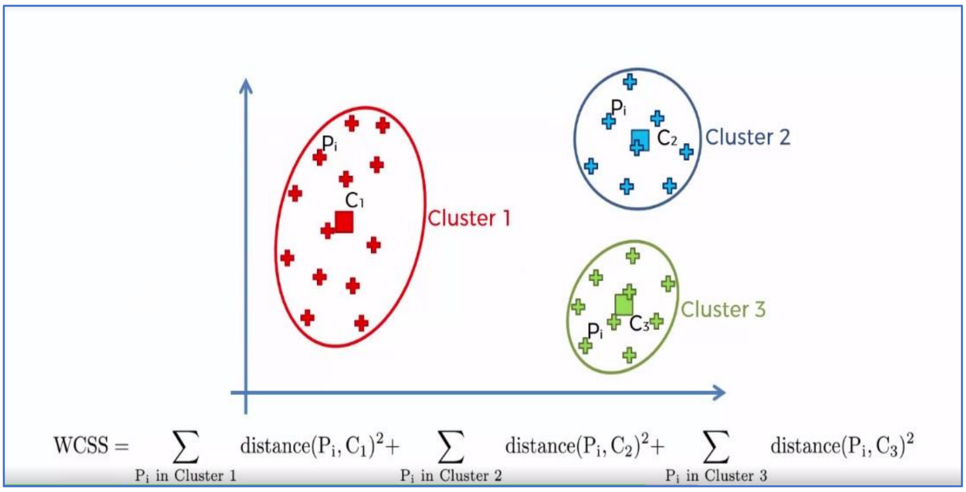
INDEX

PRACTICAL NO.1: K-MEANS CLUSTERING

FILE USED: Mall_Customers.csv

CODE:

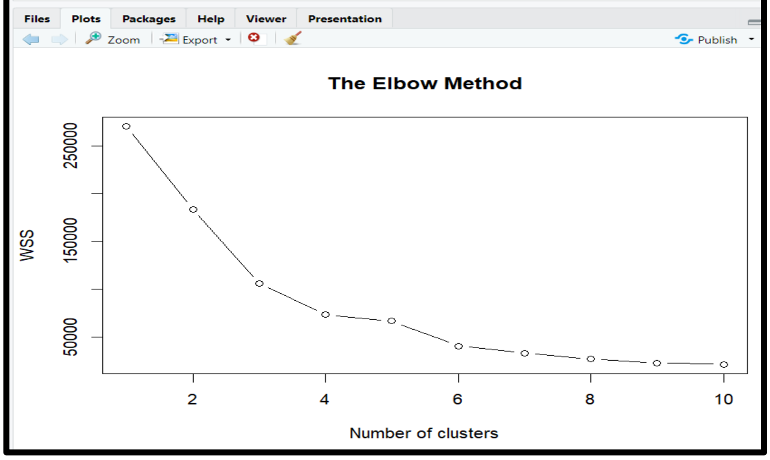
```
# Importing the dataset
dataset = read.csv('E:\\BDAPractical\\Mall_Customers.csv')
#Displays contents-6 rows by default
head(dataset)
#Column 4th & 5th
dataset = dataset[4:5]
#Displays contents-6 rows by default
head(dataset)
#within cluster-sum of squares
wcss = vector()
```



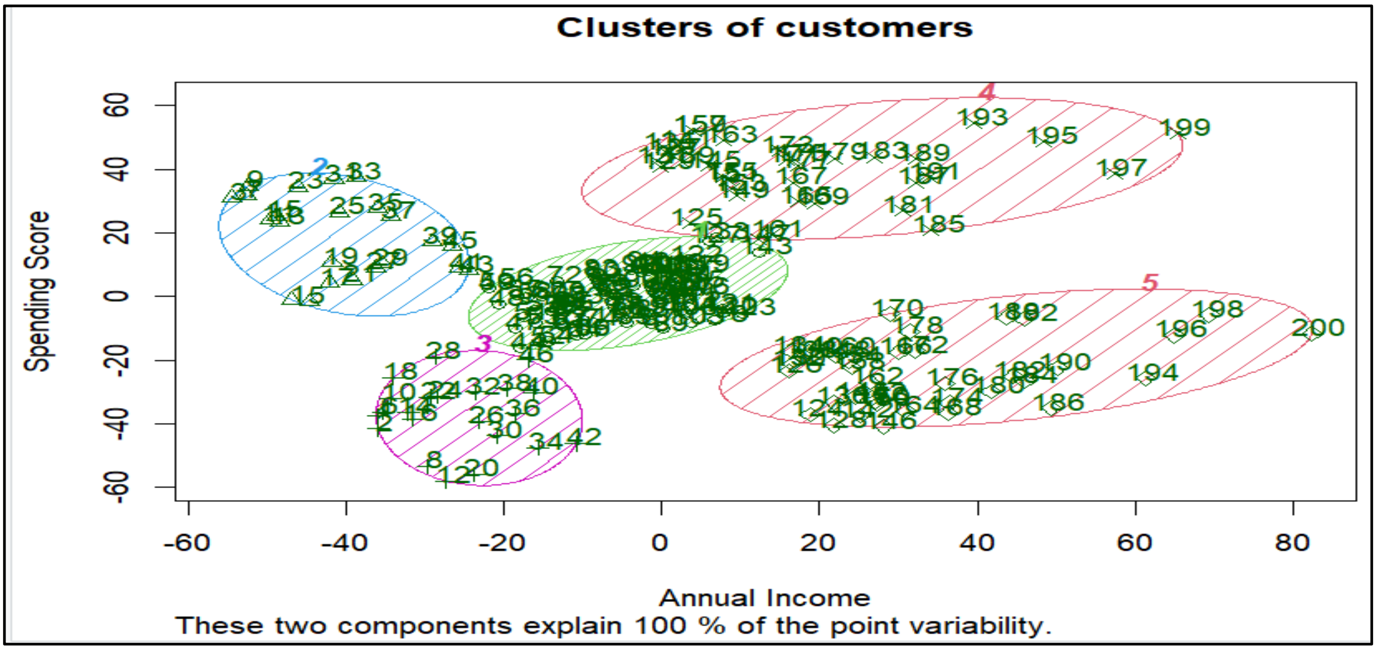

```
#It results in a vector with a number for each cluster
for (i in 1:10) wcss[i] = sum(kmeans(dataset, i)$withinss)
#plot(x,y,type='b' indicates both points and lines)
plot(1:10,
     wcss,
     type = 'b',
     main = paste('The Elbow Method'),
     xlab = 'Number of clusters',
     ylab = 'WSS')

# Fitting K-Means to the dataset with no of clusters = 5
kmeans = kmeans(x = dataset, centers = 5)
#vector of number ranging from 1 to 5

y_kmeans = kmeans$cluster->
# Visualising the clusters
library(cluster)
clusplot(dataset,->x
        y_kmeans,->y
        lines = 0,-> no distance lines will appear on plot
        shade = TRUE,-> according to density
        color = TRUE, -> according to density
        labels = 2,-> all points and ellipses are labelled in the plot
        main = paste('Clusters of customers'),
        xlab = 'Annual Income',
        ylab = 'Spending Score')
```

```
> dataset = read.csv('E:\\BDAPractical\\Mall_Customers.csv')
> head(dataset)
  CustomerID  Genre Age Annual.Income..k.. Spending.Score..1.100.
1          1   Male  19                 15                     39
2          2   Male  21                 15                     81
3          3 Female  20                 16                      6
4          4 Female  23                 16                     77
5          5 Female  31                 17                     40
6          6 Female  22                 17                     76
> dataset = dataset[4:5]
> head(dataset)
  Annual.Income..k.. Spending.Score..1.100.
1                 15                     39
2                 15                     81
3                 16                      6
4                 16                     77
5                 17                     40
6                 17                     76
```

```
Data
  dataset      200 obs. of 2 variables
  kmeans       List of  9
Values
  i            10L
  wcss         num [1:10] 269981 183653 106348 73680 66665 ...
  y_kmeans     int [1:200] 2 4 2 4 2 4 2 4 2 4 ...
```



OUTPUT:



PRACTICAL NO.2: APRIORI ALGORITHM

FILE USED: Groceries (Inbuilt)

CODE:

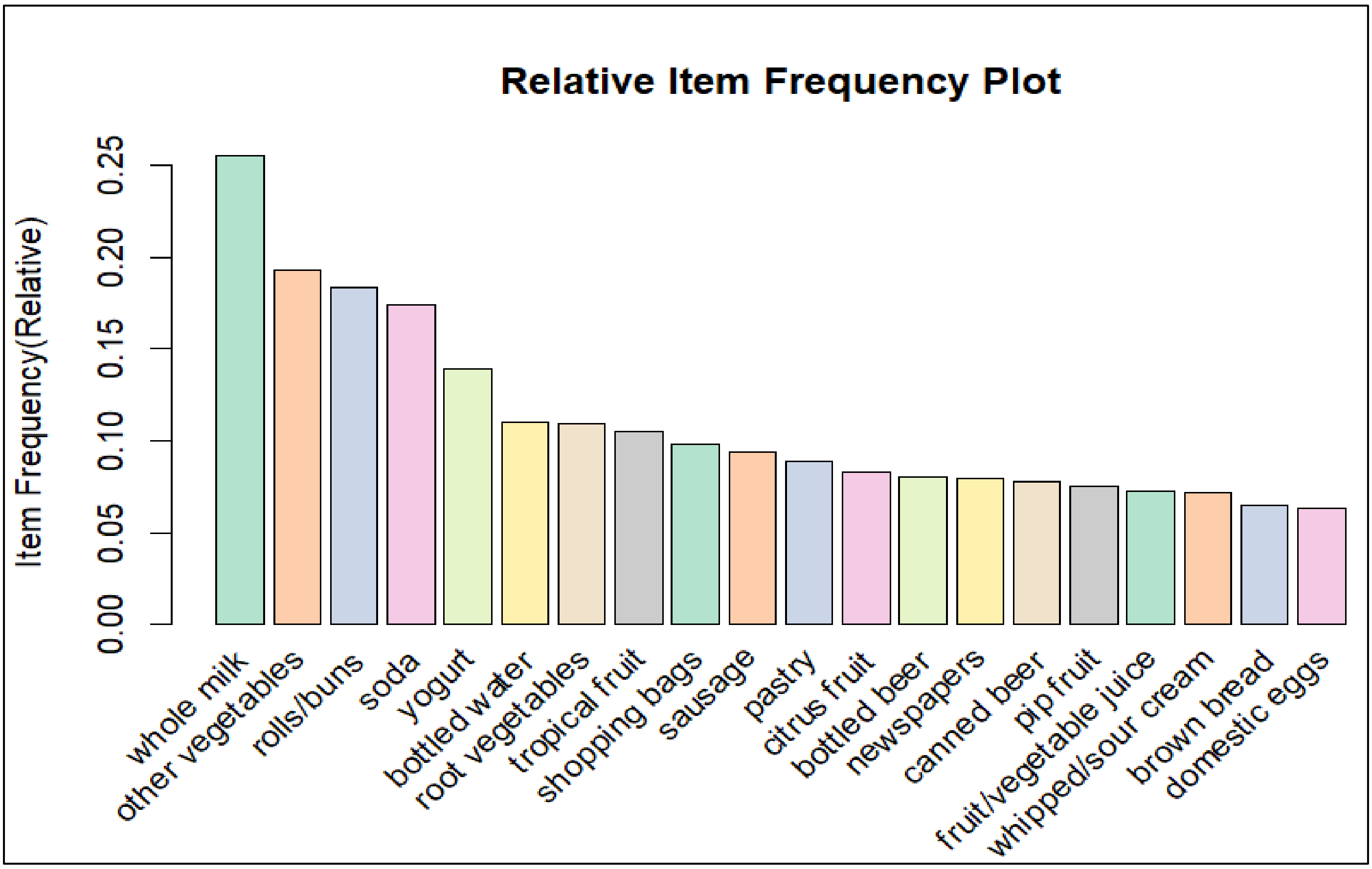
install.packages("arules") ⇒Representing, Manipulating & Analyzing Data & Patterns
install.packages("arulesViz") ⇒Visualizing Association Rules & Frequent Itemsets
install.packages("RColorBrewer") ⇒Palette that provides color schemes for maps and other graphics
library(arules)
library(arulesViz)
library(RColorBrewer)
data("Groceries") ⇒Predefined in RPackage with 9835 records
Groceries ⇒Display ->9835 rows-> 169 cols
summary(Groceries) ⇒Gives statistical information of dataset
class(Groceries) ⇒Transaction (Returns class attribute of dataset provided)
rules = apriori(Groceries, parameter = list(supp = 0.02, conf = 0.2))
⇒Transaction Data, Min. Support, Min. Confidence, For Itemset-1
summary(rules) ⇒Gives statistical information of rules
inspect(rules[1:10]) ⇒Prints the first 10 strong association rules
arules::itemFrequencyPlot(Groceries, topN = 20, ⇒Bar Plot for itemFrequencies, 20 items will be plotted
col = brewer.pal(8, 'Pastel2'), 8-> color with repetition (3 to 8), Pastel2->color scheme
main = 'Relative Item Frequency Plot',
type = "relative", ⇒Considers %value
ylab = "Item Frequency(Relative)")
itemset = apriori(Groceries, parameter = list(minlen=2, maxlen=2, support=0.02, target="frequent itemset"))
⇒Transaction Data, Get itemset of Length 2, For Itemset-2
summary(itemset)
inspect(itemset[1:10]) ⇒10 values
itemsets_3 = apriori(Groceries, parameter = list(minlen=3, maxlen=3, support=0.02, target="frequent itemset"))
⇒Transaction Data, Get itemset of Length 3, For Itemset-3
summary(itemsets_3)
inspect(itemsets_3)
⇒2 values as a result displayed

```
                                                    call
 apriori(data = Groceries, parameter = list(supp = 0.02, conf = 0.2))
> inspect(rules[1:10])
     lhs                    rhs                support    confidence coverage   lift
[1]  {}                  => {whole milk}       0.25551601 0.2555160  1.00000000 1.0000000
[2]  {frozen vegetables} => {whole milk}       0.02043721 0.4249471  0.04809354 1.6630940
[3]  {beef}              => {whole milk}       0.02125064 0.4050388  0.05246568 1.5851795
[4]  {curd}              => {whole milk}       0.02613116 0.4904580  0.05327911 1.9194805
[5]  {pork}              => {other vegetables} 0.02165735 0.3756614  0.05765125 1.9414764
[6]  {pork}              => {whole milk}       0.02216573 0.3844797  0.05765125 1.5047187
[7]  {frankfurter}       => {whole milk}       0.02053889 0.3482759  0.05897306 1.3630295
[8]  {bottled beer}      => {whole milk}       0.02043721 0.2537879  0.08052872 0.9932367
[9]  {brown bread}       => {whole milk}       0.02521607 0.3887147  0.06487036 1.5212930
[10] {margarine}         => {whole milk}       0.02419929 0.4131944  0.05856634 1.6170980
```

```
call
 apriori(data = Groceries, parameter = list(minlen = 2, maxlen = 2, support = 0.02, target = "frequ
ent itemset"))
> inspect(itemset[1:10])
     items                        support    count
[1]  {whole milk, frozen vegetables} 0.02043721 201
[2]  {beef, whole milk}              0.02125064 209
[3]  {whole milk, curd}              0.02613116 257
[4]  {pork, other vegetables}        0.02165735 213
[5]  {pork, whole milk}              0.02216573 218
[6]  {frankfurter, whole milk}       0.02053889 202
[7]  {whole milk, bottled beer}      0.02043721 201
[8]  {whole milk, brown bread}       0.02521607 248
[9]  {whole milk, margarine}         0.02419929 238
[10] {other vegetables, butter}      0.02003050 197
```

```
call
 apriori(data = Groceries, parameter = list(minlen = 3, maxlen = 3, support = 0.02, target = "frequ
ent itemset"))
> inspect(itemsets_3)
    items                                        support    count
[1] {root vegetables, other vegetables, whole milk} 0.02318251 228
[2] {other vegetables, whole milk, yogurt}          0.02226741 219
```

OUTPUT:



PRACTICAL NO.3: REGRESSION

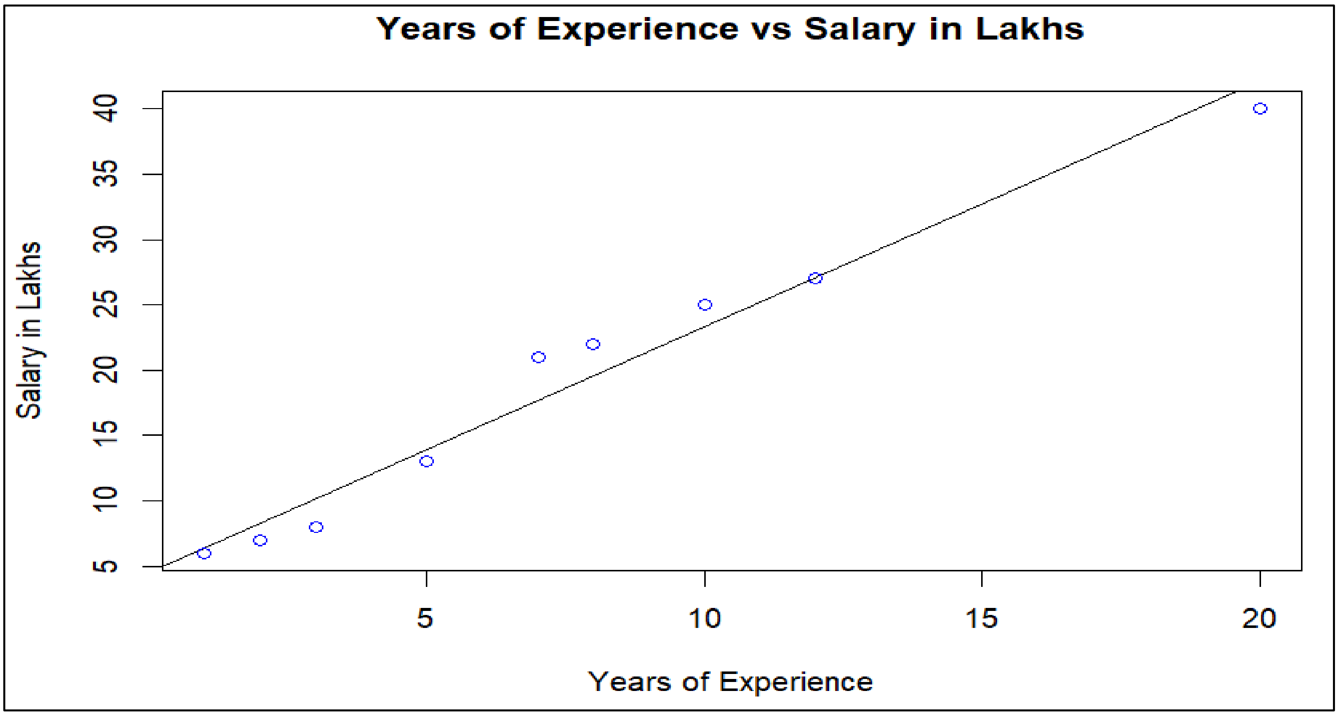
A. LINEAR REGRESSION

CODE:

years_of_exp = c(7,5,1,3,5,8,10,12,20,2) ⇒Combine Values in a vector/list
salary_in_lakhs = c(21,13,6,8,13,22,25,27,40,7)
employee.data = data.frame(years_of_exp, salary_in_lakhs)⇒Data displayed in table format
employee.data
Estimation of the salary of an employee, based on his year of experience.
model <- lm(salary_in_lakhs ~ years_of_exp, data = employee.data)
⇒linear models, formula, data (relationship between both)
summary(model)
Visualization of Regression
plot(years_of_exp,salary_in_lakhs,data = employee.data,col = "blue",main = "Years of Experience vs Salary in Lakhs", abline(model),xlab = "Years of Experience",ylab = "Salary in Lakhs")
⇒color of bubble
⇒vertical/ horizontal/ regression lines to a graph

OUTPUT:

```
> employee.data
   years_of_exp salary_in_lakhs
1             7              21
2             5              13
3             1               6
4             3               8
5             5              13
6             8              22
7            10              25
8            12              27
9            20              40
10            2               7
```



B. LOGISTIC REGRESSION

FILE USED: studentmarks.csv

CODE:

#fetch the data
college <- read.csv("E:\\\\BDAPractical\\\\studentmarks.csv")⇒ Assigned to college (400 rows, 4 cols)
head(college) ⇒ First 6 values
nrow(college) ⇒ Number of rows
install.packages("caTools") # For Logistic regression⇒ Contains several basic utility functions
library(caTools)
split <- sample.split(college, SplitRatio = 0.75) ⇒75% Training Set, 25% Testing Set
split

training_reg <- subset(college, split == "TRUE")
test_reg <- subset(college, split == "FALSE")
⇒admit gre gpa rank

Training model
fit_logistic_model <- glm(admit ~ ., ⇒Generalized Linear Model
data = training_reg,
family = "binomial")⇒Dependent variable is binary

Predict test data based on model
predict_reg <- predict(fit_logistic_model,
test_reg, type = "response")⇒Output probabilities in normal scale
predict_reg

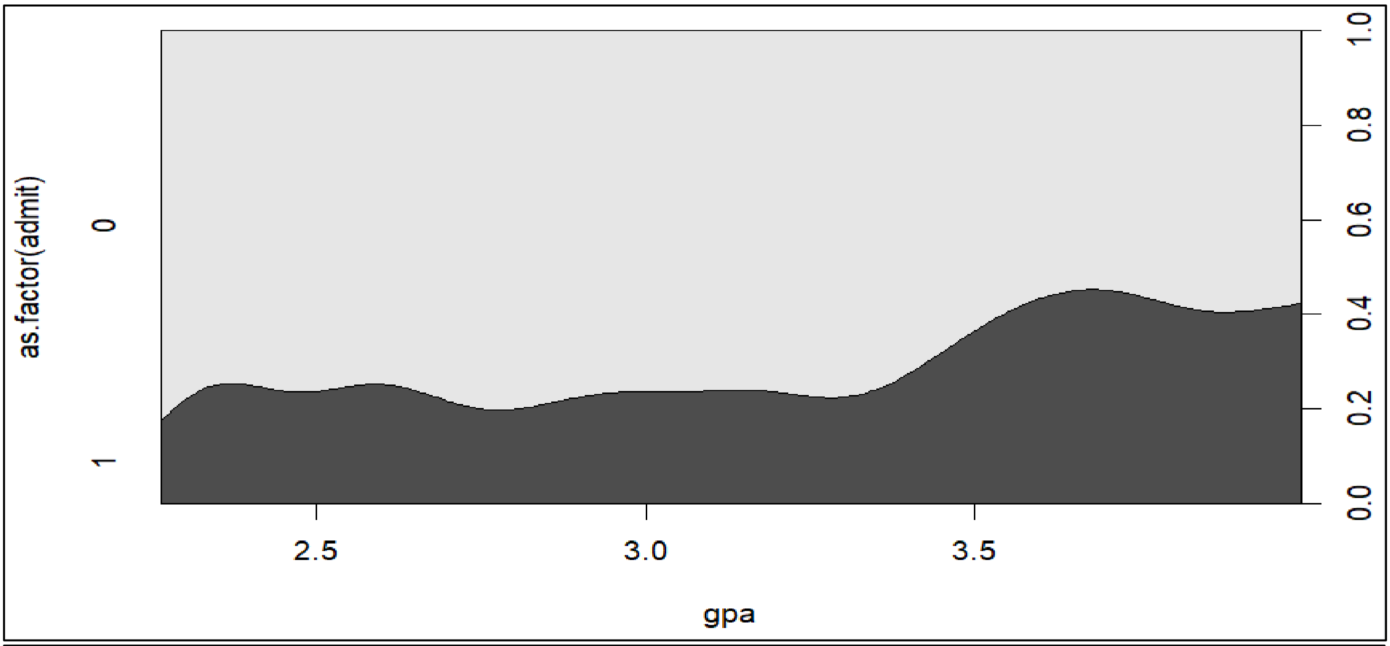
cdplot(as.factor(admit)~ gpa, data=college) ⇒Conditional Densities of y changes over variable x
cdplot(as.factor(admit)~ gre, data=college)
cdplot(as.factor(admit)~ rank, data=college)
⇒single value/ returns factor object

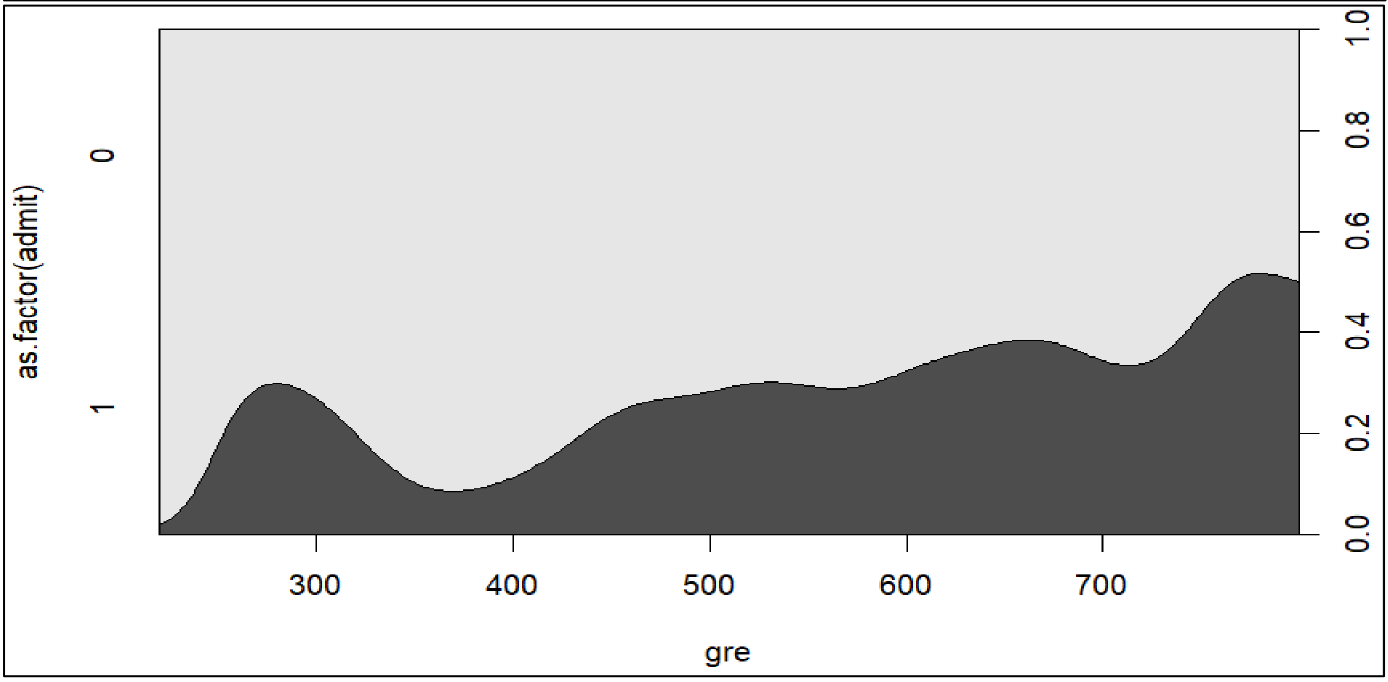
Changing probabilities
predict_reg <- ifelse(predict_reg >0.5, 1, 0) ⇒Conditional Operator
predict_reg

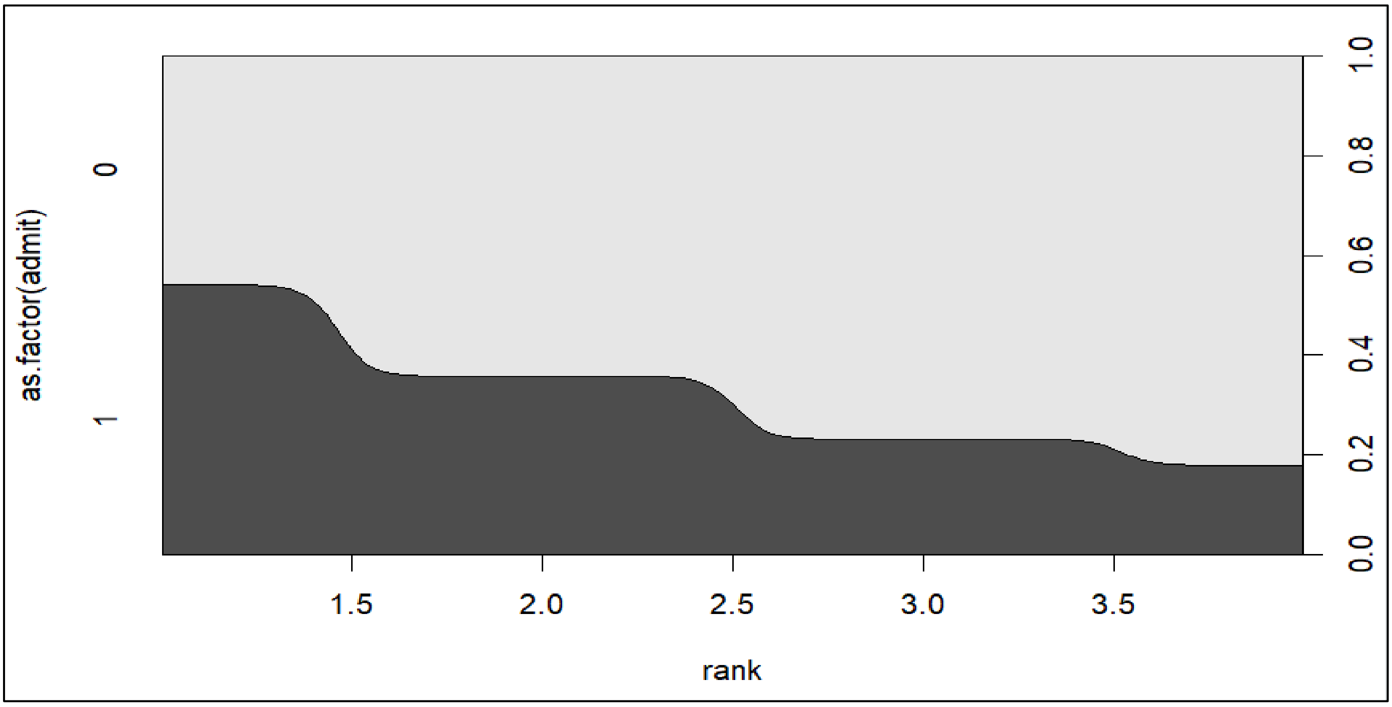
Evaluating model accuracy
using confusion matrix
table(test_reg$admit, predict_reg)
⇒Total 100 values

OUTPUT:

```
> table(test_reg$admit, predict_reg)
   predict_reg
     0  1
  0 53 12
  1 26  9
```


as.factor(admit)
0
1
2.5
3.0
3.5
gpa
0.0
0.2
0.4
0.6
0.8
1.0


as.factor(admit)
0
1
300
400
500
600
700
gre
0.0
0.2
0.4
0.6
0.8
1.0


as.factor(admit)
0
1
1.5
2.0
2.5
3.0
3.5
rank
0.0
0.2
0.4
0.6
0.8
1.0

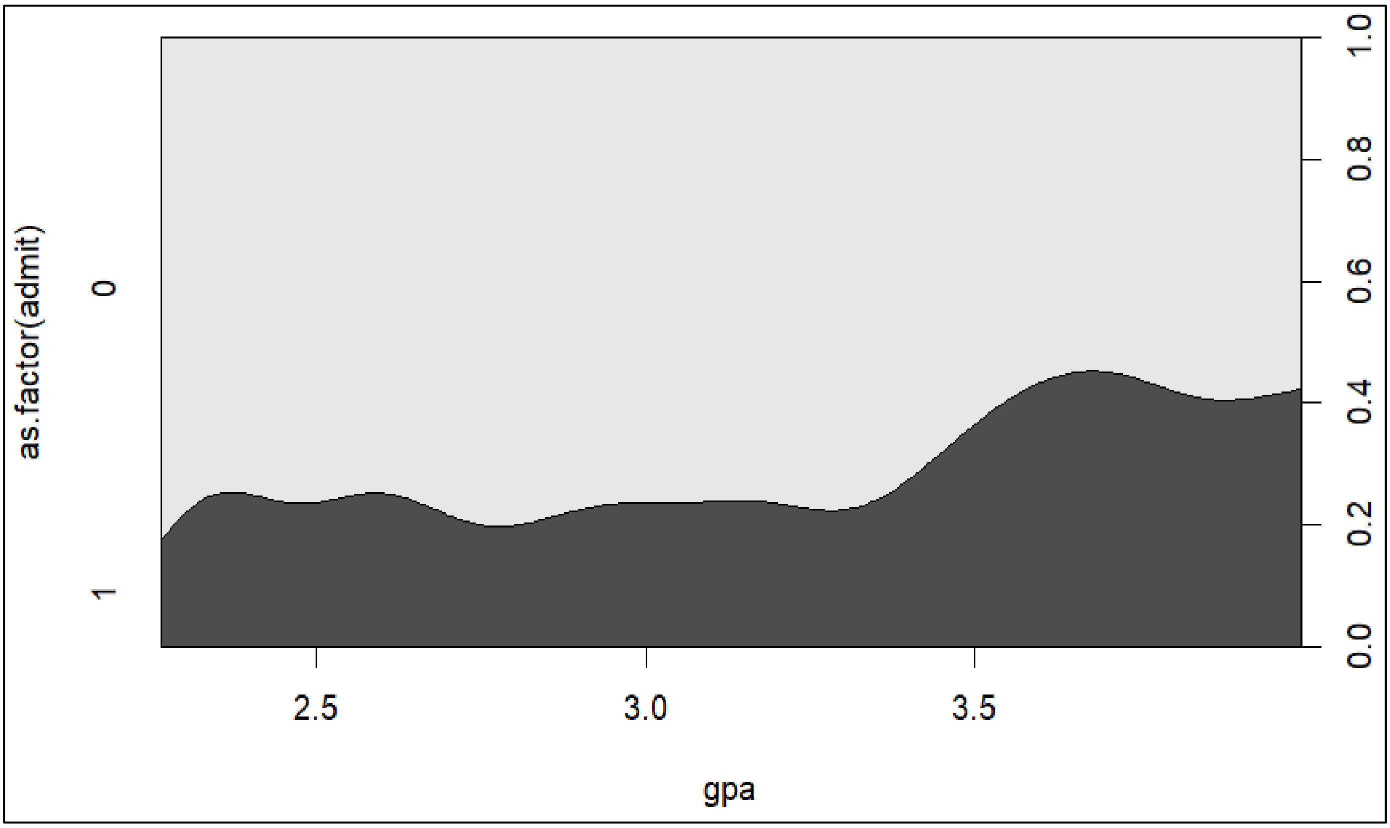
C. MULTIPLE REGRESSION

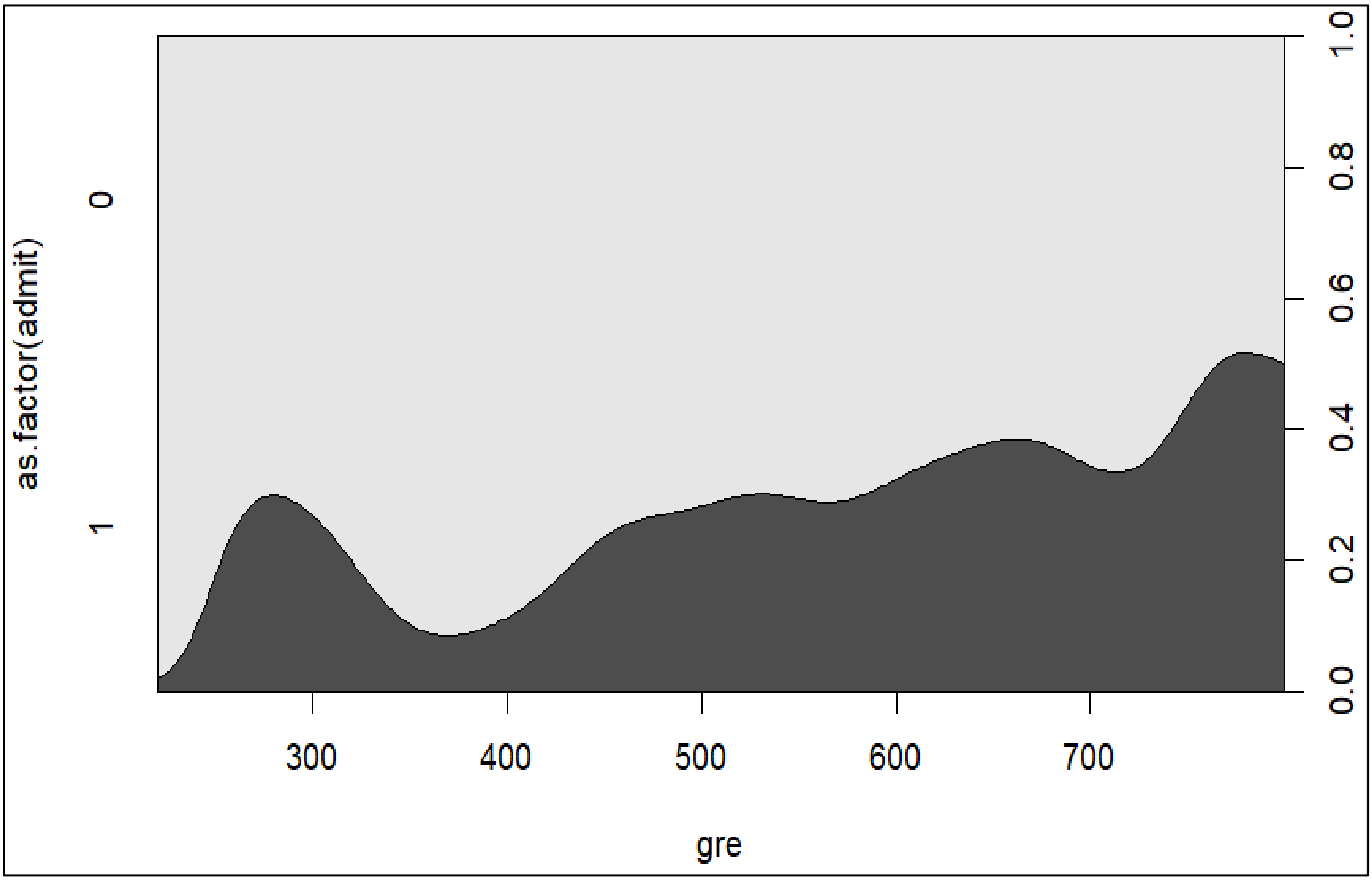
FILE USED: studentmarks.csv

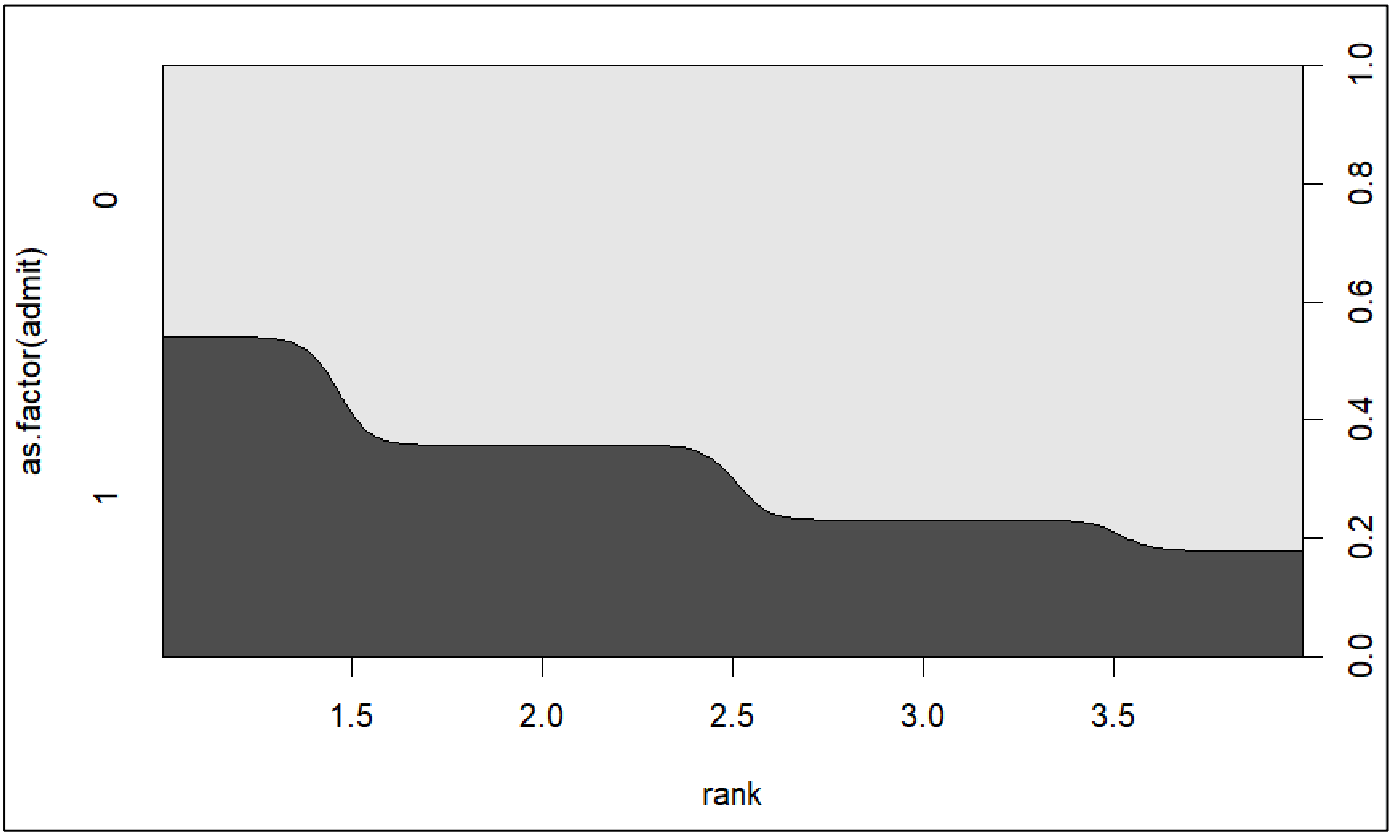
CODE:

```
#fetch the data
StudentData <- read.csv("E:\\BDAPractical\\studentmarks.csv")
head(StudentData)
nrow(StudentData)
install.packages("caTools")    # For Logistic regression
library(caTools)
split <- sample.split(StudentData, SplitRatio = 0.75)
split
training_reg <- subset(StudentData, split == "TRUE")
test_reg <- subset(StudentData, split == "FALSE")

# Training model
fit_MRegressor_model <- lm(formula = admit ~ gre+gpa+rank,
                    data = training_reg)
# Predict test data based on model
predict_reg <- predict(fit_MRegressor_model,
                     newdata = test_reg)
predict_reg
cdplot(as.factor(admit)~ gpa, data=StudentData)
cdplot(as.factor(admit)~ gre, data=StudentData)
cdplot(as.factor(admit)~ rank, data=StudentData)
```

OUTPUT:




as.factor(admit)
0
1
0.0
0.2
0.4
0.6
0.8
1.0
300
400
500
600
700
gre


as.factor(admit)
0
1
0.0
0.2
0.4
0.6
0.8
1.0
1.5
2.0
2.5
3.0
3.5
rank

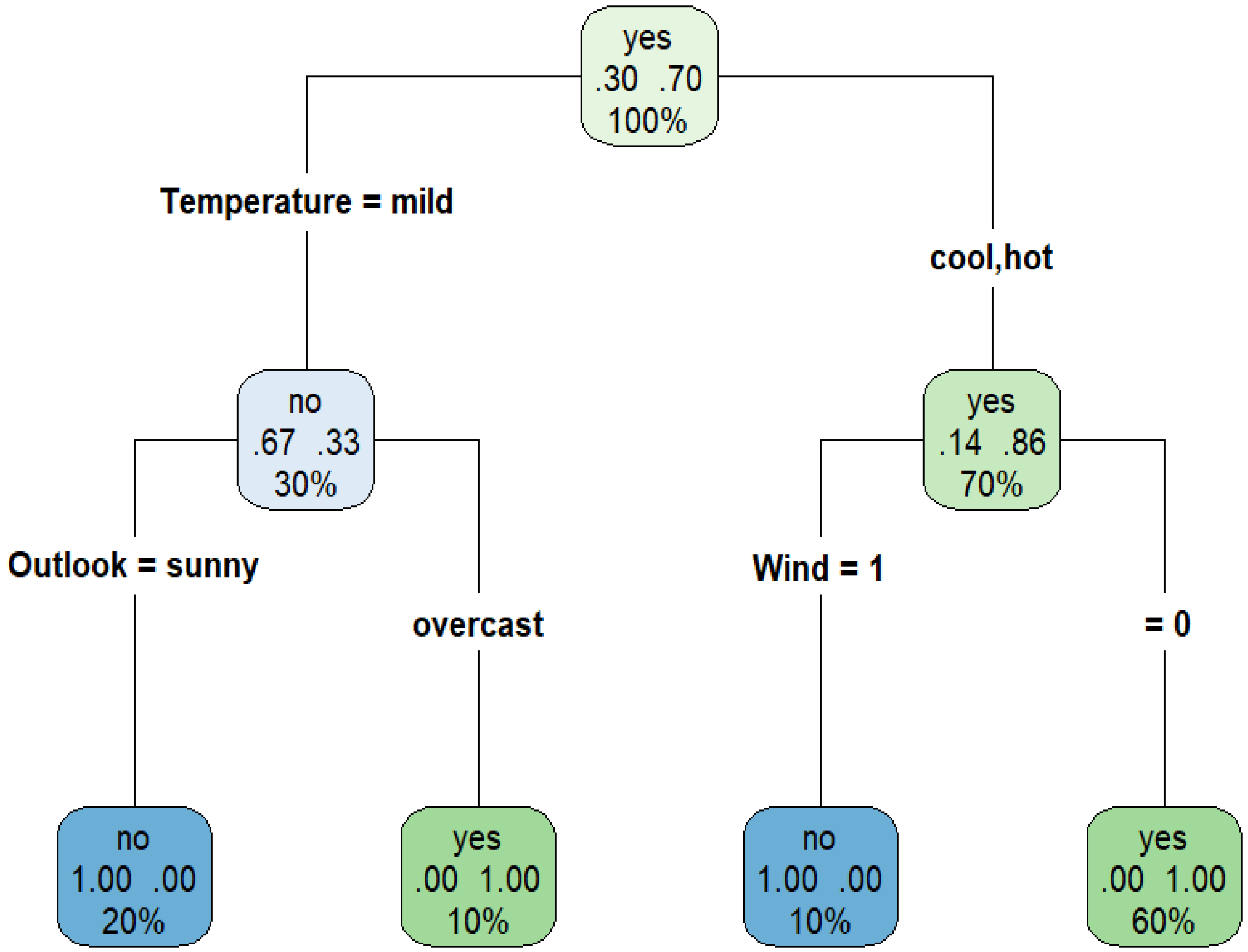
PRACTICAL NO.4: DECISION TREE

FILE USED: DTdata.csv

CODE:

```
install.packages("rpart") #install packages for modeling decision trees
install.packages("rpart.plot") #install packages for plotting decision trees
#load libraries
library(rpart)
library(rpart.plot)
#header and sep for proper alignment and indentation
play_decision<-read.table("E:\\BDAPractical\\DTdata.csv",header=TRUE,sep=",")
play_decision
summary(play_decision)
#Play-OutputVariable, Outlook+Temperature+Humidity+Wind-InputVariable
#rpart(formula, classification tree, dataset, controls the tree growth, purity measure-with split either information or gini
fit<- rpart (Play~Outlook+Temperature+Humidity+Wind, method="class", data=play_decision,
control=rpart.control (minsplit=1),parms=list(split='information'))
summary(fit)
rpart.plot(fit,type=4,extra=104)
```

OUTPUT:



```
> play_decision
   Play  Outlook Temperature Humidity  Wind
1   yes    rainy        cool   normal FALSE
2    no    rainy        cool   normal  TRUE
3   yes overcast         hot     high FALSE
4    no    sunny        mild     high FALSE
5   yes    rainy        cool   normal FALSE
6   yes    sunny        cool   normal FALSE
7   yes    rainy        cool   normal FALSE
8   yes    sunny         hot   normal FALSE
9   yes overcast        mild     high  TRUE
10   no    sunny        mild     high  TRUE
```

1.

Play	Outlook	Temperature	Humidity	Wind
yes	rainy	cool	normal	FALSE
yes	overcast	hot	high	FALSE
yes	rainy	cool	normal	FALSE
yes	sunny	cool	normal	FALSE
yes	rainy	cool	normal	FALSE
yes	sunny	hot	normal	FALSE
yes	overcast	mild	high	TRUE

100% data

Play=yes
Left(No)-3/10=0.30
Right(Yes)-7/10=0.70

2.

Play	Outlook	Temperature	Humidity	Wind
no	sunny	mild	high	FALSE
yes	overcast	mild	high	TRUE
no	sunny	mild	high	TRUE

Temperature=mild
Total Values=3 (30%) with
Left-2 play(no)=0.67,
Right-1 play(yes)=0.33

3.

Play	Outlook	Temperature	Humidity	Wind
yes	rainy	cool	normal	FALSE
no	rainy	cool	normal	TRUE
yes	overcast	hot	high	FALSE
yes	rainy	cool	normal	FALSE
yes	sunny	cool	normal	FALSE
yes	rainy	cool	normal	FALSE
yes	sunny	hot	normal	FALSE

Temperature=cool, hot
Total Values=7 (70%) with
Left-6 play(yes)=0.86
Right-1 play(no)=0.14

4.

Play	Outlook	Temperature	Humidity	Wind
no	sunny	mild	high	FALSE
yes	overcast	mild	high	TRUE
no	sunny	mild	high	TRUE

Temperature=Mild & Outlook=sunny
Total Values=2(20%)
no=1
yes=0

Temperature=Mild & Outlook=overcast
Total Values=1(10%)
no=0
yes=1

5.

Play	Outlook	Temperature	Humidity	Wind
yes	rainy	cool	normal	FALSE
no	rainy	cool	normal	TRUE
yes	overcast	hot	high	FALSE
yes	rainy	cool	normal	FALSE
yes	sunny	cool	normal	FALSE
yes	rainy	cool	normal	FALSE
yes	sunny	hot	normal	FALSE

Temperature=Cool, Hot & WIND= 1(TRUE)
Total Values=1(10%)
no=1
yes=0

Temperature=Cool, Hot & WIND= 0(FALSE)
Total Values=6(60%)
no=0
yes=1

PRACTICAL NO.5: NAÏVE BAYES CLASSIFICATION

FILE USED: sample1.csv

CODE:

install.packages("e1071") #The e1071 package contains the naiveBayes function. It allows numeric and factor variables to be used in the naive bayes model.
library(e1071) #load the library
sample<-read.table("E:\\BDAPractical\\sample1.csv",header=TRUE,sep=",") #read the data into a table from the file

define the data frames for the Naive Bayes classifier
traindata<-as.data.frame(sample[1:14,])
testdata<-as.data.frame(sample[15,])
traindata
testdata

```
> traindata
        Age Income JobSatisfaction    Desire Enrolls
1      <=30   High              No      Fair      No
2      <=30   High              No Excellent      No
3  31 to 40   High              No      Fair     Yes
4       >40 Medium              No      Fair     Yes
5       >40    Low             Yes      Fair     Yes
6       >40    Low             Yes Excellent      No
7  31 to 40    Low             Yes Excellent     Yes
8      <=30 Medium              No      Fair      No
9      <=30    Low             Yes      Fair     Yes
10      >40 Medium             Yes      Fair     Yes
11     <=30 Medium             Yes Excellent     Yes
12 31 to 40 Medium              No Excellent     Yes
13 31 to 40   High             Yes      Fair     Yes
14      >40 Medium              No Excellent      No
> testdata
    Age Income JobSatisfaction Desire Enrolls
15 <=30 Medium             Yes   Fair
```

#Compute the prior probabilities P(c) for Enrolls, where C = {Yes,No}.
tprior<-table(traindata$Enrolls)
tprior
tprior<-tprior/sum(tprior)
tprior

```
> tprior<-table(traindata$Enrolls)
> tprior

 No Yes 
  5   9 
> tprior<-tprior/sum(tprior)
> tprior

       No       Yes 
0.3571429 0.6428571 
```

#compute conditional probabilities P(A|C), where A ={Age, Income, JobSatisfaction,Desire} and C ={Yes, No}.
ageCounts<-table(traindata[,c("Enrolls","Age")])
ageCounts<-ageCounts/rowSums(ageCounts)
ageCounts

incomeCounts<-table(traindata[,c("Enrolls","Income")])
incomeCounts<-incomeCounts/rowSums(incomeCounts)

incomeCounts

jsCounts<-table(traindata[,c("Enrolls","JobSatisfaction")])
jsCounts<-jsCounts/rowSums(jsCounts)
jsCounts

desireCounts<-table(traindata[,c("Enrolls","Desire")])
desireCounts<-desireCounts/rowSums(desireCounts)
desireCounts

```
> ageCounts<-table(traindata[,c("Enrolls","Age")])
> ageCounts<-ageCounts/rowSums(ageCounts)
> ageCounts
       Age
Enrolls      <=30       >40  31 to 40
    No  0.6000000 0.4000000 0.0000000
    Yes 0.2222222 0.3333333 0.4444444
>
> incomeCounts<-table(traindata[,c("Enrolls","Income")])
> incomeCounts<-incomeCounts/rowSums(incomeCounts)
> incomeCounts
       Income
Enrolls      High       Low    Medium
    No  0.4000000 0.2000000 0.4000000
    Yes 0.2222222 0.3333333 0.4444444
>
> jsCounts<-table(traindata[,c("Enrolls","JobSatisfaction")])
> jsCounts<-jsCounts/rowSums(jsCounts)
> jsCounts
       JobSatisfaction
Enrolls        No       Yes
    No  0.8000000 0.2000000
    Yes 0.3333333 0.6666667
>
> desireCounts<-table(traindata[,c("Enrolls","Desire")])
> desireCounts<-desireCounts/rowSums(desireCounts)
> desireCounts
       Desire
Enrolls Excellent      Fair
    No  0.6000000 0.4000000
    Yes 0.3333333 0.6666667
```

#probability P(c|A) is determined by the product of P(a|c,) times the (c1) where c1 =Yes and c2 =No.
prob_Yes<-
ageCounts["Yes",testdata[,c("Age")]]*incomeCounts["Yes",testdata[,c("Income")]]*jsCounts["Yes",testdata[,c("JobSatisfaction")]]*desireCounts["Yes",testdata[,c("Desire")]]*tprior["Yes"]
prob_Yes

```
> prob_Yes
       Yes
0.02821869
```

prob_No<-
ageCounts["No",testdata[,c("Age")]]*incomeCounts["No",testdata[,c("Income")]]*jsCounts["No",testdata[,c("JobSatisfaction")]]*desireCounts["No",testdata[,c("Desire")]]*tprior["No"]
prob_No

```
> prob_No
         No
0.006857143
```

#The larger value of P(Yes|A) and P(No|A) determines the predicted result of the output variable.
max(prob_Yes,prob_No)

```
> max(prob_Yes,prob_No)
[1] 0.02821869
```

#Naive Bayes function computes the conditional probabilities. The function takes the form of naive Bayes (formula, data,…), where the arguments are defined as follows.
formula: A formula of the form class ~ xl + x2 + ... assuming xl, x2 ... are conditionally independent & data: A data frame of factors
model<-naiveBayes(Enrolls~Age+Income+JobSatisfaction+Desire,traindata)
model

```
> model

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
       No       Yes 
0.3571429 0.6428571 

Conditional probabilities:
     Age
Y          <=30       >40  31 to 40
  No  0.6000000 0.4000000 0.0000000
  Yes 0.2222222 0.3333333 0.4444444

     Income
Y          High       Low    Medium
  No  0.4000000 0.2000000 0.4000000
  Yes 0.2222222 0.3333333 0.4444444

     JobSatisfaction
Y            No       Yes
  No  0.8000000 0.2000000
  Yes 0.3333333 0.6666667

     Desire
Y     Excellent      Fair
  No  0.6000000 0.4000000
  Yes 0.3333333 0.6666667
```

#predicting the outcome of Enrolls with the test data shows the result is Enrolls= Yes
results<-predict(model,testdata)
results

```
> results<-predict(model,testdata)
> results
[1] Yes
Levels: No Yes
```

PRACTICAL NO.6: SVM CLASSIFICATION

CODE:

set.seed(10111) #to make sure that we get the same results for randomization

#a matrix x is created using the matrix() function. The rnorm() function generates 40 random numbers from a normal distribution with mean 0 and standard deviation 1. These numbers are arranged into a matrix with 20 rows and 2 columns.
x = matrix(rnorm(40), 20, 2)

#A vector y is created using the rep() function. It contains 20 elements, with the first 10 being -1 and the second 10 being 1
y = rep(c(-1, 1), c(10, 10))

#The code then modifies the values of x for the rows where y is equal to 1. The rows are selected using the logical expression y == 1, and the values in those rows are increased by 1 using the + operator.
x[y == 1,] = x[y == 1,] + 1

#the plot() function is used to create a scatter plot of the data. The col argument sets the color of the points based on the values in y, with -1 being blue and 1 being red. The pch argument sets the shape of the points to a filled circle.
plot(x, col = y + 3, pch = 19)

#e1071 package provides functions for statistical learning and data mining.
library(e1071)

#creates a data frame called "dat" with two columns: "x" and "y". The "x" column is assumed to already exist in the workspace, while the "y" column is created by converting an existing variable "y" into a factor using the "as.factor()" function.
dat = data.frame(x, y = as.factor(y))

#The formula "y ~ ." specifies that the response variable is "y" and all other columns in the data frame should be used as predictors. The "kernel" argument specifies that a linear kernel should be used, while the "cost" argument sets the cost parameter to 10. The "scale" argument is set to FALSE, which means that the data will not be scaled before fitting the model.
svmfit = svm(y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)

#prints the SVM model object to the console
print(svmfit)

#The plot will show the decision boundary of the SVM model and the support vectors. The svmfit object is the result of fitting an SVM model to the data dat. The dat object contains the data used to fit the SVM model.
plot(svmfit, dat)

OUTPUT: