Practical No.: 1

Aim: To implement basic commands, data types and operation in R.

Theory: 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.

Features of R programming

1. It is a simple and effective programming language which has been well developed.
2. It is data analysis software.
3. It is a well-designed, easy, and effective language which has the concepts of user-defined, looping, conditional, and various I/O facilities.
4. For different types of calculation on arrays, lists and vectors, R contains a suite of operators.
5. It is an open-source, powerful, and highly extensible software.
6. It provides highly extensible graphical techniques.

There are the following types of operators used in R:

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operators
5. Miscellaneous Operators

- **Arithmetic Operators**

Arithmetic operators are the symbols which are used to represent arithmetic math operations. The operators act on each and every element of the vector. There are various arithmetic operators which are supported by R.

S. No	Operator	Description
1.	+	This operator is used to add two vectors in R.
2.	-	This operator is used to divide a vector from another one.
3.	*	This operator is used to multiply two vectors with each other.
4.	/	This operator divides the vector from another one.
5.	%%	This operator is used to find the remainder of the first vector with the second vector.

6.	%/%	This operator is used to find the division of the first vector with the second(quotient).
7.	^	This operator raised the first vector to the exponent of the second vector.

- **Relational Operators**

A relational operator is a symbol which defines some kind of relation between two entities. 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. There are the following relational operators which are supported by R:

S. No	Operator	Description
1.	>	This operator will return TRUE when every element in the first vector is greater than the corresponding element of the second vector.
2.	<	This operator will return TRUE when every element in the first vector is less then the corresponding element of the second vector.
3.	<=	This operator will return TRUE when every element in the first vector is less than or equal to the corresponding element of another vector.
4.	>=	This operator will return TRUE when every element in the first vector is greater than or equal to the corresponding element of another vector.
5.	==	This operator will return TRUE when every element in the first vector is equal to the corresponding element of the second vector.
6.	!=	This operator will return TRUE when every element in the first vector is not equal to the corresponding element of the second vector.

- **Logical Operators**

The logical operators allow a program to make a decision on the basis of multiple conditions. In the program, each operand is considered as a condition which can be evaluated to a false or true value. The value of the conditions is used to determine the overall value of the op1 **operator** op2.

S. No	Operator	Description
1.	&	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.
2.	|	This operator is called the Logical OR operator. This operator takes the first element of both the vector and returns TRUE if one of them is TRUE.
3.	!	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.
4.	&&	This operator takes the first element of both the vector and gives TRUE as a result, only if both are TRUE.
5.	||	This operator takes the first element of both the vector and gives the result TRUE, if one of them is true.

- **Assignment Operators**

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 the following types of assignment

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.

- **Miscellaneous Operators**

Miscellaneous operators are used for a special and specific purpose. These operators are not used for general mathematical or logical computation. There are the following miscellaneous operators which are supported in R

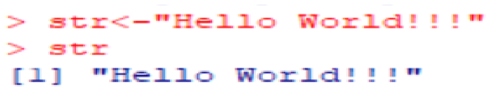
S. No	Operator	Description
1.	:	The colon operator is used to create the series of numbers in sequence for a vector.
2.	%in%	This is used when we want to identify if an element belongs to a vector.
3.	%*%	It is used to multiply a matrix with its transpose.

Data Types in R Programming

Data type	Example	Description
Logical	True, False	It is a special data type for data with only two possible values which can be construed as true/false.
Numeric	12,32,112,5432	Decimal value is called numeric in R, and it is the default computational data type.
Integer	3, 66, 2346	It stores the value as an integer,
Complex	Z=1+2i, t=7+3i	A complex value in R is defined as the pure imaginary value i.
Character	'a', '"good"', "TRUE", '35.4'	In R programming, a character is used to represent string values. Set of character.
Raw	‘hello’ is converted into 68 65 6c 6c 6f	A raw data type is used to holds raw bytes.

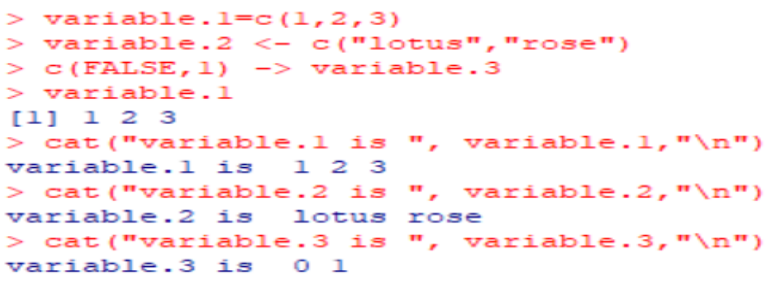
Program:

1. Print hello world.



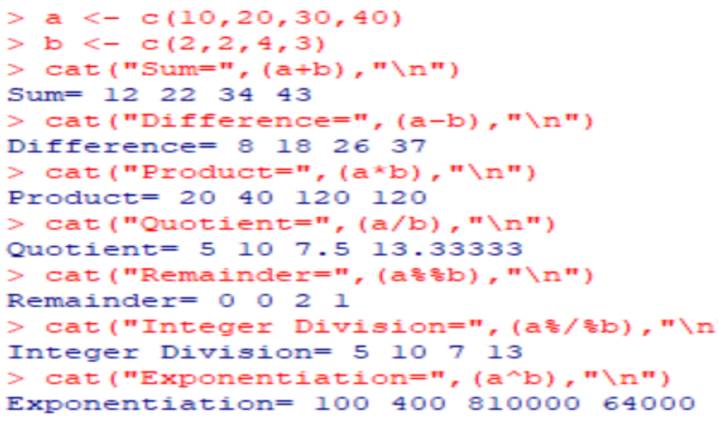
```
> str<-"Hello World!!!"
> str
[1] "Hello World!!!"
```

2. Various way to declared variable declaration



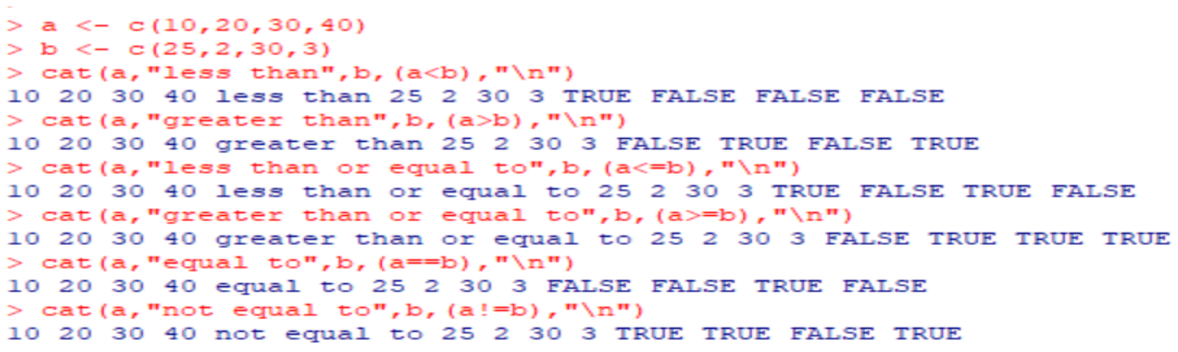
```
> variable.1=c(1,2,3)
> variable.2 <- c("lotus","rose")
> c(FALSE,1) -> variable.3
> variable.1
[1] 1 2 3
> cat("variable.1 is ", variable.1,"\n")
variable.1 is  1 2 3
> cat("variable.2 is ", variable.2,"\n")
variable.2 is  lotus rose
> cat("variable.3 is ", variable.3,"\n")
variable.3 is  0 1
```

3. Perform arithmetic operations



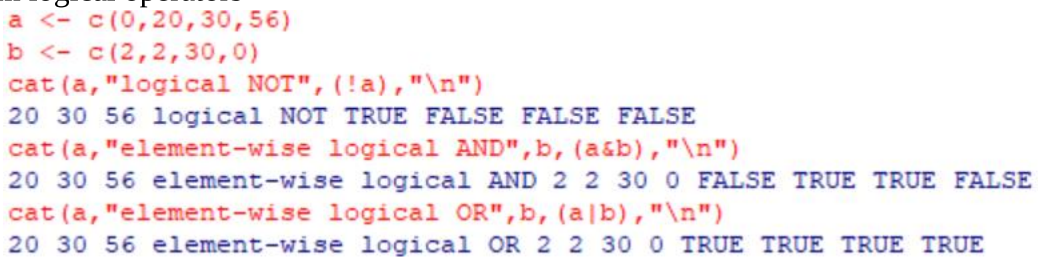
```
> a <- c(10,20,30,40)
> b <- c(2,2,4,3)
> cat("Sum=",(a+b),"\n")
Sum= 12 22 34 43
> cat("Difference=",(a-b),"\n")
Difference= 8 18 26 37
> cat("Product=",(a*b),"\n")
Product= 20 40 120 120
> cat("Quotient=",(a/b),"\n")
Quotient= 5 10 7.5 13.33333
> cat("Remainder=",(a%%b),"\n")
Remainder= 0 0 2 1
> cat("Integer Division=",(a%/%b),"\n")
Integer Division= 5 10 7 13
> cat("Exponentiation=",(a^b),"\n")
Exponentiation= 100 400 810000 64000
```

4. Perform relational operators

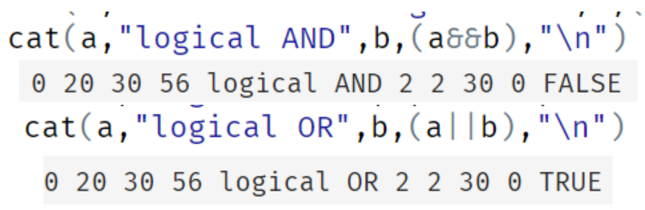


```
> a <- c(10,20,30,40)
> b <- c(25,2,30,3)
> cat(a,"less than",b,(a<b),"\n")
10 20 30 40 less than 25 2 30 3 TRUE FALSE FALSE FALSE
> cat(a,"greater than",b,(a>b),"\n")
10 20 30 40 greater than 25 2 30 3 FALSE TRUE FALSE TRUE
> cat(a,"less than or equal to",b,(a<=b),"\n")
10 20 30 40 less than or equal to 25 2 30 3 TRUE FALSE TRUE FALSE
> cat(a,"greater than or equal to",b,(a>=b),"\n")
10 20 30 40 greater than or equal to 25 2 30 3 FALSE TRUE TRUE TRUE
> cat(a,"equal to",b,(a==b),"\n")
10 20 30 40 equal to 25 2 30 3 FALSE FALSE TRUE FALSE
> cat(a,"not equal to",b,(a!=b),"\n")
10 20 30 40 not equal to 25 2 30 3 TRUE TRUE FALSE TRUE
```

5. Perform logical operators

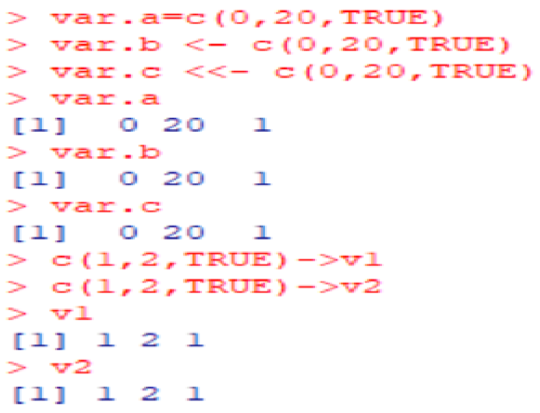


```
a <- c(0,20,30,56)
b <- c(2,2,30,0)
cat(a,"logical NOT",(!a),"\n")
20 30 56 logical NOT TRUE FALSE FALSE FALSE
cat(a,"element-wise logical AND",b,(a&b),"\n")
20 30 56 element-wise logical AND 2 2 30 0 FALSE TRUE TRUE FALSE
cat(a,"element-wise logical OR",b,(a|b),"\n")
20 30 56 element-wise logical OR 2 2 30 0 TRUE TRUE TRUE TRUE
```



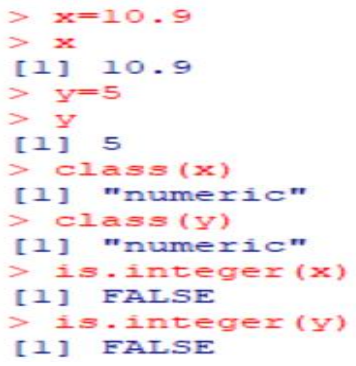
```
cat(a,"logical AND",b,(a&&b),"\n")
0 20 30 56 logical AND 2 2 30 0 FALSE
cat(a,"logical OR",b,(a||b),"\n")
0 20 30 56 logical OR 2 2 30 0 TRUE
```

6. Perform assignment operators



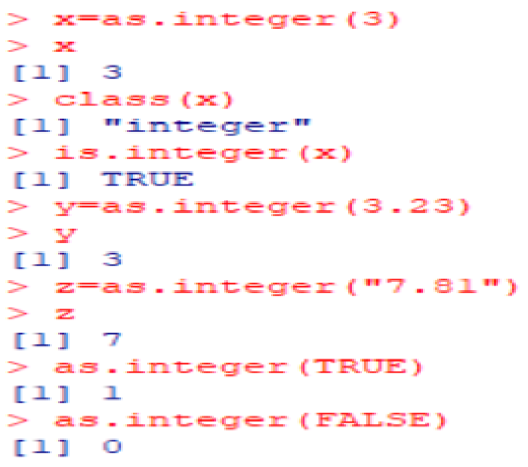
```
> var.a=c(0,20,TRUE)
> var.b <- c(0,20,TRUE)
> var.c <<- c(0,20,TRUE)
> var.a
[1]  0 20  1
> var.b
[1]  0 20  1
> var.c
[1]  0 20  1
> c(1,2,TRUE)->v1
> c(1,2,TRUE)->v2
> v1
[1] 1 2 1
> v2
[1] 1 2 1
```

7. Demonstrate Numeric data types



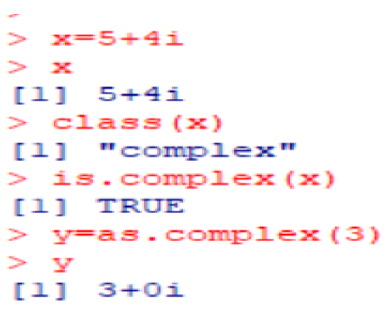
```
> x=10.9
> x
[1] 10.9
> y=5
> y
[1] 5
> class(x)
[1] "numeric"
> class(y)
[1] "numeric"
> is.integer(x)
[1] FALSE
> is.integer(y)
[1] FALSE
```

8. Demonstrate Integer data types



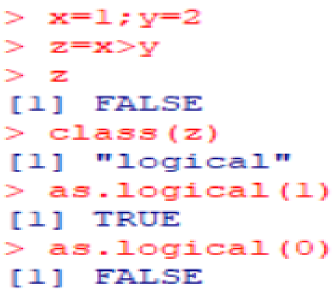
```
> x=as.integer(3)
> x
[1] 3
> class(x)
[1] "integer"
> is.integer(x)
[1] TRUE
> y=as.integer(3.23)
> y
[1] 3
> z=as.integer("7.81")
> z
[1] 7
> as.integer(TRUE)
[1] 1
> as.integer(FALSE)
[1] 0
```

9. Demonstrate Complex data types



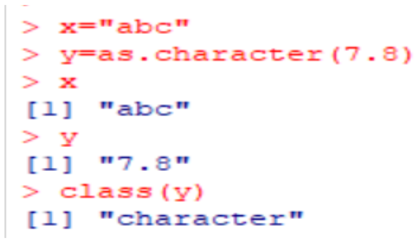
```
> x=5+4i
> x
[1] 5+4i
> class(x)
[1] "complex"
> is.complex(x)
[1] TRUE
> y=as.complex(3)
> y
[1] 3+0i
```

10. Demonstrate Logical data types



```
> x=1;y=2
> z=x>y
> z
[1] FALSE
> class(z)
[1] "logical"
> as.logical(1)
[1] TRUE
> as.logical(0)
[1] FALSE
```

11. Demonstrate Character data types



```
> x="abc"
> y=as.character(7.8)
> x
[1] "abc"
> y
[1] "7.8"
> class(y)
[1] "character"
```

Conclusion: We have successfully learned about the basics of R language, and also execute or implement some of the basic program successfully.

Practical No.: 2

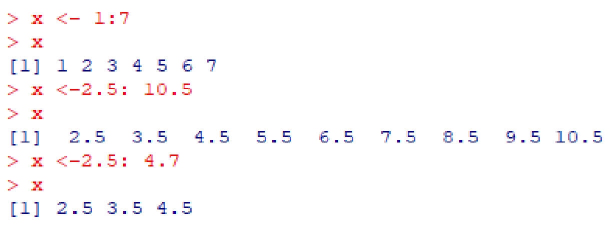
Aim: Data analysis using vectors, lists, factors and data frames in R.

Theory:

- **Lists:** In R, lists are the second type of vector. Lists are the objects of R which contain elements of different types such as number, vectors, string and another list inside it. It can also contain a function or a matrix as its elements. A list is a data structure which has components of mixed data types. We can say, a list is a generic vector which contains other objects.
- **Matrix:** In R, a two-dimensional rectangular data set is known as a matrix. A matrix is created with the help of the vector input to the matrix function. On R matrices, we can perform addition, subtraction, multiplication, and division operation. In the R matrix, elements are arranged in a fixed number of rows and columns. The matrix elements are the real numbers. In R, we use matrix function, which can easily reproduce the memory representation of the matrix. In the R matrix, all the elements must share a common basic type.
- **Data Frame:** A data frame is a two-dimensional array-like structure or a table in which a column contains values of one variable, and rows contains one set of values from each column. A data frame is a special case of the list in which each component has equal length. A data frame is used to store data table and the vectors which are present in the form of a list in a data frame, are of equal length. In a simple way, it is a list of equal length vectors. A matrix can contain one type of data, but a data frame can contain different data types such as numeric, character, factor, etc.
- **Factors:** The factor is a data structure which is used for fields which take only predefined finite number of values. These are the variable which takes a limited number of different values. These are the data objects which are used to categorize the data and to store it on multiple levels. It can store both integers and strings values, and are useful in the column that has a limited number of unique values.

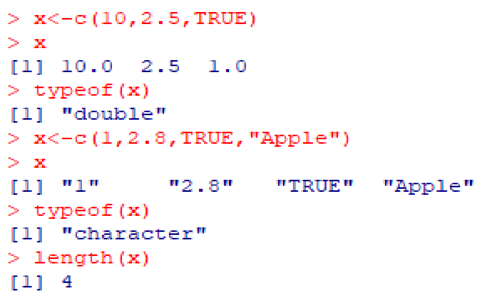
Program:

1. Creating Vector by using colon (:) operator.



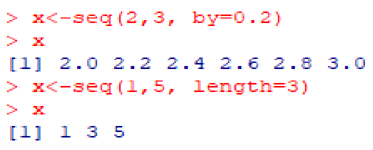
```
> x <- 1:7
> x
[1] 1 2 3 4 5 6 7
> x <-2.5: 10.5
> x
[1]  2.5  3.5  4.5  5.5  6.5  7.5  8.5  9.5 10.5
> x <-2.5: 4.7
> x
[1] 2.5 3.5 4.5
```

2. Creating Vector by using c() function.



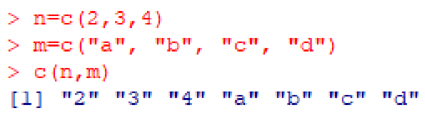
```
> x<-c(10,2.5,TRUE)
> x
[1] 10.0  2.5  1.0
> typeof(x)
[1] "double"
> x<-c(1,2.8,TRUE,"Apple")
> x
[1] "1"     "2.8"   "TRUE"  "Apple"
> typeof(x)
[1] "character"
> length(x)
[1] 4
```

3. Creating Vector by using sequence (seq) function.



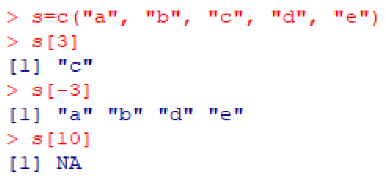
```
> x<-seq(2,3, by=0.2)
> x
[1] 2.0 2.2 2.4 2.6 2.8 3.0
> x<-seq(1,5, length=3)
> x
[1] 1 3 5
```

4. Combining vectors.

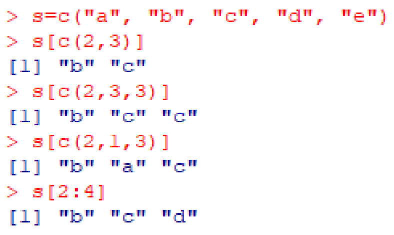


```
> n=c(2,3,4)
> m=c("a", "b", "c", "d")
> c(n,m)
[1] "2" "3" "4" "a" "b" "c" "d"
```

5. Accessing vector element by using Integer vector as index.

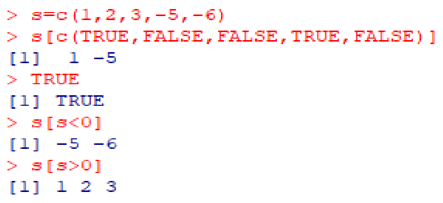


```
> s=c("a", "b", "c", "d", "e")
> s[3]
[1] "c"
> s[-3]
[1] "a" "b" "d" "e"
> s[10]
[1] NA
```



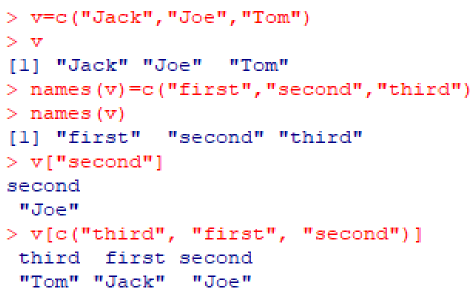
```
> s=c("a", "b", "c", "d", "e")
> s[c(2,3)]
[1] "b" "c"
> s[c(2,3,3)]
[1] "b" "c" "c"
> s[c(2,1,3)]
[1] "b" "a" "c"
> s[2:4]
[1] "b" "c" "d"
```

6. Accessing vector element by using logical vector as index.



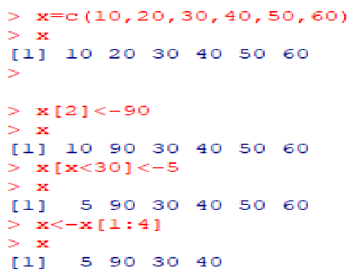
```
> s=c(1,2,3,-5,-6)
> s[c(TRUE,FALSE,FALSE,TRUE,FALSE)]
[1]  1 -5
> TRUE
[1] TRUE
> s[s<0]
[1] -5 -6
> s[s>0]
[1] 1 2 3
```

7. Accessing vector element by using character vector as index.



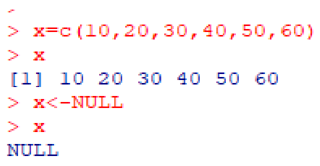
```
> v=c("Jack","Joe","Tom")
> v
[1] "Jack" "Joe"  "Tom"
> names(v)=c("first","second","third")
> names(v)
[1] "first"  "second" "third"
> v["second"]
second
 "Joe"
> v[c("third", "first", "second")]
 third  first second
 "Tom" "Jack"  "Joe"
```

8. Modifying vectors.



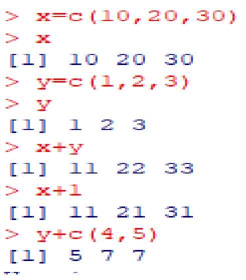
```
> x=c(10,20,30,40,50,60)
> x
[1] 10 20 30 40 50 60
>

> x[2]<-90
> x
[1] 10 90 30 40 50 60
> x[x<30]<-5
> x
[1]  5 90 30 40 50 60
> x<-x[1:4]
> x
[1]  5 90 30 40
```

9. Deleting vectors.



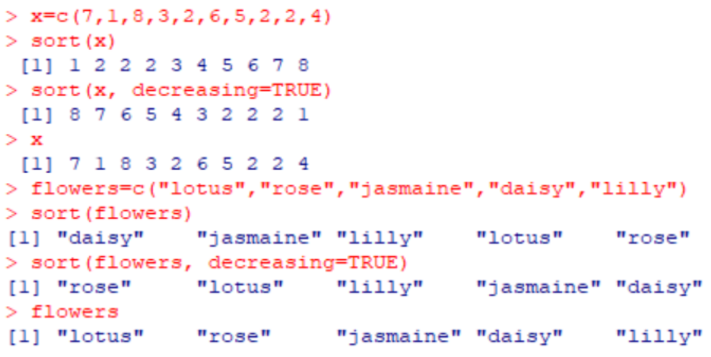
```
> x=c(10,20,30,40,50,60)
> x
[1] 10 20 30 40 50 60
> x<-NULL
> x
NULL
```

10. Vector arithmetic and recycling.



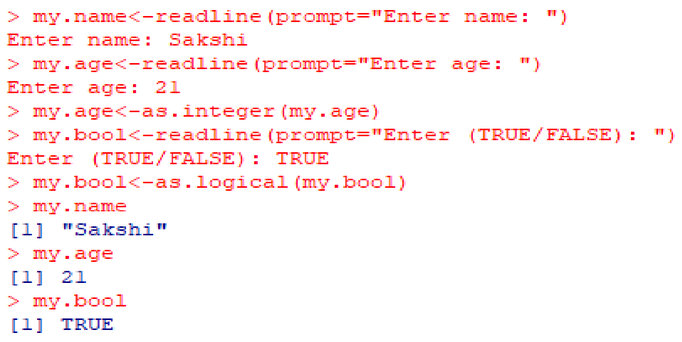
```
> x=c(10,20,30)
> x
[1] 10 20 30
> y=c(1,2,3)
> y
[1] 1 2 3
> x+y
[1] 11 22 33
> x+1
[1] 11 21 31
> y+c(4,5)
[1] 5 7 7
```

11. Vector element sorting.



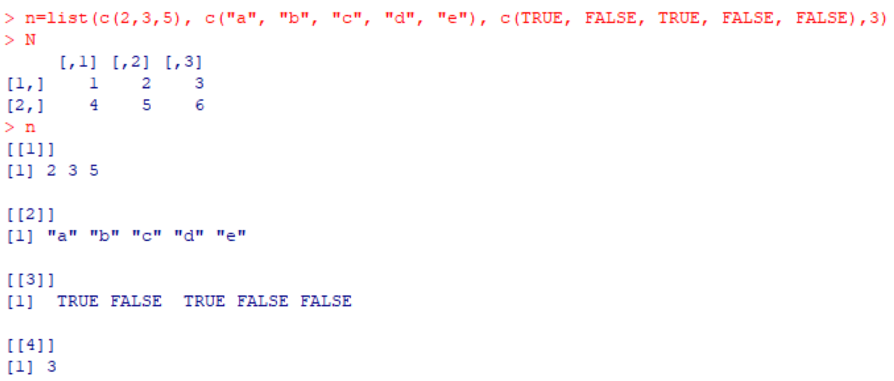
```
> x=c(7,1,8,3,2,6,5,2,2,4)
> sort(x)
 [1] 1 2 2 2 3 4 5 6 7 8
> sort(x, decreasing=TRUE)
 [1] 8 7 6 5 4 3 2 2 2 1
> x
 [1] 7 1 8 3 2 6 5 2 2 4
> flowers=c("lotus","rose","jasmaine","daisy","lilly")
> sort(flowers)
[1] "daisy"    "jasmaine" "lilly"    "lotus"    "rose"
> sort(flowers, decreasing=TRUE)
[1] "rose"     "lotus"    "lilly"    "jasmaine" "daisy"
> flowers
[1] "lotus"    "rose"     "jasmaine" "daisy"    "lilly"
```

12. Reading vectors.



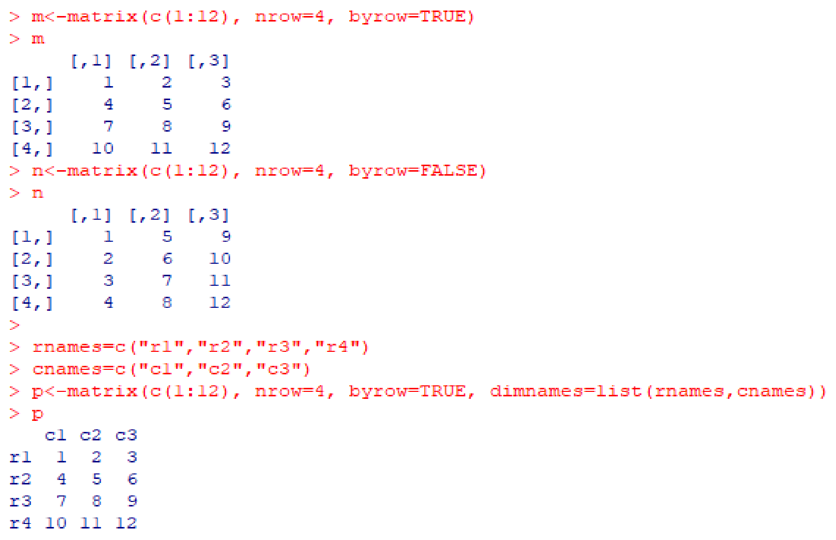
```
> my.name<-readline(prompt="Enter name: ")
Enter name: Sakshi
> my.age<-readline(prompt="Enter age: ")
Enter age: 21
> my.age<-as.integer(my.age)
> my.bool<-readline(prompt="Enter (TRUE/FALSE): ")
Enter (TRUE/FALSE): TRUE
> my.bool<-as.logical(my.bool)
> my.name
[1] "Sakshi"
> my.age
[1] 21
> my.bool
[1] TRUE
```

13. Creating Lists.



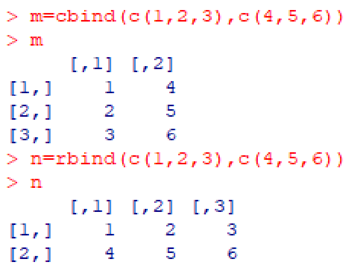
```
> n=list(c(2,3,5), c("a", "b", "c", "d", "e"), c(TRUE, FALSE, TRUE, FALSE, FALSE),3)
> N
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> n
[[1]]
[1] 2 3 5

[[2]]
[1] "a" "b" "c" "d" "e"

[[3]]
[1]  TRUE FALSE  TRUE FALSE FALSE

[[4]]
[1] 3
```

14. Creating Matrices.



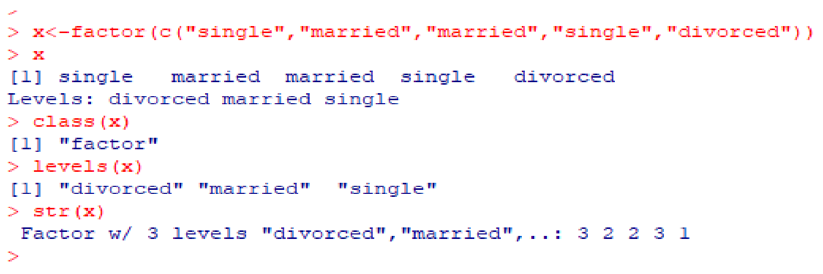
```
> m<-matrix(c(1:12), nrow=4, byrow=TRUE)
> m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> n<-matrix(c(1:12), nrow=4, byrow=FALSE)
> n
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
>
> rnames=c("r1","r2","r3","r4")
> cnames=c("c1","c2","c3")
> p<-matrix(c(1:12), nrow=4, byrow=TRUE, dimnames=list(rnames,cnames))
> p
   c1 c2 c3
r1  1  2  3
r2  4  5  6
r3  7  8  9
r4 10 11 12
```

15. Creating Matrices by using cbind() and rbind().



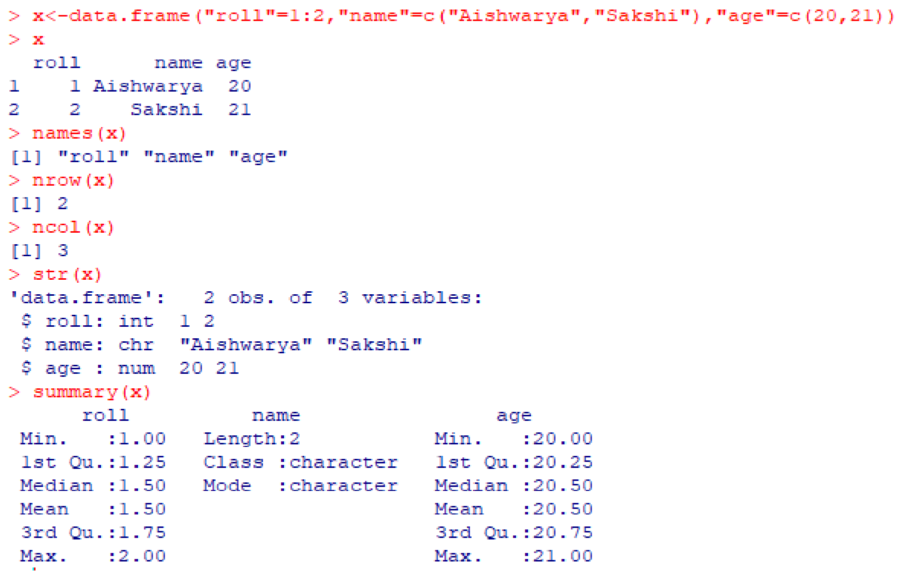
```
> m=cbind(c(1,2,3),c(4,5,6))
> m
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> n=rbind(c(1,2,3),c(4,5,6))
> n
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

16. Creating Factors.



```
> x<-factor(c("single","married","married","single","divorced"))
> x
[1] single   married  married  single   divorced
Levels: divorced married single
> class(x)
[1] "factor"
> levels(x)
[1] "divorced" "married"  "single"
> str(x)
 Factor w/ 3 levels "divorced","married",..: 3 2 2 3 1
>
```

17. Creating Data Frames.



```
> x<-data.frame("roll"=1:2,"name"=c("Aishwarya","Sakshi"),"age"=c(20,21))
> x
  roll      name age
1    1 Aishwarya  20
2    2    Sakshi  21
> names(x)
[1] "roll" "name" "age"
> nrow(x)
[1] 2
> ncol(x)
[1] 3
> str(x)
'data.frame':   2 obs. of  3 variables:
 $ roll: int  1 2
 $ name: chr  "Aishwarya" "Sakshi"
 $ age : num  20 21
> summary(x)
      roll          name                 age       
 Min.   :1.00   Length:2           Min.   :20.00  
 1st Qu.:1.25   Class :character   1st Qu.:20.25  
 Median :1.50   Mode  :character   Median :20.50  
 Mean   :1.50                      Mean   :20.50  
 3rd Qu.:1.75                      3rd Qu.:20.75  
 Max.   :2.00                      Max.   :21.00  
```

Conclusion: We have successfully learned about the vectors, list, matrices, factors, data frames in R language, and also execute or implement some of the program successfully.

Practical No.: 3

Aim: Implement random sampling with and without replacement and stratified sampling in R with reproducible sample.

Theory:

Sampling is the practice of selecting an individual group from a population to study the whole population.

Sampling type comes under two broad categories:

- **Probability sampling** - Probability sampling allows every member of the population a chance to get selected. It is mainly used in quantitative research when you want to produce results representative of the whole population.
- **Non-probability sampling** - n non-probability sampling, not every individual has a chance of being included in the sample. This sampling method is easier and cheaper but also has high risks of sampling bias. It is often used in exploratory and qualitative research with the aim to develop an initial understanding of the population.

Types of Probability sampling:

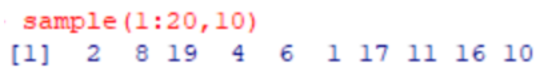
- **Simple random sampling**: In simple random sampling, the researcher selects the participants randomly. There are a number of data analytics tools like random number generators and random number tables used that are based entirely on chance.
- **Stratified sampling**: In stratified sampling, the population is subdivided into subgroups, called strata, based on some characteristics (age, gender, income, etc.). After forming a subgroup, you can then use random or systematic sampling to select a sample for each subgroup. This method allows you to draw more precise conclusions because it ensures that every subgroup is properly represented. in this a subset of observations are selected randomly from each group of the observations defined by the value of a stratifying variable, and once an observation is selected it cannot be selected again.
-

❖ **Sampling without replacement**, in which a subset of the observations are selected randomly, and once an observation is selected it cannot be selected again.

❖ **Sampling with replacement**, in which a subset of observations are selected randomly, and an observation may be selected more than once.

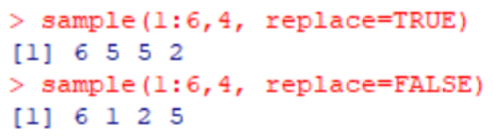
Program:

1. Random sampling.



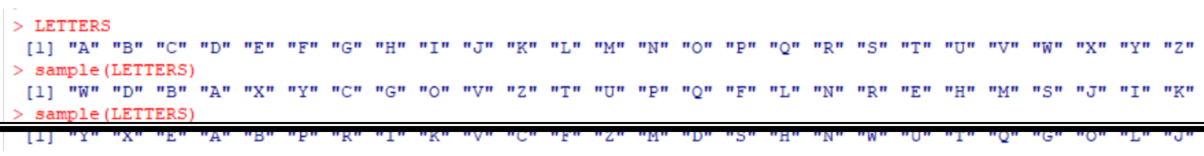
```
sample(1:20,10)
[1]  2  8 19  4  6  1 17 11 16 10
```

2. Random sampling with and without replacement.



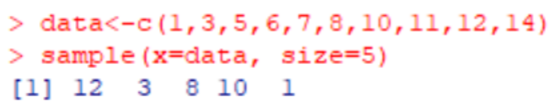
```
> sample(1:6,4, replace=TRUE)
[1] 6 5 5 2
> sample(1:6,4, replace=FALSE)
[1] 6 1 2 5
```

3. Random sampling for character.



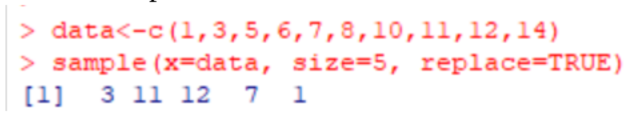
```
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
> sample(LETTERS)
 [1] "W" "D" "B" "A" "X" "Y" "C" "G" "O" "V" "Z" "T" "U" "P" "Q" "F" "L" "N" "R" "E" "H" "M" "S" "J" "I" "K"
> sample(LETTERS)
 [1] "Y" "X" "E" "A" "B" "P" "R" "I" "K" "V" "C" "F" "Z" "M" "D" "S" "H" "N" "W" "U" "T" "Q" "G" "O" "L" "J"
```

4. Random sampling on vector without replacement.



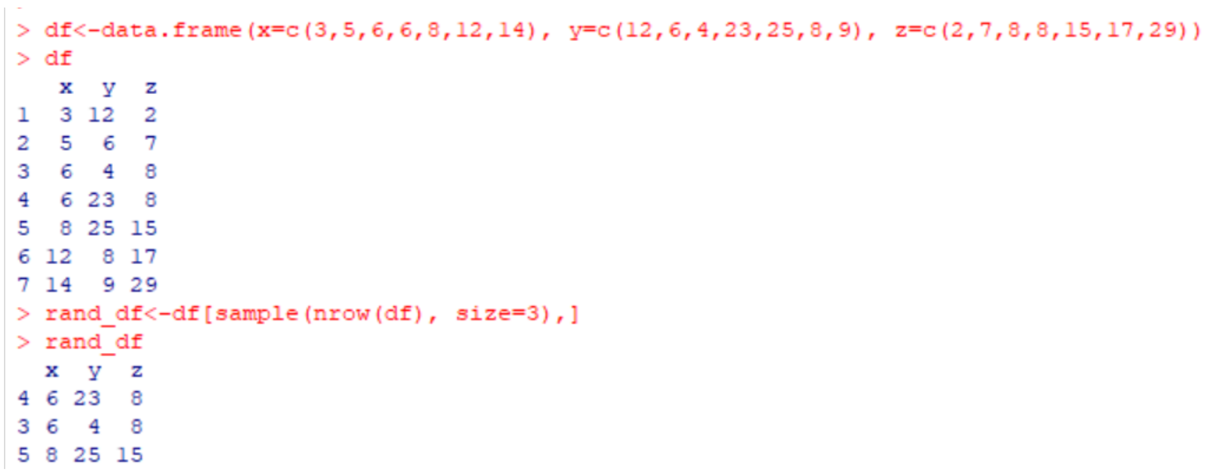
```
> data<-c(1,3,5,6,7,8,10,11,12,14)
> sample(x=data, size=5)
[1] 12  3  8 10  1
```

5. Random sampling on vector with replacement.



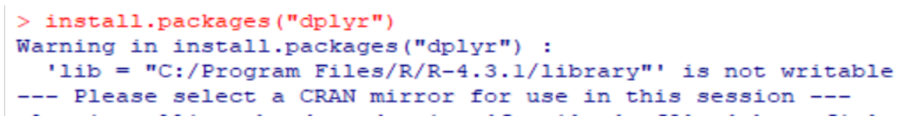
```
> data<-c(1,3,5,6,7,8,10,11,12,14)
> sample(x=data, size=5, replace=TRUE)
[1]  3 11 12  7  1
```

6. Random sampling on data frames.

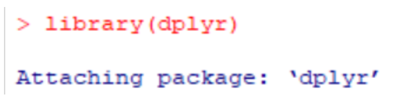


```
> df<-data.frame(x=c(3,5,6,6,8,12,14), y=c(12,6,4,23,25,8,9), z=c(2,7,8,8,15,17,29))
> df
   x  y  z
1  3 12  2
2  5  6  7
3  6  4  8
4  6 23  8
5  8 25 15
6 12  8 17
7 14  9 29
> rand_df<-df[sample(nrow(df), size=3),]
> rand_df
  x  y  z
4 6 23  8
3 6  4  8
5 8 25 15
```

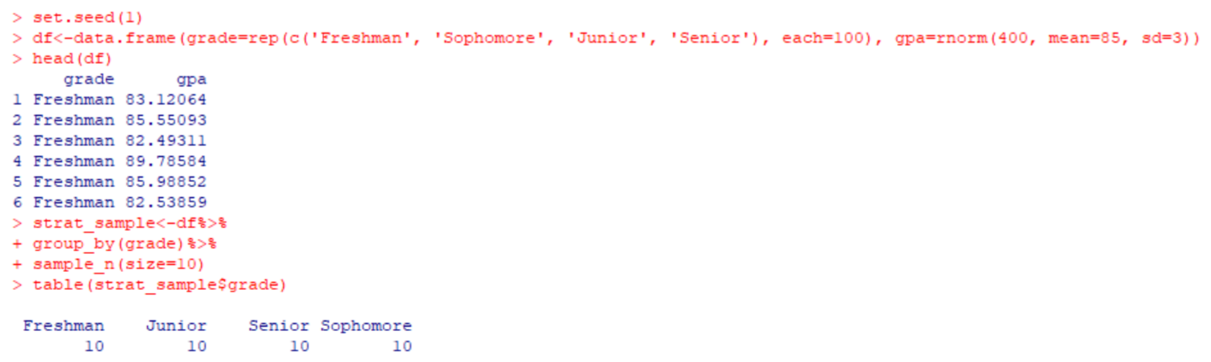
7. Stratified sampling on data frames using number of rows.



```
> install.packages("dplyr")
Warning in install.packages("dplyr") :
  'lib = "C:/Program Files/R/R-4.3.1/library"' is not writable
--- Please select a CRAN mirror for use in this session ---
```

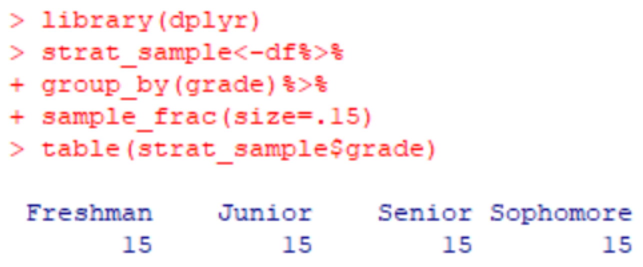


```
> library(dplyr)

Attaching package: 'dplyr'
```



```
> set.seed(1)
> df<-data.frame(grade=rep(c('Freshman', 'Sophomore', 'Junior', 'Senior'), each=100), gpa=rnorm(400, mean=85, sd=3))
> head(df)
     grade      gpa
1 Freshman 83.12064
2 Freshman 85.55093
3 Freshman 82.49311
4 Freshman 89.78584
5 Freshman 85.98852
6 Freshman 82.53859
> strat_sample<-df%>%
+ group_by(grade)%>%
+ sample_n(size=10)
> table(strat_sample$grade)

 Freshman    Junior    Senior Sophomore 
       10        10        10        10 
```

8. Stratified sampling on data frames using fraction of rows.



```
> library(dplyr)
> strat_sample<-df%>%
+ group_by(grade)%>%
+ sample_frac(size=.15)
> table(strat_sample$grade)

 Freshman    Junior    Senior Sophomore 
       15        15        15        15 
```

Conclusion: We have successfully, implement the random sampling with and without replacement and stratified sampling in R.

Practical No.: 4

Aim: To implement calculation of the measures of central tendency mean, median and mode in R.

Theory:

The **measure of central tendency** in R Language represents the whole set of data by a single value. It gives us the location of the central points. There are three main measures of central tendency:

- Mean: It is the sum of observations divided by the total number of observations. It is also defined as average which is the sum divided by count.

  Formula:

  $$\text{Mean } (\bar{x}) = \frac{\sum x}{n}$$

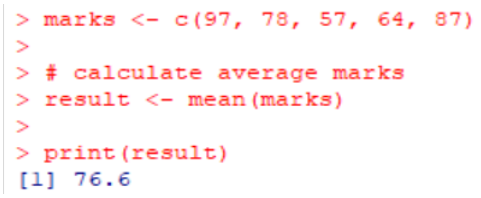
  Where, **n** = number of terms
- Median: I t is the middle value of the data set. It splits the data into two halves. If the number of elements in the data set is odd then the center element is median and if it is even then the median would be the average of two central elements.
  Formula:

Odd	Even
$\frac{n+1}{2}$	$\frac{n}{2}, \frac{n}{2}+1$

  Where, **n** = number of terms
- Mode: It is the value that has the highest frequency in the given data set. The data set may have no mode if the frequency of all data points is the same. Also, we can have more than one mode if we encounter two or more data points having the same frequency.

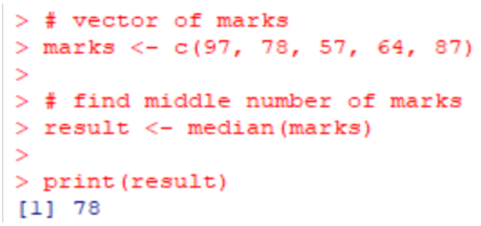
Program:

1. To calculated mean on vector.



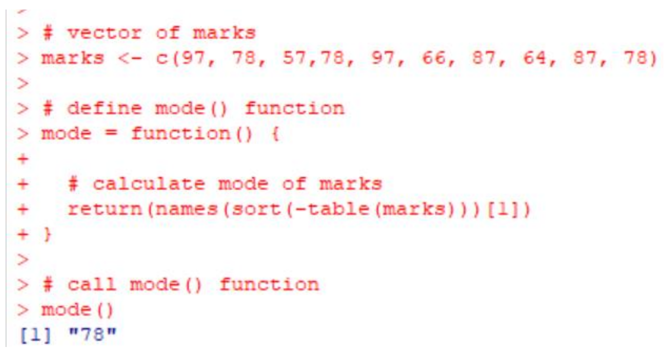
```
> marks <- c(97, 78, 57, 64, 87)
>
> # calculate average marks
> result <- mean(marks)
>
> print(result)
[1] 76.6
```

2. To calculated median on vector.



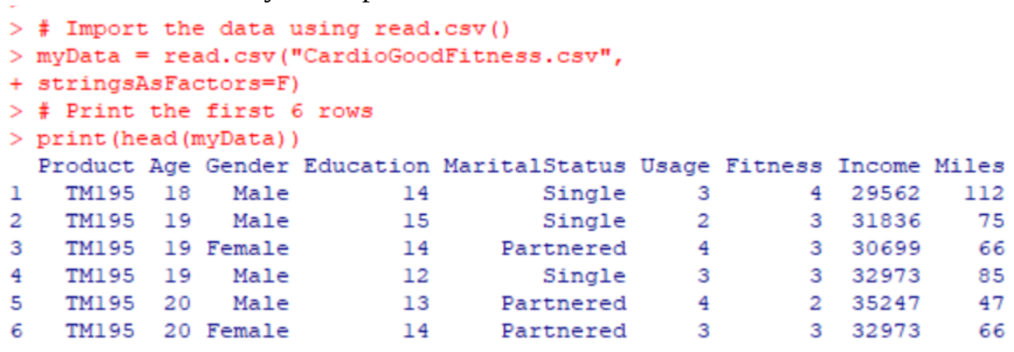
```
> # vector of marks
> marks <- c(97, 78, 57, 64, 87)
>
> # find middle number of marks
> result <- median(marks)
>
> print(result)
[1] 78
```

3. To calculated mode on vector.



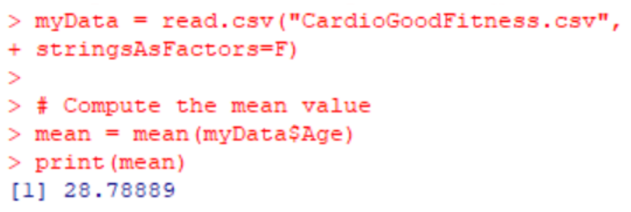
```
> # vector of marks
> marks <- c(97, 78, 57,78, 97, 66, 87, 64, 87, 78)
>
> # define mode() function
> mode = function() {
+
+   # calculate mode of marks
+   return(names(sort(-table(marks)))[1])
+ }
>
> # call mode() function
> mode()
[1] "78"
```

4. Calculated the central tendency on imported file.



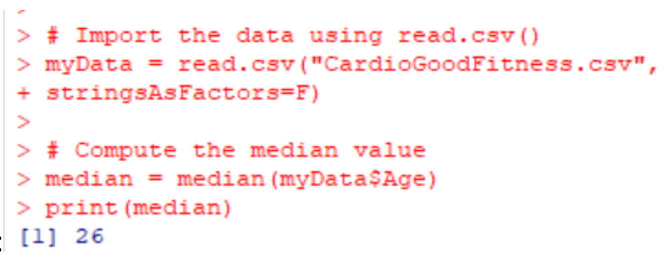
```
> # Import the data using read.csv()
> myData = read.csv("CardioGoodFitness.csv",
+ stringsAsFactors=F)
> # Print the first 6 rows
> print(head(myData))
  Product Age Gender Education MaritalStatus Usage Fitness Income Miles
1   TM195  18   Male        14        Single     3       4  29562   112
2   TM195  19   Male        15        Single     2       3  31836    75
3   TM195  19 Female        14     Partnered     4       3  30699    66
4   TM195  19   Male        12        Single     3       3  32973    85
5   TM195  20   Male        13     Partnered     4       2  35247    47
6   TM195  20 Female        14     Partnered     3       3  32973    66
```

5. Calculated the mean on imported file.



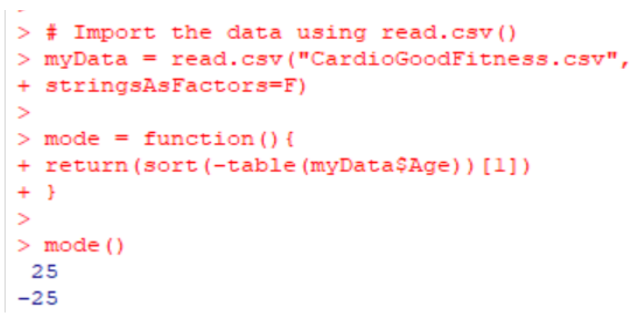
```
> myData = read.csv("CardioGoodFitness.csv",
+ stringsAsFactors=F)
>
> # Compute the mean value
> mean = mean(myData$Age)
> print(mean)
[1] 28.78889
```

6. Calculated the median on imported file.



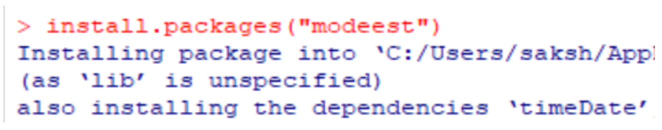
```
> # Import the data using read.csv()
> myData = read.csv("CardioGoodFitness.csv",
+ stringsAsFactors=F)
>
> # Compute the median value
> median = median(myData$Age)
> print(median)
[1] 26
```

7. Calculated the mode on imported file.

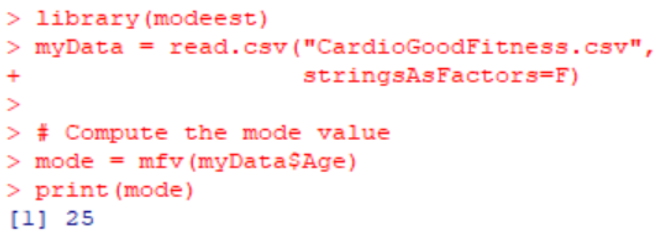


```
> # Import the data using read.csv()
> myData = read.csv("CardioGoodFitness.csv",
+ stringsAsFactors=F)
>
> mode = function(){
+ return(sort(-table(myData$Age))[1])
+ }
>
> mode()
 25
-25
```

8. Calculated the mode on imported file using modeest library.



```
> install.packages("modeest")
Installing package into 'C:/Users/saksh/App
(as 'lib' is unspecified)
also installing the dependencies 'timeDate'
```



```
> library(modeest)
> myData = read.csv("CardioGoodFitness.csv",
+                   stringsAsFactors=F)
>
> # Compute the mode value
> mode = mfv(myData$Age)
> print(mode)
[1] 25
```

Conclusion: We have successfully, implement and calculation of the measures of central tendency mean, median and mode in R.

Practical No.: 5

Aim: To implement calculation of the measures of variability range, standard deviation, variance, percentile, interquartile range.

Theory:

Variability (also known as **Statistical Dispersion**) is another feature of descriptive statistics. Measures of central tendency and variability together comprise of descriptive statistics. Variability shows the spread of a data set around a point.

Measures of Variability:

- **Variance** is a measure that shows how far each value is from a particular point, preferably the mean value. Mathematically, it is defined as the average of squared differences from the mean value.
  Formula:

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}$$

- **Standard deviation** in statistics measures the spreadness of data values with respect to mean and mathematically, is calculated as square root of variance.
  Formula:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}}$$

- **Range** is the difference between the maximum and minimum value of a data set. In R language, **max()** and **min()** is used to find the same, unlike **range()** function that returns the minimum and maximum value of the data set.

- **Interquartile Range** is based on splitting a data set into parts called as quartiles. There are 3 quartile values (Q1, Q2, Q3) that divide the whole data set into 4 equal parts. Q2 specifies the median of the whole data set.

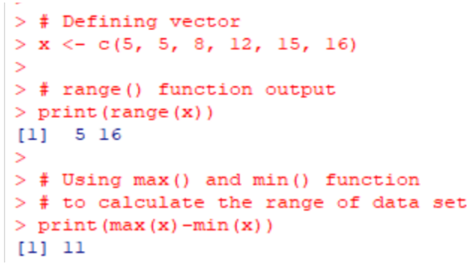
  Mathematically, the interquartile range is depicted as:

  IQR = Q3 – Q1

  where, Q3 = specifies the median of n largest values, **Q1** = specifies the median of n smallest values.
- **Percentiles** are measures of central tendency, which depict that out of the total data about certain percent data lies below it.

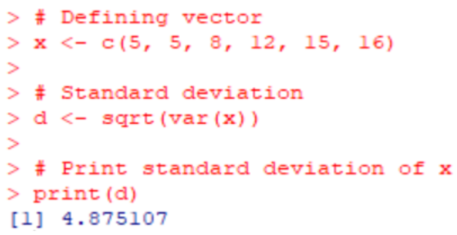
Program:

1. To calculated range on vector.



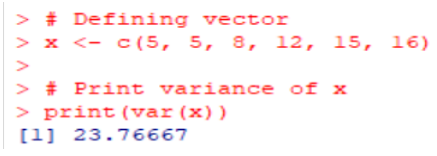
```
> # Defining vector
> x <- c(5, 5, 8, 12, 15, 16)
>
> # range() function output
> print(range(x))
[1]  5 16
>
> # Using max() and min() function
> # to calculate the range of data set
> print(max(x)-min(x))
[1] 11
```

2. To calculated standard deviation on vector.



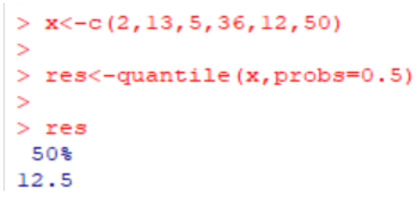
```
> # Defining vector
> x <- c(5, 5, 8, 12, 15, 16)
>
> # Standard deviation
> d <- sqrt(var(x))
>
> # Print standard deviation of x
> print(d)
[1] 4.875107
```

3. To calculated variance on vector.



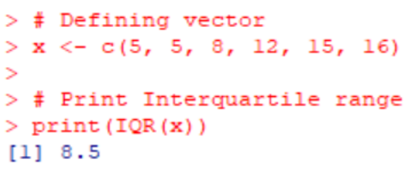
```
> # Defining vector
> x <- c(5, 5, 8, 12, 15, 16)
>
> # Print variance of x
> print(var(x))
[1] 23.76667
```

4. To calculated percentile on vector.



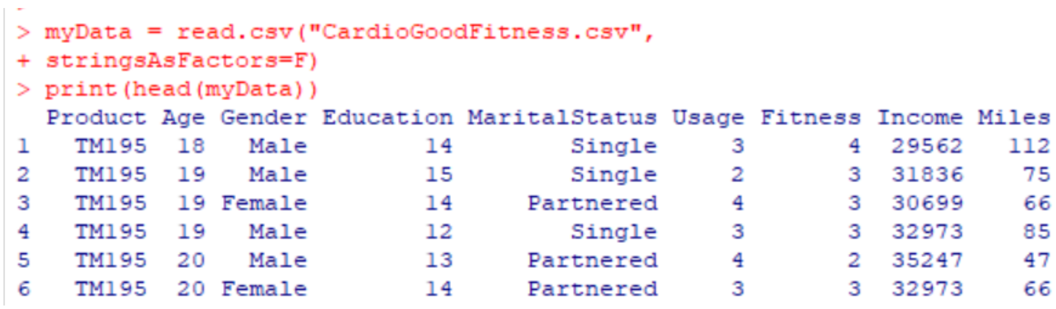
```
> x<-c(2,13,5,36,12,50)
>
> res<-quantile(x,probs=0.5)
>
> res
 50%
12.5
```

5. To calculated interquartile range on vector.



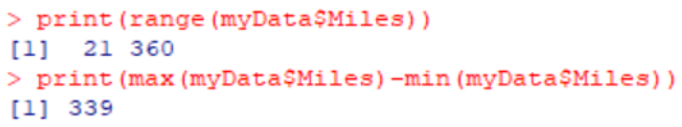
```
> # Defining vector
> x <- c(5, 5, 8, 12, 15, 16)
>
> # Print Interquartile range
> print(IQR(x))
[1] 8.5
```

6. Calculated the variability on imported file.



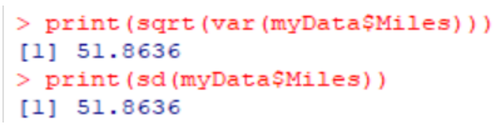
```
> myData = read.csv("CardioGoodFitness.csv",
+ stringsAsFactors=F)
> print(head(myData))
  Product Age Gender Education MaritalStatus Usage Fitness Income Miles
1   TM195  18   Male        14        Single     3       4  29562   112
2   TM195  19   Male        15        Single     2       3  31836    75
3   TM195  19 Female        14     Partnered     4       3  30699    66
4   TM195  19   Male        12        Single     3       3  32973    85
5   TM195  20   Male        13     Partnered     4       2  35247    47
6   TM195  20 Female        14     Partnered     3       3  32973    66
```

7. Calculated the range on imported file.



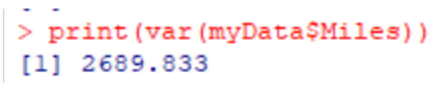
```
> print(range(myData$Miles))
[1]  21 360
> print(max(myData$Miles)-min(myData$Miles))
[1] 339
```

8. Calculated the standard deviation on imported file.



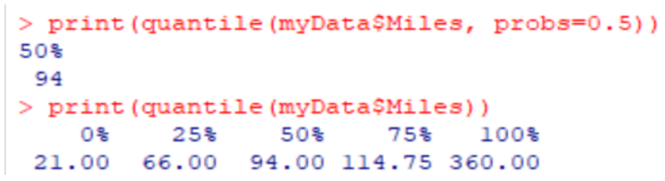
```
> print(sqrt(var(myData$Miles)))
[1] 51.8636
> print(sd(myData$Miles))
[1] 51.8636
```

9. Calculated the variance on imported file.

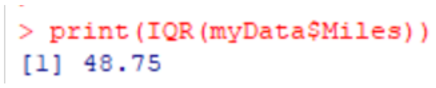


```
> print(var(myData$Miles))
[1] 2689.833
```

10. Calculated the percentile on imported file.


```
> print(quantile(myData$Miles, probs=0.5))
50%
 94
> print(quantile(myData$Miles))
    0%    25%    50%    75%   100%
 21.00  66.00  94.00 114.75 360.00
```

11. Calculated the interquartile range on imported file.


```
> print(IQR(myData$Miles))
[1] 48.75
```

Conclusion: We have successfully, implement and calculation of the measures of variability range, standard deviation, variance, percentile, interquartile range.

Practical No.: 6

Aim: To implement data visualization in R.

Theory:

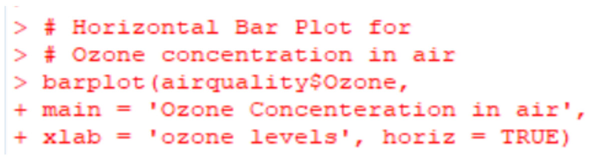
Data visualization is the technique used to deliver insights in data using visual cues such as graphs, charts, maps, and many others. This is useful as it helps in intuitive and easy understanding of the large quantities of data and thereby make better decisions regarding it. R is a language that is designed for statistical computing, graphical data analysis, and scientific research. It is usually preferred for data visualization as it offers flexibility and minimum required coding through its packages.

Types of Data Visualizations:

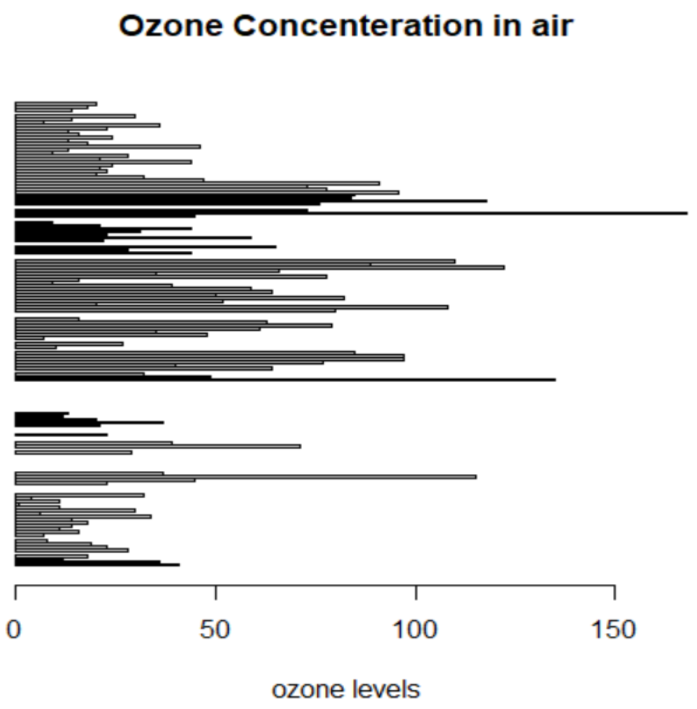
- Bar plot: There are two types of bar plots- horizontal and vertical which represent data points as horizontal or vertical bars of certain lengths proportional to the value of the data item. They are generally used for continuous and categorical variable plotting.
- Histogram: A histogram is like a bar chart as it uses bars of varying height to represent data distribution.
- Box plot: The statistical summary of the given data is presented graphically using a boxplot. A boxplot depicts information like the minimum and maximum data point, the median value, first and third quartile, and interquartile range.
- Scatter plot: A scatter plot is composed of many points on a Cartesian plane. Each point denotes the value taken by two parameters and helps us easily identify the relationship between them.
- Heatmap: Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix.
- Map visualization: Here we are using maps package to visualize and display geographical maps using an R programming language.
- 3D graph: Here we will use preps() function, This function is used to create 3D surfaces in perspective view. This function will draw perspective plots of a surface over the x–y plane.

Program:

1. Horizontal Bar plot

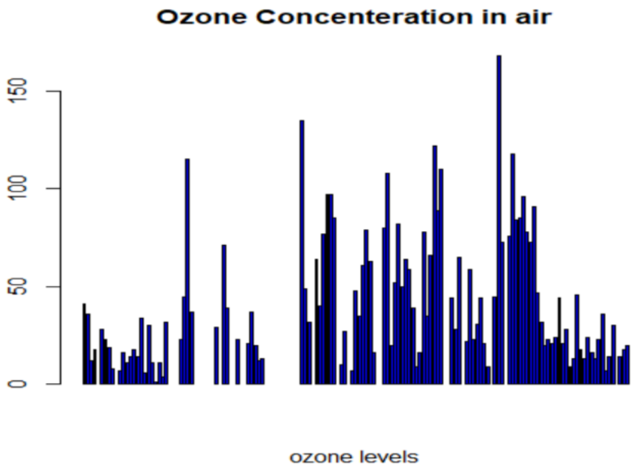


```
> # Horizontal Bar Plot for
> # Ozone concentration in air
> barplot(airquality$Ozone,
+ main = 'Ozone Concenteration in air',
+ xlab = 'ozone levels', horiz = TRUE)
```



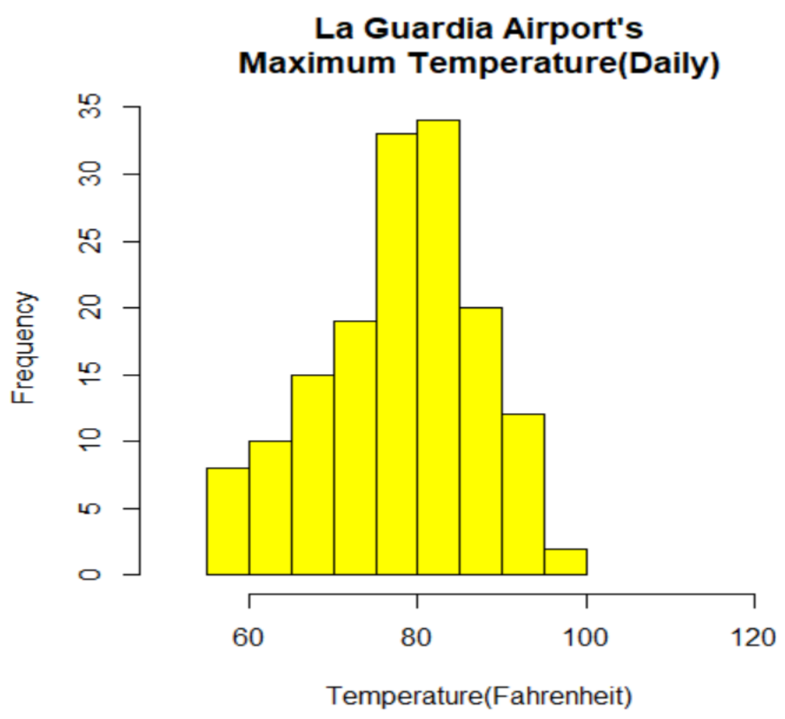
2. Vertical Bar Plot

```
> # Vertical Bar Plot for
> # Ozone concentration in air
> barplot(airquality$Ozone, main = 'Ozone Concenteration in air',
+ xlab = 'ozone levels', col ='blue', horiz = FALSE)
```

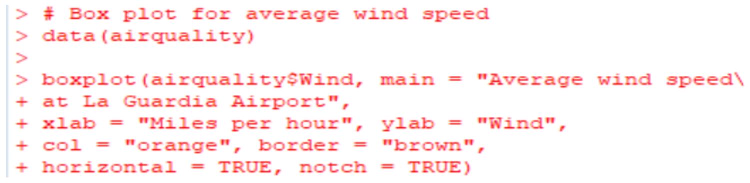


3. Histogram

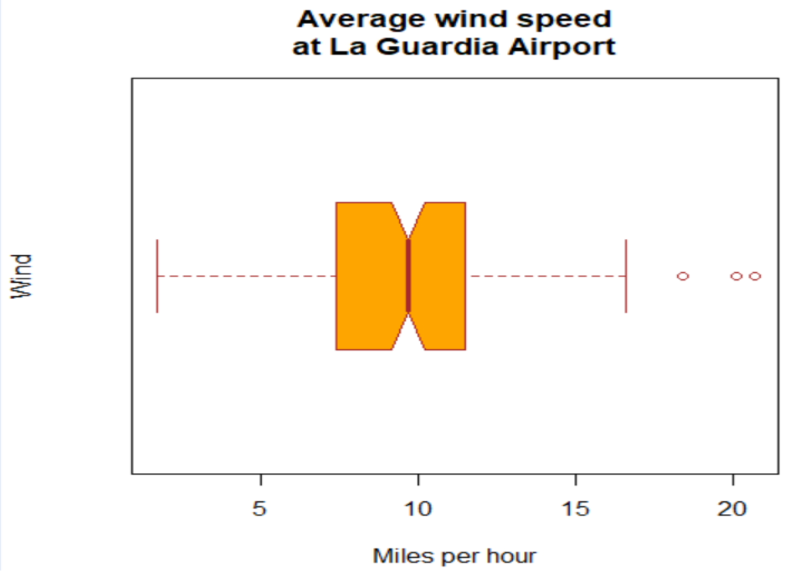
```
> # Histogram for Maximum Daily Temperature
> data(airquality)
>
> hist(airquality$Temp, main ="La Guardia Airport's\
+ Maximum Temperature(Daily)",
+ xlab ="Temperature(Fahrenheit)",
+ xlim = c(50, 125), col ="yellow",
+ freq = TRUE)
```



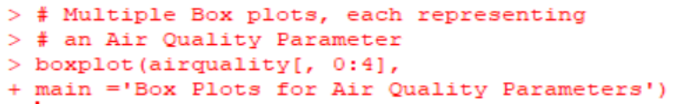
4. Box plot



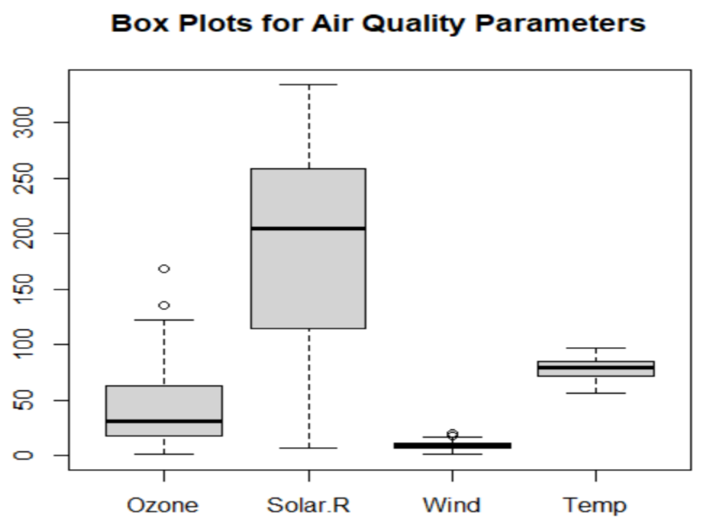
```
> # Box plot for average wind speed
> data(airquality)
>
> boxplot(airquality$Wind, main = "Average wind speed\
+ at La Guardia Airport",
+ xlab = "Miles per hour", ylab = "Wind",
+ col = "orange", border = "brown",
+ horizontal = TRUE, notch = TRUE)
```



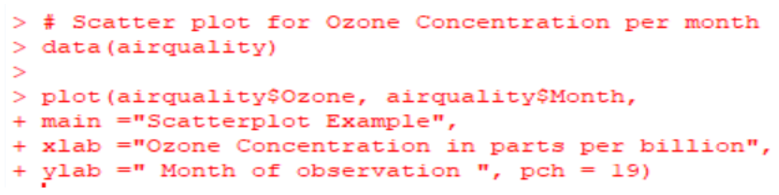
5. Multiple box plots



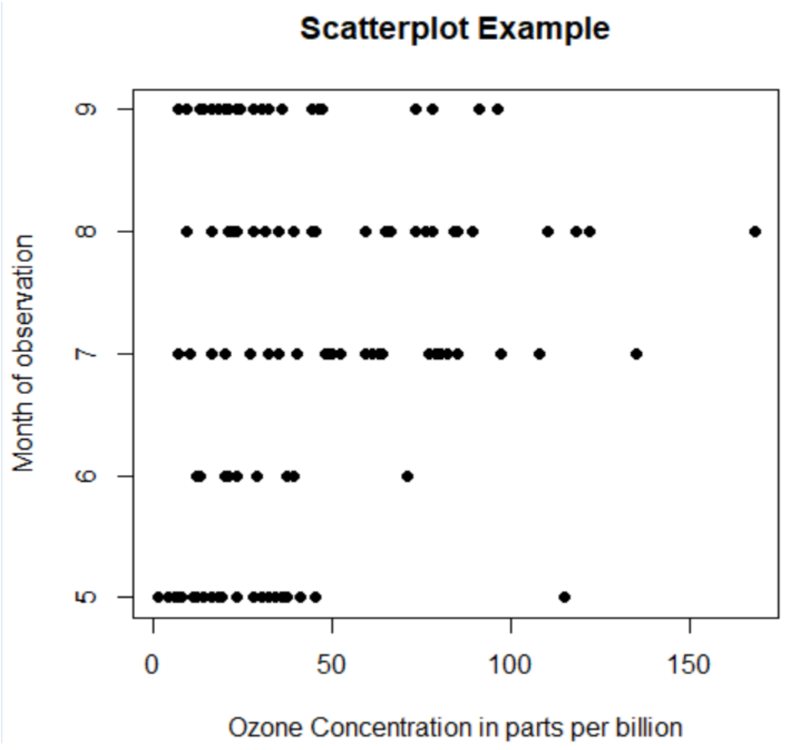
```
> # Multiple Box plots, each representing
> # an Air Quality Parameter
> boxplot(airquality[, 0:4],
+ main ='Box Plots for Air Quality Parameters')
```



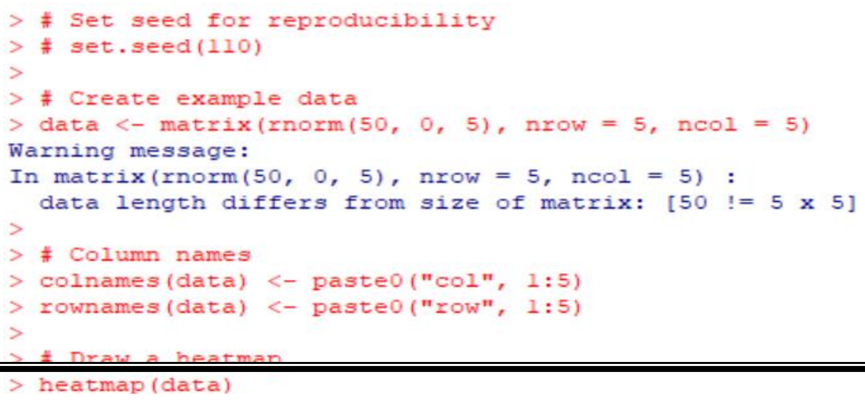
6. Scatter plot



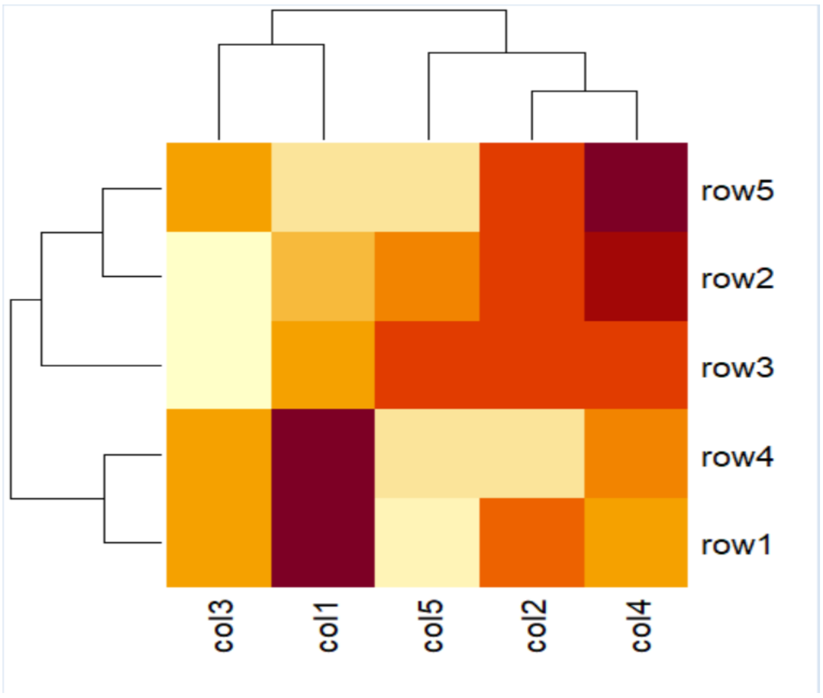
```
> # Scatter plot for Ozone Concentration per month
> data(airquality)
>
> plot(airquality$Ozone, airquality$Month,
+ main ="Scatterplot Example",
+ xlab ="Ozone Concentration in parts per billion",
+ ylab =" Month of observation ", pch = 19)
```



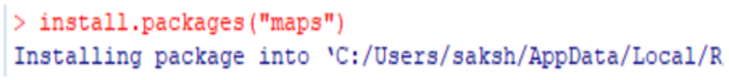
7. Heatmap



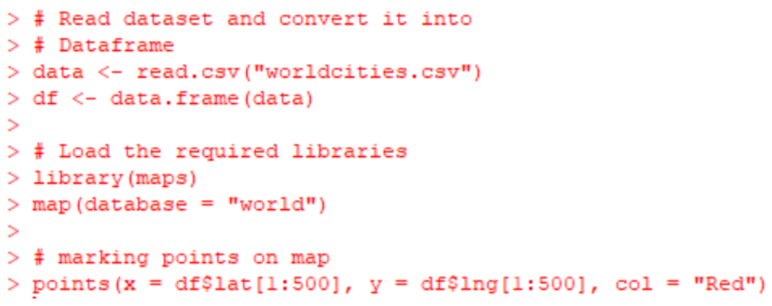
```
> # Set seed for reproducibility
> # set.seed(110)
>
> # Create example data
> data <- matrix(rnorm(50, 0, 5), nrow = 5, ncol = 5)
Warning message:
In matrix(rnorm(50, 0, 5), nrow = 5, ncol = 5) :
  data length differs from size of matrix: [50 != 5 x 5]
>
> # Column names
> colnames(data) <- paste0("col", 1:5)
> rownames(data) <- paste0("row", 1:5)
>
> # Draw a heatmap
> heatmap(data)
```



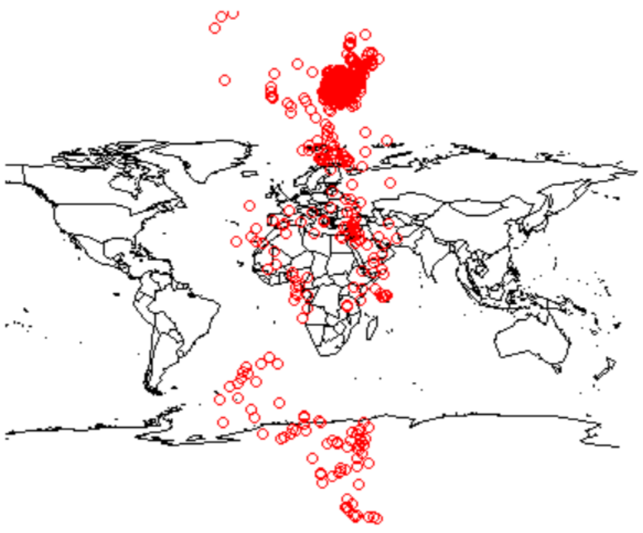
8. Map visualization



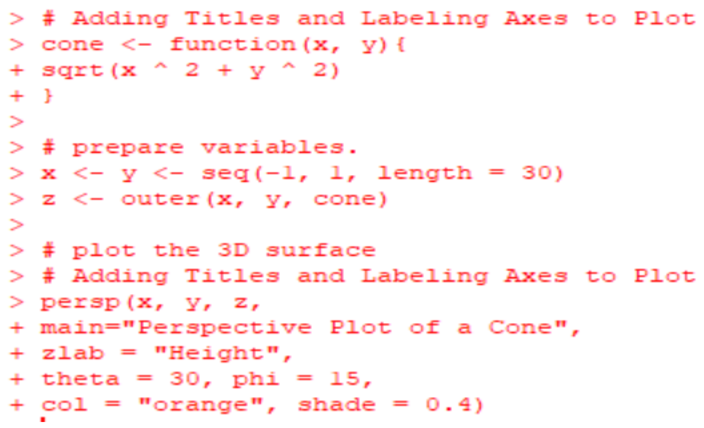
```
> install.packages("maps")
Installing package into 'C:/Users/saksh/AppData/Local/R
```



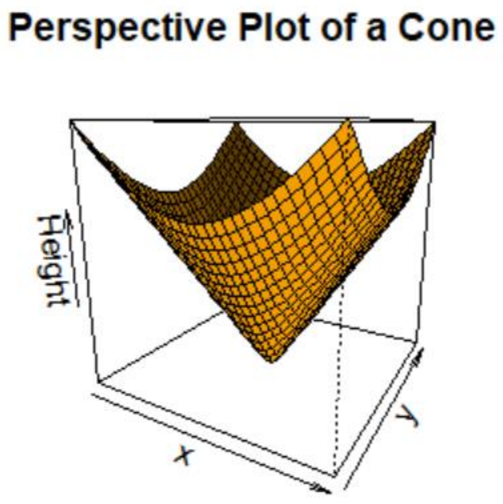
```
> # Read dataset and convert it into
> # Dataframe
> data <- read.csv("worldcities.csv")
> df <- data.frame(data)
>
> # Load the required libraries
> library(maps)
> map(database = "world")
>
> # marking points on map
> points(x = df$lat[1:500], y = df$lng[1:500], col = "Red")
```



9. 3D graphs in R.



```
> # Adding Titles and Labeling Axes to Plot
> cone <- function(x, y){
+ sqrt(x ^ 2 + y ^ 2)
+ }
>
> # prepare variables.
> x <- y <- seq(-1, 1, length = 30)
> z <- outer(x, y, cone)
>
> # plot the 3D surface
> # Adding Titles and Labeling Axes to Plot
> persp(x, y, z,
+ main="Perspective Plot of a Cone",
+ zlab = "Height",
+ theta = 30, phi = 15,
+ col = "orange", shade = 0.4)
```



Conclusion: We have successfully, implement data visualization in R.

Practical No.: 7

Aim: To implement a program to calculate Date and Time in R.

Theory:

R Provides us various functions to deal with dates and times.
In R, we use Sys.Date() and Sys.time() to get the current date and time respectively based on the local system.

- **Sys.Date()** - returns the current date.
- **Sys.time()** - returns the current date, time, and timezone.

The **lubridate** package in R makes the extraction and manipulation of some parts of the date value more efficient.

we have used the **now()** function provided by the lubridate package to get the current date with time and timezone. n R, we use the **year(), month(),** and **mday()** function provided by the lubridate package to extract years, months, and days respectively from multiple date values.

- **year(dates)** - returns all years from dates.
- **month(dates)** - returns all months from dates.
- **mday(dates)** - returns days from dates.

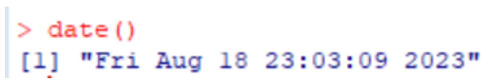
Times in R are represented by the POSIXct or POSIXlt class and Dates are represented by the Date class. The **as.Date()** function handles dates in R without time. This function takes the date as a String in the format YYYY-MM-DD or YYY/MM/DD.

Code	Description
%d	Decimal date
%m	Decimal month
%Y	4-digit year
%H	Decimal hours (24 hour)
%M	Decimal minute
%S	Decimal second
%h	Abbreviated month

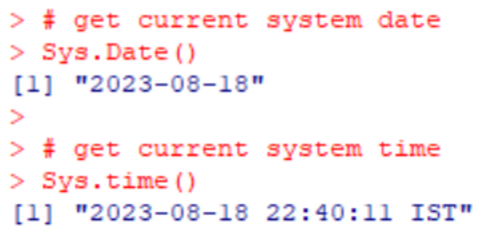
We can use the **update()** function to update multiple dates values all at once. **update()** return a date with the specified elements updated. Elements not specified will be left unaltered.

Program:

1. Get Current System Date, and Time in R

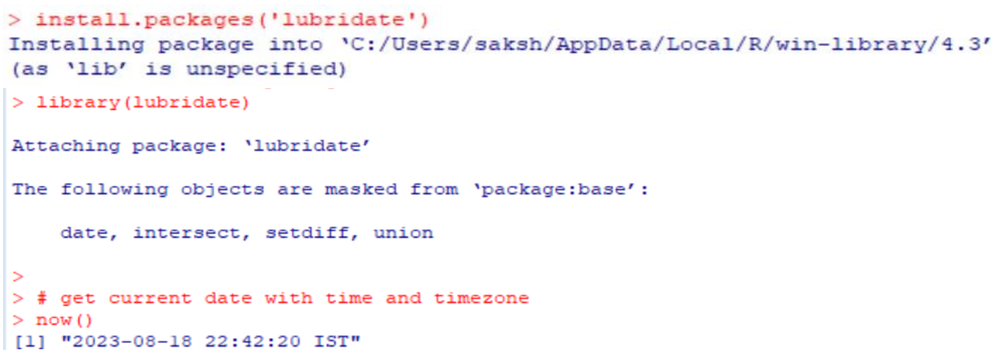


```
> date()
[1] "Fri Aug 18 23:03:09 2023"
```



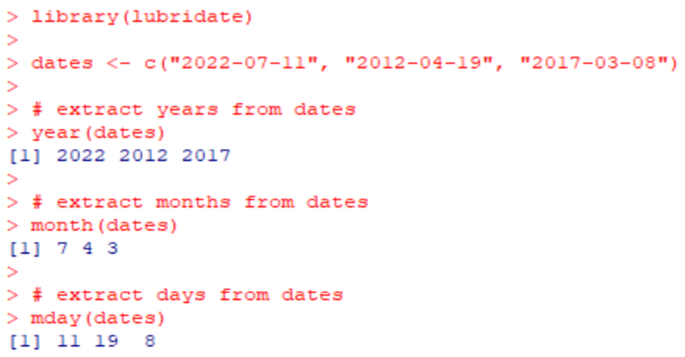
```
> # get current system date
> Sys.Date()
[1] "2023-08-18"
>
> # get current system time
> Sys.time()
[1] "2023-08-18 22:40:11 IST"
```

2. Get Current Date Using R lubridate Package



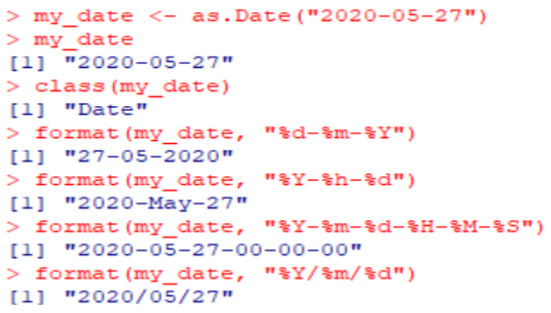
```
> install.packages('lubridate')
Installing package into 'C:/Users/saksh/AppData/Local/R/win-library/4.3'
(as 'lib' is unspecified)
> library(lubridate)

Attaching package: 'lubridate'

The following objects are masked from 'package:base':

    date, intersect, setdiff, union

>
> # get current date with time and timezone
> now()
[1] "2023-08-18 22:42:20 IST"
```

3. Extraction Years, Months, and Days from Multiple Date Values in R



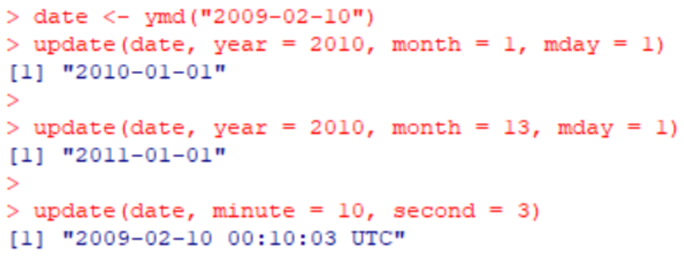
```
> library(lubridate)
>
> dates <- c("2022-07-11", "2012-04-19", "2017-03-08")
>
> # extract years from dates
> year(dates)
[1] 2022 2012 2017
>
> # extract months from dates
> month(dates)
[1] 7 4 3
>
> # extract days from dates
> mday(dates)
[1] 11 19  8
```

4. Manipulate Date Values in R



```
> my_date <- as.Date("2020-05-27")
> my_date
[1] "2020-05-27"
> class(my_date)
[1] "Date"
> format(my_date, "%d-%m-%Y")
[1] "27-05-2020"
> format(my_date, "%Y-%h-%d")
[1] "2020-May-27"
> format(my_date, "%Y-%m-%d-%H-%M-%S")
[1] "2020-05-27-00-00-00"
> format(my_date, "%Y/%m/%d")
[1] "2020/05/27"
```

5. Using update() to Update dates Values in R



```
> date <- ymd("2009-02-10")
> update(date, year = 2010, month = 1, mday = 1)
[1] "2010-01-01"
>
> update(date, year = 2010, month = 13, mday = 1)
[1] "2011-01-01"
>
> update(date, minute = 10, second = 3)
[1] "2009-02-10 00:10:03 UTC"
```

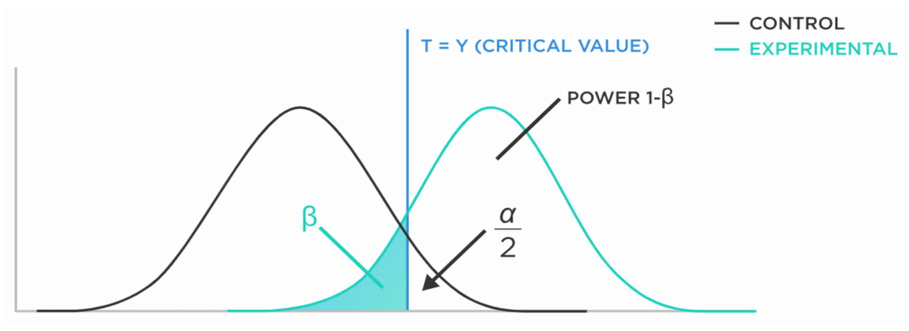
Conclusion: We have successfully learned about the date and time related concept, and also implement date and time program successfully.

Practical No.: 8

Aim: To study and implement statistical power analysis using Python.

Theory:

A **power analysis** is a calculation used to estimate the smallest sample size needed for an experiment, given a required significance level, statistical power, and effect size. It helps to determine if a result from an experiment or survey is due to chance, or if it is genuine and significant.



Statistical power is made of four related parts.

1. **Effect size**: The quantified size of a result present in the population.
2. **Sample size**: A number of things measured: How many dogs do we need to observe?
3. **Significance**: The level of significance used in the experiment, generally 5 percent.
4. **Statistical power**: The probability of accepting the alternative hypothesis.

All these variables are inter-linked; more dogs tested can make the effect easier to detect, and the statistical power may be increased by growing the significance level.

A power analysis estimates one of these four parameters, when given the values for the remaining three. Most commonly, it is used to estimate the minimum sample size needed for an experiment. Additionally, multiple power analyses are used to provide a curve of one parameter versus another, for instance, if the change in the effect size is due to the changes in a sample size. This is incredibly useful when designing an experiment to structure it better, have a better power size, and hopefully result in a more statistically significant result.

This is important because testing, experiments, and surveys are expensive to conduct. An organization does not want to run an experiment and realize afterwards that the sample size was too small to determine if the outcome was genuine or not.

Program:

1.
```
from math import sqrt
from statsmodels.stats.power import TTestIndPower
n1, n2=4,4
s1,s2=5**2, 5**2
s=sqrt(((n1-1) * s1 + (n2-1)* s2)/(n1+n2 -2))
u1,u2=90,85
d=(u1-u2)/s
print(f"Effect size: {d}")
alpha=0.05
power= 0.8
```

```
obj=TTestIndPower()
n=obj.solve_power(effect_size=d, alpha= alpha, power=power, ratio=1, alternative='two-sided')
print("Sample Size/Number needed in each group: {:.3f}".format(n))
```

Output:

```
Effect size: 1.0
Sample Size/Number needed in each group: 16.715
```

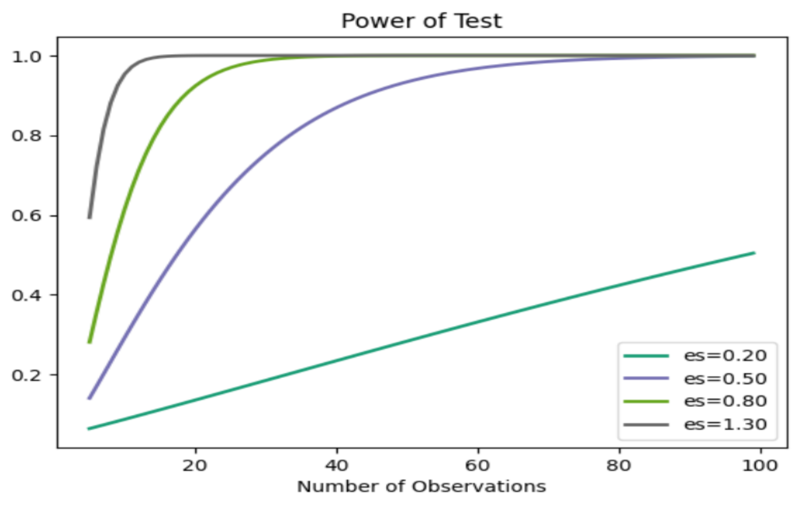
2.
```
from statsmodels.stats.power import TTestPower
power=TTestPower()
n_test=power.solve_power(nobs=40, effect_size=0.5, power=None, alpha=0.05)
print("Power: {:.3f}".format(n_test))
```

Output:

```
Power: 0.869
```

3.
```
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.stats.power import TTestPower
effect_size=np.array([0.2, 0.5, 0.8,1.3])
sample_size=np.array(range(5,100))
obj=TTestPower()
obj.plot_power(dep_var='nobs', nobs=sample_size, effect_size=effect_size)
plt.show()
```

Output:



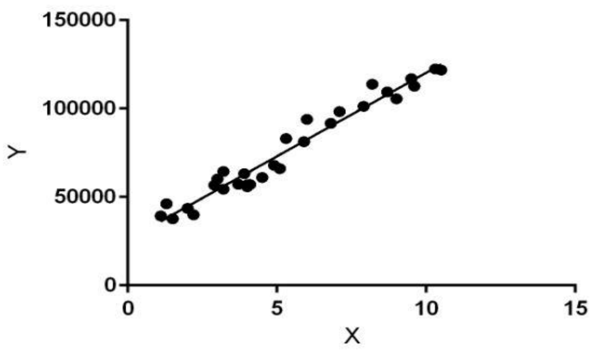
Conclusion: We have successfully learned about the power analysis, and also execute or implement some of the program successfully.

Practical No.: 9

Aim: Write a program to implement Linear regression.

Theory:

Linear Regression is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.



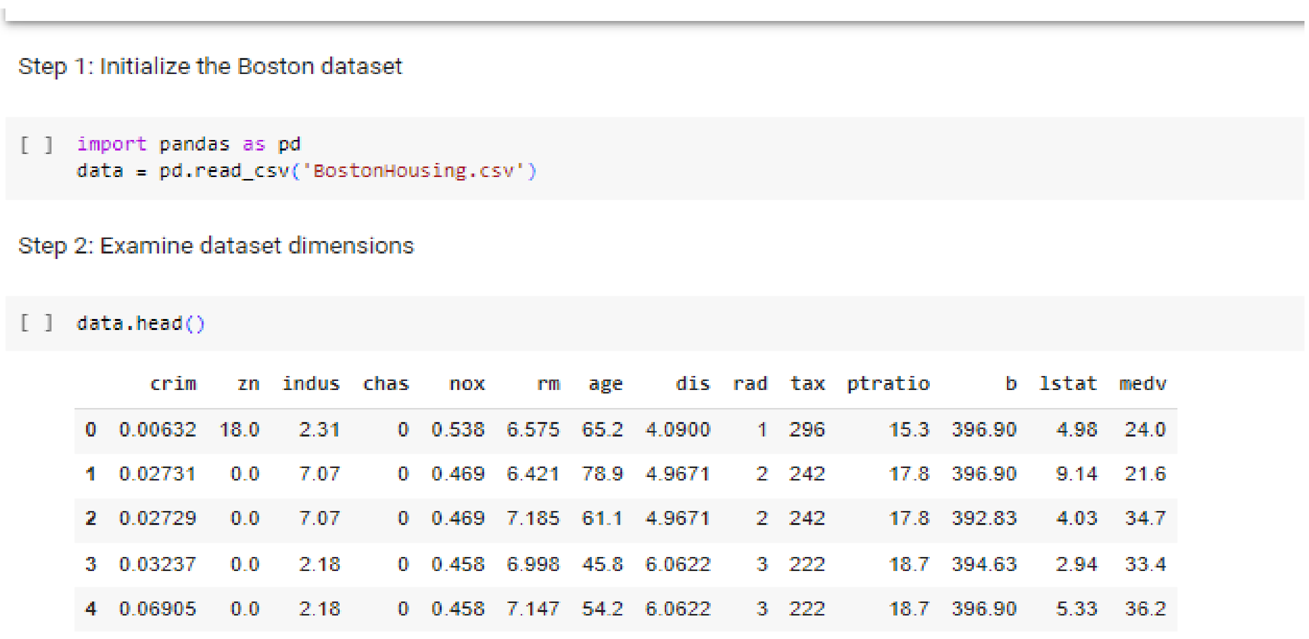
Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.
In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

Hypothesis function for Linear Regression:

$$Y = \theta_1 + \theta_2 \cdot X$$

While training the model we are given:
X: input training data (univariate – one input variable(parameter))
Y: labels to data (supervised learning)
When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ_1 and θ_2 values.
θ_1: intercept
θ_2: coefficient of x
Once we find the best θ_1 and θ_2 values, we get the best fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.

Program



Step 1: Initialize the Boston dataset

```
[ ]  import pandas as pd
     data = pd.read_csv('BostonHousing.csv')
```

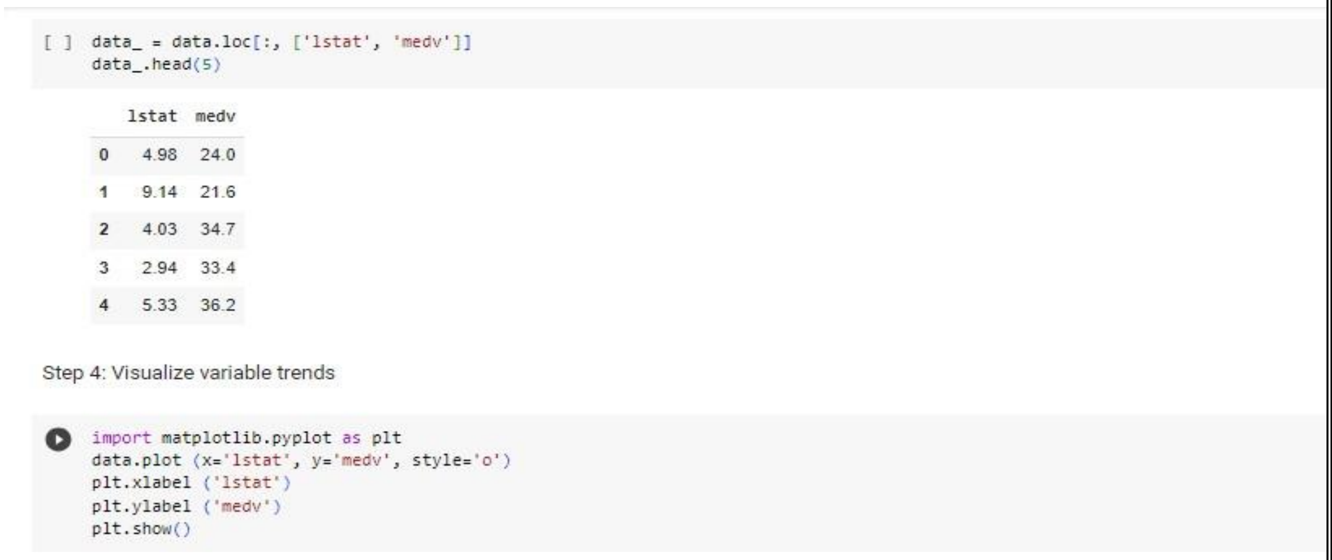
Step 2: Examine dataset dimensions

```
[ ]  data.head()
```

```
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio       b  lstat  medv
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3  396.90   4.98  24.0
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8  396.90   9.14  21.6
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8  392.83   4.03  34.7
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7  394.63   2.94  33.4
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7  396.90   5.33  36.2
```

LSTAT - % lower status of the population

MEDV - Median value of owner-occupied homes in $1000's



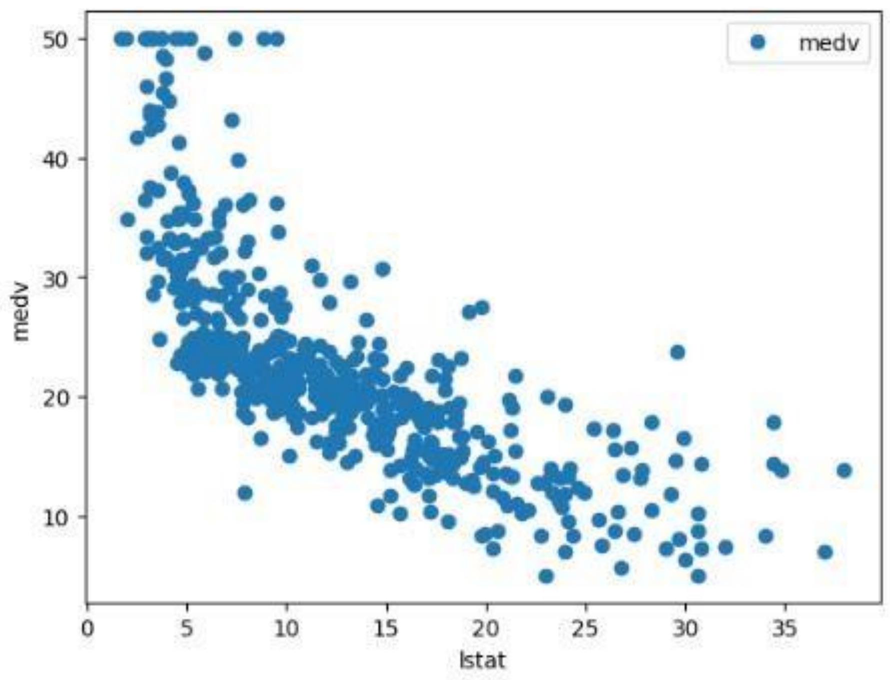
```
[ ]  data_ = data.loc[:, ['lstat', 'medv']]
     data_.head(5)
```

```
   lstat  medv
0   4.98  24.0
1   9.14  21.6
2   4.03  34.7
3   2.94  33.4
4   5.33  36.2
```

Step 4: Visualize variable trends

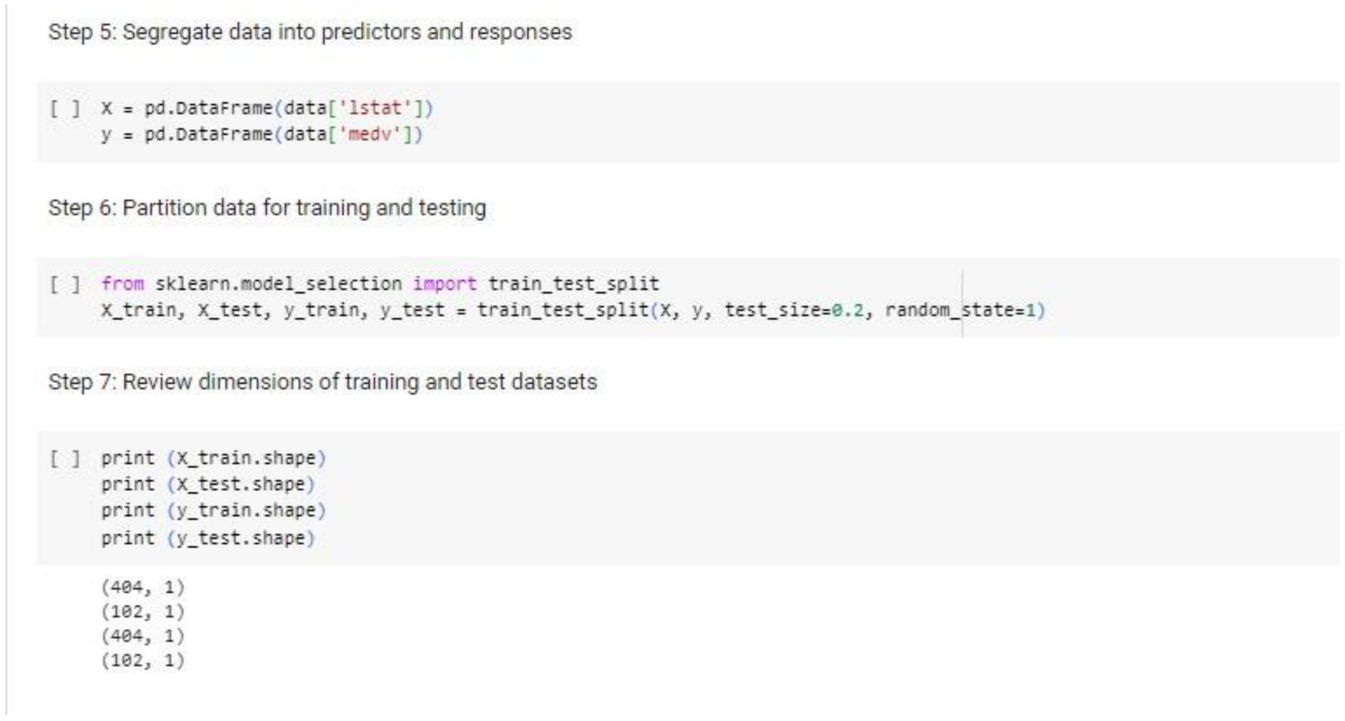
```
import matplotlib.pyplot as plt
data.plot (x='lstat', y='medv', style='o')
plt.xlabel ('lstat')
plt.ylabel ('medv')
plt.show()
```

[ ]



Step 5: Segregate data into predictors and responses

```
[ ] X = pd.DataFrame(data['lstat'])
    y = pd.DataFrame(data['medv'])
```



Step 5: Segregate data into predictors and responses

```
[ ]  X = pd.DataFrame(data['lstat'])
     y = pd.DataFrame(data['medv'])
```

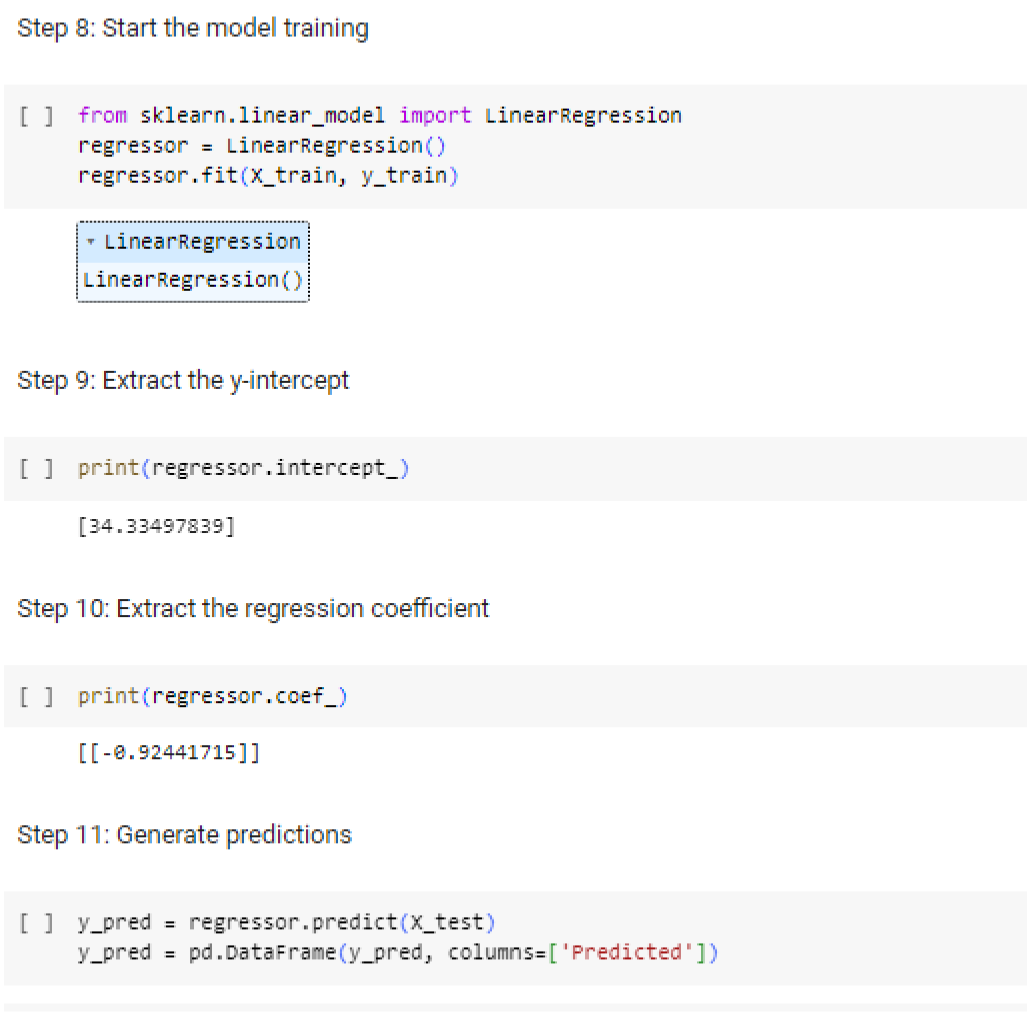
Step 6: Partition data for training and testing

```
[ ]  from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

Step 7: Review dimensions of training and test datasets

```
[ ]  print (X_train.shape)
     print (X_test.shape)
     print (y_train.shape)
     print (y_test.shape)
```

```
     (404, 1)
     (102, 1)
     (404, 1)
     (102, 1)
```



Step 8: Start the model training

```
[ ]  from sklearn.linear_model import LinearRegression
     regressor = LinearRegression()
     regressor.fit(X_train, y_train)
```

```
     ▾ LinearRegression
     LinearRegression()
```

Step 9: Extract the y-intercept

```
[ ]  print(regressor.intercept_)
```

```
     [34.33497839]
```

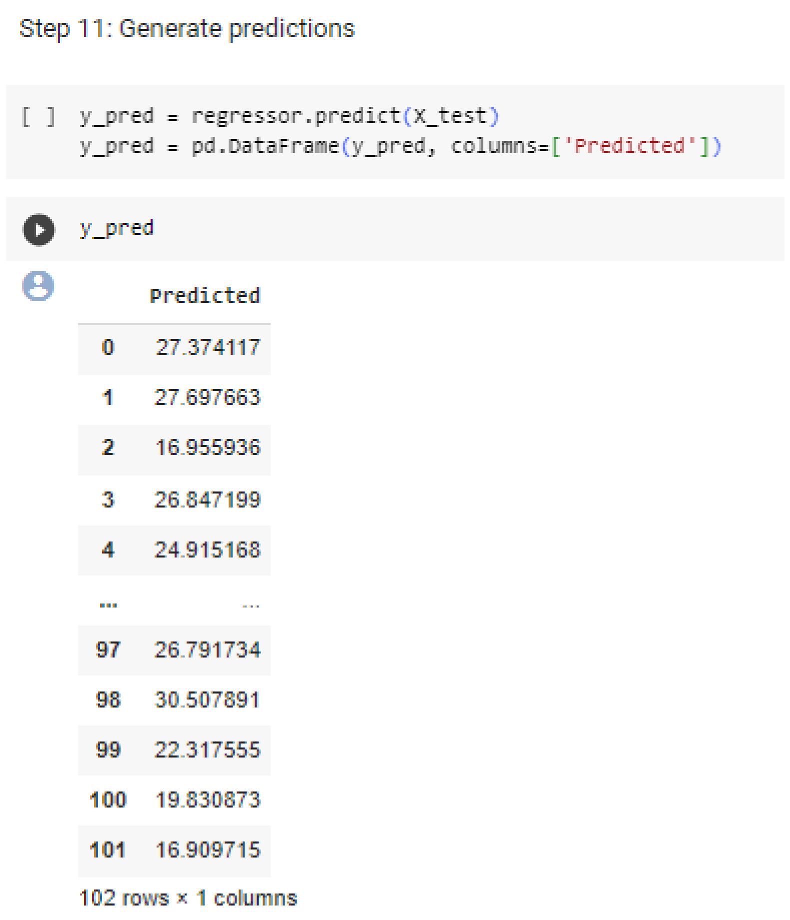
Step 10: Extract the regression coefficient

```
[ ]  print(regressor.coef_)
```

```
     [[-0.92441715]]
```

Step 11: Generate predictions

```
[ ]  y_pred = regressor.predict(X_test)
     y_pred = pd.DataFrame(y_pred, columns=['Predicted'])
```



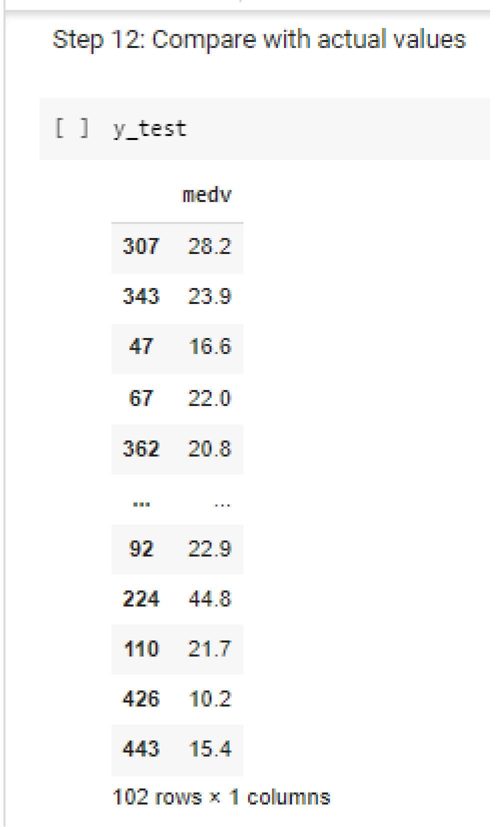
Step 11: Generate predictions

```
[ ] y_pred = regressor.predict(X_test)
    y_pred = pd.DataFrame(y_pred, columns=['Predicted'])
```

```
y_pred
```

	Predicted
0	27.374117
1	27.697663
2	16.955936
3	26.847199
4	24.915168
...	...
97	26.791734
98	30.507891
99	22.317555
100	19.830873
101	16.909715

102 rows × 1 columns

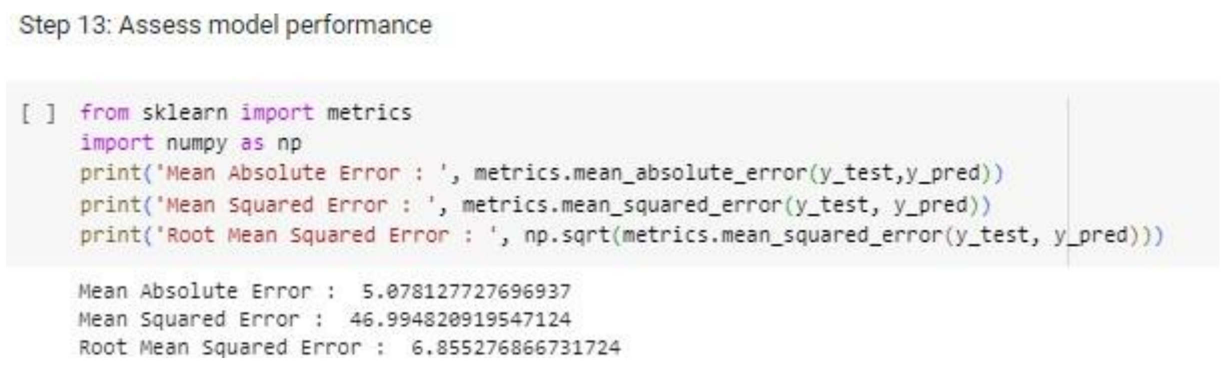


Step 12: Compare with actual values

```
[ ] y_test
```

	medv
307	28.2
343	23.9
47	16.6
67	22.0
362	20.8
...	...
92	22.9
224	44.8
110	21.7
426	10.2
443	15.4

102 rows × 1 columns



Step 13: Assess model performance

```
[ ] from sklearn import metrics
    import numpy as np
    print('Mean Absolute Error : ', metrics.mean_absolute_error(y_test,y_pred))
    print('Mean Squared Error : ', metrics.mean_squared_error(y_test, y_pred))
    print('Root Mean Squared Error : ', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error :  5.078127727696937
Mean Squared Error :  46.994820919547124
Root Mean Squared Error :  6.855276866731724
```

Conclusion: We have successfully implemented and executed some of the basic programs of simple linear regression successfully.

Practical No. 10

Aim: To implement multilinear regression in python

Theory:

Multiple linear regression (MLR) is used to determine a mathematical relationship among several random variables.[1] In other terms, MLR examines how multiple independent variables are related to one dependent variable. Once each of the independent factors has been determined to predict the dependent variable, the information on the multiple variables can be used to create an accurate prediction on the level of effect they have on the outcome variable. The model creates a relationship in the form of a straight line (linear) that best approximates all the individual data points.

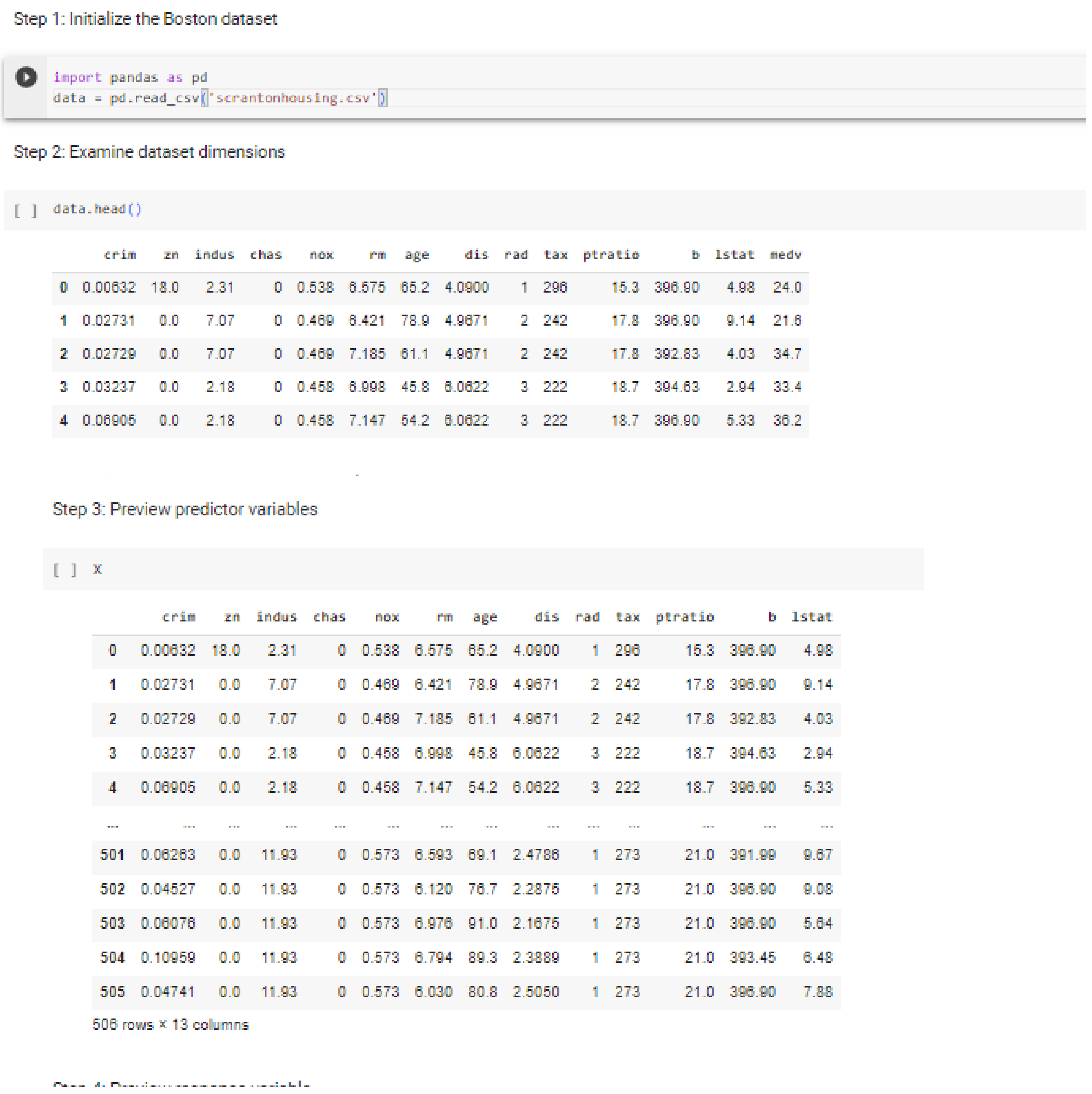
Referring to the MLR equation above, in our example:

- y_i = dependent variable—the price of XOM
- x_{i1} = interest rates
- x_{i2} = oil price
- x_{i3} = value of S&P 500 index
- x_{i4}= price of oil futures
- B_0 = y-intercept at time zero
- B_1 = regression coefficient that measures a unit change in the dependent variable whenx_{i1} changes - the change in XOM price when interest rates change
- B_2 = coefficient value that measures a unit change in the dependent variable whenx_{i2} changes—the change in XOM price when oil prices change

The least-squares estimates—B_0, B_1, B_2…B_p—are usually computed by statistical software. As many variables can be included in the regression model in which each independent variable is differentiated with a number—1,2, 3, 4...p. The multiple regression model allows an analyst to predict an outcome based on information provided on multiple explanatory variables.

Still, the model is not always perfectly accurate as each data point can differ slightly from the outcome predicted by the model. The residual value, E, which is the difference between the actual outcome and the predicted outcome, is included in the model to account for such slight variations.

Program



Step 1: Initialize the Boston dataset

```
import pandas as pd
data = pd.read_csv('scrantonhousing.csv')
```

Step 2: Examine dataset dimensions

```
[ ] data.head()
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

Step 3: Preview predictor variables

```
[ ] x
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88

506 rows × 13 columns

Step 4: Preview response variable

Step 4: Preview response variable

```
[ ] y
```

	medv
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2
...	...
501	22.4
502	20.6
503	23.9
504	22.0
505	11.9

506 rows × 1 columns

Step 5: Partition data for training and testing

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=5)
```

Step 6: Review dimensions of training and test datasets

```
[ ] print (X_train.shape)
    print (X_test.shape)
    print (y_train.shape)
    print (y_test.shape)
```

```
(404, 13)
(102, 13)
(404, 1)
(102, 1)
```

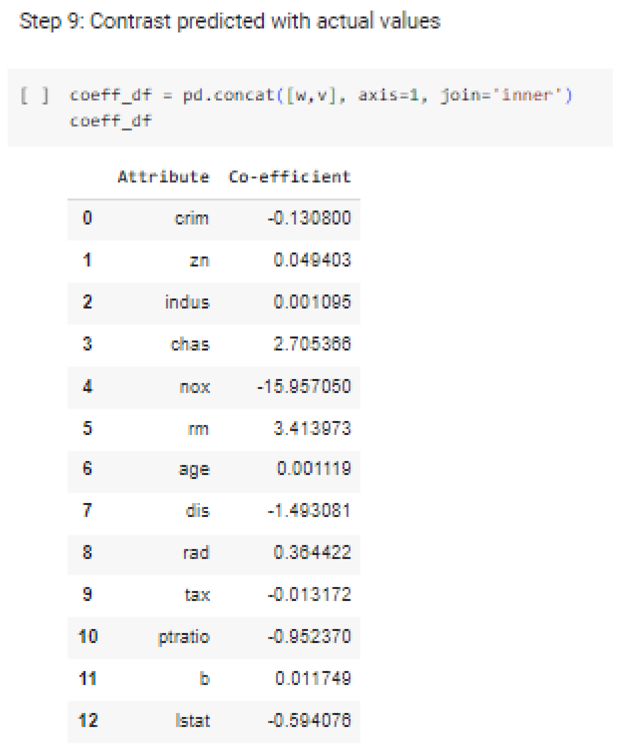
Step 7: Commence model training

```
[ ] from sklearn.linear_model import LinearRegression
    regressor = LinearRegression()
    regressor.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

Step 8: Examine chosen model coefficients

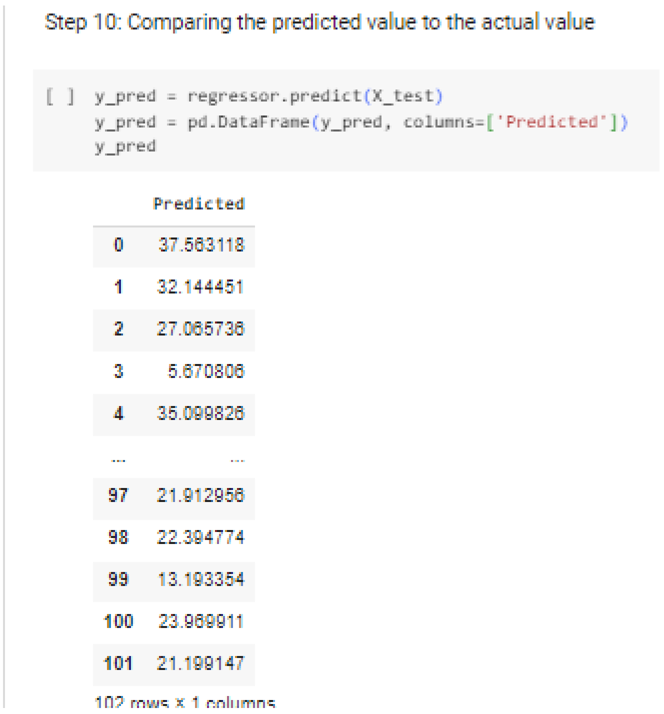
```
[ ] v = pd.DataFrame(regressor.coef_,index=['Co-efficient']).transpose()
    w = pd.DataFrame(X.columns, columns=['Attribute'])
```



Step 9: Contrast predicted with actual values

```
[ ] coeff_df = pd.concat([w,v], axis=1, join='inner')
    coeff_df
```

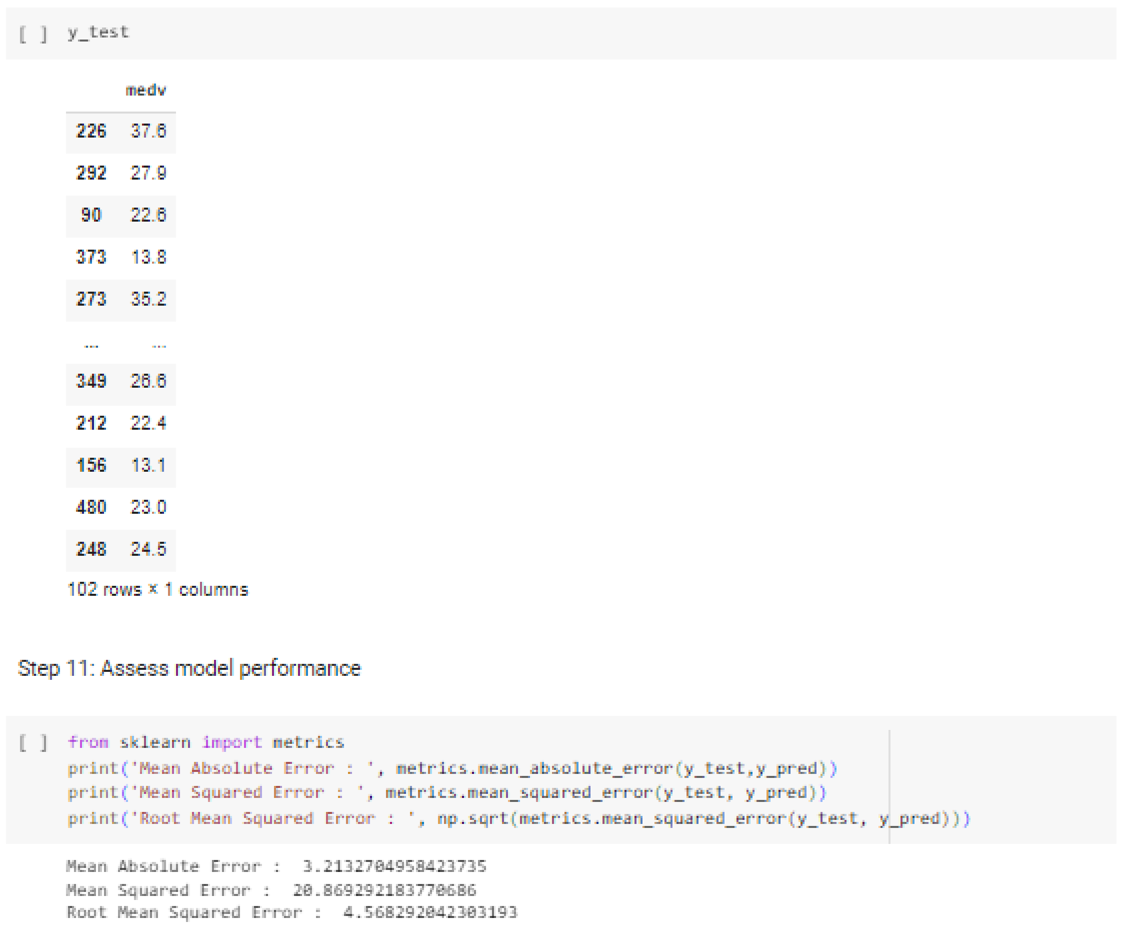
```
     Attribute  Co-efficient
0         crim     -0.130800
1           zn      0.049403
2        indus      0.001095
3         chas      2.705366
4          nox    -15.957050
5           rm      3.413973
6          age      0.001119
7          dis     -1.493081
8          rad      0.364422
9          tax     -0.013172
10     ptratio     -0.952370
11           b      0.011749
12       lstat     -0.594076
```



Step 10: Comparing the predicted value to the actual value

```
[ ] y_pred = regressor.predict(X_test)
    y_pred = pd.DataFrame(y_pred, columns=['Predicted'])
    y_pred
```

```
     Predicted
0    37.583118
1    32.144451
2    27.065736
3     5.670806
4    35.099826
...        ...
97   21.912956
98   22.394774
99   13.193354
100  23.969911
101  21.199147
102 rows × 1 columns
```



```
[ ] y_test
```

	medv
226	37.6
292	27.9
90	22.6
373	13.8
273	35.2
...	...
349	26.6
212	22.4
156	13.1
480	23.0
248	24.5

102 rows × 1 columns

Step 11: Assess model performance

```
[ ] from sklearn import metrics
    print('Mean Absolute Error : ', metrics.mean_absolute_error(y_test,y_pred))
    print('Mean Squared Error : ', metrics.mean_squared_error(y_test, y_pred))
    print('Root Mean Squared Error : ', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error :  3.2132704958423735
Mean Squared Error :  20.869292183770686
Root Mean Squared Error :  4.568292042303193
```

Conclusion: We have successfully implemented and executed some of the basic programs for multiple linear regression successfully.

Practical No. 11

Aim: To implement logistic regression in python

Theory:

Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class or not. It is a kind of statistical algorithm, which analyze the relationship between a set of independent variables and the dependent binary variables. It is a powerful tool for decision-making. For example email spam or not. Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class. It is used for classification algorithms its name is logistic regression. it's referred to as regression because it takes the output of the linear regression function as input and uses a sigmoid function to estimate the probability for the given class. The difference between linear regression and logistic regression is that linear regression output is the continuous value that can be anything while logistic regression predicts the probability that an instance belongs to a given class or not.

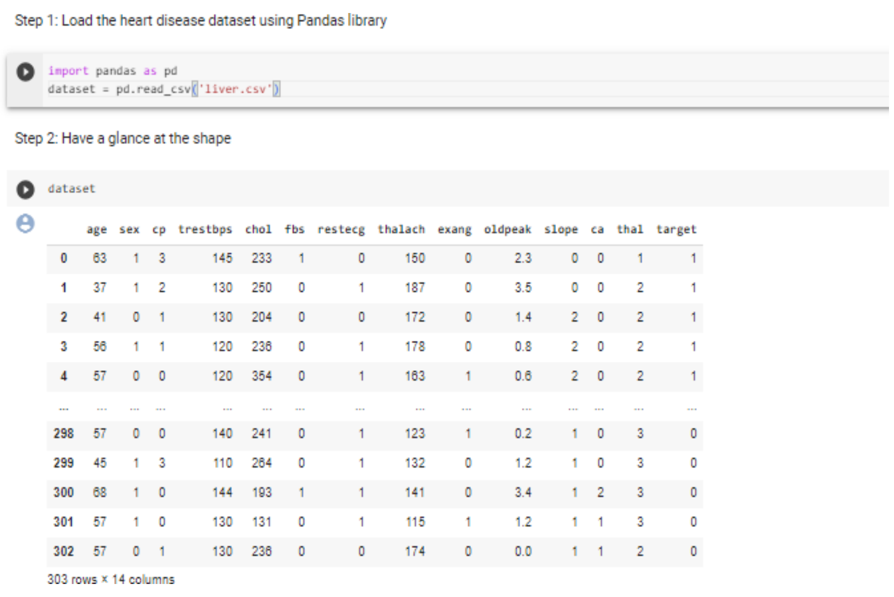
Logistic Regression:

It is used for predicting the categorical dependent variable using a given set of independent variables.

- Logistic regression predicts the output of a categorical dependent variable. Therefore theoutcome must be a categorical or discrete value.
- It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the likelihood of something such as whetherthe cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
- Logistic Regression can be used to classify the observations using different types of dataand can easily determine the most effective variables used for the classification.

- **Vector:** A **vector** is a basic data structure which plays an important role in R programming. In R, a sequence of elements which share the same data type is known as vector. A vector supports logical, integer, double, character, complex, or raw data type. The elements which are contained in vector known as **components** of the vector. We can check the type of vector with the help of the **typeof()** function.

Program



Step 1: Load the heart disease dataset using Pandas library

```python
import pandas as pd
dataset = pd.read_csv('liver.csv')
```

Step 2: Have a glance at the shape

```python
dataset
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  ca  thal  target
  0   63    1   3       145   233    1        0      150      0      2.3      0   0     1       1
  1   37    1   2       130   250    0        1      187      0      3.5      0   0     2       1
  2   41    0   1       130   204    0        0      172      0      1.4      2   0     2       1
  3   56    1   1       120   236    0        1      178      0      0.8      2   0     2       1
  4   57    0   0       120   354    0        1      163      1      0.6      2   0     2       1
...  ...  ...  ..       ...   ...  ...      ...      ...    ...      ...    ...  ..   ...     ...
298   57    0   0       140   241    0        1      123      1      0.2      1   0     3       0
299   45    1   3       110   264    0        1      132      0      1.2      1   0     3       0
300   68    1   0       144   193    1        1      141      0      3.4      1   2     3       0
301   57    1   0       130   131    0        1      115      1      1.2      1   1     3       0
302   57    0   1       130   236    0        0      174      0      0.0      1   1     2       0
303 rows × 14 columns
```

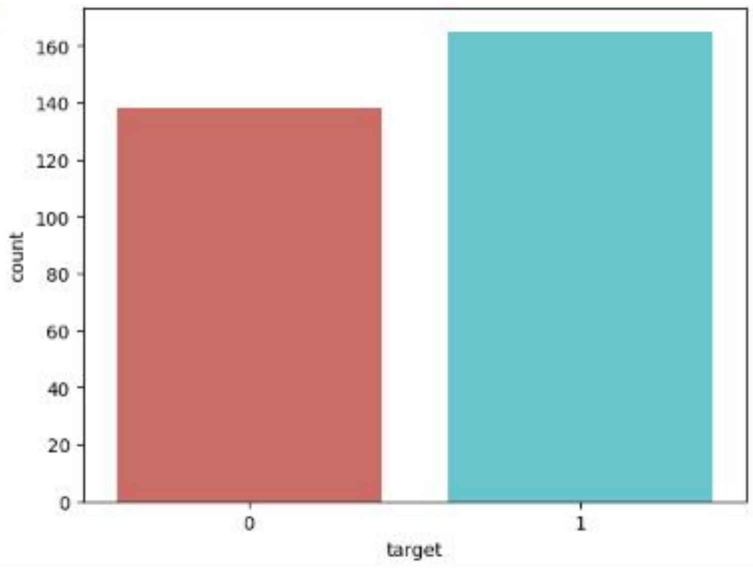
```
[ ] print(dataset.shape)

    (303, 14)
```

```
[ ] X = pd.DataFrame(dataset.iloc[:,:-1])
    y = pd.DataFrame(dataset.iloc[:,-1])
```

Step 4: Visualize the change in the variables

```
[ ] dataset['target'].value_counts()

    1    165
    0    138
    Name: target, dtype: int64
```

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.countplot (x= 'target',data = dataset, palette = 'hls')
plt.show()
```



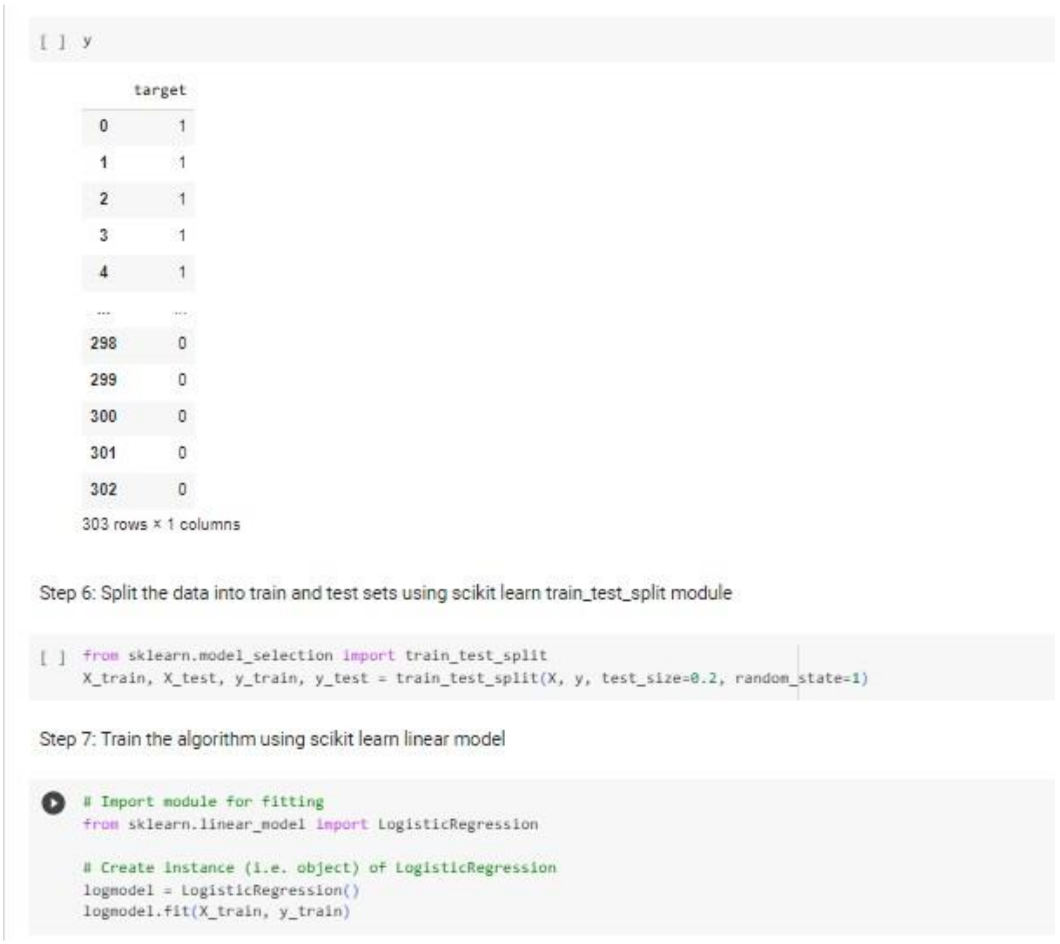
Step 5: Divide the data into independent and dependent variables

```
[ ] X = pd.DataFrame(dataset.iloc[:,:-1])
    y = pd.DataFrame(dataset.iloc[:,-1])
```

```
[ ] X
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2

303 rows × 13 columns



```
[ ] y
```

	target
0	1
1	1
2	1
3	1
4	1
...	...
298	0
299	0
300	0
301	0
302	0

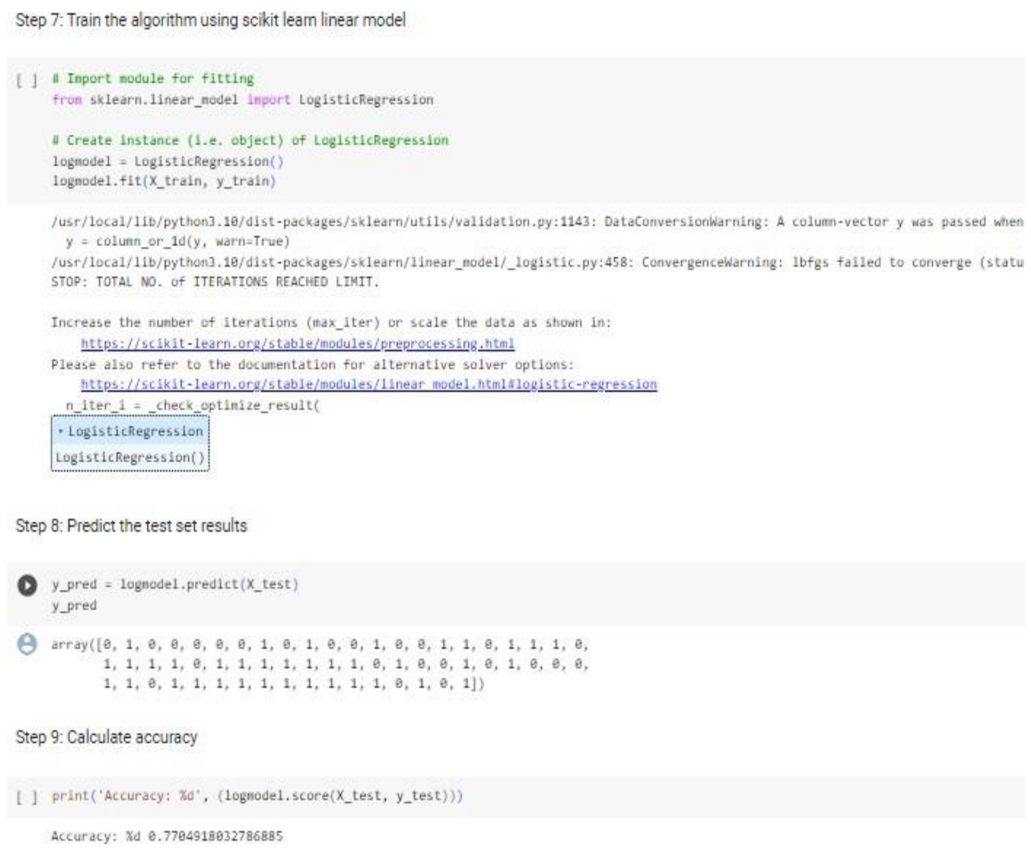
303 rows × 1 columns

Step 6: Split the data into train and test sets using scikit learn train_test_split module

```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

Step 7: Train the algorithm using scikit learn linear model

```
# Import module for fitting
from sklearn.linear_model import LogisticRegression

# Create instance (i.e. object) of LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train, y_train)
```



Step 7: Train the algorithm using scikit learn linear model

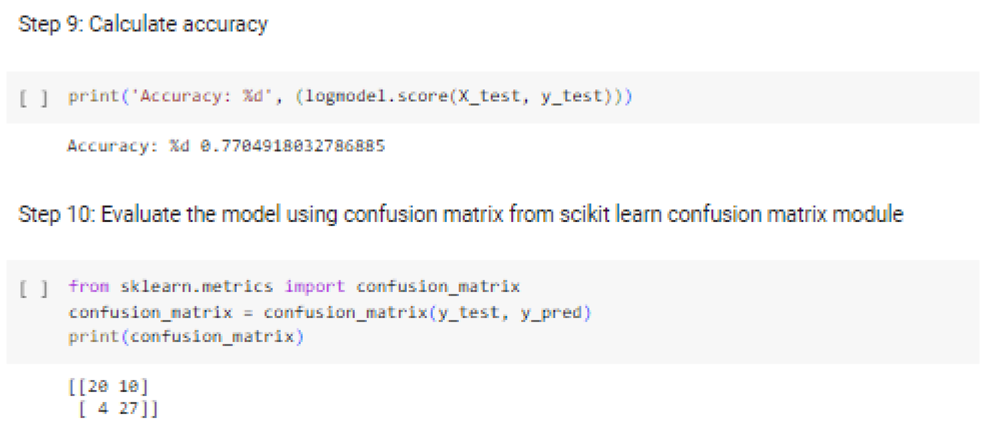
```
[ ] # Import module for fitting
    from sklearn.linear_model import LogisticRegression

    # Create instance (i.e. object) of LogisticRegression
    logmodel = LogisticRegression()
    logmodel.fit(X_train, y_train)

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (statu
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
▸ LogisticRegression
LogisticRegression()
```

Step 8: Predict the test set results

```
y_pred = logmodel.predict(X_test)
y_pred

array([0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1])
```

Step 9: Calculate accuracy

```
[ ] print('Accuracy: %d', (logmodel.score(X_test, y_test)))

Accuracy: %d 0.7704918032786885
```



Step 9: Calculate accuracy

```
[ ] print('Accuracy: %d', (logmodel.score(X_test, y_test)))
```

```
Accuracy: %d 0.7704918032786885
```

Step 10: Evaluate the model using confusion matrix from scikit learn confusion matrix module

```
[ ] from sklearn.metrics import confusion_matrix
    confusion_matrix = confusion_matrix(y_test, y_pred)
    print(confusion_matrix)
```

```
[[20 10]
 [ 4 27]]
```

Conclusion: We have successfully implemented and executed some of the basic program for logistic regression successfully.

Practical No.: 12

Aim: Write a program to calculate the gradient descent and local minima in Linear regression.

Theory:

Gradient descent is an optimization algorithm used in machine learning to minimize the cost function by iteratively adjusting parameters in the direction of the negative gradient, aiming to find the optimal set of parameters.

The cost function represents the discrepancy between the predicted output of the model and the actual output. The goal of gradient descent is to find the set of parameters that minimizes this discrepancy and improves the model's performance. The algorithm operates by calculating the gradient of the cost function, which indicates the direction and magnitude of steepest ascent. However, since the objective is to minimize the cost function, gradient descent moves in the opposite direction of the gradient, known as the negative gradient direction.

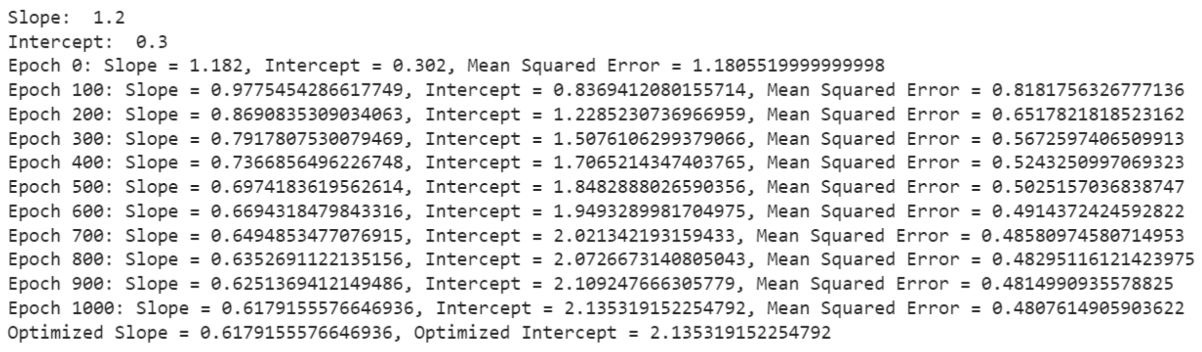
The point in a curve which is minimum when compared to its preceding and succeeding points is called local minima. The cost function may consist of many minimum points. The gradient may settle on any one of the minima, which depends on the initial point (i.e initial parameters(theta)) and the learning rate. Therefore, the optimization may converge to different points with different starting points and learning rate.

Program:

1.
```
import random
X = [1, 2, 3, 4, 5]  # Independent variable (single value)
y = [2, 4, 5, 4, 5]  # Dependent variable (target)
slope = 1.2  #random.random()
intercept = 0.3 #random.random()
print("Slope: ", slope)
print("Intercept: ", intercept)
learning_rate = 0.01
epochs = 1001
for epoch in range(epochs):
    gradient_slope = 0
    gradient_intercept = 0
    for i in range(len(X)):
        y_pred = slope * X[i] + intercept
        error = y_pred - y[i]
        gradient_slope += 2 * X[i] * error
        gradient_intercept += 2 * error
    slope -= learning_rate * (gradient_slope / len(X))
    intercept -= learning_rate * (gradient_intercept / len(X))
    total_error = 0
```

```
        for i in range(len(X)):
            y_pred = slope * X[i] + intercept
            total_error += (y_pred - y[i]) ** 2
        mean_squared_error = total_error / len(X)
        if epoch % 100 == 0:
            print(f"Epoch {epoch}: Slope = {slope}, Intercept = {intercept}, Mean Squared
    Error = {mean_squared_error}")
    print(f"Optimized Slope = {slope}, Optimized Intercept = {intercept}")
```

Output:



```
Slope:  1.2
Intercept:  0.3
Epoch 0: Slope = 1.182, Intercept = 0.302, Mean Squared Error = 1.1805519999999998
Epoch 100: Slope = 0.9775454286617749, Intercept = 0.8369412080155714, Mean Squared Error = 0.8181756326777136
Epoch 200: Slope = 0.8690835309034063, Intercept = 1.2285230736966959, Mean Squared Error = 0.6517821818523162
Epoch 300: Slope = 0.7917807530079469, Intercept = 1.5076106299379066, Mean Squared Error = 0.5672597406509913
Epoch 400: Slope = 0.7366856496226748, Intercept = 1.7065214347403765, Mean Squared Error = 0.5243250997069323
Epoch 500: Slope = 0.6974183619562614, Intercept = 1.8482888026590356, Mean Squared Error = 0.5025157036838747
Epoch 600: Slope = 0.6694318479843316, Intercept = 1.9493289981704975, Mean Squared Error = 0.4914372424592822
Epoch 700: Slope = 0.6494853477076915, Intercept = 2.021342193159433, Mean Squared Error = 0.48580974580714953
Epoch 800: Slope = 0.6352691122135156, Intercept = 2.0726673140805043, Mean Squared Error = 0.48295116121423975
Epoch 900: Slope = 0.6251369412149486, Intercept = 2.109247666305779, Mean Squared Error = 0.4814990935578825
Epoch 1000: Slope = 0.6179155576646936, Intercept = 2.135319152254792, Mean Squared Error = 0.4807614905903622
Optimized Slope = 0.6179155576646936, Optimized Intercept = 2.135319152254792
```

Conclusion: We have successfully learned about the gradient descent and local minima, and also execute or implement the program successfully.