Assignment 2 Applied Data Science	
Name	Basma Reda Shaban Abd-Elsalam Abd-Elwahab
Email	babde014@uottawa.ca

Part 1: Classification

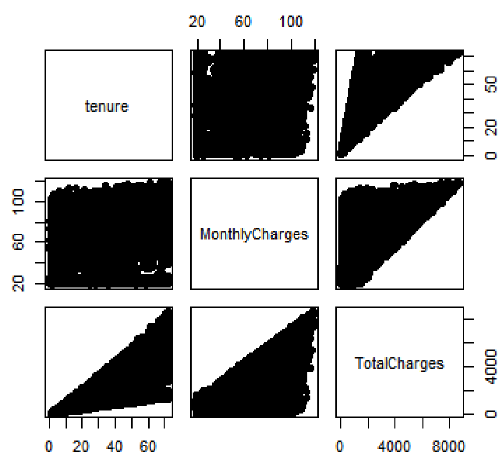
Import the churn dataset and print the head of the data:

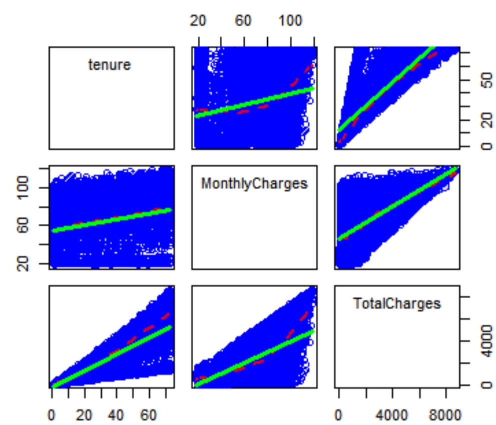
```
################### load the DataSet ###################
#Read the dataSet
Churn_DataSet <- read.csv("C:/Users/hp/Downloads/Assignment 2 (1)/Assignment 2/Churn_Dataset.csv")
head(Churn_DataSet)
```

1- Scatter plot matrix to show the relations between the variables:

```
#the pairs function can't operate with non-numeric arguments, so we must choose the numeric columns.
data = subset(Churn_DataSet, select = c("tenure","MonthlyCharges","TotalCharges"))
pairs(data, pch = 19)
```

```
scatterplotMatrix(~tenure + MonthlyCharges + TotalCharges, data = Churn_DataSet,
                  diagonal = FALSE,            # Remove kernel density estimates
                  regLine = list(col = "green", # Linear regression line color
                                 lwd = 3),      # Linear regression line width
                  smooth = list(col.smooth = "red",   # Non-parametric mean color
                                col.spread = "blue"))
```

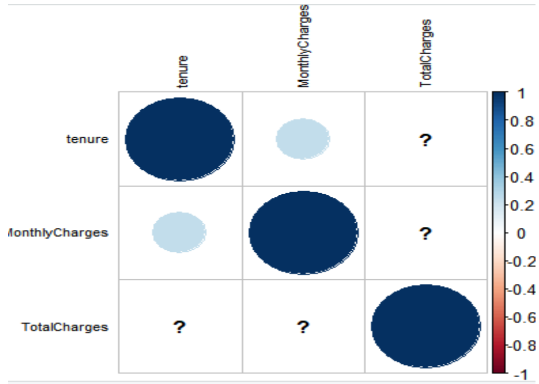




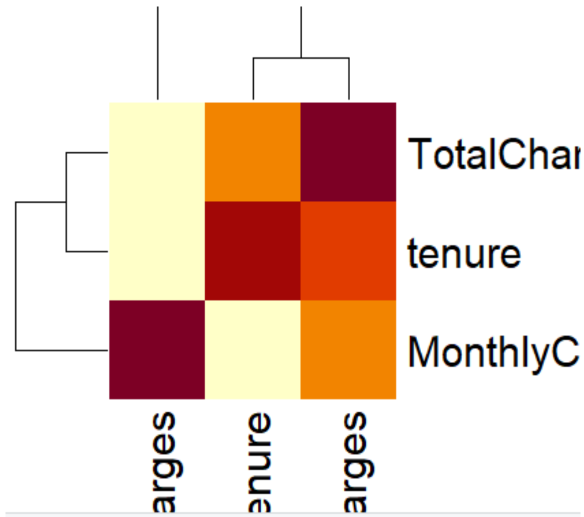
Here is a high correlation between MonthlyCharges and TotalCharges

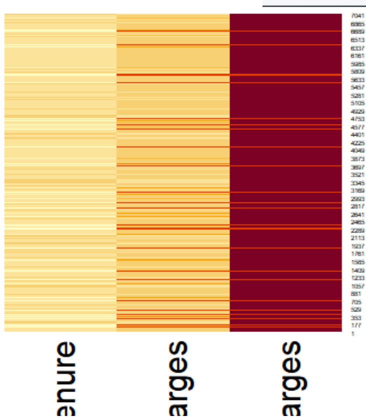
Correlation matrix:

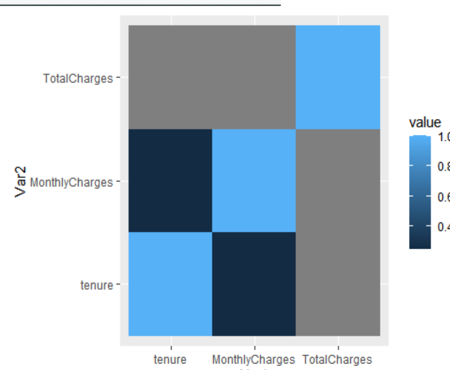
```
corrplot(cor(data),
         method = "circle",
         type = "full",
         diag = TRUE,
         tl.col = "black",
         bg = "white",
         title = "",
         col = NULL,
         tl.cex =0.7,
         cl.ratio =0.2)
```



Heatmap to determine the correlated attributes:







There is a high positive correlation between TotalCharges and Tenure, and TotalChanrges and MonthlyCharges.

2- - **Check for missing values.**
- **Remove missing values.**
- **Drop the customer id column.**
- **Remove the duplicated values**
- **Covert categorical variables to numeric.**

```r
-------------------------------------------------------- #Step 2
################### Check for missing values ###################
anyNA(Churn_DataSet)

#print the number of missing values
sum(is.na(Churn_DataSet))

#find the columns with missing values(NA)
NA_list <- colnames(Churn_DataSet)[apply(Churn_DataSet, 2, anyNA) ]
NA_list

#remove the missing values
Churn_DataSet_Clean <- Churn_DataSet %>%
  na.omit()
```

```
> anyNA(Churn_DataSet)
[1] TRUE
>
> #print the number of missing values
> sum(is.na(Churn_DataSet))
[1] 11
>
> #find the columns with missing values(NA)
> NA_list <- colnames(Churn_DataSet)[apply(Churn_DataSet, 2, anyNA)
> NA_list
[1] "TotalCharges"
>
> #remove the missing values
> Churn_DataSet_Clean <- Churn_DataSet %>%
+   na.omit()
>
>
> #check again if any missing values exits
> anyNA(Churn_DataSet_Clean)
[1] FALSE
> sum(is.na(Churn_DataSet_Clean))
[1] 0
```

```r
########## drop the customerID columns from the dataset ##########
Churn_DataSet_Clean <- Churn_DataSet_Clean %>% select(-customerID)

########## remove the duplicated values from the dataset ##########
duplicated(Churn_DataSet_Clean)
Churn_DataSet_Clean = subset(Churn_DataSet_Clean,
                             !duplicated(Churn_DataSet_Clean))

########## convert categorical variables to numerical ##########
md.pattern(Churn_DataSet_Clean, plot = FALSE)
```

```
     gender SeniorCitizen Partner Dependents tenure PhoneService MultipleLines InternetService
7032      1             1       1          1      1            1             1               1
          0             0       0          0      0            0             0               0
     OnlineSecurity OnlineBackup DeviceProtection TechSupport StreamingTV StreamingMovies Contract
7032              1            1                1           1           1               1        1
                  0            0                0           0           0               0        0
     PaperlessBilling PaymentMethod MonthlyCharges TotalCharges Churn
7032                1             1              1            1     1 0
                    0             0              0            0     0 0
>
```

3- Train the decision tree model:

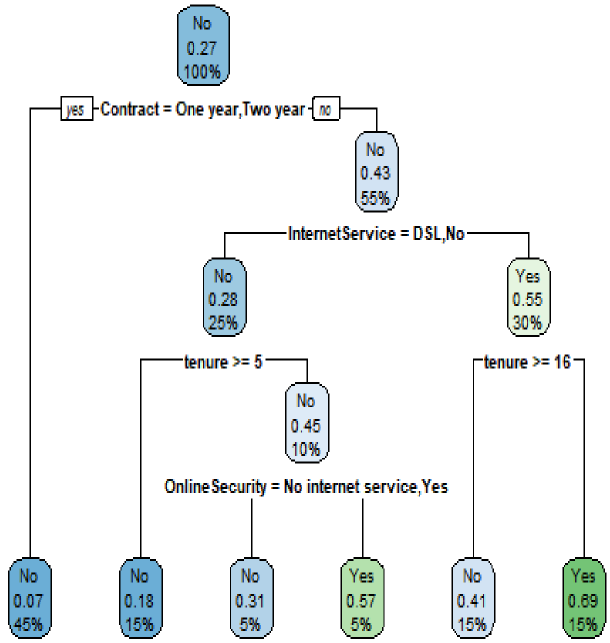
- Split the dataset into train and test split:

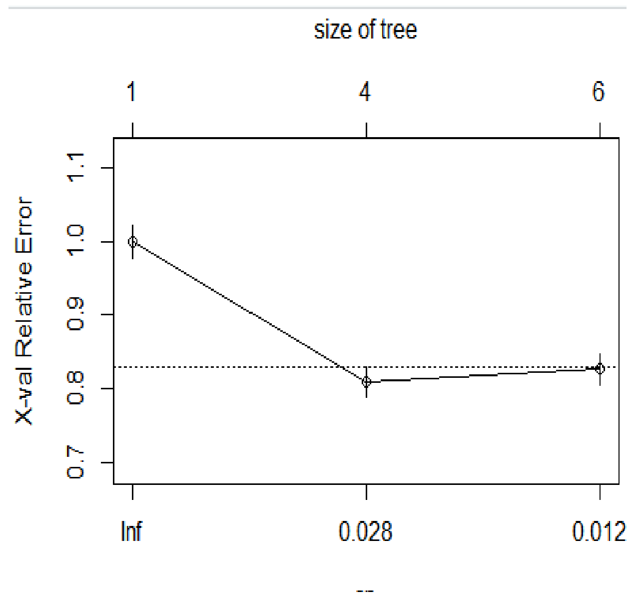
```r
#################### Split the DataSet #######################
set.seed(123)
df <- sample.split(Y = Churn_DataSet_Clean$Churn, SplitRatio = 0.8)
trainingSet <- subset(x = Churn_DataSet_Clean, df == TRUE)
testingSet <- subset(x = Churn_DataSet_Clean, df == FALSE)
dim(trainingSet)
dim(testingSet)
```

- Build the decision tree model:

```r
#################### Build Decision Tree ###################
DecisionTreeModel <- rpart(Churn ~ .,
                           data = trainingSet,
                           method = "class")

##################### plot the model ######################
rpart.plot(DecisionTreeModel)
plotcp(DecisionTreeModel)

##################### model predection ####################
y_pred <- predict(DecisionTreeModel,
                  newdata = testingSet ,
                  type = "class")
```

-





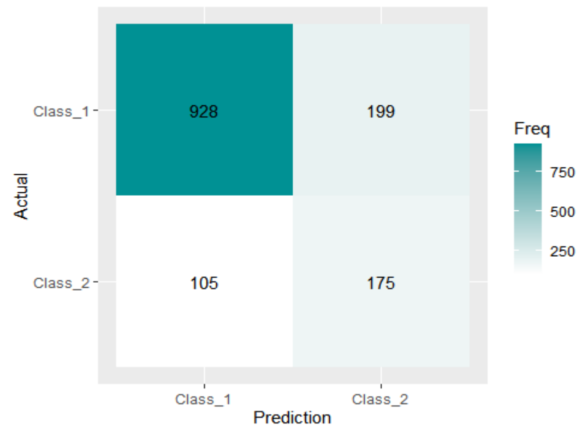
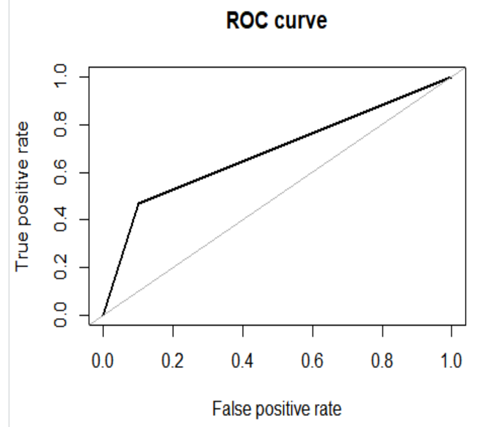
The tree classified customers as 45% of them who have contract for one or two years could churn with very low probability of 0.07 and the other 55% of customers who didn't have a contract could churn with probability of 0.43 ,25% of those 55% of customers who didn’t have internet service and there tenure < = 5 haven’t churn with probability of 0.18, and the other 10% of 25% who have tenure >= 5 and didn’t have online security wouldn’t churn with probability of 0.31, and who have online security and internet services would churn with probability 0.57, and the customers who don’t have contract with one year or two years, wouldn’t churn with probability of 0.43 and the customers who have internet service with DSL = no and tenure >=16 wouldn’t churn with 0.41 and customers would churn with probability 0.69

Rules of the tree:

- customers who didn't hold contract would churn with high probability.
- customers who didn't receive online security may churn.
- customers who have tenure less than 6 tends to churn more that the customers who have tenure more than 16.

After evaluating the decision tree model:

Accuracy	Precision	Recall	F1-Score	AUC
0.7839	**0.8984**	**0.8234**	**0.8593**	**0.683**

Confusion matrix	ROC curve
	

4- Try different ways to improve the decision tree:

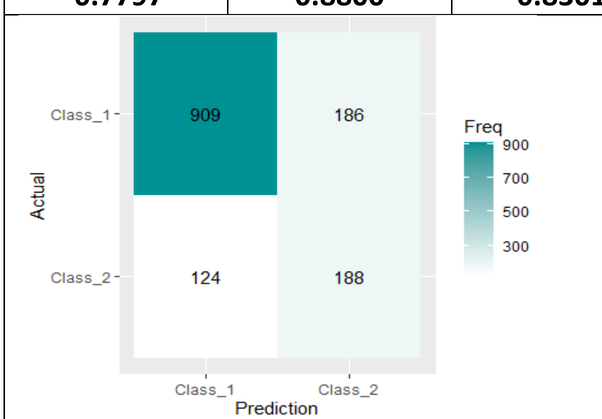
- Gini split

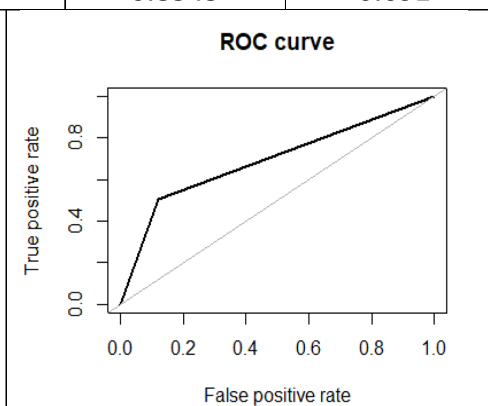
```
############ pecify the cross-validation method ###########
ct <- trainControl(method = "cv", number = 10)

#fit a decision tree model and use k-fold CV to evaluate performance
decisionTreeGini <- train(Churn ~ . , data = trainingSet,
                          method = "rpart",
                          parms = list(split = "gini"),
                          trControl = ct,
                          tuneLength = 100)
```

```
5625 samples
  19 predictor
   2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 5063, 5062, 5063, 5062, 5063, 5063, ..
```

After evaluating the decision tree with Gini split model:

Accuracy	Precision	Recall	F1-Score	AUC
0.7797	**0.8800**	**0.8301**	**0.8543**	**0.691**



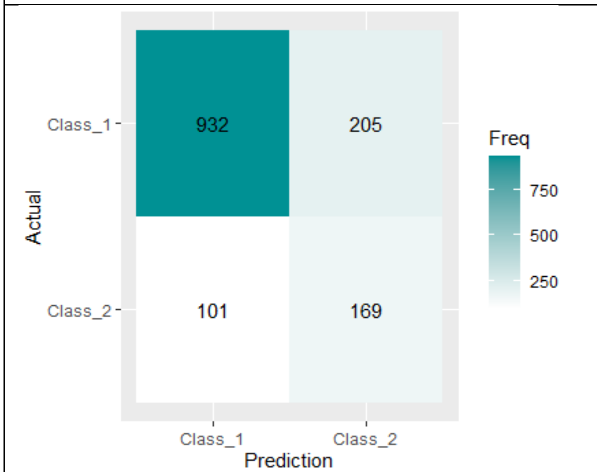
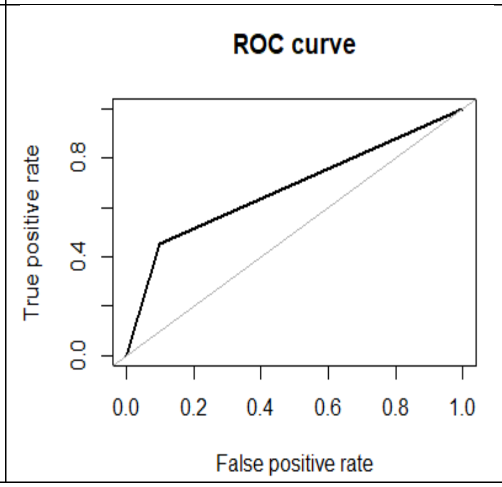


- Split by information:

```
                    ################### split tree with information ##################
decisionTreeInformation <- rpart(Churn ~ .,
                                  data = trainingSet,
                                  method = "class",
                                  parms = list(split = "information"))
decisionTreeInformation
plotcp(decisionTreeInformation)
```

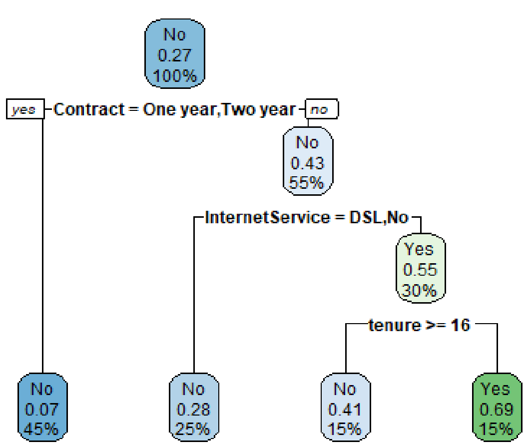
After evaluating the decision tree with information gain split model:

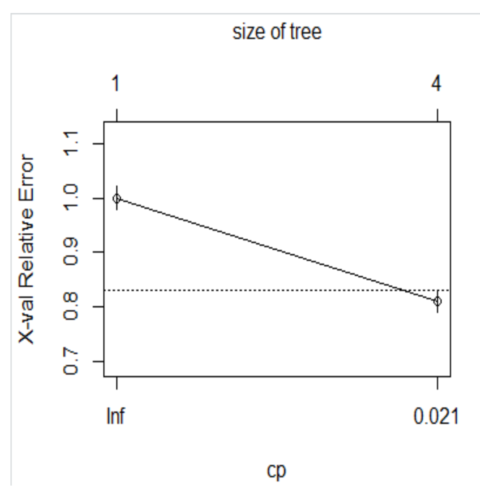
Accuracy	Precision	Recall	F1-Score	AUC
0.7825	0.9022	0.8197	0.8590	0.677

Confusion matrix	ROC curve
	

- Prune the tree after splitting:

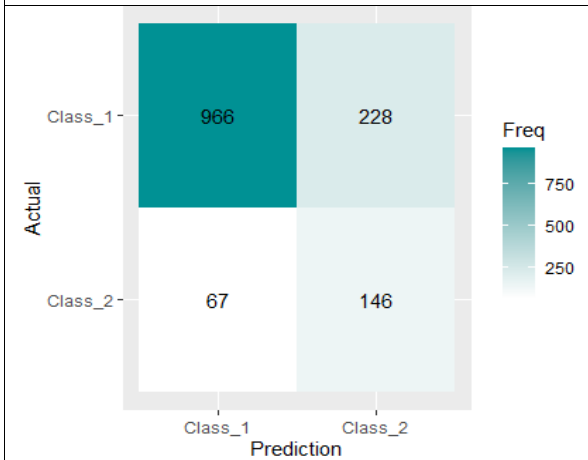
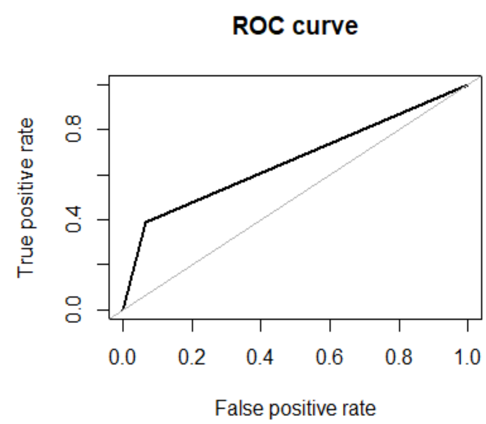
```
                    #################### Prune the tree #####################
decitionTreePrune <- rpart(Churn ~ .,
                           data = trainingSet,
                           method = "class",
                          control = rpart.control(cp = 0.0082,
                                                  maxdepth = 3,
                                                  minsplit = 2))
rpart.plot(decitionTreePrune)
plotcp(decitionTreePrune)
```





After evaluating decision tree model after pruning:

Accuracy	Precision	Recall	F1-Score	AUC
0.7903	0.9351	0.8090	0.8675	0.663

Confusion matrix	ROC curve
	

After pruning the tree, the accuracy increased by 1 percentage, it is not enough for improving the decision tree model.

5- Train XGBOOST model:

- **Converting the training and testing data into matrix and converting the y label as a factor:**

```
# convert the train and test data into xgboost matrix type.
xgboost_train = xgb.DMatrix(data=X_train, label=y_train)
xgboost_test = xgb.DMatrix(data=X_test, label=y_test)
```

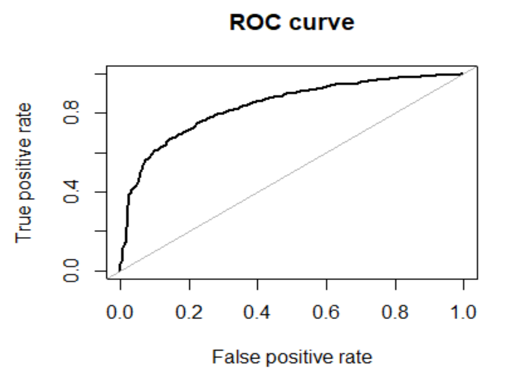
- **Train the XGBOOST model:**

```
################ train XGBOOST model ################
XGBmodel <- xgboost(data = xgboost_train,# the data
                    max.depth=3, , # max depth
                    nrounds=70) # max number of boosting iteration

summary(XGBmodel)
```

```
                Length Class              Mode
handle              1  xgb.Booster.handle externalptr
raw             85293  -none-             raw
niter               1  -none-             numeric
evaluation_log      2  data.table         list
call               14  -none-             call
params              2  -none-             list
callbacks           2  -none-             list
feature_names      19  -none-             character
nfeatures           1  -none-             numeric
```

After evaluating the XGBOOST model:

Accuracy	Precision	Recall	F1-Score	AUC
0.7974	0.8369	0.8993	0.8670	0.839

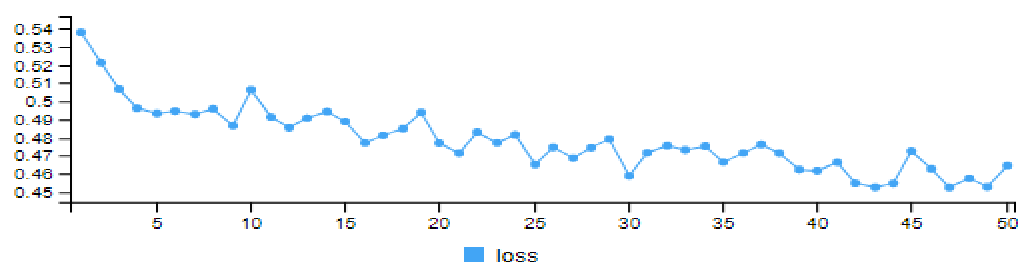
Confusion matrix	ROC curve
	

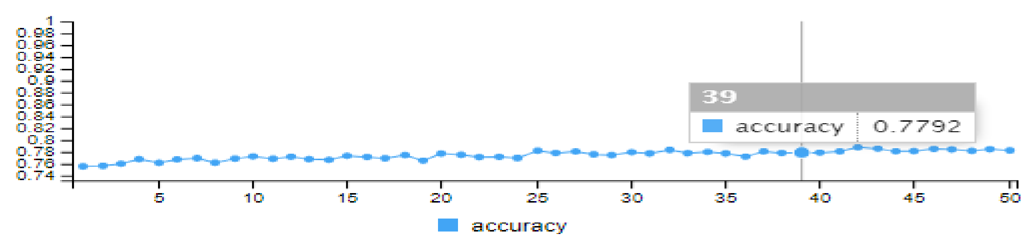
According to the high F1-Score,and accuracies of training and testing, there is no sign of overfitting.

6- Train the Deep neural network with 3 dense layers:

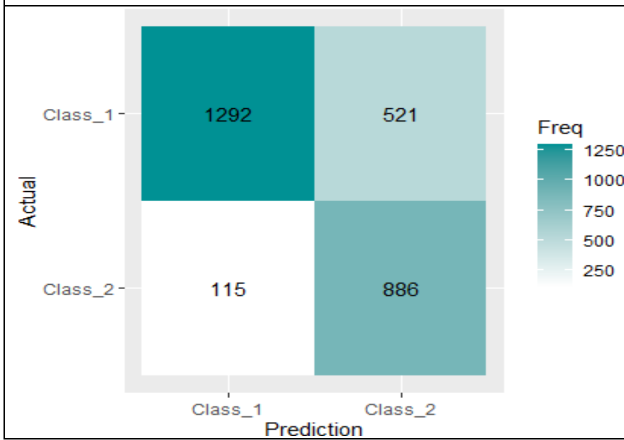
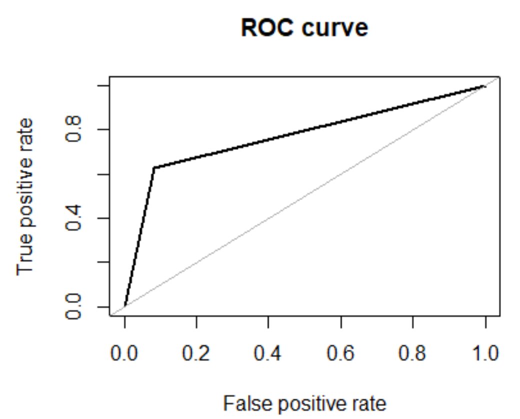
```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 128, input_shape = 19) %>%
  layer_dropout(rate=0.1)%>%
  layer_activation(activation = 'tanh') %>%
  layer_dense(units = 64)%>%
  layer_activation(activation = 'tanh')%>%
  layer_dropout(rate=0.1)%>%
  layer_dense(units = 2) %>%
  layer_activation(activation = 'sigmoid')
```

```
#compiling the defined model with metric = accuracy and optimiser as adam.
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = 'adam',
  metrics = c('accuracy')
)
```





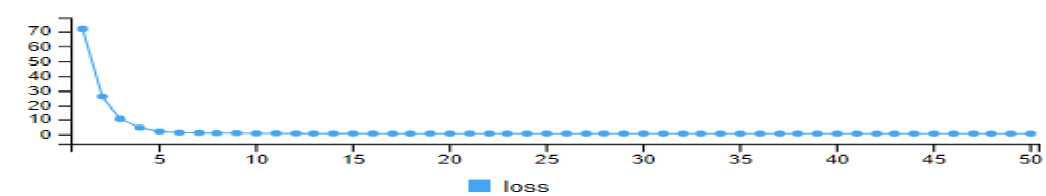
After evaluating the DNN model:

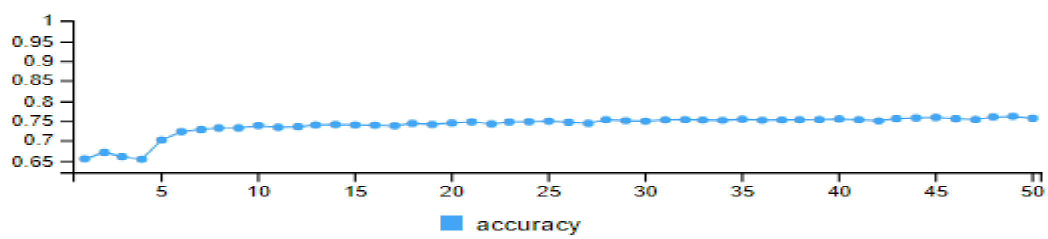
<table>
<tr><th>Accuracy</th><th>Precision</th><th>Recall</th><th>F1-Score</th><th>AUC</th></tr>
<tr><td>0.774</td><td>0.9183</td><td>0.7126</td><td>0.8025</td><td>0.774</td></tr>
<tr><th colspan="3">Confusion matrix</th><th colspan="2">ROC curve</th></tr>
<tr><td colspan="3">
</td><td colspan="2">
</td></tr>
</table>

-Trying to change the activation function:

```
model2 <- keras_model_sequential()
model2 %>%
  layer_dense(units = 128, input_shape = 19) %>%
  layer_dropout(rate=0.4)%>%
  layer_activation(activation = 'relu') %>%
  layer_dense(units = 64)%>%
  layer_activation(activation = 'relu')%>%
  layer_dropout(rate=0.4)%>%
  layer_dense(units = 2) %>%
  layer_activation(activation = 'sigmoid')
```

```
#compiling the defined model with metric = accuracy and optimiser as adam.
model2 %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = 'adam',
  metrics = c('accuracy')
)

#fitting the model on the training dataset
model2 %>% fit(train_keras_x, train_keras_y, epochs = 50, batch_size = 128)
```

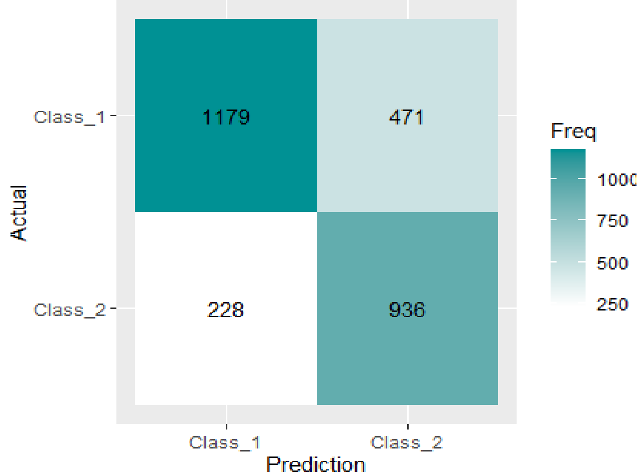




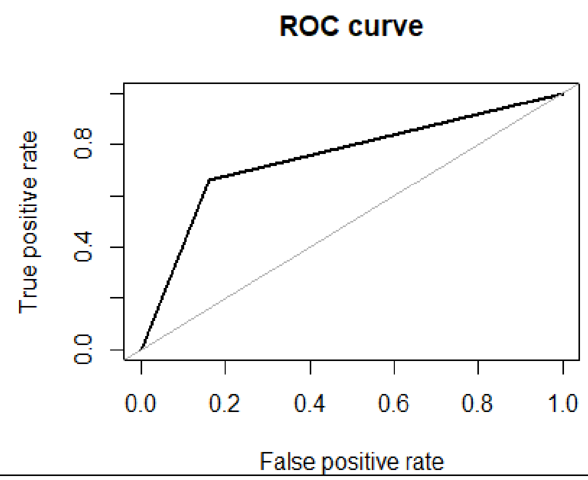
After evaluating the DNN model2:

Accuracy	Precision	Recall	F1-Score	AUC
0.7516	0.8380	0.7145	0.7145	0.752

Confusion matrix



ROC curve



After changing the activation function to 'relu' rather than 'tanh' the accuracy had been changed, and the dropout rate with 0.4 rather than 0.1, the accuracy, percision and F1-Score decreased, but the Recall increased.

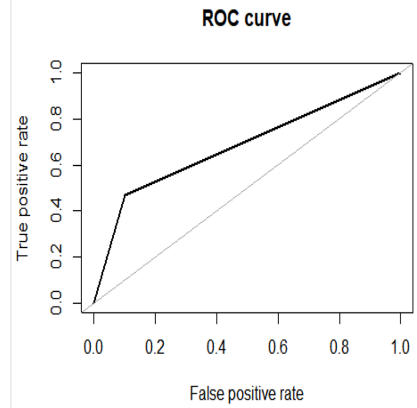
7- Compare between the models to get the best and worst one:

Model	Accuracy	Precision	Recall	F1-Score
Decision tree	0.7839	0.8984	0.8234	0.8593
DT with Gini split	0.7797	0.8800	0.8301	0.8543
DT with information split	0.7825	0.9022	0.8197	0.8590
DT after pruning	0.7903	0.9351	0.8090	0.8675
XGBOOST	0.7974	0.8369	0.8993	0.8670
DNN model 1	0.774	0.9183	0.7126	0.8025
DNN model 2	0.7516	0.8380	0.7145	0.7145

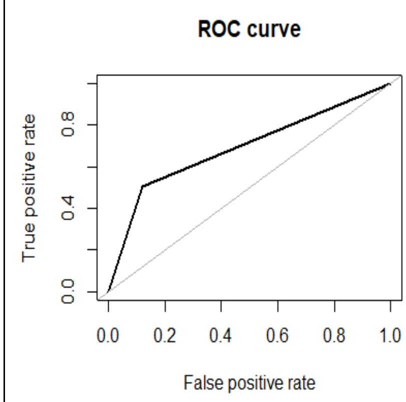
	Accuracy	Precision	Recall	F1-Score
Best model	XGBOOST	DT after pruning	XGBOOST	DT after pruning
Worst model	DNN model 2	DNN model 1	DNN model 2	DNN model 1

8- ROC curve for each model:

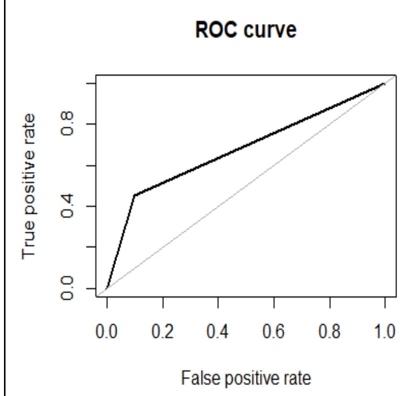
DT model



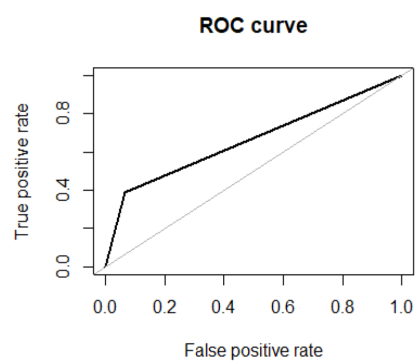
DT with Gini split model



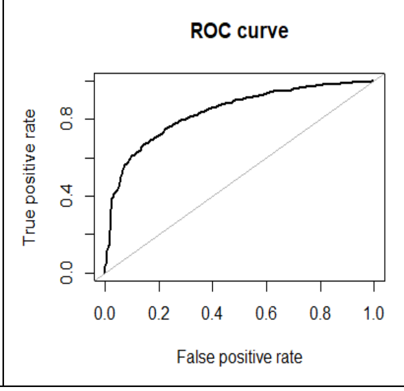
DT with information split



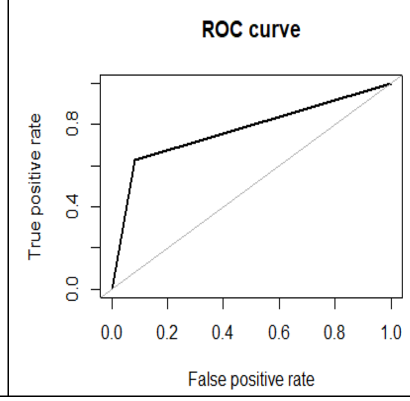
DT model after pruning



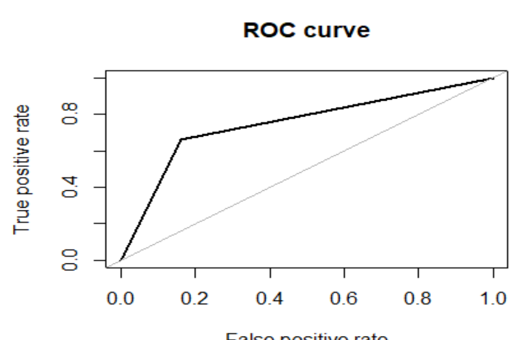
XGBOOST model



DNN model 1



DNN model 2



Model name	AUC Value
DT	0.683
DT with Gini split	0.691
DT with information gain split	0.677
DT after pruning	0.663
XGBOOST	0.839
DNN model 1	0.774
DNN model 2	0.752

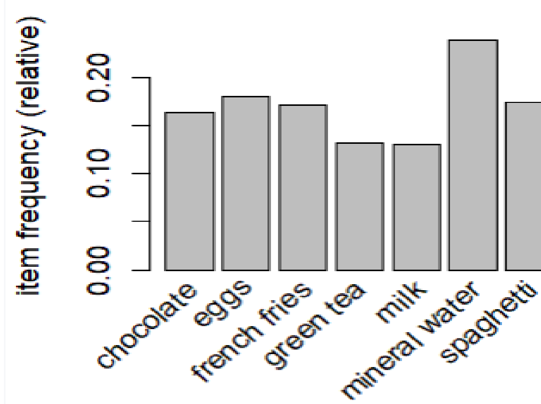
Part B:

```
#Load the transaction dataset
data=read.transactions("C:/Users/hp/Downloads/Assignment 2 (1)/Assignment 2/transactions.csv",
                      format='basket',header =TRUE ,sep=',')
summary(data)
```

```
> summary(data)
transactions as itemMatrix in sparse format with
 7500 rows (elements/itemsets/transactions) and
 119 columns (items) and a density of 0.03287171

most frequent items:
mineral water          eggs     spaghetti  french fries     chocolate       (Other)
         1787          1348          1306          1282          1229         22386

element (itemset/transaction) length distribution:
sizes
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   18   19
1754 1358 1044  816  667  493  391  324  259  139  102   67   40   22   17    4    1    2

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   3.000   3.912   5.000  19.000

includes extended item information - examples:
             labels
1           almonds
2 antioxydant juice
3         asparagus
```

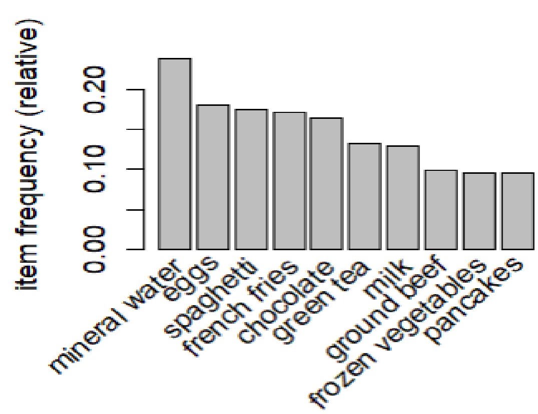
Plot the frequency of the items:

```
# plot the frequency of the items
itemFrequency(data[,1:3])
itemFrequencyPlot(data, support = 0.1)
```



1- Plot the top 10 transactions:

```
#PLot for Top 10 Transactions
itemFrequencyPlot(data, topN = 10)
```



2- Display the rules sorted by descending lift value:

```
association_rule_1 <- apriori(data, parameter = list(support = 0.002,
                                                     confidence =0.20,
                                                     maxlen = 3))

association_rule_1

# Display the rules, sorted by descending lift value
association_rule_lift_sort <- sort(association_rule_1, by = "lift")
```

```
> association_rule_1 <- apriori(data, parameter = list(support = 0.002,
+                                                      confidence =0.20,
+                                                      maxlen = 3))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target  ext
        0.2    0.1    1 none FALSE            TRUE       5   0.002      1      3  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 15

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[119 item(s), 7500 transaction(s)] done [0.00s].
sorting and recoding items ... [115 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 done [0.00s].
writing ... [2186 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
>
> association_rule_1
set of 2186 rules
>
```

Display the rules sorted by descending lift value

```
# Display the rules, sorted by descending lift value
association_rule_lift_sort <- sort(association_rule_1, by = "lift")
# Display the rules, sorted by descending support value
inspect(association_rule_lift_sort)
```

	lhs		rhs	support	confidence	coverage	lift	count
[1]	{escalope, mushroom cream sauce}	=>	{pasta}	0.002533333	0.4418605	0.005733333	28.084352	19
[2]	{escalope, pasta}	=>	{mushroom cream sauce}	0.002533333	0.4318182	0.005866667	22.647807	19
[3]	{mushroom cream sauce, pasta}	=>	{escalope}	0.002533333	0.9500000	0.002666667	11.974790	19
[4]	{parmesan cheese, tomatoes}	=>	{frozen vegetables}	0.002133333	0.6666667	0.003200000	6.993007	16
[5]	{mineral water, whole wheat pasta}	=>	{olive oil}	0.003866667	0.4027778	0.009600000	6.127451	29
[6]	{frozen vegetables, parmesan cheese}	=>	{tomatoes}	0.002133333	0.3902439	0.005466667	5.705320	16
[7]	{burgers, herb & pepper}	=>	{ground beef}	0.002266667	0.5483871	0.004133333	5.580601	17
[8]	{light cream, mineral water}	=>	{chicken}	0.002400000	0.3272727	0.007333333	5.454545	18
[9]	{french fries, mushroom cream sauce}	=>	{escalope}	0.002000000	0.4285714	0.004666667	5.402161	15
[10]	{fromage blanc}	=>	{honey}	0.003333333	0.2450980	0.013600000	5.178128	25
[11]	{ground beef, shrimp}	=>	{herb & pepper}	0.002933333	0.2558140	0.011466667	5.171441	22
[12]	{ground beef, low fat yogurt}	=>	{herb & pepper}	0.002400000	0.2500000	0.009600000	5.053908	18
[13]	{spaghetti, tomato sauce}	=>	{ground beef}	0.003066667	0.4893617	0.006266667	4.979936	23
[14]	{chocolate, parmesan cheese}	=>	{frozen vegetables}	0.002000000	0.4687500	0.004266667	4.916958	15
[15]	{meatballs, spaghetti}	=>	{tomatoes}	0.002133333	0.3333333	0.006400000	4.873294	16
[16]	{chocolate, whole wheat pasta}	=>	{olive oil}	0.002000000	0.3191489	0.006266667	4.855207	15
[17]	{light cream}	=>	{chicken}	0.004533333	0.2905983	0.015600000	4.843305	34
[18]	{frozen vegetables, herb & pepper}	=>	{ground beef}	0.002800000	0.4666667	0.006000000	4.748982	21
[19]	{mineral water, tomato sauce}	=>	{ground beef}	0.002666667	0.4651163	0.005733333	4.733205	20
[20]	{pasta}	=>	{escalope}	0.005866667	0.3728814	0.015733333	4.700185	44

3- Select the rule from QII-b with the greatest lift. Compare this rule with the highest lift, rule for maximum length of 2.

```
association_rule_2 <- apriori(data, parameter = list(support = 0.002,
                                                     confidence =0.20,
                                                     maxlen = 2))

# Display the rules, sorted by descending lift value
association_rule_lift_sort2 <- sort(association_rule_2, by = "lift")
```

```
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target  ext
        0.2    0.1    1 none FALSE            TRUE       5   0.002      1      2  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 15

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[119 item(s), 7500 transaction(s)] done [0.01s].
sorting and recoding items ... [115 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 done [0.00s].
writing ... [368 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

Display the rules sorted by descending lift value:

```
# Display the rules, sorted by descending support value
inspect(association_rule_lift_sort2)
```

```
     lhs                      rhs                 support     confidence coverage    lift     count
[1]  {fromage blanc}       => {honey}             0.003333333 0.2450980  0.013600000 5.178128  25
[2]  {light cream}         => {chicken}           0.004533333 0.2905983  0.015600000 4.843305  34
[3]  {pasta}               => {escalope}          0.005866667 0.3728814  0.015733333 4.700185  44
[4]  {pasta}               => {shrimp}            0.005066667 0.3220339  0.015733333 4.514494  38
[5]  {whole wheat pasta}   => {olive oil}         0.008000000 0.2714932  0.029466667 4.130221  60
[6]  {extra dark chocolate} => {chicken}          0.002800000 0.2333333  0.012000000 3.888889  21
[7]  {tomato sauce}        => {ground beef}       0.005333333 0.3773585  0.014133333 3.840147  40
[8]  {mushroom cream sauce} => {escalope}         0.005733333 0.3006993  0.019066667 3.790327  43
[9]  {barbecue sauce}      => {turkey}            0.002533333 0.2345679  0.010800000 3.751086  19
[10] {extra dark chocolate} => {olive oil}        0.002666667 0.2222222  0.012000000 3.380663  20
[11] {herb & pepper}       => {ground beef}       0.016000000 0.3234501  0.049466667 3.291555 120
[12] {gluten free bar}     => {pancakes}          0.002133333 0.3076923  0.006933333 3.236595  16
[13] {shallot}             => {cookies}           0.002000000 0.2586207  0.007733333 3.216675  15
[14] {light cream}         => {olive oil}         0.003200000 0.2051282  0.015600000 3.120612  24
[15] {almonds}             => {burgers}           0.005200000 0.2565789  0.020266667 2.942419  39
[16] {parmesan cheese}     => {frozen vegetables} 0.005466667 0.2751678  0.019866667 2.886375  41
[17] {strong cheese}       => {spaghetti}         0.003733333 0.4827586  0.007733333 2.772350  28
[18] {blueberries}         => {ground beef}       0.002400000 0.2608696  0.009200000 2.654711  18
[19] {bacon}               => {burgers}           0.002000000 0.2307692  0.008666667 2.646436  15
[20] {whole weat flour}    => {pancakes}          0.002266667 0.2463768  0.009200000 2.591621  17
[21] {bacon}               => {pancakes}          0.002133333 0.2461538  0.008666667 2.589276  16
[22] {whole wheat pasta}   => {milk}              0.009866667 0.3348416  0.029466667 2.583655  74
```

Determine which rule has better fit:

```
# The highest lift rule with maxlen = 3
inspect(association_rule_lift_sort[1])

#he highest lift rule with maxlen = 2
inspect(association_rule_lift_sort2[1])
```

	lhs <chr>	<chr>	rhs <chr>	support <dbl>	confidence <dbl>	coverage <dbl>	lift <dbl>	count <int>
[1]	{escalope, mushroom cream sauce}	=>	{pasta}	0.002532996	0.4418605	0.005732569	28.0881	19

1 row

lhs <chr>	<chr>	rhs <chr>	support <dbl>	confidence <dbl>	coverage <dbl>	lift <dbl>	count <int>
{fromage blanc}	=>	{honey}	0.003332889	0.245098	0.01359819	5.164271	25

i) **Rule1** has the **best better fit**, because the left of rule 1 is highest than the lift of rule 2.
Rule 2 has the greater support than Rule 1.

ii) **if I were a marketing manager, I will choose Rule 1 because Rule1 has the highest confidence and highest lift, because, the higher the confidence, the greater the likelihood that the item will be purchased or, in other words, the greater the return rate you can expect for a given rule and lift summarizes the strength of association between the products, the larger the lift the greater the link between the two products.**