SRI RAMAKRISHNA ENGINEERING COLLEGE
[Educational Service: SNR Sons Charitable Trust]
[Autonomous Institution, Reaccredited by NAAC with ‘A+’ Grade]
[Approved by AICTE and Permanently Affiliated to Anna University, Chennai]
[ISO 9001:2015 Certified and all eligible programmes Accredited by NBA]
VATTAMALAIPALAYAM, N.G.G.O. COLONY POST, COIMBATORE – 641 022.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this is the bonafide record of works done by Mr./Mrs. ____________________________________ in 20CS2E14 – DATA SCIENCE USING R (Theory Cum Lab) LABORATORY of this Institution for VI Semester during the Academic Year 2022 – 2023.

Faculty In-Charge	**HOD – CSE**
(Mrs.S.EZHILIN FREEDA AP(Sl.Gr)/CSE)	**(Dr.A.GRACESELVARANI,Prof/Head)**

Date:

Register Number: ______________________

Submitted for the VI Semester B.E.-CSE Practical Examination held on ____________________ during the Academic Year 2022 – 2023.

Internal Examiner	**External Examiner**

<u>**INDEX**</u>

EXP NO. : 01	**INTRODUCTION TO R**
DATE : 16.12.2022	

AIM:

To study about the basic commands and graphics available in R language.

a) BASIC COMMANDS:

DATA TYPES:

DESCRIPTION:

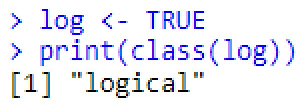
1.Logical:

Logical data type is used to represent values like TRUE and FALSE values.

Program:

```
log <- TRUE
print(class(log))
```

Output:



```
> log <- TRUE
> print(class(log))
[1] "logical"
```

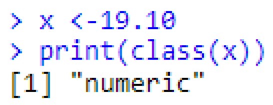
2.Numeric:

Numeric datatype is used to represent values like 17.89, 50, 10901 etc.

Program:

```
x <-19.10
print(class(x))
```

Output:



```
> x <-19.10
> print(class(x))
[1] "numeric"
```

3.Integer:

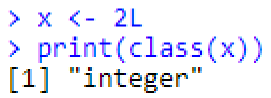
Integer datatype is used to represent integer values.

Program:

x <- 2L

print(class(x))

Output:



```
> x <- 2L
> print(class(x))
[1] "integer"
```

4.Character:

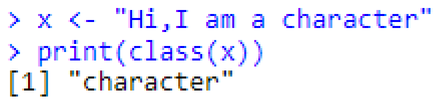
Character datatype is used to represent integer values within single, double and triple quotes.

Program:

x <- “Hi,I am a character”

print(class(x))

Output:



```
> x <- "Hi,I am a character"
> print(class(x))
[1] "character"
```

5.Complex:

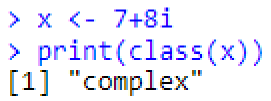
Complex datatype is used to represent complex values.

Program:

x <- 7+8i

print(class(x))

Output:



```
> x <- 7+8i
> print(class(x))
[1] "complex"
```

6.Raw:

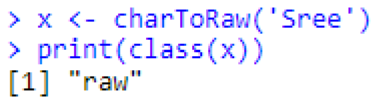
Raw datatype is used to represent values in the form of raw data.

Program:

x <- charToRaw(“Sree”)

print(class(x))

Output:



```
> x <- charToRaw('Sree')
> print(class(x))
[1] "raw"
```

R-DATA STRUCTURES:

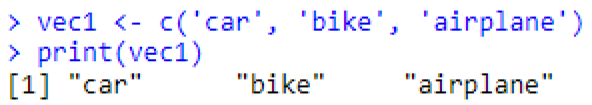
1.Vector:

- A vector is simply a list of items that are of the same data type.
- To combine the list of items to a vector, use the c() function and separate the items by commas.

Program:

```
vec1 <- c('car', 'bike', 'airplane')
print(vec1)
```

Output:



```
> vec1 <- c('car', 'bike', 'airplane')
> print(vec1)
[1] "car"      "bike"     "airplane"
```

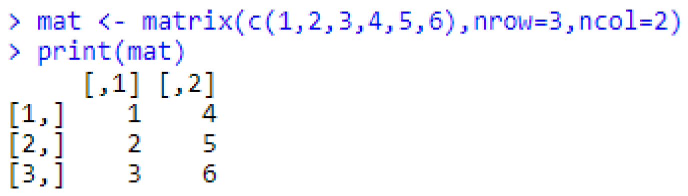
2.Matrices:

- A matrix is a two-dimensional data set with columns and rows.
- A column is a vertical representation of data, while a row is a horizontal representation of data.
- A matrix can be created with the matrix() function. Specify the nrow and ncol parameter to get the number of rows and columns.

Program:

```
mat <- matrix(c(1,2,3,4,5,6),nrow=3,ncol=2)
print(mat)
```

Output:



```
> mat <- matrix(c(1,2,3,4,5,6),nrow=3,ncol=2)
> print(mat)
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

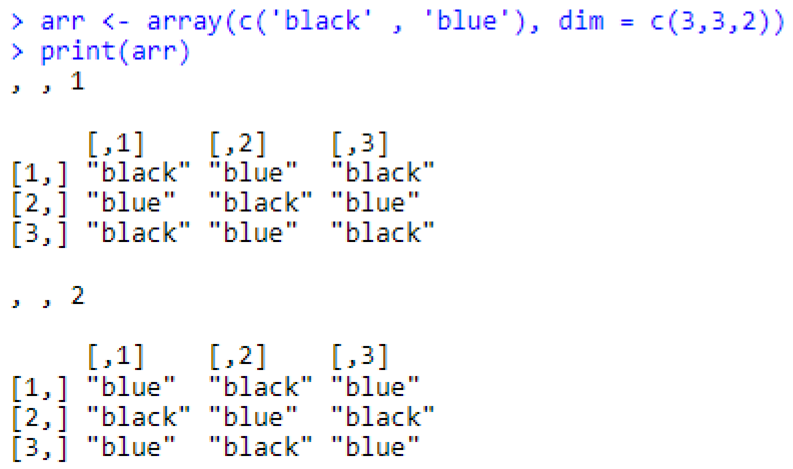
3.Arrays:

- Arrays can have more than two dimensions.
- Array() function is used to create an array and 'dim' parameter is used to specify the dimension.

Program:

```
arr <- array(c('black' , 'blue'), dim = c(3,3,2))
print(arr)
```

Output:



```
> arr <- array(c('black' , 'blue'), dim = c(3,3,2))
> print(arr)
, , 1

     [,1]    [,2]    [,3]   
[1,] "black" "blue"  "black"
[2,] "blue"  "black" "blue" 
[3,] "black" "blue"  "black"

, , 2

     [,1]    [,2]    [,3]   
[1,] "blue"  "black" "blue" 
[2,] "black" "blue"  "black"
[3,] "blue"  "black" "blue" 
```

4.List:

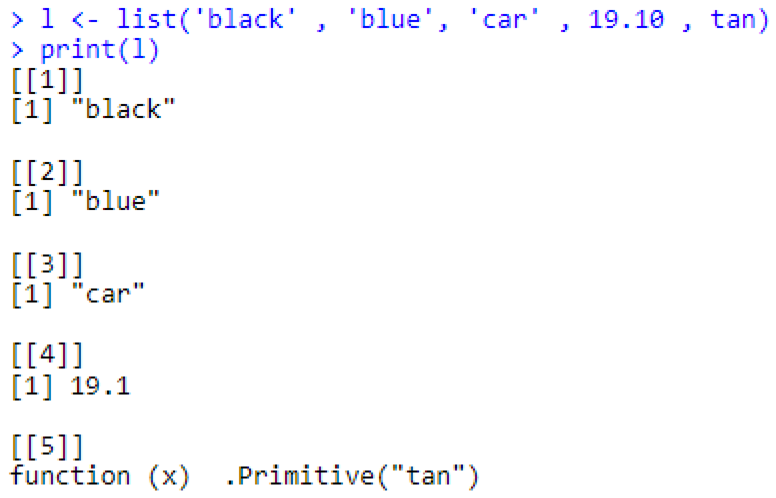
- ➢ A list in R can contain many different data types.
- ➢ A list is a collection of data which is ordered and changeable.
- ➢ list() function is used to create a list.

Program:

l <- list(‘black’ , ‘blue’, ‘car’ , 19.10 , tan)

print(l)

Output:



```
> l <- list('black' , 'blue', 'car' , 19.10 , tan)
> print(l)
[[1]]
[1] "black"

[[2]]
[1] "blue"

[[3]]
[1] "car"

[[4]]
[1] 19.1

[[5]]
function (x)  .Primitive("tan")
```

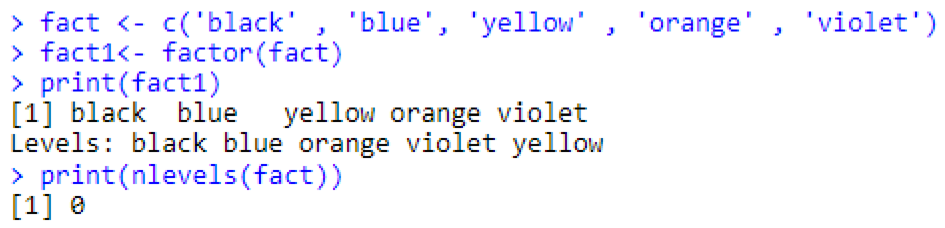
5.Factors:

- ➢ Factors are used to categorize data.
- ➢ They are created using vector and factor() functions.
- ➢ The nlevels() function gives the count of levels.

Program:

```
fact <- c(‘black’ , ‘blue’, ‘yellow’ , ‘orange’ , ‘violet’)
fact1<- factor(fact)
print(fact1)
print(nlevels(fact))
```

Output:



```
> fact <- c('black' , 'blue', 'yellow' , 'orange' , 'violet')
> fact1<- factor(fact)
> print(fact1)
[1] black  blue   yellow orange violet
Levels: black blue orange violet yellow
> print(nlevels(fact))
[1] 0
```

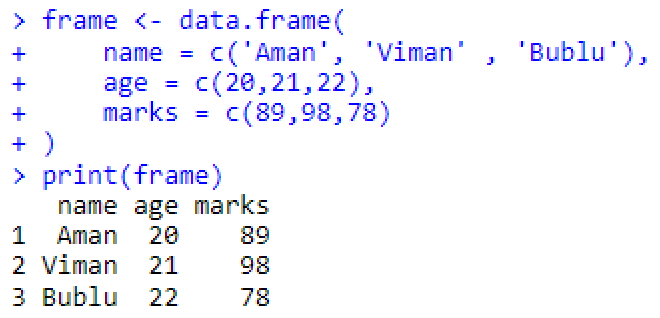
6.Data Frames:

- ➢ Data frames are data displayed in the format of a table.
- ➢ They can have different data types inside it.
- ➢ data.frame() function is used to create a data frame.

Program:

```
frame <- data.frame(
            name = c(‘Aman’, ‘Viman’ , ‘Bublu’),
            age = c(20,21,22),
            marks = c(89,98,78)
        )
print(frame)
```

Output:



```
> frame <- data.frame(
+     name = c('Aman', 'Viman' , 'Bublu'),
+     age = c(20,21,22),
+     marks = c(89,98,78)
+ )
> print(frame)
   name age marks
1  Aman  20    89
2 Viman  21    98
3 Bublu  22    78
```

b)GRAPHICS:

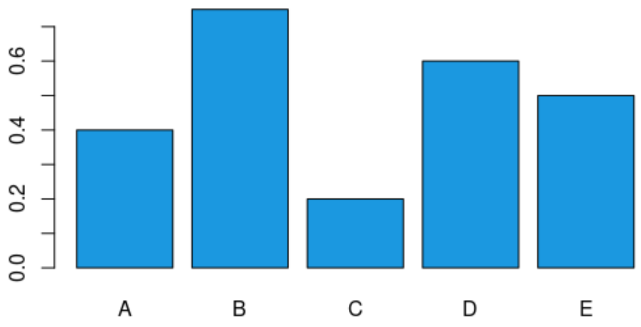
1.Bar Plot:

- ➢ A barplot (or barchart; bar graph) illustrates the association between a numeric and a categorical variable.
- ➢ The barplot represents each category as a bar and reflects the corresponding numeric value with the bar’s size.

Program:

```
values <- c(0.4, 0.75, 0.2, 0.6, 0.5)
group <- LETTERS[1:5]
barplot(values,col = "#1b98e0",names.arg = group)
```

Output:



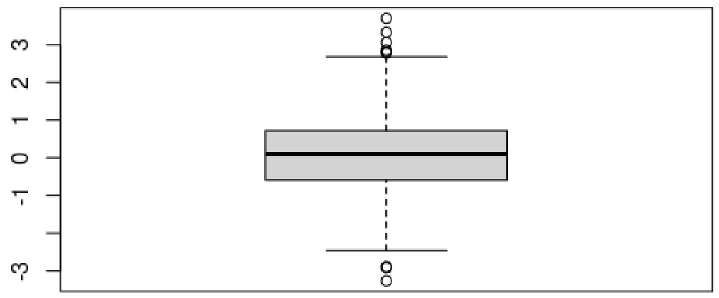
2.Box Plot:

- ➢ A boxplot (or box-and-whisker plot) displays the distribution of a numerical variable based on five summary statistics:
  - ○ minimum non-outlier
  - ○ first quartile
  - ○ median
  - ○ third quartile
  - ○ maximum non-outlier.
- ➢ Furthermore, boxplots show the positioning of outliers and whether the data is skewed.

Program:

```
set.seed(8642)
x <- rnorm(1000)
boxplot(x)
```

Output:



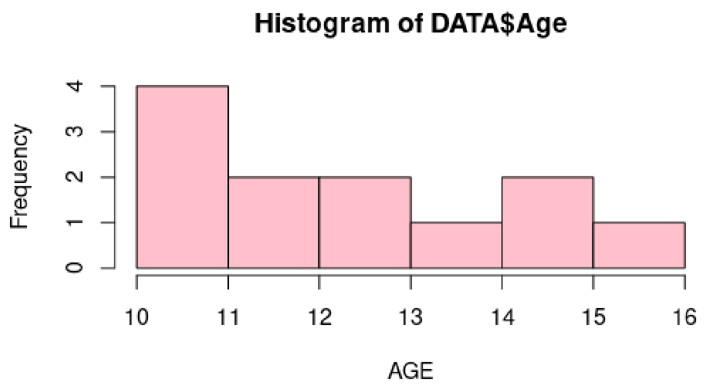
3.Histogram:

- ➢ A histogram groups continuous data into ranges and plots this data as bars.
- ➢ The height of each bar shows the amount of observations within each range.

Program:

```
frame <- data.frame(
                    name = c('Aman', 'Viman' , 'Bublu'),
                    age = c(20,21,22),
                    marks = c(89,98,78)
                )
print(frame)
```

Output:



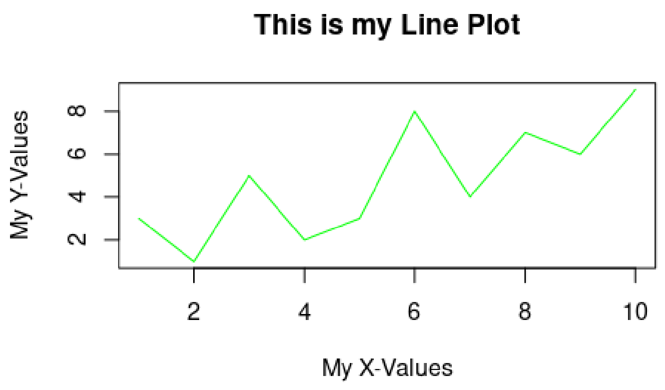
4.Line Plot:

- ➢ A line plot (or line graph; line chart) visualizes values along a sequence (e.g. over time).
- ➢ Line plots consist of an x-axis and a y-axis.
- ➢ The x-axis usually displays the sequence and the y-axis the values corresponding to each point of the sequence.

Program:

```
x <- 1:10
y1 <- c(3, 1, 5, 2, 3, 8, 4, 7, 6, 9)
plot(x, y1, type = "l",main = "This is my Line Plot",xlab = "My
X-Values",ylab="My Y-Values",col="green")
```

Output:



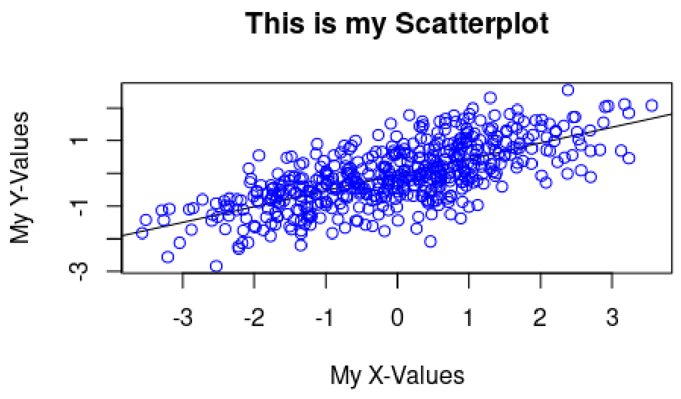
5.Scatter Plot:

➢ A scatterplot (or scatter plot, scatter graph, scatter chart, scattergram, scatter diagram) displays two numerical variables with points, whereby each point represents the value of one variable on the x-axis and the value of the other variable on the y-axis.

Program:

```
set.seed(42424)
x <- rnorm(500)
y <- x + rnorm(500)
plot(y,x,col = "blue",abline(lm(x~y)),main = "This is my Scatterplot", xlab =
    "My X-Values", ylab = "My Y-Values")
```

Output:



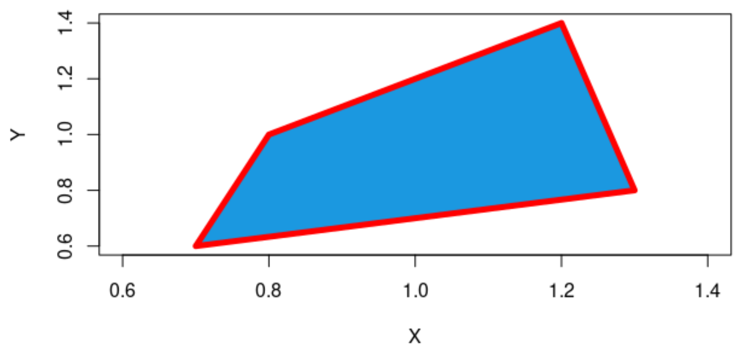
6.Polygon Plot:

➢ A polygon plot displays a plane geometric figure (i.e., a polygon) within the plot.

Program:

```
plot(1, 1, col = "white", xlab = "X", ylab = "Y")                    # Draw empty plot
polygon(x = c(0.7, 1.3, 1.2, 0.8),        # X-Coordinates of polygon
y = c(0.6, 0.8, 1.4, 1),                  # Y-Coordinates of polygon
col = "#1b98e0",                          # Color of polygon
border = "red",                           # Color of polygon border
lwd = 5)                                  # Thickness of border
```

Output:



RESULT:

Thus, the basic commands in R language have been executed and verified successfully.

EXP NO. : 02	GETTING USED TO R : DESCRIBING DATA
DATE : 23.12.2022	

AIM:

To study about the basic commands to describe data in R language.

a)Viewing and manipulating data:

ALGORITHM:

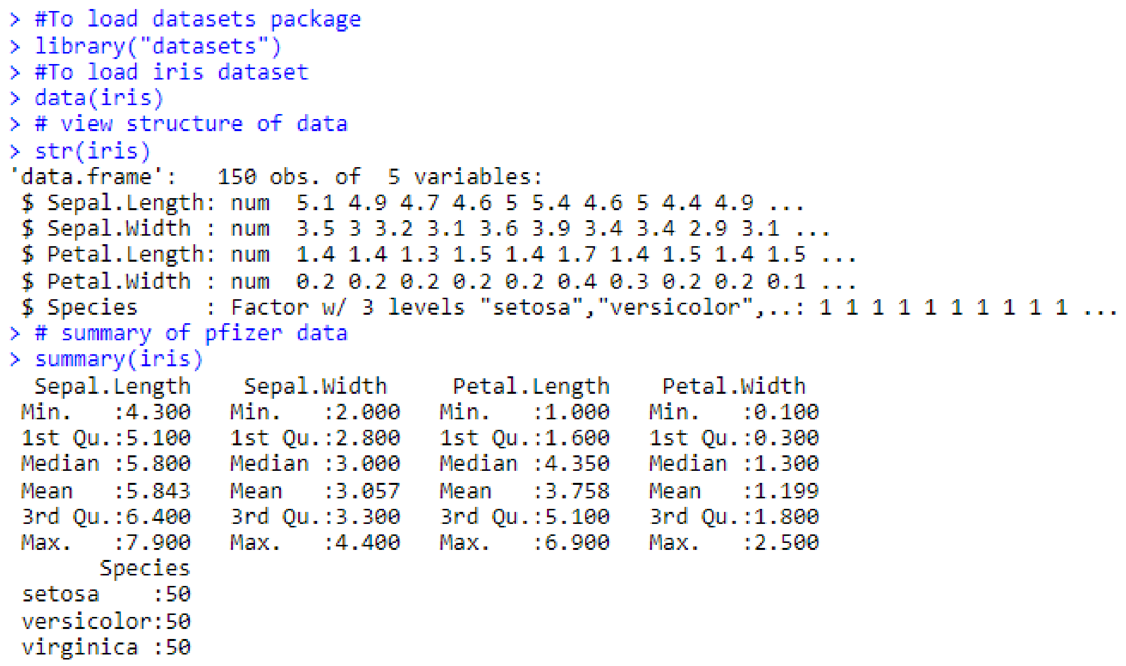
Step 1 : Import the library "dpylr" to manipulate data.
Step 2 : Import the iris dataset using the library datasets.
Step 3 : To view the structure of the dataset use str () function.
Step 4 : To find insights (mean, median, mode, etc.) from a dataset use summary () function.
Step 5 : To select certain columns use select () function.
Step 6 : To fetch rows with matching criteria in the dataset use filter () function.
Step 7 : To creates new columns and preserves the existing columns in a dataset use mutate () function.
Step 8 : To sort rows by variables in both an ascending and descending order use arrange () function.
Step 9 : To group observations within a dataset by one or more variables use group_by function.
Step 10 : To wrap multiple functions together we can use pipe operator (%>%).

PROGRAM:

```
install.packages("dplyr")
#To load dplyr package
library("dplyr")
#To load datasets package
library("datasets")
#To load iris dataset
data(iris)
# view structure of data
str(iris)
# summary of pfizer data
summary(iris)
#To select the following columns
selected <- select(iris, Sepal.Length, Sepal.Width, Petal.Length)
head(selected)
#To select the first 3 rows with Species as setosa
```

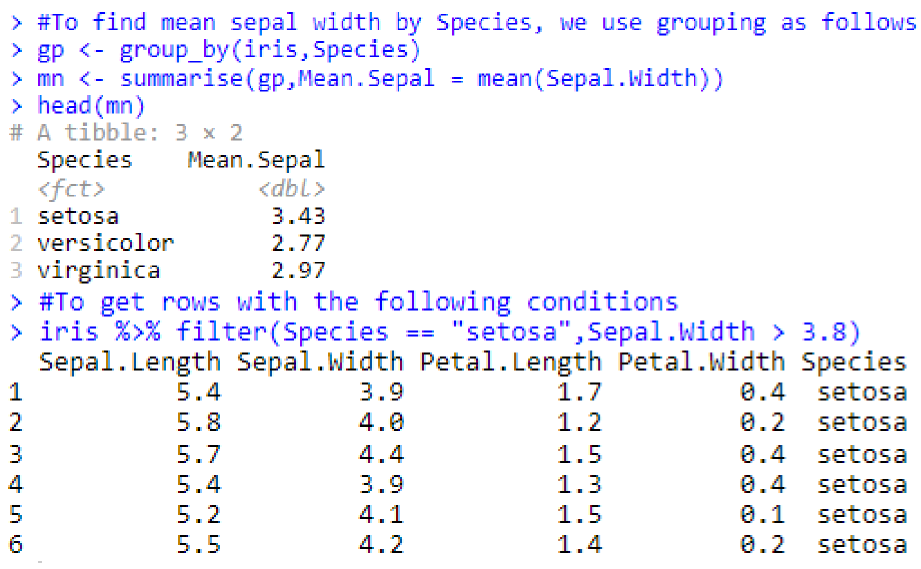
```
filtered <- filter(iris, Species == "setosa" )
head(filtered,3)
#To create a column “Greater.Half” which stores TRUE if given condition is TRUE
col1 <- mutate(iris, Greater.Half = Sepal.Width > 0.5 * Sepal.Length)
tail(col1)
#To arrange Sepal Width in ascending order
arranged <- arrange(col1, Sepal.Width)
head(arranged)
#To arrange Sepal Width in descending order
arranged <- arrange(col1, desc(Sepal.Width))
head(arranged)
#To find mean sepal width by Species, we use grouping as follows
gp <- group_by(iris,Species)
mn <- summarise(gp,Mean.Sepal = mean(Sepal.Width))
head(mn)
#To get rows with the following conditions
iris %>% filter(Species == "setosa",Sepal.Width > 3.8)
```

OUTPUT:



```
> #To load datasets package
> library("datasets")
> #To load iris dataset
> data(iris)
> # view structure of data
> str(iris)
'data.frame':	150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
> # summary of pfizer data
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
 Median :5.800   Median :3.000   Median :4.350   Median :1.300  
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
       Species  
 setosa    :50  
 versicolor:50  
 virginica :50  
```

```
> #To select the following columns
> selected <- select(iris, Sepal.Length, Sepal.Width, Petal.Length)
> head(selected)
  Sepal.Length Sepal.Width Petal.Length
1          5.1         3.5          1.4
2          4.9         3.0          1.4
3          4.7         3.2          1.3
4          4.6         3.1          1.5
5          5.0         3.6          1.4
6          5.4         3.9          1.7
> #To select the first 3 rows with Species as setosa
> filtered <- filter(iris, Species == "setosa" )
> head(filtered,3)
  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
> #To create a column “Greater.Half” which stores TRUE if given condition is TRUE
> col1 <- mutate(iris, Greater.Half = Sepal.Width > 0.5 * Sepal.Length)
> tail(col1)
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
145          6.7         3.3          5.7         2.5 virginica
146          6.7         3.0          5.2         2.3 virginica
147          6.3         2.5          5.0         1.9 virginica
148          6.5         3.0          5.2         2.0 virginica
149          6.2         3.4          5.4         2.3 virginica
150          5.9         3.0          5.1         1.8 virginica
    Greater.Half
145        FALSE
146        FALSE
147        FALSE
148        FALSE
149         TRUE
150         TRUE
> #To arrange Sepal Width in ascending order
> arranged <- arrange(col1, Sepal.Width)
> head(arranged)
  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1          5.0         2.0          3.5         1.0 versicolor
2          6.0         2.2          4.0         1.0 versicolor
3          6.2         2.2          4.5         1.5 versicolor
4          6.0         2.2          5.0         1.5  virginica
5          4.5         2.3          1.3         0.3     setosa
6          5.5         2.3          4.0         1.3 versicolor
  Greater.Half
1        FALSE
2        FALSE
3        FALSE
4        FALSE
5         TRUE
6        FALSE
> #To arrange Sepal Width in descending order
> arranged <- arrange(col1, desc(Sepal.Width))
> head(arranged)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species Greater.Half
1          5.7         4.4          1.5         0.4  setosa         TRUE
2          5.5         4.2          1.4         0.2  setosa         TRUE
3          5.2         4.1          1.5         0.1  setosa         TRUE
4          5.8         4.0          1.2         0.2  setosa         TRUE
5          5.4         3.9          1.7         0.4  setosa         TRUE
6          5.4         3.9          1.3         0.4  setosa         TRUE
```



```
> #To find mean sepal width by Species, we use grouping as follows
> gp <- group_by(iris,Species)
> mn <- summarise(gp,Mean.Sepal = mean(Sepal.Width))
> head(mn)
# A tibble: 3 x 2
  Species    Mean.Sepal
  <fct>           <dbl>
1 setosa           3.43
2 versicolor       2.77
3 virginica        2.97
> #To get rows with the following conditions
> iris %>% filter(Species == "setosa",Sepal.Width > 3.8)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.4         3.9          1.7         0.4  setosa
2          5.8         4.0          1.2         0.2  setosa
3          5.7         4.4          1.5         0.4  setosa
4          5.4         3.9          1.3         0.4  setosa
5          5.2         4.1          1.5         0.1  setosa
6          5.5         4.2          1.4         0.2  setosa
```

b)Plotting data:

ALGORITHM:

Step 1 : Import the “iris” dataset using library dataset.

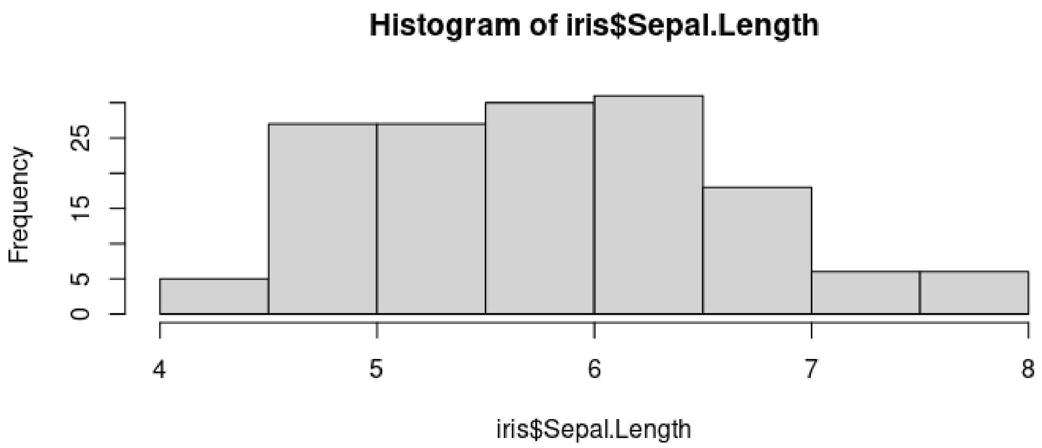
Step 2 : To plot the data in histogram use hist() function.

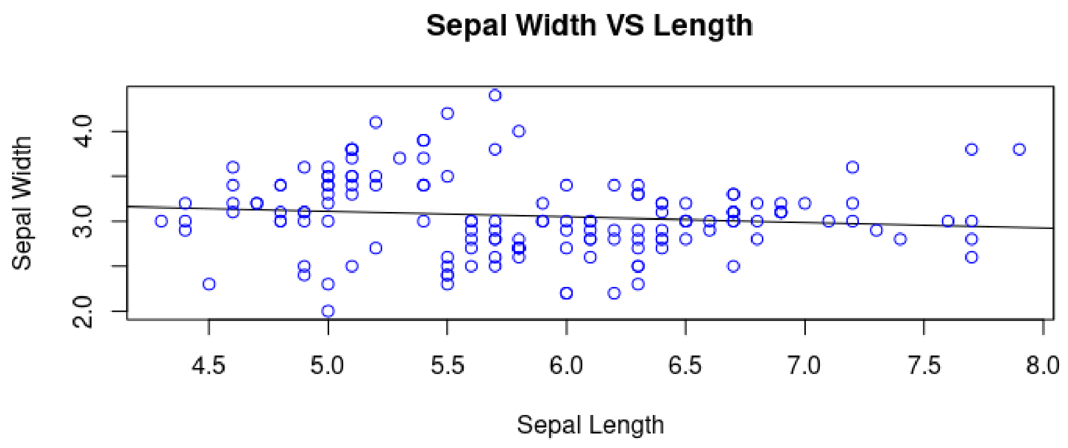
Step 3 : To plot the data in scatter plot use plot() function.

PROGRAM:

```
#To load datasets package
library("datasets")
#To load iris dataset
data(iris)
#Visualization using histogram
hist(iris$Sepal.Length)
#Visualization using scatterplot
x <- iris$Sepal.Length
y <- iris$Sepal.Width
plot(x,y,col = "blue",abline(lm(y~x)),main = "Sepal Width VS Length", xlab = "Sepal Length",
ylab = "Sepal Width")
```

OUTPUT:





c)Reading in your own data:

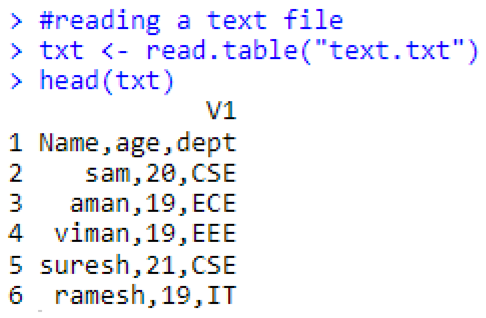
ALGORITHM:

Step 1 : Create your own dataset.
Step 2 : To read a text file use the function read.table().
Step 3 : To read an excel file use the function read.csv().

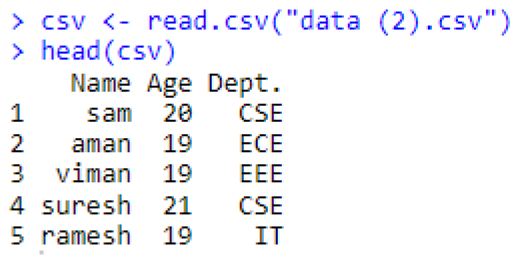
PROGRAM:

```
#reading a text file
txt <- read.table("text.txt")
head(txt)
#reading a csv file
csv <- read.csv("data (2).csv")
head(csv)
```

OUTPUT:



```
> #reading a text file
> txt <- read.table("text.txt")
> head(txt)
                V1
1 Name,age,dept
2    sam,20,CSE
3   aman,19,ECE
4  viman,19,EEE
5 suresh,21,CSE
6  ramesh,19,IT
```



```
> csv <- read.csv("data (2).csv")
> head(csv)
    Name Age Dept.
1    sam  20   CSE
2   aman  19   ECE
3  viman  19   EEE
4 suresh  21   CSE
5 ramesh  19    IT
```

RESULT:

Thus, the basic commands to describe data in R language have been executed and verified successfully.

EXP NO. : 3A	LINEAR REGRESSION
DATE : 06.01.2023	

AIM:

To write an R program to implement linear regression.

Linear Regression:

DESCRIPTION:

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 (X’s) values are known.

ALGORITHM:

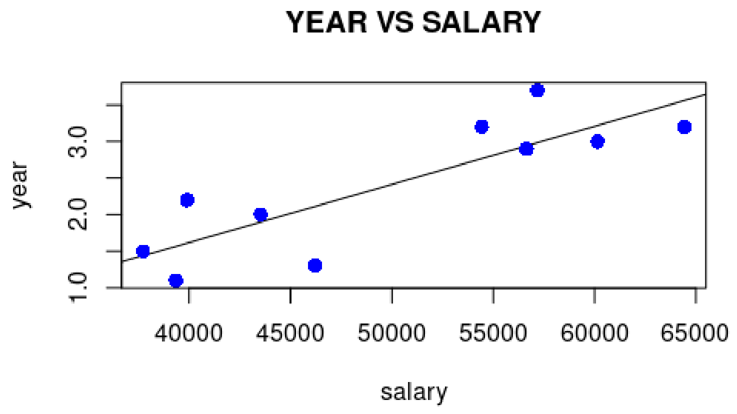
Step 1 : Create a csv file named as linear_regression and insert the midterm exam marks and final exam marks and import it.
Step 2 : Create 2 vectors named as x and y and assign the midterm exam marks and final exam marks to the vectors x and y respectively.
Step 3 : Create a relation between midterm and final exam marks using lm() function.
Step 4 : To predict final exam marks using midterm mark use predict() function and print the output.
Step 5 : To plot a scatter graph use plot function and pass the x vector, y vector and relation. Set the title of the graph using the main parameter and label the x and y axes using xlab and ylab.

PROGRAM:

```
x <- c(linear$Years)
y <- c(linear$Salary)
relation <- lm(y~x)
print(relation)
print(summary(relation))
cor(x,y)
b <- data.frame(x=1.2)
result <- predict(relation)
print(paste("When year is 1.2 the salary will be"))
plot(y,x,col="blue",main="YEAR VS
SALARY",abline(lm(x~y)),cex=1.3,pch=16,xlab="salary",ylab="year")
```

OUTPUT:

```
> x <- c(linear$Years)
> y <- c(linear$Salary)
> relation <- lm(y~x)
> print(relation)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
      28217         9021

> print(summary(relation))

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-8171.3 -3695.9  -717.2  4219.7  7362.1

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    28217       5130   5.501 0.000573 ***
x               9021       2003   4.503 0.001995 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5482 on 8 degrees of freedom
Multiple R-squared:  0.7171,    Adjusted R-squared:  0.6817
F-statistic: 20.28 on 1 and 8 DF,  p-value: 0.001995

> cor(x,y)
[1] 0.8468007

> print(paste("When year is 1.2 the salary will be"))
[1] "When year is 1.2 the salary will be"
> plot(y,x,col="blue",main="YEAR VS SALARY",abline(lm(x~y)),cex=1.
3,pch=16,xlab="salary",ylab="year")
```



RESULT:

Thus, the implementation of linear regression in R language has been executed and verified successfully.

EXP NO. : 3B	LOGISTIC REGRESSION
DATE : 20.01.2023	

AIM:

To write an R program to implement logistic regression.

Logistic Regression:

DESCRIPTION:

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

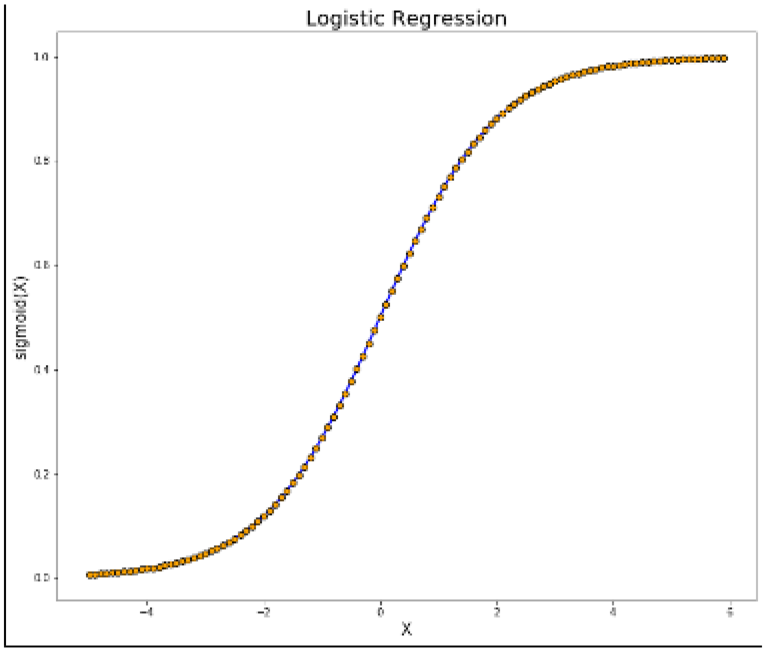
ALGORITHM:

Step 1 : Install the packages “caTools” and “ROCR”.
Step 2 : Load the Dataset and split the dataset into test and train.
Step 3 : Train the model using glm() function.
Step 4 : Analyze the dataset using summary() function.
Step 5 : Predict the test data using predict() function.
Step 6 : Evaluate the accuracy of the model using a confusion matrix.

PROGRAM:

```
install.packages("caTools")
install.packages("ROCR")
mark<-read.csv("logistic.csv")
View(mark)
library(caTools)
library(ROCR)
split <- sample.split(mark, SplitRatio = 0.8)
train_reg <- subset(mark, split == "TRUE")
test_reg <- subset(mark, split == "FALSE")
logistic_model <- glm(Admission ~ Maths + Physics,data = train_reg,family = "binomial")
summary(logistic_model)
predict_reg <- predict(logistic_model,test_reg, type = "response")
predict_reg <- ifelse(predict_reg >0.5, 1, 0)
table(test_reg$Admission, predict_reg)
predict_reg
missing_classerr <- mean(predict_reg != test_reg$Admission)
print(paste('Accuracy =', 1 - missing_classerr))
```

OUTPUT:



RESULT:

Thus, the implementation of logistic regression in R language has been executed and verified successfully.

EXP NO. : 3C	**MULTIPLE REGRESSION**
DATE : 21.01.2023	

AIM:

To write an R program to implement multiple regression.

Multiple Regression:

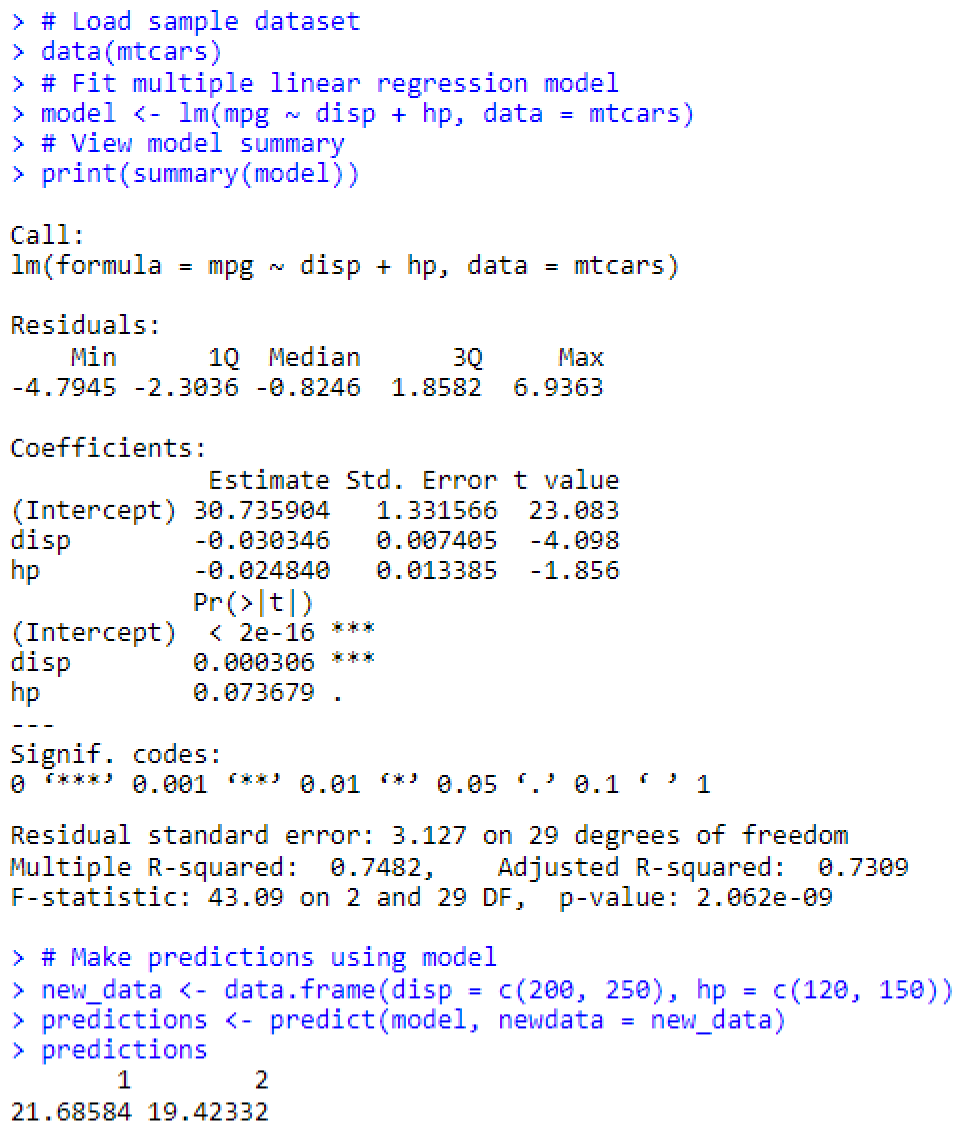
ALGORITHM:

Step 1 : Install the packages "dplyr"..
Step 2 : Load the Dataset.
Step 3 : Fit linear regression model using lm() function.
Step 4 : Analyze the dataset using summary() function.
Step 5 : Predict the test data using predict() function.

PROGRAM:

```
install.packages("dplyr")
# Load libraries
library(dplyr)
# Load sample dataset
data(mtcars)
# Fit multiple linear regression model
model <- lm(mpg ~ disp + hp, data = mtcars)
# View model summary
print(summary(model))
# Make predictions using model
new_data <- data.frame(disp = c(200, 250), hp = c(120, 150))
predictions <- predict(model, newdata = new_data)
predictions
```

OUTPUT:



```
> # Load sample dataset
> data(mtcars)
> # Fit multiple linear regression model
> model <- lm(mpg ~ disp + hp, data = mtcars)
> # View model summary
> print(summary(model))

Call:
lm(formula = mpg ~ disp + hp, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max 
-4.7945 -2.3036 -0.8246  1.8582  6.9363 

Coefficients:
             Estimate Std. Error t value
(Intercept) 30.735904   1.331566  23.083
disp        -0.030346   0.007405  -4.098
hp          -0.024840   0.013385  -1.856
            Pr(>|t|)    
(Intercept)  < 2e-16 ***
disp        0.000306 ***
hp          0.073679 .  
---
Signif. codes:  
0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.127 on 29 degrees of freedom
Multiple R-squared:  0.7482,	Adjusted R-squared:  0.7309 
F-statistic: 43.09 on 2 and 29 DF,  p-value: 2.062e-09

> # Make predictions using model
> new_data <- data.frame(disp = c(200, 250), hp = c(120, 150))
> predictions <- predict(model, newdata = new_data)
> predictions
       1        2 
21.68584 19.42332 
```

RESULT:

Thus, the implementation of multiple regression in R language has been executed and verified successfully.

EXP NO. : 04	TEST OF SIGNIFICANCE
DATE : 27.01.2023	

AIM:

To write an R program to implement a test of significance.

ALGORITHM:

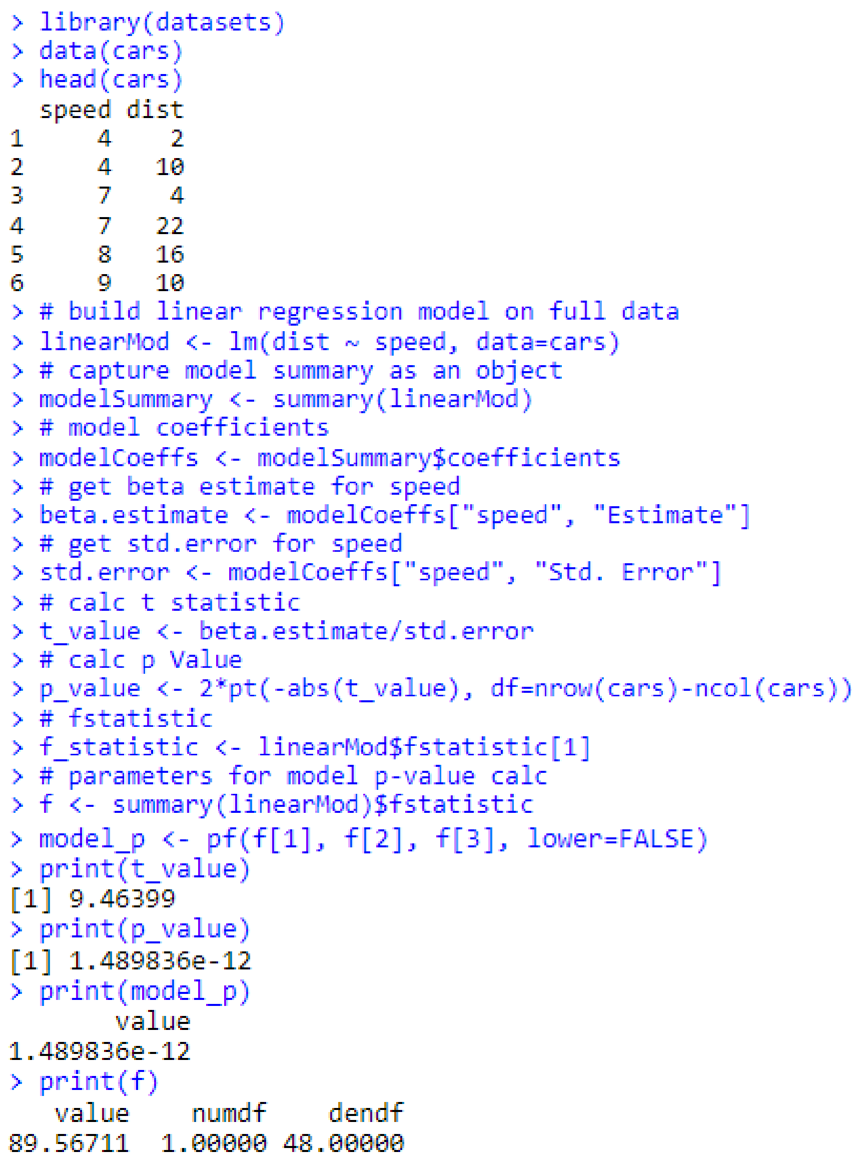
Step 1 : Import the cars dataset from the library datasets.
Step 2 : Create a linear model between the attributes distance and speed.
Step 3 : Capture the summary of the dataset using the summary() function.
Step 4 : Extract the model coefficients from the summary of the model. Select the beta estimate like speed and estimate attributes from the model coefficients.
Step 5 : To extract standard error get speed and std error from the model coefficients.
Step 6 : Calculate the t value, p value, model p and f statistic value and print the result.

PROGRAM:

```
library(datasets)
data(cars)
head(cars)
# build linear regression model on full data
linearMod <- lm(dist ~ speed, data=cars)
# capture model summary as an object
modelSummary <- summary(linearMod)
# model coefficients
modelCoeffs <- modelSummary$coefficients
# get beta estimate for speed
beta.estimate <- modelCoeffs["speed", "Estimate"]
# get std.error for speed
std.error <- modelCoeffs["speed", "Std. Error"]
# calc t statistic
t_value <- beta.estimate/std.error
# calc p Value
p_value <- 2*pt(-abs(t_value), df=nrow(cars)-ncol(cars))
# fstatistic
f_statistic <- linearMod$fstatistic[1]
# parameters for model p-value calc
f <- summary(linearMod)$fstatistic
model_p <- pf(f[1], f[2], f[3], lower=FALSE)
```

```
print(t_value)
print(p_value)
print(model_p)
print(f)
```

OUTPUT:



```
> library(datasets)
> data(cars)
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
> # build linear regression model on full data
> linearMod <- lm(dist ~ speed, data=cars)
> # capture model summary as an object
> modelSummary <- summary(linearMod)
> # model coefficients
> modelCoeffs <- modelSummary$coefficients
> # get beta estimate for speed
> beta.estimate <- modelCoeffs["speed", "Estimate"]
> # get std.error for speed
> std.error <- modelCoeffs["speed", "Std. Error"]
> # calc t statistic
> t_value <- beta.estimate/std.error
> # calc p Value
> p_value <- 2*pt(-abs(t_value), df=nrow(cars)-ncol(cars))
> # fstatistic
> f_statistic <- linearMod$fstatistic[1]
> # parameters for model p-value calc
> f <- summary(linearMod)$fstatistic
> model_p <- pf(f[1], f[2], f[3], lower=FALSE)
> print(t_value)
[1] 9.46399
> print(p_value)
[1] 1.489836e-12
> print(model_p)
       value
1.489836e-12
> print(f)
   value    numdf    dendf
89.56711  1.00000 48.00000
```

RESULT:

Thus, the implementation of test of significance in R language have been executed and verified successfully.

EXP NO. : 05	RESIDUAL ANALYSIS
DATE : 03.02.2023	

AIM:

To write an R program to implement residual analysis.

ALGORITHM:

Step 1 : Install the necessary packages namely digest and ggplot2.
Step 2 : Import the mtcars dataset from the library datasets.
Step 3 : Create a linear model between the attributes mpg and wt.
Step 4 : Save the predicted and residual values.
Step 5 : Fit the Regression Line and its residuals using ggplot2 library.
Step 6 : Similarly perform other plots such as QQ-plot, density plot, scale-location plot, residuals vs leverage plot and residuals vs fitted plot.

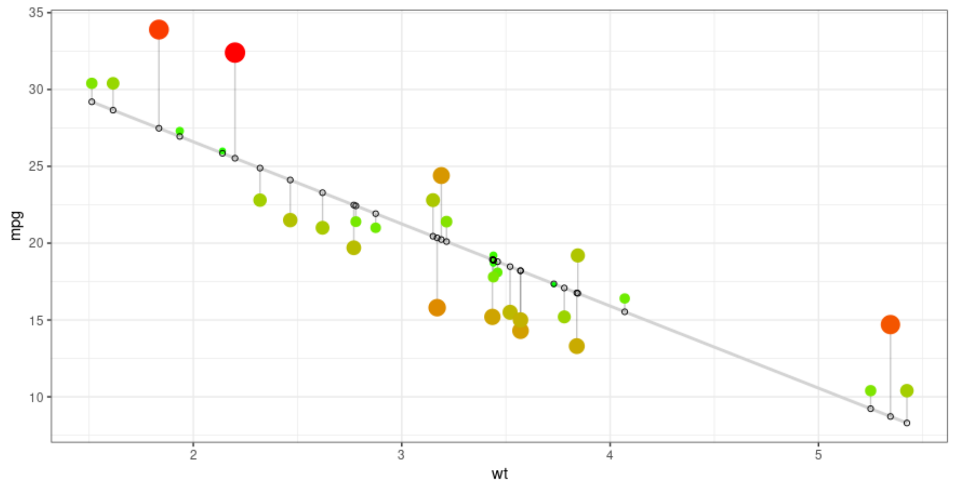
PROGRAM:

```
install.packages("digest")
install.packages("ggplot2")
library(ggplot2)
data("mtcars")
head(mtcars)
d <- mtcars
fit <- lm(mpg ~ wt, data = d) # fit the model
#get list of residuals
res <- resid(fit)
# Save the predicted values
d$predicted <- predict(fit)
# Save the residual values
d$residuals <- residuals(fit)
ggplot(d, aes(x = wt, y = mpg)) + geom_smooth(method = "lm", se = FALSE, color =
"lightgrey") +geom_segment(aes(xend = wt, yend = predicted), alpha = .2) +
geom_point(aes(color = abs(residuals), size = abs(residuals))) +scale_color_continuous(low =
"green", high = "red") +  guides(color = "none", size = "none") +  geom_point(aes(y =
predicted), shape = 1) + theme_bw()
summary(fit)
# Residuals vs Fitted Plot
plot(fit, which=1, col=c("blue"))
# Q-Q Plot
```

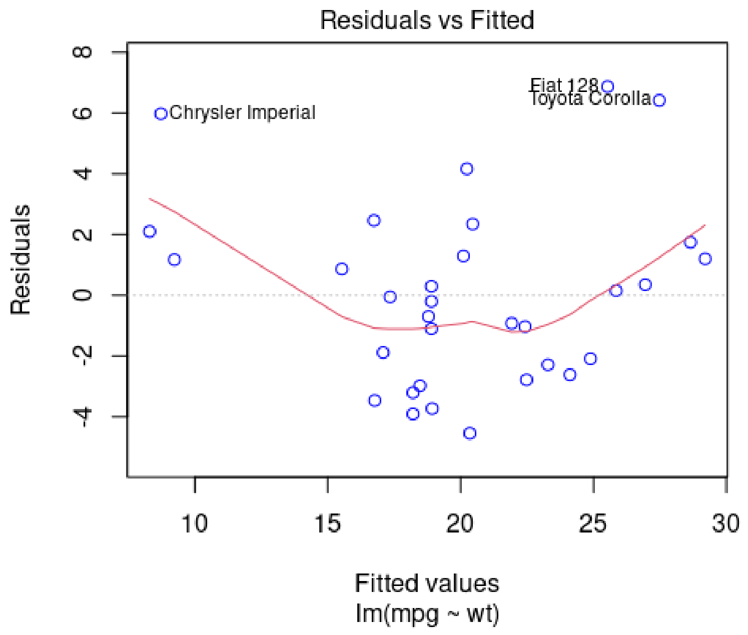
```
plot(fit, which=2, col=c("red"))
# Scale-Location Plot
plot(fit, which=3, col=c("blue"))
# Residuals vs Leverage
plot(fit, which=5, col=c("blue"))
#Create density plot of residuals
plot(density(res))
```

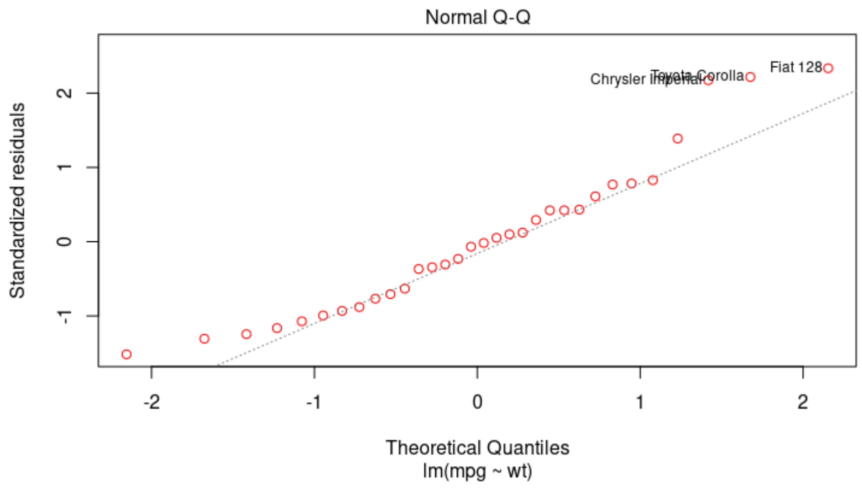
OUTPUT:

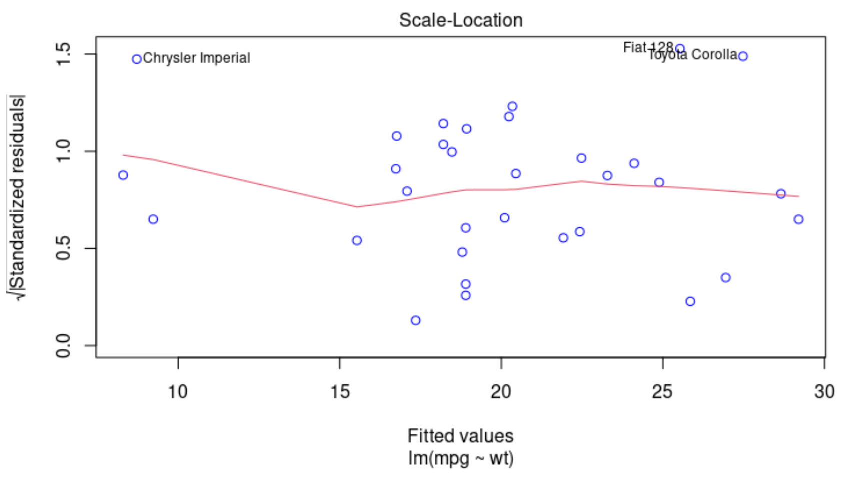
```
> library(ggplot2)
> data("mtcars")
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0
                  gear carb
Mazda RX4            4    4
Mazda RX4 Wag        4    4
Datsun 710           4    1
Hornet 4 Drive       3    1
Hornet Sportabout    3    2
Valiant              3    1
> d <- mtcars
> fit <- lm(mpg ~ wt, data = d) # fit the model
> #get list of residuals
> res <- resid(fit)
> # Save the predicted values
> d$predicted <- predict(fit)
> # Save the residual values
> d$residuals <- residuals(fit)
```

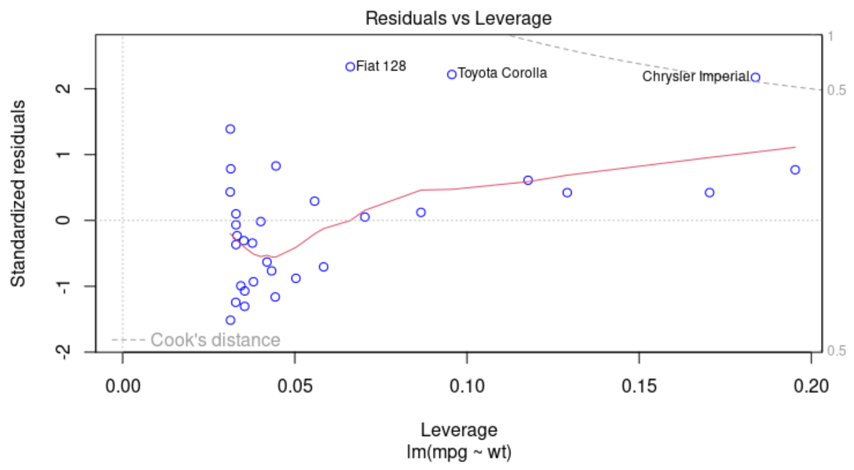


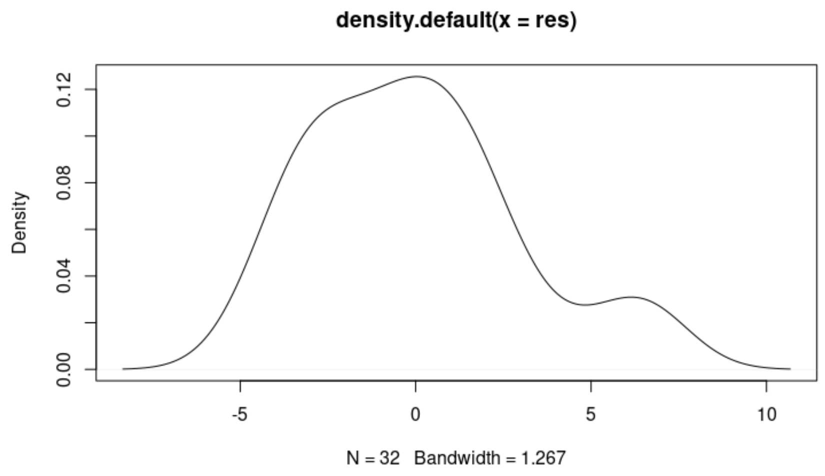
```
> summary(fit)

Call:
lm(formula = mpg ~ wt, data = d)

Residuals:
    Min      1Q  Median      3Q     Max 
-4.5432 -2.3647 -0.1252  1.4096  6.8727 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
wt           -5.3445     0.5591  -9.559 1.29e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.046 on 30 degrees of freedom
Multiple R-squared:  0.7528,	Adjusted R-squared:  0.7446 
F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```











RESULT:

Thus, the implementation of residual analysis in R language has been executed and verified successfully.

EXP NO. : 06	POLYNOMIAL REGRESSION IN R
DATE : 10.02.2023	

AIM:

To write an R program to implement polynomial regression.

ALGORITHM:

Step 1 : Install the packages and import them using library() function.
Step 2 : Load the dataset.
Step 3 : Visualize the data using ggplot() function.
Step 4 : Preprocess the data(splitting the data).
Step 5 : Apply the polynomial regression model to the dataset.
Step 6 : Plot and evaluate the model.

PROGRAM:

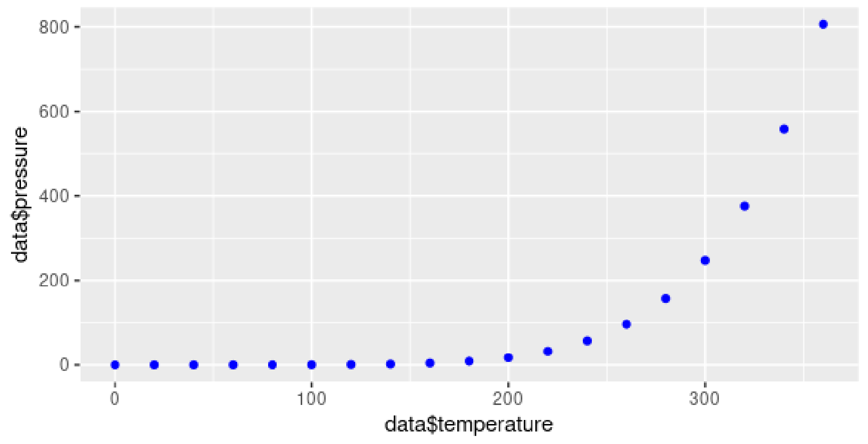
```
install.packages('caTools')
install.packages('Metrics')
install.packages('ggplot2')
library(caTools)
library(Metrics)
# Load the dataset
data = read.csv("polynomial_dataset.csv")
head(data)
# plotting the graph
library(ggplot2)
ggplot() +geom_point(aes(x = data$temperature, y = data$pressure),colour = 'blue')
split = sample.split(data$pressure, SplitRatio = 2/3)
training = subset(data, split == TRUE)
testing = subset(data, split == FALSE)
data$temperature2= data$temperature ^  2
data$temperature3= data$temperature ^  3
data$temperature4 = data$temperature  ^  4
polynomial_reg = lm(formula = pressure~ .,data = data)
summary(polynomial_reg)
x_grid = seq(min(data$temperature), max(data$temperature), 0.1)
```

ggplot() +geom_point(aes(x = data$temperature, y = data$pressure),colour = 'red')
+ geom_line(aes(x = x_grid, y = predict(polynomial_reg, newdata = data.frame(temperature = x_grid, temperature2 = x_grid^2, temperature3 = x_grid^3,temperature4 = x_grid^4))),
    colour = 'blue') +ggtitle('Real or Predicted (Polynomial Regression)') +xlab('temperature') + ylab('pressure')

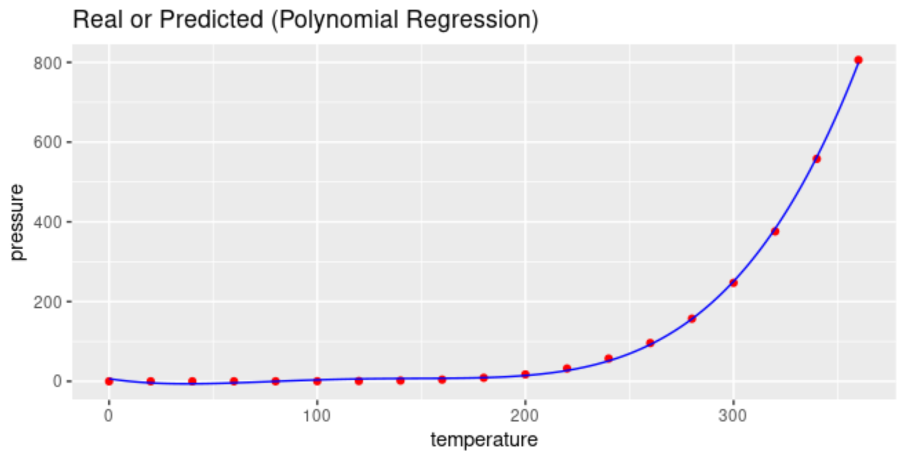
# Making prediction on the test data
poly_pred <- predict(object = polynomial_reg)
RMSE<- rmse(poly_pred, testing$pressure)
RMSE
MAE<-mae(poly_pred, testing$pressure)
MAE

OUTPUT:

```
> library(caTools)
> library(Metrics)
> # Load the dataset
> data = read.csv("polynomial_dataset.csv")
> head(data)
  temperature pressure
1           0   0.0002
2          20   0.0012
3          40   0.0060
4          60   0.0300
5          80   0.0900
6         100   0.2700
> # plotting the graph
> library(ggplot2)
> ggplot() +geom_point(aes(x = data$temperature, y = data$pressure),colour = 'blue')
```



```
> split = sample.split(data$pressure, SplitRatio = 2/3)
> training = subset(data, split == TRUE)
> testing = subset(data, split == FALSE)
> data$temperature2= data$temperature ^  2
> data$temperature3= data$temperature ^  3
> data$temperature4 = data$temperature  ^  4
> polynomial_reg = lm(formula = pressure~ .,data = data)
> summary(polynomial_reg)

Call:
lm(formula = pressure ~ ., data = data)

Residuals:
    Min      1Q  Median      3Q     Max 
-7.1989 -4.2112  0.2224  4.0172  7.0729 

Coefficients:
               Estimate Std. Error t value Pr(>|t|)    
(Intercept)   6.453e+00  4.645e+00   1.389 0.186418    
temperature  -7.992e-01  1.893e-01  -4.223 0.000852 ***
temperature2  1.588e-02  2.226e-03   7.135 5.06e-06 ***
temperature3 -1.052e-04  9.415e-06 -11.179 2.31e-08 ***
temperature4  2.341e-07  1.297e-08  18.056 4.28e-11 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 5.38 on 14 degrees of freedom
Multiple R-squared:  0.9996,	Adjusted R-squared:  0.9994 
F-statistic:  7841 on 4 and 14 DF,  p-value: < 2.2e-16
```



```
> RMSE
[1] 351.3687
> MAE
[1] 210.1428
```

RESULT:

Thus, the implementation of polynomial regression in R language has been executed and verified successfully.

EXP NO. : 07	**QUALITATIVE PREDICTORS IN R**
DATE : 17.02.2023	

AIM:

To write an R program to implement qualitative predictors.

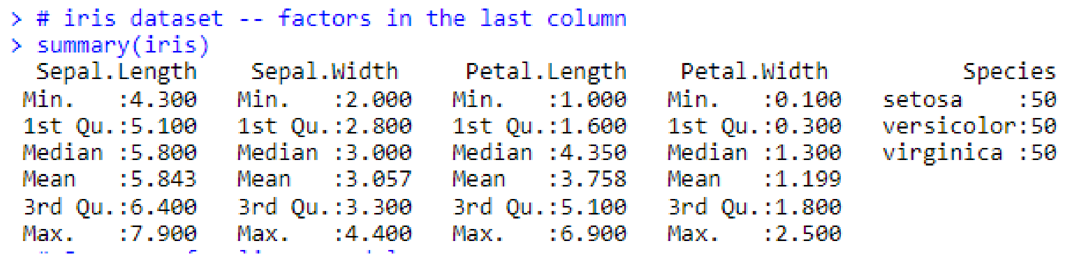
ALGORITHM:

Step 1 : Load the iris dataset.
Step 2 : Set a linear relationship between the variables.
Step 3 : Set different levels as reference.
Step 4 : Find the coefficient of the model using confint() function.
Step 5 : Evaluate the model using contrasts() function.

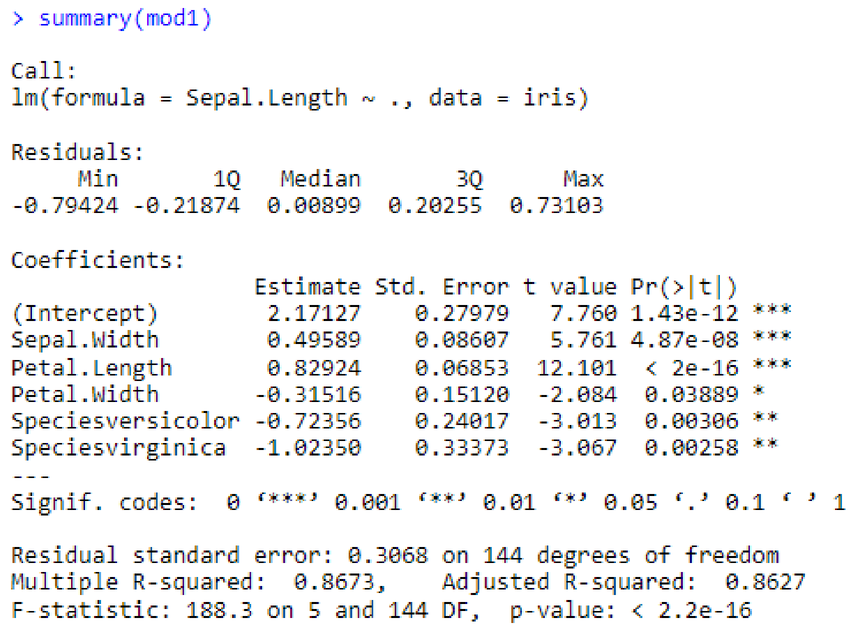
PROGRAM:

```
# iris dataset -- factors in the last column
summary(iris)
# Summary of a linear model
mod1 <- lm(Sepal.Length ~ ., data = iris)
summary(mod1)
# Speciesversicolor (D1) coefficient: -0.72356. The average increment of
# Sepal.Length when the species is versicolor instead of setosa (reference)
# Speciesvirginica (D2) coefficient: -1.02350. The average increment of
# Sepal.Length when the species is virginica instead of setosa (reference)
# Both dummy variables are significant
# How to set a different level as reference (versicolor)
iris$Species <- relevel(iris$Species, ref = "versicolor")
# Same estimates, except for the dummy coefficients
mod2 <- lm(Sepal.Length ~ ., data = iris)
summary(mod2)
# Speciessetosa (D1) coefficient: 0.72356. The average increment of
# Sepal.Length when the species is setosa instead of versicolor (reference)
# Speciesvirginica (D2) coefficient: -0.29994. The average increment of
# Sepal.Length when the species is virginica instead of versicolor (reference)
# Both dummy variables are significant

# Coefficients of the model
confint(mod2)
# The coefficients of Speciessetosa and Speciesvirginica are
# significantly positive and negative, respectively

# Show the dummy variables employed for encoding a factor
contrasts(iris$Species)
iris$Species <- relevel(iris$Species, ref = "setosa")
contrasts(iris$Species)
```

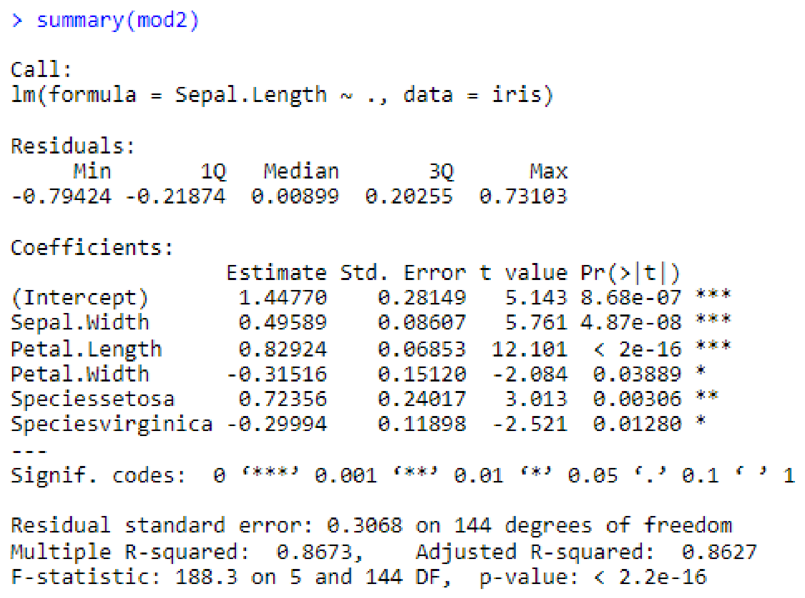
OUTPUT:



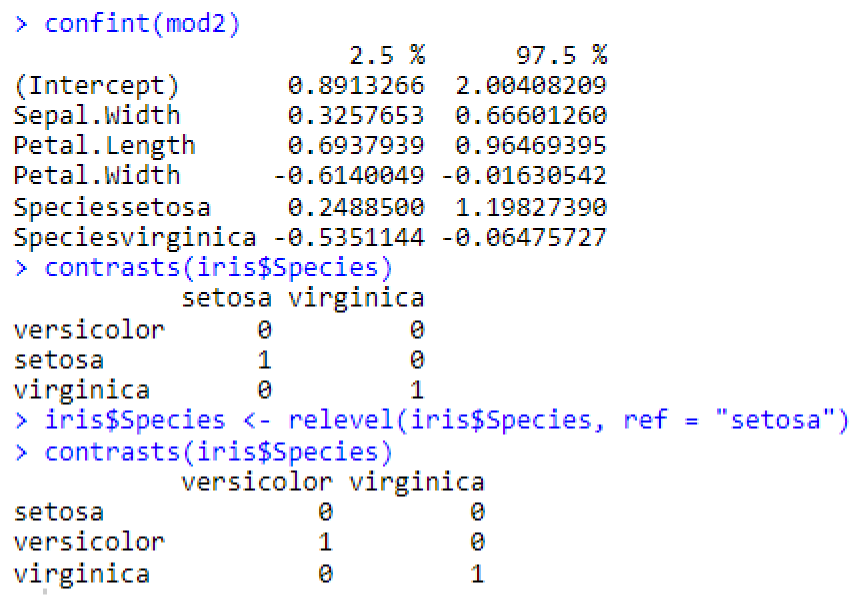
```
> # iris dataset -- factors in the last column
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species  
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50  
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50  
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50  
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199                  
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800                  
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500                  
```



```
> summary(mod1)

Call:
lm(formula = Sepal.Length ~ ., data = iris)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.79424 -0.21874  0.00899  0.20255  0.73103 

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)    
(Intercept)        2.17127    0.27979   7.760 1.43e-12 ***
Sepal.Width        0.49589    0.08607   5.761 4.87e-08 ***
Petal.Length       0.82924    0.06853  12.101  < 2e-16 ***
Petal.Width       -0.31516    0.15120  -2.084  0.03889 *  
Speciesversicolor -0.72356    0.24017  -3.013  0.00306 ** 
Speciesvirginica  -1.02350    0.33373  -3.067  0.00258 ** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.3068 on 144 degrees of freedom
Multiple R-squared:  0.8673,	Adjusted R-squared:  0.8627 
F-statistic: 188.3 on 5 and 144 DF,  p-value: < 2.2e-16
```



```
> summary(mod2)

Call:
lm(formula = Sepal.Length ~ ., data = iris)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.79424 -0.21874  0.00899  0.20255  0.73103 

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)    
(Intercept)       1.44770    0.28149   5.143 8.68e-07 ***
Sepal.Width       0.49589    0.08607   5.761 4.87e-08 ***
Petal.Length      0.82924    0.06853  12.101  < 2e-16 ***
Petal.Width      -0.31516    0.15120  -2.084  0.03889 *  
Speciessetosa     0.72356    0.24017   3.013  0.00306 ** 
Speciesvirginica -0.29994    0.11898  -2.521  0.01280 *  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.3068 on 144 degrees of freedom
Multiple R-squared:  0.8673,	Adjusted R-squared:  0.8627 
F-statistic: 188.3 on 5 and 144 DF,  p-value: < 2.2e-16
```



```
> confint(mod2)
                      2.5 %      97.5 %
(Intercept)       0.8913266  2.00408209
Sepal.Width       0.3257653  0.66601260
Petal.Length      0.6937939  0.96469395
Petal.Width      -0.6140049 -0.01630542
Speciessetosa     0.2488500  1.19827390
Speciesvirginica -0.5351144 -0.06475727
> contrasts(iris$Species)
           setosa virginica
versicolor      0         0
setosa          1         0
virginica       0         1
> iris$Species <- relevel(iris$Species, ref = "setosa")
> contrasts(iris$Species)
           versicolor virginica
setosa              0         0
versicolor          1         0
virginica           0         1
```

RESULT:

Thus, the implementation of qualitative predictors in R language has been executed and verified successfully.

EXP NO. : 08	ESTIMATING A LINEAR RELATIONSHIP
DATE : 24.02.2023	

AIM:

To write an R program to estimate a linear relationship.

a)A statistical model for a linear relationship:

ALGORITHM:

Step 1 : We have a collection of observations but we do not know the values of the coefficients β0,β1,…,βkβ0,β1,…,βk. These need to be estimated from the data.

Step 2 : We choose the values of β0,β1,…,βkβ0,β1,…,βk that minimize T:

∑t=1ε2t=T∑t=1(yt−β0−β1x1,t−β2x2,t−⋯−βkxk,t)2.

Step 3 : The tslm() function fits a linear regression model to time series data. It is similar to the lm() function which is widely used for linear models, but tslm() provides additional facilities for handling time series.

Step 4 : The "t value" is the ratio of an estimated ββ coefficient to its standard error and the last column gives the p-value.

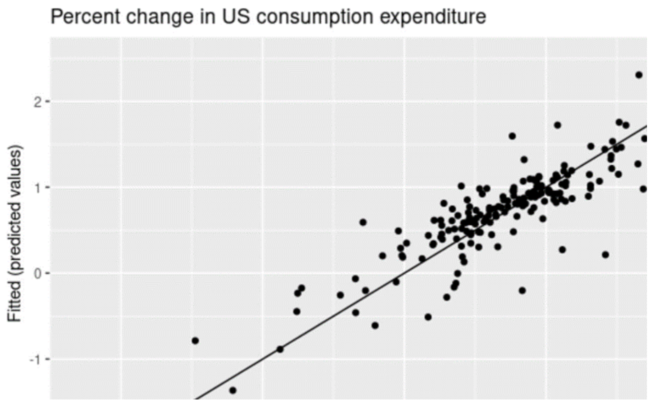
Step 5 : Predictions of yy can be obtained by using the estimated coefficients in the regression equation and setting the error term to zero.

Step 6 : Note that these are predictions of the data used to estimate the model, not genuine forecasts of future values of yy.

PROGRAM:

```
cbind(Data =uschange[,"Consumption"],
Fitted = fitted(fit.consMR)) %>%
as.data.frame()      %>%
ggplot(aes(x=Data, y=Fitted)) +
geom_point() +
ylab("Fitted (predicted values)") +
xlab("Data (actual values)") +
ggtitle("Percent change in US consumption expenditure") +
geom_abline(intercept=0, slope=1)
```

OUTPUT:



b)Least squares estimates:

ALGORITHM:

Step 1 : Load the data into the R environment.

Step 2 : These conditions are verified in R linear fit models with plots, illustrated later.

Step 3 : If a plot of residuals versus fitted values shows a dependence pattern then a linear model is likely invalid.

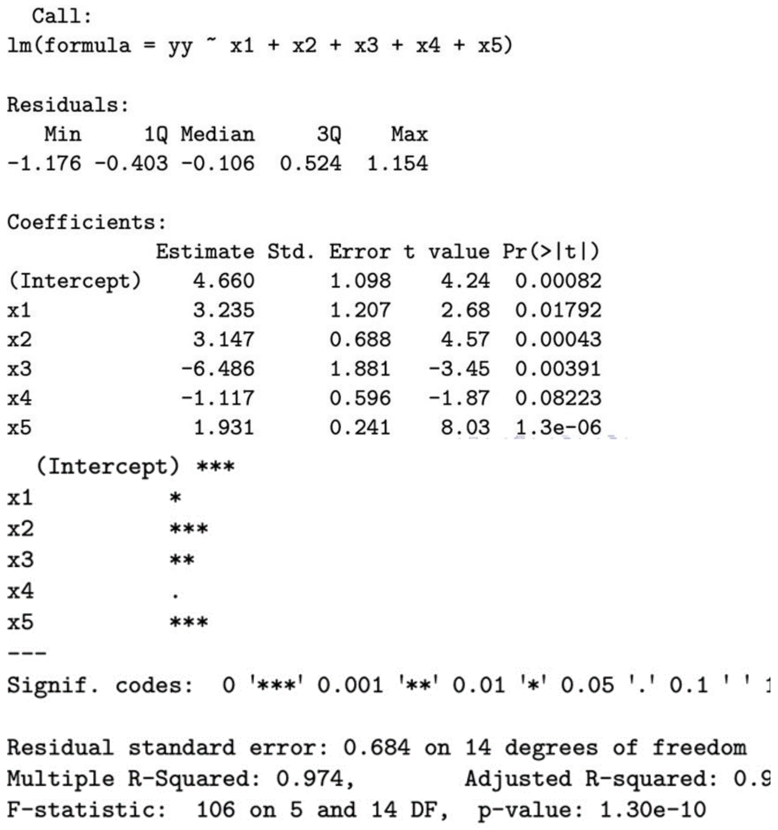
Step 4 : This can be used as a measure of the model's quality and compare linear models with different sets of explanatory variables.

Step 5 : A response variable Y and explanatory variables X1, X2, ...,Xk from continuous random variables.

PROGRAM:

```
lmFit <- lm(Y ~ X1 + ... + Xk)
lmFit1 <- lm(yy ~ x1 + x2 + x3 + x4 + + x5)
summary(lmFit1)
```

OUTPUT:



```
  Call:
lm(formula = yy ~ x1 + x2 + x3 + x4 + x5)

Residuals:
   Min     1Q Median     3Q    Max
-1.176 -0.403 -0.106  0.524  1.154

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    4.660      1.098    4.24  0.00082
x1             3.235      1.207    2.68  0.01792
x2             3.147      0.688    4.57  0.00043
x3            -6.486      1.881   -3.45  0.00391
x4            -1.117      0.596   -1.87  0.08223
x5             1.931      0.241    8.03  1.3e-06
  (Intercept) ***
x1          *
x2          ***
x3          **
x4          .
x5          ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.684 on 14 degrees of freedom
Multiple R-Squared: 0.974,       Adjusted R-squared: 0.9
F-statistic:  106 on 5 and 14 DF,  p-value: 1.30e-10
```

c)The R function lm():

ALGORITHM:

Step 1 : Load the data into the R environment.

Step 2 : Fit linear model.

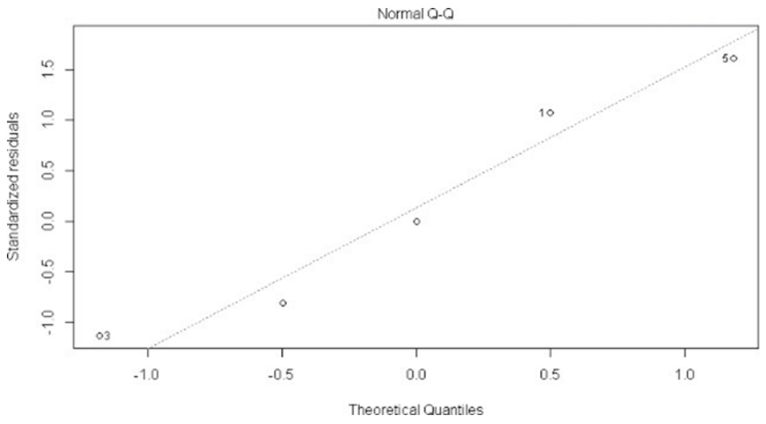
Step 3 : View summary of linear model.

Step 4 : Plot the graph.

PROGRAM:

```
df <- data.frame( x= c(1,2,3,4,5), y= c(1,5,8,15,26))
linear_model <- lm(y ~ x^2, data=df)
summary(linear_model)
plot(linear_model)
```

OUTPUT:



RESULT:

Thus, the estimation of a linear relationship in R language has been executed and verified successfully.

EXP NO. : 09	PACKAGES IN R LANGUAGE
DATE : 03.03.2023	

AIM:

To write an R program to estimate a linear relationship.

R STATS PACKAGE:

R stats is a package that contains many useful functions for statistical calculations and random number generation. In the following table you will see some of the information on this package:

PACKAGE	STATS
Date	October 3,2017
Version	3.5.0
Title	The R stats package

THE CAR PACKAGE:

This package includes many functions for: ANOVA analysis, matrix and vector transformations, printing readable tables of coefficients from several regression models, creating residual plots, tests for the autocorrelation of error terms, and many other general interest statistical and graphing functions.In the following table you will see some of the information on this package:

PACKAGE	CAR
Date	June 25,2017
Version	2.1-5
Title	Companion to applied regression

THE MASS PACKAGE:

This package includes many useful functions and data examples, including functions for estimating linear models through generalized least squares (GLS), fitting negative binomial linear models, the robust fitting of linear models, and Kruskal's non-metric multidimensional scaling.In the following table you will see some of the information on this package:

PACKAGE	MASS
Date	October 2,2017
Version	7.3-4.7
Title	Support functions and datasets for venables and Ripley’s MASS

THE CARET PACKAGE:

This package contains many functions to streamline the model training process for complex regression and classification problems. The package utilizes a number of R packages.In the following table you will see listed some of the information on this package:

PACKAGE	CARET
Date	September 7,2017
Version	6.0-77
Title	Classification and regression training

THE GLMNET PACKAGE:

This package contains many extremely efficient procedures in order to fit the entire Lasso or ElasticNet regularization path for linear regression, logistic and multinomial regression models, Poisson regression, and the Cox model. Multiple response Gaussian and grouped multinomial regression are the two recent additions.In the following table you will see listed some of the information on this package:

PACKAGE	glmnet
Date	September 21,2017
Version	2.0-13
Title	Lasso and Elastic-Net generalized linear models

THE SGD PACKAGE:

This package contains a fast and flexible set of tools for large scale estimation. It features many stochastic gradient methods, built-in models, visualization tools, automated hyperparameter tuning, model checking, interval estimation, and convergence diagnostics. In the following table you will see listed some of the information on this package:

PACKAGE	sgd
Date	January 5,2016
Version	1.1
Title	Stochastic gradient descent for scalable estimation

THE BLR PACKAGE:

This package performs a special case of linear regression named Bayesian linear regression. In Bayesian linear regression, the statistical analysis is undertaken within the context of a Bayesian inference.In the following table you will see listed some of the information on this package:

PACKAGE	BLR
Date	December 3,2014
Version	1.4
Title	Bayesian Linear Regression

THE LARS PACKAGE:

This package contains efficient procedures for fitting an entire Lasso sequence with the cost of a single least squares fit. Least angle regression and infinitesimal forward stagewise regression are related to the Lasso.In the following table you will see listed some of the information on this package:

PACKAGE	LARS
Date	April 23,2013
Version	1.2
Title	Least Angle Regression,Lasso and forward stagewise

RESULT:

Thus, the packages in R language have been studied successfully.

EXP NO. : 10	TIME SERIES ANALYSIS AND FORECASTING USING R
DATE : 23.03.2023	

AIM:

To write an R program to implement time series analysis and forecasting.

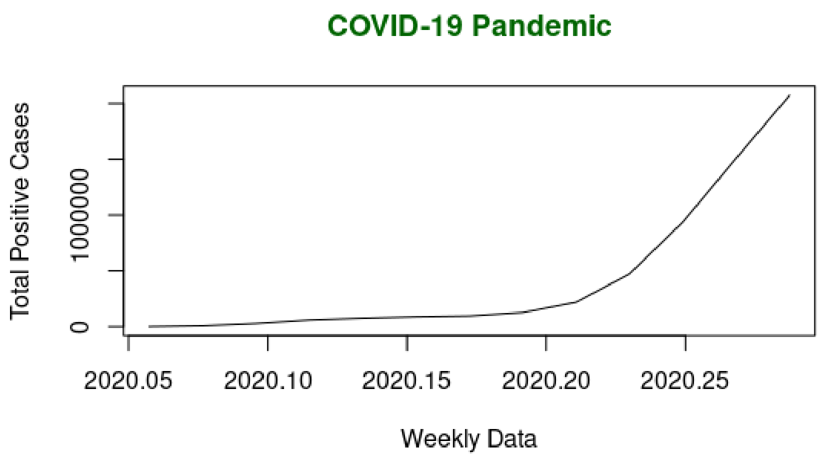
ALGORITHM:

Step 1 : Load the data into the R environment.
Step 2 : Library required for decimal_date() function.
Step 3 : Create a time series object.
Step 4 : Plot the graph.
Step 5 : Forecast model using arima model.
Step 6 : Plot the forecasting model graph.

PROGRAM:

```
#TIME SERIES
install.packages("lubridate")
x <- c(580, 7813, 28266, 59287, 75700,87820, 95314, 126214, 218843, 471497,936851,
1508725, 2072113)
library(lubridate)
mts <- ts(x, start = decimal_date(ymd("2020-01-22")),frequency = 365.25 / 7)
plot(mts, xlab ="Weekly Data",ylab ="Total Positive Cases", main ="COVID-19 Pandemic",
    col.main ="darkgreen")
```

OUTPUT:

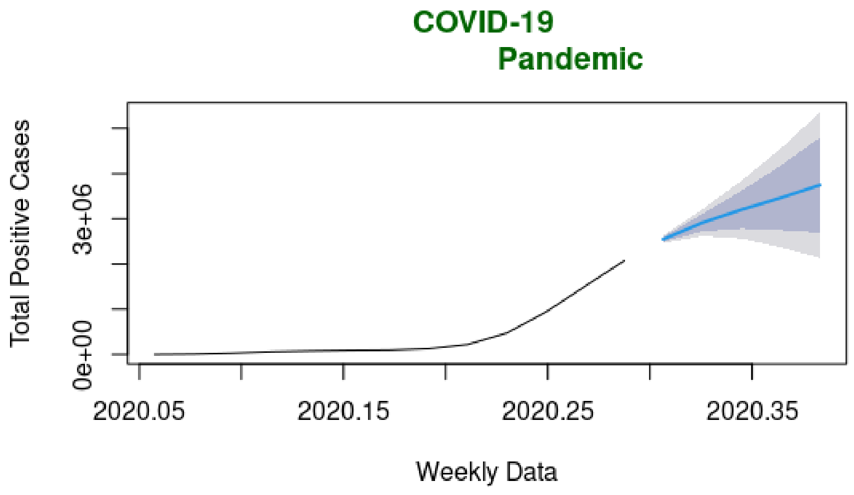


PROGRAM:

#FORCASTING:

```
install.packages("lubridate")
install.packages("forecast")
x <- c(580, 7813, 28266, 59287, 75700,87820, 95314, 126214, 218843,471497, 936851,
    1508725, 2072113)
library(lubridate)
library(forecast)
mts <- ts(x, start = decimal_date(ymd("2020-01-22")),frequency = 365.25 / 7)
fit <- auto.arima(mts)
f <- forecast(fit, 5)
plot(f, xlab ="Weekly Data", ylab ="Total Positive Cases",main ="COVID-19
                  Pandemic", col.main ="darkgreen")
```

OUTPUT:



RESULT:

Thus, time series and forecasting in R language has been executed and verified successfully.