# HW5

**Divya Kamma UIN: 670505193**
**Preethi Reddy Nandanuru UIN: 654074552**
**Ashrita Cheetirala UIN: 659259358**

```
#Loading the datasets
library(tidyverse)

library(readxl)

library(dplyr)

library(rpart)

library(randomForest)

library(ggplot2)

library(factoextra)

library(scales)

library(cluster)

library(fastDummies)

#Reading the dataset into R
raw_data = read_excel("/Users/ashritacheetirala/Desktop/UIC/Sem 2/Data
Mining/HW5/IMB881-XLS-ENG.xlsx",sheet=2)
```

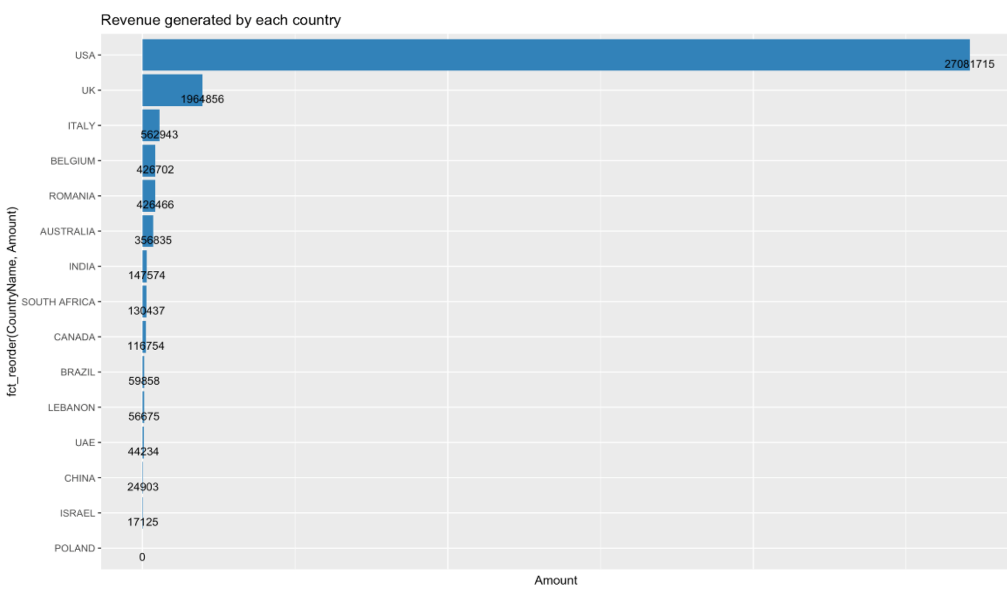
Preprocessing the data

```
#Removing Unnecessary columns from the dataset
df <- subset (raw_data, select = -
c(CustomerOrderNo,Custorderdate,UnitName,TotalArea))
#df

#Converting all binary and categorical  variables to factors
cols <-
c("OrderType","OrderCategory","CustomerCode","CountryName","ITEM_NAME","Quali
tyName","DesignName","ColorName","ShapeName")
df[cols]<- lapply(df[cols], factor)
#str(df)
#summary(df)
typeof(df$QtyRequired)

## [1] "double"
```

```
df$QtyRequired <- as.factor(df$QtyRequired)
df$QtyRequired <- as.numeric(df$QtyRequired)
df$Amount <- as.integer(df$Amount)
df$AreaFt <- as.integer(df$AreaFt)
```

### Q1 Visualising the data to provide key insights

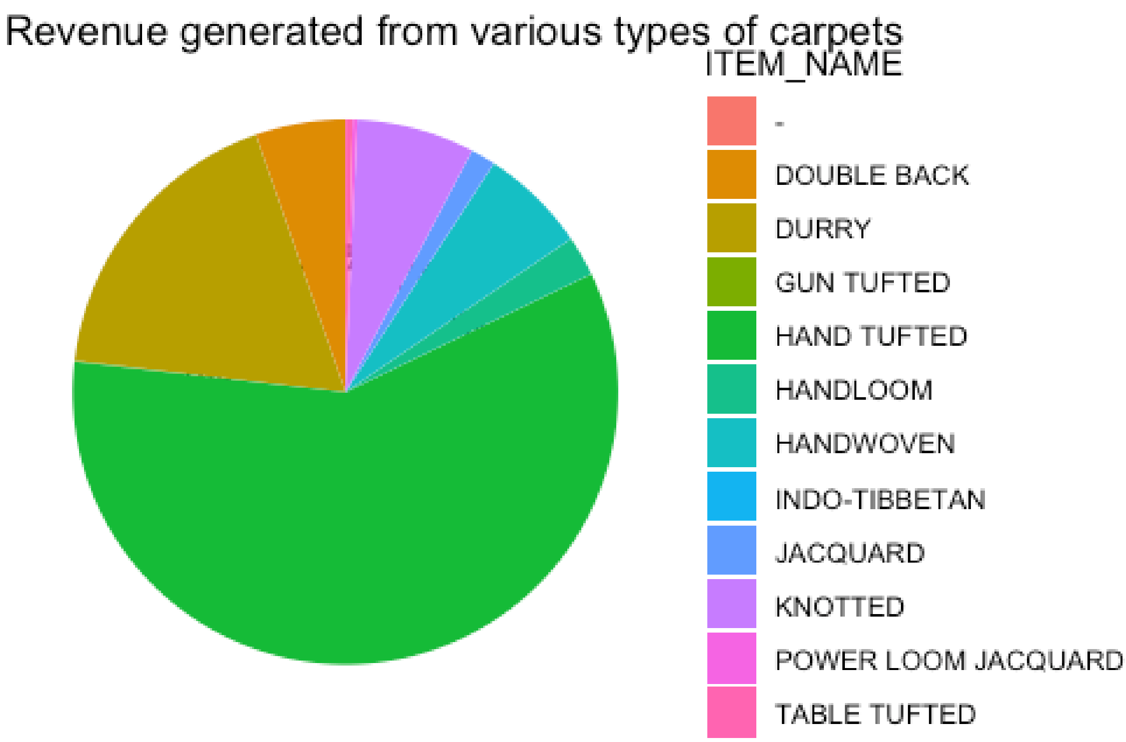
```
#Creating a dataframe to visulaise the revenue generated by various items
group_country <- df %>%
  group_by(CountryName) %>%
  dplyr::summarise(Amount = sum(Amount)) %>%
  as.data.frame()
#Revenue generated by each country
bar <- ggplot(group_country, aes(fct_reorder(CountryName, Amount),Amount))+
      geom_bar(stat="identity", fill="steelblue")+
      geom_text(aes(label=Amount), vjust=1.6, color="black", size=3.5)+
      coord_flip()+
      theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank())
bar
```




Insight: The United States sells the most carpets, followed by the United Kingdom. Other countries that contribute significantly to revenue include Italy, Romania, Australia, India, Canada, and South Africa.

```
#Creating a dataframe to visulaise the revenue generated by various items
grouped_item <- df %>%
  group_by(ITEM_NAME) %>%
  dplyr::summarise(Amount = sum(Amount)) %>%
  as.data.frame()
#Revenue generated for various types of carpets
pie <- ggplot(grouped_item, aes(x = "", y = Amount, fill = ITEM_NAME)) +
  theme_void()+
```

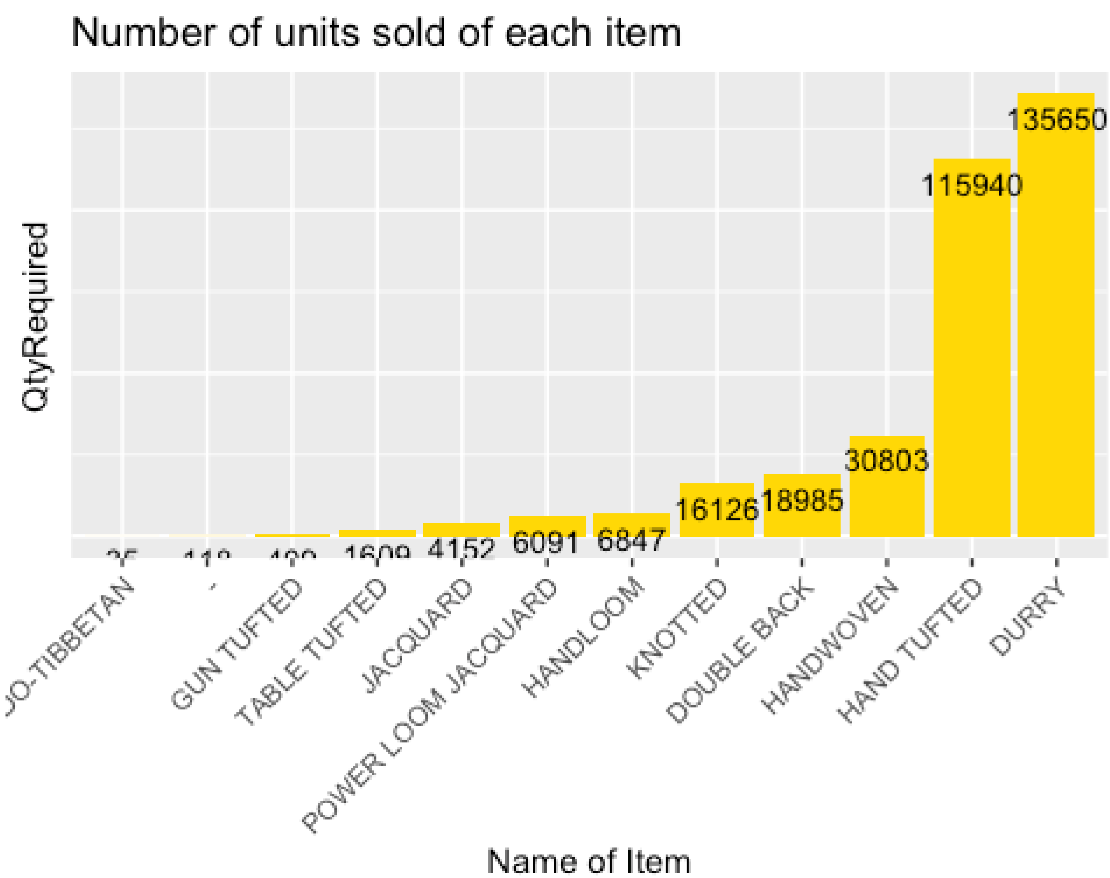
```
  geom_text(aes(label = paste0(round(Amount/sum(Amount)*100), "%")), position
= position_stack(vjust = 0.5))+
  geom_bar(width = 1, stat = "identity") +
  coord_polar(theta = "y", start = 0) +
  ggtitle("Revenue generated from various types of carpets")
pie
```




Insight: With this visualisation we can see that Hand Tufted carpet contributed the highest to the revenue followed by Durry and the lowest being power loom jacquared.

```
#Creating a dataframe to visulaise the revenue generated by various items
group_item <- df %>%
  group_by(ITEM_NAME) %>%
  dplyr::summarise(QtyRequired = sum(QtyRequired)) %>%
  as.data.frame()
#Number of units of each item sold
col <- group_item %>%
    ggplot(aes(fct_reorder(ITEM_NAME, QtyRequired),  QtyRequired))+
  geom_col(fill="gold") +
  labs(x="Name of Item")+
  geom_text(aes(label=QtyRequired), vjust=1.6, color="black", size=3.5)+
  theme(axis.text.y=element_blank(),axis.ticks.y=element_blank())+
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1))+
```

```
  ggtitle("Number of units sold of each item")
col
```




Insight: The “Durry” type carpets were sold the most, but it was the second highest contributor of revenue and significantly lower than “Hand Tufted” in terms of revenue, so we can say it is much cheaper in price. Because “Hand Tufted” carpet is the highest revenue generator but the number of units sold is much lower, we can say that it is on the expensive side and thus a premium quality carpet.

**Q2 What ML models can Champo Carpets use to solve their problems**

Champo Carpets can use various ML algorithms to solve their problem. Champo Carpet's main aim is to reduce the number of false positives. When the order is actually not converted, but they are predicted as converted, there will be loss for Champo carpets as the samples made are wasted and it is expensive to make each sample. Hence, the champo carpets must be focusing on improving  the precision(actually not converted but predicted as converted).

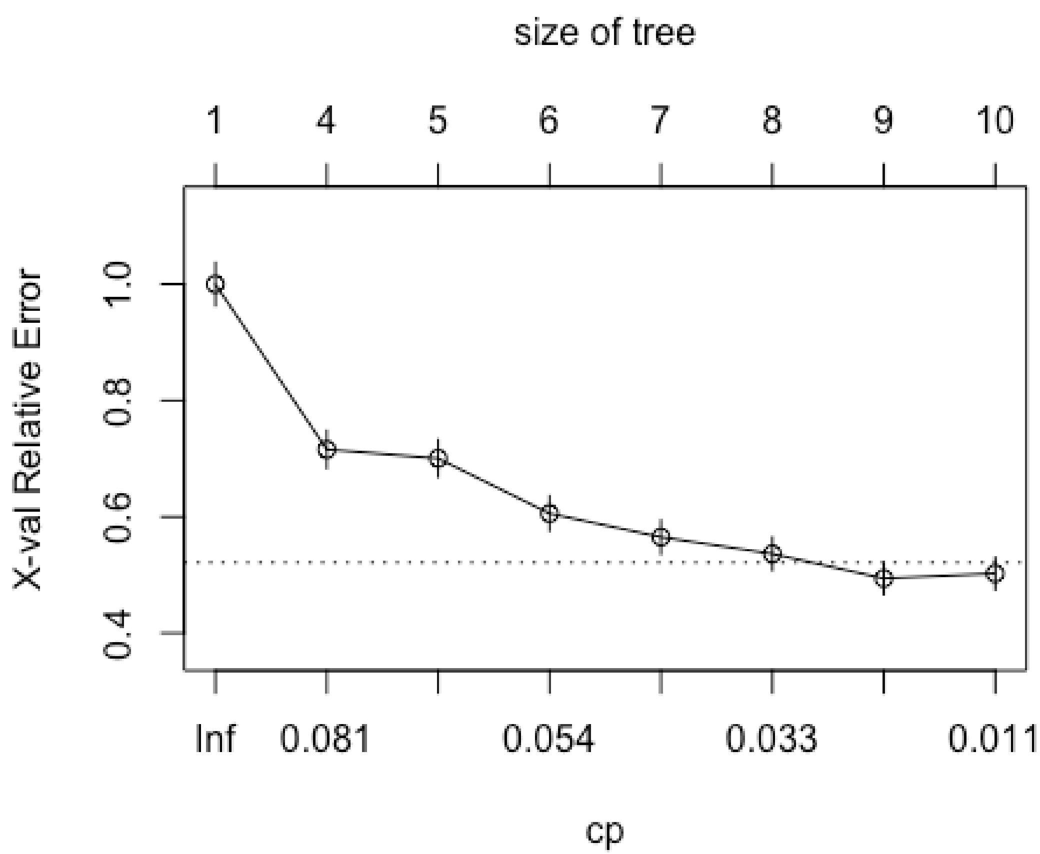
All the ML classification algorithms like decision trees, randoForest can be used to find out the precision of test data. When it comes to improving the precision, it is better to use randomForest than decision trees as it prevents overfitting by using multiple trees. Neural networks can also be used to predict the future data as it discovers any complex relations hidden in the data.

With regression, we can predict the output based on input variables. We can find out the importance of the variables by checking if they are statistically significant or not. Logistic regression can be performed to understand the relationship between predictor variables and probability of orders getting converted to samples.

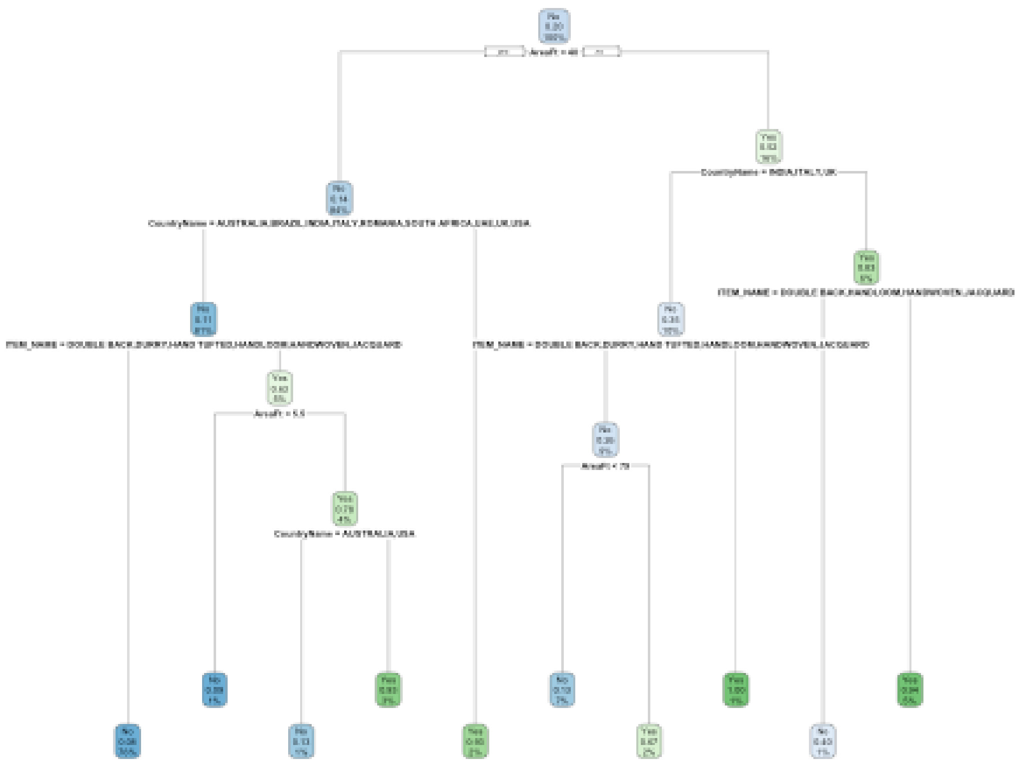
### Q3 Various ML Models on Balanced and imbalanced data

```
data_sample = read_excel("/Users/ashritacheetirala/Desktop/UIC/Sem 2/Data
Mining/HW5/IMB881-XLS-ENG.xlsx",sheet=4)
library('fastDummies')
data_sample <- dummy_cols(data_sample, select_columns = 'CountryName')

#data cleaning
data_sample_decion_tree <- subset(data_sample,select = -
c(USA,UK,Italy,Belgium,Romania,Australia,India,`Hand Tufted`,Durry,`Double
Back`,`Hand
Woven`,Knotted,Jacquard,Handloom,Other,REC,Round,Square,CountryName_AUSTRALIA
,CountryName_BELGIUM,CountryName_BRAZIL,CountryName_CANADA,CountryName_CHINA,
CountryName_INDIA,CountryName_ISRAEL,CountryName_ITALY,CountryName_POLAND,Cou
ntryName_ROMANIA,`CountryName_SOUTH
AFRICA`,CountryName_UAE,CountryName_UK,CountryName_USA,CustomerCode))
str(data_sample_decion_tree)

## tibble [5,820 × 6] (S3: tbl_df/tbl/data.frame)
##  $ CountryName     : chr [1:5820] "INDIA" "USA" "USA" "USA" ...
##  $ QtyRequired     : num [1:5820] 1 1 2 1 1 1 1 1 1 1 ...
##  $ ITEM_NAME       : chr [1:5820] "HAND TUFTED" "HAND TUFTED" "HAND
TUFTED" "HAND TUFTED" ...
##  $ ShapeName       : chr [1:5820] "REC" "REC" "REC" "REC" ...
##  $ AreaFt          : num [1:5820] 80 80 80 80 80 80 80 40 108 54 ...
##  $ Order Conversion: num [1:5820] 1 1 1 1 1 1 1 1 0 1 ...
##  - attr(*, ".internal.selfref")=<externalptr>

data_sample_decion_tree$`Order Conversion` <-
ifelse(data_sample_decion_tree$`Order Conversion`==1,"Yes","No")
df1<-data_sample_decion_tree

cols <- c("CountryName","ITEM_NAME","ShapeName","Order Conversion")
df1[cols]<- lapply(df1[cols], factor)
str(df1)

## tibble [5,820 × 6] (S3: tbl_df/tbl/data.frame)
##  $ CountryName     : Factor w/ 14 levels "AUSTRALIA","BELGIUM",..: 6 14 14
14 14 6 6 14 14 6 ...
##  $ QtyRequired     : num [1:5820] 1 1 2 1 1 1 1 1 1 1 ...
##  $ ITEM_NAME       : Factor w/ 11 levels "DOUBLE BACK",..: 4 4 4 4 4 1 1 4
4 4 ...
##  $ ShapeName       : Factor w/ 3 levels "REC","ROUND",..: 1 1 1 1 1 1 1 1
```

```
1 1 ...
##  $ AreaFt          : num [1:5820] 80 80 80 80 80 80 80 40 108 54 ...
##  $ Order Conversion: Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 1 2
...
##  - attr(*, ".internal.selfref")=<externalptr>

#decision tree
set.seed(60)
indx <- sample(2, nrow(df1), replace=TRUE, prob=c(0.5,0.5))#dividing the
dataset into training and test with 50% in train and 50% in test
train <-df1 [indx==1, ] #assigning all the rows with index 1 to train
test <- df1 [indx==2, ] #assigning all the rows with index 2 to test
library("rpart.plot")
tree_m1 <- rpart(`Order Conversion` ~ ., train, parms = list(split = "gini"
)) #constructing the decision tree using rpart print( tree_m1) #printing the
decision tree

plotcp(tree_m1)
```




```
rpart.plot(tree_m1)
```



```
tree_pred_class_1 <- predict(tree_m1, train, type = "class")#using predict
function to predict the classes of training data
trainerror_1 <- mean(tree_pred_class_1 != train$`Order Conversion`)
#calculating the training error
trainerror_1

## [1] 0.09051144

tree_pred_test_1 <- predict(tree_m1, newdata=test, type = "class")#using
predict function to predict the classes of test data
testerror_1 <- mean(tree_pred_test_1 != test$`Order Conversion`) #calculating
the test error
testerror_1

## [1] 0.1014747

difference <- testerror_1 - trainerror_1
difference

## [1] 0.01096328

CM <- table(tree_pred_test_1,test$`Order Conversion`)
print(CM)
```

```
##
## tree_pred_test_1   No  Yes
##              No  2220  239
##              Yes   50  339
```

```
#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
precision_test
```

```
## [1] 0.5865052
```

```
prop.table(table(df1$`Order Conversion`))
```

```
##
##        No       Yes
## 0.7991409 0.2008591
```

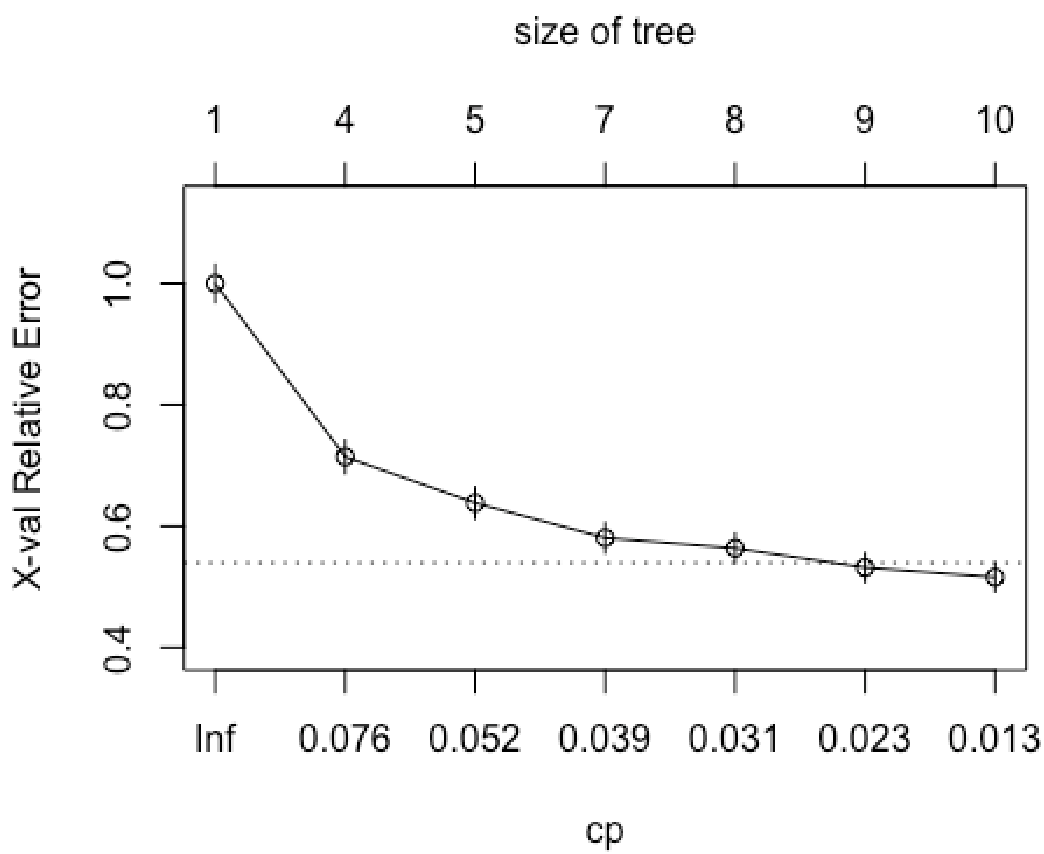
```
minsplt <- c(15, 51, 104) #assigning random vector values to minsplit
minbckt <- c(5, 17, 38) #assigning random vector values to minbckt
#looping through to try different combinations of minsplit and minbucket
for (i in minsplt){
for (j in minbckt){
tree_m1 <- rpart(`Order Conversion` ~ ., train, parms = list(split = "gini"
), control = rpart.control(minbucket = j, minsplit =i, cp=0.01))
tree_pred_class_1 <- predict(tree_m1, train, type = "class")#using predict
function to predict the classes of training data
trainerror_1 <- mean(tree_pred_class_1 != train$`Order Conversion`)
#calculating the training error
tree_pred_test_1 <- predict(tree_m1, test, type = "class")#using predict
function to predict the classes of test data
testerror_1 <- mean(tree_pred_test_1 != test$`Order Conversion`) #calculating
the test error
dif <- testerror_1-trainerror_1 #finding out the difference between test
error and training error
CM <- table(tree_pred_test_1, test$`Order Conversion`)
print(CM)
#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
print(precision_test)
}}
```

```
##
## tree_pred_test_1   No  Yes
```

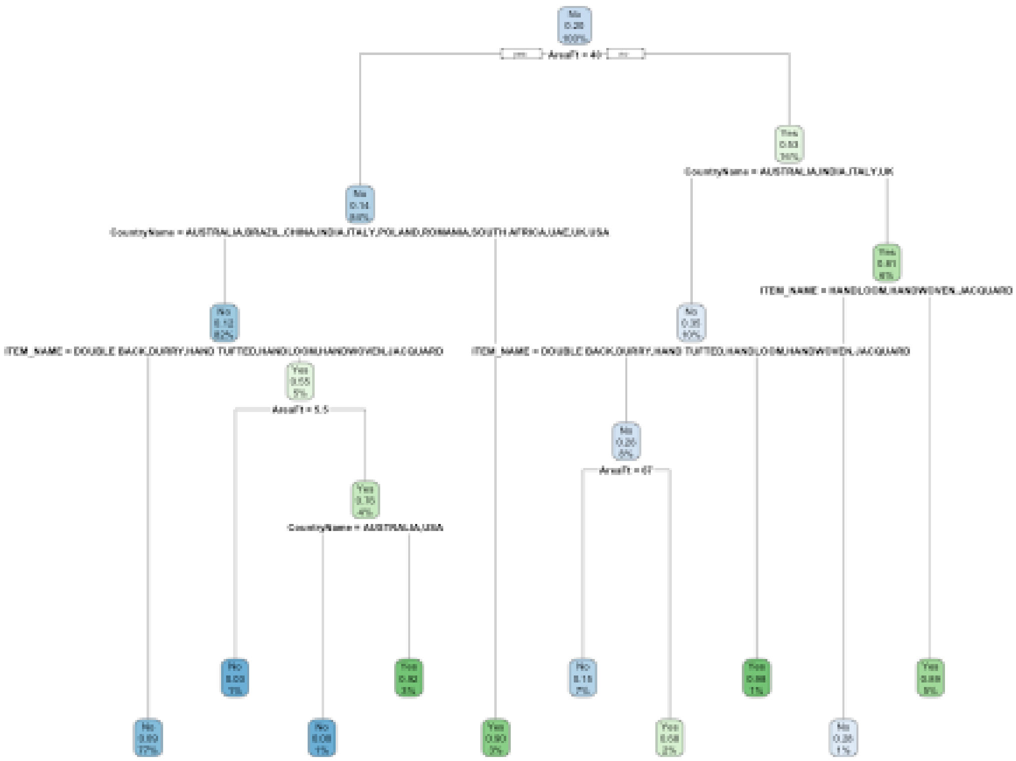
```
##              No  2220  239
##              Yes   50  339
## [1] 0.5865052
##
## tree_pred_test_1   No  Yes
##              No  2220  239
##              Yes   50  339
## [1] 0.5865052
##
## tree_pred_test_1   No  Yes
##              No  2180  249
##              Yes   90  329
## [1] 0.5692042
##
## tree_pred_test_1   No  Yes
##              No  2220  239
##              Yes   50  339
## [1] 0.5865052
##
## tree_pred_test_1   No  Yes
##              No  2220  239
##              Yes   50  339
## [1] 0.5865052
##
## tree_pred_test_1   No  Yes
##              No  2180  249
##              Yes   90  329
## [1] 0.5692042
##
## tree_pred_test_1   No  Yes
##              No  2220  239
##              Yes   50  339
## [1] 0.5865052
##
## tree_pred_test_1   No  Yes
##              No  2220  239
##              Yes   50  339
## [1] 0.5865052
##
## tree_pred_test_1   No  Yes
##              No  2180  249
##              Yes   90  329
## [1] 0.5692042
```

```
#decision tree
set.seed(60)
indx <- sample(2, nrow(df1), replace=TRUE, prob=c(0.7,0.3))#dividing the
dataset into training and test with 50% in train and 50% in test
train <-df1 [indx==1, ] #assigning all the rows with index 1 to train
test <- df1 [indx==2, ] #assigning all the rows with index 2 to test
```

```
library("rpart.plot")
tree_m2 <- rpart(`Order Conversion` ~ ., train, parms = list(split = "gini"
)) #constructing the decision tree using rpart print( tree_m2) #printing the
decision tree
plotcp(tree_m2)
```




```
rpart.plot(tree_m2)
```



```
printcp(tree_m2)

##
## Classification tree:
## rpart(formula = `Order Conversion` ~ ., data = train, parms = list(split =
"gini"))
##
## Variables actually used in tree construction:
## [1] AreaFt      CountryName ITEM_NAME
##
## Root node error: 816/4042 = 0.20188
##
## n= 4042
##
##          CP nsplit rel error  xerror     xstd
## 1 0.097222      0   1.00000 1.00000 0.031274
## 2 0.058824      3   0.70833 0.71446 0.027373
## 3 0.046569      4   0.64951 0.63848 0.026107
## 4 0.031863      6   0.55637 0.58088 0.025068
## 5 0.030637      7   0.52451 0.56373 0.024743
```

```
## 6 0.017157      8   0.49387 0.53186 0.024121
## 7 0.010000      9   0.47672 0.51593 0.023799
```

```
tree_pred_class_2 <- predict(tree_m2, train, type = "class")#using predict
function to predict the classes of training data
trainerror_2 <- mean(tree_pred_class_2 != train$`Order Conversion`)
#calculating the training error
trainerror_2
```

```
## [1] 0.09623949
```

```
tree_pred_test_2 <- predict(tree_m2, newdata=test, type = "class")#using
predict function to predict the classes of test data
testerror_2 <- mean(tree_pred_test_2 != test$`Order Conversion`) #calculating
the test error
testerror_2
```

```
## [1] 0.09167604
```

```
difference <- testerror_2 - trainerror_2
difference
```

```
## [1] -0.004563445
```

```
CM <- table(tree_pred_test_2,test$`Order Conversion`)
print(CM)
```

```
##
## tree_pred_test_2   No  Yes
##               No  1393  131
##               Yes   32  222
```

```
#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
print(precision_test)
```

```
## [1] 0.6288952
```

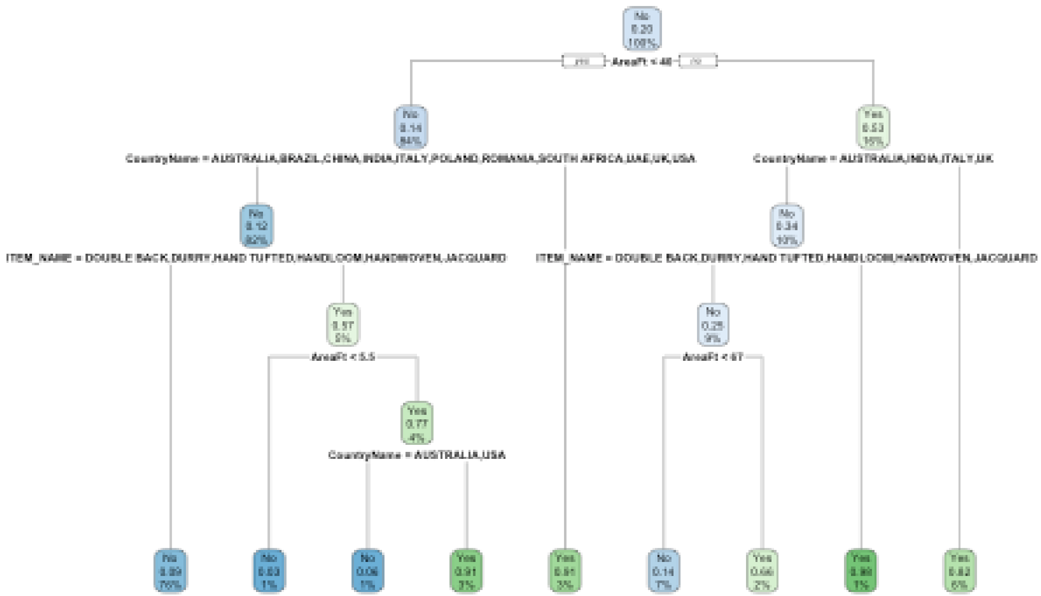
```
minsplt <- c(15, 51, 104) #assigning random vector values to minsplit
minbckt <- c(5, 17, 38) #assigning random vector values to minbckt
#looping through to try different combinations of minsplit and minbucket
for (i in minsplt){
for (j in minbckt){
tree_m2 <- rpart(`Order Conversion` ~ ., train, parms = list(split = "gini"
), control = rpart.control(minbucket = j, minsplit =i, cp=0.01))
tree_pred_class_2 <- predict(tree_m2, train, type = "class")#using predict
function to predict the classes of training data
trainerror_2 <- mean(tree_pred_class_2 != train$`Order Conversion`)
```

```
#calculating the training error
tree_pred_test_2 <- predict(tree_m2, test, type = "class")#using predict
function to predict the classes of test data
testerror_2 <- mean(tree_pred_test_2 != test$`Order Conversion`) #calculating
the test error
dif <- testerror_2-trainerror_2 #finding out the difference between test
error and training error
CM <- table(tree_pred_test_2, test$`Order Conversion`)
print(CM)
#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
print(precision_test)
}}
```

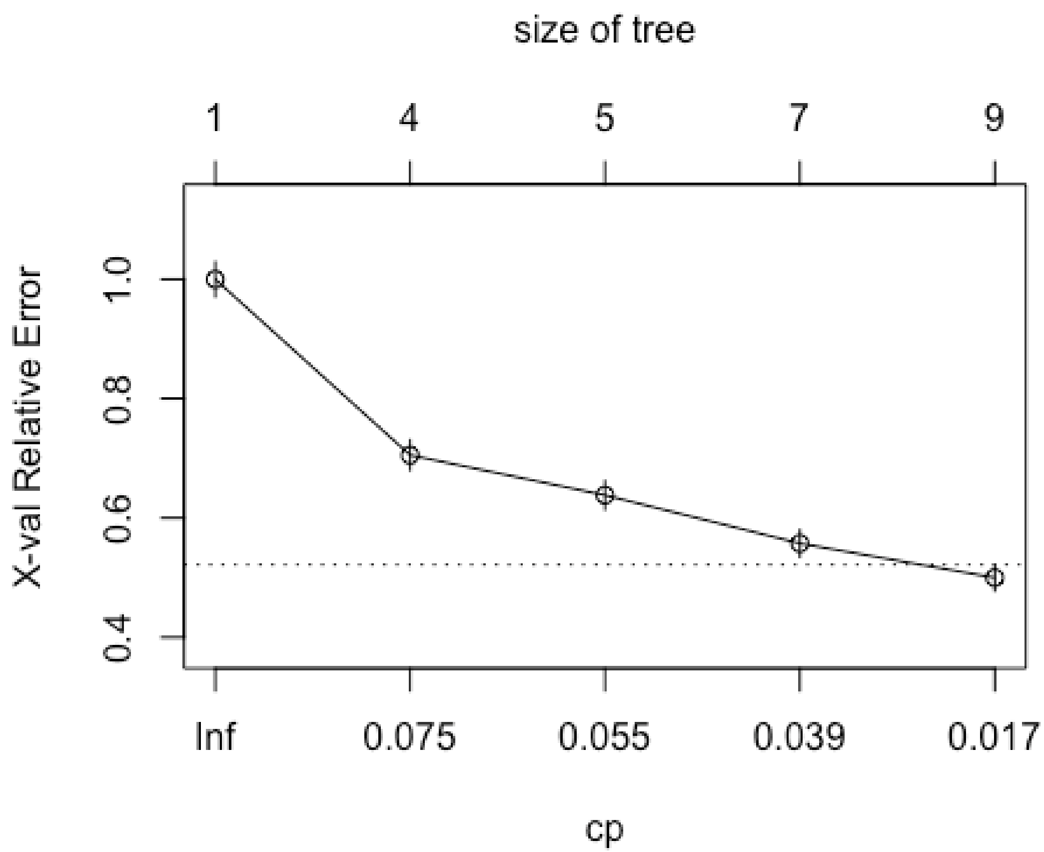
```
##
## tree_pred_test_2   No  Yes
##              No  1393  131
##              Yes   32  222
## [1] 0.6288952
##
## tree_pred_test_2   No  Yes
##              No  1393  131
##              Yes   32  222
## [1] 0.6288952
##
## tree_pred_test_2   No  Yes
##              No  1379  119
##              Yes   46  234
## [1] 0.6628895
##
## tree_pred_test_2   No  Yes
##              No  1393  131
##              Yes   32  222
## [1] 0.6288952
##
## tree_pred_test_2   No  Yes
##              No  1393  131
##              Yes   32  222
## [1] 0.6288952
##
## tree_pred_test_2   No  Yes
##              No  1379  119
##              Yes   46  234
## [1] 0.6628895
##
## tree_pred_test_2   No  Yes
```

```
##              No  1393  131
##              Yes   32  222
## [1] 0.6288952
##
## tree_pred_test_2   No  Yes
##              No  1393  131
##              Yes   32  222
## [1] 0.6288952
##
## tree_pred_test_2   No  Yes
##              No  1379  119
##              Yes   46  234
## [1] 0.6628895
```

```
#decision tree
set.seed(60)
indx <- sample(2, nrow(df1), replace=TRUE, prob=c(0.8,0.2))#dividing the
dataset into training and test with 50% in train and 50% in test
train <- df1 [indx==1, ] #assigning all the rows with index 1 to train
test <- df1 [indx==2, ] #assigning all the rows with index 2 to test
library("rpart.plot")
tree_m3 <- rpart(`Order Conversion` ~ ., train, parms = list(split = "gini"
)) #constructing the decision tree using rpart print( tree_m2) #printing the
decision tree
rpart.plot(tree_m3)
```




```
plotcp(tree_m3)
```




```
printcp(tree_m3)

##
## Classification tree:
## rpart(formula = `Order Conversion` ~ ., data = train, parms = list(split =
"gini"))
##
## Variables actually used in tree construction:
## [1] AreaFt      CountryName ITEM_NAME
##
## Root node error: 940/4631 = 0.20298
##
## n= 4631
##
##          CP nsplit rel error  xerror     xstd
## 1 0.098936      0   1.00000 1.00000 0.029119
## 2 0.057447      3   0.70319 0.70532 0.025356
## 3 0.052128      4   0.64574 0.63830 0.024312
## 4 0.029787      6   0.54149 0.55745 0.022933
## 5 0.010000      8   0.48191 0.50000 0.021862

tree_pred_class_3 <- predict(tree_m3, train, type = "class")#using predict
function to predict the classes of training data
```

```
trainerror_3 <- mean(tree_pred_class_3 != train$`Order Conversion`)
#calculating the training error
trainerror_3

## [1] 0.09781905

tree_pred_test_3 <- predict(tree_m3, newdata=test, type = "class")#using
predict function to predict the classes of test data
testerror_3 <- mean(tree_pred_test_3 != test$`Order Conversion`) #calculating
the test error
testerror_3

## [1] 0.09167368

difference <- testerror_3 - trainerror_3
difference

## [1] -0.00614537

CM <- table(tree_pred_test_3,test$`Order Conversion`)
print(CM)

##
## tree_pred_test_3  No Yes
##              No  937  86
##              Yes  23 143

#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
print(precision_test)

## [1] 0.6244541

minsplt <- c(15, 51, 104) #assigning random vector values to minsplit
minbckt <- c(5, 17, 38) #assigning random vector values to minbckt
#looping through to try different combinations of minsplit and minbucket
for (i in minsplt){
for (j in minbckt){
tree_m3 <- rpart(`Order Conversion` ~ ., train, parms = list(split = "gini"
), control = rpart.control(minbucket = j, minsplit =i, cp=0.01))
tree_pred_class_3 <- predict(tree_m2, train, type = "class")#using predict
function to predict the classes of training data
trainerror_3 <- mean(tree_pred_class_3 != train$`Order Conversion`)
#calculating the training error
tree_pred_test_3 <- predict(tree_m3, test, type = "class")#using predict
function to predict the classes of test data
testerror_3 <- mean(tree_pred_test_3 != test$`Order Conversion`) #calculating
the test error
```

```
dif <- testerror_3-trainerror_3 #finding out the difference between test
error and training error
CM <- table(tree_pred_test_3, test$`Order Conversion`)
print(CM)
#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
print(precision_test)
}}
```

```
##
## tree_pred_test_3  No Yes
##              No  937  86
##              Yes  23 143
## [1] 0.6244541
##
## tree_pred_test_3  No Yes
##              No  937  86
##              Yes  23 143
## [1] 0.6244541
##
## tree_pred_test_3  No Yes
##              No  933  85
##              Yes  27 144
## [1] 0.628821
##
## tree_pred_test_3  No Yes
##              No  937  86
##              Yes  23 143
## [1] 0.6244541
##
## tree_pred_test_3  No Yes
##              No  937  86
##              Yes  23 143
## [1] 0.6244541
##
## tree_pred_test_3  No Yes
##              No  933  85
##              Yes  27 144
## [1] 0.628821
##
## tree_pred_test_3  No Yes
##              No  937  86
##              Yes  23 143
## [1] 0.6244541
##
## tree_pred_test_3  No Yes
```

```
##                No  937  86
##                Yes  23 143
## [1] 0.6244541
##
## tree_pred_test_3  No Yes
##                No  933  85
##                Yes  27 144
## [1] 0.628821
```

When the order is actually not converted, but they are predicted as converted, there will be loss for Champo carpets as the samples made are wasted. Hence, we chose precision as our performance metric as it is crucial to reduce the False positives(actually not converted but predicted as converted)

| | Gini 50:50 | | Cp=0.01 |
|---|---|---|---|
| minsplit | minbucket | Precision | |
| 15 | 5 | 0.5865052 | |
| 15 | 17 | 0.5865052 | |
| 51 | 5 | 0.5865052 | |
| 51 | 17 | 0.5865052 | |
| 104 | 5 | 0.5865052 | |
| 104 | 17 | 0.5865052 | |
| 15 | 38 | 0.5692042 | |
| 51 | 38 | 0.5692042 | |
| 104 | 38 | 0.5692042 | |

| Gini 70:30 | | | Cp=0.01 |
|---|---|---|---|
| minsplit | minbucket | Precision | |
| 15 | 38 | 0.6628895 | |
| 51 | 38 | 0.6628895 | |
| 104 | 38 | 0.6628895 | |
| 15 | 5 | 0.6288952 | |
| 15 | 17 | 0.6288952 | |
| 51 | 5 | 0.6288952 | |
| 51 | 17 | 0.6288952 | |
| 104 | 5 | 0.6288952 | |
| 104 | 17 | 0.6288952 | |

| | Gini 80:20 | | Cp=0.01 |
|---|---|---|---|
| minsplit | minbucket | Precision | |
| 15 | 38 | 0.628821 | |
| 51 | 38 | 0.628821 | |
| 104 | 38 | 0.628821 | |
| 15 | 5 | 0.6244541 | |
| 15 | 17 | 0.6244541 | |
| 51 | 5 | 0.6244541 | |
| 51 | 17 | 0.6244541 | |
| 104 | 5 | 0.6244541 | |
| 104 | 17 | 0.6244541 | |

According to the decision trees that have been constructed above, we can conclude that 70:30 split with the highlighted pruning parameters gives us the best recall.

Constructing random forests with different ntree values and finding the best mtry for each model to derive a single model with the best performance

```
#Random Forest Model
rf <- randomForest(`Order Conversion` ~ ., data = df1, mtry = sqrt(ncol(df1)-
1), ntree = 100, proximity = T, importance = T)
imp_variables<-importance(rf, type = 2)
imp_variables

##               MeanDecreaseGini
## CountryName          397.73962
## QtyRequired           75.43505
## ITEM_NAME            325.66212
## ShapeName             11.19718
## AreaFt               437.28314

#Model1
set.seed(60)
indx <- sample(2, nrow(df1), replace=TRUE, prob=c(0.8,0.2))#dividing the
dataset into training and test with 80% in train and 20% in test
train <- df1 [indx==1, ] #assigning all the rows with index 1 to train
test <- df1 [indx==2, ] #assigning all the rows with index 2 to test
pr.err <- c()
for(mt in seq(1,ncol(train))) {
 rf1 <- randomForest(`Order Conversion`~., data = train,
       ntree = 100, mtry = ifelse(mt == ncol(train), mt-1, mt))
```

```
predicted <- predict(rf1, newdata = test, type = "class")
pr.err <- c(pr.err,mean(test$`Order Conversion` != predicted))
}
bestmtry <- which.min(pr.err)
print(bestmtry)

## [1] 2

rf1 <- randomForest(`Order Conversion`~., data = train, ntree = 100, mtry
=bestmtry)
print(rf1)

##
## Call:
##  randomForest(formula = `Order Conversion` ~ ., data = train,      ntree =
100, mtry = bestmtry)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 8.08%
## Confusion matrix:
##       No Yes class.error
## No  3596  95  0.02573828
## Yes  279 661  0.29680851

predicted <- predict(rf1, newdata = test, type = "class")
CM <- table(predicted, test$`Order Conversion`)
print(CM)

##
## predicted  No Yes
##       No  942  62
##       Yes  18 167

TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
precision_test

## [1] 0.7292576

#Model2
set.seed(60)
indx <- sample(2, nrow(df1), replace=TRUE, prob=c(0.8,0.2))#dividing the
dataset into training and test with 80% in train and 20% in test
train <- df1 [indx==1, ] #assigning all the rows with index 1 to train
test <- df1 [indx==2, ] #assigning all the rows with index 2 to test
pr.err <- c()
for(mt in seq(1,ncol(train))) {
```

```
 rf1 <- randomForest(`Order Conversion`~., data = train,
       ntree = 300, mtry = ifelse(mt == ncol(train), mt-1, mt))
predicted <- predict(rf1, newdata = test, type = "class")
pr.err <- c(pr.err,mean(test$`Order Conversion` != predicted))
}
bestmtry <- which.min(pr.err)
print(bestmtry)

## [1] 3

rf2 <- randomForest(`Order Conversion`~., data = train, ntree = 100, mtry
=bestmtry)
print(rf2)

##
## Call:
##  randomForest(formula = `Order Conversion` ~ ., data = train,      ntree =
100, mtry = bestmtry)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 7.99%
## Confusion matrix:
##       No Yes class.error
## No  3587 104  0.02817665
## Yes  266 674  0.28297872

predicted <- predict(rf2, newdata = test, type = "class")
CM <- table(predicted, test$`Order Conversion`)
print(CM)

##
## predicted  No Yes
##       No  943  64
##       Yes  17 165

TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
precision_test

## [1] 0.720524
```

According to the randomforest models constructed above, the randomforest model1 is considered as it gives us the better precision.

We can also determine the important variables by looking at the gini reduction of each variable. We can see that AreaFt has highest gini reduction and hence the most important variable.

```
#Logistic Regression

set.seed(60)

indx <- sample(2, nrow(df1), replace=TRUE, prob=c(0.8,0.2))#dividing the
dataset into training and test with 80% in train and 20% in test
train <- df1 [indx==1, ] #assigning all the rows with index 1 to train
test <-df1 [indx==2, ] #assigning all the rows with index 2 to test

logitModel <- glm(`Order Conversion` ~ ., data = train, family = "binomial")
summary(logitModel)

##
## Call:
## glm(formula = `Order Conversion` ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.1078  -0.5763  -0.2752  -0.1939   2.9399
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                  -3.969318   1.059539  -3.746 0.000179 ***
## CountryNameBELGIUM            5.842083   1.110103   5.263 1.42e-07 ***
## CountryNameBRAZIL           -11.109894 441.372931  -0.025 0.979918
## CountryNameCANADA             5.410693   1.320485   4.098 4.18e-05 ***
## CountryNameCHINA            -10.993900 882.743997  -0.012 0.990063
## CountryNameINDIA              0.060527   1.043021   0.058 0.953724
## CountryNameISRAEL            17.903304 497.845236   0.036 0.971313
## CountryNameITALY             -0.904968   1.277944  -0.708 0.478856
## CountryNamePOLAND           -12.907589 624.194723  -0.021 0.983502
## CountryNameROMANIA            3.282605   1.209900   2.713 0.006665 **
## CountryNameSOUTH AFRICA     -10.834542 441.372514  -0.025 0.980416
## CountryNameUAE              -13.074579 624.194707  -0.021 0.983288
## CountryNameUK                 1.818146   1.058550   1.718 0.085873 .
## CountryNameUSA                1.520523   1.044529   1.456 0.145475
## QtyRequired                   0.001828   0.008499   0.215 0.829678
## ITEM_NAMEDURRY                0.163334   0.205398   0.795 0.426495
## ITEM_NAMEGUN TUFTED           2.567763   0.436026   5.889 3.89e-09 ***
## ITEM_NAMEHAND TUFTED         -0.113388   0.191289  -0.593 0.553345
## ITEM_NAMEHANDLOOM            -0.038127   0.362919  -0.105 0.916332
## ITEM_NAMEHANDWOVEN           -0.551874   0.252609  -2.185 0.028911 *
## ITEM_NAMEINDO-TIBBETAN       18.341305 624.194734   0.029 0.976558
## ITEM_NAMEJACQUARD            -0.321314   0.431676  -0.744 0.456670
## ITEM_NAMEKNOTTED              2.664355   0.257088  10.364  < 2e-16 ***
## ITEM_NAMEPOWER LOOM JACQUARD  5.236311   0.421876  12.412  < 2e-16 ***
```

```
## ITEM_NAMETABLE TUFTED          2.742235   0.468253   5.856 4.73e-09 ***
## ShapeNameROUND                 0.699944   0.392780   1.782 0.074745 .
## ShapeNameSQUARE                0.710171   0.737405   0.963 0.335514
## AreaFt                         0.057997   0.002640  21.970  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4672.7  on 4630  degrees of freedom
## Residual deviance: 3015.0  on 4603  degrees of freedom
## AIC: 3071
##
## Number of Fisher Scoring iterations: 13
```

```
anova(logitModel, test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: Order Conversion
##
## Terms added sequentially (first to last)
##
##
##              Df Deviance Resid. Df Resid. Dev Pr(>Chi)
## NULL                          4630     4672.7
## CountryName 13   457.41      4617     4215.3   <2e-16 ***
## QtyRequired  1     0.08      4616     4215.2   0.7772
## ITEM_NAME   10   521.23      4606     3694.0   <2e-16 ***
## ShapeName    2     0.10      4604     3693.9   0.9530
## AreaFt       1   678.90      4603     3015.0   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Pred <- predict(logitModel, newdata = test, type = "response")
err<-mean(Pred != test$`Order Conversion`)
err
```

```
## [1] 1
```

```
Class <- ifelse(Pred >= 0.5, "YES", "NO")
```

```
with(logitModel, null.deviance - deviance)
```

```
## [1] 1657.708
```

```
with(logitModel, df.null, df.residual)
```

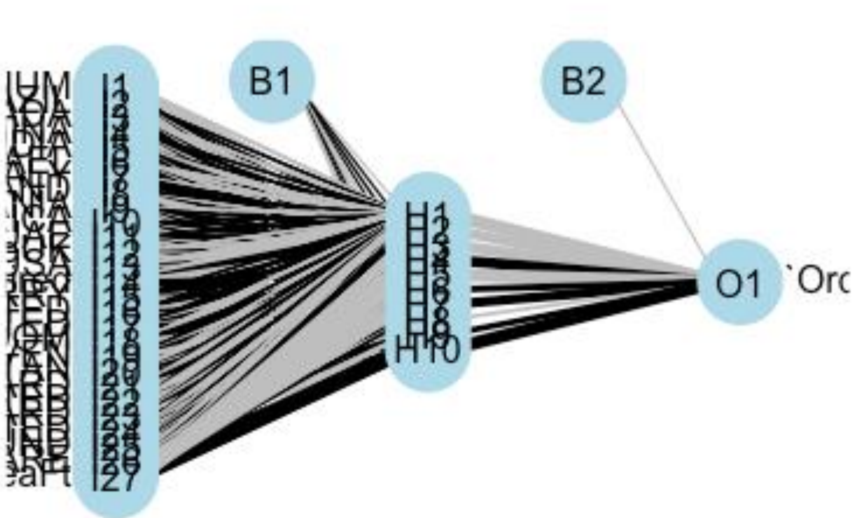
```
## [1] 4630
```

```
with(logitModel, pchisq(null.deviance - deviance, df.null - df.residual,
lower.tail = FALSE))

## [1] 0
```

According to the P values that we have got above, we can say that the variables CountryNameBELGIUM, CountryNameCANADA, CountryNameROMANIA, ITEM_NAMEGUN TUFTED, ITEM_NAMEHANDWOVEN, ITEM_NAMEKNOTTED, ITEM_NAMEPOWER LOOM JACQUARD, TEM_NAMETABLE TUFTED, AreaFt are significant as they are less than alpha (1%).

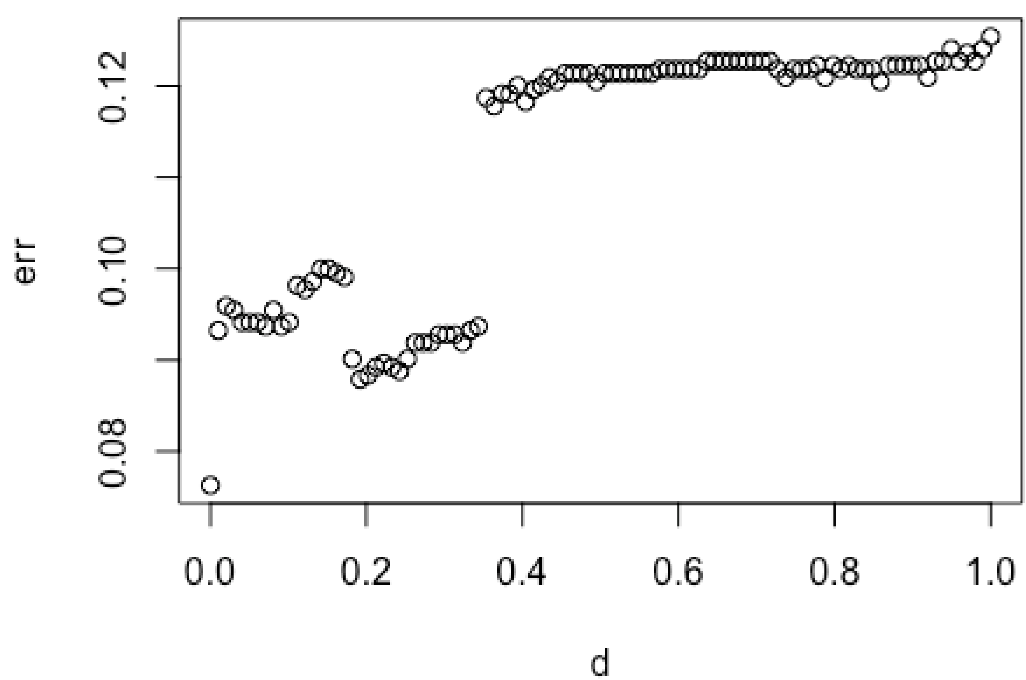
The anova table gives the residual deviance of null model and other variables. The more the difference in deviance between the null and residual, the best our model is doing against the null model. Hence, in the above table we can see that adding CountryName reduces the deviance. And AreaFt improves the AIC drastically and hence it is an important variable.

```
#Neural network
library(dplyr)
myscale <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}
df1 <- df1 %>% mutate_if(is.numeric, myscale)

indx <- sample(2, nrow(df1), replace=TRUE, prob=c(0.8,0.2))#dividing the
dataset into training and test with 80% in train and 20% in test
train <- df1 [indx==1, ] #assigning all the rows with index 1 to train
test <-df1 [indx==2, ] #assigning all the rows with index 2 to test

library(nnet)
nnModel <- nnet(`Order Conversion` ~ ., data = train, linout = FALSE,
                size = 10, decay = 0.01, maxit = 500)

summary(nnModel)

#nnModel$wts
#nnModel$fitted.values

#install.packages("NeuralNetTools")
library(NeuralNetTools)
plotnet(nnModel)
```




```
nn.preds = predict(nnModel, test)

nn.preds = as.factor(predict(nnModel, test, type = "class"))


CM <- table(nn.preds, test$`Order Conversion`)
print(CM)

##
## nn.preds  No Yes
##      No  915  76
##      Yes  36 177

#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
precision_test

## [1] 0.6996047
```

```
#decay parameter
set.seed(60)
indx <- sample(2, nrow(train), replace = T, prob = c(0.5, 0.5))
train2 <- train[indx == 1, ]
validation <- train[indx == 2, ]

err <- vector("numeric", 100)
d <- seq(0.0001, 1, length.out=100)
k = 1
for(i in d) {
  mymodel <- nnet(`Order Conversion` ~., data = train2, decay = i, size = 10,
maxit = 1000)
  pred.class <- predict(mymodel, newdata = validation, type = "class")
  err[k] <- mean(pred.class != validation$`Order Conversion`)
  k <- k +1
}

plot(d, err)
```

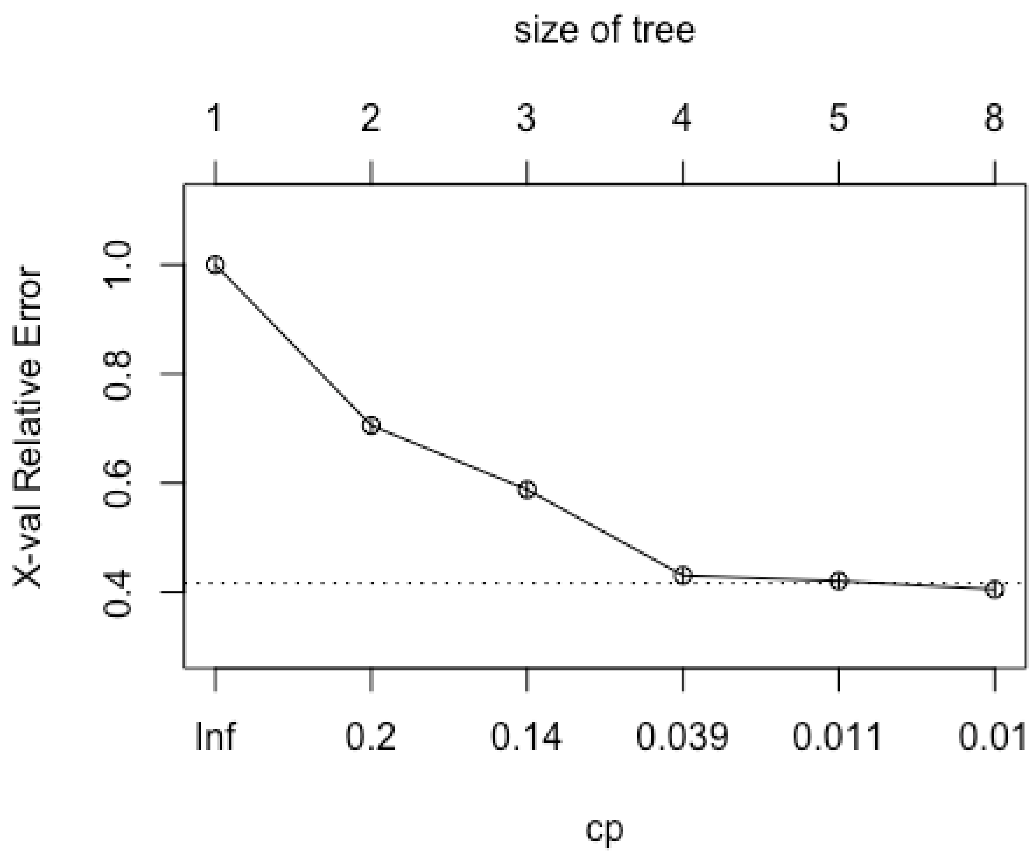



From the graph we can say that 0 is the best decay parameter as it gives the least error.

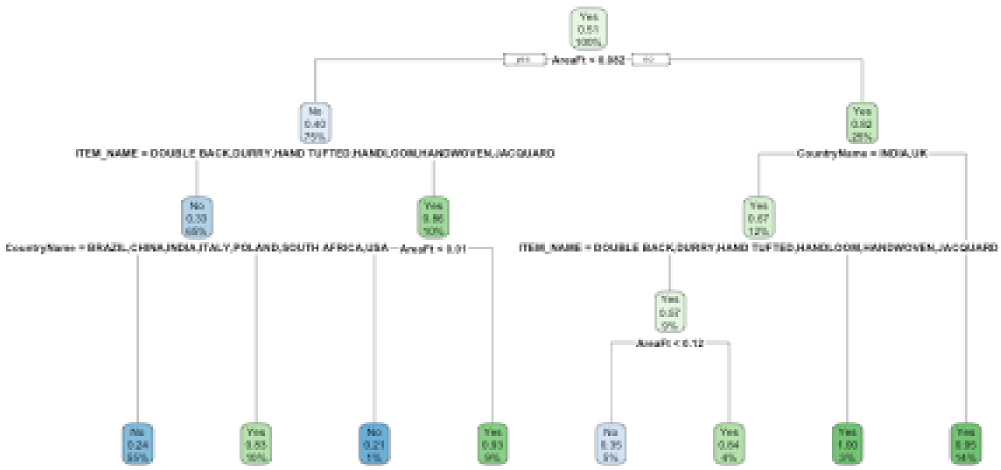
Balancing the data

```
library(ROSE)
```

```
## Loaded ROSE 0.0-4

colnames(df1)[which(names(df1) == "Order Conversion")] <- "Target"

#balancing the data
balanced_data <- ovun.sample(Target~.,data = df1,method = "over",N =
9600)$data
summary(balanced_data$Target)

##   No  Yes
## 4651 4949

#decision tree
set.seed(60)
indx <- sample(2, nrow(balanced_data), replace=TRUE,
prob=c(0.5,0.5))#dividing the dataset into training and test with 50% in
train and 50% in test
train <- balanced_data [indx==1, ] #assigning all the rows with index 1 to
train
test <- balanced_data [indx==2, ] #assigning all the rows with index 2 to
test
library("rpart.plot")
tree_m1 <- rpart(Target ~ ., train, parms = list(split = "gini" ))
#constructing the decision tree using rpart print( tree_m1) #printing the
decision tree

plotcp(tree_m1)
```




```
rpart.plot(tree_m1)
```



```
tree_pred_class_1 <- predict(tree_m1, train, type = "class")#using predict
function to predict the classes of training data
trainerror_1 <- mean(tree_pred_class_1 != train$Target) #calculating the
training error
trainerror_1

## [1] 0.1902208

tree_pred_test_1 <- predict(tree_m1, newdata=test, type = "class")#using
predict function to predict the classes of test data
testerror_1 <- mean(tree_pred_test_1 != test$Target) #calculating the test
error
testerror_1

## [1] 0.1874606

difference <- testerror_1 - trainerror_1
difference

## [1] -0.002760204

CM <- table(tree_pred_test_1,test$Target)
print(CM)
```

```
##
## tree_pred_test_1   No  Yes
##               No  2120  751
##               Yes  140 1742
```

```r
#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
precision_test
```

```
## [1] 0.6987565
```

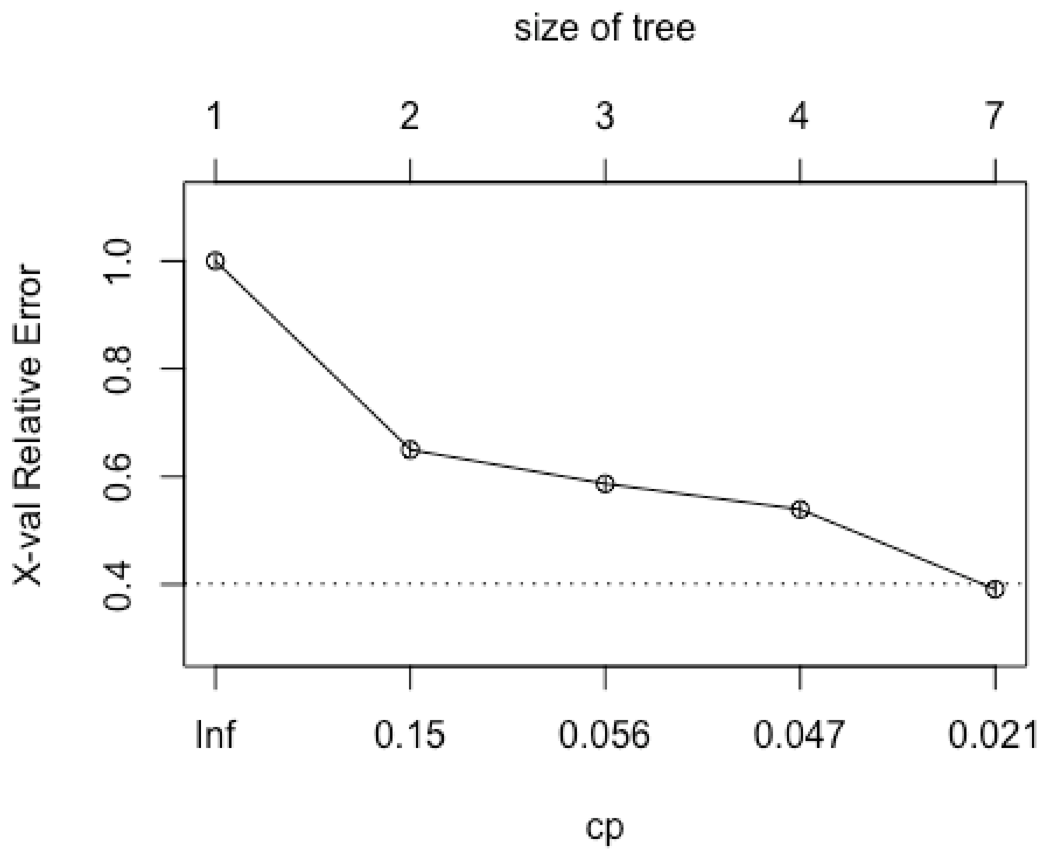
```r
minsplt <- c(15, 51, 104) #assigning random vector values to minsplit
minbckt <- c(5, 17, 38) #assigning random vector values to minbckt
#looping through to try different combinations of minsplit and minbucket
for (i in minsplt){
for (j in minbckt){
tree_m1 <- rpart(Target ~ ., train, parms = list(split = "gini" ), control =
rpart.control(minbucket = j, minsplit =i, cp=0.01))
tree_pred_class_1 <- predict(tree_m1, train, type = "class")#using predict
function to predict the classes of training data
trainerror_1 <- mean(tree_pred_class_1 != train$Target) #calculating the
training error
tree_pred_test_1 <- predict(tree_m1, test, type = "class")#using predict
function to predict the classes of test data
testerror_1 <- mean(tree_pred_test_1 != test$Target) #calculating the test
error
dif <- testerror_1-trainerror_1 #finding out the difference between test
error and training error
CM <- table(tree_pred_test_1, test$Target)
print(CM)
#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
print(precision_test)
}}
```

```
##
## tree_pred_test_1   No  Yes
##               No  2120  751
##               Yes  140 1742
## [1] 0.6987565
##
## tree_pred_test_1   No  Yes
##               No  2120  751
```

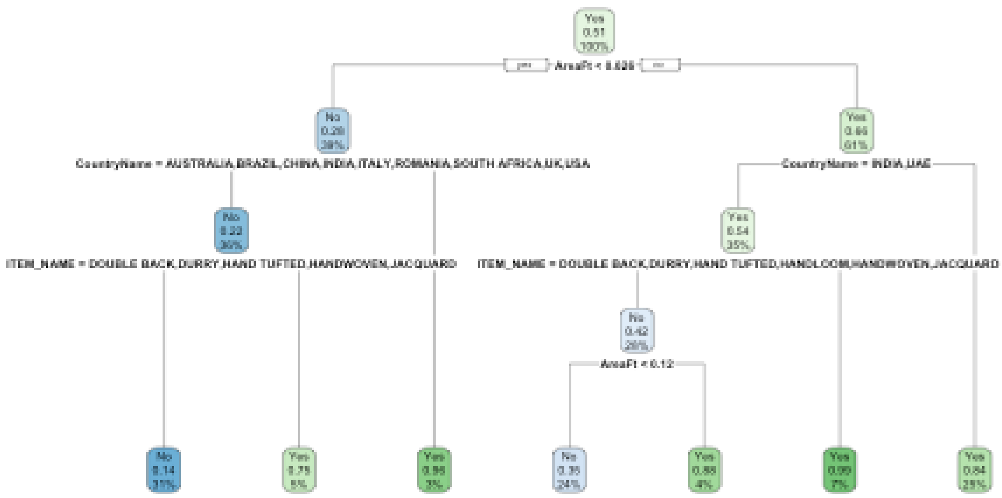
```
##                Yes  140 1742
## [1] 0.6987565
##
## tree_pred_test_1   No  Yes
##                No  2120  751
##                Yes  140 1742
## [1] 0.6987565
##
## tree_pred_test_1   No  Yes
##                No  2120  751
##                Yes  140 1742
## [1] 0.6987565
##
## tree_pred_test_1   No  Yes
##                No  2120  751
##                Yes  140 1742
## [1] 0.6987565
##
## tree_pred_test_1   No  Yes
##                No  2120  751
##                Yes  140 1742
## [1] 0.6987565
##
## tree_pred_test_1   No  Yes
##                No  2120  751
##                Yes  140 1742
## [1] 0.6987565
##
## tree_pred_test_1   No  Yes
##                No  2120  751
##                Yes  140 1742
## [1] 0.6987565
##
## tree_pred_test_1   No  Yes
##                No  2120  751
##                Yes  140 1742
## [1] 0.6987565
```

```
#decision tree
set.seed(60)
indx <- sample(2, nrow(balanced_data), replace=TRUE,
prob=c(0.7,0.3))#dividing the dataset into training and test with 50% in
train and 50% in test
train <- balanced_data [indx==1, ] #assigning all the rows with index 1 to
train
test <- balanced_data [indx==2, ] #assigning all the rows with index 2 to
test
library("rpart.plot")
tree_m2 <- rpart(Target ~ ., train, parms = list(split = "gini" ))
```

```
#constructing the decision tree using rpart print( tree_m2) #printing the decision tree

plotcp(tree_m2)
```




```
rpart.plot(tree_m2)
```




```
printcp(tree_m2)

##
## Classification tree:
## rpart(formula = Target ~ ., data = train, parms = list(split = "gini"))
##
## Variables actually used in tree construction:
## [1] AreaFt      CountryName ITEM_NAME
##
## Root node error: 3250/6699 = 0.48515
##
## n= 6699
##
##         CP nsplit rel error  xerror      xstd
## 1 0.350154      0   1.00000 1.00000 0.0125864
## 2 0.063077      1   0.64985 0.64985 0.0117010
## 3 0.049538      2   0.58677 0.58677 0.0113644
## 4 0.043692      3   0.53723 0.53908 0.0110675
## 5 0.010000      6   0.38431 0.39108 0.0098743

tree_pred_class_2 <- predict(tree_m2, train, type = "class")#using predict
function to predict the classes of training data
trainerror_2 <- mean(tree_pred_class_2 != train$Target) #calculating the
```

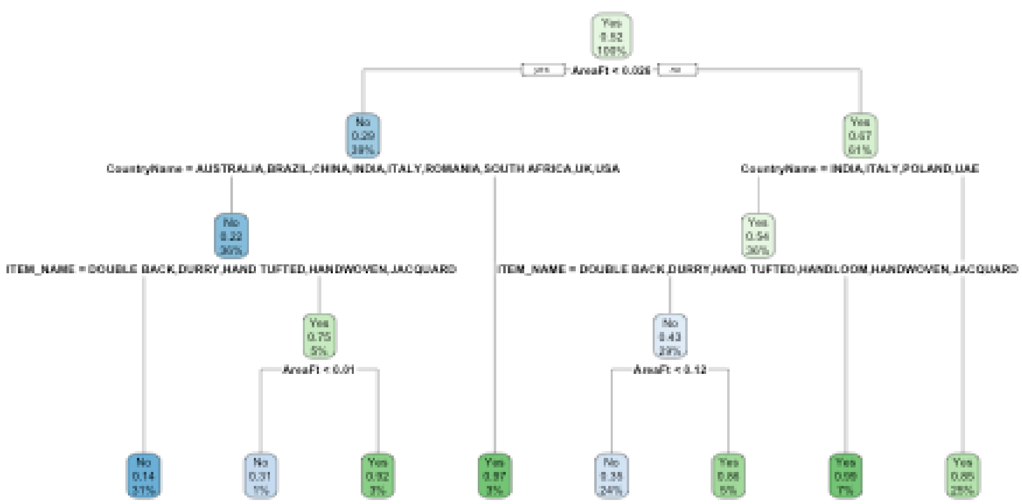
```
training error
trainerror_2

## [1] 0.1864457

tree_pred_test_2 <- predict(tree_m2, newdata=test, type = "class")#using
predict function to predict the classes of test data
testerror_2 <- mean(tree_pred_test_2 != test$Target) #calculating the test
error
testerror_2

## [1] 0.202344

difference <- testerror_2 - trainerror_2
difference

## [1] 0.01589828

CM <- table(tree_pred_test_2,test$Target)
print(CM)

##
## tree_pred_test_2   No  Yes
##              No  1192  378
##              Yes  209 1122

#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
print(precision_test)

## [1] 0.748

minsplt <- c(15, 51, 104) #assigning random vector values to minsplit
minbckt <- c(5, 17, 38) #assigning random vector values to minbckt
#looping through to try different combinations of minsplit and minbucket
for (i in minsplt){
for (j in minbckt){
tree_m2 <- rpart(Target ~ ., train, parms = list(split = "gini" ), control =
rpart.control(minbucket = j, minsplit =i, cp=0.01))
tree_pred_class_2 <- predict(tree_m2, train, type = "class")#using predict
function to predict the classes of training data
trainerror_2 <- mean(tree_pred_class_2 != train$Target) #calculating the
training error
tree_pred_test_2 <- predict(tree_m2, test, type = "class")#using predict
function to predict the classes of test data
testerror_2 <- mean(tree_pred_test_2 != test$Target) #calculating the test
error
dif <- testerror_2-trainerror_2 #finding out the difference between test
```

```
error and training error
CM <- table(tree_pred_test_2, test$Target)
print(CM)
#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
print(precision_test)
}}
```

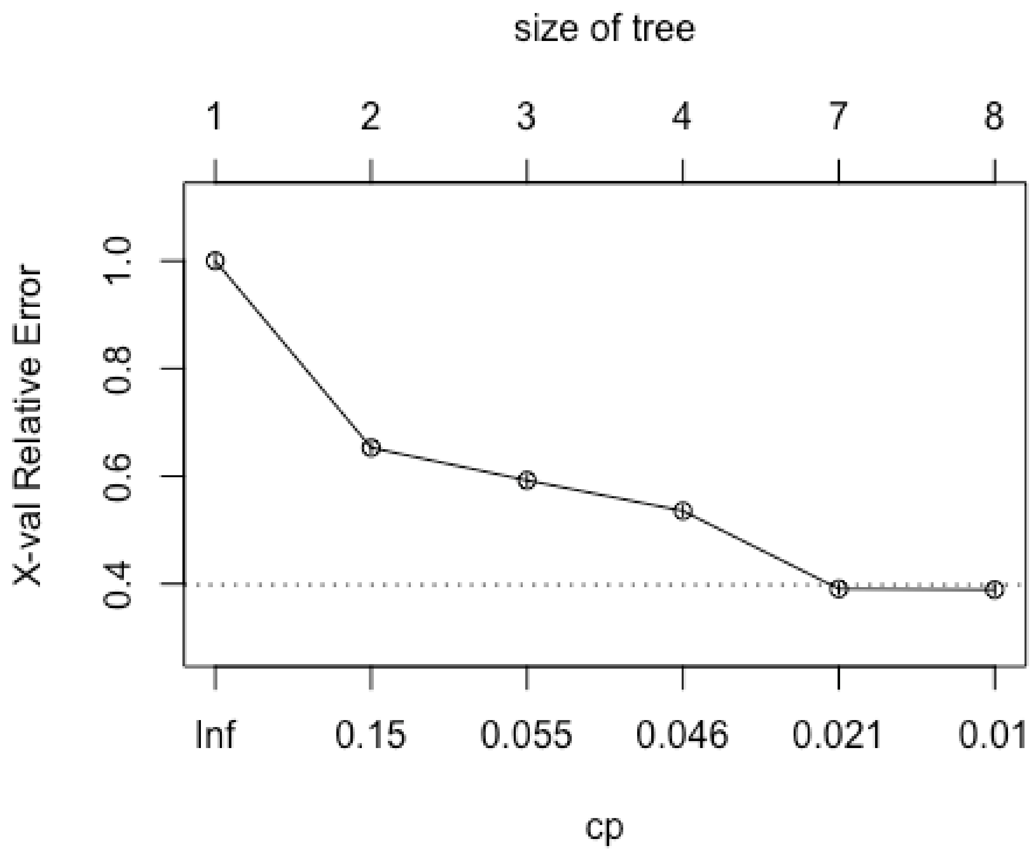
```
##
## tree_pred_test_2   No  Yes
##              No  1192  378
##              Yes  209 1122
## [1] 0.748
##
## tree_pred_test_2   No  Yes
##              No  1192  378
##              Yes  209 1122
## [1] 0.748
##
## tree_pred_test_2   No  Yes
##              No  1192  378
##              Yes  209 1122
## [1] 0.748
##
## tree_pred_test_2   No  Yes
##              No  1192  378
##              Yes  209 1122
## [1] 0.748
##
## tree_pred_test_2   No  Yes
##              No  1192  378
##              Yes  209 1122
## [1] 0.748
##
## tree_pred_test_2   No  Yes
##              No  1192  378
##              Yes  209 1122
## [1] 0.748
##
## tree_pred_test_2   No  Yes
##              No  1192  378
##              Yes  209 1122
## [1] 0.748
##
## tree_pred_test_2   No  Yes
##              No  1192  378
```

```
##               Yes  209 1122
## [1] 0.748
##
## tree_pred_test_2   No  Yes
##               No  1192  378
##               Yes  209 1122
## [1] 0.748
```

```
#decision tree
set.seed(60)
indx <- sample(2, nrow(balanced_data), replace=TRUE,
prob=c(0.8,0.2))#dividing the dataset into training and test with 50% in
train and 50% in test
train <- balanced_data [indx==1, ] #assigning all the rows with index 1 to
train
test <- balanced_data [indx==2, ] #assigning all the rows with index 2 to
test
library("rpart.plot")
tree_m3 <- rpart(Target ~ ., train, parms = list(split = "gini" ))
#constructing the decision tree using rpart print( tree_m2) #printing the
decision tree
rpart.plot(tree_m3)
```



```
plotcp(tree_m3)
```




```
printcp(tree_m3)

## 
## Classification tree:
## rpart(formula = Target ~ ., data = train, parms = list(split = "gini"))
## 
## Variables actually used in tree construction:
## [1] AreaFt      CountryName ITEM_NAME  
## 
## Root node error: 3702/7656 = 0.48354
## 
## n= 7656 
## 
##         CP nsplit rel error  xerror      xstd
## 1 0.347380      0   1.00000 1.00000 0.0118113
## 2 0.061588      1   0.65262 0.65289 0.0109856
## 3 0.049433      2   0.59103 0.59238 0.0106856
## 4 0.042950      3   0.54160 0.53566 0.0103545
## 5 0.010535      6   0.38709 0.39087 0.0092535
## 6 0.010000      7   0.37655 0.38925 0.0092388
```

```
tree_pred_class_3 <- predict(tree_m3, train, type = "class")#using predict
function to predict the classes of training data
trainerror_3 <- mean(tree_pred_class_3 != train$Target) #calculating the
training error
trainerror_3

## [1] 0.1820794

tree_pred_test_3 <- predict(tree_m3, newdata=test, type = "class")#using
predict function to predict the classes of test data
testerror_3 <- mean(tree_pred_test_3 != test$Target) #calculating the test
error
testerror_3

## [1] 0.1985597

difference <- testerror_3 - trainerror_3
difference

## [1] 0.01648026

CM <- table(tree_pred_test_3,test$Target)
print(CM)

##
## tree_pred_test_3  No Yes
##              No  834 271
##              Yes 115 724

#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
print(precision_test)

## [1] 0.7276382

minsplt <- c(15, 51, 104) #assigning random vector values to minsplit
minbckt <- c(5, 17, 38) #assigning random vector values to minbckt
#looping through to try different combinations of minsplit and minbucket
for (i in minsplt){
for (j in minbckt){
tree_m3 <- rpart(Target ~ ., train, parms = list(split = "gini" ), control =
rpart.control(minbucket = j, minsplit =i, cp=0.01))
tree_pred_class_3 <- predict(tree_m2, train, type = "class")#using predict
function to predict the classes of training data
trainerror_3 <- mean(tree_pred_class_3 != train$Target) #calculating the
training error
tree_pred_test_3 <- predict(tree_m3, test, type = "class")#using predict
function to predict the classes of test data
```

```r
testerror_3 <- mean(tree_pred_test_3 != test$Target) #calculating the test
error
dif <- testerror_3-trainerror_3 #finding out the difference between test
error and training error
CM <- table(tree_pred_test_3, test$Target)
print(CM)
#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
print(precision_test)
}}
```

```
##
## tree_pred_test_3  No Yes
##              No  834 271
##              Yes 115 724
## [1] 0.7276382
##
## tree_pred_test_3  No Yes
##              No  834 271
##              Yes 115 724
## [1] 0.7276382
##
## tree_pred_test_3  No Yes
##              No  834 271
##              Yes 115 724
## [1] 0.7276382
##
## tree_pred_test_3  No Yes
##              No  834 271
##              Yes 115 724
## [1] 0.7276382
##
## tree_pred_test_3  No Yes
##              No  834 271
##              Yes 115 724
## [1] 0.7276382
##
## tree_pred_test_3  No Yes
##              No  834 271
##              Yes 115 724
## [1] 0.7276382
##
## tree_pred_test_3  No Yes
##              No  834 271
##              Yes 115 724
## [1] 0.7276382
```

```
##
## tree_pred_test_3  No Yes
##               No  834 271
##               Yes 115 724
## [1] 0.7276382
##
## tree_pred_test_3  No Yes
##               No  834 271
##               Yes 115 724
## [1] 0.7276382
```

We can observe from the precision values that we got, that balanced data gives us better precision than unbalanced data.

| Gini 50:50 | | | Cp=0.01 |
|---|---|---|---|
| minsplit | minbucket | Precision | |
| 15 | 5 | 0.6843161 | |
| 15 | 17 | 0.6843161 | |
| 15 | 38 | 0.6843161 | |
| 51 | 5 | 0.6843161 | |
| 51 | 17 | 0.6843161 | |
| 51 | 38 | 0.6843161 | |
| 104 | 5 | 0.6843161 | |
| 104 | 17 | 0.6843161 | |
| 104 | 38 | 0.6843161 | |

| Gini 70:30 | | | Cp=0.01 |
|---|---|---|---|
| minsplit | minbucket | Precision | |
| 15 | 5 | 0.78 | |
| 15 | 17 | 0.78 | |
| 15 | 38 | 0.78 | |
| 51 | 5 | 0.78 | |
| 51 | 17 | 0.78 | |
| 51 | 38 | 0.78 | |
| 104 | 5 | 0.78 | |
| 104 | 17 | 0.78 | |
| 104 | 38 | 0.78 | |

| Gini 80:20 | | | Cp=0.01 |
|---|---|---|---|
| minsplit | minbucket | Recall | |
| 15 | 5 | 0.7778894 | |
| 15 | 17 | 0.7778894 | |
| 15 | 38 | 0.7778894 | |
| 51 | 5 | 0.7778894 | |
| 51 | 17 | 0.7778894 | |
| 51 | 38 | 0.7778894 | |
| 104 | 5 | 0.7778894 | |
| 104 | 17 | 0.7778894 | |
| 104 | 38 | 0.7778894 | |

We can observe from the above table that the 80:20 split gives us the best precision.

```
#Random Forest Model
set.seed(60)
indx <- sample(2, nrow(balanced_data), replace=TRUE,
prob=c(0.8,0.2))#dividing the dataset into training and test with 80% in
train and 20% in test
train <- balanced_data [indx==1, ] #assigning all the rows with index 1 to
train
test <- balanced_data [indx==2, ] #assigning all the rows with index 2 to
test
pr.err <- c()
for(mt in seq(1,ncol(train))) {
 rf1 <- randomForest(Target~., data = train,
       ntree = 100, mtry = ifelse(mt == ncol(train), mt-1, mt))
predicted <- predict(rf1, newdata = test, type = "class")
pr.err <- c(pr.err,mean(test$Target != predicted))
}
bestmtry <- which.min(pr.err)
print(bestmtry)

## [1] 5

rf1 <- randomForest(Target~., data = train, ntree = 100, mtry =bestmtry)
print(rf1)

##
## Call:
##  randomForest(formula = Target ~ ., data = train, ntree = 100,      mtry =
```

```
bestmtry)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 12.16%
```

```
predicted <- predict(rf1, newdata = test, type = "class")
CM <- table(predicted, test$Target)
print(CM)
```

```
##
## predicted  No Yes
##       No  880 172
##       Yes  69 823
```

```
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
precision_test
```

```
## [1] 0.8271357
```

```
set.seed(60)
indx <- sample(2, nrow(balanced_data), replace=TRUE,
prob=c(0.8,0.2))#dividing the dataset into training and test with 80% in
train and 20% in test
train <- balanced_data [indx==1, ] #assigning all the rows with index 1 to
train
test <- balanced_data [indx==2, ] #assigning all the rows with index 2 to
test
pr.err <- c()
for(mt in seq(1,ncol(train))) {
 rf1 <- randomForest(Target~., data = train,
       ntree = 300, mtry = ifelse(mt == ncol(train), mt-1, mt))
predicted <- predict(rf1, newdata = test, type = "class")
pr.err <- c(pr.err,mean(test$Target != predicted))
}
bestmtry <- which.min(pr.err)
print(bestmtry)
```

```
## [1] 4
```

```
rf2 <- randomForest(Target~., data = train, ntree = 100, mtry =bestmtry)
print(rf2)
```

```
##
## Call:
##  randomForest(formula = Target ~ ., data = train, ntree = 100,      mtry =
```

```
bestmtry)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 11.99%
## Confusion matrix:
##       No  Yes class.error
## No  3452  250  0.06753106
## Yes  668 3286  0.16894284
```

```
predicted <- predict(rf2, newdata = test, type = "class")
CM <- table(predicted, test$Target)
print(CM)
```

```
##
## predicted  No Yes
##       No  885 175
##       Yes  64 820
```

```
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
precision_test
```

```
## [1] 0.8241206
```

There is a drastic improvement in precision when balanced data is used for randomforest. On balanced data, the randomForest Model2 with 300 trees gives better precision

```
#Logistic Regression

set.seed(60)

indx <- sample(2, nrow(balanced_data), replace=TRUE,
prob=c(0.8,0.2))#dividing the dataset into training and test with 80% in
train and 20% in test
train <- balanced_data [indx==1, ] #assigning all the rows with index 1 to
train
test <-balanced_data [indx==2, ] #assigning all the rows with index 2 to test

logitModel <- glm(Target ~ ., data = train, family = "binomial")
summary(logitModel)
```

```
##
## Call:
## glm(formula = Target ~ ., family = "binomial", data = train)
```

```
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.6168  -0.7443   0.0538   0.7345   2.4466
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                   -1.73015    0.57476  -3.010  0.00261 **
## CountryNameBELGIUM             5.56692    0.70592   7.886 3.12e-15 ***
## CountryNameBRAZIL            -13.51580  394.77524  -0.034  0.97269
## CountryNameCANADA              4.15756    0.91906   4.524 6.08e-06 ***
## CountryNameCHINA             -13.39621  882.74356  -0.015  0.98789
## CountryNameINDIA              -0.93033    0.56220  -1.655  0.09796 .
## CountryNameISRAEL             15.75427  354.28018   0.044  0.96453
## CountryNameITALY              -0.60759    0.66721  -0.911  0.36248
## CountryNamePOLAND              0.08240    1.35240   0.061  0.95142
## CountryNameROMANIA             2.21371    0.70408   3.144  0.00167 **
## CountryNameSOUTH AFRICA       -1.05206    1.20109  -0.876  0.38108
## CountryNameUAE               -15.63008  624.19408  -0.025  0.98002
## CountryNameUK                  1.07320    0.57629   1.862  0.06257 .
## CountryNameUSA                 0.55027    0.56406   0.976  0.32929
## QtyRequired                    9.18468    1.83327   5.010 5.44e-07 ***
## ITEM_NAMEDURRY                 0.36096    0.12786   2.823  0.00476 **
## ITEM_NAMEGUN TUFTED            2.81916    0.35017   8.051 8.23e-16 ***
## ITEM_NAMEHAND TUFTED          -0.03210    0.12273  -0.262  0.79368
## ITEM_NAMEHANDLOOM              0.15271    0.24291   0.629  0.52958
## ITEM_NAMEHANDWOVEN            -0.82262    0.16403  -5.015 5.30e-07 ***
## ITEM_NAMEINDO-TIBBETAN        15.65510  254.82673   0.061  0.95101
## ITEM_NAMEJACQUARD              0.05356    0.24970   0.215  0.83015
## ITEM_NAMEKNOTTED               3.08302    0.18518  16.649  < 2e-16 ***
## ITEM_NAMEPOWER LOOM JACQUARD   5.46627    0.37800  14.461  < 2e-16 ***
## ITEM_NAMETABLE TUFTED          3.37862    0.38821   8.703  < 2e-16 ***
## ShapeNameROUND                 0.75019    0.27554   2.723  0.00648 **
## ShapeNameSQUARE                0.88950    0.44240   2.011  0.04437 *
## AreaFt                        28.66382    0.89948  31.867  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 10605.2  on 7655  degrees of freedom
## Residual deviance:  6632.7  on 7628  degrees of freedom
## AIC: 6688.7
##
## Number of Fisher Scoring iterations: 13
```

```
confint(logitModel)
```

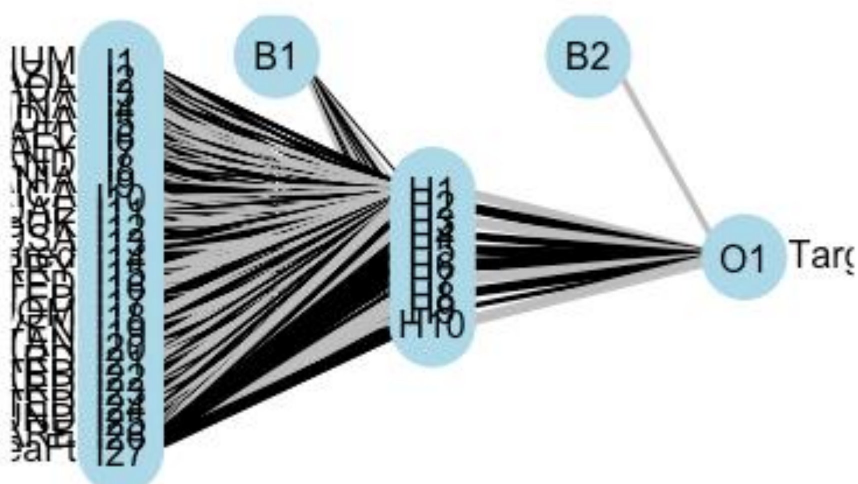
```
##                                     2.5 %      97.5 %
## (Intercept)                   -2.82671004  -0.5378689
```

```
## CountryNameBELGIUM             4.16693681   6.9747573
## CountryNameCANADA              2.46543237   6.2224458
## CountryNameINDIA              -2.10160847   0.1401458
## CountryNameISRAEL            355.60575763 410.5245517
## CountryNameITALY              -1.96391504   0.6760615
## CountryNamePOLAND             -2.51486182   3.3001728
## CountryNameROMANIA             0.80405904   3.5949433
## CountryNameSOUTH AFRICA       -3.59947636   1.1159540
## CountryNameUK                 -0.12200986   2.1727522
## CountryNameUSA                -0.62393990   1.6248051
## QtyRequired                    5.69163551  12.8286411
## ITEM_NAMEDURRY                 0.11188821   0.6132677
## ITEM_NAMEGUN TUFTED            2.15535762   3.5357081
## ITEM_NAMEHAND TUFTED          -0.27153457   0.2097729
## ITEM_NAMEHANDLOOM             -0.32474092   0.6286332
## ITEM_NAMEHANDWOVEN            -1.14569822  -0.5023626
## ITEM_NAMEJACQUARD             -0.43923118   0.5412555
## ITEM_NAMEKNOTTED               2.72550417   3.4519095
## ITEM_NAMEPOWER LOOM JACQUARD   4.78730611   6.2878616
## ITEM_NAMETABLE TUFTED          2.66522604   4.2028536
## ShapeNameROUND                 0.20928216   1.2923194
## ShapeNameSQUARE                0.01938992   1.7776316
## AreaFt                        26.92291146  30.4492539

with(logitModel, null.deviance - deviance)

## [1] 3972.466

with(logitModel, df.null, df.residual)

## [1] 7655

with(logitModel, pchisq(null.deviance - deviance, df.null - df.residual,
lower.tail = FALSE))

## [1] 0
```

When compared to unbalanced data, balanced data gives us more number of significant variables. Also, the balanced data gives better AIC than unbalanced data.

```
#Neural network
library(dplyr)
myscale <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}
balanced_data <- balanced_data %>% mutate_if(is.numeric, myscale)

indx <- sample(2, nrow(balanced_data), replace=TRUE,
prob=c(0.8,0.2))#dividing the dataset into training and test with 80% in
train and 20% in test
train <- balanced_data [indx==1, ] #assigning all the rows with index 1 to
```

```
train
test <-balanced_data [indx==2, ] #assigning all the rows with index 2 to test

library(nnet)
nnModel <- nnet(Target ~ ., data = train, linout = FALSE,
                size = 10, decay = 0.01, maxit = 500)

summary(nnModel)

nnModel$fitted.values

#install.packages("NeuralNetTools")
library(NeuralNetTools)
plotnet(nnModel)
```



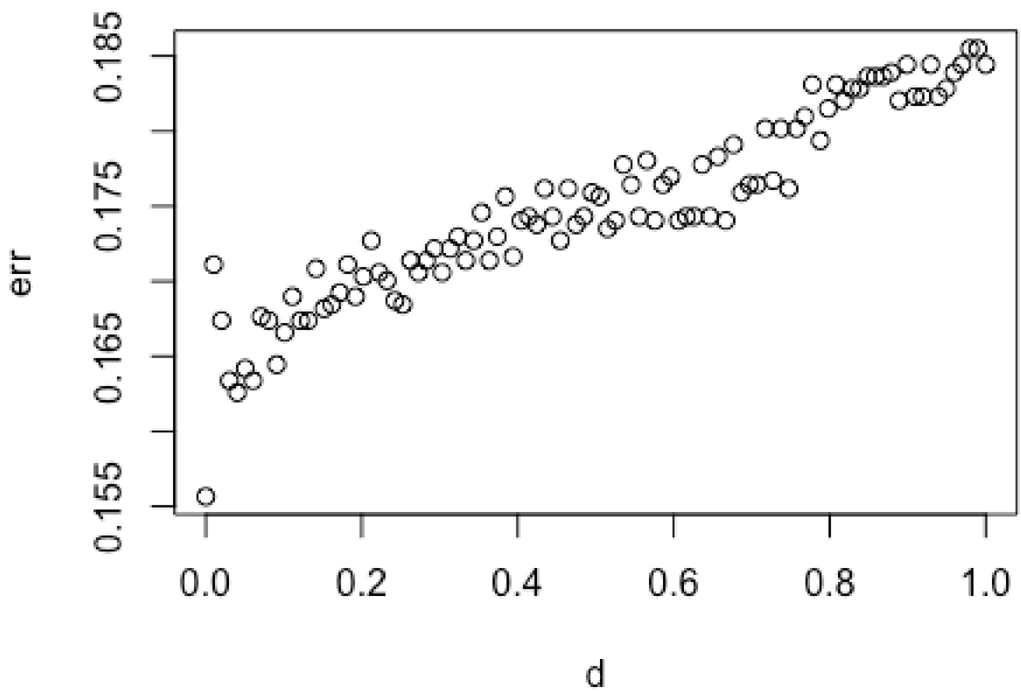

```
nn.preds = predict(nnModel, test)

nn.preds = as.factor(predict(nnModel, test, type = "class"))


CM <- table(nn.preds, test$Target)
print(CM)
```

```
##
## nn.preds  No Yes
##      No  900 189
##      Yes  70 782
```

```r
#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
precision_test
```

```
## [1] 0.8053553
```

```r
#decay parameter
set.seed(60)
indx <- sample(2, nrow(train), replace = T, prob = c(0.5, 0.5))
train2 <- train[indx == 1, ]
validation <- train[indx == 2, ]

err <- vector("numeric", 100)
d <- seq(0.0001, 1, length.out=100)
k = 1
for(i in d) {
  mymodel <- nnet(Target ~., data = train2, decay = i, size = 10, maxit =
1000)
  pred.class <- predict(mymodel, newdata = validation, type = "class")
  err[k] <- mean(pred.class != validation$Target)
  k <- k +1
}
```




The neural network done on balanced data gives better precision of 80% than unbalanced data.

From the graph we can see that the best decay parameter is 0 as it gives the least error.

```
#Ada boosting
set.seed(60)

indx <- sample(2, nrow(balanced_data), replace=TRUE,
prob=c(0.8,0.2))#dividing the dataset into training and test with 80% in
train and 20% in test
train <- balanced_data [indx==1, ] #assigning all the rows with index 1 to
train
test <-balanced_data [indx==2, ] #assigning all the rows with index 2 to test


library(adabag)

model <- boosting(Target~., data=train, boos=TRUE, mfinal=10)
print(names(model))

## [1] "formula"    "trees"      "weights"    "votes"      "prob"
## [6] "class"      "importance" "terms"      "call"
```

```
print(model$trees[1])

## [[1]]
## n= 7656
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 7656 3674 Yes (0.47988506 0.52011494)
##    2) AreaFt< 13.16665 3050  931 No (0.69475410 0.30524590)
##      4) CountryName=AUSTRALIA,BRAZIL,CHINA,INDIA,ITALY,ROMANIA,SOUTH
AFRICA,USA 2677  602 No (0.77512140 0.22487860)
##        8) ITEM_NAME=DOUBLE BACK,DURRY,HAND
TUFTED,HANDLOOM,HANDWOVEN,JACQUARD 2308  308 No (0.86655113 0.13344887)
##         16) QtyRequired< 9.5 2189  227 No (0.89629968 0.10370032) *
##         17) QtyRequired>=9.5 119   38 Yes (0.31932773 0.68067227) *
##        9) ITEM_NAME=GUN TUFTED,INDO-TIBBETAN,KNOTTED,POWER LOOM
JACQUARD,TABLE TUFTED 369   75 Yes (0.20325203 0.79674797)
##         18) AreaFt< 5.5 61    9 No (0.85245902 0.14754098) *
##         19) AreaFt>=5.5 308   23 Yes (0.07467532 0.92532468) *
##      5) CountryName=BELGIUM,CANADA,ISRAEL,UK 373   44 Yes (0.11796247
0.88203753) *
##    3) AreaFt>=13.16665 4606 1555 Yes (0.33760313 0.66239687)
##      6) CountryName=INDIA,ITALY,UAE 2697 1271 Yes (0.47126437 0.52873563)
##       12) ITEM_NAME=DOUBLE BACK,DURRY,HAND
TUFTED,HANDLOOM,HANDWOVEN,JACQUARD 2168  903 No (0.58348708 0.41651292)
##         24) AreaFt< 51.5 1823  621 No (0.65935272 0.34064728)
##           48) QtyRequired< 4.5 1766  574 No (0.67497169 0.32502831) *
##           49) QtyRequired>=4.5 57   10 Yes (0.17543860 0.82456140) *
##         25) AreaFt>=51.5 345   63 Yes (0.18260870 0.81739130) *
##       13) ITEM_NAME=GUN TUFTED,KNOTTED,POWER LOOM JACQUARD,TABLE TUFTED
529    6 Yes (0.01134216 0.98865784) *
##      7) CountryName=AUSTRALIA,BELGIUM,POLAND,ROMANIA,UK,USA 1909  284 Yes
(0.14876899 0.85123101) *

pred = predict(model, test) ##using predict function to predict the classes
of test data

CM<- print(pred$confusion)

##                  Observed Class
## Predicted Class  No Yes
##             No  852 221
##             Yes  97 774

TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
precision_test
```

```
## [1] 0.7778894
```

The precision on ada boosting method on imbalanced data is 77% which is better than decision trees.

### Q4. Data strategy

Data Preprocessing We consider the "data for clustering" sheet for customer segmentation using clustering.As a first step to preprocessing, since categorical variables cannot be used in the clustering algorithms we remove the row labels column. To be more specific, the range of categorical variables (e.g. row variables in this data) is discrete (one of the customers name), hence cannot be directly combined with a continuous variable and measured the distance in the same manner. Since any clustering algorithm interpret the closeness between data points based on a distance measure, it is important to reconcile all dimensions into a standard scale. An appropriate type of data transformation should be selected to align with the distribution of the data. For the case of this dataset we standardize all the variables between 0 and 1. With the variables we have, we can divide the dataset into two subsets. One which will have the variables Sum of QtyRequired, Sum of TotalArea and Sum of Amount. The other variables will be part of an other subset. This way we can cluster the similar customers in a better way. We can use principal component analysis to reduce the dimensions in the case of second subset. We'll also use elbow method to determine the number of actual clusters.

Benefit to business Clustering algorithm helps to better understand customers. Customer with comparable characteristics often have similar interest, thus business can benefit from this technique by creating tailored samples for each customer segment.Determine appropriate product pricing, Design an optimal distribution strategy can be the other benefits.

### Q5. Clustering Algorithms

For globular shaped clusters, center-based alogrithms (K means) are more adaptable where as if the cluster are irregular in shape and have a lot of noise then density based algorithms (DBSCAN) are more applicable. Since we have reduced the dimensions of the datasets as explained in Q4 using K means clustering algorithm will give us the desired results. K means algorithm uses the euclidean distance to interpret the closeness between data points. The number of clusters K can be found out using the elbow method.

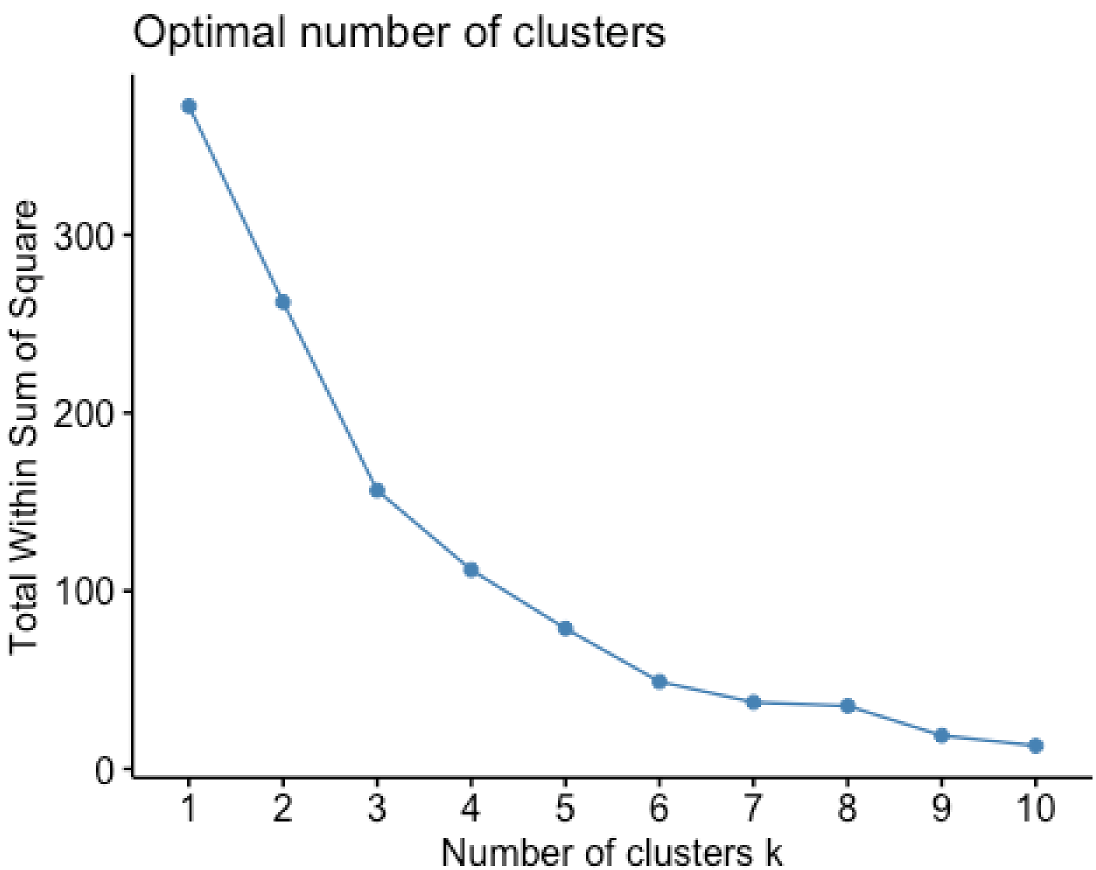
### Q6. Using K-means to solve the problem

```r
champo_carpets1 = read_excel("/Users/ashritacheetirala/Desktop/UIC/Sem 2/Data Mining/HW5/IMB881-XLS-ENG.xlsx",sheet=6)

#Remove the character variable column and also standardize all the remaining columns
champo_carpets = subset(champo_carpets1, select = -c(1) )
champo_carpets = sapply(champo_carpets, rescale)
```

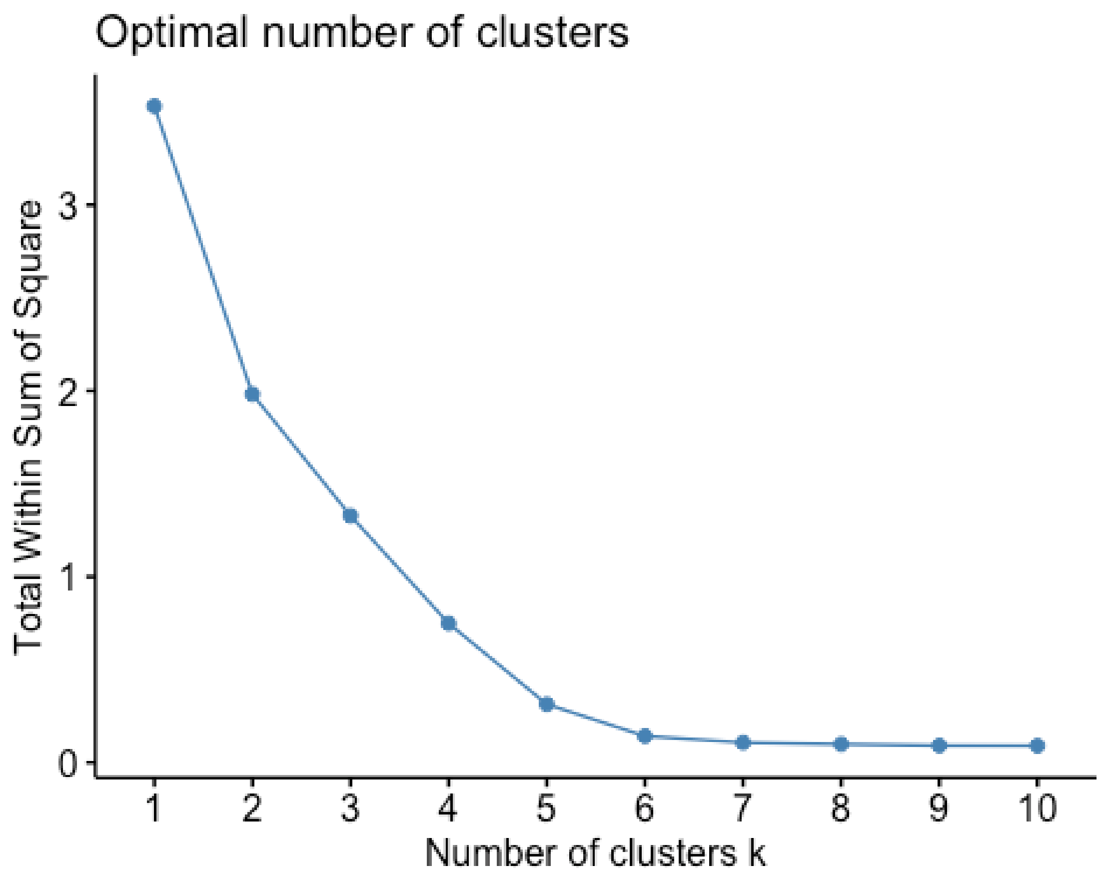
```
#Divide the dataframe to two subsets for k means clustering
carpets_sum = subset(champo_carpets, select = c(1:3))
carpets_type = subset(champo_carpets, select = c(4:13))

#Use pca for carpets_type subset to extract the principal features and
discard the remaining
carpets_type.pca <- prcomp(carpets_type, center = TRUE,scale. = TRUE)
summary(carpets_type.pca)

## Importance of components:
##                            PC1    PC2    PC3     PC4     PC5     PC6
PC7
## Standard deviation     1.8148 1.5172 1.1081 0.91615 0.89416 0.87327
0.72883
## Proportion of Variance 0.3293 0.2302 0.1228 0.08393 0.07995 0.07626
0.05312
## Cumulative Proportion  0.3293 0.5595 0.6823 0.76627 0.84622 0.92248
0.97560
##                            PC8    PC9    PC10
## Standard deviation     0.32941 0.3193 0.18299
## Proportion of Variance 0.01085 0.0102 0.00335
## Cumulative Proportion  0.98645 0.9967 1.00000
```

From summary it is evident that we can capture 85% of the information in the dataset (10 variables) can be encapsulated by just the first 5 Principal Components.

```
#pick the principle components from summary
carpets_transform = as.data.frame(-carpets_type.pca$x[,1:5])

#Consider the number of clusters for the carpets type dataset
fviz_nbclust(carpets_transform, kmeans, method = 'wss')
```

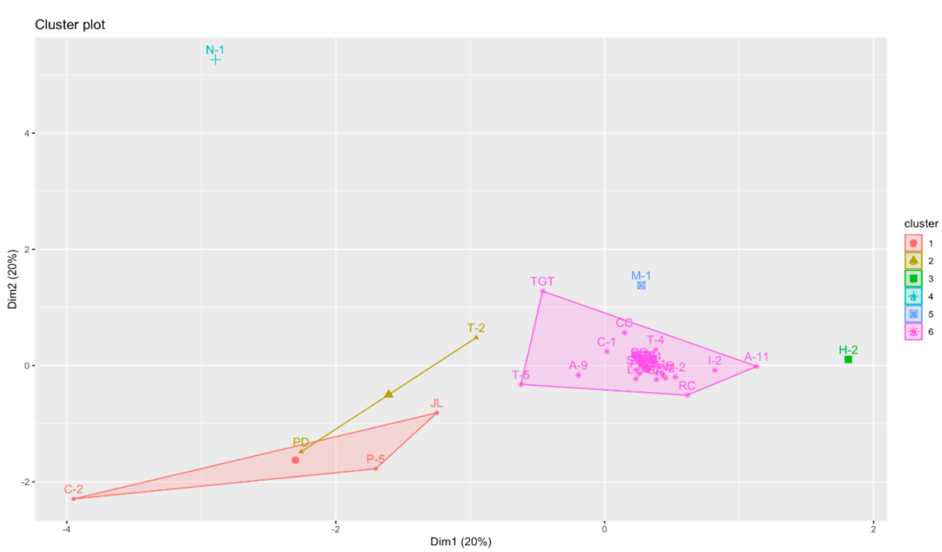



```
#Consider the number of clusters for the carpets sum dataset
fviz_nbclust(carpets_sum, kmeans, method = 'wss')
```




In both the cases we can see that there is no significant difference in the sum of squares from k value equal to 6. Hence we choose the number of clusters to be 6.

```
#Applying k-means on carpets_transform datset
kmeans_type = kmeans(carpets_transform, centers = 6, nstart = 100)
fviz_cluster(kmeans_type, data = carpets_transform)
```



**The characteristics of clusters**

Cluster 1 (C2, P5, JL) - Customers from this cluster belong to countries USA and UK. They prefer Durry, Hand Tufted and knotted carpets. We can also say that these customers prefer Chindi stripe and tikki designs.

Cluster2 (T2, PD)- Customers from this cluster belong to the countries Belgium and Italy. They place orders in small quantity for Jacquard type carpets and they prefer rectangular type of carpets.

Cluster 3(N1)- The customer belongs to the country USA and has ordered hand tufted carpets in a very high number thereby creating a huge revenue.

Cluster 4 (H2)- The customer belongs to the country USA and generally prefers Durry carpets that are round in shape and have a jute design.

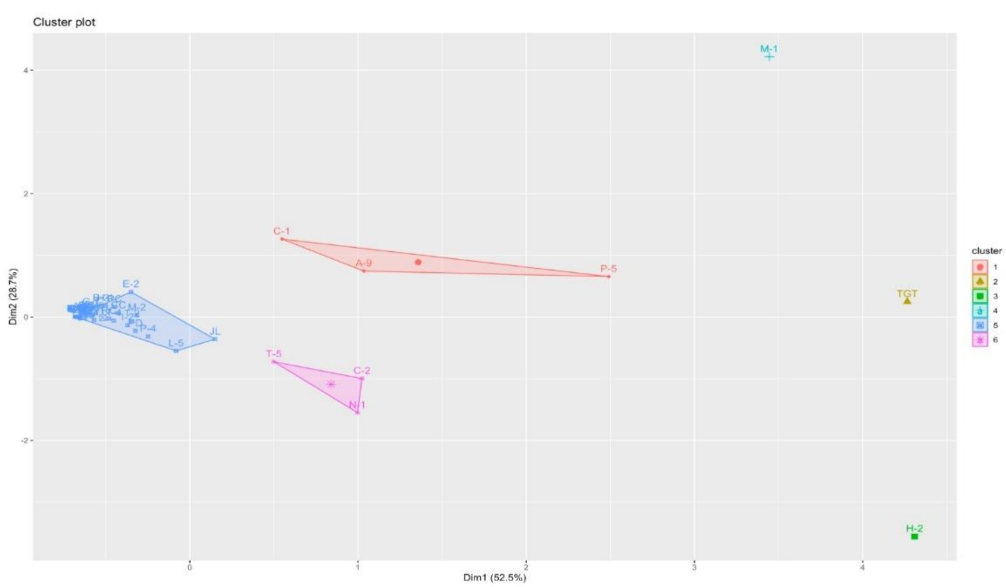
Cluster 5(M1)- This customer generally prefers ordering hand woven carpets in bulk.

Cluster 6 (rest of the customers)- Highest revenue generating customers in this belong to Australia and Brazil. Variety of items are preferred by the customers in this cluster but majorly sticking to rectangular carpets.

The significant variables are ITEM NAME, Country Name and CustomerCode.

```
rownames(carpets_transform) <- champo_carpets1$`Row Labels`
```

```
#Applying k-means on carpets_transform datset
kmeans_sum = kmeans(carpets_sum, centers = 6, nstart = 100)
fviz_cluster(kmeans_sum, data = carpets_sum)
```

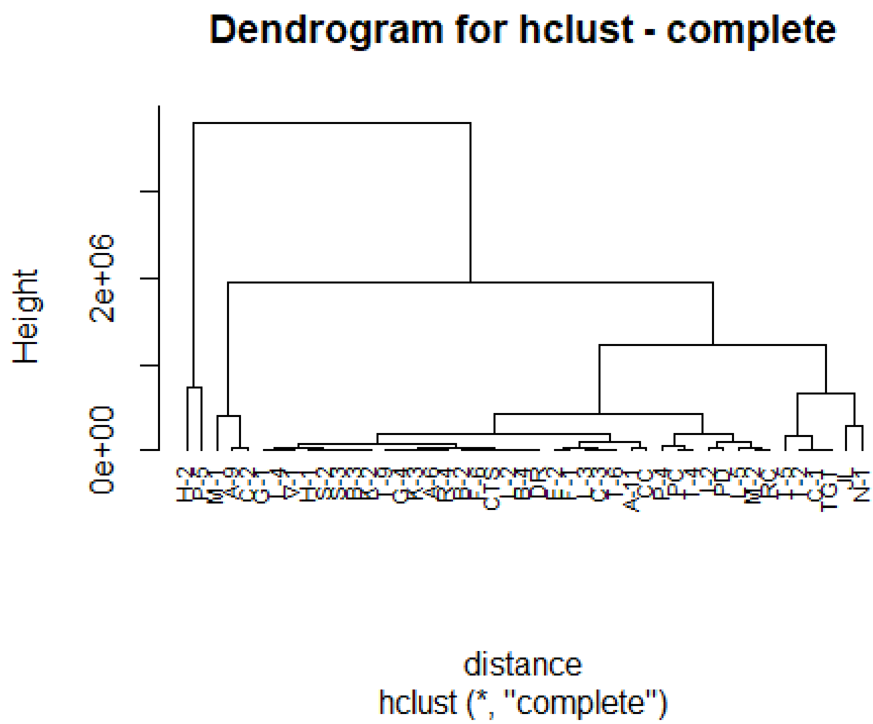



```
rownames(carpets_sum) <- champo_carpets1$`Row Labels`
```

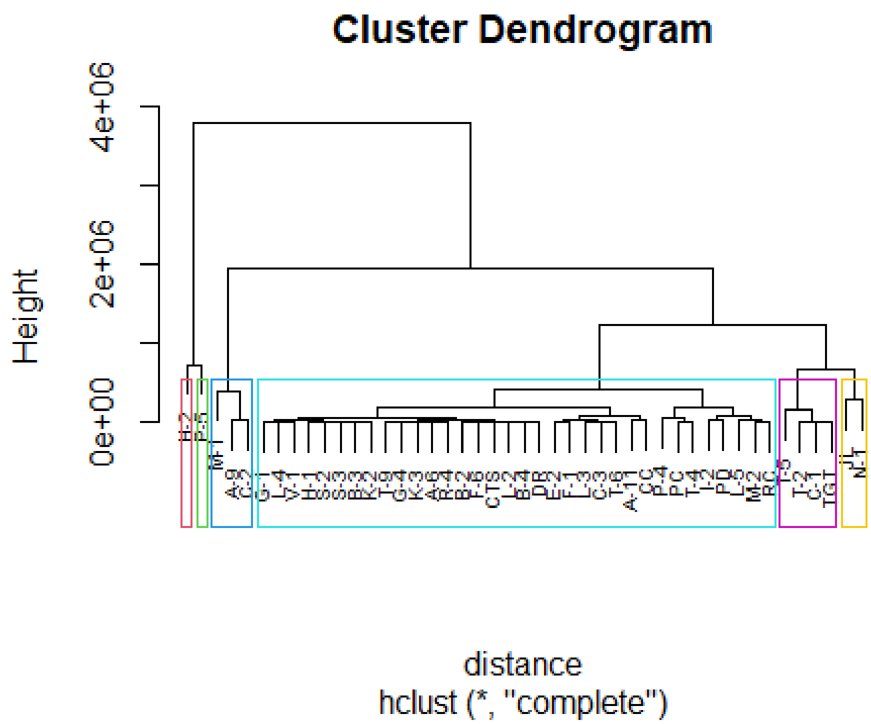
The significant variables are sumofquantity, sumoftotalarea and sumofamount.

We can see that majority of the customers belong to a single cluster in both the cases. The clustering is also very similar between the two graphs thus strengthening the confidence of it.

```
hc <- read_excel("C:\\Users\\pnanda4\\Downloads\\IMB881-XLS-
ENG.xlsx",sheet=6)
hc1<-hc
hc1<- hc1[2:14]
rownames(hc1) <- hc$`Row Labels`

myscale <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}
hc1 <- hc1 %>% mutate_if(is.numeric, myscale)

distance <- dist(hc1, method = "euclidean")

head(distance)

## [1]  179156.6 1406675.7  170592.9  126777.2  159161.6  382216.6

hcomplete <- hclust(distance, method = "complete")
plot(hcomplete, cex = 0.7, hang = -2, main = "Dendrogram for hclust -
complete")
```




```
clusters <- cutree(hcomplete, k =6)
#tapply(hc$QtyRequired, clusters, mean)
table(clusters)

## clusters
##  1  2  3  4  5  6
## 34  3  4  1  2  1

plot(hcomplete, cex = 0.6)
rect.hclust(hcomplete, k =6, border = 2:8)
```




Using the hierarchical model, we can say that T-5, T-2, C-1 and TGT are similar to each other. Similarly, JL and N-1 are similar each other. Also, M1, A-9 and C-2 belong to the same cluster as they are similar to each other. H-2 and P-5 belong to their respective clusters. The rest of the customers belong to one individual cluster.

**Q7. Recommender System**

```
#Reading the dataset and finding out the closest customers using pearson
correlation

recommendation <- read_excel("/Users/ashritacheetirala/Desktop/UIC/Sem 2/Data
Mining/HW5/IMB881-XLS-ENG.xlsx",sheet=5)
recommend <- recommendation[-1]
#View(recommend)
rownames(recommend) <- recommendation$Customer

#Converting the data frame into matrix

rec_mat <- as.matrix(recommend)
sim_mat <- cor(t(rec_mat), method="pearson")
sim_mat

##             H-2       P-5       M-1       A-9       C-2        JL       N-1
## H-2   1.0000000 0.8659981 0.5415715 0.6641305 0.9033397 0.7446284 0.6114096
## P-5   0.8659981 1.0000000 0.8336868 0.7751253 0.9589029 0.9344309 0.6756801
## M-1   0.5415715 0.8336868 1.0000000 0.8151151 0.7494426 0.9344207 0.7135211
```

```
## A-9  0.6641305 0.7751253 0.8151151 1.0000000 0.7693574 0.8737103 0.9664486
## C-2  0.9033397 0.9589029 0.7494426 0.7693574 1.0000000 0.9153526 0.6523960
## JL   0.7446284 0.9344309 0.9344207 0.8737103 0.9153526 1.0000000 0.7746348
## N-1  0.6114096 0.6756801 0.7135211 0.9664486 0.6523960 0.7746348 1.0000000
## T-5  0.9471737 0.9476862 0.6776479 0.7460193 0.9731465 0.8551048 0.6611148
## C-1  0.5502775 0.6682785 0.7740842 0.9719355 0.6390955 0.7906429 0.9747681
## T-2  0.5969608 0.7505879 0.9046281 0.9417065 0.7219792 0.9014601 0.9032514
## I-2  0.6547912 0.8043482 0.8779630 0.9761316 0.7900757 0.9185082 0.9312471
## PD   0.9401632 0.9472701 0.6575236 0.6740061 0.9625333 0.8265674 0.5755628
## L-5  0.9330793 0.9203102 0.5874798 0.6285056 0.9330136 0.7818332 0.5531404
## M-2  0.7176143 0.7795960 0.7678128 0.9759890 0.7688731 0.8515561 0.9700521
## RC   0.6904147 0.8644982 0.9101938 0.9536490 0.8293924 0.9392132 0.8827523
## P-4  0.9030605 0.9300383 0.6460863 0.6667890 0.9229486 0.8177634 0.5988357
## T-4  0.7114337 0.7934776 0.7927370 0.9781281 0.7809864 0.8617072 0.9638685
## PC   0.4772455 0.5756253 0.7060052 0.9414133 0.5564110 0.7190281 0.9660584
## A-11 0.7297883 0.8240350 0.7293014 0.6743015 0.8152687 0.7838678 0.5806570
## CC   0.4842250 0.5719846 0.6995288 0.9474876 0.5535602 0.7146102 0.9772065
##            T-5       C-1       T-2       I-2        PD       L-5       M-2
## H-2  0.9471737 0.5502775 0.5969608 0.6547912 0.9401632 0.9330793 0.7176143
## P-5  0.9476862 0.6682785 0.7505879 0.8043482 0.9472701 0.9203102 0.7795960
## M-1  0.6776479 0.7740842 0.9046281 0.8779630 0.6575236 0.5874798 0.7678128
## A-9  0.7460193 0.9719355 0.9417065 0.9761316 0.6740061 0.6285056 0.9759890
## C-2  0.9731465 0.6390955 0.7219792 0.7900757 0.9625333 0.9330136 0.7688731
## JL   0.8551048 0.7906429 0.9014601 0.9185082 0.8265674 0.7818332 0.8515561
## N-1  0.6611148 0.9747681 0.9032514 0.9312471 0.5755628 0.5531404 0.9700521
## T-5  1.0000000 0.6219683 0.6744844 0.7488884 0.9907584 0.9808896 0.7744172
## C-1  0.6219683 1.0000000 0.9375788 0.9524963 0.5442108 0.4968347 0.9668606
## T-2  0.6744844 0.9375788 1.0000000 0.9735411 0.6163044 0.5551117 0.9279183
## I-2  0.7488884 0.9524963 0.9735411 1.0000000 0.6874765 0.6388967 0.9638978
## PD   0.9907584 0.5442108 0.6163044 0.6874765 1.0000000 0.9866896 0.7068966
## L-5  0.9808896 0.4968347 0.5551117 0.6388967 0.9866896 1.0000000 0.6768874
## M-2  0.7744172 0.9668606 0.9279183 0.9638978 0.7068966 0.6768874 1.0000000
## RC   0.7948463 0.9153465 0.9433488 0.9740144 0.7482847 0.6947030 0.9426701
## P-4  0.9595940 0.5439550 0.6087854 0.6869860 0.9598137 0.9737393 0.7072836
## T-4  0.7852430 0.9660572 0.9355473 0.9724799 0.7218742 0.6867251 0.9920560
## PC   0.5524053 0.9880356 0.9083601 0.9175229 0.4655521 0.4270901 0.9457463
## A-11 0.8023475 0.5839128 0.6903659 0.7560527 0.8065837 0.7650626 0.6743700
## CC   0.5503088 0.9872845 0.9049166 0.9153931 0.4594361 0.4213774 0.9433600
##             RC       P-4       T-4        PC      A-11        CC
## H-2  0.6904147 0.9030605 0.7114337 0.4772455 0.7297883 0.4842250
## P-5  0.8644982 0.9300383 0.7934776 0.5756253 0.8240350 0.5719846
## M-1  0.9101938 0.6460863 0.7927370 0.7060052 0.7293014 0.6995288
## A-9  0.9536490 0.6667890 0.9781281 0.9414133 0.6743015 0.9474876
## C-2  0.8293924 0.9229486 0.7809864 0.5564110 0.8152687 0.5535602
## JL   0.9392132 0.8177634 0.8617072 0.7190281 0.7838678 0.7146102
## N-1  0.8827523 0.5988357 0.9638685 0.9660584 0.5806570 0.9772065
## T-5  0.7948463 0.9595940 0.7852430 0.5524053 0.8023475 0.5503088
## C-1  0.9153465 0.5439550 0.9660572 0.9880356 0.5839128 0.9872845
## T-2  0.9433488 0.6087854 0.9355473 0.9083601 0.6903659 0.9049166
## I-2  0.9740144 0.6869860 0.9724799 0.9175229 0.7560527 0.9153931
```

```
## PD   0.7482847 0.9598137 0.7218742 0.4655521 0.8065837 0.4594361
## L-5  0.6947030 0.9737393 0.6867251 0.4270901 0.7650626 0.4213774
## M-2  0.9426701 0.7072836 0.9920560 0.9457463 0.6743700 0.9433600
## RC   1.0000000 0.7344490 0.9439898 0.8655842 0.7434087 0.8607116
## P-4  0.7344490 1.0000000 0.7181670 0.4742203 0.7782477 0.4703558
## T-4  0.9439898 0.7181670 1.0000000 0.9435962 0.7315986 0.9407796
## PC   0.8655842 0.4742203 0.9435962 1.0000000 0.5137656 0.9961769
## A-11 0.7434087 0.7782477 0.7315986 0.5137656 1.0000000 0.5113069
## CC   0.8607116 0.4703558 0.9407796 0.9961769 0.5113069 1.0000000
```

The following recommendations can be made to the customers

Using the correlation matrix we can see that customers N1 and C1 have a very high correlation of 97%. Therefore after constructing the recommender matrix we can recommend N1 to order Knotted carpets in neutral shades.

We can also see that T-5 and PD are similar as they have correlation of 99%and we can recommend Hand Tufted carpets to PD that are round in shape and in shades of pink and blush pink.

Another set of similar customers are PC and CC as they have correlation of 99% and we can recommend handloom carpets to PC that are round and in shades of navy and blue.

****Q8 Final Recommendations****

Our final recommedation to Champo carpets would be to use all these different models as they provide different insights.

- We can suggest them the important factors that led to conversion of orders are AreaFt, CountryName, QtyRequired as we have seen the ANOVA table of logistic regression predicting them as significant features.

- We can use the recommender systems to recommend products to customers depending on their similarity with the other customers while Kmeans can help us understand various segments of customers that we have and then make better strategies to increase their conversion rate thereby targeting the customers.

- Using Association rules, we can also suggest them the items that go well with their purchase history.

- The company can also prioritize using balanced data as it can give more accurate and precise results.

- After running all the ML models above we can recommend randomForest for the Champo Carpets as it gives highest precision of 82%

### Association rules(extra credit)

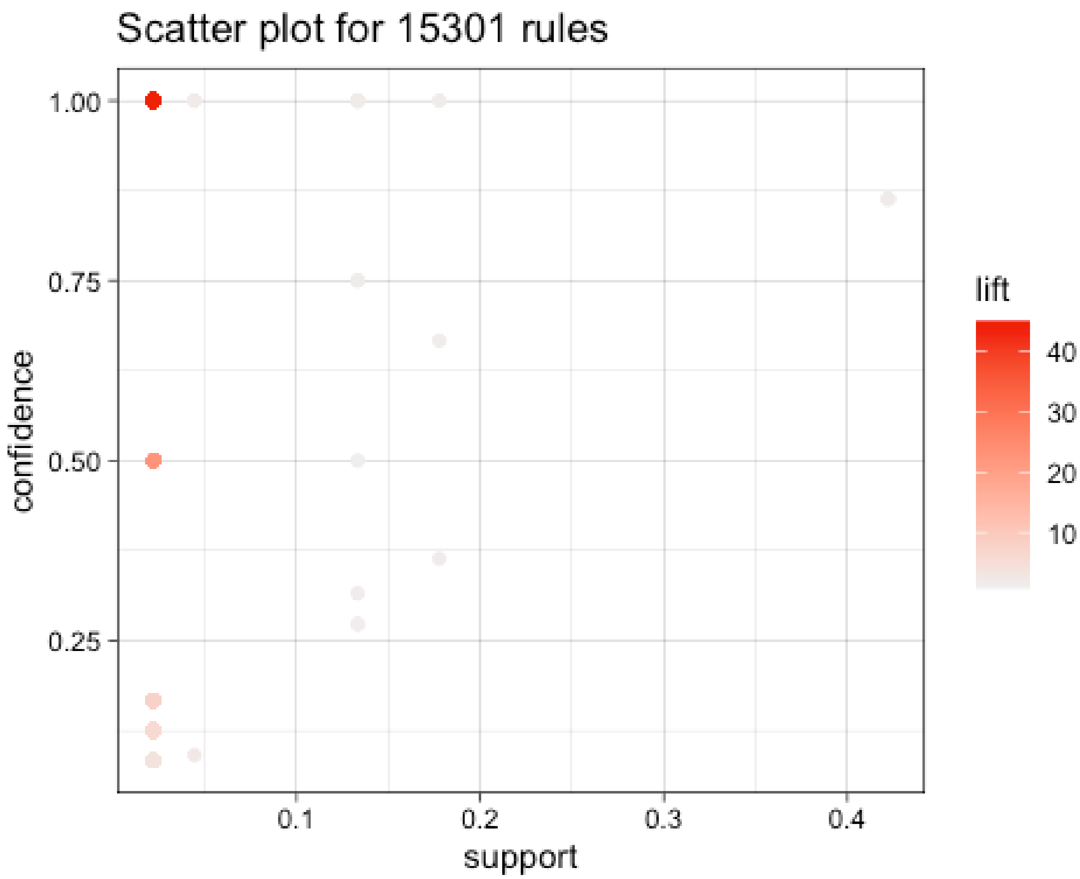
```
library(arules)

library(arulesViz)
ass <- read_excel("/Users/ashritacheetirala/Desktop/UIC/Sem 2/Data
Mining/HW5/IMB881-XLS-ENG.xlsx",sheet=7)
ass <- ass[2:8]
colnames(ass)

## [1] "Sum of QtyRequired" "Sum of TotalArea"   "Sum of Amount"
## [4] "DURRY"              "HANDLOOM"           "DOUBLE BACK"
## [7] "JACQUARD"

ass <- ass %>% mutate_if(is.numeric,as.character)
ass <- ass %>% mutate_if(is.character,as.factor)
rules <- apriori(ass, parameter = list(supp=0.001, minlen=3,
maxlen=5,conf=0.08))

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##        0.08    0.1    1 none FALSE            TRUE       5   0.001      3
##  maxlen target  ext
##       5  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 0
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[217 item(s), 45 transaction(s)] done [0.00s].
## sorting and recoding items ... [217 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5

## Warning in apriori(ass, parameter = list(supp = 0.001, minlen = 3, maxlen
= 5, :
## Mining stopped (maxlen reached). Only patterns up to a length of 5
returned!
```

```
##  done [0.00s].
## writing ... [15301 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
rules_conf <- sort (rules, by="confidence", decreasing=TRUE)
inspect(rules_conf[1:10])
```

```
##      lhs                                rhs                              support
confidence   coverage      lift count
## [1]  {Sum of QtyRequired=2466,
##       Sum of TotalArea=139.59}  => {Sum of Amount=185404.1}  0.02222222
1 0.02222222 45.000000     1
## [2]  {Sum of QtyRequired=2466,
##       Sum of Amount=185404.1}   => {Sum of TotalArea=139.59} 0.02222222
1 0.02222222 45.000000     1
## [3]  {Sum of TotalArea=139.59,
##       Sum of Amount=185404.1}   => {Sum of QtyRequired=2466} 0.02222222
1 0.02222222 45.000000     1
## [4]  {Sum of QtyRequired=2466,
##       Sum of TotalArea=139.59}  => {DURRY=1021}              0.02222222
1 0.02222222 45.000000     1
## [5]  {Sum of QtyRequired=2466,
##       DURRY=1021}               => {Sum of TotalArea=139.59} 0.02222222
1 0.02222222 45.000000     1
## [6]  {Sum of TotalArea=139.59,
##       DURRY=1021}               => {Sum of QtyRequired=2466} 0.02222222
1 0.02222222 45.000000     1
## [7]  {Sum of QtyRequired=2466,
##       Sum of TotalArea=139.59}  => {HANDLOOM=1445}           0.02222222
1 0.02222222 45.000000     1
## [8]  {Sum of QtyRequired=2466,
##       HANDLOOM=1445}            => {Sum of TotalArea=139.59} 0.02222222
1 0.02222222 45.000000     1
## [9]  {Sum of TotalArea=139.59,
##       HANDLOOM=1445}            => {Sum of QtyRequired=2466} 0.02222222
1 0.02222222 45.000000     1
## [10] {Sum of QtyRequired=2466,
##       Sum of TotalArea=139.59}  => {DOUBLE BACK=0}           0.02222222
1 0.02222222  1.730769     1
```

```
plot(rules,jitter=0)
```


Scatter plot for 15301 rules
1.00
0.75
0.50
0.25
confidence
0.1
0.2
0.3
0.4
support
lift
40
30
20
10