0 0 0 ...

  $ gear: num   4 4 4 3 3 3 3 4 4 4 ...

  $ carb: num   4 4 1 1 2 1 4 2 2 4 ...
```

#str(cars)

```
'data.frame':    50 obs. of  2 variables:
 $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
 $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

#summary(mtcars)

```
      mpg             cyl             disp
hp
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   :
52.0
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.:
96.5
 Median :19.20   Median :6.000   Median :196.3
Median :123.0
 Mean   :20.09   Mean   :6.188   Mean   :230.7
Mean   :146.7
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd
Qu.:180.0
 Max.   :33.90   Max.   :8.000   Max.   :472.0
Max.   :335.0
      drat             wt             qsec
vs
 Min.   :2.760   Min.   :1.513   Min.   :14.50
Min.   :0.0000
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st
Qu.:0.0000
 Median :3.695   Median :3.325   Median :17.71
Median :0.0000
```

```
 Mean   :3.597   Mean   :3.217   Mean   :17.85
Mean   :0.4375

 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd
Qu.:1.0000

 Max.   :4.930   Max.   :5.424   Max.   :22.90
Max.   :1.0000

       am              gear            carb

 Min.   :0.0000   Min.   :3.000   Min.   :1.000

 1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000

 Median :0.0000   Median :4.000   Median :2.000

 Mean   :0.4062   Mean   :3.688   Mean   :2.812

 3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000

 Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

#quantile(mtcars)

```
45%
167.22
```

#quantile(cars)

```
94%
 24
```

WEEK 6:

EXPERIMENT NO:6(b) **DATE:**

AIM: Write an R script to find subset of dataset by using subset (), aggregate () functions on iris dataset.

DESCRIPTION:

Aggregate() Function in R Splits the data into subsets, computes summary statistics for each subsets and returns the result in a group by form.

Aggregate() function is useful in performing all the aggregate operations like sum,count,mean, minimum and Maximum.

Syntax: aggregate(x, by, FUN, ..., simplify = TRUE, drop = TRUE)

PROGRAM:

```
agg_mean = aggregate(iris[,1:4],by=list(iris$Species),FUN=mean,
na.rm=TRUE)

print(agg_mean)

agg_sum = aggregate(iris[,1:4],by=list(iris$Species),FUN=sum,
na.rm=TRUE)

agg_sum

agg_count = aggregate(iris[,1:4],by=list(iris$Species),FUN=length)

agg_count

agg_max =
aggregate(iris[,1:4],by=list(iris$Species),FUN=max,
na.rm=TRUE)
agg_max

agg_min =
aggregate(iris[,1:4],by=list(iris$Species),FUN=min,
na.rm=TRUE)
agg_min
subset(iris, Species == "setosa")[1:5,]
```

OUTPUT:

```
    Group.1 Sepal.Length Sepal.Width Petal.Length Petal.
Width
1     setosa        5.006       3.428        1.462
 0.246
2 versicolor        5.936       2.770        4.260
 1.326
3  virginica        6.588       2.974        5.552
 2.026
```

```
  Group.1 Sepal.Length Sepal.Width Petal.Length Petal.W
idth
1     setosa        250.3       171.4         73.1
  12.3
2 versicolor        296.8       138.5        213.0
  66.3
3  virginica        329.4       148.7        277.6
 101.3
```

```
Group.1 Sepal.Length Sepal.Width Petal.Length Petal.Wid
th
1     setosa           50          50           50
    50
2 versicolor           50          50           50
    50
3  virginica           50          50           50
    50
```

```
Group.1 Sepal.Length Sepal.Width Petal.Length Petal.Wid
th
1     setosa          5.8         4.4          1.9
   0.6
```

```
2 versicolor          7.0         3.4          5.1
   1.8
3  virginica          7.9         3.8          6.9
   2.5
```

```
  Group.1 Sepal.Length Sepal.Width Petal.Length Petal.W
idth
1     setosa          4.3         2.3          1.0
   0.1
2 versicolor          4.9         2.0          3.0
   1.0
3  virginica          4.9         2.2          4.5
   1.4
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa

EXPERIMENT NO:7(a) **DATE:**

AIM: Reading different types of data sets (.txt, .csv) from Web or disk and writing in file in specific disk location.

DESCRIPTION: Usually we will be using data already in a file that we need to read into R in order to work on it. R can read data from a variety of file formats—for example, files created as text, or in Excel, SPSS or Stata. We will mainly be reading files in text format .txt or .csv (comma-separated, usually created in Excel).

To read an entire data frame directly, the external file will normally have a special form

- **The first line of the file should have a *name* for each variable in the data frame.**
- **Each additional line of the file has as its first item a *row label* and the values for each variable.**

Read CSV Files into R: If your separates the values with a , or ;, you usually are working with a .csv file.

write.csv() uses "." for the decimal point and a comma (",") for the separator.

write.csv2() uses a comma (",") for the decimal point and a semicolon (";") for the separator.

syntax :

write.csv(my_data, file = "my_data.csv")

write.csv2(my_data, file = "my_data.csv")

PROGRAM:

```
df <-
read.table("https://s3.amazonaws.com/assets.datacamp.com/blog
_assets/test.txt", header = FALSE)
```

```
print(df)

df <-
read.table("https://s3.amazonaws.com/assets.datacamp.com/blog
_assets/test.csv", header = FALSE, sep = ",")

df <-
read.csv("https://s3.amazonaws.com/assets.datacamp.com/blog_
assets/test.csv",header = FALSE)

df <-
read.csv2("https://s3.amazonaws.com/assets.datacamp.com/blog
_assets/test.csv", header= FALSE)

df

data("mtcars") # Writing mtcars data

write.table(mtcars, file = "mtcars.txt", sep = "\t",row.names = TRUE,
col.names = NA)

write.table(mtcars, file = "mtcars.txt", sep = "\t",row.names =
FALSE)
```

OUTPUT:

```
    V1 V2 V3
1   1   6   a
2   2   7   b
3   3   8   c
4   4   9  d
5  5  10   e

               V1
1 Col1,Col2,Col3
```

```
2                 1,2,3

3                 4,5,6

4                 7,8,9

5                 a,b,c

[execution complete with exit code 0]

[execution complete with exit code 0]
```

EXPERIMENT NO:7(b)
DATA:

AIM:Reading Excel data sheet in R.

DESCRIPTION: In this article, we will be discussing two different techniques to read or import an excel file in R.

Approach

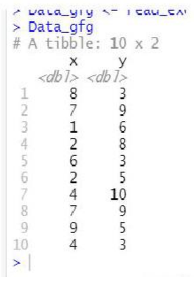
- Import module
- Pass path of the file to required function
- Read file
- Display content.

<u>read_excel():</u> function is basically used to import/read an excel file and it can only be accessed after importing of the readxl library in R language..
Syntax: read_excel(path)
<u>read.xlsx()</u>: function is imported from the xlsx library of R language and used to read/import an excel file in R language.
Syntax: read.xlsx(path)

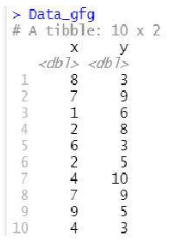
PROGRAM:

```
library(readxl)
Data_gfg <- read_excel("Data_gfg.xlsx")
Data_gfg

install.packages("xlsx")
Data_gfg <-read.xlsx('Data_gfg.xlsx')
Data_gfg
```

Output:



```
> Data_gfg <- read_ex
> Data_gfg
# A tibble: 10 x 2
       x     y
   <dbl> <dbl>
 1     8     3
 2     7     9
 3     1     6
 4     2     8
 5     6     3
 6     2     5
 7     4    10
 8     7     9
 9     9     5
10     4     3
> |
```



```
> Data_gfg
# A tibble: 10 x 2
       x     y
   <dbl> <dbl>
 1     8     3
 2     7     9
 3     1     6
 4     2     8
 5     6     3
 6     2     5
 7     4    10
 8     7     9
 9     9     5
10     4     3
```

EXPERIMENT NO:7(c)
DATA:

AIM: Reading XML data sheet in R.

DESCRIPTION:

XML which stands for Extensible Markup Language is made up of markup tags, wherein each tag illustrates the information carried by the particular attribute in the XML file. We can work with the XML files using the XML package provided by R. The package has to be explicitly installed using the following command:

install.packages("XML")

<u>Reading XML File:</u>
The XML file can be read after installing the package and then parsing it with xmlparse() function.
PROGRAM:

```
library("XML")
library("methods")
# the contents of sample.xml are parsed
data <- xmlParse(file = "sample.xml")
print(data)
```

OUTPUT:

```
1
Alia
620
IT
2
Brijesh
440
Commerce
3
Yash
600
Humanities
4
Mallika
660
IT
5
Zayn
560            IT
```

Week 8:

EXPERIMENT NO:8(a)
DATE:

AIM: Implement R Script to create a Pie chart, Bar Chart, scatter plot and Histogram (Introduction to ggplot2 graphics)

DESCRIPTION:

Types of Graphs in R:A variety of graphs is available in R, and the use is solely governed by the context. However, exploratory analysis requires the use of certain graphs in R, which must be used for analyzing data.

1. Histogram

A histogram is a graphical tool that works on a single variable. Numerous variable values are grouped into bins, and a number of values termed as the frequency are calculated. This calculation is then used to plot frequency bars in the respective beans. The height of a bar is represented by frequency.

In R, we can employ the hist() function as shown below, to generate the histogram.

2. Scatterplot:This plot is a simple chart type, but a very crucial one having tremendous significance. The chart gives the idea about a correlation amongst variables and is a handy tool in an exploratory analysis.

3.bar chart: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 bar charts. R can draw both vertical and Horizontal bars in the bar chart. In bar chart each of the bars can be given different colors.

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

4.pie chart: **A pie-chart is a representation of values as slices of a circle with different colors. The slices are labeled 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.

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

ggplot2 package in R Programming Language: also termed as Grammar of Graphics is a free, open-source, and easy-to-use visualization package widely used in **R**. It is the most powerful visualization package.

Building Blocks of layers with the grammar of graphics:

- **Data: The element is the data set itself**
- **Aesthetics: The data is to map onto the Aesthetics attributes such as x-axis, y-axis, color, fill, size, labels, alpha, shape, line width, line type**
- **Geometrics: How our data being displayed using point, line, histogram, bar, boxplot**
- **Facets: It displays the subset of the data using Columns and rows**
- **Statistics: Binning, smoothing, descriptive, intermediate**
- **Coordinates: the space between data and display using Cartesian, fixed, polar, limits**
- **Themes: Non-data link**

PROGRAM:

```
hist(trees$Height, breaks = 10, col = "orange", main = "Histogram of Tree heights", xlab = "Height Bin")

attach(trees)
plot(Girth, Height, main = "Scatterplot of Girth vs Height", xlab = "Tree Girth", ylab = "Tree Height")
abline(lm(Height ~ Girth), col = "blue", lwd = 2)

# Create the data for the chart

H <- c(7,12,28,3,41)
```

M <- c("Mar","Apr","May","Jun","Jul")

png(file = "barchart_months_revenue.png")# Give the chart file a name

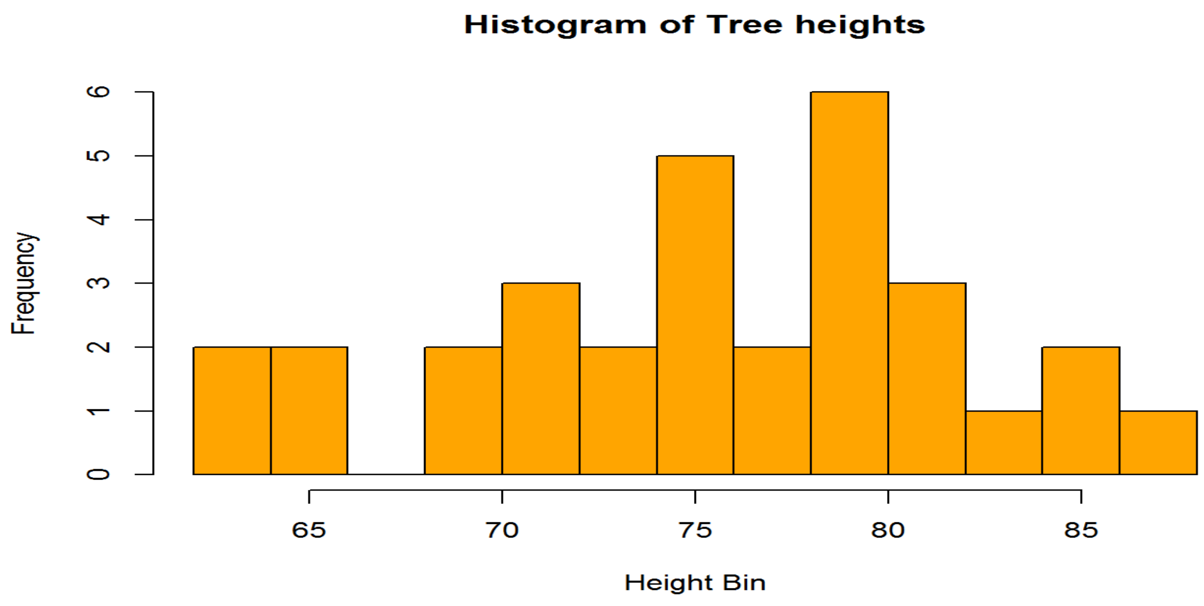
barplot(H,names.arg=M,xlab="Month",ylab="Revenue",col="blue",

main="Revenue chart",border="red")# Plot the bar chart

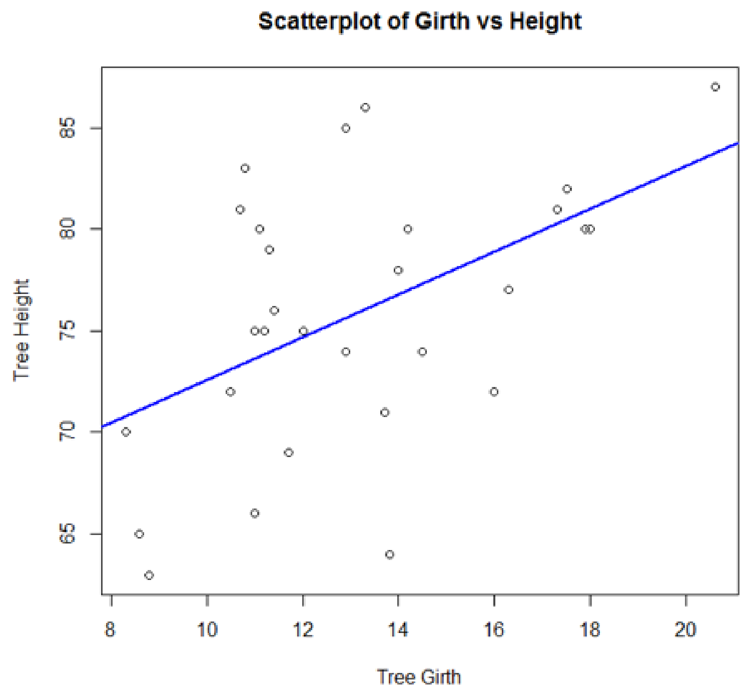
dev.off()# Save the file

```
# Create data for the pie chart
x <-    c(21, 62, 10,53)
labels <-    c("London","New York","Singapore","Mumbai")

piepercent<- round(100*x/sum(x), 1)

# Give the chart file a name.
png(file = "city_percentage_legends.jpg")

# Plot the chart.
pie(x, labels = piepercent, main = "City pie chart",col =
rainbow(length(x)))
legend("topright", c("London","New
York","Singapore","Mumbai"), cex = 0.8,
   fill = rainbow(length(x)))

# Save the file.
 Dev.off()
```

OUTP

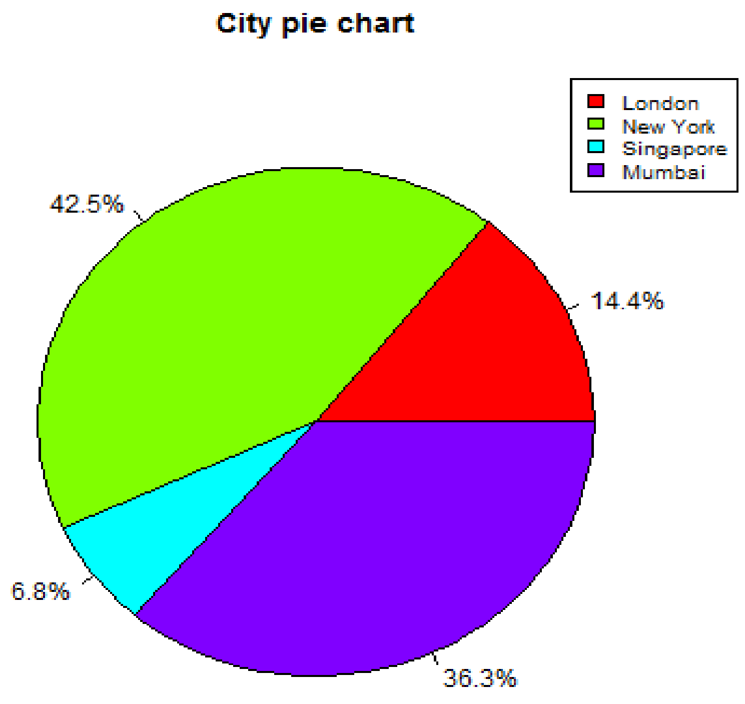


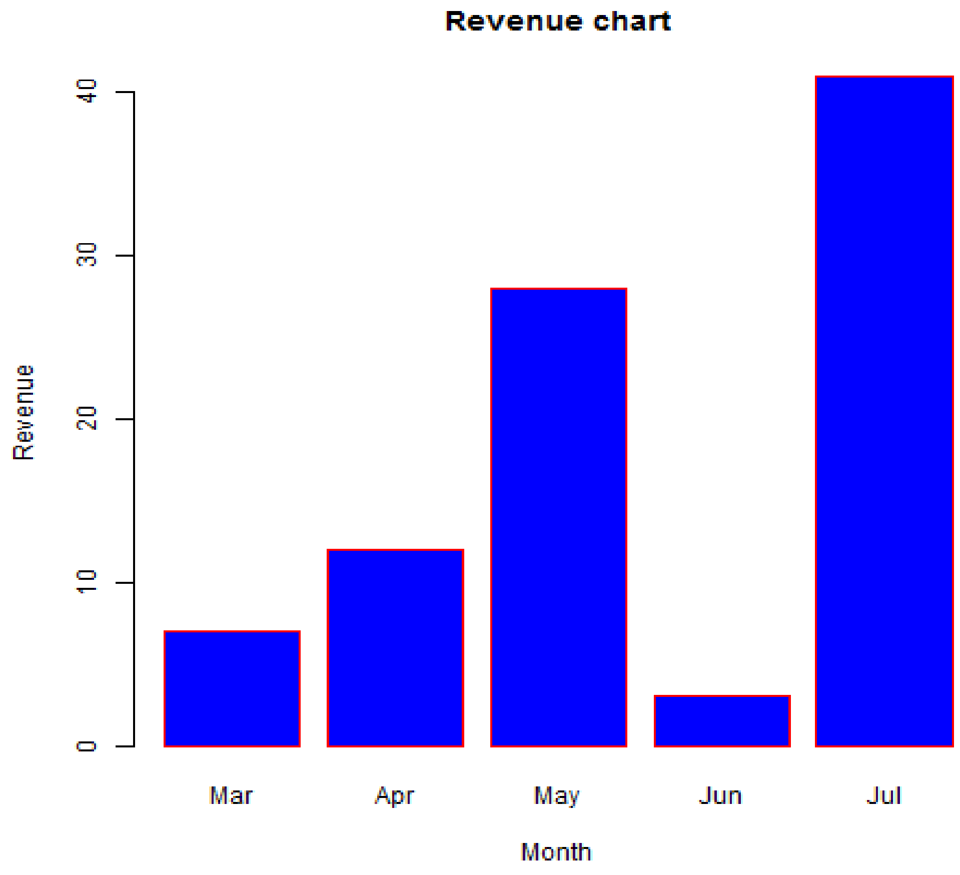
Histogram of Tree heights
Frequency
0
1
2
3
4
5
6
65
70
75
80
85
Height Bin



Scatterplot of Girth vs Height
Tree Height
65
70
75
80
85
8
10
12
14
16
18
20
Tree Girth

—


City pie chart
London
New York
Singapore
Mumbai
42.5%
14.4%
6.8%
36.3%


Revenue chart
Revenue
0
10
20
30
40
Mar
Apr
May
Jun
Jul
Month

EXPERIMENT NO:8(b)
DATE:

AIM: Implement R Script to perform mean, median, mode, range, summary, variance, standard deviation operations.

DESCRIPTION:

Mean: Calculate sum of all the values and divide it with the total number of values in the data set.

Median: The middle value of the data set.

Mode: The most occurring number in the data set. For calculating mode, there is no default function in R.

Variance: How far a set of data values are spread out from their mean.

Standard Deviation: A measure that is used to quantify the amount of variation or dispersion of a set of data values.

range(): function is used to find the lowest and highest value of the vector .range() function of a vector with NA values by using na.rm = FALSE.Highest and lowest value of the column in dataframe is also accomplished using range() function.

summary in R:The summary is a built-in R function used to produce result summaries of various model fitting functions. The summary() function implores specific methods that depend on the class of the first argument.

PROGRAM:

```
  x<-c(1,2,3,4,5,1,2,3,1,2,4,5,2,3,1,1,2,3,5,6)

  mean.result=mean(x)

print(mean.result)
```

```
median.result=median(x)

print(median.result)

mode.result=mode(x)

print(mode.result)

varience.result=var(x)

print(varience.result)

sd.result=sqrt(var(x))

print(sd.result)

range(x,na.rm=TRUE)

summary(x)

print(summary(x))
```

OUTPUT:

```
[1] 2.8

[1] 2.5

[1] 1

[1] 2.484211

[1] 1.576138

[1] 1   6

[1] Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    1.75    2.50    2.80    4.00    6.00
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    1.75    2.50    2.80    4.00    6.00
```

WEEK 9:
EXPERIMENT NO:9(a)
DATE:
AIM: Implement R Seript to perform Normal, Binomial distributions
DESCRIPTION: Binomial distribution in R is a probability distribution used in statistics. The outcomes from different trials are independent. Binomial distribution helps us to find the individual probabilities as well as cumulative probabilities over a certain range.
dbinom() Function:
This function is used to find probability at a particular value for a data that follows binomial distribution i.e. it finds: P(X = k)
Syntax:dbinom(k, n, p)

pbinom() Function:The function pbinom() is used to find the cumulative probability of a data following binomial distribution till a given value ie it finds:P(X = k)

Syntax: `pbinom(k, n, p)`

qbinom() Function:This function is used to find the nth quantile, that is if P(x <= k) is given, it finds k.
Syntax:qbinom(P, n, p)

rbinom() Function:This function generates n random variables of a particular probability.

Syntax:rbinom(n, N, p)

Normal Distribution is a probability function used in statistics that tells about how the data values are distributed. It is generally observed that data distribution is normal when there is a random collection of data from independent sources.

dnorm() function in R programming measures density function of distribution.

Syntax:dnorm(x, mean, sd)

pnorm() function is the cumulative distribution function which measures the probability that a random number X takes a value less than or equal to x

syntax:pnorm(x, mean, sd)

qnorm() function is the inverse of pnorm() function. It takes the probability value and gives output which corresponds to the probability value. It is useful in finding the percentiles of a normal distribution.

Syntax: qnorm(p, mean, sd)

rnorm() function in R programming is used to generate a vector of random numbers which are normally distributed. Syntax:rnorm(x, mean, sd)

PROGRAM:

```
dbinom(3, size = 13, prob = 1 / 6)

probabilities <- dbinom(x = c(0:10), size = 10, prob = 1 / 6)

data.frame(x, probs)

pbinom(3, size = 13, prob = 1 / 6)

qbinom(0.8419226, size = 13, prob = 1 / 6)



rbinom(8, size = 13, prob = 1 / 6)

# creating a sequence of values between -15 to 15 with a
difference of 0.1
x = seq(-15, 15, by=0.1)
y = dnorm(x, mean(x), sd(x))
png(file="dnormExample.png" # output to be present as
```

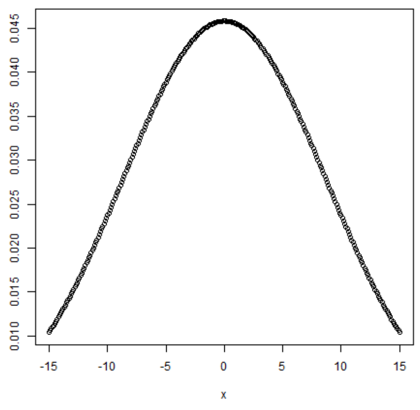
PNG file
plot(x, y) # Plot the graph.

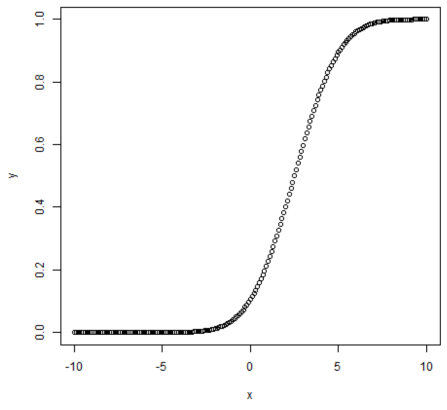
```
# creating a sequence of values between -10 to
10 with a difference of 0.1
x <- seq(-10, 10, by=0.1)
y <- pnorm(x, mean = 2.5, sd = 2)
png(file="pnormExample.png")# output to be
present as PNG file
plot(x, y) # Plot the graph.

# Create a sequence of probability values
incrementing by 0.02.
x <- seq(0, 1, by = 0.02)
y <- qnorm(x, mean(x), sd(x))# output to be
present as PNG file
png(file = "qnormExample.png")
plot(x, y) # Plot the graph.

# Create a vector of 1000 random numberswith
mean=90 and sd=5
x <- rnorm(10000, mean=90, sd=5)
png(file = "rnormExample.png")# output to be
present as PNG file
hist(x, breaks=50) # Create the histogram with
50 bars
```

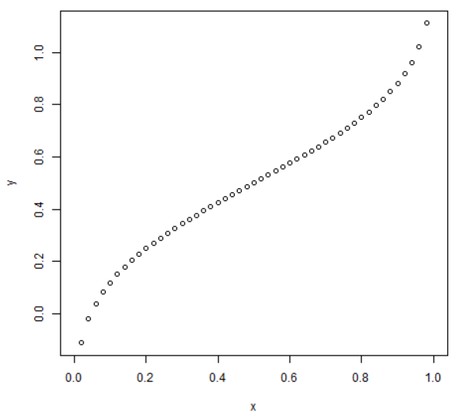
OUTPUT:

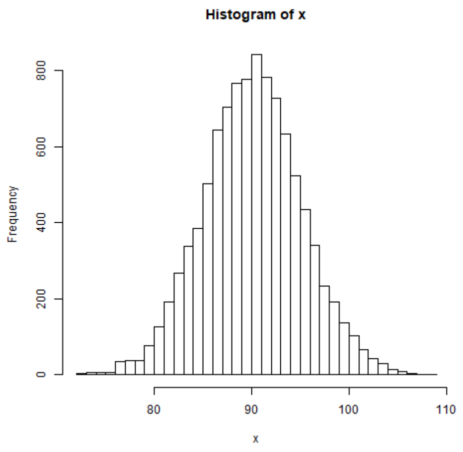
```
0.2138454
data.frame(probabilities)
     probabilities
```

```
1     1.615056e-01
2     3.230112e-01
3     2.907100e-01
4     1.550454e-01
5     5.426588e-02
6     1.302381e-02
7     2.170635e-03
8     2.480726e-04
9     1.860544e-05
10    8.269086e-07
11    1.653817e-08

[1] 0.8419226

[1] 3

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






y
x
0.0
0.2
0.4
0.6
0.8
1.0


Histogram of x
Frequency
x
0
200
400
600
800
80
90
100
110

EXPERIMENT NO:9(b)
DATE:
AIM: Implement R Script to perform correlation, Linear and multiple regression.
DESCRIPTION:
Correlation: is a statistical measure that indicates how strongly two variables are related. It involves the relationship between multiple variables as well. Generally, it lies between -1 and +1. It is a scaled version of covariance and provides the direction and strength of a relationship.
R Language provides two methods to calculate the pearson correlation coefficient. By using the functions cor() or cor.test() it can be calculated. It can be noted that cor() computes the correlation coefficient whereas cor.test() computes the test for association or correlation between paired samples.
Syntax: cor(x, y, method = "pearson")
cor.test(x, y, method = "pearson")
Linear regression :is used to predict the value of an outcome variable Y based on one or more input predictor variables X. The aim is to establish a linear relationship (a mathematical formula) between the predictor variable(s) and the response variable, so that, we can use this formula to estimate the value of the response Y, when only the predictors (Xs) values are known.
lm() Function:This function creates the relationship model between the predictor and the response variable.
Syntax: lm(formula,data)
predict():
Syntax:predict(object, newdata)
Multiple Linear Regression : It is the most common form of Linear Regression. Multiple Linear Regression basically describes how a single response variable Y depends linearly on a number of predictor variables.
PROGRAM:

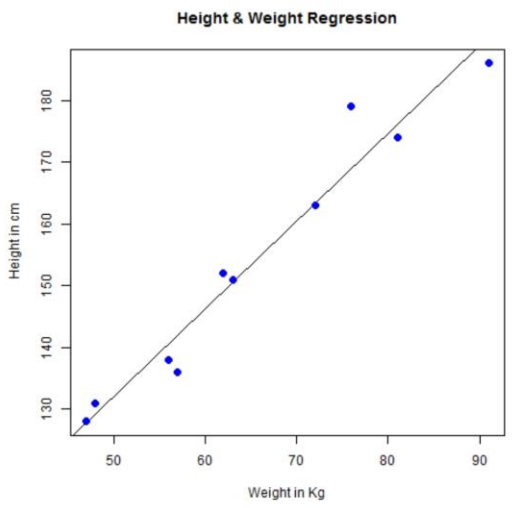
```
 x<-c(1,2,3,4,5,6,7)
```

```
  y<-(1,3,6,2,7,4,5)
result=cor(x,y,method=”pearson”)
cat(“pearson correlation coefficient is :”,result)

result1=cor.test(x,y,method=”pearson”)
cat(“pearson correlation coefficient is :”,result1)

x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
relation <- lm(y~x)
print(relation)
print(summary(relation))
a <- data.frame(x = 170)
result <- predict(relation,a)
print(result)
png(file = "linearregression.png")
plot(y,x,col = "blue",main = "Height & Weight Regression",
abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab =
"Height in cm")
input <- mtcars[,c("mpg","disp","hp","wt")]
model <- lm(mpg~disp+hp+wt, data = input) # create relation ship
model
print(model)
cat("# # # # The Coefficient Values # # # ","\n")
a <- coef(model)[1]
  print(a)
Xdisp <- coef(model)[2]
Xhp <- coef(model)[3]
Xwt <- coef(model)[4]
print(Xdisp)
  print(Xhp)
print(Xwt)
```

OUTPUT:

```
Pearson correlation coefficient is: 0.5357143
```

```
Pearson's product-moment correlation

data:    x and y
t = 1.4186, df = 5, p-value = 0.2152
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
  -0.3643187    0.9183058
sample estimates:
          cor
0.5357143
Call: lm(formula = y ~ x)
Coefficients: (Intercept) x -38.4551 0.6746
Call:
lm(formula = y ~ x)
Residuals:
Min 1Q Median 3Q Max
-6.3002 -1.6629 0.0412 1.8944 3.9775
Coefficients:
Estimate Std. Error t value Pr(>|t|) (Intercept) -38.45509 8.04901
-4.778 0.00139 ** x 0.67461 0.05191 12.997 1.16e-06 ***
---
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 3.253 on 8 degrees of freedom
Multiple R-squared: 0.9548, Adjusted R-squared: 0.9491
F-statistic: 168.9 on 1 and 8 DF, p-value: 1.164e-06

1
76.22869
```



Call: lm(formula = mpg ~ disp + hp + wt, data = input)
Coefficients: (Intercept) disp hp wt 37.105505 -0.000937 -0.031157
-3.800891

```
# # # # The Coefficient Values # # #
  (Intercept)
  37.10551
disp
-0.0009370091
hp
-0.03115655
wt
-3.800891
```