

EXPERIMENT NO:1

AIM: Implement basic commands in R,R Graphics,Indexeing data,loading data,Additional graphical and numerical summaries .

BASIC COMMANDS IN R:

DESCRIPTION: R uses functions to perform operations. To run a function called funcname , we type funcname(input1, input2) , where the inputs (or arguments) input1 and input2 tell R how to run the function. A function can have any number of inputs. For example, to create a vector of numbers, we use the function c() (for concatenate). Any numbers inside the parentheses are joined together.

CODE:

```
x <- c(1,3,2,5)
x
x = c(1,6,2)
x
y = c(1,4,3)
length(x)
length(y)
x+y
ls()
rm(x,y)
ls()
character(0)
rm(list=ls())
?matrix
x=matrix(data=c(1,2,3,4), nrow=2, ncol=2)
x'
```

OUTPUT:

```
R 4.3.0 · ~/ -->
> x <- c(1,3,2,5)
> x
[1] 1 3 2 5
> x = c(1,6,2)
> x
[1] 1 6 2
> y = c(1,4,3)
> length(x)
[1] 3
> length(y)
[1] 3
> x+y
[1] 2 10 5
> ls()
[1] "x" "y"
> rm(x,y)
> ls()
character(0)
> character(0)
character(0)
> rm(list=ls())
> ?matrix
> x=matrix(data=c(1,2,3,4), nrow=2, ncol=2)
> x
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

CODE:

```
x=matrix(c(1,2,3,4) ,2,2)
matrix(c(1,2,3,4) ,2,2,byrow=TRUE)
sqrt(x)
x^2
x=rnorm(50)
y=x+rnorm(50,mean=50,sd=.1)
cor(x,y)
set.seed(1303)
rnorm(50)
set.seed(3)
y=rnorm(100)
mean(y)
var(y)
sqrt(var(y))
sd(y)
```

OUTPUT:

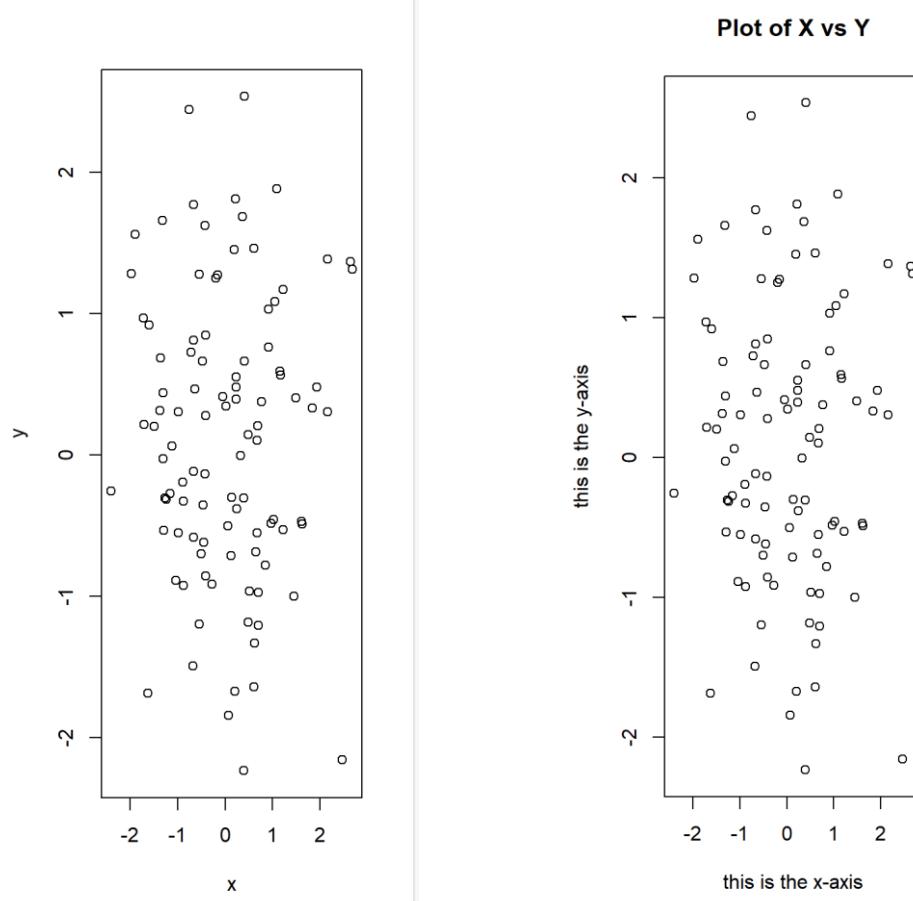
```
> x=matrix(c(1,2,3,4) ,2,2)
> matrix(c(1,2,3,4) ,2,2,byrow=TRUE)
[,1] [,2]
[1,] 1 2
[2,] 3 4
> sqrt(x)
[,1] [,2]
[1,] 1.000000 1.732051
[2,] 1.414214 2.000000
> x^2
[,1] [,2]
[1,] 1 9
[2,] 4 16
> x=rnorm(50)
> y=x+rnorm(50,mean=50,sd=.1)
> cor(x,y)
[1] 0.9934096
> set.seed(1303)
> rnorm(50)
[1] -1.1439763145 1.3421293656 2.1853904757 0.5363925179 0.0631929665 0.5022344825 -0.0004167247 0.5658198405 -0.5725226890
[10] -1.1102250073 -0.0486871234 -0.6956562176 0.8289174803 0.2066528551 -0.2356745091 -0.5563104914 -0.3647543571 0.8623550343
[19] -0.6307715354 0.3136021252 -0.9314953177 0.8238676185 0.5233707021 0.7069214120 0.4202043256 -0.2690521547 -1.5103172999
[28] -0.6902124766 -0.1434719524 -1.0135274099 1.5732737361 0.0127465055 0.8726470499 0.4220661905 -0.0188157917 2.6157489689
[37] -0.6931401748 -0.2663217810 -0.7206364412 1.3677342065 0.2640073322 0.6321868074 -1.3306509858 0.0268888182 1.0406363208
[46] 1.3120237985 -0.0300020767 -0.2500257125 0.0234144857 1.6598706557
> set.seed(3)
> y=rnorm(100)
> mean(y)
[1] 0.01103557
> var(y)
[1] 0.7328675
> sqrt(var(y))
[1] 0.8560768
> sd(y)
[1] 0.8560768
> sd(y)
[1] 0.8560768
```

GRAPHICS IN R:

DESCRIPTION: The plot() function is the primary way to plot data in R . For instance, plot(x,y) produces a scatterplot of the numbers in x versus the numbers in y . There are many additional options that can be passed in to the plot() function.

CODE:

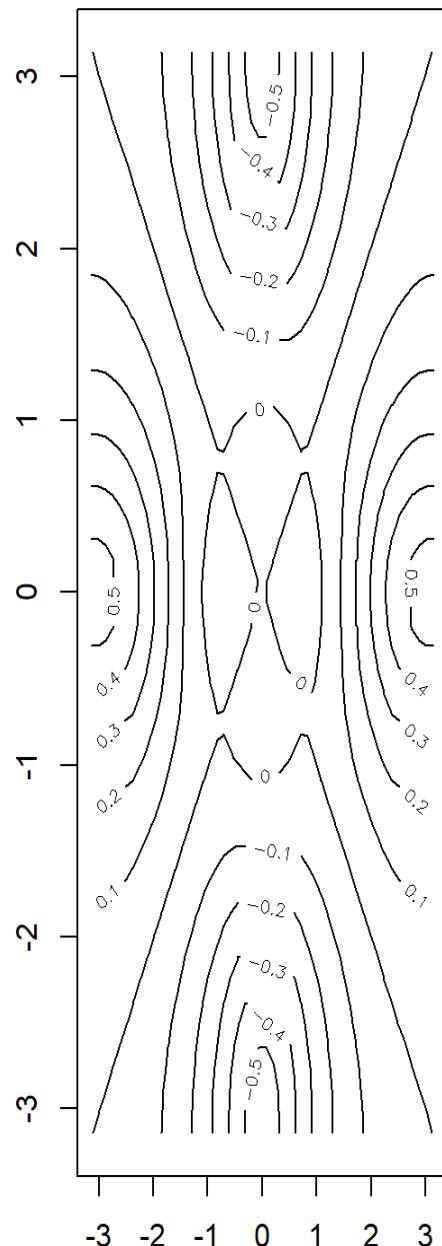
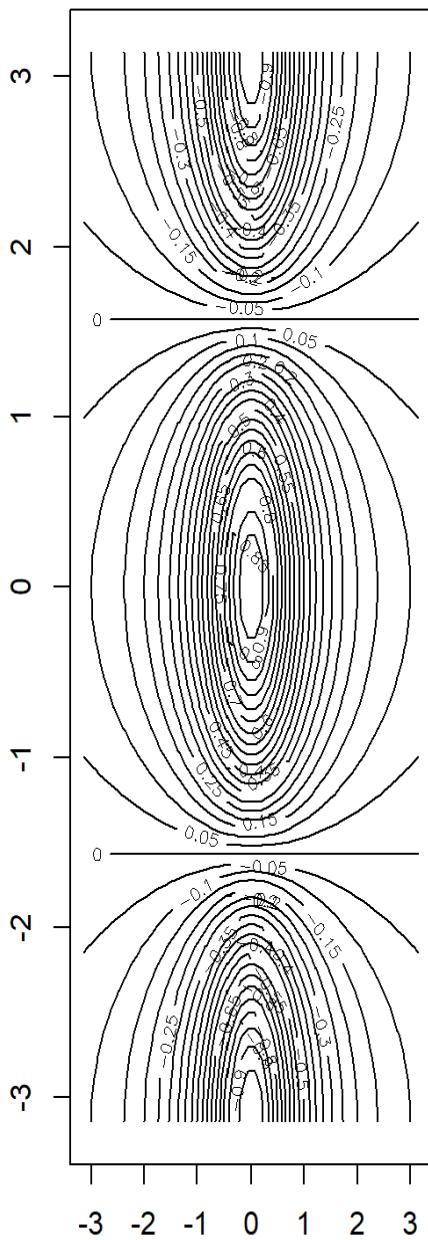
```
x=rnorm(100)
y=rnorm(100)
plot(x,y)
plot(x,y,xlab="this is the x-axis",ylab="this is the y-axis", main="Plot of X vs Y")
pdf("Figure.pdf")
plot(x,y,col="green")
dev.off()
x=seq(1,10)
x
x=1:10
x
```

OUTPUT:

```
> x=seq(1,10)
> x
[1]  1  2  3  4  5  6  7  8  9 10
> x=1:10
> x
[1]  1  2  3  4  5  6  7  8  9 10
```

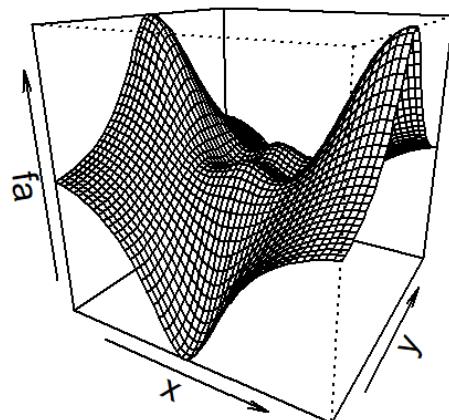
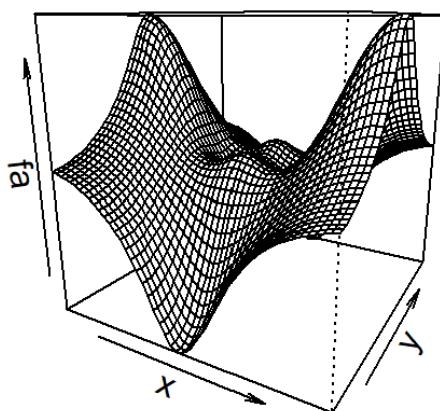
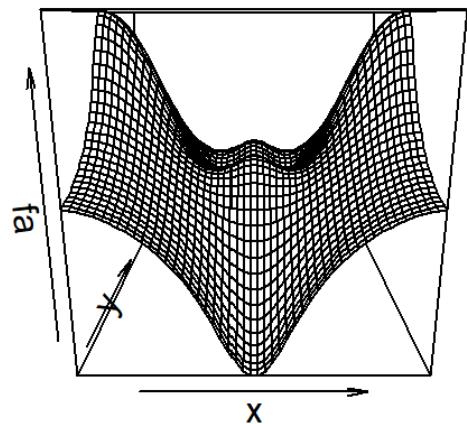
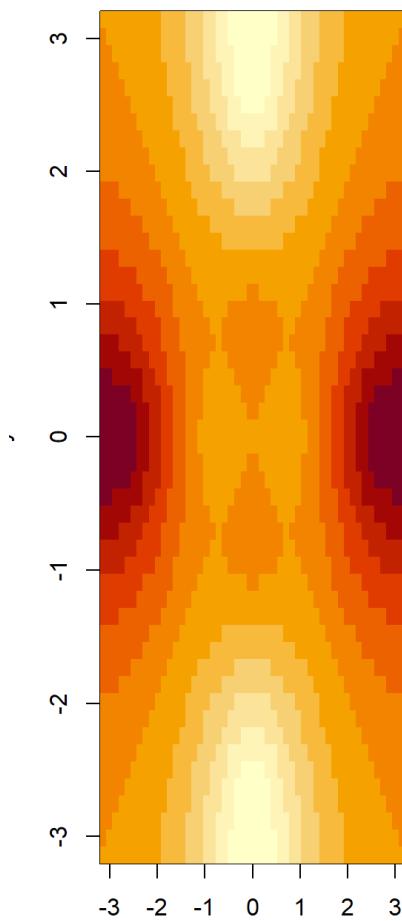
CODE:

```
x=seq(-pi,pi,length =50)
y=x
f=outer(x,y,function(x,y)cos(y)/(1+x^2))
contour(x,y,f)
contour(x,y,f,nlevels=45,add=T)
fa=(f-t(f))/2
contour(x,y,fa,nlevels=15)
```

OUTPUT:

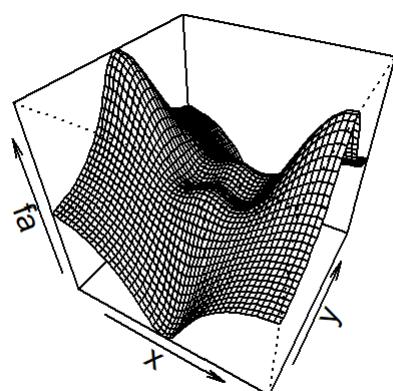
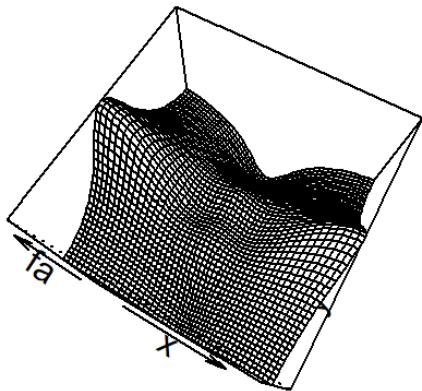
CODE:

```
image(x,y,fa)
persp(x,y,fa)
persp(x,y,fa,theta =30)
persp(x,y,fa,theta=30,phi=20)
```

OUTPUT:

CODE:

```
persp(x,y,fa,theta=30,phi=70)
persp(x,y,fa,theta=30,phi=40)
```

OUTPUT:**INDEXING DATA:**

DESCRIPTION: We often wish to examine part of a set of data. Suppose that our data is stored in the matrix A .

CODE:

```
A=matrix(1:16,4,4)
A
A[2,3]
A[c(1,3),c(2,4)]
A[1:3,2:4]
A[1:2,]
```

OUTPUT:

```
R 4.3.0 · ~/🔗
> A=matrix(1:16,4,4)
> A
[,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
> A[2,3]
[1] 10
> A[c(1,3),c(2,4)]
[,1] [,2]
[1,]    5   13
[2,]    7   15
> A[1:3,2:4]
[,1] [,2] [,3]
[1,]    5    9   13
[2,]    6   10   14
[3,]    7   11   15
> A[1:2,]
[,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
```

CODE:

```
A[,1:2]  
A[1,]  
A[-c(1,3),]  
A[-c(1,3),-c(1,3,4)]  
dim(A)
```

OUTPUT:

```

> A[,1:2]
      [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
> A[,1]
[1] 1 5 9 13
> A[-c(1,3),]
      [,1] [,2] [,3] [,4]
[1,]    2    6   10   14
[2,]    4    8   12   16
> A[-c(1,3),-c(1,3,4)]
[1] 6 8
> dim(A)
[1] 4 4
>

```

LOADING DATA:

DESCRIPTION: For most analyses, the first step involves importing a data set into R . The `read.table()` function is one of the primary ways to do this. The help file `read.table()` contains details about how to use this function. We can use the function `write.table()` to export data.

CODE:

```
auto<- read_excel("C:/Users/Hp/OneDrive/Desktop/ML/auto_mpg.csv.xlsx")
str(auto)
auto[1:4,]
auto=na.omit(auto)
dim(auto)
names(auto)
```

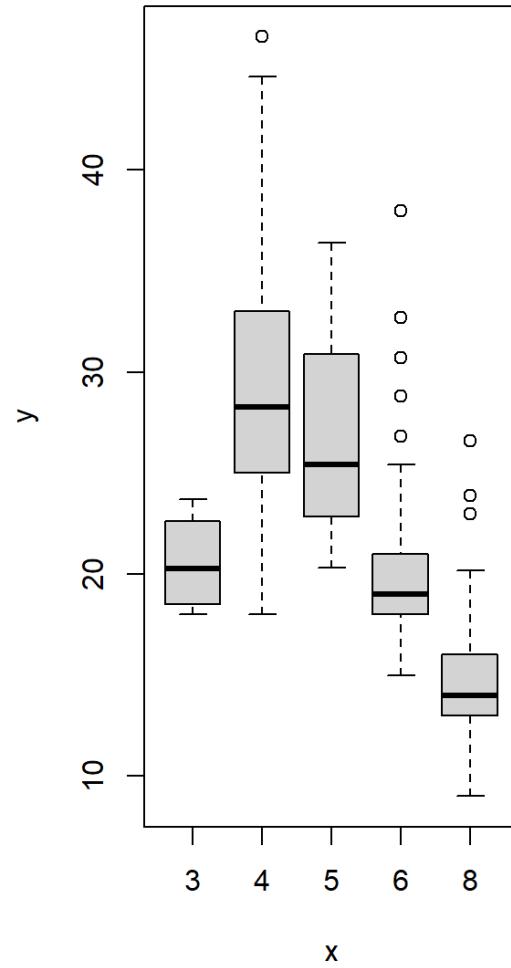
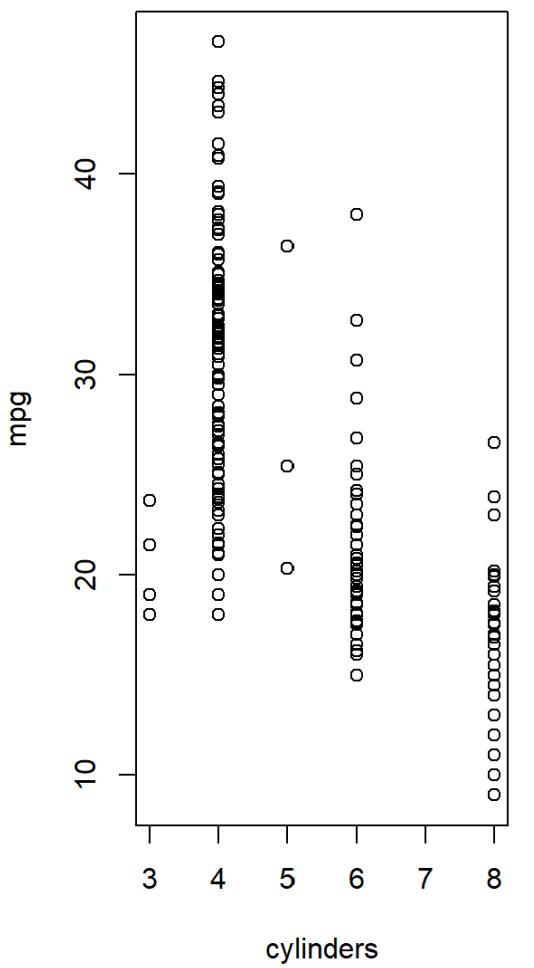
OUTPUT:

ADDITIONAL GRAPHICAL AND NUMERICAL SUMMARIES :

DESCRIPTION: We can use the plot() function to produce scatterplots of the quantitative variables. However, simply typing the variable names will produce an error message, because R does not know to look in the Auto data set for those variables.

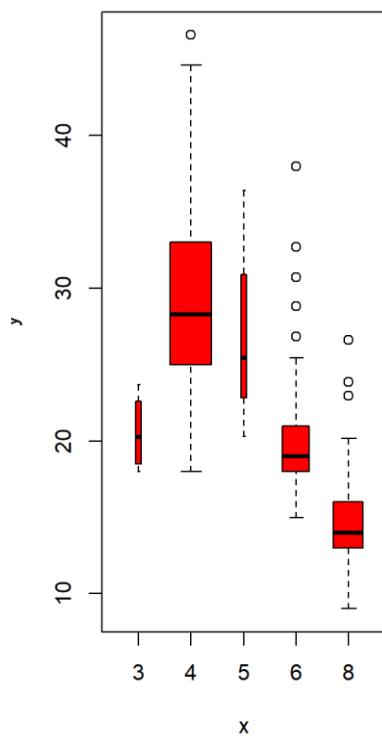
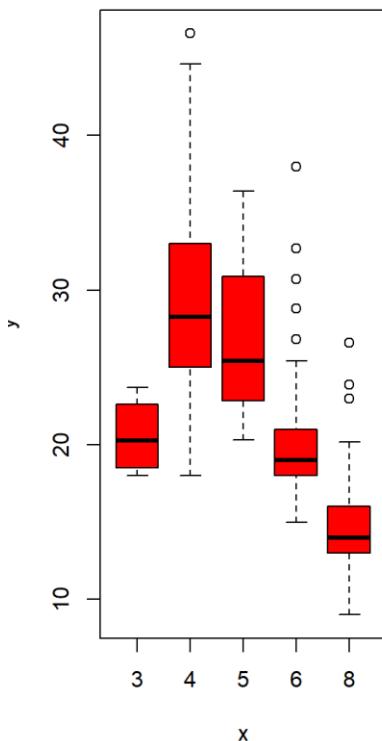
CODE:

```
attach(auto)
plot(cylinders, mpg)
cylinders=as.factor(cylinders)
plot(cylinders,mpg)
```

OUTPUT:

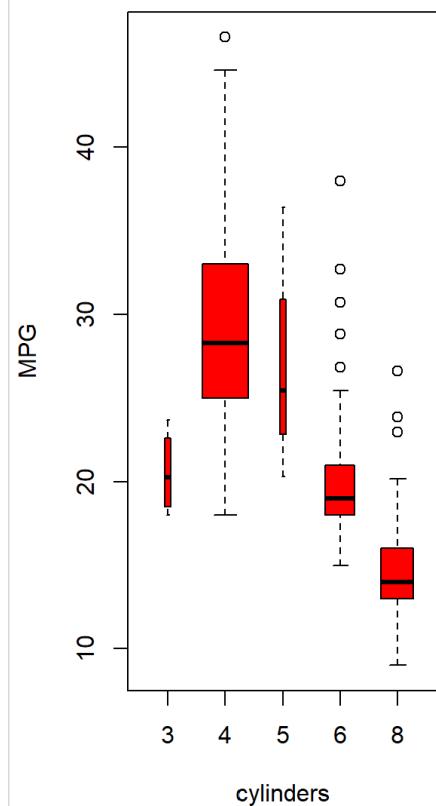
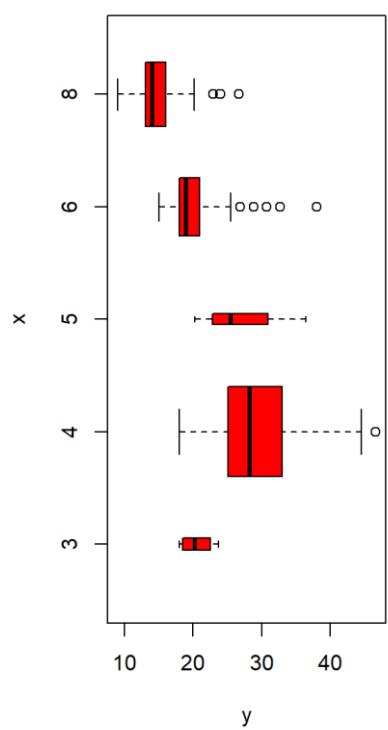
CODE:

```
plot(cylinders,mpg,col="red")
plot(cylinders,mpg,col="red",varwidth=T)
```

OUTPUT:

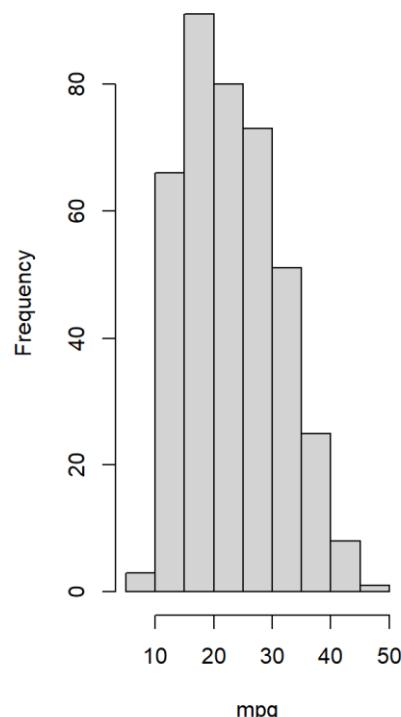
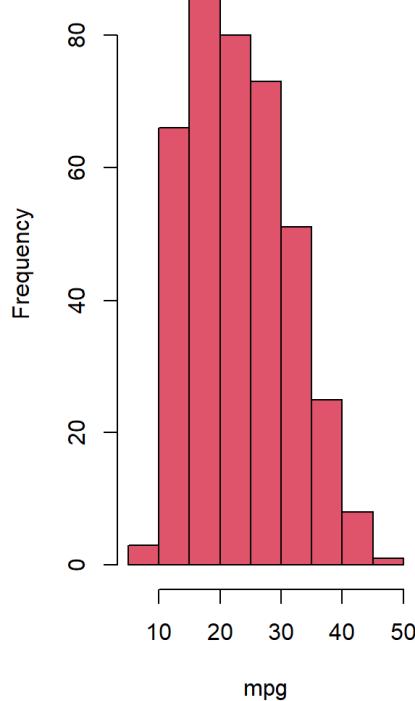
CODE:

```
plot(cylinders,mpg,col="red",varwidth=T, horizontal =T)
plot(cylinders,mpg,col="red",varwidth=T,xlab="cylinders",ylab="MPG")
```

OUTPUT:

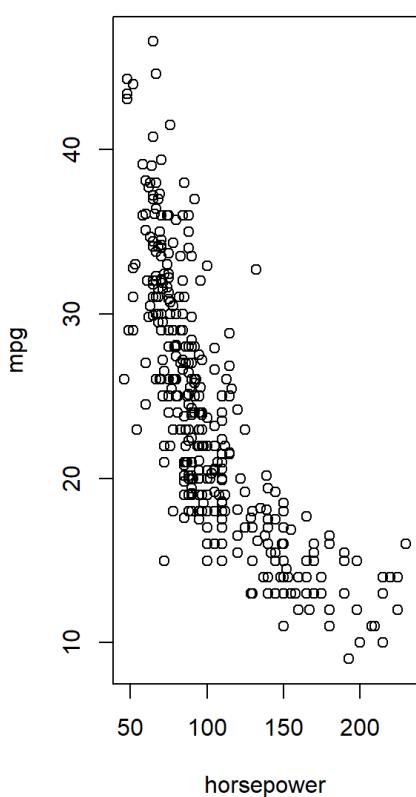
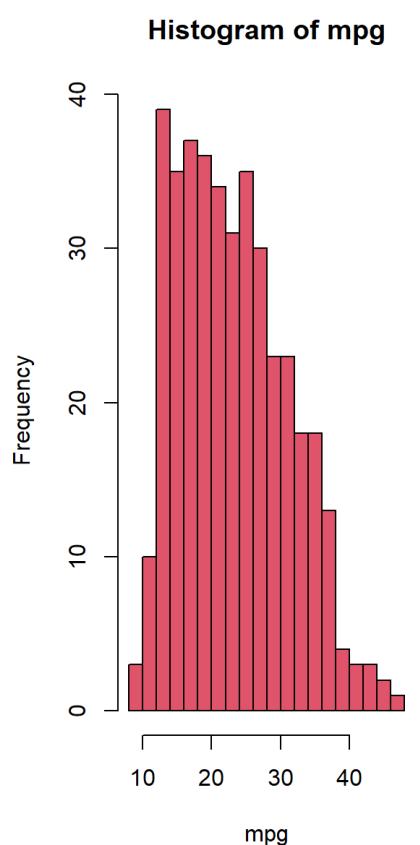
CODE:

```
hist(mpg)  
hist(mpg,col=2)
```

OUTPUT:**Histogram of mpg****Histogram of mpg**

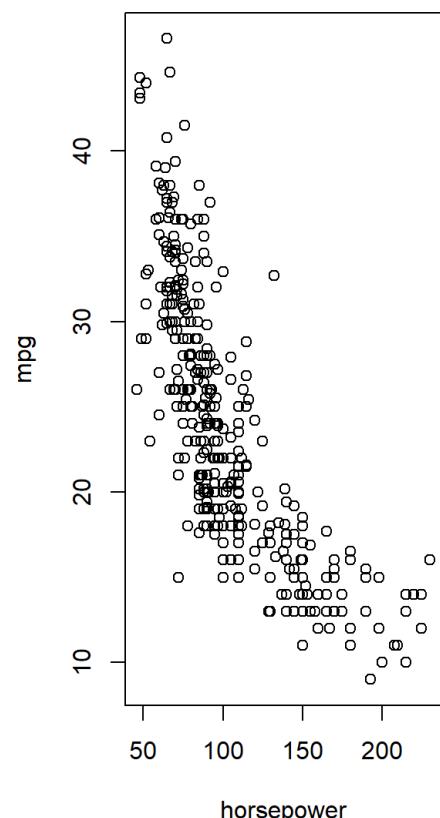
CODE:

```
hist(mpg,col=2,breaks=15)  
plot(horsepower,mpg)
```

OUTPUT:

CODE:

```
identify(horsepower,mpg,car.name)
summary(auto)
summary(auto$mpg)
```

OUTPUT:

```
> summary(auto)
   mpg      cylinders      displacement      horsepower      weight      acceleration      model year      origin 
Min.   : 9.00  Min.   :3.000  Min.   :68.0  Length:398  Min.   :1613  Min.   :8.00  Min.   :70.00  Min.   :1.000 
1st Qu.:17.50  1st Qu.:4.000  1st Qu.:104.2  Class :character  1st Qu.:2224  1st Qu.:13.82  1st Qu.:73.00  1st Qu.:1.000 
Median :23.00  Median :4.000  Median :148.5  Mode  :character  Median :2804  Median :15.50  Median :76.00  Median :1.000 
Mean   :23.51  Mean   :5.455  Mean   :193.4  Mode  :character  Mean   :2970  Mean   :15.57  Mean   :76.01  Mean   :1.573 
3rd Qu.:29.00  3rd Qu.:8.000  3rd Qu.:262.0  Mode  :character  3rd Qu.:3608  3rd Qu.:17.18  3rd Qu.:79.00  3rd Qu.:2.000 
Max.   :46.60  Max.   :8.000  Max.   :455.0  Mode  :character  Max.   :5140  Max.   :24.80  Max.   :82.00  Max.   :3.000 

car name
Length:398
Class :character
Mode  :character
```

```
> summary(auto$mpg)
  Min. 1st Qu. Median  Mean 3rd Qu.  Max. 
  9.00 17.50 23.00 23.51 29.00 46.60
```

EXPERIMENT NO.-2

AIM: Implement the code using of simple linear regression on the Auto data set.

- a. Use the lm() function to perform a simple linear regression with mpg as the response and horsepower as the predictor.
- b. Plot the response and the predictor. Use the abline() function to display the least squares regression line.
- c. Use the plot() function to produce diagnostic plots of the least squares regression fit. Comment on any problems you see with the fit.
- d. Produce a scatterplot matrix which includes all of the variables in the data set.

(2.a) AIM: Use the lm() function to perform a simple linear regression with mpg as the response and horsepower as the predictor.

DESCRIPTION: The library() function is used to load libraries, or groups of functions and library() data sets that are not included in the base R distribution. Basic functions that perform least squares linear regression and other simple analyses come standard with the base distribution, but more exotic functions require additional libraries. Here we load the MASS package, which is a very large collection of data sets and functions. . Alternatively, this can be done at the R command line via install.packages("ISLR").The lm() function to fit a simple linear regression lm() model, with medv as the response and lstat as the predictor. The basic syntax is lm(y~x,data), where y is the response, x is the predictor, and data is the data set in which these two variables are kept

CODE:

```
library(ISLR)
library(MASS)
data("auto")
head(auto)
lm.fit<-lm(mpg~horsepower,data=Auto)
summary(lm.fit)
```

OUTPUT:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year
1	18	8	307	130	3504	12.0	70
2	15	8	350	165	3693	11.5	70
3	18	8	318	150	3436	11.0	70
4	16	8	304	150	3433	12.0	70
5	17	8	302	140	3449	10.5	70
6	15	8	429	198	4341	10.0	70
	origin		name				
1	1	chevrolet	chevelle	malibu			
2	1	buick	skylark	320			
3	1	plymouth	satellite				
4	1	amc	rebel	sst			
5	1	ford	torino				
6	1	ford	galaxie	500			

```
Call:  
lm(formula = mpg ~ horsepower, data = Auto)  
  
Residuals:  
    Min      1Q  Median      3Q     Max  
-13.5710 -3.2592 -0.3435  2.7630 16.9240  
  
Coefficients:  
            Estimate Std. Error t value Pr(>|t|)  
(Intercept) 39.935861   0.717499  55.66 <2e-16 ***  
horsepower  -0.157845   0.006446 -24.49 <2e-16 ***  
---  
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1  
  
Residual standard error: 4.906 on 390 degrees of freedom  
Multiple R-squared:  0.6059, Adjusted R-squared:  0.6049  
F-statistic: 599.7 on 1 and 390 DF, p-value: < 2.2e-16
```

CODE:

```
predict(lm.fit,data.frame(horsepower=c(98)),interval="prediction")  
predict(lm.fit,data.frame(horsepower=c(98)),interval="confidence")
```

OUTPUT:

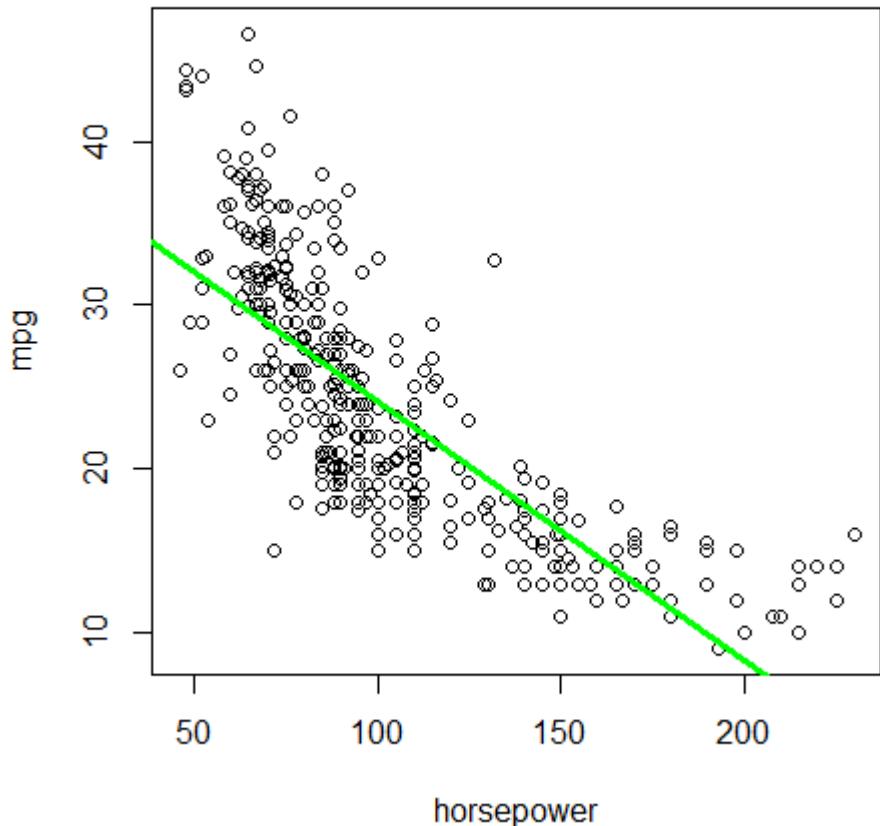
```
> predict(lm.fit,data.frame(horsepower=c(98)),interval="prediction")  
    fit    lwr     upr  
1 24.46708 14.8094 34.12476  
> predict(lm.fit,data.frame(horsepower=c(98)),interval="confidence")  
    fit    lwr     upr  
1 24.46708 23.97308 24.96108
```

(2.b) AIM : Plot the response and the predictor. Use the abline() function to display the least squares regression line.

DESCRIPTION: The abline() function can be used to draw any line, not just the least squares regression line. To draw a line with intercept a and slope b , we type abline(a, b). Below we experiment with some additional settings for plotting lines and points. The lwd=3 command causes the width of the regression line to be increased by a factor of 3; this works for the plot() and lines() functions also.

CODE:

```
attach(Auto)
plot(horsepower,mpg)
abline(lm.fit,lwd=5,col="blue")
```

OUTPUT:

(2.c) AIM: Use the plot() function to produce diagnostic plots of the least squares regression fit. Comment on any problems you see with the fit.

DESCRIPTION: Plot() function in R programming Language is defined as a generic function for plotting. The plot() function in R is “used to create a plot”. It has many options and arguments to control many things, such as the plot type, labels, titles, and colors. For example, plot(x, y) creates a scatter plot of x and y numeric vectors.

CODE:

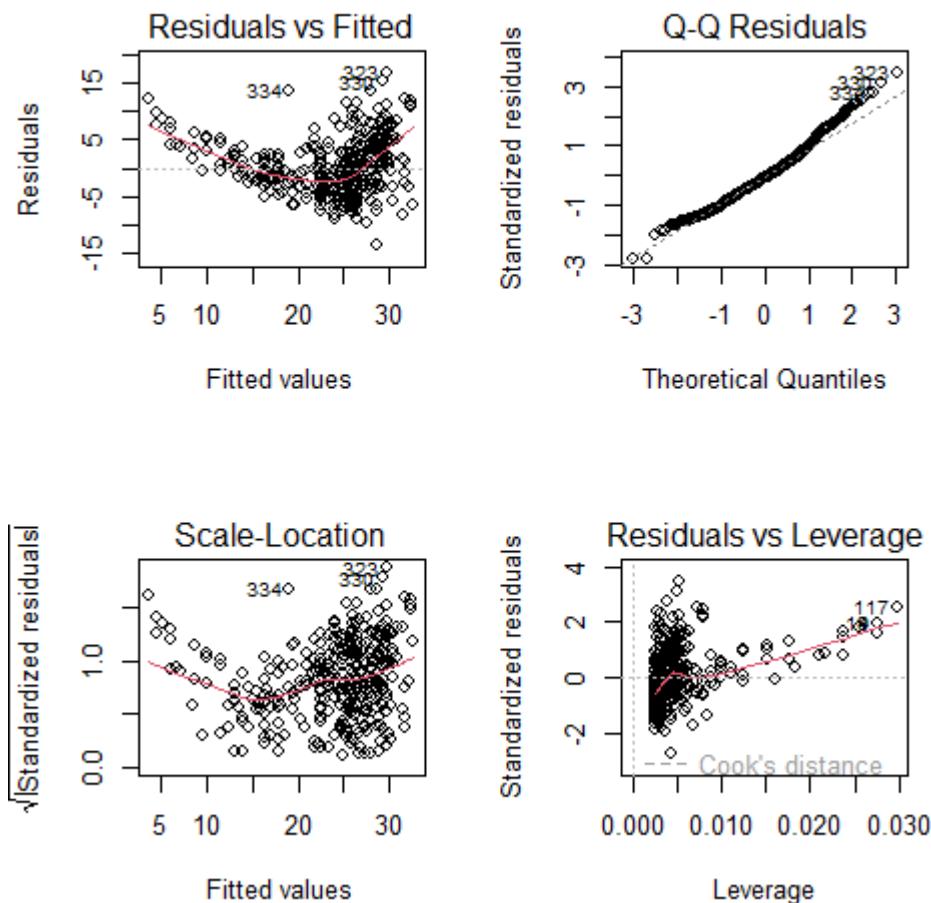
```
which.max(hatvalues(lm.fit))
par(mfrow=c(2,2))
plot(lm.fit)
```

OUTPUT:

```
> which.max(hatvalues(lm.fit))
```

117

116



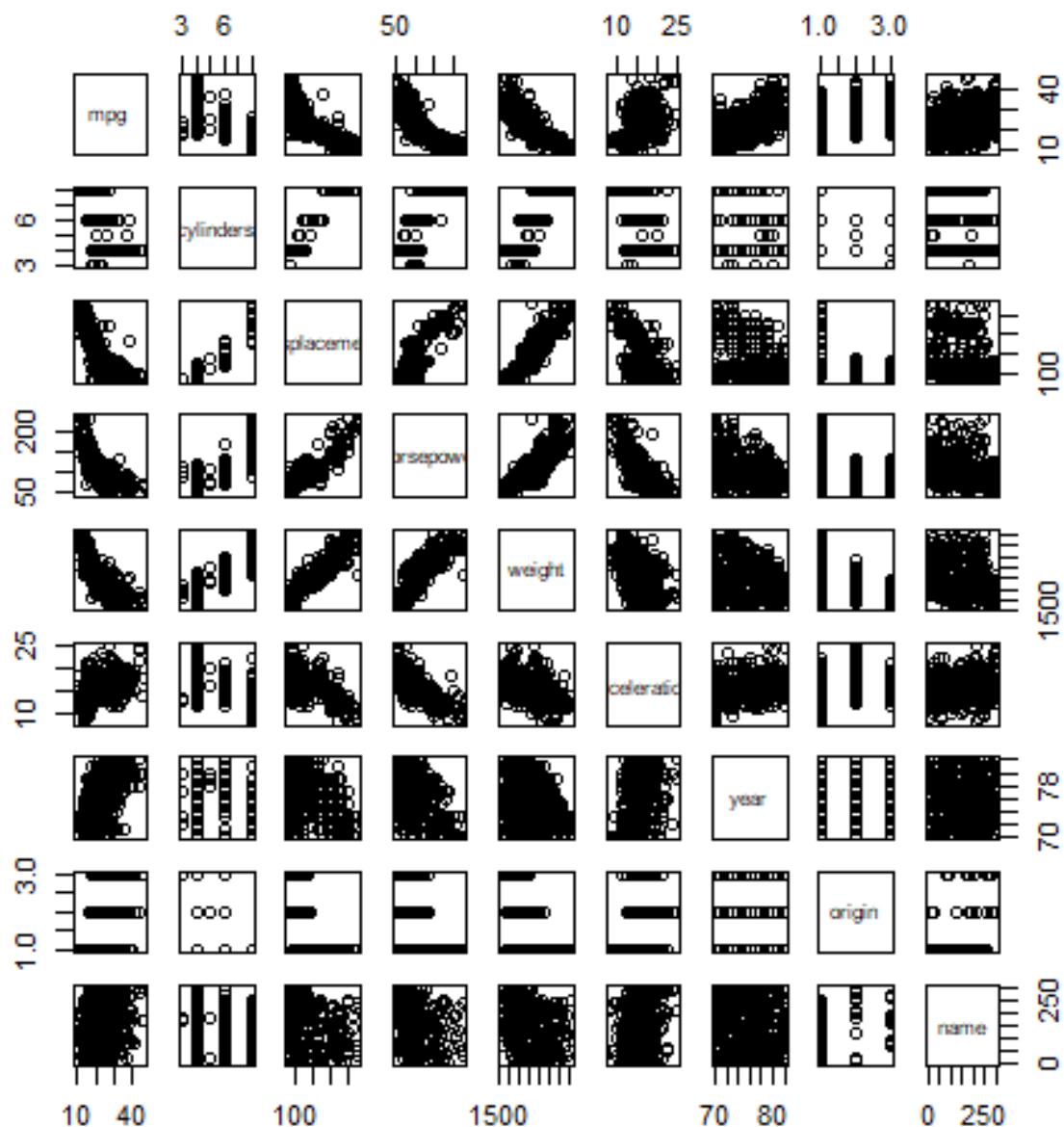
(2.d) AIM: . Produce a scatterplot matrix which includes all of the variables in the data set.

DESCRIPTION: A scatterplot is a set of dotted points representing individual data pieces on the horizontal and vertical axis.In a graph in which the values of two variables are plotted along the axis X and Y,the pattern of the resulting points reveals a correlation between them.We can create a scatter plot in R programming Language using the plot() function.

CODE:

```
Pairs(Auto)
```

OUTPUT:



EXPERIMENT NO:- 3

AIM:- Using multiple linear regression perform the following tasks on the Auto data set.

(a) Produce a scatter plot matrix which includes all of the variable sin the data set.

(b) Compute the matrix of correlations between the variables using the function `cor()` . You will need to exclude the name variable, `cor()` which is qualitative.

(c) Use the `lm()` function to perform a multiple linear regression with `mpg` as the response and all other variables except `name` as the predictors. Use the `summary()` function to print the results. Comment on the output, That is

- i. Is there a relationship between the predictors and the response?
- ii. Which predictors appear to have a statistically significant relationship to the response?
- iii. What does the coefficient for the year variable suggest?

(d) Use the `plot()` function to produce diagnostic plots of the linear regression fit. Comment on any problems you see with the fit. Do the residual plots suggest any unusually large outliers? Does the leverage plot identify any observations with unusually high leverage?

(e) Use the * and : symbols to fit linear regression models with interaction effects. Do any interactions appear to be statistically significant?

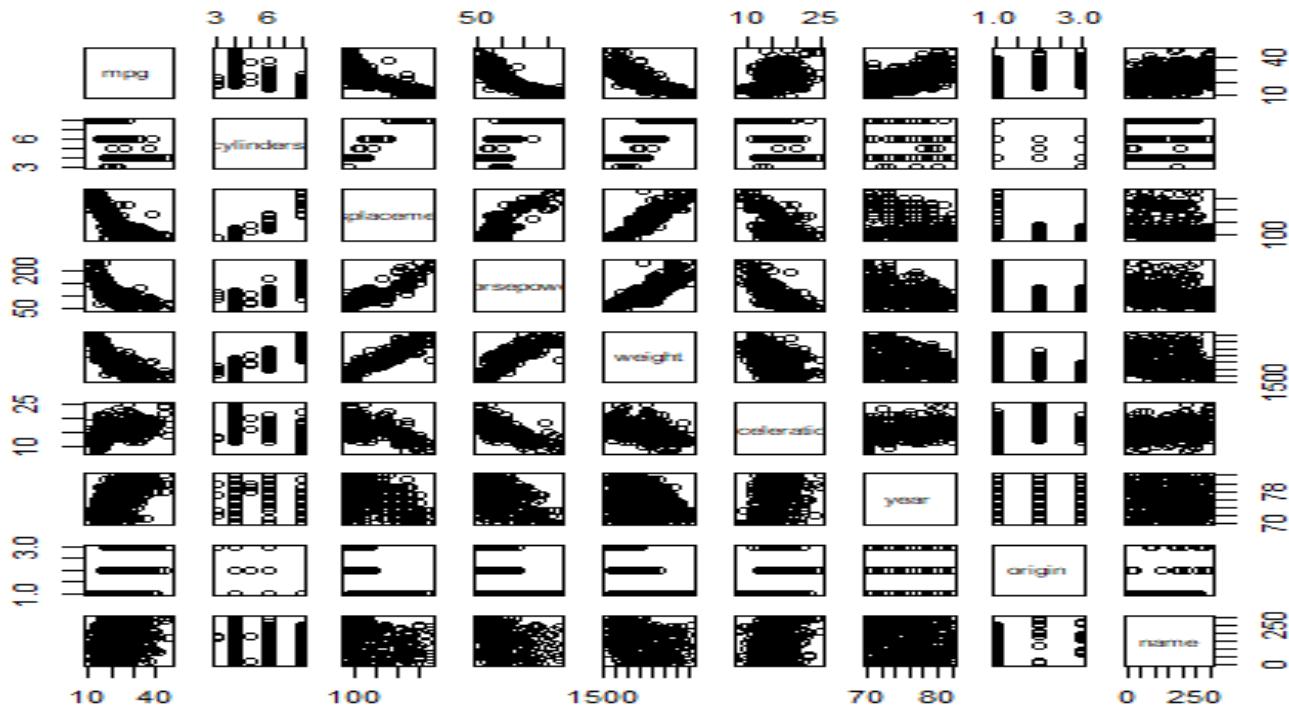
(f) Try a few different transformations of the variables, such as `log(X)`, \sqrt{X} , X^2 . Comment on your findings.

(3a) AIM:- Produce a scatter plot matrix which includes all of the variable sin the data set.

DESCRIPTION: A scatterplot is a set of dotted points representing individual data pieces on the horizontal and vertical axis. In a graph in which the values of two variables are plotted along the axis X and Y, the pattern of the resulting points reveals a correlation between them. We can create a scatter plot in R programming Language using the `plot()` function.

CODE:

```
data("Auto", package = "ISLR")
pairs(Auto)
```

OUTPUT:

(3b) AIM:- Compute the matrix of correlations between the variables using the function cor() . You will need to exclude the name variable, cor() which is qualitative.

DESCRIPTION: cor() function in R Language is used to measure the correlation coefficient value between two vectors. This returns a simple correlation matrix showing the correlations between pairs of variables (devices). You can choose the correlation coefficient to be computed using the method parameter. The default method is Pearson, but you can also compute Spearman or Kendall coefficients.

CODE:

```
cor(subset(Auto, select = -name))
```

OUTPUT:

```
> cor(subset(Auto, select = -name))
      mpg cylinders displacement horsepower weight acceleration year origin
mpg      1.0000000 -0.7776175 -0.8051269 -0.7784268 -0.8322442  0.4233285 0.5805410 0.5652088
cylinders -0.7776175 1.0000000  0.9508233  0.8429834  0.8975273 -0.5046834 -0.3456474 -0.5689316
displacement -0.8051269 0.9508233  1.0000000  0.8972570  0.9329944 -0.5438005 -0.3698552 -0.6145351
horsepower  -0.7784268 0.8429834  0.8972570  1.0000000  0.8645377 -0.6891955 -0.4163615 -0.4551715
weight      -0.8322442 0.8975273  0.9329944  0.8645377  1.0000000 -0.4168392 -0.3091199 -0.5850054
acceleration 0.4233285 -0.5046834 -0.5438005 -0.6891955 -0.4168392  1.0000000 0.2903161 0.2127458
year        0.5805410 -0.3456474 -0.3698552 -0.4163615 -0.3091199  0.2903161 1.0000000 0.1815277
origin      0.5652088 -0.5689316 -0.6145351 -0.4551715 -0.5850054  0.2127458 0.1815277 1.0000000
```

(3c) Use the lm() function to perform a multiple linear regression with mpg as the response and all other variables except name as the predictors. Use the summary() function to print the results. Comment on the output, That is

- i. Is there a relationship between the predictors and the response?
- ii. Which predictors appear to have a statistically significant relationship to the response?
- iii. What does the coefficient for the year variable suggest?

DESCRIPTION: The lm() function to fit a simple linear regression lm() model, with medv as the response and lstat as the predictor. The basic syntax is lm(y~x,data), where y is the response, x is the predictor, and data is the data set in which these two variables are kept.

CODE:

```
lm.fit1 <- lm(mpg ~ . - name, data = Auto)
summary(lm.fit1)
```

OUTPUT:

```
> lm.fit1 <- lm(mpg ~ . - name, data = Auto)
> summary(lm.fit1)

Call:
lm(formula = mpg ~ . - name, data = Auto)

Residuals:
    Min      1Q  Median      3Q     Max 
-9.5903 -2.1565 -0.1169  1.8690 13.0604 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -17.218435   4.644294  -3.707  0.00024 ***
cylinders   -0.493376   0.323282  -1.526  0.12780    
displacement  0.019896   0.007515   2.647  0.00844 **  
horsepower   -0.016951   0.013787  -1.230  0.21963    
weight       -0.006474   0.000652  -9.929  < 2e-16 ***
acceleration  0.080576   0.098845   0.815  0.41548    
year         0.750773   0.050973  14.729  < 2e-16 ***
origin       1.426141   0.278136   5.127  4.67e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.328 on 384 degrees of freedom
Multiple R-squared:  0.8215, Adjusted R-squared:  0.8182 
F-statistic: 252.4 on 7 and 384 DF,  p-value: < 2.2e-16
```

(3.c)**(i). Is there a relationship between the predictors and the response?**

Yes, there is a relationship between the predictors and the response by testing the null hypothesis of whether all the regression coefficients are zero. The F-statistic is far from 1 (with a small p-value), indicating evidence against the null hypothesis.

(3.c)**(ii). Which predictors appear to have a statistically significant relationship to the response?**

Looking at the p-values associated with each predictor's t-statistic, we see that displacement, weight, year, and origin have a statistically significant relationship , while cylinders, horsepower, and acceleration do not.

(3.c)**(iii). What does the coefficient for the `year` variable suggest?**

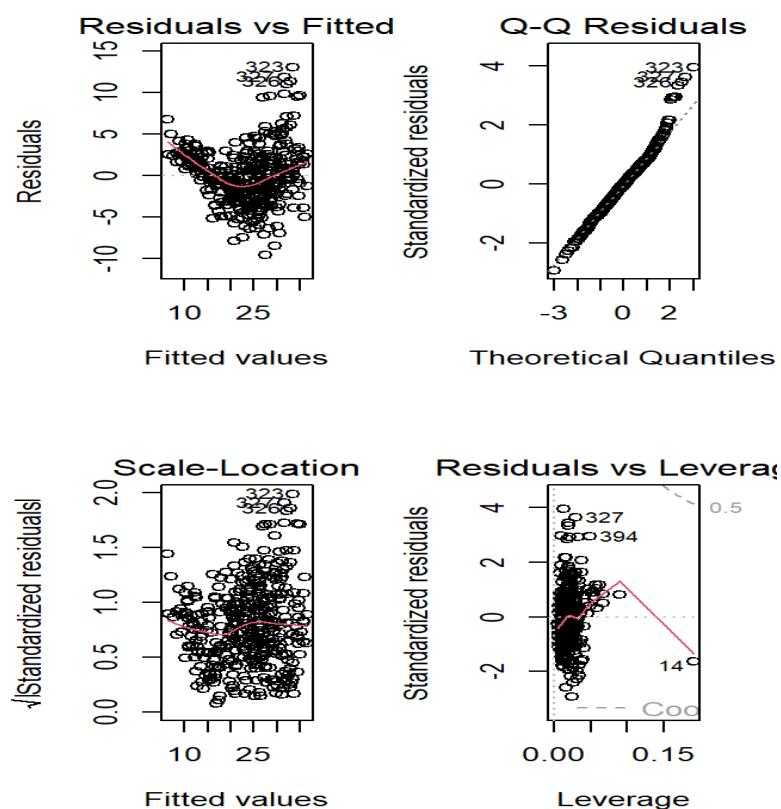
The regression coefficient for year, 0.7507727, suggests that for every one year, mpg increases by the coefficient. In other words, cars become more fuel efficient every year by almost 1 mpg / year.

(3d) AIM:- Use the plot() function to produce diagnostic plots of the linear regression fit. Comment on any problems you see with the fit. Do the residual plots suggest any unusually large outliers? Does the leverage plot identify any observations with unusually high leverage?

DESCRIPTION: Based on the plot of Residuals versus fitted values there appears to be non linearity in the data and based on the Residuals vs Leverage plot below, there does appear to be outliers due to the fact that there are data points outside of the -2 and 2 standardized residuals. And there does appear to be high leverage observations greater than 0.05.

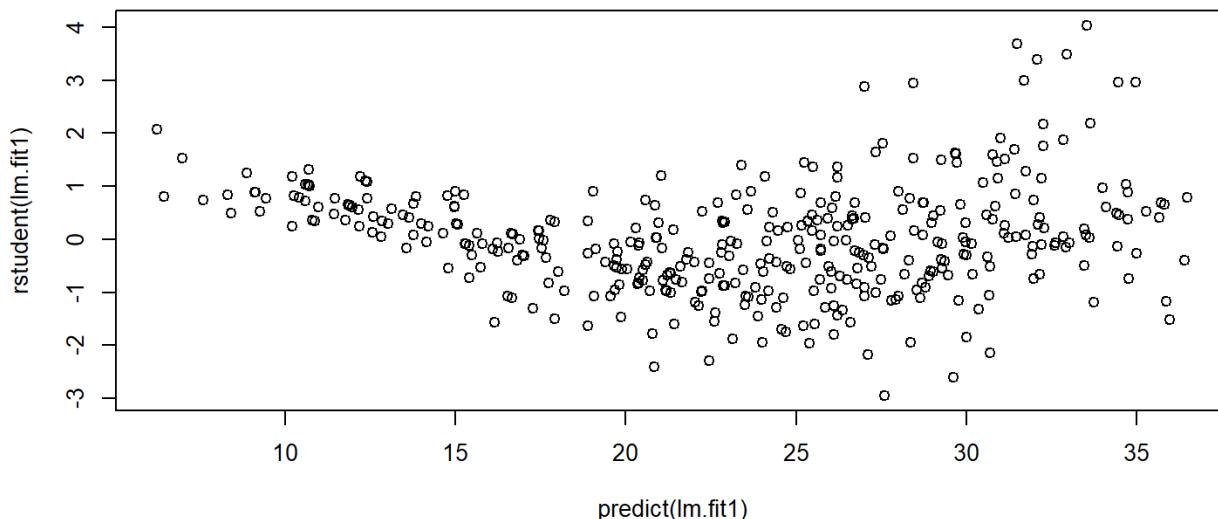
CODE:

```
par(mfrow = c(2, 2))
plot(lm.fit1)
```

OUTPUT:

CODE:

```
plot(predict(lm.fit1),rstudent(lm.fit1))
```

OUTPUT:

(3e) AIM:- Use the * and : symbols to fit linear regression models with interaction effects. Do any interactions appear to be statistically significant?

DESCRIPTION: Based on here it appears that the interactions between 'cylinders' and 'displacement' are not statistically significant but the interactions between 'horsepower' and 'weight' is significant and the interaction between 'acceleration' and 'year' is significant.

CODE:

```
lm.fit2 <- lm(mpg ~ cylinders * displacement + displacement * weight, data = Auto)
summary(lm.fit2)
```

OUTPUT:

```
> lm.fit2<-lm(mpg ~cylinders*displacement+displacement*weight,data=Auto)
> summary(lm.fit2)

Call:
lm(formula = mpg ~ cylinders * displacement + displacement *
    weight, data = Auto)

Residuals:
    Min      1Q      Median      3Q      Max 
-13.2934 -2.5184 -0.3476  1.8399 17.7723 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 5.262e+01 2.237e+00 23.519 < 2e-16 ***
cylinders   7.606e-01 7.669e-01  0.992  0.322    
displacement -7.351e-02 1.669e-02 -4.403 1.38e-05 ***
weight       -9.888e-03 1.329e-03 -7.438 6.69e-13 ***
cylinders:displacement -2.986e-03 3.426e-03 -0.872  0.384    
displacement:weight     2.128e-05 5.002e-06  4.254 2.64e-05 ***  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.103 on 386 degrees of freedom
Multiple R-squared:  0.7272,    Adjusted R-squared:  0.7237 
F-statistic: 205.8 on 5 and 386 DF,  p-value: < 2.2e-16
```

(3f) AIM:- Try a few different transformations of the variables, such as log(X), V X, X^2 . Comment on your findings.

DESCRIPTION: Applying the log() function on a vector, data frame, or other data set in R results in a log transformation. To avoid applying a logarithm to a 0 number, 1 is added to the base value prior to applying the logarithm.

CODE: summary(lm(mpg ~ . -name + log(acceleration), data=Auto))
summary(lm(mpg ~ . -name + I(horsepower^2), data=Auto))

OUTPUT:

```
> summary(lm(mpg ~ . -name + log(acceleration), data=Auto))

Call:
lm(formula = mpg ~ . - name + log(acceleration), data = Auto)

Residuals:
    Min      1Q  Median      3Q     Max 
-9.7931 -2.0052 -0.1279  1.9299 13.1085 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 4.552e+01 1.479e+01  3.077  0.00224 ***
cylinders   -2.796e-01 3.193e-01 -0.876  0.38172  
displacement 8.042e-03 7.805e-03  1.030  0.30344  
horsepower   -3.434e-02 1.401e-02 -2.450  0.01473 *  
weight       -5.343e-03 6.854e-04 -7.795 6.15e-14 *** 
acceleration 2.167e+00 4.782e-01  4.532 7.82e-06 *** 
year         7.560e-01 4.978e-02 15.186 < 2e-16 *** 
origin        1.329e+00 2.724e-01  4.877 1.58e-06 *** 
log(acceleration) -3.513e+01 7.886e+00 -4.455 1.10e-05 *** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.249 on 383 degrees of freedom
Multiple R-squared:  0.8303,    Adjusted R-squared:  0.8267 
F-statistic: 234.2 on 8 and 383 DF,  p-value: < 2.2e-16

> summary(lm(mpg ~ . -name + I(horsepower^2), data=Auto))

Call:
lm(formula = mpg ~ . - name + I(horsepower^2), data = Auto)

Residuals:
    Min      1Q  Median      3Q     Max 
-8.5497 -1.7311 -0.2236  1.5877 11.9955 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 1.3236564  4.6247696  0.286 0.774872  
cylinders   0.3489063  0.3048310  1.145 0.253094  
displacement -0.0075649  0.0073733 -1.026 0.305550  
horsepower  -0.3194633  0.0343447 -9.302 < 2e-16 *** 
weight      -0.0032712  0.0006787 -4.820 2.07e-06 *** 
acceleration -0.3305981  0.0991849 -3.333 0.000942 *** 
year         0.7353414  0.0459918 15.989 < 2e-16 *** 
origin       1.0144130  0.2545545  3.985 8.08e-05 *** 
I(horsepower^2) 0.0010060  0.0001065  9.449 < 2e-16 *** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.001 on 383 degrees of freedom
Multiple R-squared:  0.8552,    Adjusted R-squared:  0.8522 
F-statistic: 282.8 on 8 and 383 DF,  p-value: < 2.2e-16
```

EXPERIMENT-4**AIM: Implementation of KNN on the German Credit Data set.**

DESCRIPTION: To minimize loss from the bank's perspective, the bank needs a decision rule regarding who to give approval of the loan and who not to. An applicant's demographic and socio-economic profiles are considered by loan managers before a decision is taken regarding his/her loan application. The German Credit Data contains data on 20 variables and the classification whether an applicant is considered a Good or a Bad credit risk for 1000 loan applicants. A predictive model developed on this data is expected to provide bank manager guidance for making a decision whether to approve a loan to a prospective applicant based on his/her profiles

DESCRIPTION:

The K-NN working can be explained on the basis of the below algorithm:

- o Step-1: Select the number K of the neighbours
- o Step-2: Calculate the Euclidean distance of K number of neighbours
- o Step-3: Take the K nearest neighbours as per the calculated Euclidean distance.
- o Step-4: Among these k neighbours, count the number of the data points in each category.
- o Step-5: Assign the new data points to that category for which the number of the neighbour is maximum.
- o Step-6: Our model is ready.2

Steps Involved in performing KNN algorithm:

1. Data Collection.
2. Preparing and exploring the data.
 - Understanding data structure.
 - Feature selection (if required)
 - Data normalization.
 - Creating Training and Test data set.
3. Training a model on data.
4. Evaluate the model performance.
5. Improve the performance of model.

1.DATACOLLECTION:**CODE:**

```
gc <- read.csv("C:/Users/Hp/OneDrive/Desktop/ML-LAB/german_credit.csv")
gc.bkup <- gc
head(gc)
```

OUTPUT:

```
> gc <- read.csv("C:/Users/Hp/OneDrive/Desktop/ML-LAB/german_credit.csv")
> gc.bkup <- gc
> head (gc)
   Creditability Account.Balance Duration.of.Credit..month. Payment.Status.of.Previous.Credit Purpose
1           1             1                  18                      4                 2
2           1             1                  9                      4                 0
3           1             2                  12                      2                 9
4           1             1                  12                      4                 0
5           1             1                  12                      4                 0
6           1             1                  10                      4                 0
   Credit.Amount Value.Savings.Stocks Length.of.current.employment Instalment.per.cent Sex...Marital.Status
1        1049                  1                  2                      4                 2
2        2799                  1                  3                      2                 3
3         841                  2                  4                      2                 2
4        2122                  1                  3                      3                 3
5        2171                  1                  3                      4                 3
6        2241                  1                  2                      1                 3
   Guarantors Duration.in.Current.address Most.valuable.available.asset Age..years. Concurrent.credits
1           1                  4                  2                  21                   3
2           1                  2                  1                  36                   3
3           1                  4                  1                  23                   3
4           1                  2                  1                  39                   3
5           1                  4                  2                  38                   1
6           1                  3                  1                  48                   3
   Type.of.apartment No.of.Credits.at.this.Bank Occupation No.of.dependents Telephone Foreign.Worker
1           1                  1                  3                  1                  1                 1
2           1                  2                  3                  2                  1                 1
3           1                  1                  2                  1                  1                 1
4           1                  2                  2                  2                  1                 2
5           2                  2                  2                  1                  1                 2
6           1                  2                  2                  2                  2                  1                 2
```

2.PREPARING AND EXPLORING THE DATA:

CODE:

```

str(gc)
gc.subset <-
gc[c('Creditability','Age..years.','Sex...Marital.Status','Occupation','Account.Balance','Credit.Amount','Length.of.current.employment','Purpose')]
head(gc.subset)
normalize <- function(x) { return ((x - min(x)) / (max(x) - min(x))) }
gc.subset.n<- as.data.frame(lapply(gc.subset[,2:8], normalize))
head(gc.subset.n)
set.seed(123)
dat.d <- sample(1:nrow(gc.subset.n),size=nrow(gc.subset.n)*0.7,replace = FALSE)
train.gc <- gc.subset[dat.d,]
test.gc <- gc.subset[-dat.d,]
train.gc_labels <- gc.subset[dat.d,1]
test.gc_labels <- gc.subset[-dat.d,1]

```

OUTPUT:

```

> str(gc)
'data.frame': 1000 obs. of 21 variables:
 $ Creditability : int 1 1 1 1 1 1 1 1 1 1 ...
 $ Account.Balance : int 1 1 2 1 1 1 1 1 4 2 ...
 $ Duration.of.Credit..month. : int 18 9 12 12 12 10 8 6 18 24 ...
 $ Payment.Status.of.Previous.Credit: int 4 4 2 4 4 4 4 4 4 2 ...
 $ Purpose : int 2 0 9 0 0 0 0 0 0 3 3 ...
 $ Credit.Amount : int 1049 2799 841 2122 2171 2241 3398 1361 1098 3758 ...
 $ Value.Savings.Stocks : int 1 1 2 1 1 1 1 1 1 3 ...
 $ Length.of.current.employment : int 2 3 4 3 3 2 4 2 1 1 ...
 $ Instalment.per.cent : int 4 2 2 3 4 1 1 2 4 1 ...
 $ Sex...Marital.Status : int 2 3 2 3 3 3 3 2 2 2 ...
 $ Guarantors : int 1 1 1 1 1 1 1 1 1 1 ...
 $ Duration.in.Current.address : int 4 2 4 2 4 3 4 4 4 4 ...
 $ Most.valuable.available.asset : int 2 1 1 1 2 1 1 1 3 4 ...
 $ Age..years. : int 21 36 23 39 38 48 39 40 65 23 ...
 $ Concurrent.Credits : int 3 3 3 3 1 3 3 3 3 3 ...
 $ Type.of.apartment : int 1 1 1 1 2 1 2 2 2 1 ...
 $ No.of.Credits.at.this.Bank : int 1 2 1 2 2 2 2 1 2 1 ...
 $ Occupation : int 3 3 2 2 2 2 2 2 2 1 ...
 $ No.of.dependents : int 1 2 1 2 1 2 1 2 1 1 ...
 $ Telephone : int 1 1 1 1 1 1 1 1 1 1 ...
 $ Foreign.worker : int 1 1 1 2 2 2 2 2 1 1 ...
> gc.subset <- gc[c('Creditability','Age..years.','Sex...Marital.Status','Occupation','Account.Balance','Credit.Amount','Length.of.current.employment','Purpose')]
> head(gc.subset)
Creditability Age..years. Sex...Marital.Status Occupation Account.Balance Credit.Amount
1 1 21 2 3 1 1049
2 1 36 3 3 1 2799
3 1 23 2 2 2 841
4 1 39 3 2 1 2122
5 1 38 3 2 1 2171
6 1 48 3 2 1 2241
Length.of.current.employment Purpose
1 2 2
2 3 0
3 4 9
4 3 0
5 3 0
6 2 0
> normalize <- function(x) { return ((x - min(x)) / (max(x) - min(x))) }
> gc.subset.n<- as.data.frame(lapply(gc.subset[,2:8], normalize))
> head(gc.subset.n)
Age..years. Sex...Marital.Status Occupation Account.Balance Credit.Amount Length.of.current.employment Purpose
1 0.03571429 0.3333333 0.6666667 0.0000000 0.04396390 0.25 0.2
2 0.30357143 0.6666667 0.6666667 0.0000000 0.14025531 0.50 0.0
3 0.07142857 0.3333333 0.3333333 0.3333333 0.03251898 0.75 0.9
4 0.35714286 0.6666667 0.3333333 0.0000000 0.10300429 0.50 0.0
5 0.33928571 0.6666667 0.3333333 0.0000000 0.10570045 0.50 0.0
6 0.51785714 0.6666667 0.3333333 0.0000000 0.10955211 0.25 0.0
> set.seed(123)
> dat.d <- sample(1:nrow(gc.subset.n),size=nrow(gc.subset.n)*0.7,replace = FALSE)
> train.gc <- gc.subset[dat.d,]
> test.gc <- gc.subset[-dat.d,]
> train.gc_labels <- gc.subset[dat.d,1]
> test.gc_labels <- gc.subset[-dat.d,1]

```

3.TRAINING A MODEL ON DATA:**CODE:**

```
library(class)
NROW(train.gc_labels)
knn.26 <- knn(train=train.gc, test=test.gc, cl=train.gc_labels, k=26)
knn.27 <- knn(train=train.gc, test=test.gc, cl=train.gc_labels, k=27)
```

OUTPUT:

```
> library(class)
> NROW(train.gc_labels)
[1] 700
> knn.26 <- knn(train=train.gc, test=test.gc, cl=train.gc_labels, k=26)
> knn.27 <- knn(train=train.gc, test=test.gc, cl=train.gc_labels, k=27)

```

4.EVALUATE THE MODEL PERFORMANCE:**CODE:**

```
ACC.26 <- 100 * sum(test.gc_labels == knn.26)/NROW(test.gc_labels)
ACC.27 <- 100 * sum(test.gc_labels == knn.27)/NROW(test.gc_labels)
ACC.26
ACC.27
table(knn.26 ,test.gc_labels)
table(knn.27 ,test.gc_labels)
install.packages('caret')
library(caret)
test.gc_labels=as.factor(test.gc_labels)
confusionMatrix(knn.26 ,test.gc_labels)
```

OUTPUT:

```
> ACC.26 <- 100 * sum(test.gc_labels == knn.26)/NROW(test.gc_labels)
> ACC.27 <- 100 * sum(test.gc_labels == knn.27)/NROW(test.gc_labels)
> ACC.26
[1] 69
> ACC.27
[1] 69
> table(knn.26 ,test.gc_labels)
  test.gc_labels
knn.26    0   1
          0   8   6
          1 87 199
> table(knn.27 ,test.gc_labels)
  test.gc_labels
knn.27    0   1
          0   8   6
          1 87 199
> install.packages('caret')
Error in install.packages : updating loaded packages
> library(caret)
> test.gc_labels=as.factor(test.gc_labels)
> confusionMatrix(knn.26 ,test.gc_labels)
Confusion Matrix and Statistics

             Reference
Prediction      0      1
      0     8     6
      1    87   199

          Accuracy : 0.69
          95% CI : (0.6343, 0.7419)
          No Information Rate : 0.6833
          P-Value [Acc > NIR] : 0.4291

          Kappa : 0.0712

McNemar's Test P-Value : <2e-16

          Sensitivity : 0.08421
          Specificity : 0.97073
          Pos Pred Value : 0.57143
          Neg Pred Value : 0.69580
          Prevalence : 0.31667
          Detection Rate : 0.02667
          Detection Prevalence : 0.04667
          Balanced Accuracy : 0.52747

'Positive' Class : 0
```

CODE:

```
confusionMatrix(knn.27 ,test.gc_labels)
```

OUTPUT:

```
> confusionMatrix(knn.27 ,test.gc_labels)
Confusion Matrix and Statistics

             Reference
Prediction      0      1
      0     8     6
      1    87   199

          Accuracy : 0.69
          95% CI  : (0.6343, 0.7419)
No Information Rate : 0.6833
P-Value [Acc > NIR] : 0.4291

          Kappa : 0.0712

McNemar's Test P-Value : <2e-16

          Sensitivity : 0.08421
          Specificity  : 0.97073
          Pos Pred Value : 0.57143
          Neg Pred Value : 0.69580
          Prevalence    : 0.31667
          Detection Rate : 0.02667
          Detection Prevalence : 0.04667
          Balanced Accuracy : 0.52747

          'Positive' Class : 0
```

5. IMPROVE THE PERFORMANCE OF MODEL:**CODE:**

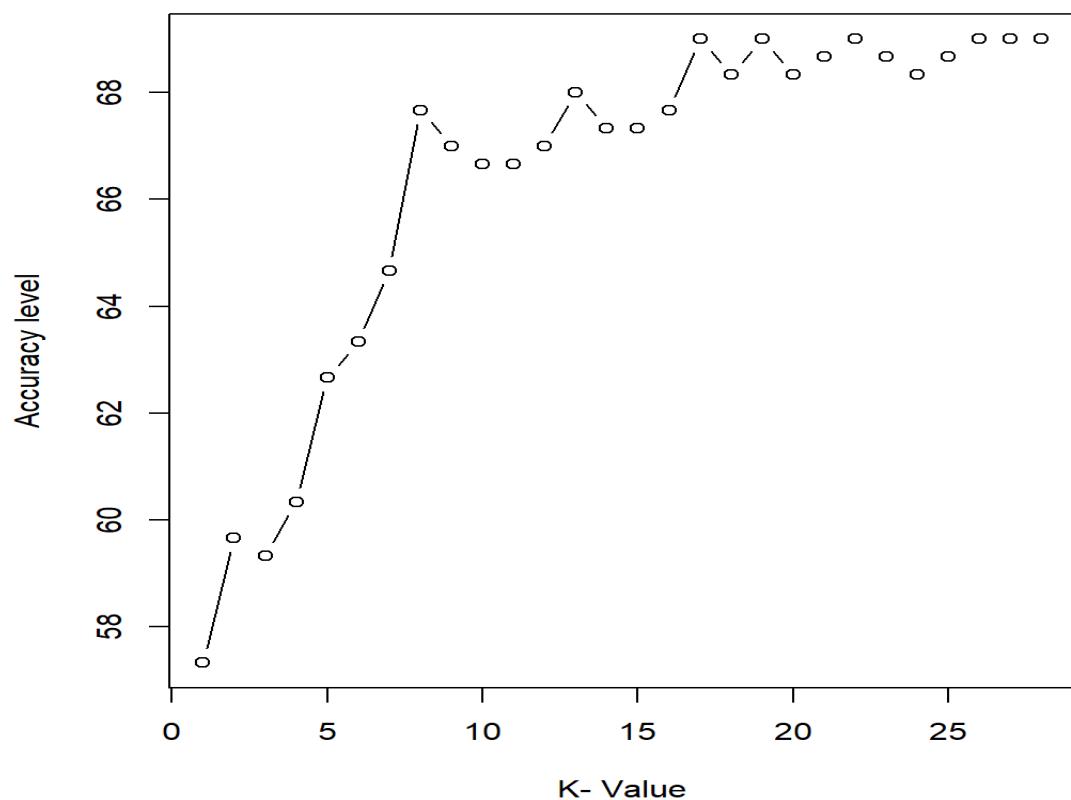
```
i=1
k.optm=1
for (i in 1:28){
  knn.mod <- knn(train=train.gc, test=test.gc, cl=train.gc_labels, k=i)
  k.optm[i] <- 100 * sum(test.gc_labels == knn.mod)/NROW(test.gc_labels)
  k=i
  cat(k,'=',k.optm[i],'\n')
}
```

OUTPUT:

```
> i=1
> k.optm=1
> for (i in 1:28){
+   knn.mod <- knn(train=train.gc, test=test.gc, cl=train.gc_labels, k=i)
+   k.optm[i] <- 100 * sum(test.gc_labels == knn.mod)/NROW(test.gc_labels)
+   k=i
+   cat(k,'=',k.optm[i],'\n')
+
1 = 57.33333
2 = 59.66667
3 = 59.33333
4 = 60.33333
5 = 62.66667
6 = 63.33333
7 = 64.66667
8 = 67.66667
9 = 67
10 = 66.66667
11 = 66.66667
12 = 67
13 = 68
14 = 67.33333
15 = 67.33333
16 = 67.66667
17 = 69
18 = 68.33333
19 = 69
20 = 68.33333
21 = 68.66667
22 = 69
23 = 68.66667
24 = 68.33333
25 = 68.66667
26 = 69
27 = 69
28 = 69
> plot(k.optm, type="b", xlab="K- value", ylab="Accuracy level")
```

CODE:

```
plot(k.optm, type="b", xlab="K- Value",ylab="Accuracy level")
```

OUTPUT:

EXPERIMENT NO: 5

AIM: To implement Linear Discriminant Analysis, Quadratic Discriminant Analysis and Naive Bayes algorithms on Stock market Dataset.

DESCRIPTION:

- **Linear Discriminant Analysis (LDA)** is a supervised learning algorithm used for classification tasks in machine learning. It is a technique used to find a linear combination of features that best separates the classes in a dataset. Quadratic Discriminant Analysis (QDA) is a supervised learning algorithm and extension of linear discriminant analysis. QDA models are designed to be used for classification problems i.e. when the response variable can be placed into classes or categories. Naive Bayes classifiers (NB) are a collection of classification algorithms based on Bayes Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle ,i.e. every pair of features being classified is independent of each other.

Formula for LDA :-

For a two-class problem, LDA calculates the discriminant function as follows:

For Class 1:

$$D_1(x) = x \cdot \frac{\mu_1}{\sigma^2} - \frac{\mu_1^2}{2\sigma^2} + \log(\pi_1)$$

For Class 2:

$$D_2(x) = x \cdot \frac{\mu_2}{\sigma^2} - \frac{\mu_2^2}{2\sigma^2} + \log(\pi_2)$$

Where:

- $D_1(x)$ and $D_2(x)$ are the discriminant functions for Class 1 and Class 2.
- x is the feature vector.
- μ_1 and μ_2 are the means of the features for Class 1 and Class 2.
- σ^2 is the common variance of the features.
- π_1 and π_2 are the prior probabilities of Class 1 and Class 2.

- **Quadratic Discriminant Analysis (QDA)** is a supervised machine learning algorithm used for classification tasks. It is a generative model, meaning that it assumes that the data is generated from a known distribution, in this case a multivariate Gaussian distribution. QDA estimates the mean and covariance matrix for each class, and then uses this information to calculate the probability of a new data point belonging to each class. The data point is then assigned to the class with the highest probability.

Formula for QDA :-

$$D_k(x) = -\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) - \frac{1}{2} \log |\Sigma_k| + \log(\pi_k)$$

Where:

- $D_k(x)$ is the discriminant function for Class k .
- x is the feature vector.
- μ_k is the mean vector of the features for Class k .
- Σ_k is the covariance matrix for Class k .
- π_k is the prior probability of Class k .

- **Naive Bayes** is a supervised machine learning algorithm used for classification tasks. It is a probabilistic classifier, which means that it predicts on the basis of the probability of an object. Naive Bayes is based on Bayes' theorem, which is a mathematical formula for calculating the probability of an event occurring, given that another event has already occurred. In the context of Naive Bayes, the event we are trying to predict is the class label of a data point, and the events we have already observed are the values of the data point's features.

Steps for Navie Bayes :-

1. Calculate the prior probabilities $P(C_k)$ for each class.
2. Estimate the likelihood $P(x|C_k)$ for each feature given each class. This is often done using probability density functions.
3. For a new observation with features x , calculate the posterior probabilities $P(C_k|x)$ for each class using Bayes' theorem.
4. Assign the observation to the class with the highest posterior probability.

Formula for Navie Bayes :-

$$P(C_k|x) = \frac{P(x|C_k) \cdot P(C_k)}{P(x)}$$

Where:

- $P(C_k|x)$ is the posterior probability of class C_k given the features x .
- $P(x|C_k)$ is the likelihood of observing features x given class C_k .
- $P(C_k)$ is the prior probability of class C_k .
- $P(x)$ is the probability of observing features x .

CODE:

Load required libraries

```
library(quantmod)
library(MASS)
library(e1071)
library(caret)
library(ggplot2)
library(pROC)

#Fetch the stock market data
symbols <- c("AAPL","MSFT")
#calculate daily returns
start_date <- "2020-01-01"
end_date <- "2023-01-01"
getSymbols(symbols, from = start_date, to = end_date)
```

OUTPUT:

```
> getSymbols(symbols, from = start_date, to = end_date)
[1] "AAPL" "MSFT"
```

CODE:

```
stock_data <- merge(Ad(AAPL),Ad(MSFT))
colnames(stock_data) <- symbols
stock_returns <- diff(log(stock_data))
#creating labels based on return threshold
return_threshold <- 0.01
stock_labels <- ifelse(apply(abs(stock_returns), 1, max) > return_threshold, "High", "Low")
# Combine returns and labels into a data frame
stock_df <- data.frame(stock_returns, stock_labels)
# Split data into training and testing sets
set.seed(123)
train_indices <- sample(1:nrow(stock_df), 0.75 * nrow(stock_df))
train_data <- stock_df[train_indices, ]
test_data <- stock_df[-train_indices, ]
# When the input data contains missing values, make sure our 'test_data' does not have any missing or NA values. We can use the is.na() function to identify them
any_na <- apply(test_data, 2, function(column)
  any(is.na(column)))
print(any_na)
```

CODE:

```
print(any_na)
```

OUTPUT:

```
> print(any_na)
  AAPL      MSFT stock_labels
  TRUE      TRUE      TRUE
```

CODE:

```
# If we find missing values, we can clean the data by removing or imputing them before using it for prediction.
```

```
# Remove rows with missing values
```

```
test_data <- test_data[complete.cases(test_data), ]
```

#LDA

```
lda_model <- lda(stock_labels ~ ., data = train_data)
lda_predictions <- predict(lda_model, newdata = test_data)$class
```

#QDA

```
qda_model <- qda(stock_labels ~ ., data = train_data)
qda_predictions <- predict(qda_model, newdata = test_data)$class
```

#NAIVEBAYES Classification

```
nb_model <- naiveBayes(stock_labels ~ ., data = train_data)
nb_predictions <- predict(nb_model, newdata = test_data)
```

#calculate accuracies

```
lda_accuracy <- mean(lda_predictions == test_data$stock_labels)
qda_accuracy <- mean(qda_predictions == test_data$stock_labels)
nb_accuracy <- mean(nb_predictions == test_data$stock_labels)
```

Print the accuracy results

```
cat("LDA ACCURACY:", lda_accuracy, "\n")
cat("QDA ACCURACY:", qda_accuracy, "\n")
cat("navive bayes accuracy:", nb_accuracy, "\n")
```

OUTPUT:

```
> cat("LDA ACCURACY:" , lda_accuracy, "\n")
LDA ACCURACY: 0.75
> cat("QDA ACCURACY:" , qda_accuracy, "\n")
QDA ACCURACY: 0.9787234
> cat("navive bayes accuracy:", nb_accuracy, "\n")
navive bayes accuracy: 0.9734043
```

CODE:

```
# Inorder to avoid the error: `data` and `reference` should be factors with the same levels.
```

```
test_data$stock_labels=as.factor(test_data$stock_labels)
```

#Calculate and print confusion matrix

```
lda_cm <- confusionMatrix(lda_predictions, test_data$stock_labels)
qda_cm <- confusionMatrix(qda_predictions, test_data$stock_labels)
nb_cm <- confusionMatrix(nb_predictions, test_data$stock_labels)
print(lda_cm)
print(qda_cm)
print(nb_cm)
```

OUTPUT:

```

> print(lda_cm)
Confusion Matrix and Statistics

      Reference
Prediction  High  Low
    High   141   47
    Low     0    0

    Accuracy : 0.75
    95% CI  : (0.6818, 0.8102)
    No Information Rate : 0.75
    P-Value [Acc > NIR] : 0.5391

    Kappa : 0
    McNemar's Test P-Value : 1.949e-11

McNemar's Test P-Value : 1.949e-11

      Sensitivity : 1.00
      Specificity : 0.00
      Pos Pred Value : 0.75
      Neg Pred Value : NaN
      Prevalence : 0.75
      Detection Rate : 0.75
      Detection Prevalence : 1.00
      Balanced Accuracy : 0.50
      'Positive' Class : High

> print(nb_cm)
Confusion Matrix and Statistics

      Reference
Prediction  High  Low
    High   137   1
    Low     4   46

    Accuracy : 0.9734
    95% CI  : (0.939, 0.9913)
    No Information Rate : 0.75
    P-Value [Acc > NIR] : <2e-16

    Kappa : 0.9306
    McNemar's Test P-Value : 0.3711

      Sensitivity : 0.9716
      Specificity : 0.9787
      Pos Pred Value : 0.9928
      Neg Pred Value : 0.9200
      Prevalence : 0.7500
      Detection Rate : 0.7287
      Detection Prevalence : 0.7340
      Balanced Accuracy : 0.9752
      'Positive' Class : High

> print(qda_cm)
Confusion Matrix and Statistics

      Reference
Prediction  High  Low
    High   139   2
    Low     2   45

    Accuracy : 0.9787
    95% CI  : (0.9464, 0.9942)
    No Information Rate : 0.75
    P-Value [Acc > NIR] : <2e-16

    Kappa : 0.9433
  
```

CODE:**# Create data frame for accuracy comparison**

```
accuracy_df <- data.frame(Classifier = c("LDA", "QDA", "Naive Bayes"), Accuracy =
c(lda_accuracy,qda_accuracy,nb_accuracy))
```

#create accuracy plot

```
accuracy_plot <- ggplot(accuracy_df, aes(x= Classifier, y= Accuracy, fill = Classifier))+ geom_bar(stat = "identity",
position = "dodge")+labs(y = "Accuracy", title = "Classifier Comparision") + theme_minimal() + theme(legend.position =
"bottom")
print(accuracy_plot)
```

#create ROC curves

```
roc_lda <- roc(test_data$stock_labels, as.numeric(lda_predictions == "high"))
roc_qda <- roc(test_data$stock_labels, as.numeric(qda_predictions == "High"))
roc_nb <- roc(test_data$stock_labels, as.numeric(nb_predictions == "High"))
```

Determine the common length of sensitivities

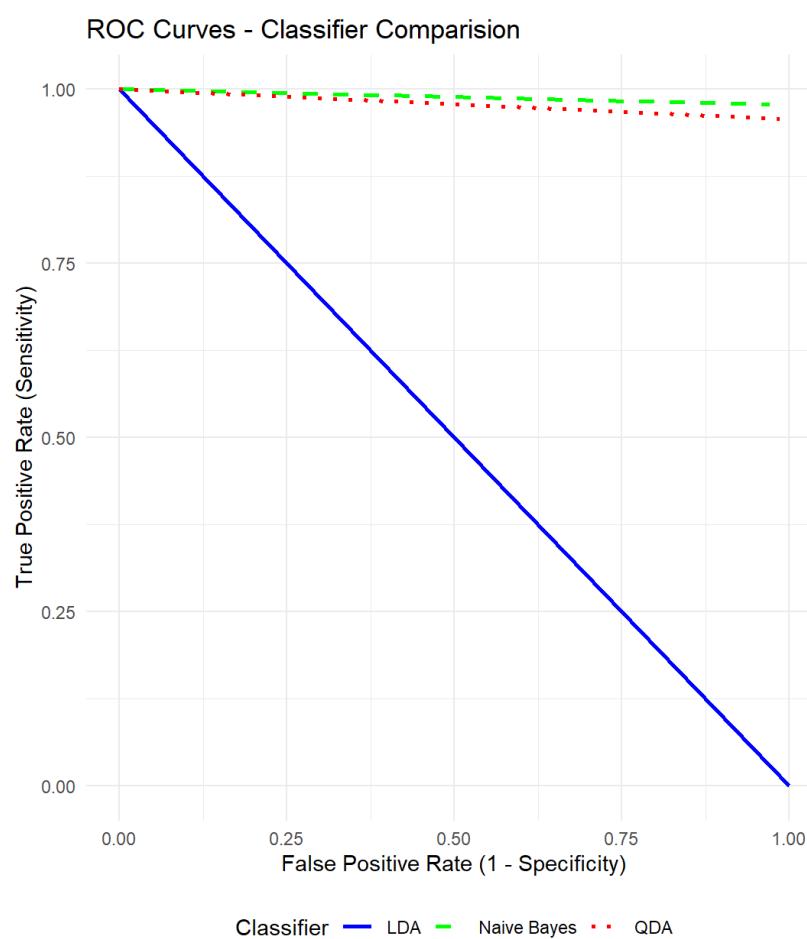
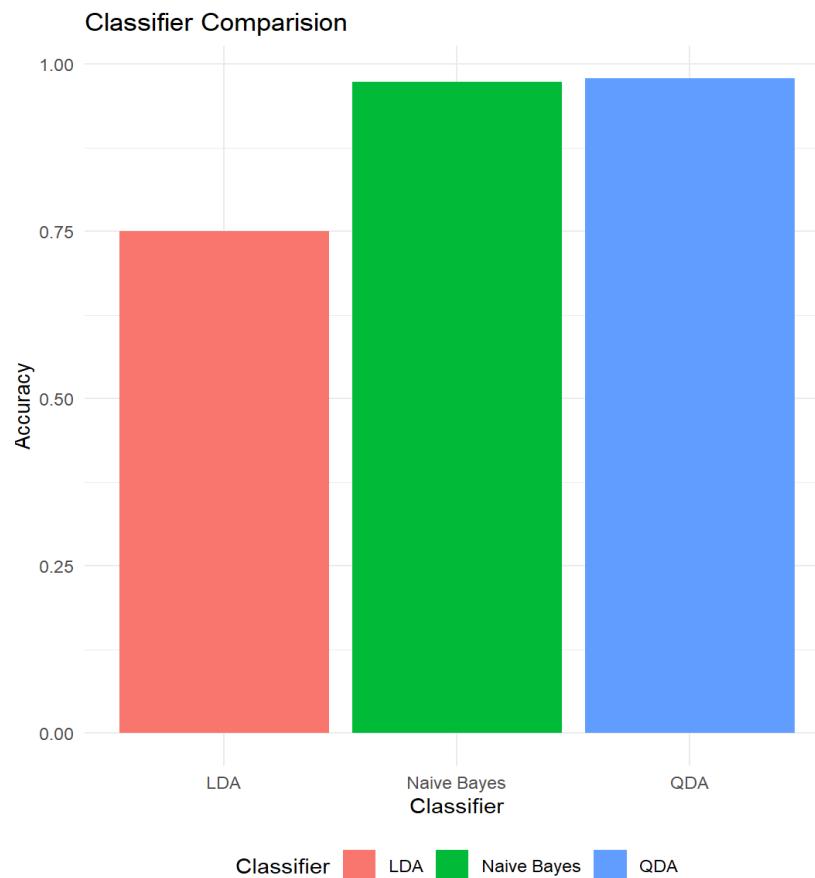
```
min_length <- min(length(roc_lda$sensitivities), length(roc_qda$sensitivities), length(roc_nb$sensitivities))
```

Combine ROC curve data into a single data frame

```
roc_data <- data.frame(
  FPR = c(roc_lda$specificities[1:min_length], roc_qda$specificities[1:min_length],
  roc_nb$specificities[1:min_length]),
  TPR = c(roc_lda$sensitivities[1:min_length], roc_qda$sensitivities[1:min_length],
  roc_nb$sensitivities[1:min_length]),
  Classifier = rep(c("LDA", "QDA", "Naive Bayes"), each = min_length)
)
```

Create combined ROC curve plot

```
roc_combined_plot <- ggplot(roc_data, aes(x = FPR, y = TPR, color = Classifier, linetype = Classifier)) +
  geom_line(linewidth = 1) +
  labs(x = "False Positive Rate (1 - Specificity)", y = "True Positive Rate (Sensitivity)", title = "ROC Curves - Classifier
Comparision") +
  scale_color_manual(values = c("blue", "green", "red"))+
  scale_linetype_manual(values = c("solid", "dashed", "dotted"))+
  theme_minimal()+
  theme(legend.position = "bottom")
print(roc_combined_plot)
```

OUTPUT:

Findings:

Quadratic Discriminant Analysis (QDA) is the best algorithm with accuracy rate 97.8% among the Linear Discriminant Analysis(LDA) with accuracy rate 75% ,naive bayes classification with accuracy rate 97.3%.

EXPERIMENT NO:6**AIM:**

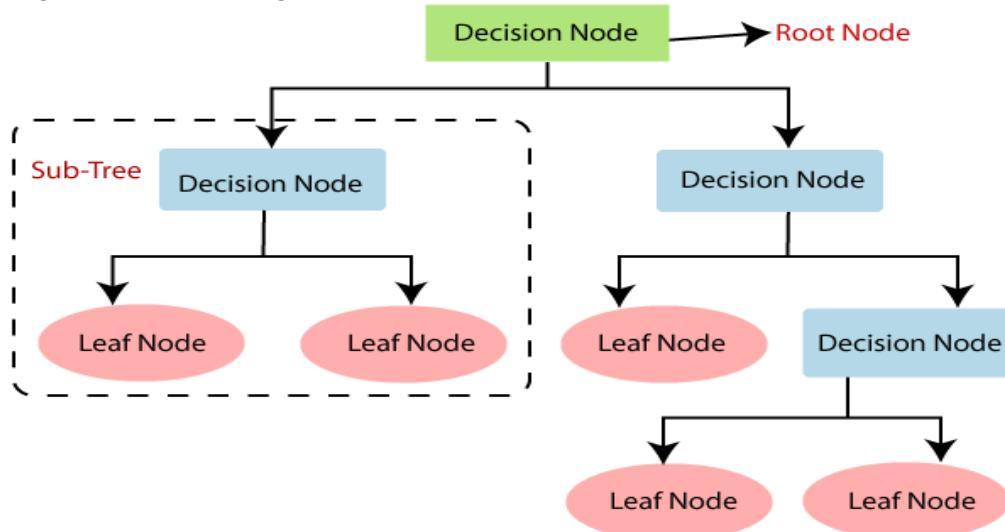
Write R program to Implement decision trees using mushrooms dataset.

DESCRIPTION:**Decision Tree Classification Algorithm:**

Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome**.

Note: A decision tree can contain categorical data (YES/NO) as well as numeric data.

- Below diagram explains the general structure of a decision tree.

**Decision Tree Terminologies:**

Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

How does the Decision Tree algorithm Work?

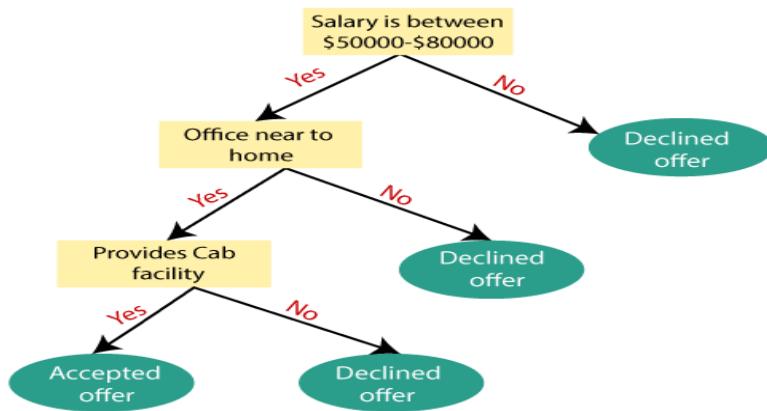
In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM)**.
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.

- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures:

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM**. By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
1. Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)]

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

$$\text{Entropy}(S) = -P(\text{yes}) \log_2 P(\text{yes}) - P(\text{no}) \log_2 P(\text{no})$$

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

2. Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- Gini index can be calculated using the below formula:

Gini Index= $1 - \sum_j P_j^2$

Pruning: Getting an Optimal Decision tree:

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

- **Cost Complexity Pruning**
- **Reduced Error Pruning.**

Steps involved in Implementation of Decision Tree:

For this, we will use the dataset "[user_data.csv](#)," which we have used in previous classification models. By using the same dataset, we can compare the Decision tree classifier with other classification models such as [KNN](#) [SVM](#), [LogisticRegression](#), etc.

Steps will also remain the same, which are given below:

- **Data Pre-processing step**
- **Fitting a Decision-Tree algorithm to the Training set**
- **Predicting the test result**
- **Test accuracy of the result(Creation of Confusion matrix)**
- **Visualizing the test set result.**

CODE:

```
library(rpart)
library(rpart.plot)
library(partykit)
library(caret)
library(pROC)
library(ROCR)
```

OUTPUT:

```
R 4.2.2 · ~/Documents · Background Jobs ×
Console Terminal × Background Jobs ×
> library(rpart)
> library(rpart.plot)
warning message:
package 'rpart.plot' was built under R version 4.2.3
> library(partykit)
Loading required package: grid
Loading required package: libcoin
Loading required package: mvtnorm
Warning messages:
1: package 'partykit' was built under R version 4.2.3
2: package 'libcoin' was built under R version 4.2.3
3: package 'mvtnorm' was built under R version 4.2.3
> library(caret)
Loading required package: ggplot2
Loading required package: lattice
Warning messages:
1: package 'caret' was built under R version 4.2.3
2: package 'ggplot2' was built under R version 4.2.3
> library(pROC)
Type 'citation("pROC")' for a citation.
Attaching package: 'pROC'

The following objects are masked from 'package:stats':
  cov, smooth, var

Warning message:
package 'pROC' was built under R version 4.2.3
> library(ROCR)
Warning message:
package 'ROCR' was built under R version 4.2.3
> |
```

CODE:

```
#Load the mushrooms data set
dataset<-read.csv("D:\VVIT.BTECH\3_1\ML_lab\exp-6\mushrooms.csv")
#Convert class labels to factors
dataset$class<-as.factor(dataset$class)
set.seed(123)
sample_indices<-sample(nrow(dataset),0.8*nrow(dataset))
train_data<-dataset[sample_indices,]
test_data<-dataset[-sample_indices,]
```

OUTPUT:

```
> dataset<-read.csv("C:\\Users\\DELL\\Downloads\\mushrooms.csv")
> #Convert class labels to factors
> dataset$class<-as.factor(dataset$class)
> set.seed(123)
> sample_indices<-sample(nrow(dataset),0.8*nrow(dataset))
> train_data<-dataset[sample_indices,]
> |
```

CODE:

```
#Build the model
tree_model<-rpart(class~,data=train_data,method="class")
print(tree_model)
#Make predictions on test set
predictions=predict(tree_model,test_data,type = "class")
#Accuracy score
accuracy=sum(predictions == test_data$class)/nrow(test_data)
cat("Accuracy:",accuracy,"\n")
```

OUTPUT:

```
> #Build the model
> tree_model<-rpart(class~,data=train_data,method="class")
> print(tree_model)
n= 6499

node), split, n, loss, yval, (yprob)
 * denotes terminal node

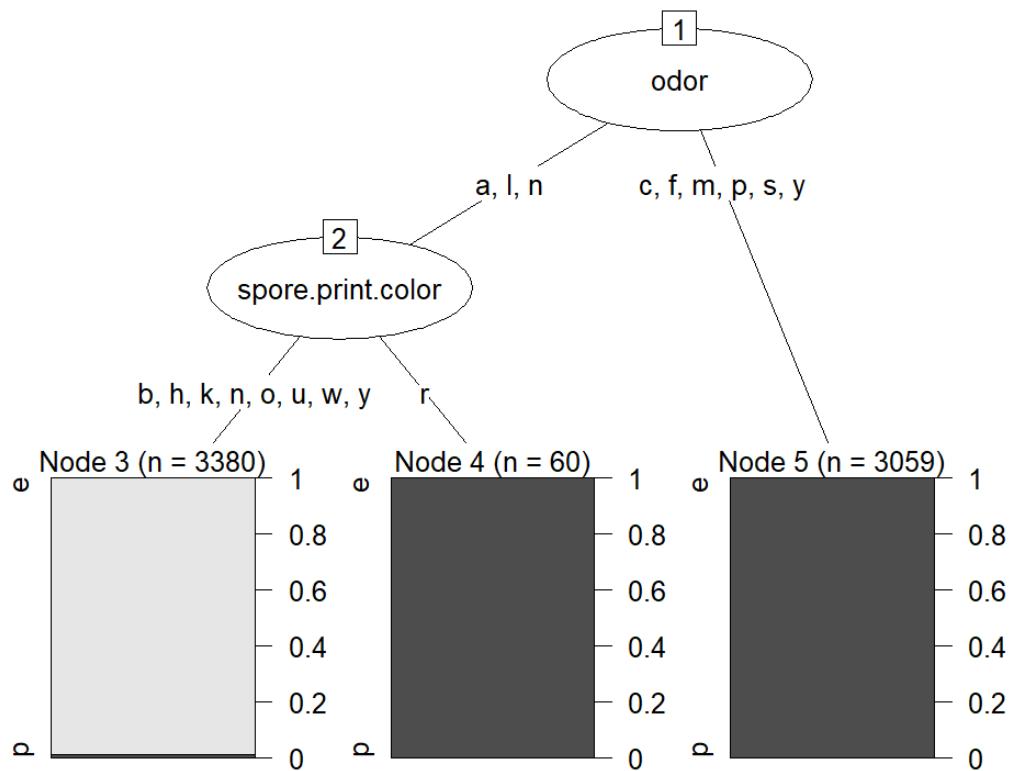
1) root 6499 3155 e (0.51454070 0.48545930)
   2) odor=a,1,n 3440  96 e (0.97209302 0.02790698)
      4) spore.print.color=b,h,k,n,o,u,w,y 3380   36 e (0.98934911 0.01065089) *
      5) spore.print.color=r 60      0 p (0.00000000 1.00000000) *
   3) odor=c,f,m,p,s,y 3059    0 p (0.00000000 1.00000000) *
> #Make predictions on test set
> predictions=predict(tree_model,test_data,type = "class")
> #Accuracy score
> accuracy=sum(predictions == test_data$class)/nrow(test_data)
> cat("Accuracy:",accuracy,"\n")
Accuracy: 0.9926154
> |
```

CODE:

```
#Visualize the tree
party_tree<-as.party(tree_model)
plot(party_tree)
```

OUTPUT:

```
> #visualize the tree
> party_tree<-as.party(tree_model)
> plot(party_tree)
> |
```



CODE:

```
#Display the few predictions
```

```
predictions=predict(tree_model,newdata=test_data,type = "class")
head(predictions)
```

```
#Calculate the precision,recall and F1-score.
```

```
actual_labels<-test_data$class
confusion_matrix<-table(actual_labels,predictions)
print(confusion_matrix)
```

OUTPUT:

```
> #Display the few predictions
> predictions=predict(tree_model,newdata=test_data,type = "class")
> head(predictions)
 3 8 10 12 14 15
 e e e e p e
Levels: e p
> actual_labels<-test_data$class
> confusion_matrix<-table(actual_labels,predictions)
> print(confusion_matrix)
      predictions
actual_labels   e   p
              e 864   0
              p 12 749
> |
```

CODE:

```
precision<-diag(confusion_matrix)/rowSums(confusion_matrix)
recall<-diag(confusion_matrix)/colSums(confusion_matrix)
f1_score<-2*(precision*recall)/(precision+recall)
```

```
result_df<-data.frame(Precision=precision,Recall=recall,F1_Score=f1_score)
rownames(result_df)<-levels(predictions)
print(result_df)
```

OUTPUT:

```
> precision<-diag(confusion_matrix)/rowSums(confusion_matrix)
> recall<-diag(confusion_matrix)/colSums(confusion_matrix)
> f1_score<-2*(precision*recall)/(precision+recall)
> result_df<-data.frame(Precision=precision,Recall=recall,F1_Score=f1_score)
> rownames(result_df)<-levels(predictions)
> print(result_df)
  Precision    Recall   F1_Score
e 1.0000000 0.9863014 0.9931034
p 0.9842313 1.0000000 0.9920530
```

CODE:

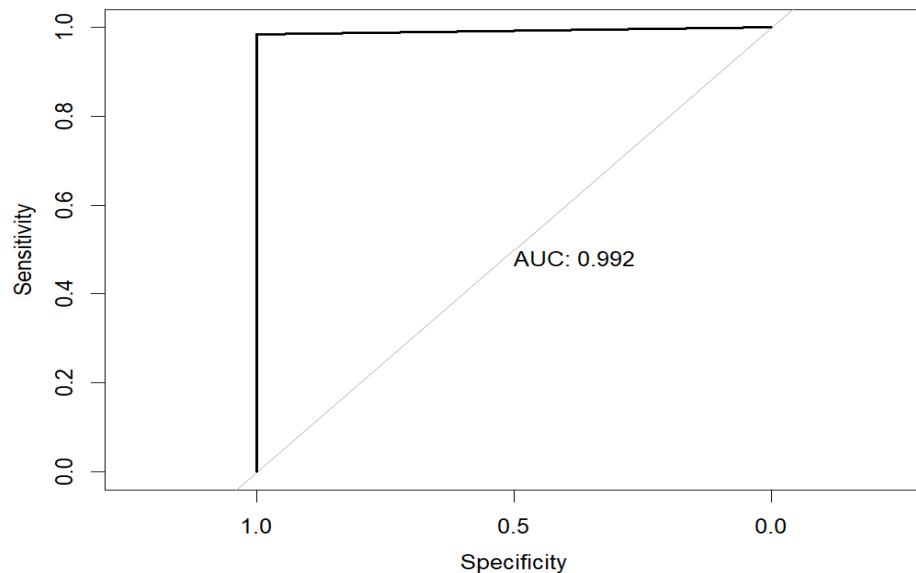
```
binary_predictions<-ifelse(predictions=="p",1,0)

roc_obj<-roc(as.numeric(test_data$class == "p"),binary_predictions)

plot(roc_obj,print.auc=TRUE)
```

OUTPUT:

```
> binary_predictions<-ifelse(predictions=="p",1,0)
> roc_obj<-roc(as.numeric(test_data$class == "p"),binary_predictions)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> plot(roc_obj,print.auc=TRUE)
> |
```

**CODE:**

```
auc_score<-auc(roc_obj)
cat("AUC-ROC:",auc_score,"\n")
summary(tree_model)
```

OUTPUT:

```

> auc_score<-auc(roc_obj)
> cat("AUC-ROC:",auc_score,"\n")
AUC-ROC: 0.9921156
> summary(tree_model)
Call:
rpart(formula = class ~ ., data = train_data, method = "class")
n= 6499

      CP nsplit rel error     xerror       xstd
1 0.96957211      0 1.0000000 1.0000000 0.012770567
2 0.01901743      1 0.03042789 0.03042789 0.003082512
3 0.01000000      2 0.01141046 0.01141046 0.001896469

Variable importance
      odor      spore.print.color      gill.color
      25          19          15
stalk.surface.above.ring      ring.type stalk.surface.below.ring
      14          13          13

Node number 1: 6499 observations,    complexity param=0.9695721
predicted class=e expected loss=0.4854593  P(node) =1
  class counts: 3344 3155
  probabilities: 0.515 0.485
  left son=2 (3440 obs) right son=3 (3059 obs)
Primary splits:
  odor      splits as LRRRLRLRRR,    improve=3060.110, (0 missing)
  spore.print.color      splits as LRLRLRLRL,    improve=1757.867, (0 missing)
  gill.color      splits as RLRLLLLLRLLL,    improve=1247.353, (0 missing)
  stalk.surface.above.ring      splits as LRLL,    improve=1120.376, (0 missing)
  stalk.surface.below.ring      splits as LRLL,    improve=1054.844, (0 missing)

Surrogate splits:
  spore.print.color      splits as LRLLLLLRL,    agree=0.862, adj=0.707, (0 split)
  gill.color      splits as RLRLLLLLLRL,    agree=0.812, adj=0.600, (0 split)
  stalk.surface.above.ring      splits as LRLL,    agree=0.781, adj=0.535, (0 split)
  ring.type      splits as RLRLL,    agree=0.780, adj=0.533, (0 split)
  stalk.surface.below.ring      splits as LRLL,    agree=0.779, adj=0.530, (0 split)

Node number 2: 3440 observations,    complexity param=0.01901743
predicted class=e expected loss=0.02790698  P(node) =0.5293122
  class counts: 3344 96
  probabilities: 0.972 0.028
  left son=4 (3380 obs) right son=5 (60 obs)
Primary splits:
  spore.print.color      splits as LLLLLRLLLL,    improve=115.40870, (0 missing)
  gill.color      splits as -LLLLLRLRL,    improve= 38.01964, (0 missing)
  stalk.color.below.ring      splits as --LLLLLR,    improve= 34.19768, (0 missing)
  cap.color      splits as RLLLLRLLLL,    improve= 21.94909, (0 missing)
  ring.number      splits as -LR,    improve= 10.39475, (0 missing)
Surrogate splits:
  gill.color      splits as -LLLLLRLRL,    agree=0.988, adj=0.333, (0 split)

Node number 3: 3059 observations
predicted class=p expected loss=0  P(node) =0.4706878
  class counts: 0 3059
  probabilities: 0.000 1.000

Node number 4: 3380 observations
predicted class=e expected loss=0.01065089  P(node) =0.52008
  class counts: 3344 36
  probabilities: 0.989 0.011

Node number 5: 60 observations
predicted class=p expected loss=0  P(node) =0.00923219
  class counts: 0 60
  probabilities: 0.000 1.000

```

CODE:

```

pruned_tree<-prune(tree_model,cp=0.01)
summary(pruned_tree)

```

OUTPUT:

```

> pruned_tree<-prune(tree_model,cp=0.01)
> summary(pruned_tree)
Call:
rpart(formula = class ~ ., data = train_data, method = "class")
n= 6499

      CP nsplit rel error     xerror       xstd
1 0.96957211      0 1.00000000 1.00000000 0.012770567
2 0.01901743      1 0.03042789 0.03042789 0.003082512
3 0.01000000      2 0.01141046 0.01141046 0.001896469

Variable importance
          odor      spore.print.color      gill.color
        25                  19                  15
stalk.surface.above.ring      ring.type stalk.surface.below.ring
        14                  13                  13

Node number 1: 6499 observations,    complexity param=0.9695721
predicted class=e expected loss=0.4854593  P(node) =1
  class counts: 3344 3155
  probabilities: 0.515 0.485
left son=2 (3440 obs) right son=3 (3059 obs)
Primary splits:
  odor           splits as LRRRLRLRRR,   improve=3060.110, (0 missing)
  spore.print.color splits as LRLLLLRLRL,  improve=1757.867, (0 missing)
  gill.color       splits as RLRRLLLLLRLLL, improve=1247.353, (0 missing)
  stalk.surface.above.ring splits as LRLL,   improve=1120.376, (0 missing)
  stalk.surface.below.ring splits as LRLL,   improve=1054.844, (0 missing)

Surrogate splits:
  spore.print.color splits as LRLLLLLRL,   agree=0.862, adj=0.707, (0 split)
  gill.color       splits as RLRRLLLLLRL,  agree=0.812, adj=0.600, (0 split)
  stalk.surface.above.ring splits as LRLL,   agree=0.781, adj=0.535, (0 split)
  ring.type        splits as RLRRRL,      agree=0.780, adj=0.533, (0 split)
  stalk.surface.below.ring splits as LRLL,   agree=0.779, adj=0.530, (0 split)

Node number 2: 3440 observations,    complexity param=0.01901743
predicted class=e expected loss=0.02790698  P(node) =0.5293122
  class counts: 3344 96
  probabilities: 0.972 0.028
left son=4 (3380 obs) right son=5 (60 obs)
Primary splits:
  spore.print.color splits as LLLLLRLLLL,  improve=115.40870, (0 missing)
  gill.color       splits as -LLLLLRLRL,   improve= 38.01964, (0 missing)
  stalk.color.below.ring splits as --L--LRLR,  improve= 34.19768, (0 missing)
  cap.color        splits as RLLLLRLLLL,  improve= 21.94909, (0 missing)
  ring.number      splits as -LR,         improve= 10.39475, (0 missing)

Surrogate splits:
  gill.color splits as -LLLLLRLRL, agree=0.988, adj=0.333, (0 split)

Node number 3: 3059 observations
predicted class=p expected loss=0  P(node) =0.4706878
  class counts: 0 3059
  probabilities: 0.000 1.000

Node number 4: 3380 observations
predicted class=e expected loss=0.01065089  P(node) =0.52008
  class counts: 3344 36
  probabilities: 0.989 0.011

Node number 5: 60 observations
predicted class=p expected loss=0  P(node) =0.00923219
  class counts: 0 60
  probabilities: 0.000 1.000

```

CODE:

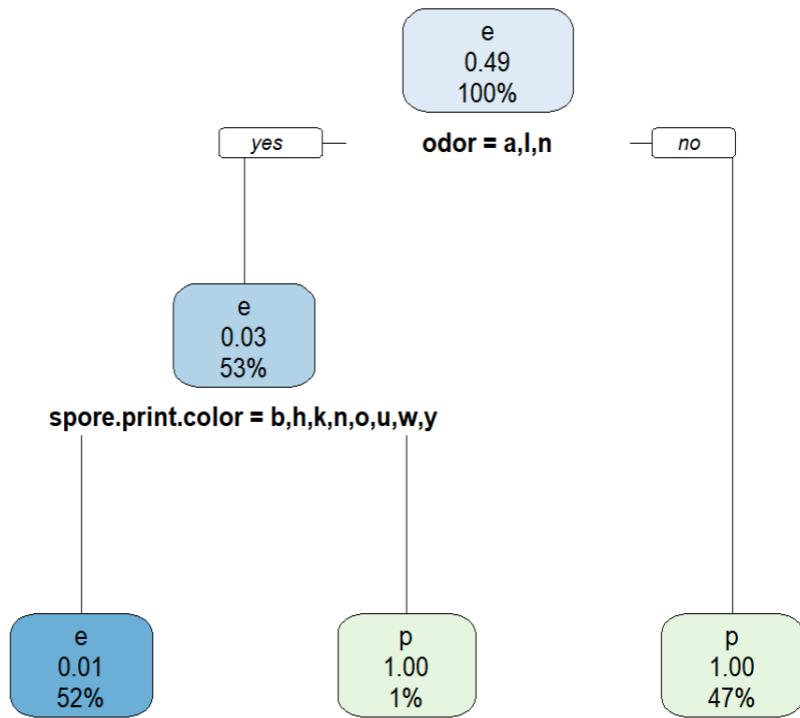
```
rpart.plot(pruned_tree)
```

OUTPUT:

```

> rpart.plot(pruned_tree)
>

```

**CODE:**

```
#make predictions on the test set using pruned_tree.
pruned_predictions<-predict(pruned_tree,newdata = test_data,type = "class")
pruned_cm<-confusionMatrix(pruned_predictions,test_data$class)
print(pruned_cm)
```

OUTPUT:

```
> pruned_predictions<-predict(pruned_tree,newdata = test_data,type = "class")
> pruned_cm<-confusionMatrix(pruned_predictions,test_data$class)
> print(pruned_cm)
Confusion Matrix and Statistics

          Reference
Prediction   e   p
      e 864  12
      p   0 749

    Accuracy : 0.9926
    95% CI  : (0.9871, 0.9962)
    No Information Rate : 0.5317
    P-value [Acc > NIR] : < 2.2e-16

    Kappa : 0.9852

McNemar's Test P-Value : 0.001496

    Sensitivity : 1.0000
    Specificity : 0.9842
    Pos Pred Value : 0.9863
    Neg Pred Value : 1.0000
    Prevalence : 0.5317
    Detection Rate : 0.5317
    Detection Prevalence : 0.5391
    Balanced Accuracy : 0.9921

    'Positive' Class : e
```

CODE:

```
pruned_accuracy<-sum(pruned_predictions==test_data$class)/length(test_data$class)
cat("Pruned Model Accuracy : ",pruned_accuracy,"\n")
pruned_cm<-confusionMatrix(pruned_predictions,test_data$class)
pruned_precision<-pruned_cm$byClass["Positive predictive value"]
pruned_recall<-pruned_cm$byClass["Sensitivity"]
pruned_f1_score<-pruned_cm$byClass["F1"]
cat("Pruned model precision: ",pruned_precision,"\n")
```

```
cat("Pruned model recall: ",pruned_recall,"\n")
cat("Pruned model f1-score: ",pruned_f1_score,"\n")
```

OUTPUT:

```
> pruned_accuracy<-sum(pruned_predictions==test_data$class)/length(test_data$class)
> cat("Pruned Model Accuracy : ",pruned_accuracy,"\n")
Pruned Model Accuracy : 0.9926154
> pruned_cm<-confusionMatrix(pruned_predictions,test_data$class)
> pruned_precision<-pruned_cm$byClass["Positive predictive value"]
> pruned_recall<-pruned_cm$byClass["Sensitivity"]
> pruned_f1_score<-pruned_cm$byClass["F1"]
> cat("Pruned model precision: ",pruned_precision,"\n")
Pruned model precision: NA
> cat("Pruned model recall: ",pruned_recall,"\n")
Pruned model recall: 1
> cat("Pruned model f1-score: ",pruned_f1_score,"\n")
Pruned model f1-score: 0.9931034
```

Findings:

Pruned model accuracy: 0.9926154

Pruned model precision: NA

Pruned model recall: 1

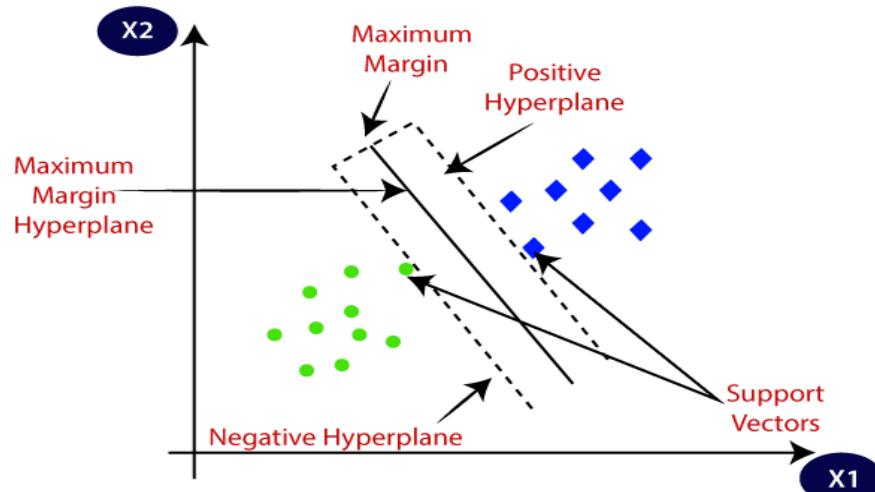
Pruned model f1-score: 0.9931034

EXPERIMENT NO:7**AIM:**

Write R program to implement Support Vector Machine(SVM) for Gene Expression Data.

DESCRIPTION:**Support Vector Machine:**

Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well it's best suited for classification. The main objective of the SVM algorithm is to find the optimal hyperplane in an N-dimensional space that can separate the data points in different classes in the feature space. The hyperplane tries that the margin between the closest points of different classes should be as maximum as possible. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

**Support Vector Machine Terminology:**

1. **Hyperplane:** Hyperplane is the decision boundary that is used to separate the data points of different classes in a feature space. In the case of linear classifications, it will be a linear equation i.e., $wx+b = 0$.
2. **Support Vectors:** Support vectors are the closest data points to the hyperplane, which makes a critical role in deciding the hyperplane and margin.
3. **Margin:** Margin is the distance between the support vector and hyperplane. The main objective of the support vector machine algorithm is to maximize the margin. The wider margin indicates better classification performance.
4. **Kernel:** Kernel is the mathematical function, which is used in SVM to map the original input data points into high-dimensional feature spaces, so, that the hyperplane can be easily found out even if the data points are not linearly separable in the original input space. Some of the common kernel functions are linear, polynomial, radial basis function (RBF), and sigmoid.

Hard Margin: The maximum-margin hyperplane or the hard margin hyperplane is a hyperplane that properly separates the data points of different categories without any misclassifications.

5. **Soft Margin:** When the data is not perfectly separable or contains outliers, SVM permits a soft margin technique. Each data point has a slack variable introduced by the soft-margin SVM formulation, which softens the strict margin requirement and permits certain

misclassifications or violations. It discovers a compromise between increasing the margin and reducing violations.

6. **C:** Margin maximisation and misclassification fines are balanced by the regularisation parameter C in SVM. The penalty for going over the margin or misclassifying data items is decided by it. A stricter penalty is imposed with a greater value of C, which results in a smaller margin and perhaps fewer misclassifications.
7. **Hinge Loss:** A typical loss function in SVMs is hinge loss. It punishes incorrect classifications or margin violations. The objective function in SVM is frequently formed by combining it with the regularisation term.
8. **Dual Problem:** A dual Problem of the optimisation problem that requires locating the Lagrange multipliers related to the support vectors can be used to solve SVM. The dual formulation enables the use of kernel tricks and more effective computing.

Mathematical intuition of Support Vector Machine:

Consider a binary classification problem with two classes, labelled as +1 and -1. We have a training dataset consisting of input feature vectors X and their corresponding class labels Y.

The equation for the linear hyperplane can be written as:

$$w^T x + b = 0$$

The vector W represents the normal vector to the hyperplane. i.e the direction perpendicular to the hyperplane. The parameter **b** in the equation represents the offset or distance of the hyperplane from the origin along the normal vector **w**.

The distance between a data point x_i and the decision boundary can be calculated as:

$$d_i = \frac{w^T x_i + b}{\|w\|}$$

where $\|w\|$ represents the Euclidean norm of the weight vector w. Euclidean norm of the normal vector W

For Linear SVM classifier:

$$\hat{y} = \begin{cases} 1 & : w^T x + b \geq 0 \\ 0 & : w^T x + b < 0 \end{cases}$$

Optimization:

- **For Hard margin linear SVM classifier:**

$$\begin{aligned} \underset{w,b}{\text{minimize}} \frac{1}{2} w^T w &= \underset{W,b}{\text{minimize}} \frac{1}{2} \|w\|^2 \\ \text{subject to } y_i(w^T x_i + b) &\geq 1 \text{ for } i = 1, 2, 3, \dots, m \end{aligned}$$

The target variable or label for the i^{th} training instance is denoted by the symbol t_i in this statement. And $t_i=-1$ for negative occurrences (when $y_i=0$) and $t_i=1$ for positive instances (when $y_i=1$) respectively. Because we require the decision boundary that satisfy the constraint: $t_i(w^T x_i + b) \geq 1$

- **For Soft margin linear SVM classifier:**

$$\begin{aligned} \underset{w,b}{\text{minimize}} \frac{1}{2} w^T w + C \sum_{i=1}^m \zeta_i \\ \text{subject to } y_i(w^T x_i + b) &\geq 1 - \zeta_i \text{ and } \zeta_i \geq 0 \text{ for } i = 1, 2, 3, \dots, m \end{aligned}$$

- **Dual Problem:** A dual Problem of the optimisation problem that requires locating the Lagrange multipliers related to the support vectors can be used to solve SVM. The optimal Lagrange multipliers $\alpha(i)$ that maximize the following dual objective function

$$\underset{\alpha}{\text{maximize}} : \frac{1}{2} \sum_{i \rightarrow m} \sum_{j \rightarrow m} \alpha_i \alpha_j t_i t_j K(x_i, x_j) - \sum_{i \rightarrow m} \alpha_i$$

where,

- α_i is the Lagrange multiplier associated with the i^{th} training sample.
- $K(x_i, x_j)$ is the kernel function that computes the similarity between two samples x_i and x_j . It allows SVM to handle nonlinear classification problems by implicitly mapping the samples into a higher-dimensional feature space.

- The term $\sum \alpha_i$ represents the sum of all Lagrange multipliers.

The SVM decision boundary can be described in terms of these optimal Lagrange multipliers and the support vectors once the dual issue has been solved and the optimal Lagrange multipliers have been discovered. The training samples that have $i > 0$ are the support vectors, while the decision boundary is supplied by:

$$w = \sum_{i \rightarrow m} \alpha_i t_i K(x_i, x) + b$$

$$t_i(w^T x_i - b) = 1 \iff b = w^T x_i - t_i$$

Types of Support Vector Machine:

Based on the nature of the decision boundary, Support Vector Machines (SVM) can be divided into two main parts:

- Linear SVM:** Linear SVMs use a linear decision boundary to separate the data points of different classes. When the data can be precisely linearly separated, linear SVMs are very suitable. This means that a single straight line (in 2D) or a hyperplane (in higher dimensions) can entirely divide the data points into their respective classes. A hyperplane that maximizes the margin between the classes is the decision boundary.
- Non-Linear SVM:** Non-Linear SVM can be used to classify data when it cannot be separated into two classes by a straight line (in the case of 2D). By using kernel functions, nonlinear SVMs can handle nonlinearly separable data. The original input data is transformed by these kernel functions into a higher-dimensional feature space, where the data points can be linearly separated. A linear SVM is used to locate a nonlinear decision boundary in this modified space.

Popular kernel functions in SVM:

The SVM kernel is a function that takes low-dimensional input space and transforms it into higher-dimensional space, i.e., it converts non separable problems to separable problems.

It is mostly useful in non-linear separation problems. Simply put the kernel, does some extremely complex data transformations and then finds out the process to separate the data based on the labels or outputs defined.

$$\text{Linear : } K(w, b) = w^T x + b$$

$$\text{Polynomial : } K(w, x) = (\gamma w^T x + b)^N$$

$$\text{Gaussian RBF: } K(w, x) = \exp(-\gamma ||x_i - x_j||^n)$$

$$\text{Sigmoid : } K(x_i, x_j) = \tanh(\alpha x_i^T x_j + b)$$

CODE:

```
# Load necessary packages
library(ISLR2)

library(pROC)

library(ggplot2)

library(e1071)

library(plotmo)

library(plotly)

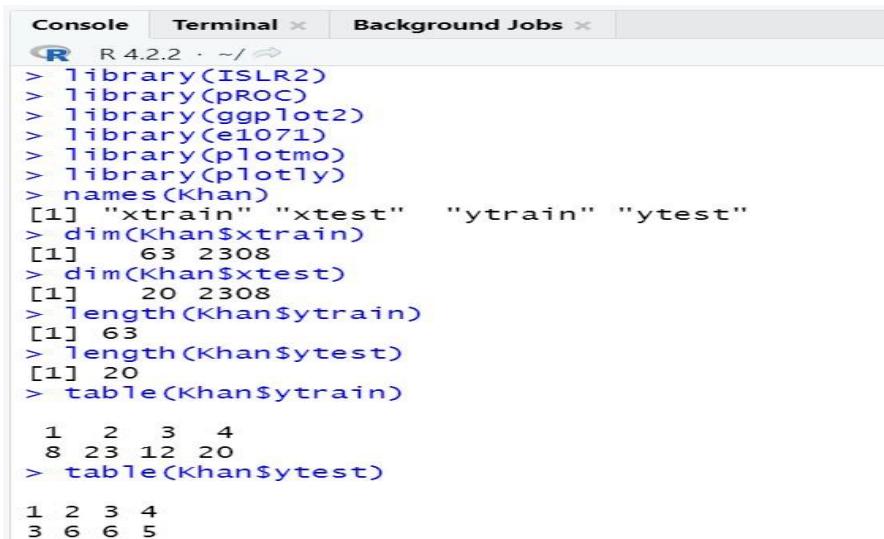
# Display column names of the Khan dataset

names(Khan)
```

```
# Check dimensions of training and testing sets
dim(Khan$xtrain)
dim(Khan$xtest)

# Check the length of training and testing labels
length(Khan$ytrain)
length(Khan$ytest)

# Display class distribution in training and testing labels
table(Khan$ytrain)
table(Khan$ytest)
```

OUTPUT:


The screenshot shows an R console window with three tabs: 'Console', 'Terminal', and 'Background Jobs'. The 'Console' tab is active, displaying the following R session:

```
R 4.2.2 · ~/cloud
> library(ISLR2)
> library(pROC)
> library(ggplot2)
> library(e1071)
> library(plotmo)
> library(plotly)
> names(Khan)
[1] "xtrain" "xtest"  "ytrain" "ytest"
> dim(Khan$xtrain)
[1] 63 2308
> dim(Khan$xtest)
[1] 20 2308
> length(Khan$ytrain)
[1] 63
> length(Khan$ytest)
[1] 20
> table(Khan$ytrain)
   1  2  3  4 
  8 23 12 20 
> table(Khan$ytest)
 1  2  3  4 
 3  6  6  5 
```

CODE:

```
# Create a dataframe 'dat' with training features and labels
dat <- data.frame(x = Khan$xtrain, y = as.factor(Khan$ytrain))

# Train an SVM model with linear kernel
out <- svm(y ~ ., data = dat, kernel = "linear", cost = 10)

# Display summary of the trained SVM model
summary(out)

# Compare the fitted values with actual labels in the training data
table(out$fitted, dat$y)

# Create a dataframe 'dat.te' with testing features and labels
dat.te <- data.frame(x = Khan$xtest, y = as.factor(Khan$ytest))

# Predict using the trained SVM model on the testing data
```

```

pred.te <- predict(out, newdata = dat.te)

# Compare the predicted values with actual labels in the testing data

table(pred.te, dat.te$y)

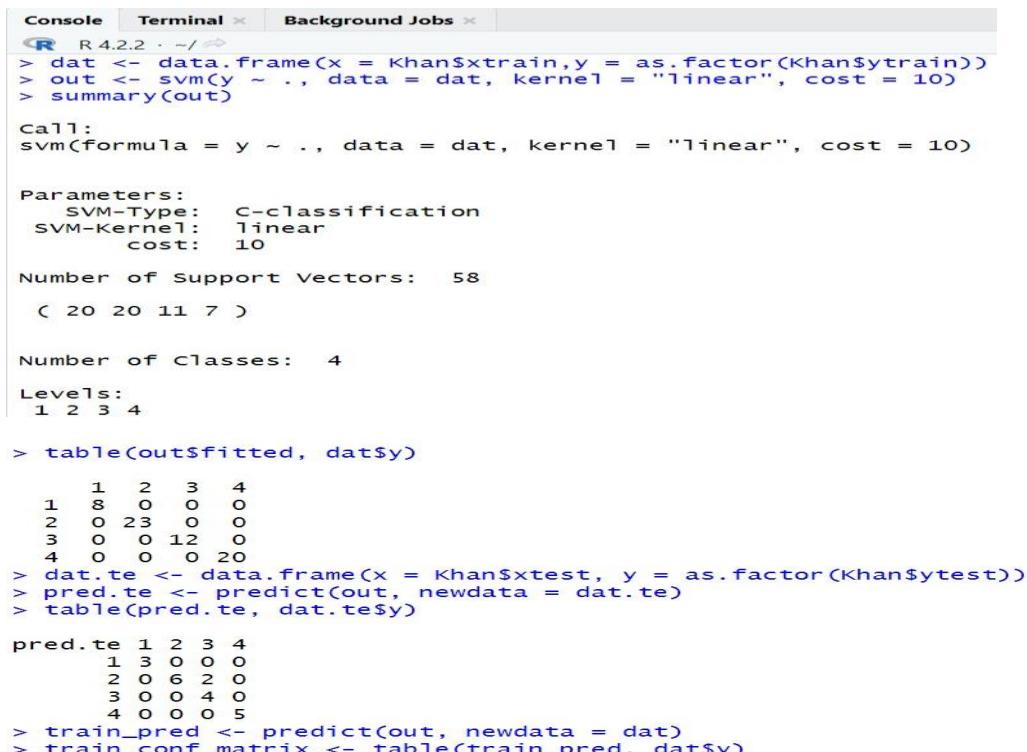
# Confusion matrix for training data

train_pred <- predict(out, newdata = dat)

train_conf_matrix <- table(train_pred, dat$y)

```

OUTPUT :



```

Console Terminal × Background Jobs ×
R 4.2.2 · ~/ ◀
> dat <- data.frame(x = Khan$xtrain, y = as.factor(Khan$ytrain))
> out <- svm(y ~ ., data = dat, kernel = "linear", cost = 10)
> summary(out)

Call:
svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10)

Parameters:
  SVM-Type: C-classification
  SVM-Kernel: linear
  cost: 10

Number of Support Vectors: 58
( 20 20 11 7 )

Number of Classes: 4
Levels:
 1 2 3 4

> table(out$fitted, dat$y)

   1   2   3   4
1  8   0   0   0
2  0  23   0   0
3  0   0  12   0
4  0   0   0  20

> dat.te <- data.frame(x = Khan$xtest, y = as.factor(Khan$ytest))
> pred.te <- predict(out, newdata = dat.te)
> table(pred.te, dat.te$y)

pred.te 1 2 3 4
1 3 0 0 0
2 0 6 2 0
3 0 0 4 0
4 0 0 0 5

> train_pred <- predict(out, newdata = dat)
> train_conf_matrix <- table(train_pred, dat$y)

```

CODE:

```

# Convert the confusion matrix to a dataframe

train_conf_df <- as.data.frame(as.table(train_conf_matrix))

# Rename the columns for better readability

colnames(train_conf_df) <- c("Predicted", "Actual", "Frequency")

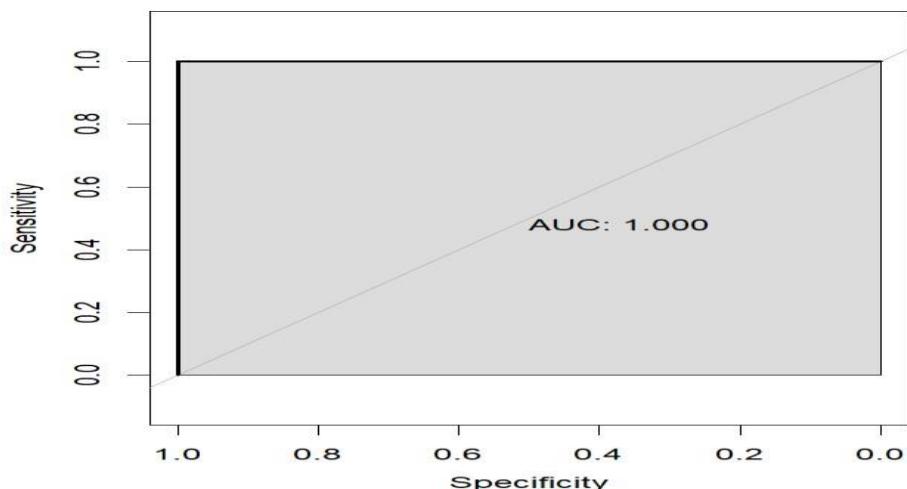
# Create the heatmap

ggplot(train_conf_df, aes(x = Predicted, y = Actual, fill = Frequency)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "blue") +
  abs(x = "Predicted Class", y = "Actual Class", fill = "Frequency") +
  theme_minimal()

# Convert factor levels to numeric (1 or 2)

numeric_pred <- as.numeric(train_pred)

```

**CODE:**

```
for (i in 2:length(train_roc$rocs)) { lines(train_roc$rocs[[i]], col = i, print.auc =
TRUE, auc.polygon = TRUE)}

# Add a legend
legend("bottomright", legend = levels(dat$y), col = 1:length(train_roc$rocs), lwd = 2)

# Calculate performance metrics for training data

train_pred <- predict(out, newdata = dat)

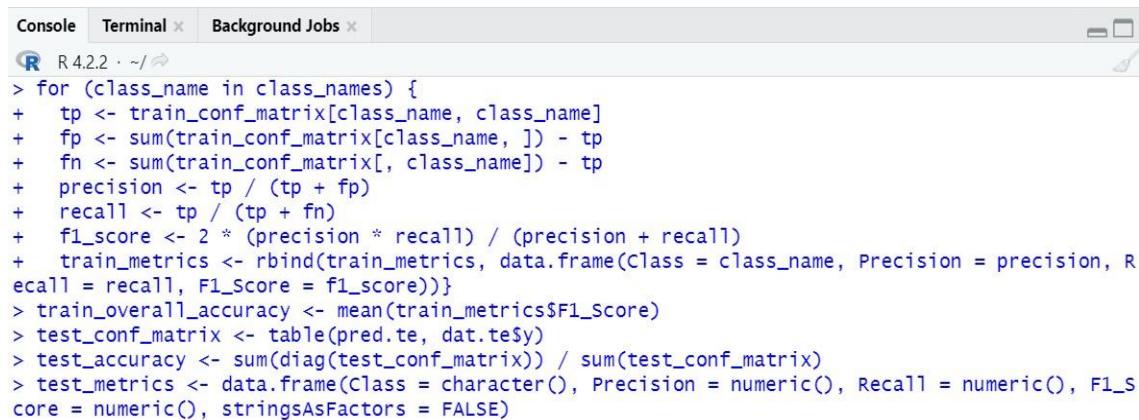
train_conf_matrix <- table(train_pred, dat$y)
# Calculate accuracy for training data
train_accuracy <- sum(diag(train_conf_matrix)) / sum(train_conf_matrix)
# Calculate precision, recall, and F1-score for each class in training data
class_names <- levels(dat$y)
train_metrics <- data.frame(Class = character(), Precision = numeric(), Recall = numeric(),
F1_Score = numeric(), stringsAsFactors = FALSE)
```

OUTPUT:

Console Terminal × Background Jobs ×

R 4.2.2 · ~/

```
> for (i in 2:length(train_roc$rocs)) {
+   lines(train_roc$rocs[[i]], col = i, print.auc = TRUE, auc.polygon = TRUE)
+ }
> legend("bottomright", legend = levels(dat$y), col = 1:length(train_roc$rocs), lwd = 2)
> train_pred <- predict(out, newdata = dat)
There were 20 warnings (use warnings() to see them)
> train_conf_matrix <- table(train_pred, dat$y)
> train_accuracy <- sum(diag(train_conf_matrix)) / sum(train_conf_matrix)
> class_names <- levels(dat$y)
> train_metrics <- data.frame(Class = character(), Precision = numeric(), Recall = numeric(), F1_
Score = numeric(), stringsAsFactors = FALSE)
```

OUTPUT:


R 4.2.2 · ~/

```
> for (class_name in class_names) {
+   tp <- train_conf_matrix[class_name, class_name]
+   fp <- sum(train_conf_matrix[, class_name]) - tp
+   fn <- sum(train_conf_matrix[, , class_name]) - tp
+   precision <- tp / (tp + fp)
+   recall <- tp / (tp + fn)
+   f1_score <- 2 * (precision * recall) / (precision + recall)
+   train_metrics <- rbind(train_metrics, data.frame(Class = class_name, Precision = precision, Recall = recall, F1_Score = f1_score))
> train_overall_accuracy <- mean(train_metrics$F1_Score)
> test_conf_matrix <- table(pred.te, dat.te$y)
> test_accuracy <- sum(diag(test_conf_matrix)) / sum(test_conf_matrix)
> test_metrics <- data.frame(Class = character(), Precision = numeric(), Recall = numeric(), F1_Score = numeric(), stringsAsFactors = FALSE)
```

CODE:

```
for (class_name in class_names) {

  tp <- test_conf_matrix[class_name, class_name]

  fp <- sum(test_conf_matrix[, class_name]) - tp

  fn <- sum(test_conf_matrix[, , class_name]) - tp

  precision <- tp / (tp + fp)

  recall <- tp / (tp + fn)

  f1_score <- 2 * (precision * recall) / (precision + recall)

  test_metrics <- rbind(test_metrics, data.frame(Class = class_name, Precision = precision, Recall = recall, F1_Score = f1_score))
}

# Calculate overall classification accuracy for testing data

test_overall_accuracy <- mean(test_metrics$F1_Score)

# Print performance metrics for both training and testing data

cat("Performance Metrics for Training Data:\n")

cat("Overall Accuracy:", train_accuracy, "\n") print(train_metrics)

cat("Overall Classification Accuracy (F1-Score):", train_overall_accuracy, "\n\n")

cat("Performance Metrics for Testing Data:\n")

cat("Overall Accuracy:", test_accuracy, "\n")

print(test_metrics)

cat("Overall Classification Accuracy (F1-Score):", test_overall_accuracy, "\n")
```

OUTPUT:

```

Console Terminal × Background Jobs ×
R 4.2.2 · ~/ ...
> for (class_name in class_names) {
+   tp <- test_conf_matrix[class_name, class_name]
+   fp <- sum(test_conf_matrix[, class_name]) - tp
+   fn <- sum(test_conf_matrix[, class_name]) - tp
+   precision <- tp / (tp + fp)
+   recall <- tp / (tp + fn)
+   f1_score <- 2 * (precision * recall) / (precision + recall)
+   test_metrics <- rbind(test_metrics, data.frame(Class = class_name, Precision = precision, Recall = recall, F1_Score = f1_score))
> test_overall_accuracy <- mean(test_metrics$F1_Score)
> cat("Performance Metrics for Training Data:\n")
Performance Metrics for Training Data:
> cat("Overall Accuracy:", train_accuracy, "\n")
Overall Accuracy: 1
> print(train_metrics)
  Class Precision Recall F1_Score
1     1       1      1      1
2     2       1      1      1
3     3       1      1      1
4     4       1      1      1
> cat("Overall classification Accuracy (F1-Score):", train_overall_accuracy, "\n\n")
Overall Classification Accuracy (F1-Score): 1

> cat("Performance Metrics for Testing Data:\n")
Performance Metrics for Testing Data:

> cat("Overall Accuracy:", test_accuracy, "\n")
Overall Accuracy: 0.9
> print(test_metrics)
  Class Precision Recall F1_Score
1     1       1.00 1.0000000 1.0000000
2     2       0.75 1.0000000 0.8571429
3     3       1.00 0.6666667 0.8000000
4     4       1.00 1.0000000 1.0000000
> cat("Overall classification Accuracy (F1-Score):", test_overall_accuracy, "\n")
Overall Classification Accuracy (F1-Score): 0.9142857

```

CODE:

```

# Create a dataframe for training metrics

train_metrics_df <- data.frame(
  Class = train_metrics$Class,
  Precision = train_metrics$Precision,
  Recall = train_metrics$Recall,
  F1_Score = train_metrics$F1_Score)

# Create a bar plot for training metrics

ggplot(train_metrics_df, aes(x = Class, y = F1_Score, fill = Class)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(
    title = "Performance Metrics for Training Data",
    y = "F1-Score",
    x = "Class"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Calculate multiclass ROC curve and AUC for training data

train_roc <- multiclass.roc(dat$y, numeric_pred)

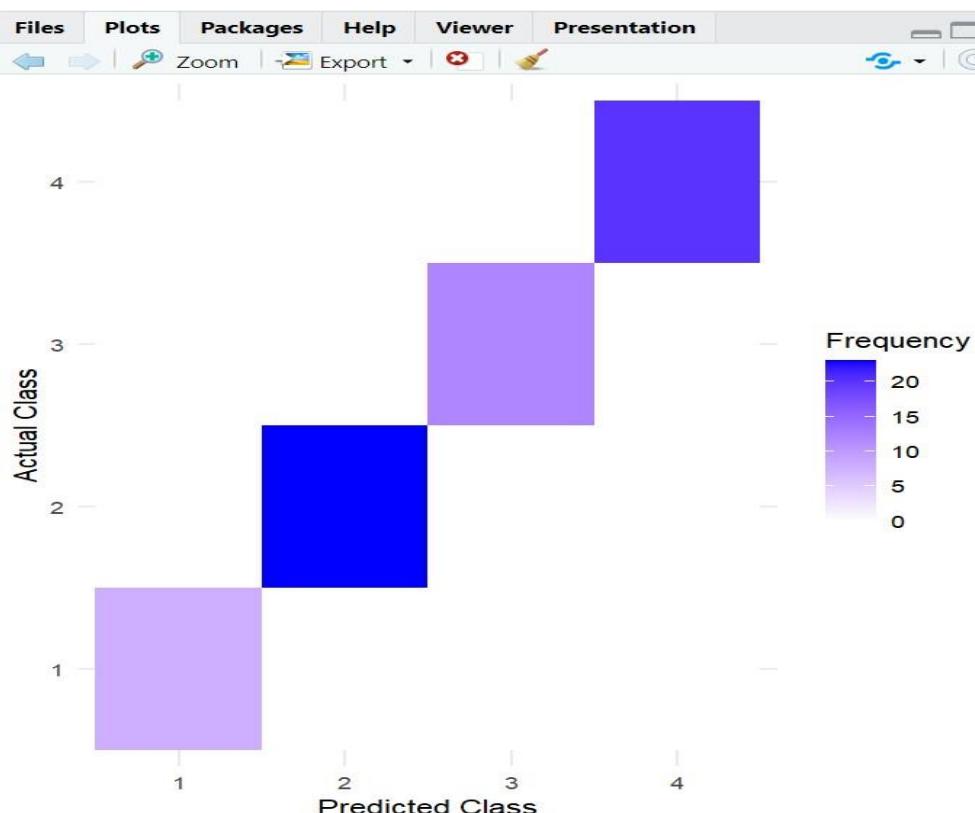
# Plot the multiclass ROC curves for each class

plot(train_roc$rocs[[1]], col = 1, print.auc = TRUE, auc.polygon =
  TRUE)

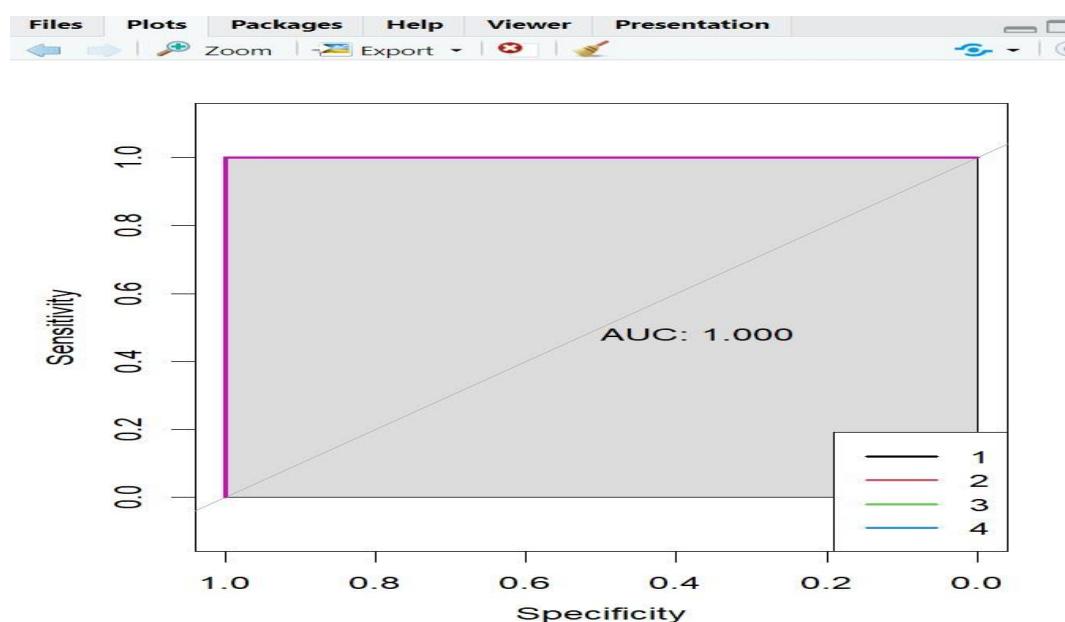
```

OUTPUT:

```
> train_conf_df <- as.data.frame(as.table(train_conf_matrix))
> colnames(train_conf_df) <- c("Predicted", "Actual", "Frequency")
> ggplot(train_conf_df, aes(x = Predicted, y = Actual, fill = Frequency)) +
+   geom_tile() +
+   scale_fill_gradient(low = "white", high = "blue") +
+   labs(x = "Predicted Class", y = "Actual Class", fill = "Frequency") +
+   theme_minimal()
```



```
> numeric_pred <- as.numeric(train_pred)
> train_roc <- multiclass.roc(dat$y, numeric_pred)
Setting direction: controls < cases
> plot(train_roc$rocs[[1]], col = 1, print.auc = TRUE, auc.polygon = TRUE)
```

**CODE:**

```

for (class_name in class_names) {
  tp <- train_conf_matrix[class_name, class_name]
  fp <- sum(train_conf_matrix[, class_name]) - tp
  fn <- sum(train_conf_matrix[, , class_name]) - tp
  precision <- tp / (tp + fp)  recall <- tp / (tp + fn)
  f1_score <- 2 * (precision * recall) / (precision + recall)
  train_metrics <- rbind(train_metrics, data.frame(Class = class_name, Precision = precision, Recall = recall,
  F1_Score = f1_score))
}
# Calculate overall classification accuracy for training data
train_overall_accuracy <- mean(train_metrics$F1_Score)
# Calculate performance metrics for testing data
test_conf_matrix <- table(pred.te, dat.te$y)
# Calculate accuracy for testing data
test_accuracy <- sum(diag(test_conf_matrix)) / sum(test_conf_matrix)
# Calculate precision, recall, and F1-score for each class in testing data
test_metrics <- data.frame(Class = character(), Precision = numeric(), Recall = numeric(), F1_Score =
  numeric(), stringsAsFactors = FALSE)

```

OUTPUT:**CODE:**

```
# Create a dataframe for testing metrics
test_metrics_df <- data.frame(Class = test_metrics$Class,
                               Precision = test_metrics$Precision,
                               Recall = test_metrics$Recall,
                               F1_Score = test_metrics$F1_Score)

# Create a bar plot for testing metrics
ggplot(test_metrics_df, aes(x = Class, y = F1_Score, fill = Class)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(
    title = "Performance Metrics for Testing Data",
    y = "F1-Score",
    x = "Class")
```

```
) +  
theme_minimal() +  
  
theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

OUTPUT:

EXPERIMENT NO: 8

AIM: To Implement Principal Component Analysis on the US Arrests data set.

DESCRIPTION:

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the **Principal Components**. It is one of the popular tools that is used for exploratory data analysis and predictive modeling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.

PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are *image processing, movie recommendation system, optimizing the power allocation in various communication channels*. It is a feature extraction technique, so it contains the important variables and drops the least important variable.

The PCA algorithm is based on some mathematical concepts such as:

- Variance and Covariance
 - Eigenvalues and Eigen factors

Some common terms used in PCA algorithm:

- **Dimensionality:** It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset.
- **Correlation:** It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
- **Orthogonal:** It defines that variables are not correlated to each other, and hence the correlation between the pair of variables is zero.
- **Eigenvectors:** If there is a square matrix M, and a non-zero vector v is given. Then v will be eigenvector if Av is the scalar multiple of v.
- **Covariance Matrix:** A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

Principal Components in PCA:

- As described above, the transformed new features or the output of PCA are the Principal Components. The number of these PCs are either equal to or less than the original features present in the dataset. Some properties of these principal components are given below:□
- The principal component must be the linear combination of the original features.□
- These components are orthogonal, i.e., the correlation between a pair of variables is zero.□
- The importance of each component decreases when going to 1 to n, it means the 1 PC has the most importance, and n PC will have the least importance.□

Steps for PCA algorithm:

1. Getting the dataset

Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.

2. Representing data into a structure

Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.

3. Standardizing the data

In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance.

If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.

4. Calculating the Covariance of Z

To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.

5. Calculating the Eigen Values and Eigen Vectors

Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.

6. Sorting the Eigen Vectors

In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P*.

7. Calculating the new features Or Principal Components

Here we will calculate the new features. To do this, we will multiply the P* matrix to the Z. In the resultant matrix Z*, each observation is the linear combination of original features. Each column of the Z* matrix is independent of each other.

8. Remove less or unimportant features from the new dataset.

The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

```
library(ggplot2)
library(gridExtra)
library(caret)
```

```
library(ISLR2)
```

```
#LOAD THE USArrests dataset
data("USArrests")
```

```
#STANDARDISE THE DATA
scaled_data<-scale(USArrests)
```

```
#perform PCA
pca_result<-prcomp(scaled_data,center=TRUE,scale.=TRUE)
```

```
#summary the PCA result
summary(pca_result)
```

```
#variable explained by each principle component
variance_explained <- pca_result$sdev^2 / sum(pca_result$sdev^2) * 100

#print the variance explained by each pc cat("variance
explained by each principal component:\n")
print(variance_explained)
```

OUTPUT:

Importance of components:

	PC1	PC2	PC3	PC4
Standard deviation	1.5749	0.9949	0.59713	0.41645
Proportion of Variance	0.6201	0.2474	0.08914	0.04336
Cumulative Proportion	0.6201	0.8675	0.95664	1.00000

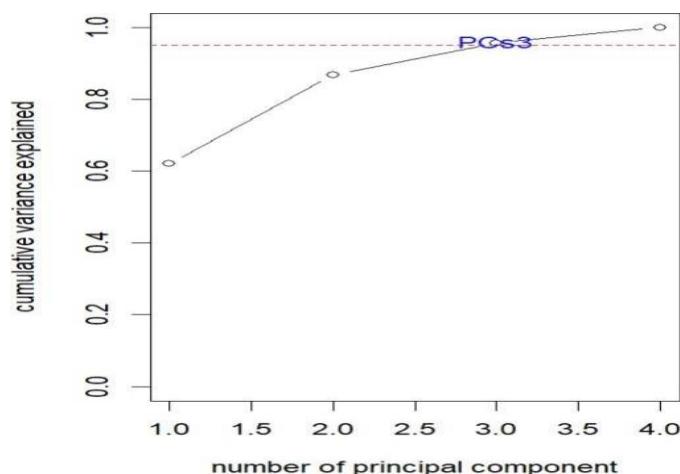
```
> #variable explained by each principle component
> variance_explained <- pca_result$sdev^2 / sum(pca_result$sdev^2) * 100
> #print the variance explained by each pc
> cat("variance explained by each principal component:\n")
variance explained by each principal component:
> print(variance_explained)
[1] 62.006039 24.744129  8.914080  4.335752
`-
```

```
#scree plot to visualize the variance explained by each pc
plot(variance_explained,type = "b",xlab="principal component",ylab = "variance explained(%)")
```

```
#calculate the cumulative proportion of variance explained
variance_explained <- pca_result$sdev^2 / sum(pca_result$sdev^2)
cumulative_variance_explained <- cumsum(variance_explained)
plot(cumulative_variance_explained, type = "b",
      xlab = "number of principal component",
      ylab = "cumulative variance explained",ylim = c(0,1))
```

```
#add a horizontal line at 0.95 to visualize the 95% threshold
abline(h=0.95,col = "red",lty=2)
```

```
#find the number of pcs needed to explain atleast 95% of the variance
num_pcs_to_reach_95_percent <- which(cumulative_variance_explained
>=0.95)[1] text(num_pcs_to_reach_95_percent,0.96, labels =
paste0("PCs",num_pcs_to_reach_95_percent), col = "blue")
```

OUTPUT:**CODE:**

```
#define the desired threshold (e.g,90%)
desired_threshold <- 0.90

#find the number of PCs needed to reach or exceed the desired threshold
num_pcs_to_reach_threshold <- which(cumulative_variance_explained >= desired_threshold)[1]

#extract the top "n" principal components
top_n_pcs <- pca_result$x[,1:num_pcs_to_reach_threshold]

#print the result you can now use top_n_pcs
cat("Number of PCs to convert at least", (desired_threshold * 100),
    "% variance:", num_pcs_to_reach_threshold, "\n")

#Retain the first 4 principal components
top_4_pcs <- pca_result$x[, 1:4]

#create a data frame with the top 4 PCs and the mpg variable
data_with_pcs <- data.frame(top_4_pcs, mpg = USArrests$Murder)

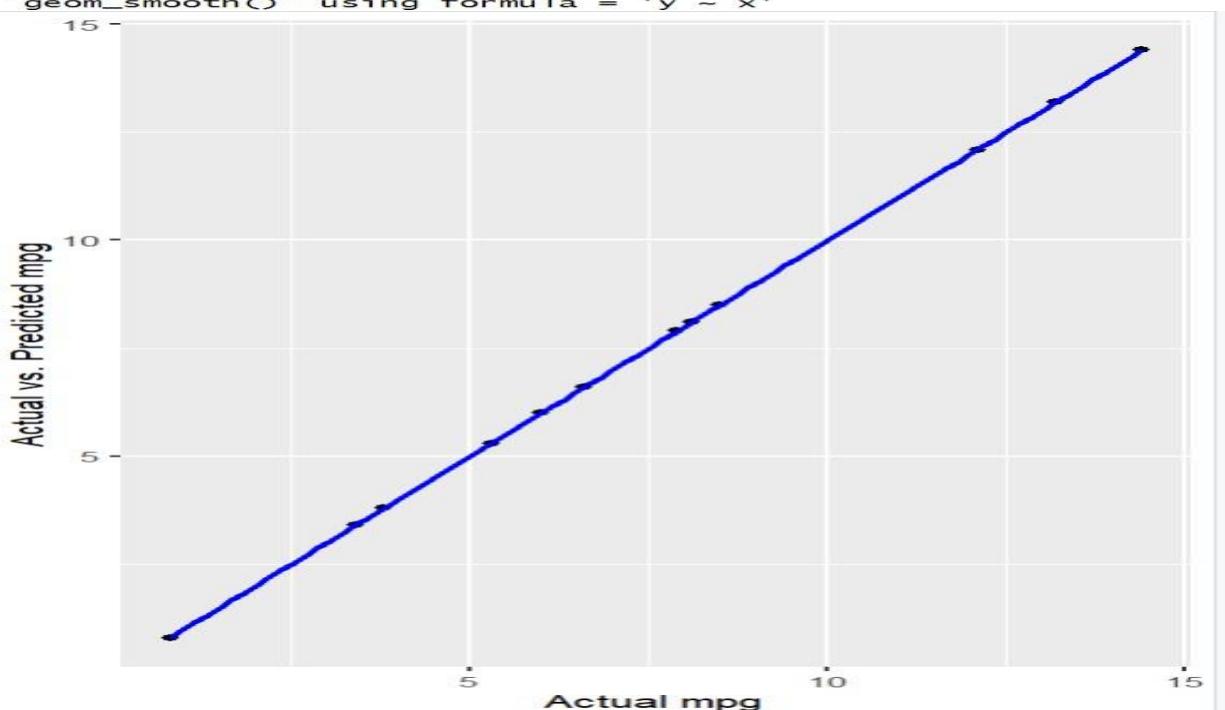
#split the data into a training set (75%) and a testing set(25%)
set.seed(123)
train_index <- createDataPartition(data_with_pcs$mpg, p = 0.75, list =
FALSE) train_data <- data_with_pcs[train_index, ] test_data <-
data_with_pcs[-train_index, ] model <- lm(mpg ~ ., data = train_data)
predictions <- predict(model, newdata = test_data) rmse <-
sqrt(mean((test_data$mpg - predictions)^2)) cat("Root Mean Squared Error
(RMSE):", rmse, "\n") result_data <- data.frame(Actual = test_data$mpg,
Predicted = predictions)

ggplot(result_data, aes(x = Actual, y = Predicted)) +
geom_point() +
geom_smooth(method = "lm", se = FALSE, color = "blue") +
labs(
x = "Actual mpg",
y = "Actual vs. Predicted mpg")
```

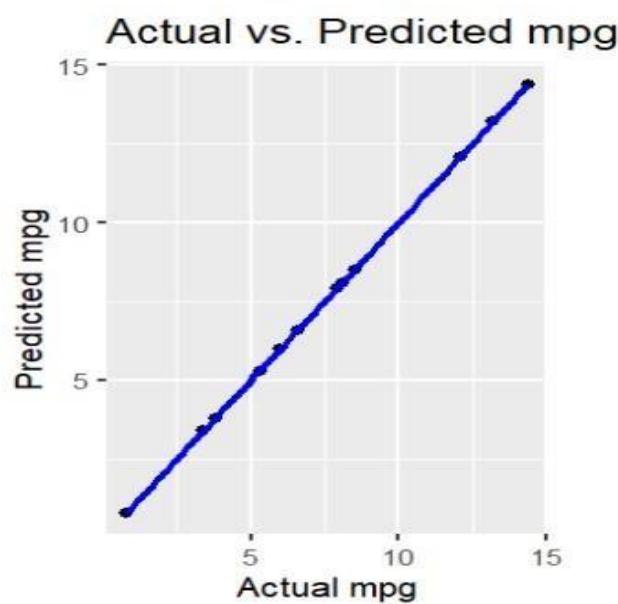
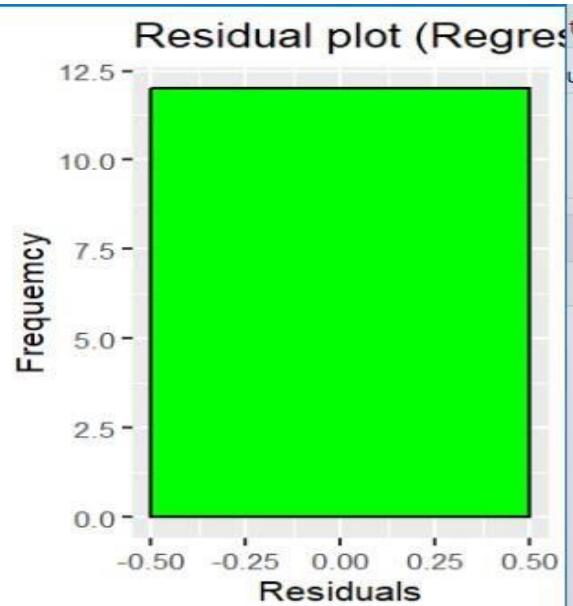
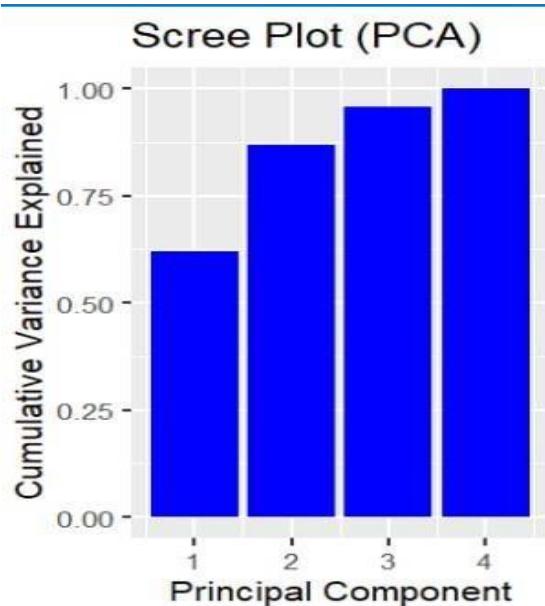
```
)  
residuals <- test_data$mpg - predictions screen_plot <-  
ggplot(NULL, aes(x = 1:ncol(top_4_pcs), y =  
cumulative_variance_explained)) +  
geom_bar(stat = "identity", fill = "blue") +  
labs(  
x = "Principal Component", y =  
"Cumulative Variance Explained",  
title = "Scree Plot (PCA)"  
)  
residuals_plot <- ggplot(NULL, aes(x = residuals)) +  
geom_histogram(binwidth = 1, fill = "green", color = "black") +  
labs(  
x = "Residuals", y = "Frequency",  
title = "Residual plot  
(Regression)"  
)  
actual_vs_predicted_plot <- ggplot(result_data, aes(x = Actual,  
y = Predicted)) + geom_point() +  
geom_smooth(method = "lm", se = FALSE, color = "blue") + labs(  
x = "Actual mpg", y = "Predicted  
mpg", title = "Actual vs.  
Predicted mpg"  
)
```

OUTPUT:

```
> #create a data frame with the top 4 PCs and the mpg variable
> data_with_pcs <- data.frame(top_4_pcs,mpg = USArests$Murder)
>
> #split the data into a training set (75%) nd a testing set(25%)
> set.seed(123)
> train_index <- createDataPartition(data_with_pcs$mpg, p = 0.75,
+                                     list = FALSE)
> train_data <- data_with_pcs[train_index, ]
> test_data <- data_with_pcs[-train_index, ]
> model <- lm(mpg ~ ., data = train_data)
> predictions <- predict(model, newdata = test_data)
> rmse <- sqrt(mean((test_data$mpg - predictions)^2))
> cat("Root Mean Squared Error (RMSE):", rmse , "\n")
Root Mean Squared Error (RMSE): 2.187121e-15
> result_data <- data.frame(Actual = test_data$mpg,
+                           Predicted = predictions)
>
> ggplot(result_data, aes(x = Actual, y = Predicted)) +
+   geom_point() +
+   geom_smooth(method = "lm", se = FALSE, color = "blue") +
+   labs(
+     x = "Actual mpg",
+     y = "Actual vs. Predicted mpg"
+   )
`geom_smooth()` using formula = 'y ~ x'
```

**CODE:**

```
summary_plot <- grid.arrange(screen_plot, residuals_plot,
                             actual_vs_predicted_plot, ncol = 2)
print(summary_plot)
```

OUTPUT:

EXPERIMENT NO:9**AIM: To Perform K-Means Clustering on NC160 Dataset.**

DESCRIPTION: K-means clustering is a technique in which we place each observation in a dataset into one of K clusters. The end goal is to have K clusters in which the observations within each cluster are quite similar to each other while the observations in different clusters are quite different from each other.

In practice, we use the following steps to perform K-means clustering:

1. Choose a value for K.

- First, we must decide how many clusters we'd like to identify in the data. Often we have to simply test several different values for K and analyze the results to see which number of clusters seems to make the most sense for a given problem.

2. Randomly assign each observation to an initial cluster, from 1 to K.

3. Perform the following procedure until the cluster assignments stop changing.

- For each of the K clusters, compute the cluster centroid. This is simply the vector of the p feature means for the observations in the kth cluster.

- Assign each observation to the cluster whose centroid is closest. Here, closest is defined using Euclidean distance.

K-Means Clustering in R.

The following tutorial provides a step-by-step example of how to perform k-means clustering in R.

Step 1: Load the Necessary Packages

First, we'll load two packages that contain several useful functions for k-means clustering in R

Step 2: Load and Prep the Data

For this example we'll use the USArests dataset built into R, which contains the number of arrests per 100,000 residents in each U.S. state in 1973 for Murder, Assault, and Rape along with the percentage of the population in each state living in urban areas, UrbanPop.

The following code shows how to do the following:

- Load the USArests dataset
- Remove any rows with missing values
- Scale each variable in the dataset to have a mean of 0 and a standard deviation of 1

Step 3: Find the Optimal Number of Clusters

To perform k-means clustering in R we can use the built-in kmeans() function, which uses the following syntax:

```
kmeans(data, centers, nstart)
```

where:

- data: Name of the dataset.
 - centers: The number of clusters, denoted k.
 - nstart: The number of initial configurations. Because it's possible that different initial starting clusters can lead to different results, it's recommended to use several different initial configurations. The k-means algorithm will find the initial configurations that lead to the smallest within-cluster variation.
- Since we don't know beforehand how many clusters is optimal, we'll create two different plots that can help us decide:

1. Number of Clusters vs. the Total Within Sum of Squares

First, we'll use the fviz_nbclust() function to create a plot of the number of clusters vs. the total within sum of squares.

Typically when we create this type of plot we look for an “elbow” where the sum of squares begins to “bend” or level off. This is typically the optimal number of clusters.

For this plot it appears that there is a bit of an elbow or “bend” at k = 4 clusters.

2. Number of Clusters vs. Gap Statistic

Another way to determine the optimal number of clusters is to use a metric known as the gap statistic, which compares the total intra-cluster variation for different values of k with their expected values for a distribution with no clustering.

We can calculate the gap statistic for each number of clusters using the `clusGap()` function from the `cluster` package along with a plot of clusters vs. gap statistic using the `fviz_gap_stat()` function

From the plot we can see that gap statistic is highest at $k = 4$ clusters, which matches the elbow method we used earlier.

Step 4: Perform K-Means Clustering with Optimal K

Lastly, we can perform k-means clustering on the dataset using the optimal value for k of 4.

From the results we can see that:

- 16 states were assigned to the first cluster
- 13 states were assigned to the second cluster
- 13 states were assigned to the third cluster
- 8 states were assigned to the fourth cluster.

We can visualize the clusters on a scatterplot that displays the first two principal components on the axes using the `fviz_cluster()` function

We can also use the `aggregate()` function to find the mean of the variables in each cluster.

We interpret this output is as follows:

- The mean number of murders per 100,000 citizens among the states in cluster 1 is 3.6.
- The mean number of assaults per 100,000 citizens among the states in cluster 1 is 78.5.
- The mean percentage of residents living in an urban area among the states in cluster 1 is 52.1%.
- The mean number of rapes per 100,000 citizens among the states in cluster 1 is 12.2.

And so on.

We can also append the cluster assignments of each state back to the original dataset.

CODE:

#Load necessary packages

```
library(factoextra)
library(cluster)
```

#load data

```
df <- USArrests
```

#remove rows with missing values

```
df <- na.omit(df)
```

#scale each variable to have a mean of 0 and sd of 1

```
df <- scale(df)
```

#view first six rows of dataset

```
head(df)
```

#Number of Clusters vs. the Total Within Sum of Squares

```
fviz_nbclust(df, kmeans, method = "wss")
```

#calculate gap statistic based on number of clusters

```
gap_stat <- clusGap(df,FUN = kmeans,
                     nstart = 25,
                     K.max = 10,
                     B = 50)
```

OUTPUT:

CODE:-

```
#plot number of clusters vs. gap statistic  
fviz_gap_stat(gap_stat)
```

```
#make this example reproducible  
set.seed(1)
```

#perform k-means clustering with k = 4 clusters

```
km <- kmeans(df, centers = 4, nstart = 25)
```

CODE:-

#view results

Km

```
#plot results of final k-means model
```

`fviz_cluster(km, data = df)`

OUTPUT-

> km
K-means clustering with 4 clusters of sizes 13, 13, 16, 8

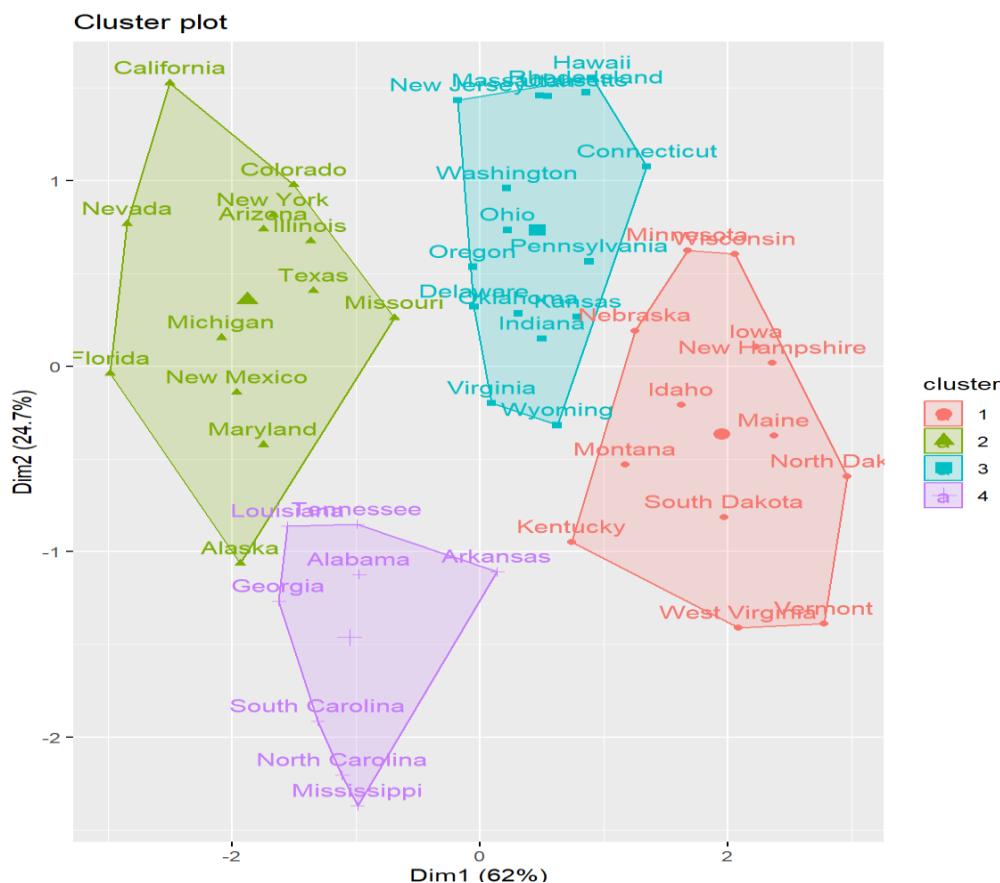
	Murder	Assault	UrbanPop	Rape
1	-0.9615407	-1.1066010	-0.9301069	-0.96676331
2	0.6950701	1.0394414	0.7226370	1.27693964
3	-0.4894375	-0.3826001	0.5758298	-0.26165379
4	1.4118898	0.8743346	-0.8145211	0.01927104

Clustering vector:							
Alabama	Alaska	Arizona	Arkansas	California	Colorado	Connecticut	
4	2	2	4	2	2	3	
Delaware	Florida	Georgia	Hawaii	Idaho	Illinois	Indiana	
3	2	4	3	1	2	3	
Iowa	Kansas	Kentucky	Louisiana	Maine	Maryland	Massachusetts	
1	3	1	4	1	2	3	
Michigan	Minnesota	Mississippi	Missouri	Montana	Nebraska	Nevada	
2	1	4	2	1	1	2	
New Hampshire	New Jersey	New Mexico	New York	North Carolina	North Dakota	Ohio	
1	3	2	2	4	1	3	
Oklahoma	Oregon	Pennsylvania	Rhode Island	South Carolina	South Dakota	Tennessee	
3	3	3	3	4	1	4	
Texas	Utah	Vermont	Virginia	Washington	West Virginia	Wisconsin	
2	3	1	3	3	1	1	
Wyoming							
3							

```
Within cluster sum of squares by cluster:  
[1] 11.952463 19.922437 16.212213 8.316061  
(between ss / total ss = 71.2 %)
```

Available components:

```
[1] "cluster"      "centers"       "totss"        "withinss"      "tot.withinss" "betweenss"     "size"  
[8] "iter"         "ifault"
```

**CODE:**

```
#find means of each cluster
aggregate(USArrests, by=list(cluster=km$cluster),mean)

#add cluster assignment to original data
final_data <- cbind(USArrests, cluster = km$cluster)

#view final data
head(final_data)
```

OUTPUT:

```
> aggregate(USArrests, by=list(cluster=km$cluster),
+           mean)
  cluster Murder Assault UrbanPop Rape
1      1  3.60000  78.53846 52.07692 12.17692
2      2 10.81538 257.38462 76.00000 33.19231
3      3  5.65625 138.87500 73.87500 18.78125
4      4 13.93750 243.62500 53.75000 21.41250
> head(final_data)
  Murder Assault UrbanPop Rape cluster
Alabama   13.2     236      58  21.2      4
Alaska    10.0     263      48  44.5      2
Arizona    8.1     294      80  31.0      2
Arkansas   8.8     190      50  19.5      4
California  9.0     276      91  40.6      2
Colorado   7.9     204      78  38.7      2
```

Pros & Cons of K-Means Clustering:

K-means clustering offers the following benefits:

- It is a fast algorithm.
- It can handle large datasets well.

However, it comes with the following potential drawbacks:

- It requires us to specify the number of clusters before performing the algorithm.
- It's sensitive to outliers.

Two alternatives to k-means clustering are k-means clustering and hierarchical clustering.

EXPERIMENT NO:10

AIM: Implement R program on Hierarchical Clustering on mtcars Dataset.

DESCRIPTION:

Hierarchical clustering in R Programming Language is an Unsupervised non-linear algorithm in which clusters are created such that they have a hierarchy(or a pre-determined ordering). For example, consider a family of up to three generations. A grandfather and mother have their children that become father and mother of their children. So, they all are grouped together to the same family i.e they form a hierarchy.

Hierarchical clustering is of two types:

- **Agglomerative Hierarchical clustering:** It starts at individual leaves and successfully merges clusters together. Its a Bottom-up approach.
- **Divisive Hierarchical clustering:** It starts at the root and recursively split the clusters. It's a top-down approach.

In hierarchical clustering, Objects are categorized into a hierarchy similar to a tree-shaped structure which is used to interpret hierarchical clustering models. The algorithm is as follows:

1. Make each data point in a single point cluster that forms **N** clusters.
2. Take the two closest data points and make them one cluster that forms **N-1** clusters.
3. Take the two closest clusters and make them one cluster that forms **N-2** clusters.
4. Repeat steps 3 until there is only one cluster.

Dendrogram is a hierarchy of clusters in which distances are converted into heights. It clusters **n** units or objects each with **p** feature into smaller groups. Units in the same cluster are joined by a horizontal line. The leaves at the bottom represent individual units. It provides a visual representation of clusters.

Thumb Rule: Largest vertical distance which doesn't cut any horizontal line defines the optimal number of clusters.

mtcars(motor trend car road test) comprise fuel consumption, performance, and 10 aspects of automobile design for 32 automobiles. It comes pre-installed with dplyr package in R.

Performing Hierarchical clustering on Dataset

Using Hierarchical Clustering algorithm on the dataset using **hclust()** which is pre-installed in stats package when R is installed.

- The values are shown as per the distance matrix calculation with the method as euclidean.
- In the model, the cluster method is average, distance is euclidean and no. of objects are 32.
- The plot dendrogram is shown with x-axis as distance matrix and y-axis as height.
- So, Tree is cut where k = 3 and each category represents its number of clusters.
- The plot denotes dendrogram after being cut. The green lines show the number of clusters as per the thumb rule.

CODE:

#loading packages

```
library(dplyr)
```

```
library(ggplot2)
```

#summary of the dataset in the package

```
head(mtcars)
```

OUTPUT:

```
> #summary of the dataset in the package
> head(mtcars)

  mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4     21.0   6 160 110 3.90 2.620 16.46  0  1   4   4
Mazda RX4 Wag 21.0   6 160 110 3.90 2.875 17.02  0  1   4   4
Datsun 710    22.8   4 108  93 3.85 2.320 18.61  1  1   4   1
Hornet 4 Drive 21.4   6 258 110 3.08 3.215 19.44  1  0   3   1
Hornet Sportabout 18.7   8 360 175 3.15 3.440 17.02  0  0   3   2
Valiant      18.1   6 225 105 2.76 3.460 20.22  1  0   3   1
```

CODE:**#finding the distance matrix**

```
distance_mat<-dist(mtcars,method='euclidean')
distance_mat
```

OUTPUT:

Merc 280C	227.8813169	64.8898713	39.3868519	1.5231546
Merc 450SE	106.4084264	175.1620073	159.8179555	122.3462489
Merc 450SL	106.4320572	175.1189767	159.7760899	122.3443771
Merc 450SLC	106.4010305	175.2118218	159.8495837	122.3934970
Cadillac Fleetwood	119.0239068	355.6627498	349.2832611	315.3904859
Lincoln Continental	104.5112999	348.9901277	341.3154316	306.6760719
Chrysler Imperial	81.4297699	338.1959373	328.4335161	292.7146896
Fiat 128	333.9792070	68.6105903	69.3127910	106.5053149
Honda Civic	344.0518316	72.0014488	78.5387212	116.7280991
Toyota Corolla	341.0218232	76.2806458	76.7731674	113.8118009
Toyota Corona	282.0508820	44.0850975	21.0962017	54.3641713
Dodge Challenger	103.9023864	192.8617917	185.8331870	152.8929263
AMC Javelin	110.3084921	180.5479760	172.5312555	139.1457974
Camaro Z28	10.0761203	273.8367985	257.7469734	219.5520854
Pontiac Firebird	80.8057339	277.4606884	271.3871978	238.1726099
Fiat X1-9	333.4843231	67.9163981	68.5564864	105.7412910
Porsche 914-2	285.1986201	39.4469276	22.1180967	57.6458160
Lotus Europa	296.4572287	72.8971106	50.1094030	74.1443580
Ford Pantera L	21.2655990	287.5238795	269.9772035	231.4081306
Ferrari Dino	226.2036333	113.3023005	80.6550953	56.8365103
Maserati Bora	107.7224977	313.8633093	288.8755628	250.5874125
Volvo 142E	275.1353516	53.6823481	24.6913548	48.8053450
Merc 450SLC	227.8813169	64.8898713	39.3868519	1.5231546
Cadillac Fleetwood	106.4084264	175.1620073	159.8179555	122.3462489
Lincoln Continental	106.4320572	175.1189767	159.7760899	122.3443771
Chrysler Imperial	106.4010305	175.2118218	159.8495837	122.3934970
Fiat 128	333.9792070	68.6105903	69.3127910	106.5053149
Honda Civic	344.0518316	72.0014488	78.5387212	116.7280991
Toyota Corolla	341.0218232	76.2806458	76.7731674	113.8118009
Toyota Corona	282.0508820	44.0850975	21.0962017	54.3641713
Dodge Challenger	103.9023864	192.8617917	185.8331870	152.8929263
AMC Javelin	110.3084921	180.5479760	172.5312555	139.1457974
Camaro Z28	10.0761203	273.8367985	257.7469734	219.5520854
Pontiac Firebird	80.8057339	277.4606884	271.3871978	238.1726099
Fiat X1-9	333.4843231	67.9163981	68.5564864	105.7412910
Porsche 914-2	285.1986201	39.4469276	22.1180967	57.6458160
Lotus Europa	296.4572287	72.8971106	50.1094030	74.1443580
Ford Pantera L	21.2655990	287.5238795	269.9772035	231.4081306
Ferrari Dino	226.2036333	113.3023005	80.6550953	56.8365103
Maserati Bora	107.7224977	313.8633093	288.8755628	250.5874125
Volvo 142E	275.1353516	53.6823481	24.6913548	48.8053450
Merc 450SLC	227.8813169	64.8898713	39.3868519	1.5231546
Cadillac Fleetwood	106.4084264	175.1620073	159.8179555	122.3462489
Lincoln Continental	106.4320572	175.1189767	159.7760899	122.3443771
Chrysler Imperial	106.4010305	175.2118218	159.8495837	122.3934970
Fiat 128	333.9792070	68.6105903	69.3127910	106.5053149
Honda Civic	344.0518316	72.0014488	78.5387212	116.7280991
Toyota Corolla	341.0218232	76.2806458	76.7731674	113.8118009
Toyota Corona	282.0508820	44.0850975	21.0962017	54.3641713
Dodge Challenger	103.9023864	192.8617917	185.8331870	152.8929263
AMC Javelin	110.3084921	180.5479760	172.5312555	139.1457974
Camaro Z28	10.0761203	273.8367985	257.7469734	219.5520854
Pontiac Firebird	80.8057339	277.4606884	271.3871978	238.1726099
Fiat X1-9	333.4843231	67.9163981	68.5564864	105.7412910
Porsche 914-2	285.1986201	39.4469276	22.1180967	57.6458160
Lotus Europa	296.4572287	72.8971106	50.1094030	74.1443580
Ford Pantera L	21.2655990	287.5238795	269.9772035	231.4081306
Ferrari Dino	226.2036333	113.3023005	80.6550953	56.8365103
Maserati Bora	107.7224977	313.8633093	288.8755628	250.5874125
Volvo 142E	275.1353516	53.6823481	24.6913548	48.8053450
Merc 450SLC	227.8813169	64.8898713	39.3868519	1.5231546
Cadillac Fleetwood	106.4084264	175.1620073	159.8179555	122.3462489
Lincoln Continental	106.4320572	175.1189767	159.7760899	122.3443771
Chrysler Imperial	106.4010305	175.2118218	159.8495837	122.3934970
Fiat 128	333.9792070	68.6105903	69.3127910	106.5053149
Honda Civic	344.0518316	72.0014488	78.5387212	116.7280991
Toyota Corolla	341.0218232	76.2806458	76.7731674	113.8118009
Toyota Corona	282.0508820	44.0850975	21.0962017	54.3641713
Dodge Challenger	103.9023864	192.8617917	185.8331870	152.8929263
AMC Javelin	110.3084921	180.5479760	172.5312555	139.1457974
Camaro Z28	10.0761203	273.8367985	257.7469734	219.5520854
Pontiac Firebird	80.8057339	277.4606884	271.3871978	238.1726099
Fiat X1-9	333.4843231	67.9163981	68.5564864	105.7412910
Porsche 914-2	285.1986201	39.4469276	22.1180967	57.6458160
Lotus Europa	296.4572287	72.8971106	50.1094030	74.1443580
Ford Pantera L	21.2655990	287.5238795	269.9772035	231.4081306
Ferrari Dino	226.2036333	113.3023005	80.6550953	56.8365103
Maserati Bora	107.7224977	313.8633093	288.8755628	250.5874125
Volvo 142E	275.1353516	53.6823481	24.6913548	48.8053450
Merc 450SLC	227.8813169	64.8898713	39.3868519	1.5231546
Cadillac Fleetwood	106.4084264	175.1620073	159.8179555	122.3462489
Lincoln Continental	106.4320572	175.1189767	159.7760899	122.3443771
Chrysler Imperial	106.4010305	175.2118218	159.8495837	122.3934970
Fiat 128	333.9792070	68.6105903	69.3127910	106.5053149
Honda Civic	344.0518316	72.0014488	78.5387212	116.7280991
Toyota Corolla	341.0218232	76.2806458	76.7731674	113.8118009
Toyota Corona	282.0508820	44.0850975	21.0962017	54.3641713
Dodge Challenger	103.9023864	192.8617917	185.8331870	152.8929263
AMC Javelin	110.3084921	180.5479760	172.5312555	139.1457974
Camaro Z28	10.0761203	273.8367985	257.7469734	219.5520854
Pontiac Firebird	80.8057339	277.4606884	271.3871978	238.1726099
Fiat X1-9	333.4843231	67.9163981	68.5564864	105.7412910
Porsche 914-2	285.1986201	39.4469276	22.1180967	57.6458160
Lotus Europa	296.4572287	72.8971106	50.1094030	74.1443580
Ford Pantera L	21.2655990	287.5238795	269.9772035	231.4081306
Ferrari Dino	226.2036333	113.3023005	80.6550953	56.8365103
Maserati Bora	107.7224977	313.8633093	288.8755628	250.5874125
Volvo 142E	275.1353516	53.6823481	24.6913548	48.8053450
Merc 450SLC	227.8813169	64.8898713	39.3868519	1.5231546
Cadillac Fleetwood	106.4084264	175.1620073	159.8179555	122.3462489
Lincoln Continental	106.4320572	175.1189767	159.7760899	122.3443771
Chrysler Imperial	106.4010305	175.2118218	159.8495837	122.3934970
Fiat 128	333.9792070	68.6105903	69.3127910	106.5053149
Honda Civic	344.0518316	72.0014488	78.5387212	116.7280991
Toyota Corolla	341.0218232	76.2806458	76.7731674	113.8118009
Toyota Corona	282.0508820	44.0850975	21.0962017	54.3641713
Dodge Challenger	103.9023864	192.8617917	185.8331870	152.8929263
AMC Javelin	110.3084921	180.5479760	172.5312555	139.1457974
Camaro Z28	10.0761203	273.8367985	257.7469734	219.5520854
Pontiac Firebird	80.8057339	277.4606884	271.3871978	238.1726099
Fiat X1-9	333.4843231	67.9163981	68.5564864	105.7412910
Porsche 914-2	285.1986201	39.4469276	22.1180967	57.6458160
Lotus Europa	296.4572287	72.8971106	50.1094030	74.1443580
Ford Pantera L	21.2655990	287.5238795	269.9772035	231.4081306
Ferrari Dino	226.2036333	113.3023005	80.6550953	56.8365103
Maserati Bora	107.7224977	313.8633093	288.8755628	250.5874125
Volvo 142E	275.1353516	53.6823481	24.6913548	48.8053450
Merc 450SLC	227.8813169	64.8898713	39.3868519	1.5231546
Cadillac Fleetwood	106.4084264	175.1620073	159.8179555	122.3462489
Lincoln Continental	106.4320572	175.1189767	159.7760899	122.3443771
Chrysler Imperial	106.4010305	175.2118218	159.8495837	122.3934970
Fiat 128	333.9792070	68.6105903	69.3127910	106.5053149
Honda Civic	344.0518316	72.0014488	78.5387212	116.7280991
Toyota Corolla	341.0218232	76.2806458	76.7731674	113.8118009
Toyota Corona	282.0508820	44.0850975	21.0962017	54.3641713
Dodge Challenger	103.9023864	192.8617917	185.8331870	152.8929263
AMC Javelin	110.3084921	180.5479760	172.5312555	139.1457974
Camaro Z28	10.0761203	273.8367985	257.7469734	219.5520854
Pontiac Firebird	80.8057339	277.4606884	271.3871978	238.1726099
Fiat X1-9	333.4843231	67.9163981	68.5564864	105.7412910
Porsche 914-2	285.1986201	39.4469276	22.1180967	57.6458160
Lotus Europa	296.4572287	72.8971106	50.1094030	74.1443580
Ford Pantera L	21.2655990	287.5238795	269.9772035	231.4081306
Ferrari Dino	226.2036333	113.3023005	80.6550953	56.8365103
Maserati Bora	107.7224977	313.8633093	288.8755628	250.5874125
Volvo 142E	275.1353516	53.6823481	24.6913548	48.8053450
Merc 450SLC	227.8813169	64.8898713	39.3868519	1.5231546
Cadillac Fleetwood	106.4084264	175.1620073	159.8179555	122.3462489
Lincoln Continental	106.4320572			

OUTPUT:

Fiat X1-9	227.8176554	417.2490481	409.4998363	396.7597522	5.1473415
Porsche 914-2	179.5720446	370.0956775	362.0145494	348.8466861	49.0644372
Lotus Europa	193.3969216	388.5350012	379.4716659	364.5994326	49.9112509
Ford Pantera L	112.8332602	134.8119464	119.7236456	95.3805385	337.1639236
Ferrari Dino	131.0704490	328.5441628	317.7063117	300.1640703	128.3950054
Maserati Bora	157.1683970	214.9366858	199.3420611	174.2936864	349.5338830
Volvo 142E	170.4843735	364.1000930	355.4009443	341.2896659	61.3301247
	Honda Civic	Toyota Corolla	Toyota Corona	Dodge Challenger	AMC Javelin
					Camaro Z28
Mazda RX4 Wag					
Datsun 710					
Hornet 4 Drive					
Hornet Sportabout					
Valiant					
Duster 360					
Merc 240D					
Merc 230					
Merc 280					
Merc 280C					
Merc 450SE					
Merc 450SL					
Merc 450SLC					
Cadillac Fleetwood					
Lincoln Continental					
Chrysler Imperial					
Fiat 128					
Honda Civic					
Toyota Corolla	14.3480626				
Toyota Corona	63.8985563	59.8451285			
Dodge Challenger	261.8498815	261.8345312	205.0347927		
AMC Javelin	248.9636504	248.6917065	191.5580526	14.0154995	
Camaro Z28	335.8883188	332.6589699	273.6316895	100.3046106	105.6062618
Pontiac Firebird	347.0655360	347.1667643	290.6240706	85.8075196	99.2836114
Fiat X1-9	14.7807070	10.3922856	51.8411748	253.6624046	240.5266823
Porsche 914-2	59.4588768	56.3243031	8.6535903	325.1490914	
Lotus Europa	64.0495153	53.8846563	31.2536926	206.6452569	193.3080584
Ford Pantera L	347.8337714	343.9920962	285.1287911	276.8924414	
Ferrari Dino	141.7044478	133.4707617	82.2355734	226.5004836	212.7568765
Maserati Bora	362.1620777	355.2601619	299.1865216	287.6179004	
Volvo 142E	73.3766041	67.7189421	12.2505275	118.7516779	123.3832044
	Pontiac Firebird	Fiat X1-9	Porsche 914-2	19.3589023	
			Lotus Europa	Ford Pantera L	
			Ferrari Dino		
Mazda RX4 Wag					
Datsun 710					
Hornet 4 Drive					
Hornet Sportabout					
Valiant					
Duster 360					
Merc 240D					
Merc 230					
Merc 280					
Merc 280C					
Merc 450SE					
Merc 450SL					
Merc 450SLC					
Cadillac Fleetwood					
Lincoln Continental					
Chrysler Imperial					
Fiat 128					
Honda Civic					
Toyota Corolla					
Toyota Corona					
Dodge Challenger					
AMC Javelin					
Camaro Z28					
Pontiac Firebird					
Fiat X1-9	339.1396182				
Porsche 914-2	292.1646488	48.3775209			
Lotus Europa	311.3862342	49.8406880	33.7678653		
Ford Pantera L	101.7389686	336.7018783	288.5852993	297.5376920	
Ferrari Dino	255.0570519	127.8210813	87.9105966	80.4553451	224.4587490
Maserati Bora	188.3240020	349.1199576	303.9222549	303.2796468	86.9383253
Volvo 142E	286.7497823	60.4120429	18.7555858	27.8104457	223.5342175
	Maserati Bora				277.4803312
Mazda RX4 Wag					70.4751034
Datsun 710					
Hornet 4 Drive					
Hornet Sportabout					
Valiant					
Duster 360					
Merc 240D					
Merc 230					
Merc 280					
Merc 280C					
Merc 450SE					
Merc 450SL					
Merc 450SLC					
Cadillac Fleetwood					
Lincoln Continental					
Chrysler Imperial					
Fiat 128					
Honda Civic					
Toyota Corolla					
Toyota Corona					
Dodge Challenger					
AMC Javelin					

OUTPUT:

Camaro Z28
 Pontiac Firebird
 Fiat X1-9
 Porsche 914-2
 Lotus Europa
 Ford Pantera L
 Ferrari Dino
 Maserati Bora
 Volvo 142E

289.1157363

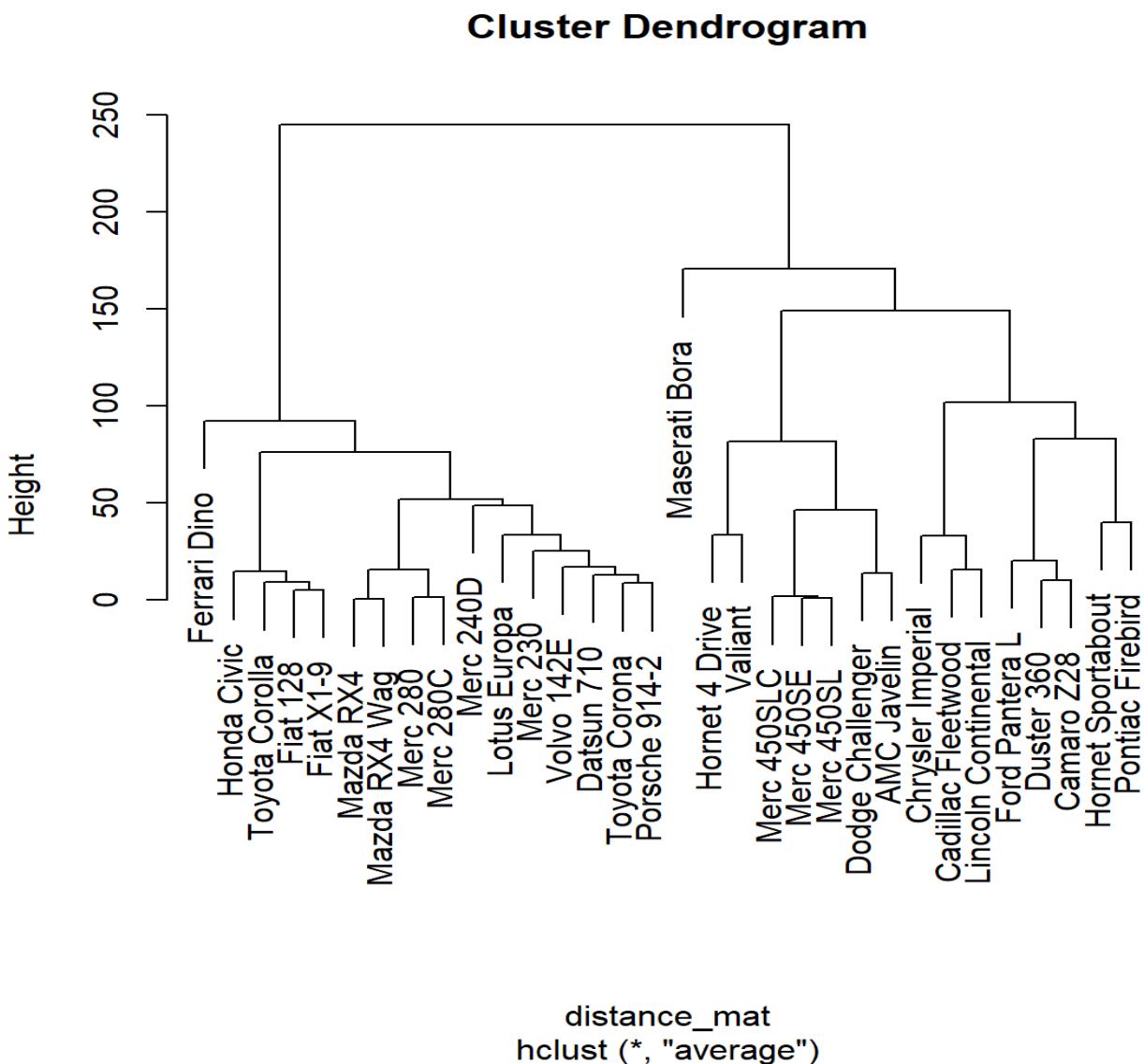
CODE:

```
#fitting the hierarchical clustering model to the training data
```

```
Hierar_cl<-hclust(distance_mat,method="average")
```

```
#plotting the dendrogram
```

```
plot(Hierar_cl)
```

OUTPUT:

CODE:

```
#choosing the number of clusters (cut by the number of clustering)
```

```
num_clusters<-3 #change this to the desired number of clusters
```

```
fit<-cutree(Hierar_cl,k=num_clusters)
```

```
fit
```

```
#display the count of datapoints in each cluster
```

```
table(fit)
```

OUTPUT:

```
> fit
   Mazda RX4      Mazda RX4 Wag      Datsun 710      Hornet 4 Drive      Hornet Sportabout
      1                  1                  1                  2                  2
      Valiant        Duster 360      Merc 240D      Merc 230          Merc 280
      2                  2                  1                  1                  1
      Merc 280C       Merc 450SE      Merc 450SL      Merc 450SLC     Cadillac Fleetwood
      1                  2                  2                  2                  2
Lincoln Continental Chrysler Imperial      Fiat 128      Honda Civic      Toyota Corolla
      2                  2                  1                  1                  1
      Toyota Corona    Dodge Challenger      AMC Javelin      Camaro Z28      Pontiac Firebird
      1                  2                  2                  2                  2
      Fiat X1-9        Porsche 914-2      Lotus Europa      Ford Pantera L      Ferrari Dino
      1                  1                  1                  2                  1
      Maserati Bora    Volvo 142E
      3                  1
>
> #display the count of datapoints in each cluster
> table(fit)
fit
 1  2  3
16 15  1
```

CODE:

```
#highlight clusters on the dendrogram
```

```
rect.hclust(Hierar_cl,k=num_clusters,border="green")
```

```
#create a dataframe with cluster assignments
```

```
clustered_data<-data.frame(mtcars,Cluster=factor(fit))
```

```
#define the custom cluster colors
```

```
custom_colors<-c("red","pink","lavender")
```

```
#create a scatterplot with custom colors
```

```
scatterplot<-ggplot(clustered_data,aes(x=mpg,
```

```

y=disp,
color=Cluster))+

geom_point(size=3)+

geom_text(aes(label=Cluster),hjust=0,vjust=0,
nudge_x=1,nudge_y=20,size=4)+

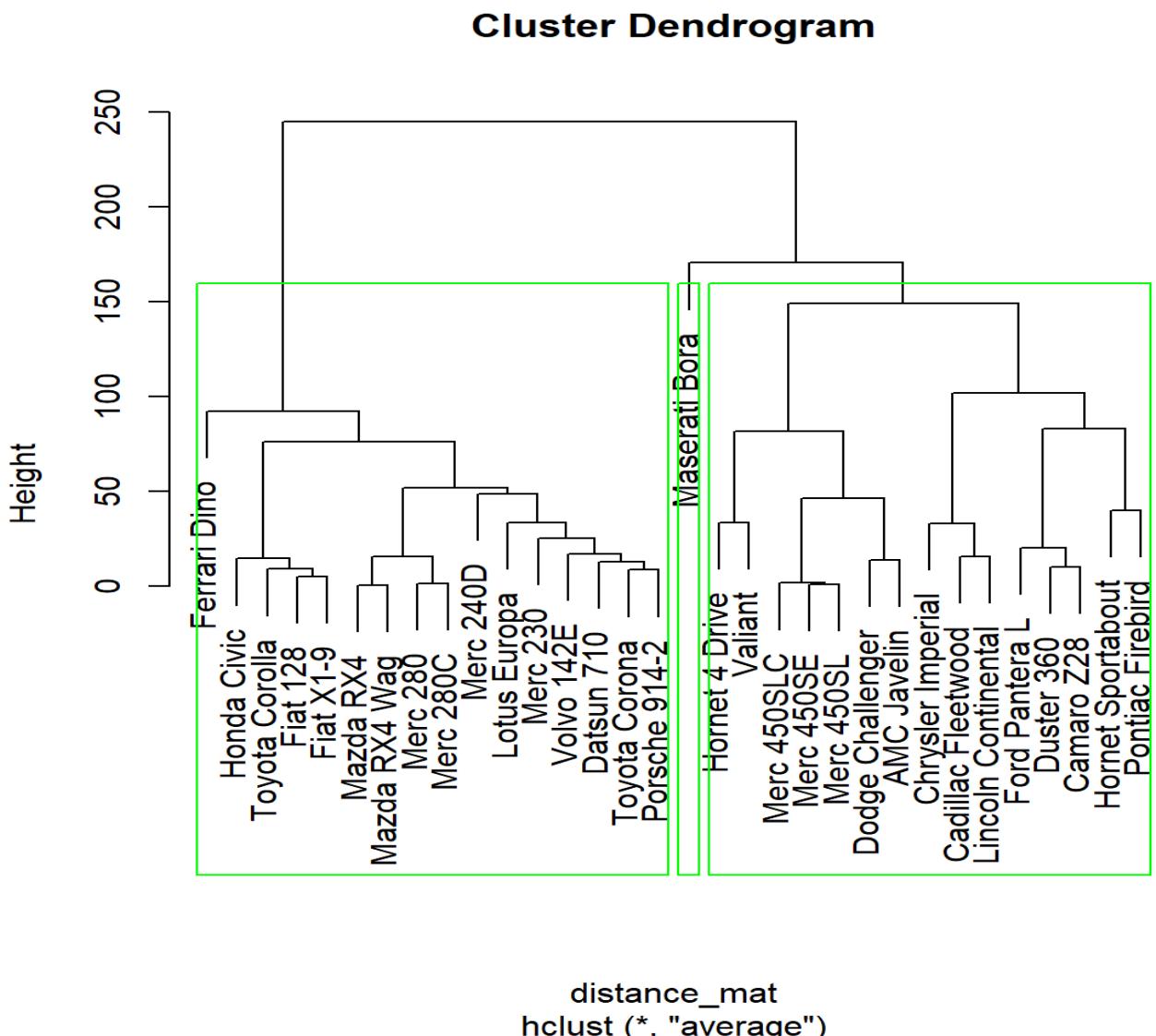
labs(x="Miles per Gallon(mpg)",y="Displacement(disp)",
title="scatterplot of clusters with custom colors")+
scale_color_manual(values=custom_colors,name='Cluster')

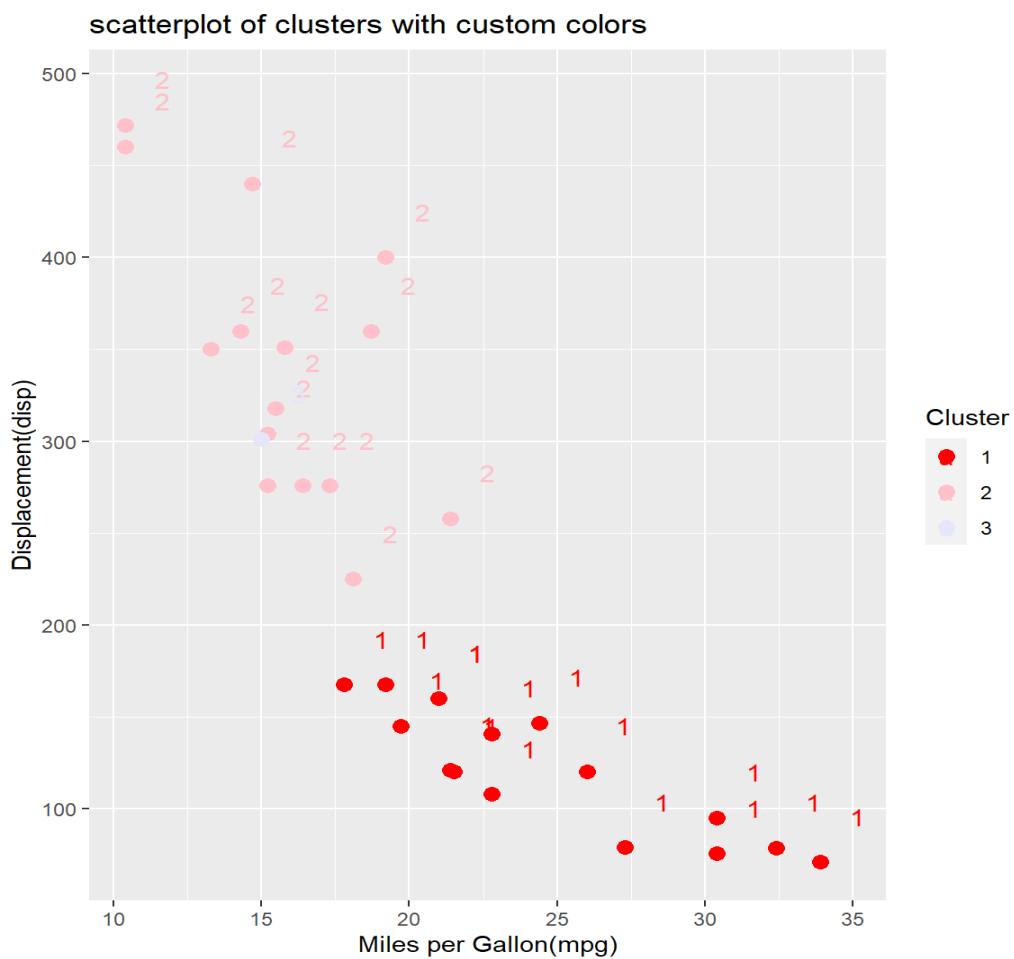
```

#display scatter plot

scatterplot

OUTPUT:



**CODE:**

```
# Calculate the mean mpg for each cluster
```

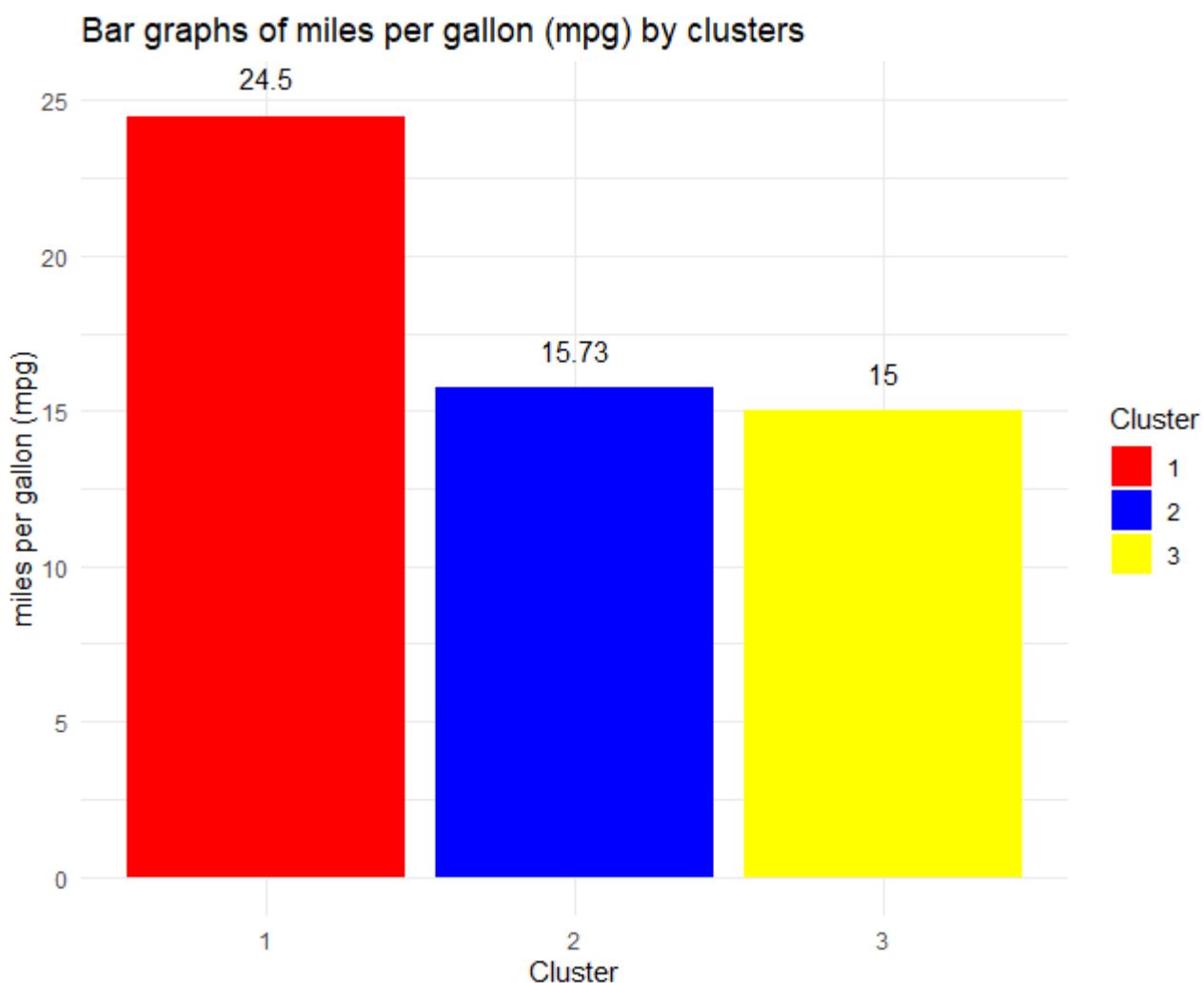
```
cluster_means <- clustered_data %>%
  group_by(Cluster) %>%
  summarise(mean_mpg = mean(mpg))
```

```
# Create bar graphs for "mpg" variable within each cluster
```

```
bar_graph <- ggplot(cluster_means, aes(x = Cluster, y = mean_mpg, fill = Cluster)) +
  geom_bar(stat = "identity", position = "dodge") +
  geom_text(aes(label = round(mean_mpg, 2), y = mean_mpg + 0.5), vjust = -0.5, position =
  position_dodge(width = 0.9)) +
  labs(x = "Cluster", y = "Miles per Gallon (mpg)", title = "Bar Graphs of miles per gallon (mpg) by
Cluster") +
  scale_fill_manual(values = custom_colors, name = "Cluster") +
  theme_minimal()
```

```
#display barplot
```

```
print(bar_graph)
```

OUTPUT:

EXPERIMENT:11

AIM: Create a R program to implement CART, K-Nearest Neighbours, SVM, LDA, Random Forest on PimaIndianDiabetes dataset and compare and visualize the results

DESCRIPTION: The “PimaIndianDiabetes” Dataset, originally from the National Institute of Diabetes and Digestive and Kidney Diseases, contains information of 768 women from a population near Phoenix, Arizona, USA. The outcome tested was Diabetes, 258 tested positive and 500 tested negative. Therefore, there is one target (dependent) variable and the 8 attributes (TYNECKI, 2018): pregnancies, OGTT(Oral Glucose Tolerance Test), blood pressure, skin thickness, insulin, BMI(Body Mass Index), age, pedigree diabetes function.

CODE:

```
# load libraries
library(mlbench)
library(caret)

# load the dataset
data(PimaIndiansDiabetes)

# prepare training scheme
control <- trainControl(method="repeatedcv", number=10, repeats=3)

# CART
set.seed(7)
fit.cart<- train(diabetes~., data=PimaIndiansDiabetes, method="rpart", trControl=control)

# LDA
set.seed(7)
fit.lda<- train(diabetes~., data=PimaIndiansDiabetes, method=" lda", trControl=control)

# SVM
set.seed(7)
fit.svm<- train(diabetes~., data=PimaIndiansDiabetes, method="svmRadial", trControl=control)

# kNN
set.seed(7)
fit.knn<- train(diabetes~., data=PimaIndiansDiabetes, method="knn", trControl=control)

# Random Forest
set.seed(7)
fit.rf<- train(diabetes~., data=PimaIndiansDiabetes, method="rf", trControl=control)

# collect resamples
results <- resamples(list(CART=fit.cart, LDA=fit.lda, SVM=fit.svm, KNN=fit.knn, RF=fit.rf))

# summarize differences between modes
summary(results)
```

OUTPUT:

```
Call:
summary.resamples(object = results)

Models: CART, LDA, SVM, KNN, RF
Number of resamples: 30

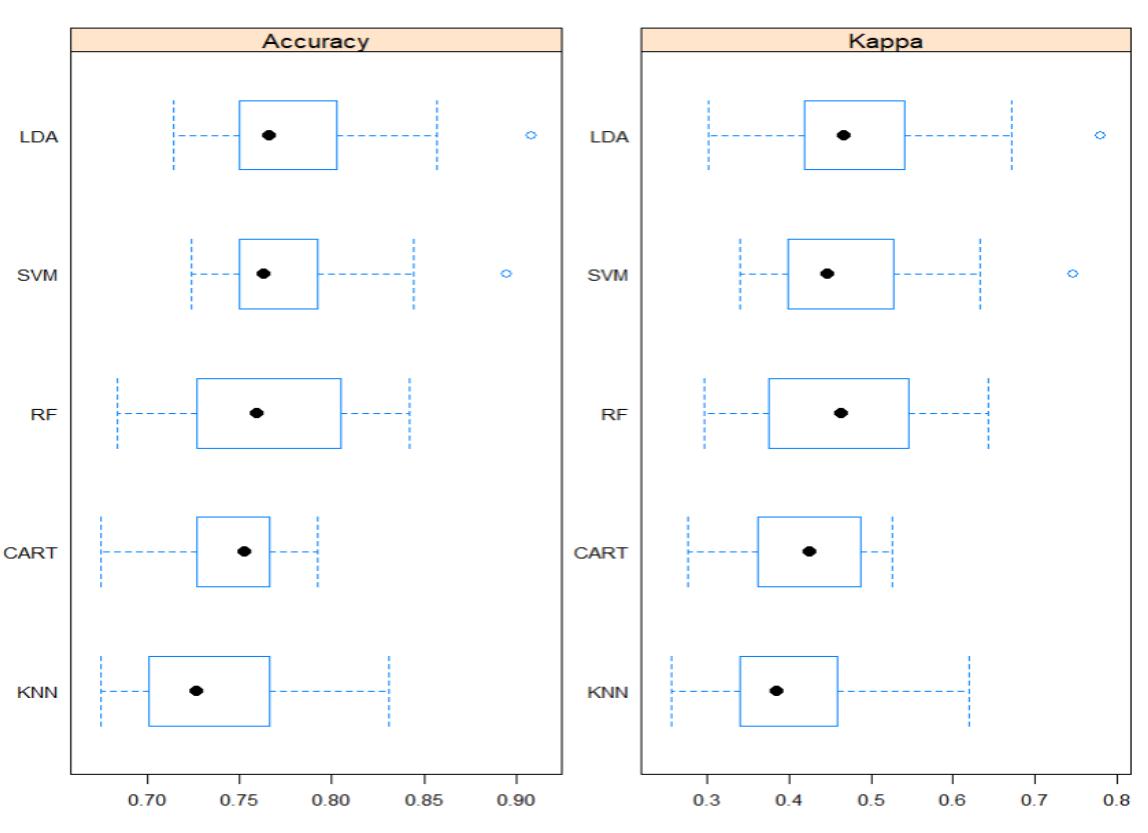
Accuracy
      Min.   1st Qu.     Median       Mean   3rd Qu.     Max. NA's
CART 0.6753247 0.7272727 0.7532468 0.7469697 0.7662338 0.7922078 0
LDA  0.7142857 0.7508117 0.7662338 0.7791069 0.8000256 0.9078947 0
SVM  0.7236842 0.7508117 0.7631579 0.7712919 0.7915243 0.8947368 0
KNN  0.6753247 0.7036056 0.7272727 0.7369503 0.7662338 0.8311688 0
RF   0.6842105 0.7305195 0.7597403 0.7638528 0.8019481 0.8421053 0

Kappa
      Min.   1st Qu.     Median       Mean   3rd Qu.     Max. NA's
CART 0.2762566 0.3620724 0.4241878 0.4151867 0.4861107 0.5250000 0
LDA  0.3011551 0.4192537 0.4662541 0.4862025 0.5308596 0.7812500 0
SVM  0.3391908 0.3997116 0.4460612 0.4621585 0.5234605 0.7475083 0
KNN  0.2553191 0.3406000 0.3841761 0.3984995 0.4539789 0.6195363 0
RF   0.2951613 0.3778304 0.4640696 0.4630809 0.5447483 0.6426332 0
```

CODE:

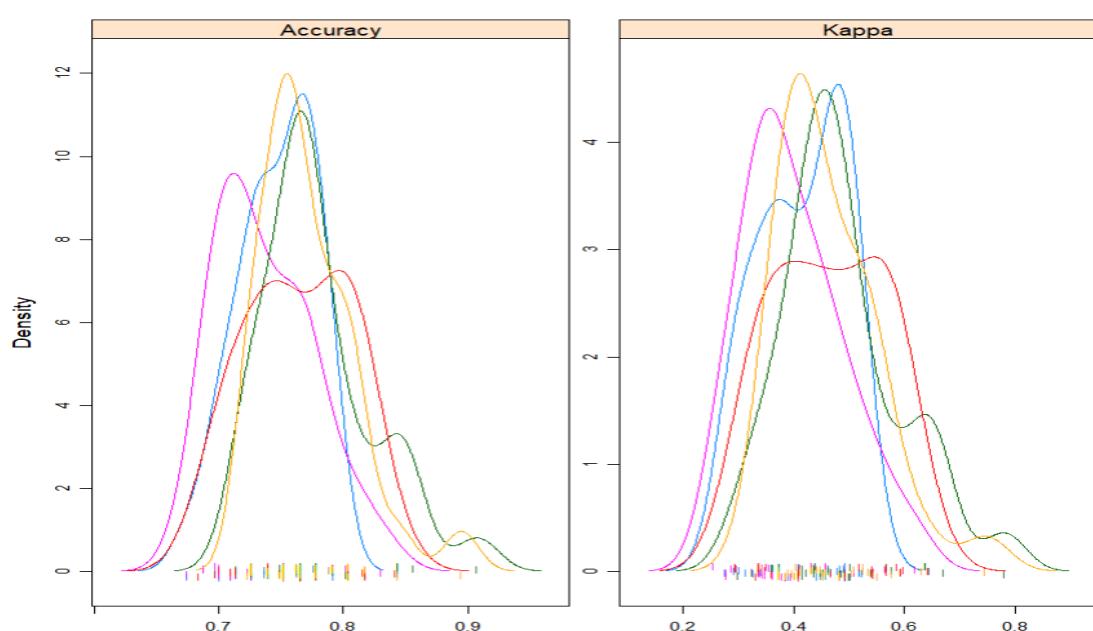
```
# box and whisker plots to compare models
```

```
scales <- list(x=list(relation="free"), y=list(relation="free"))
bwplot(results, scales=scales)
```

OUTPUT:**CODE:**

```
# density plots of accuracy
```

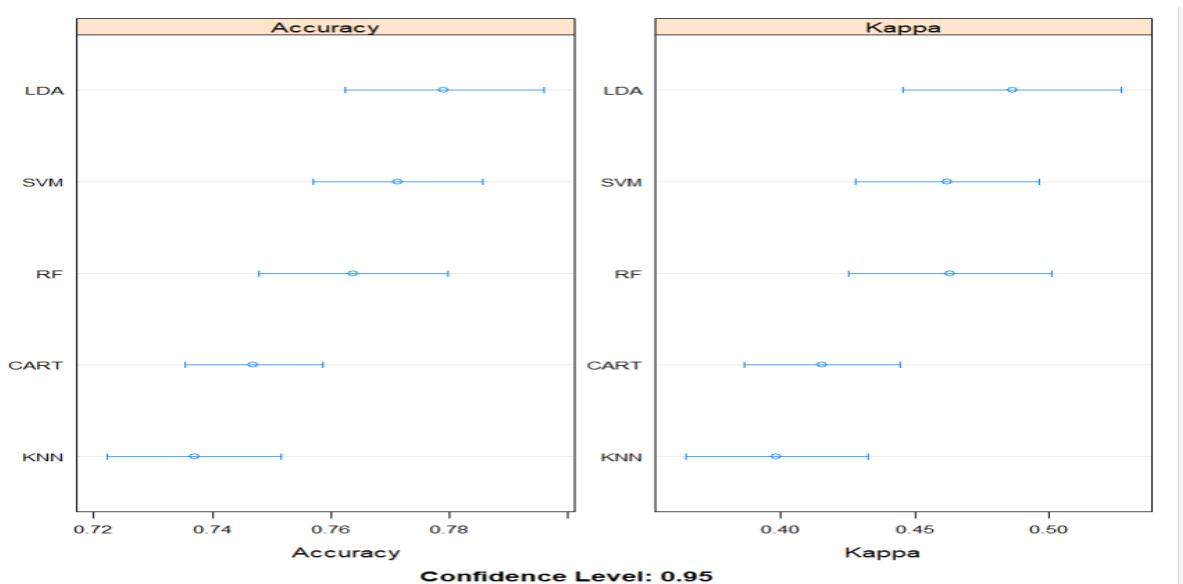
```
scales <- list(x=list(relation="free"), y=list(relation="free"))
densityplot(results, scales=scales, pch = "|")
```

OUTPUT:

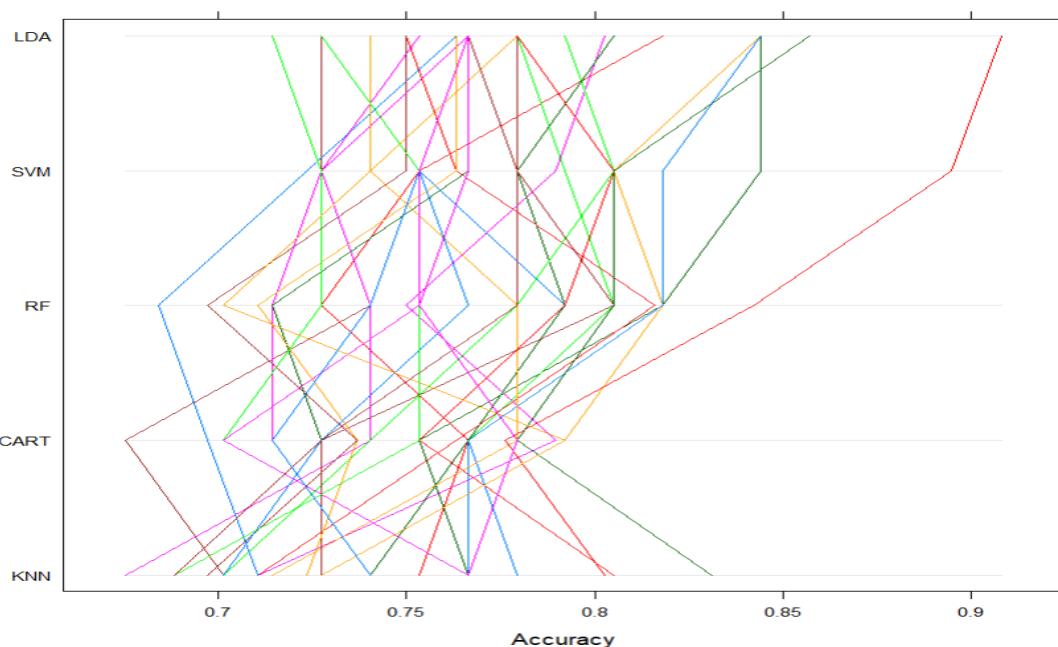
CODE:

```
# dot plots of accuracy
```

```
scales <- list(x=list(relation="free"), y=list(relation="free"))
dotplot(results, scales=scales)
```

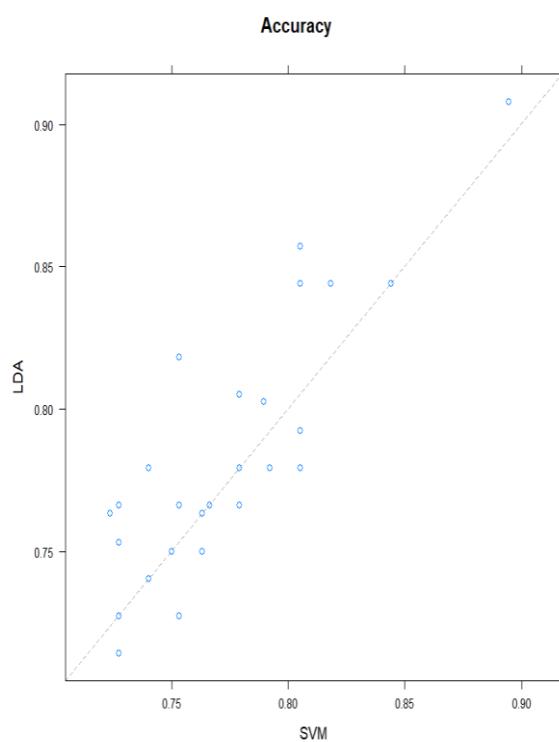
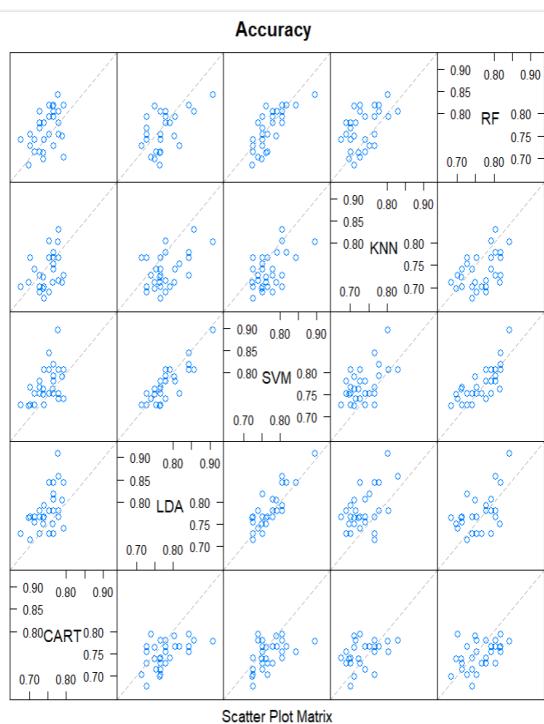
OUTPUT:**CODE:**

```
parallelplot(results) # parallel plots to compare models
```

OUTPUT:**CODE:**

```
splom(results) # pair-wise scatterplots of predictions to compare models
```

```
xyplot(results, models=c("LDA", "SVM")) # xyplot plots to compare models
```

OUTPUT:**CODE:**

```
diffs <- diff(results) # difference in model predictions
summary(diffs) # summarize p-values for pair-wise comparisons
```

OUTPUT:

```
> summary(diffs)

Call:
summary.diff.resamples(object = diffs)

p-value adjustment: bonferroni
Upper diagonal: estimates of the difference
Lower diagonal: p-value for H0: difference = 0

Accuracy
      CART      LDA      SVM      KNN      RF 
CART -0.032137 -0.024322 0.010019 -0.016883
LDA  0.0011862   0.007815 0.042157  0.015254
SVM  0.0116401  0.9156892   0.034342 0.007439
KNN  1.0000000 6.68e-05 0.0002941   -0.026902
RF   0.2727542 0.4490617 1.0000000 0.0183793

Kappa
      CART      LDA      SVM      KNN      RF 
CART -0.0710158 -0.0469717 0.0166872 -0.0478942
LDA  0.0008086   0.0240440 0.0877029  0.0231215
SVM  0.0258079 0.3562734   0.0636589 -0.0009225
KNN  1.0000000 0.0003858 0.0040823   -0.0645814
RF   0.0211763 1.0000000 1.0000000 0.0158974
```

EXPERIMENT NO:12

AIM:Create aR program to implement Regression Comparison on Boston Housing dataset and compare and visualize the results

DESCRIPTION:The “Boston” Housing dataset is a commonly used dataset for regression tasks in machine learning. It contains various features related to housing in Boston, such as the crime rate, average number of rooms per dwelling, and more. The target variable in this dataset is typically the median value of owner-occupied homes (in thousands of dollars).

CODE:

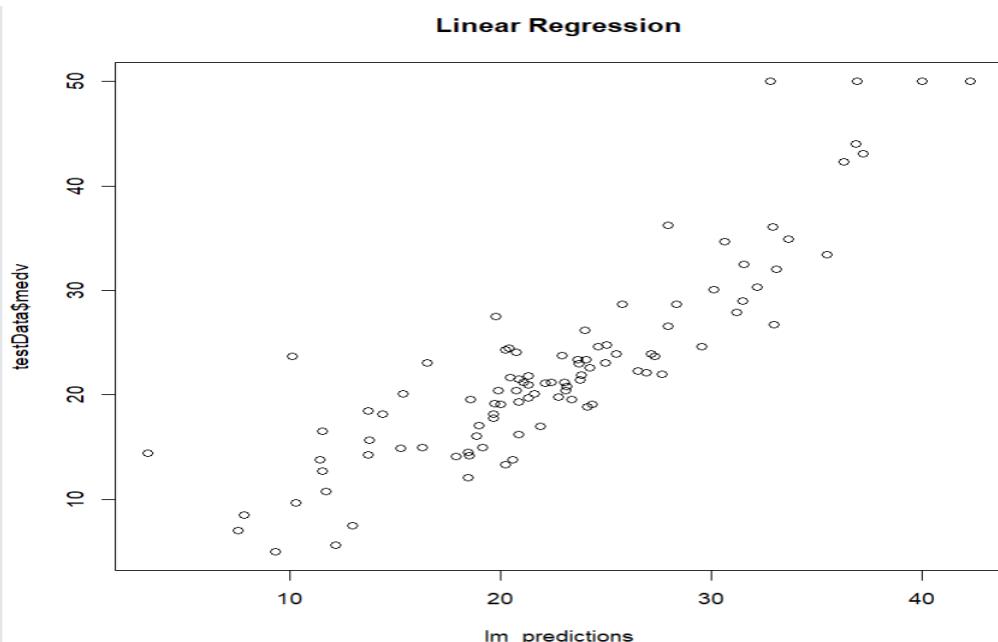
```
# Load necessary libraries
```

```
library(MASS)
library(caret)
library(e1071)
library(randomForest)
library(rpart)
library(rpart.plot)
data(Boston) # Load BostonHousing dataset
set.seed(123) # Set seed for reproducibility
# Split data into training and testing sets
```

```
trainIndex<- createDataPartition(Boston$medv, p = 0.8, list = FALSE)
trainData<- Boston[trainIndex, ]
testData<- Boston[-trainIndex, ]
```

Linear Regression

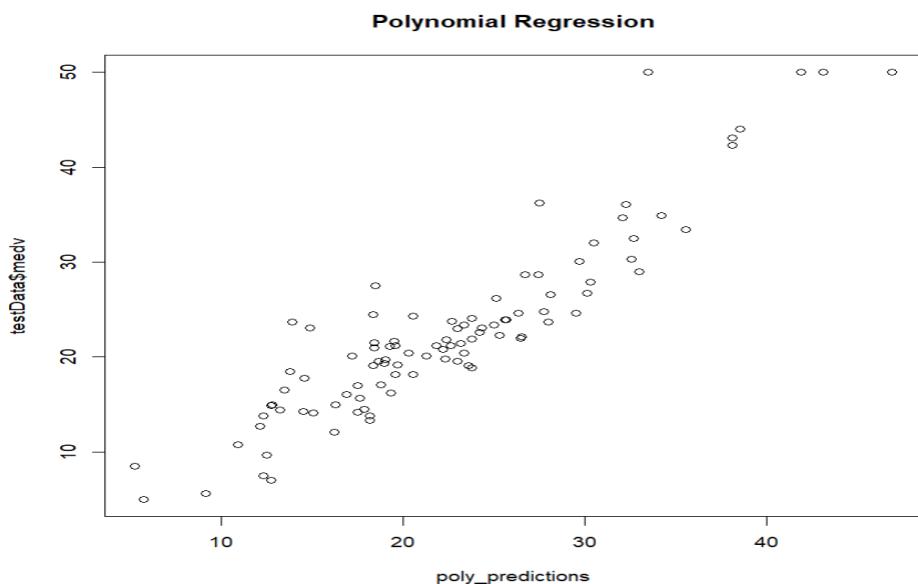
```
lm_model<- lm(medv ~ ., data = trainData)
lm_predictions<- predict(lm_model, newdata = testData)
plot(lm_predictions, testData$medv, main = "Linear Regression")
```

OUTPUT:

CODE:

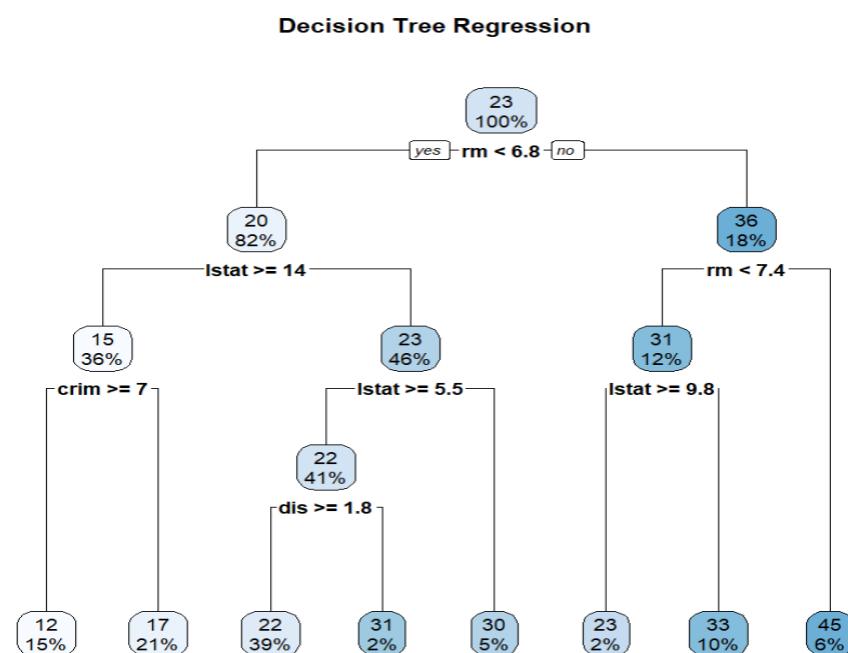
```
# Polynomial Regression (excluding 'chas')
```

```
poly_model<- lm(medv ~ poly(crim, 2) + poly(zn, 2) + poly(indus, 2) + polynox, 2) + poly(rm, 2) + poly(age, 2) +
poly(dis, 2) + poly(rad, 2) + poly(tax, 2) + poly(ptratio, 2) + poly(black, 2) + poly(lstat, 2), data = trainData)
poly_predictions<- predict(poly_model, newdata = testData)
plot(poly_predictions, testData$medv, main = "Polynomial Regression")
```

OUTPUT:**CODE:**

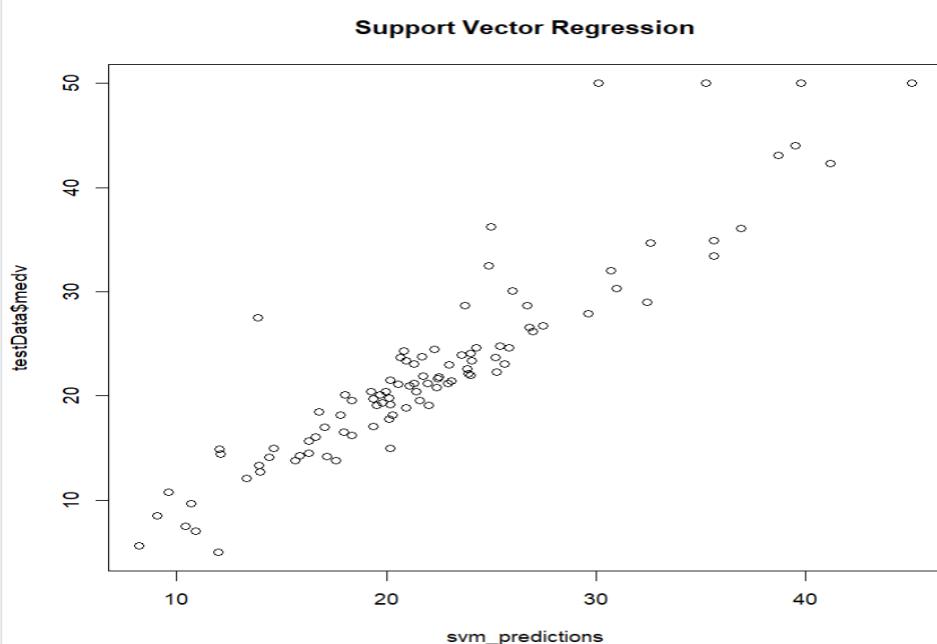
```
# Decision Tree Regression
```

```
tree_model<- rpart(medv ~ ., data = trainData, method = "anova")
tree_predictions<- predict(tree_model, newdata = testData)
rpart.plot(tree_model, main = "Decision Tree Regression")
```

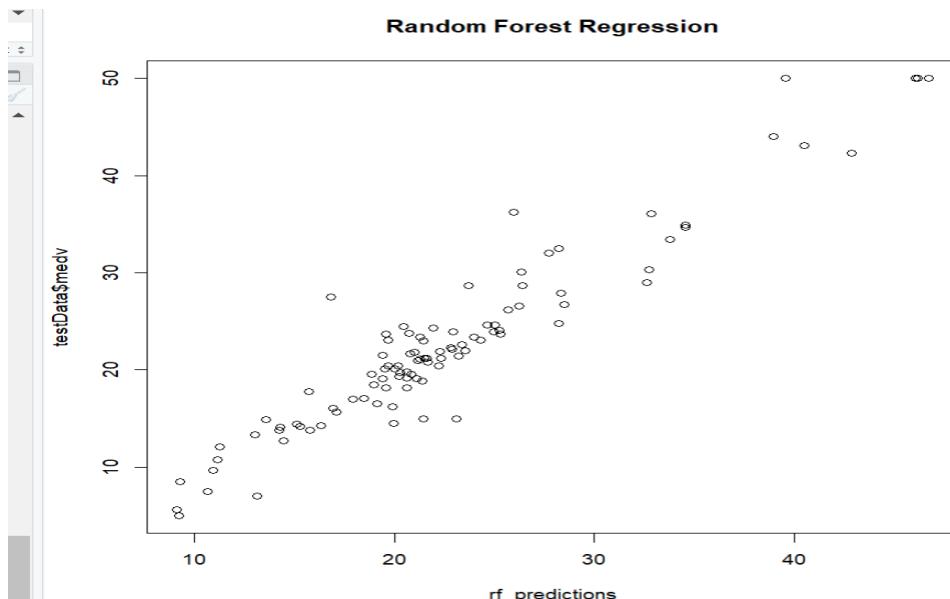
OUTPUT:

CODE:**# Support Vector Regression**

```
svm_model<- svm(medv ~ ., data = trainData)  
svm_predictions<- predict(svm_model, newdata = testData)  
plot(svm_predictions, testData$medv, main = "Support Vector Regression")
```

OUTPUT:**CODE:****# Random Forest Regression**

```
rf_model<- randomForest(medv ~ ., data = trainData)  
rf_predictions<- predict(rf_model, newdata = testData)  
plot(rf_predictions, testData$medv, main = "Random Forest Regression")
```

OUTPUT:

CODE:**# Model Summaries**

```
summary(lm_model)
```

```
summary(poly_model)
```

OUTPUT:

```
Call:
lm(formula = medv ~ ., data = trainData)

Residuals:
    Min      1Q  Median      3Q     Max 
-14.9550 -2.7996 -0.4647  1.7767 25.0993 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 37.733617   5.619935   6.714 6.63e-11 ***
crim        -0.093857   0.039157  -2.397 0.016999 *  
zn          0.039436   0.015987   2.467 0.014062 *  
indus       -0.012988   0.069595  -0.187 0.852059    
chas        2.290187   0.940621   2.435 0.015346 *  
nox         -17.130560  4.342272  -3.945 9.45e-05 *** 
rm          3.499219   0.451445   7.751 7.87e-14 *** 
age         0.009823   0.015510   0.633 0.526905    
dis         -1.390769  0.230614  -6.031 3.77e-09 *** 
rad          0.330939   0.077135   4.290 2.25e-05 *** 
tax         -0.012386  0.004342  -2.852 0.004568 **  
ptratio     -0.960676  0.150307  -6.391 4.66e-10 *** 
black       0.009841   0.002935   3.353 0.000877 *** 
lstat      -0.562095  0.059180  -9.498 < 2e-16 *** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 4.801 on 393 degrees of freedom
Multiple R-squared: 0.7346, Adjusted R-squared: 0.7258
F-statistic: 83.68 on 13 and 393 DF, p-value: < 2.2e-16

CODE:

```
print(tree_model)
```

OUTPUT:

```
> print(tree_model)
n= 407

node), split, n, deviance, yval
  * denotes terminal node

1) root 407 34125.5800 22.51057
  2) rm< 6.8375 334 12681.2200 19.62695
    4) lstat>=14.4 146 2607.9150 15.05822
      8) crim>=7.006285 60 659.0840 11.86000 *
      9) crim< 7.006285 86 906.9406 17.28953 *
    5) lstat< 14.4 188 4659.1330 23.17500
      10) lstat>=5.51 167 2966.0360 22.35329
        20) dis>=1.7548 160 1307.6640 21.98312 *
        21) dis< 1.7548 7 1135.3290 30.81429 *
      11) lstat< 5.51 21 683.6381 29.70952 *
    3) rm>=6.8375 73 5959.9890 35.70411
      6) rm< 7.443 49 2037.2070 31.28367
        12) lstat>=9.76 8 440.5750 23.02500 *
        13) lstat< 9.76 41 944.5190 32.89512 *
      7) rm>=7.443 24 1010.4700 44.72917 *
```

```
Call:
lm(formula = medv ~ poly(crim, 2) + poly(zn, 2) + poly(indus,
2) + poly(nox, 2) + poly(rm, 2) + poly(age, 2) + poly(dis,
2) + poly(rad, 2) + poly(tax, 2) + poly(ptratio, 2) + poly(black,
2) + poly(lstat, 2), data = trainData)

Residuals:
    Min      1Q  Median      3Q     Max 
-22.8962 -2.1719 -0.1371  1.8323 25.3764 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 22.5106   0.1972 114.143 < 2e-16 ***
poly(crim, 2)1 -38.0606   7.3862  -5.153 4.12e-07 ***
poly(crim, 2)2  16.4345   6.4240   2.558 0.01090 *  
poly(zn, 2)1  -3.3820   7.5876  -0.446 0.65605    
poly(zn, 2)2   8.4794   4.3750   1.938 0.05334 .  
poly(indus, 2)1 3.6202   8.5393   0.424 0.67185    
poly(indus, 2)2  5.8043   6.3912   0.908 0.36436    
poly(nox, 2)1  -56.7798  12.3064  -4.614 5.40e-06 *** 
poly(nox, 2)2   3.2721   6.7800   0.483 0.62965    
poly(rm, 2)1  43.9986   5.8369   7.538 3.50e-13 *** 
poly(rm, 2)2  35.1658   4.8284   7.283 1.88e-12 *** 
poly(age, 2)1   6.9322   8.1659   0.849 0.39645    
poly(age, 2)2   3.5476   4.6824   0.758 0.44913    
poly(dis, 2)1  -51.3516  10.0059  -5.132 4.57e-07 *** 
poly(dis, 2)2   12.4765   5.7688   2.163 0.03118 *  
poly(rad, 2)1  68.9626  22.0153   3.132 0.00187 ** 
poly(rad, 2)2   -5.2775   5.3401  -0.988 0.32365    
poly(tax, 2)1  -40.7625  20.4724  -1.991 0.04718 *  
poly(tax, 2)2   6.0232   8.3002   0.726 0.46848    
poly(ptratio, 2)1 -35.9445   5.8536  -6.141 2.06e-09 *** 
poly(ptratio, 2)2  11.9187   5.3280   2.237 0.02586 *  
poly(black, 2)1 12.7193   4.7348   2.686 0.00754. ** 
poly(black, 2)2  -4.1461   4.2448  -0.977 0.32931    
poly(lstat, 2)1  -88.7852  7.6135 -11.662 < 2e-16 *** 
poly(lstat, 2)2  29.0043   5.2507   5.524 6.14e-08 *** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.979 on 382 degrees of freedom
Multiple R-squared: 0.8228, Adjusted R-squared: 0.8117
F-statistic: 73.91 on 24 and 382 DF, p-value: < 2.2e-16
```

CODE:

```
summary(svm_model)
print(rf_model)
```

OUTPUT:

```
Call:
svm(formula = medv ~ ., data = trainData)

Parameters:
  SVM-Type: eps-regression
  SVM-Kernel: radial
    cost: 1
    gamma: 0.07692308
  epsilon: 0.1

Number of Support Vectors: 269
```

```
Call:
randomForest(formula = medv ~ ., data = trainData)
Type of random forest: regression
Number of trees: 500
No. of variables tried at each split: 4

Mean of squared residuals: 11.59743
% Var explained: 86.17
```

CODE:**# Calculate RMSE for each model**

```
lm_rmse<- rmse(testData$medv - lm_predictions)
poly_rmse<- rmse(testData$medv - poly_predictions)
tree_rmse<- rmse(testData$medv - tree_predictions)
svm_rmse<- rmse(testData$medv - svm_predictions)
rf_rmse<- rmse(testData$medv - rf_predictions)
```

Print RMSEs for each model

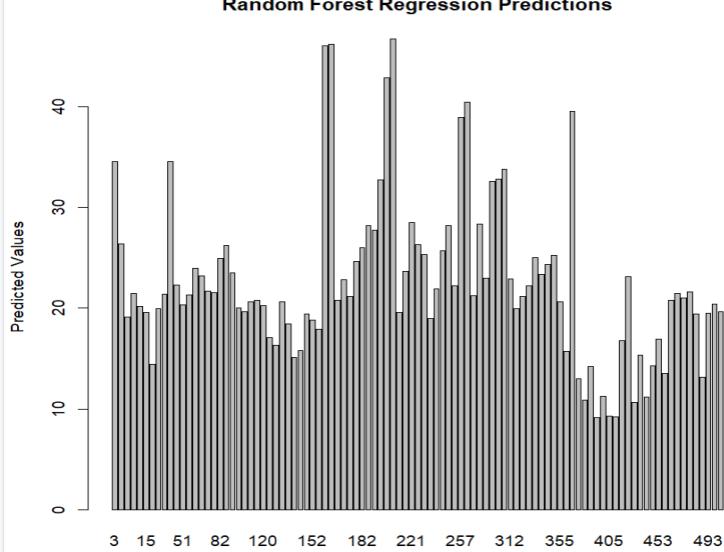
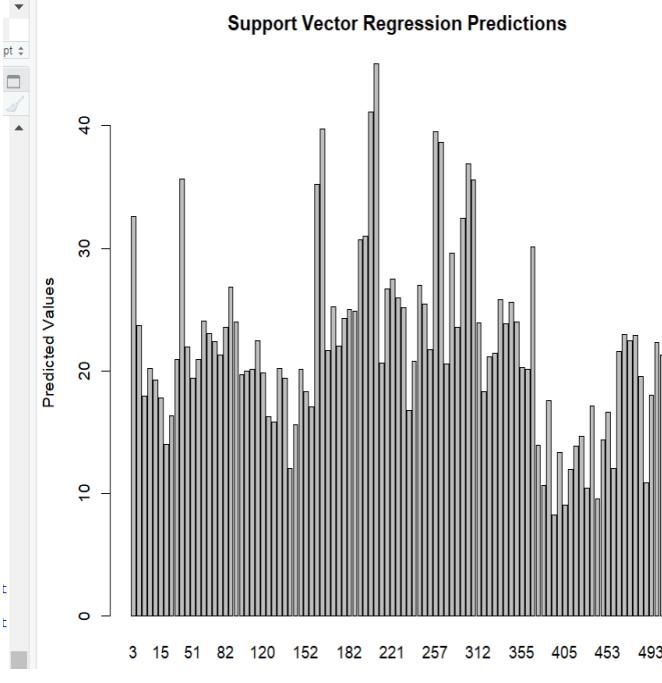
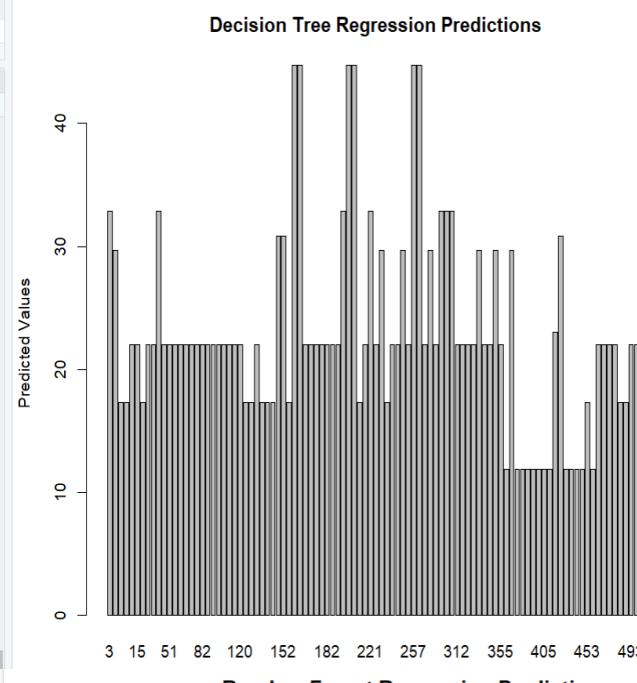
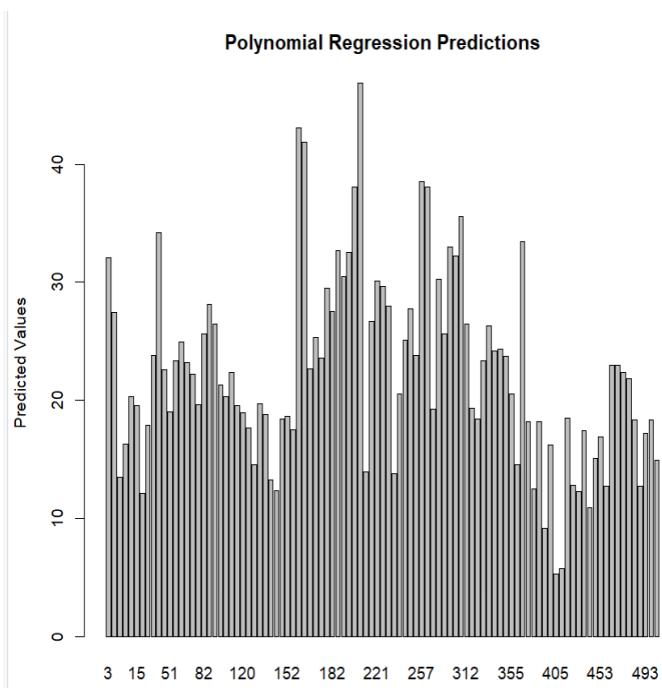
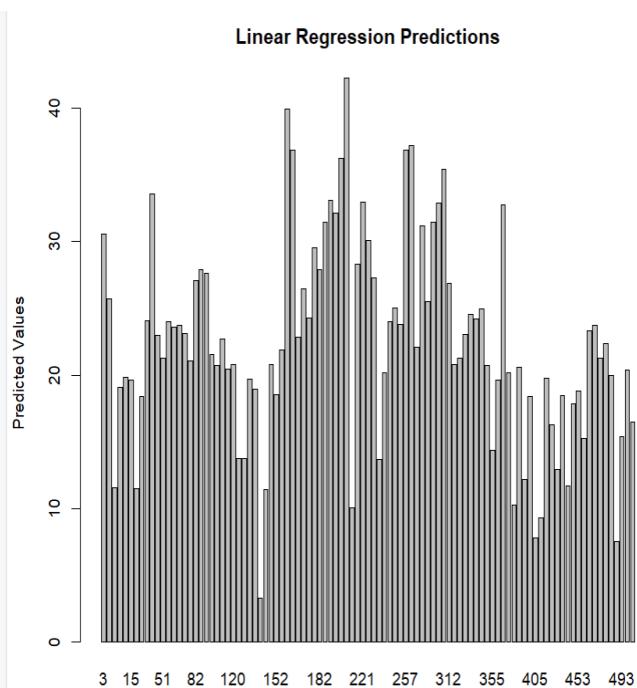
```
cat("Linear Regression RMSE:",lm_rmse,"\n")
cat("Polynomial Regression RMSE:",ploy_rmse,"\n")
cat("Decision Tree Regression RMSE:",tree_rmse,"\n")
cat("Support Vector Regression RMSE:",svm_rmse,"\n")
cat("Random Forest Regression RMSE:",rf_rmse,"\n")
```

OUTPUT:

```
> cat("Linear Regression RMSE:", lm_rmse, "\n")
Linear Regression RMSE: 4.588948
> cat("Polynomial Regression RMSE:", poly_rmse, "\n")
Polynomial Regression RMSE: 3.770119
> cat("Decision Tree Regression RMSE:", tree_rmse, "\n")
Decision Tree Regression RMSE: 4.925522
> cat("Support Vector Regression RMSE:", svm_rmse, "\n")
Support Vector Regression RMSE: 3.91893
> cat("Random Forest Regression RMSE:", rf_rmse, "\n")
Random Forest Regression RMSE: 3.036031
```

CODE:**# Create bar graphs for each model's predictions**

```
models <- c("Linear Regression", "Polynomial Regression", "Decision Tree Regression", "Support Vector Regression",
"Random Forest Regression")
predictions <- list(lm_predictions, poly_predictions, tree_predictions, svm_predictions, rf_predictions)
for (i in 1:length(models)) {
  par(mfrow=c(1,1))
  barplot(predictions[[i]], main = paste(models[i], "Predictions"), ylab = "Predicted Values")}
```

OUTPUT:

EXPERIMENT NO:13

AIM:Create a R program to implement Clustering Comparison on College dataset and compare and visualize the results

DESCRIPTION:The "College" dataset is another commonly used dataset, typically for tasks like clustering and classification. This dataset contains information about various colleges in the United States. The dataset includes features related to the colleges, such as their names, types (public or private), number of students, acceptance rates, and more. One of the common uses of this dataset is for clustering colleges based on their characteristics.

CODE:

```
# Load required libraries
```

```
library(ISLR2)
```

```
library(cluster)
```

```
library(dbSCAN)
```

```
library(factoextra)
```

```
library(mclust)
```

```
library(ggplot2)
```

```
library(kohonen)
```

```
library(e1071) # For FCM
```

```
library(reshape2) # For data manipulation
```

```
data("College") # Load the College dataset
```

```
# Select relevant features for clustering (excluding the college name)
```

```
college_data<- College[, -1] # Exclude the first column (college name)
```

```
scaled_data<- scale(college_data)# Standardize the data
```

```
# Create a function to calculate silhouette scores
```

```
calculate_silhouette<- function(method, data, clusters) {
```

```
  if (!is.null(clusters)) {
```

```
    silhouette_score<- mean(silhouette(clusters, dist(data)))
```

```
    return(silhouette_score)
```

```
  } else {
```

```
    return(NA)
```

```
}
```

```
# Initialize a dataframe to store results
```

```
results <- data.frame(Method = character(0), Silhouette = numeric(0))
```

Perform K-Means clustering

```
kmeans_result<- kmeans(scaled_data, centers = 3, nstart = 25)

kmeans_silhouette<- calculate_silhouette("K-Means", scaled_data, kmeans_result$cluster)

results <- rbind(results, data.frame(Method = "K-Means", Silhouette = kmeans_silhouette))
```

Perform Hierarchical Clustering

```
hierarchical_result<- hclust(dist(scaled_data))

hierarchical_silhouette<- calculate_silhouette("Hierarchical", scaled_data, cutree(hierarchical_result, k = 3))

results <- rbind(results, data.frame(Method = "Hierarchical", Silhouette = hierarchical_silhouette))
```

Perform DBSCAN clustering

```
dbscan_result<- dbSCAN(scaled_data, eps = 3, MinPts = 5)

dbscan_silhouette<- calculate_silhouette("DBSCAN", scaled_data, dbscan_result$cluster)

results <- rbind(results, data.frame(Method = "DBSCAN", Silhouette = dbscan_silhouette))
```

Perform Gaussian Mixture Model (GMM) clustering

```
gmm_result<- Mclust(scaled_data, G=3)
```

OUTPUT:

```
|> gmm_result<- Mclust(scaled_data, G=3)
| fitting ...
| ====== | 100%
```

```
gmm_silhouette<- calculate_silhouette("GMM", scaled_data, gmm_result$classification)

results <- rbind(results, data.frame(Method = "GMM", Silhouette = gmm_silhouette))
```

Perform Self-Organizing Maps (SOM) clustering

```
som_result<- som(scale(scaled_data), grid = somgrid(3, 3, "hexagonal"))

som_silhouette<- calculate_silhouette("SOM", scale(scaled_data), som_result$unit.classif)

results <- rbind(results, data.frame(Method = "SOM", Silhouette = som_silhouette))
```

Perform Fuzzy C-Means (FCM) clustering

```
fcm_result<- cmeans(scaled_data, centers = 3, m = 1.2)

fcm_clusters<- fcm_result$cluster

fcm_silhouette<- calculate_silhouette("FCM", scaled_data, fcm_clusters)

results <- rbind(results, data.frame(Method = "FCM", Silhouette = fcm_silhouette))

Print(results)
```

OUTPUT:

```
<- results ~ read.csv(results, data=...>
> print(results)
      Method Silhouette
1     K-Means  1.6405589
2 Hierarchical  1.1458327
3       DBSCAN  0.4602613
4        GMM  1.1020827
5        SOM  2.4366948
6       FCM  1.6403808
> |
```

CODE:

```
# Find the index of the method with the highest silhouette score
```

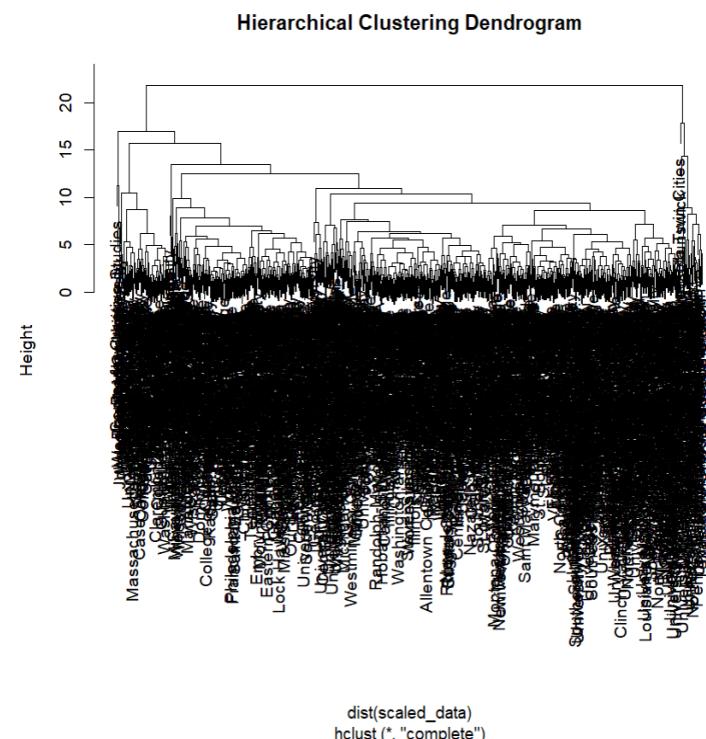
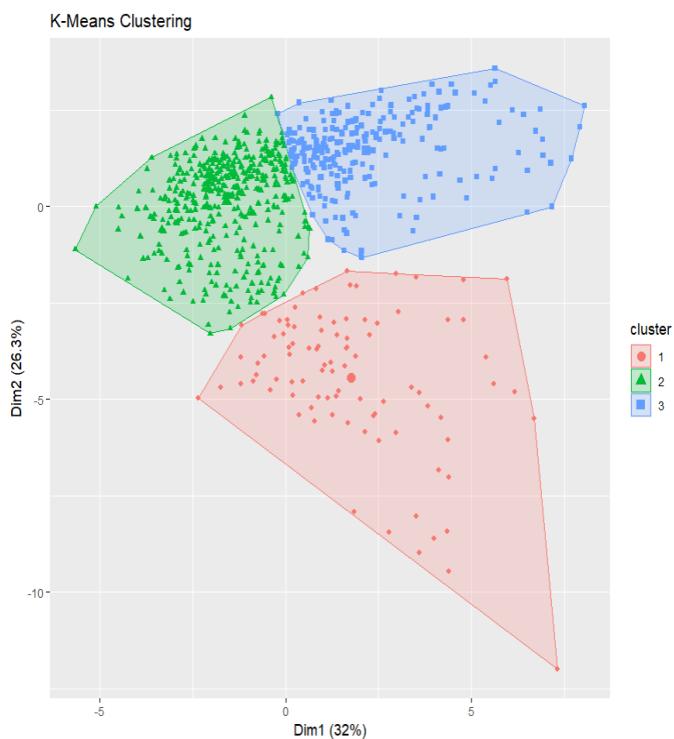
```
best_method_index<- which.max(results$Silhouette)
```

```
best_method<- results$Method[best_method_index] # Retrieve the best method based on the index
```

```
# Visualize K-Means clustering
```

```
fviz_cluster(kmeans_result, data = scaled_data, geom = "point", main = "K-Means Clustering")
```

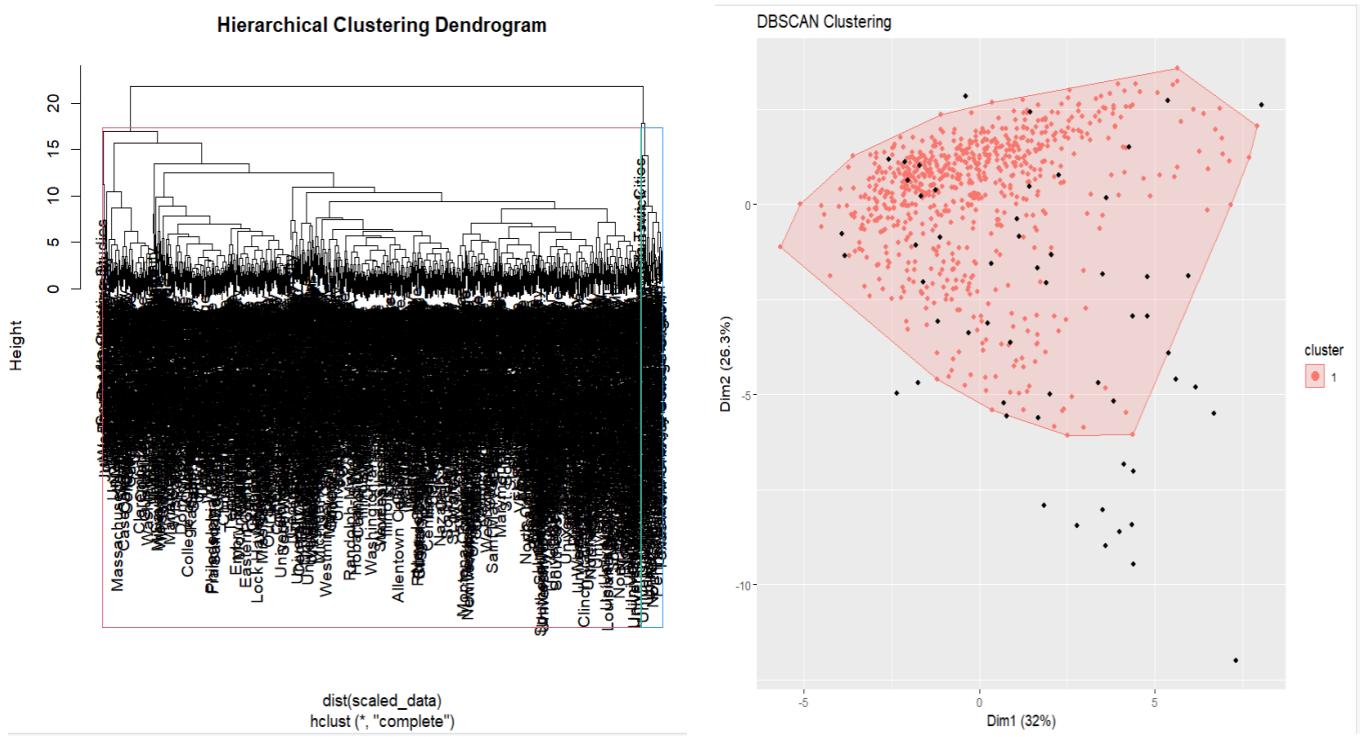
```
plot(hierarchical_result, main = "Hierarchical Clustering Dendrogram") # Visualize Hierarchical Clustering
```

OUTPUT:**CODE:**

```
rect.hclust(hierarchical_result, k = 3, border = 2:4)
```

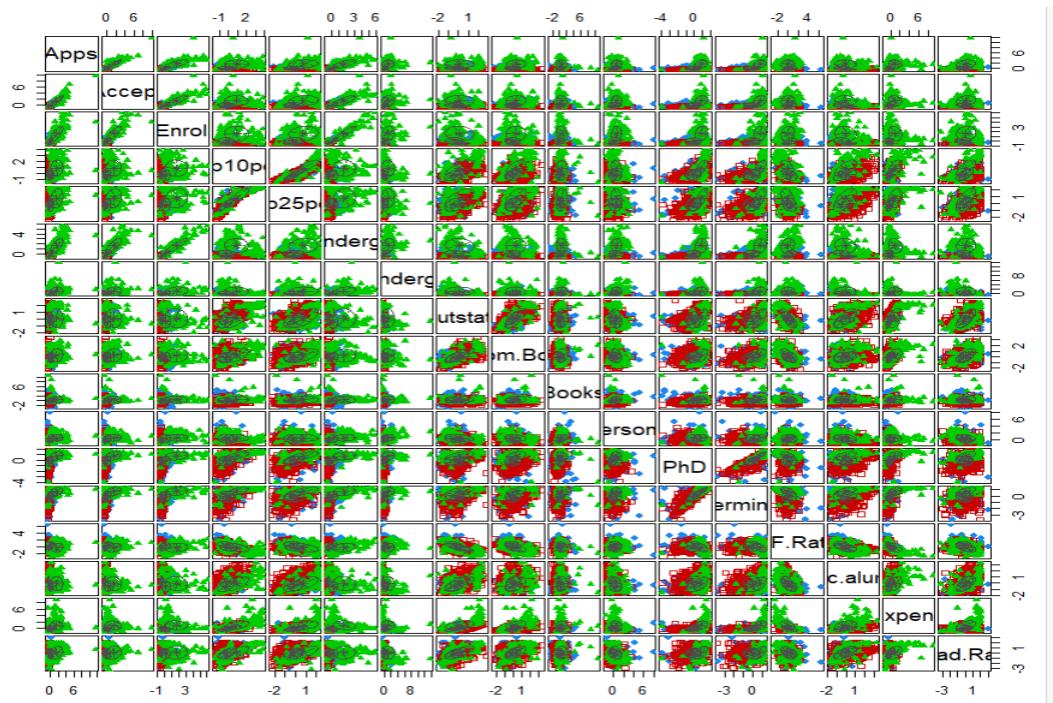
```
# Visualize DBSCAN clustering
```

```
fviz_cluster(dbSCAN_result, data = scaled_data, geom = "point", main = "DBSCAN Clustering")
```

OUTPUT:**CODE:**

```
# Visualize GMM clustering
```

```
plot(gmm_result, what = "classification", main = "GMM Clustering")
```

OUTPUT:**CODE:**

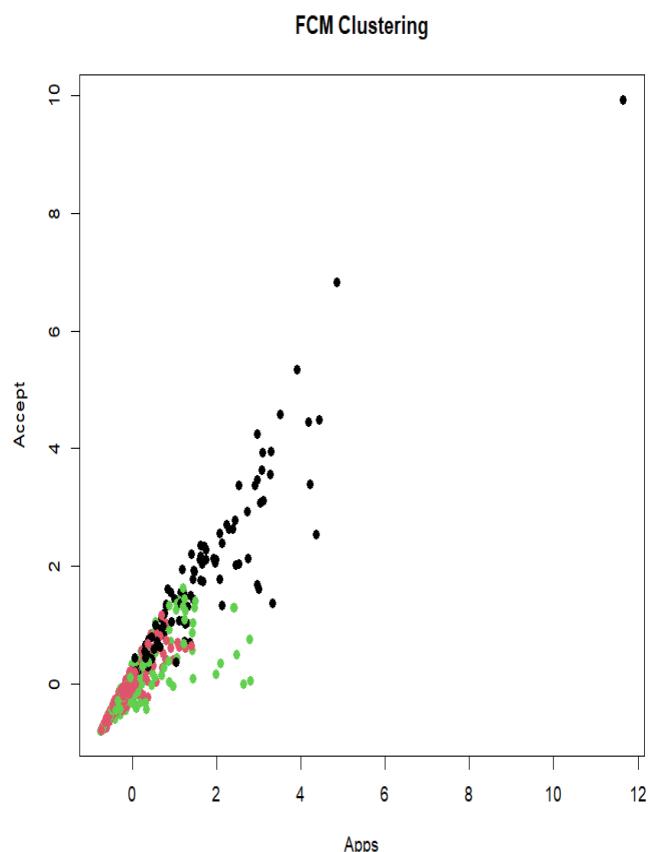
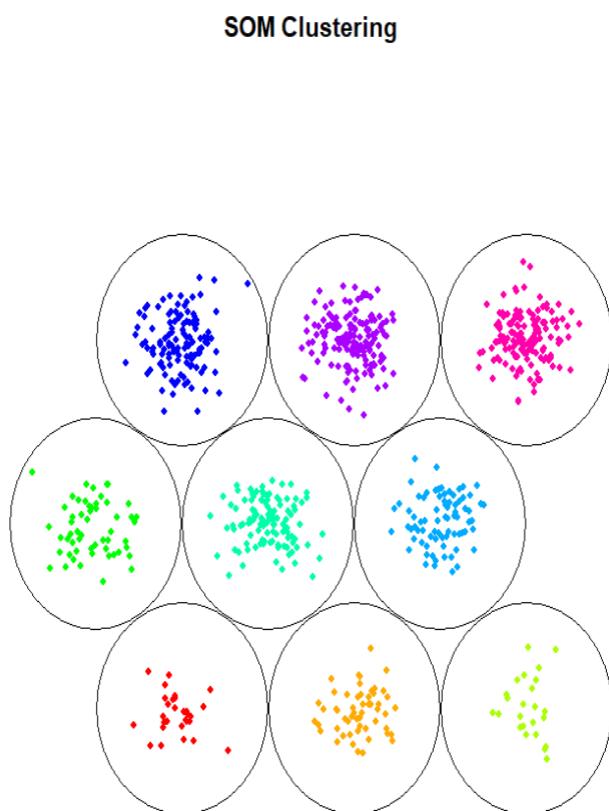
```
# Perform Self-Organizing Maps (SOM) clustering
som_result<- som(scale(scaled_data), grid = somgrid(3, 3, "hexagonal"))
cat("SOM Cluster Summary:\n")
print(som_result)
```

OUTPUT:

```
> cat("SOM Cluster Summary:\n")
SOM Cluster Summary:
> print(som_result)
SOM of size 3x3 with a hexagonal topology.
Training data included.
> |
```

CODE:

```
cluster_assignments<- som_result$unit.classif# Get cluster assignments from the SOM result
unique_clusters<- unique(cluster_assignments)# Create a vector of unique cluster IDs
cluster_colors<- rainbow(length(unique_clusters))# Create a color palette for the clusters
assigned_colors<- cluster_colors[cluster_assignments]# Map cluster assignments to colors based on cluster IDs
# Visualize SOM Clustering with assigned colors
plot(som_result, type = "mapping", pchs = 20, col = assigned_colors, main = "SOM Clustering")
plot(scaled_data, col = fcm_clusters, pch = 19, main = "FCM Clustering") # Plot FCM clustering assignments
```

OUTPUT:**CODE:**

```
# Print the best method and its silhouette score
Cat("Best Clustering Method:",best_method,"\n")
Cat("Silhouette score:",results$Silhouette[best_method_index],"\n")
```

OUTPUT:

```
<-- plotSilhouette_data, col = true_clusters, pch = 15, main = "Best Clustering"
> cat("Best Clustering Method:",best_method,"\n")
Best Clustering Method: SOM
> cat("Silhouette Score:",results$Silhouette[best_method_index],"\n")
Silhouette Score: 2.436695
```

CODE:

Create a bar plot of silhouette scores with different colors for each bar

```
ggplot(results, aes(x = Method, y = Silhouette, fill = Method)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = round(Silhouette, 2), vjust = -0.5), position = position_dodge(0.9)) + # Add labels on top of bars
  labs(title = "Silhouette Scores for Clustering Algorithms", y = "Silhouette Score") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) # Rotate x-axis labels for better readability
```

OUTPUT: