QUESTION 1

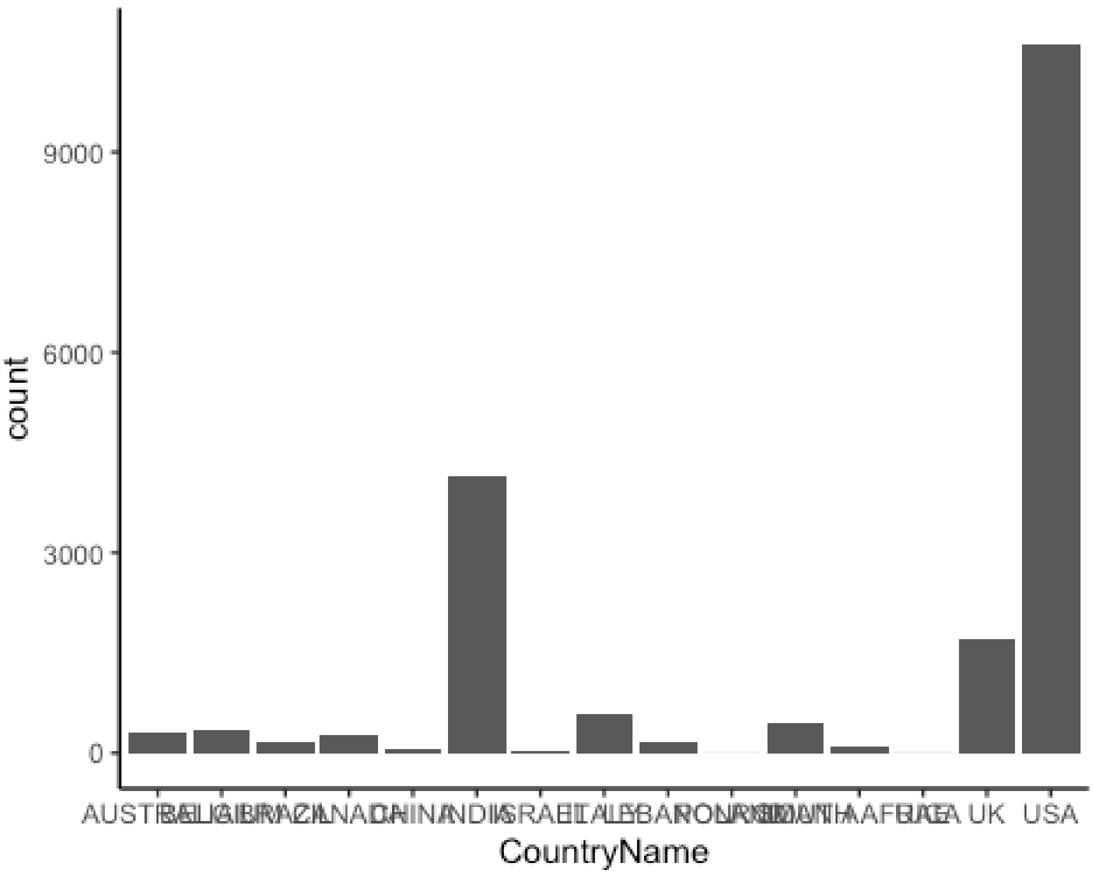
Cleaning Raw data set

```
#Defining the vector to contain all possible values which need to be replaced
with NA.
na_strings <- c("NA", "N A", "N / A", "N/A", "N/ A", "Not Available", "NOt
available","-","", " ")

#Reading excel into a variable by mentioning the path of the excel file into
function read_excel & Setting missing values in variable to NA
raw_data<-read_excel("Champo Carpets.xlsx",sheet=2,na = na_strings)

#Removing Column "Customer Order date, as it does not qualify for
Numeric/Factor
raw_data <- subset(raw_data,select = -
c(Custorderdate,CustomerOrderNo,TotalArea))

#Calculating mean of Amount.
Amount_mean <- mean(raw_data$Amount)

#Replacing 0 in numeric variables to the mean of the variable values.
raw_data <- raw_data %>%
            mutate(Amount = ifelse(Amount == 0,Amount_mean,Amount))

#str(raw_data)
#view(raw_data)
```

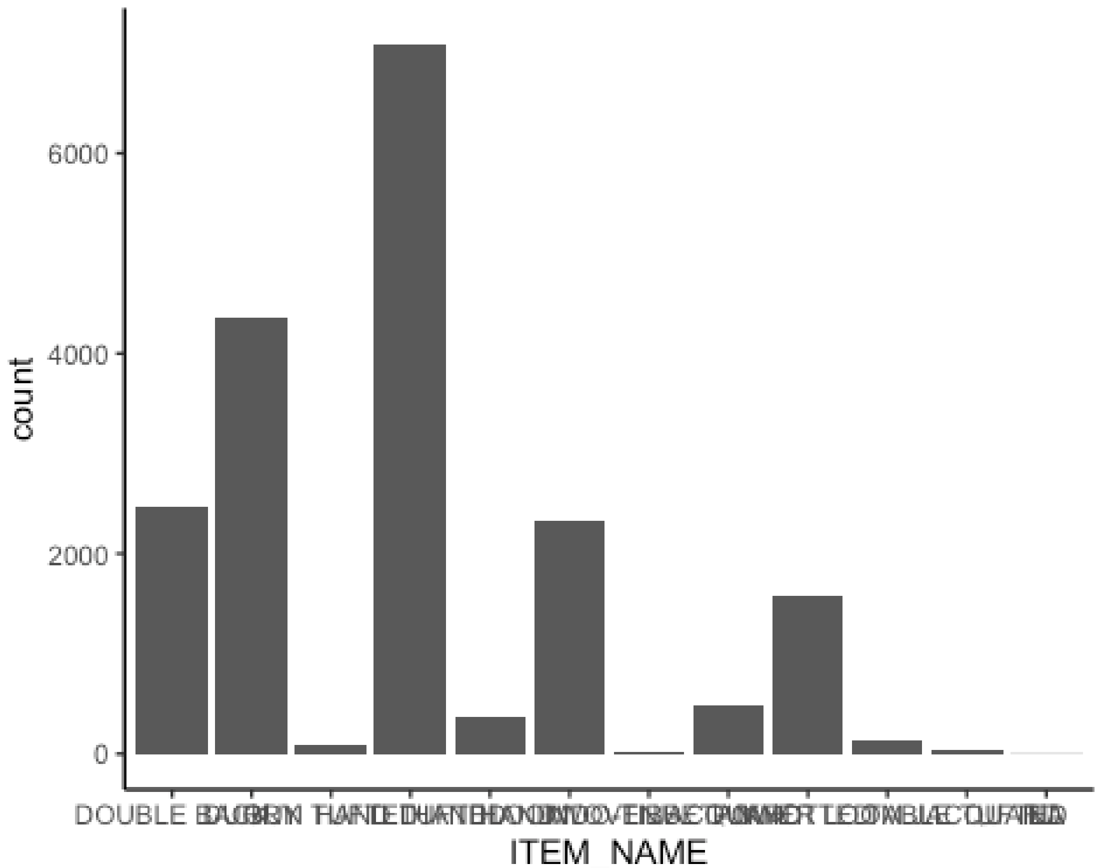
Exploratory Data Analysis on Raw Data set

Bar Graphs for important variables in Raw dataset

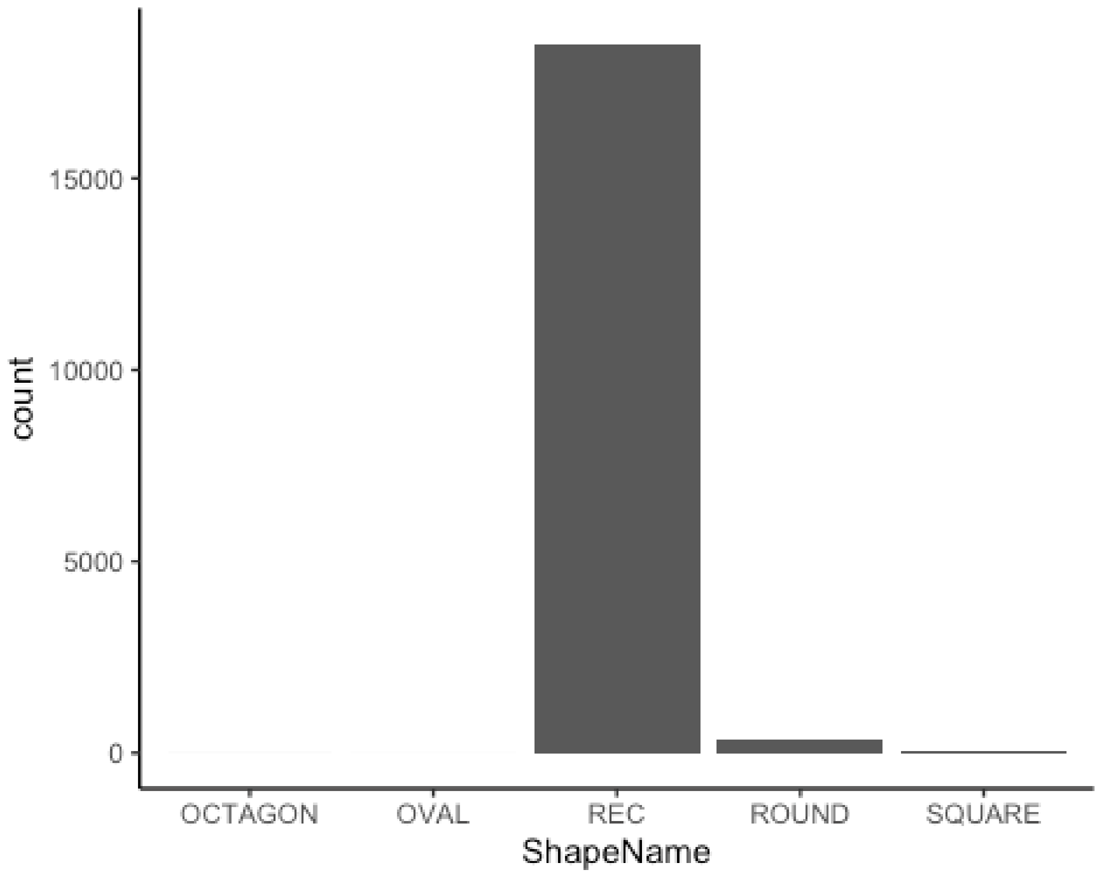
```
ggplot(raw_data)+
  geom_bar(aes(x=CountryName)) + theme_classic()
```




```
#From the graph, it is clear that USA, India and UK are the countries with
highest number of samples and orders.

ggplot(raw_data)+
  geom_bar(aes(x=ITEM_NAME)) + theme_classic()
```




```
#Hand Tufted, Durray, and Double Back are the most popular items.

ggplot(raw_data)+
  geom_bar(aes(x=ShapeName)) + theme_classic()
```




```
#Majority of the items were of Rectangular shape.
```

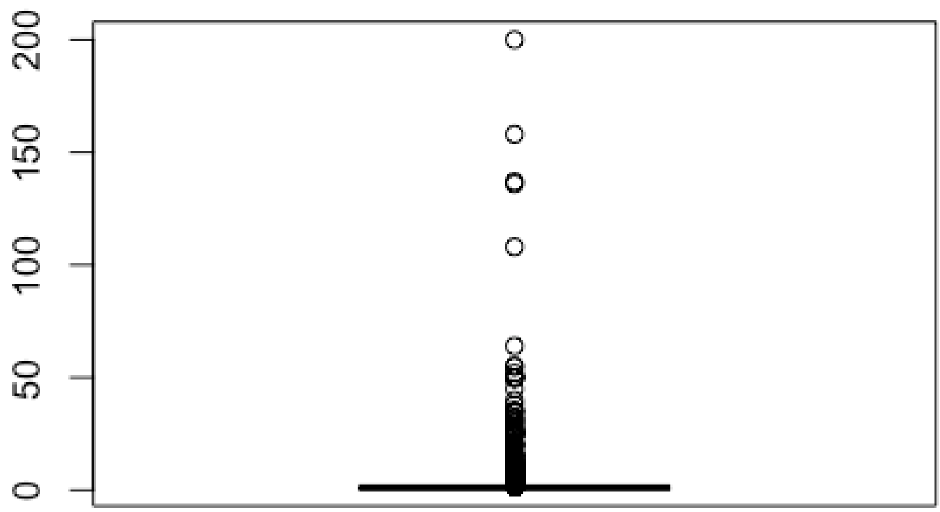
Cleaning Sample data set

```
#Reading excel into a variable by mentioning the path of the excel file into
function read_excel & creating NA
sample_data<-read_excel("Champo Carpets.xlsx",sheet=4)

#Removing the columns that are not required
sample_data <- subset(sample_data, select = -
c(USA,UK,Italy,Belgium,Romania,Australia,India,
                             `Hand Tufted`,Durry,`Double Back`,`Hand Woven`,

Knotted,Jacquard,Handloom,Other,REC,Round,Square))


#Converting required variables to categorical
sample_data$CustomerCode <- as.factor(sample_data$CustomerCode)
sample_data$CountryName <- as.factor(sample_data$CountryName)
sample_data$ITEM_NAME <- as.factor(sample_data$ITEM_NAME)
sample_data$ShapeName <- as.factor(sample_data$ShapeName)
```

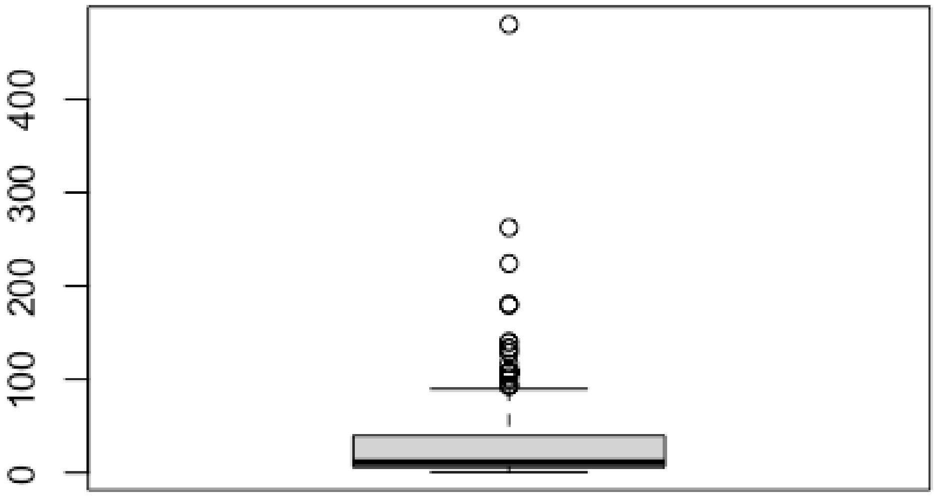
```
#str(sample_data)
#view(sample_data)
```

Exploratory Data Analysis on Sample Data Set

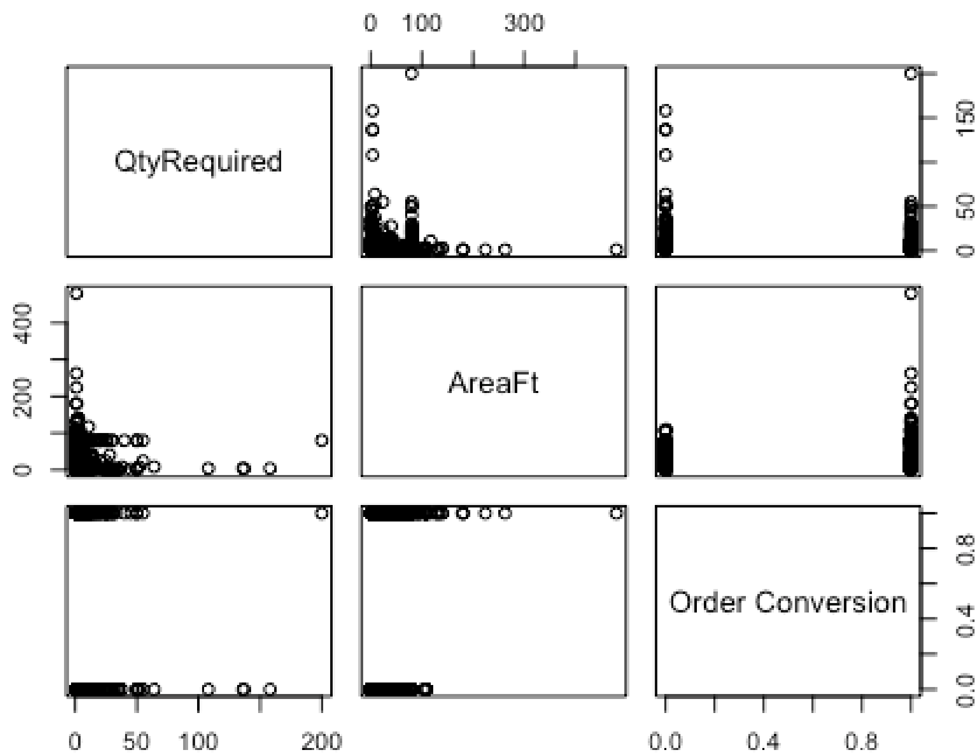
```
#Checking if there are any NA values
sum(is.na(sample_data))

## [1] 0

#Finding the outliers in numerical variables
boxplot(sample_data$QtyRequired)
```




```
#boxplot(sample_data$QtyRequired)$out

boxplot(sample_data$AreaFt)
```




```
#boxplot(sample_data$AreaFt)$out


#Finding correlation between numerical variables and target variable
cors <- cor(sample_data[,(colnames(sample_data) %in% c('QtyRequired',
'AreaFt'))],
            sample_data[,colnames(sample_data) %in% c('Order Conversion')])
cors

##              Order Conversion
## QtyRequired        0.0567018
## AreaFt             0.3357027

plot(sample_data[,(colnames(sample_data) %in% c('QtyRequired',
'AreaFt','Order Conversion'))])
```

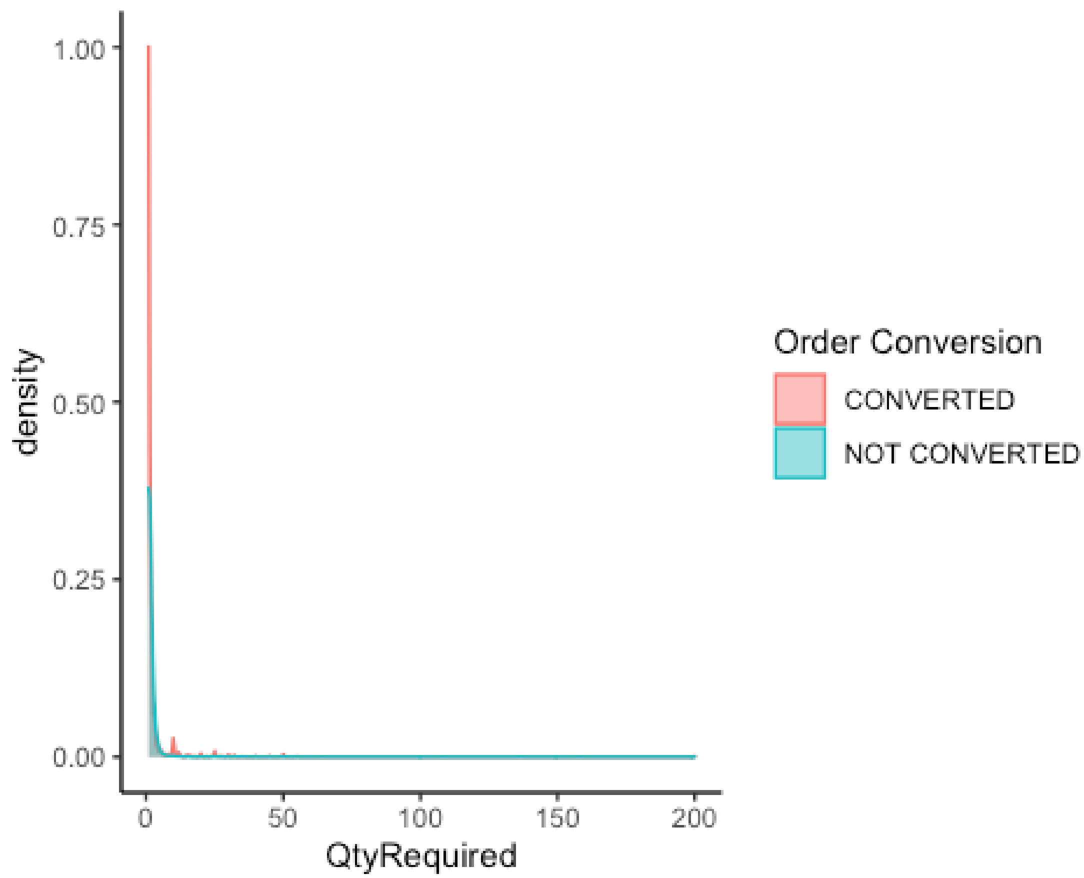



```
#Based on the correlation values, QtyRequired seems to be an important
numerical variable.

Converted_NotConverted <- sample_data$`Order Conversion`<-
as.factor(ifelse(sample_data$`Order Conversion` == 1,"CONVERTED","NOT
CONVERTED"))


#Calculate chi-square values for categorical variables (Descending order of
X-Squared, Higher = More important)

chisq.test(sample_data$CustomerCode, sample_data$`Order Conversion`,
correct=FALSE)

## Warning in chisq.test(sample_data$CustomerCode, sample_data$`Order
## Conversion`, : Chi-squared approximation may be incorrect

##
##  Pearson's Chi-squared test
##
## data:  sample_data$CustomerCode and sample_data$`Order Conversion`
## X-squared = 934.19, df = 33, p-value < 2.2e-16
```

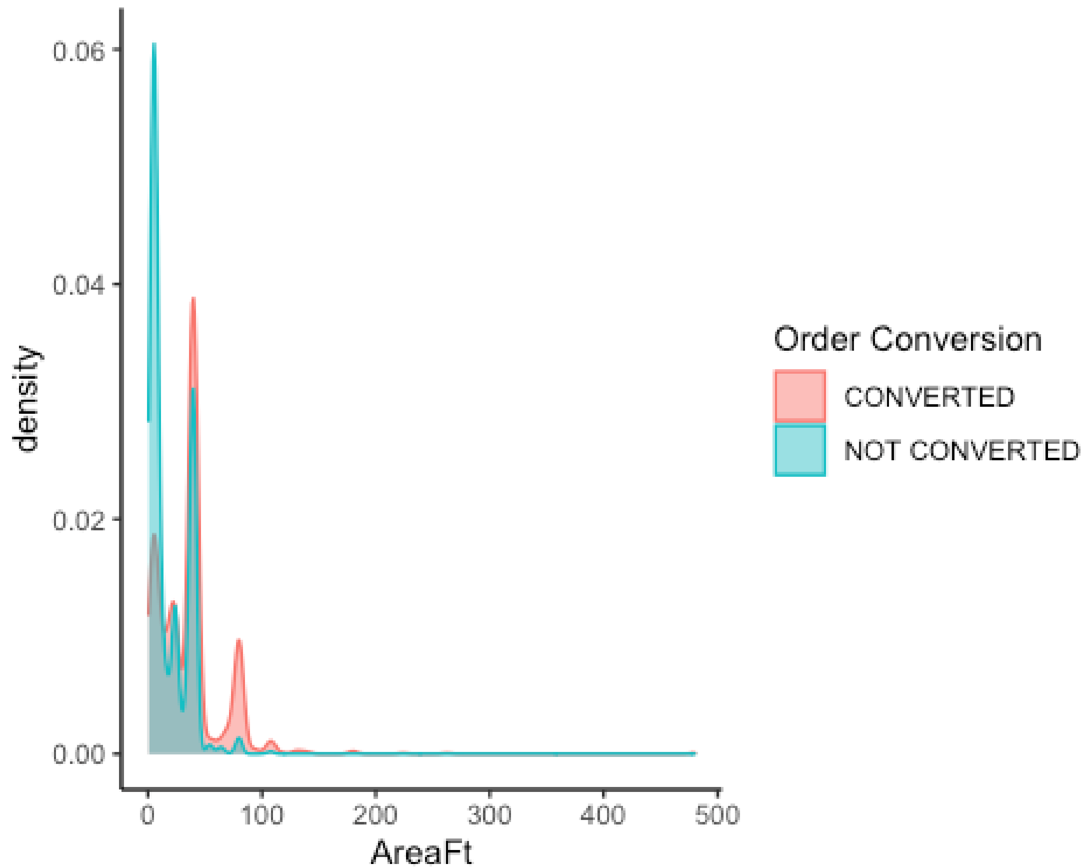
```r
chisq.test(sample_data$CountryName, sample_data$`Order Conversion`,
correct=FALSE)

## Warning in chisq.test(sample_data$CountryName, sample_data$`Order
Conversion`, :
## Chi-squared approximation may be incorrect

##
##  Pearson's Chi-squared test
##
## data:  sample_data$CountryName and sample_data$`Order Conversion`
## X-squared = 671.46, df = 13, p-value < 2.2e-16

chisq.test(sample_data$ITEM_NAME, sample_data$`Order Conversion`,
correct=FALSE)

## Warning in chisq.test(sample_data$ITEM_NAME, sample_data$`Order
Conversion`, :
## Chi-squared approximation may be incorrect

##
##  Pearson's Chi-squared test
##
## data:  sample_data$ITEM_NAME and sample_data$`Order Conversion`
## X-squared = 679.04, df = 10, p-value < 2.2e-16

chisq.test(sample_data$ShapeName, sample_data$`Order Conversion`,
correct=FALSE)

## Warning in chisq.test(sample_data$ShapeName, sample_data$`Order
Conversion`, :
## Chi-squared approximation may be incorrect

##
##  Pearson's Chi-squared test
##
## data:  sample_data$ShapeName and sample_data$`Order Conversion`
## X-squared = 9.4222, df = 2, p-value = 0.008995

#Based on the chi-square values, below are the important categorical
variables:
#CustomerCode
#CountryName
#ITEM_NAME



#Proportion of converted to not converted cases.

tbl <- table(Converted_NotConverted)
tbl_pct <- cbind(tbl,round(prop.table(tbl)*100,2))
colnames(tbl_pct) <- c('Count','Percentage')
knitr::kable(tbl_pct, format = "markdown")
```

	Count	Percentage
CONVERTED	1169	20.09
NOT CONVERTED	4651	79.91

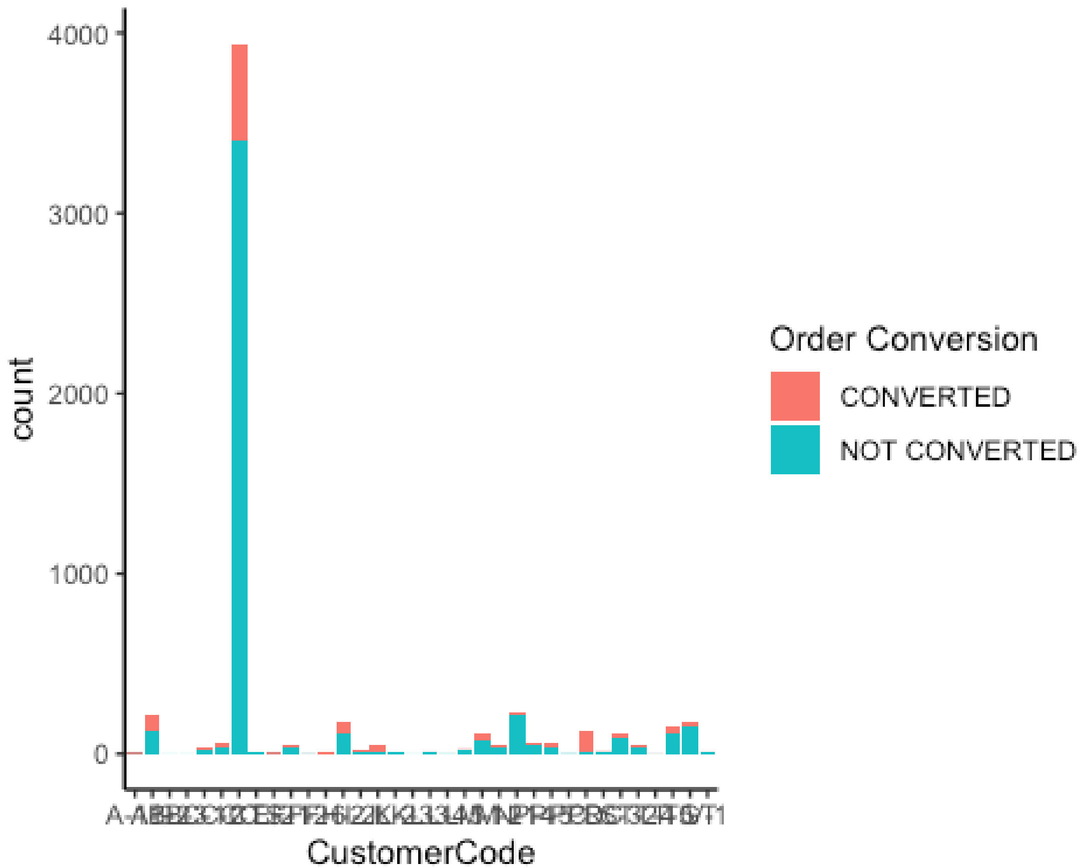
```
#Density Graphs - Finding correlation between numerical variables and target variable.

ggplot(sample_data)+
  geom_density(aes(x=QtyRequired,color=`Order Conversion`,fill=`Order Conversion`),alpha=0.5) + theme_classic()
```



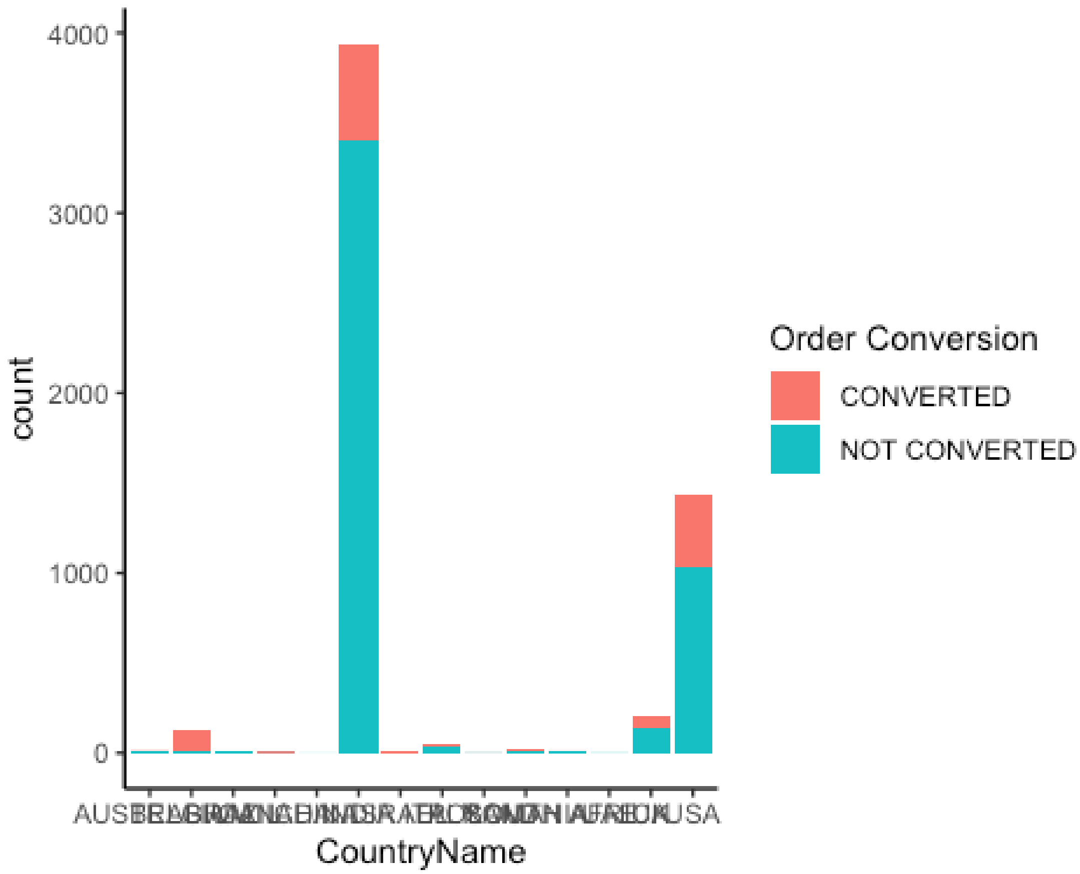

```
#The plot shows higher density of Converted and Not converted cases when the QtyRequired is around 10. Then it's a flat line showing as QtyRequired increases, we do not have cases for Converted and Not Converted.

ggplot(sample_data)+
  geom_density(aes(x=AreaFt,color=`Order Conversion`,fill=`Order Conversion`),alpha=0.5) + theme_classic()
```



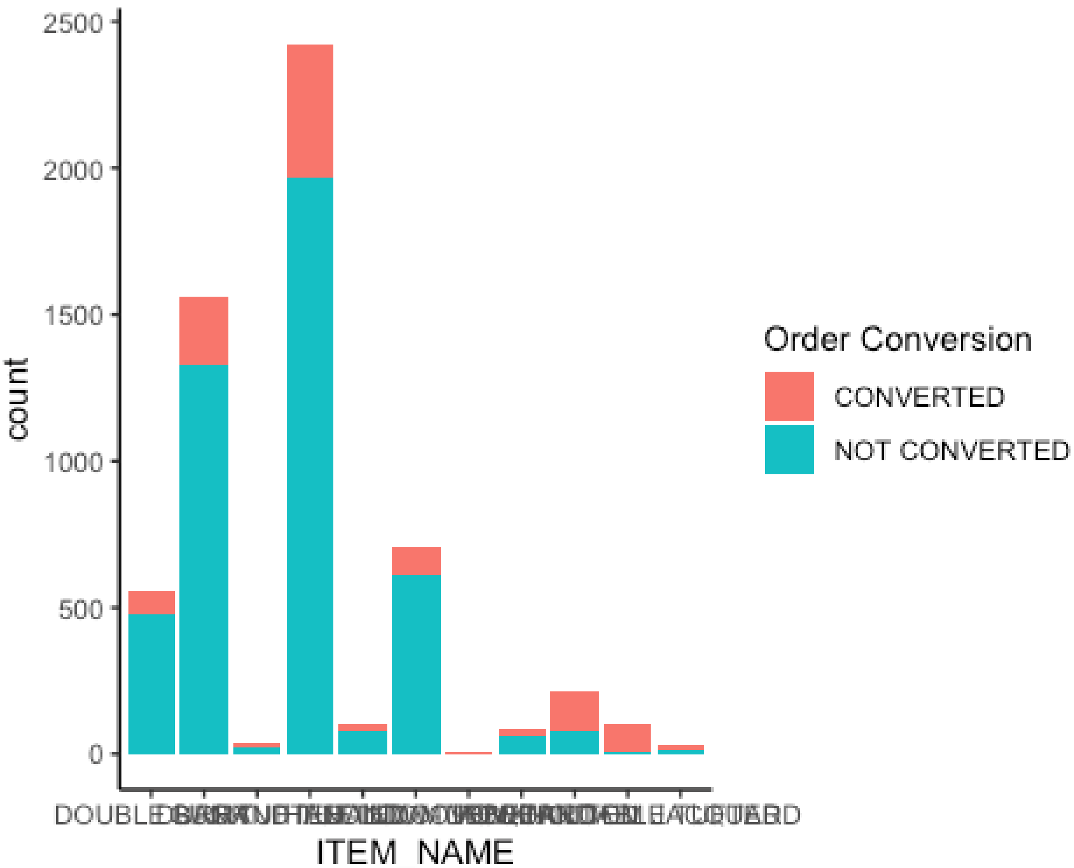

```
#The plot shows higher density of Not converted cases when the AreaFt is
between 0 to 50. Between 50 to 100 AreaFt we see a moderate density for
Converted cases. Beyond AreaFt of 100, we have very less cases of Converted
and Not Converted.


#Bar Graphs - Finding correlation between categorical variables and target
variable.

ggplot(sample_data)+
  geom_bar(aes(x=CustomerCode,fill=`Order Conversion`)) + theme_classic()
```



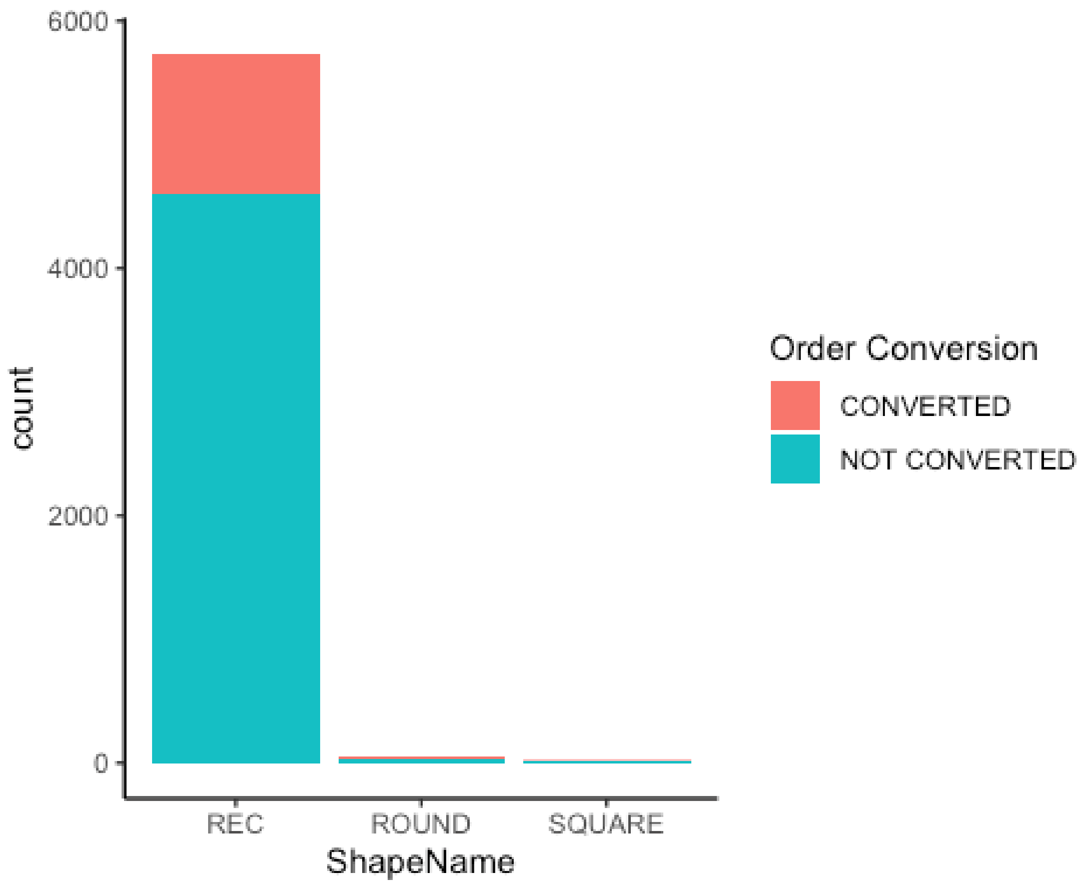

```
#For customer code CC we have the highest number of Converted and Not
Converted cases. For the rest of the Customer code values, there arent many
cases.

ggplot(sample_data)+
  geom_bar(aes(x=CountryName,fill=`Order Conversion`)) + theme_classic()
```




```
#We can see that India and US are the countries to which highest number of
samples were sent. Most of them were not converted. Most of the samples that
were sent to Belgium were converted.

ggplot(sample_data)+
  geom_bar(aes(x=ITEM_NAME,fill=`Order Conversion`)) + theme_classic()
```




```
#Majority of the sample items that were sent: Double Back, Durray, Hand
Tufted and Handwoven. Most of them were not converted.

ggplot(sample_data)+
  geom_bar(aes(x=ShapeName,fill=`Order Conversion`)) + theme_classic()
```




```
#As seen from the graph, majority of the samples that were sent were of
Rectangle shape.
```

QUESTION 2

Champo Carpets' business problems with appropriate solutions:

In many instances, the samples sent to the customers were not getting converted into orders. Customers sent different samples with similar designs. These samples costed the Champo Carpets a lot (low conversion rate)

1: To avoid creation of costly sample designs instead, creating appropriate samples which could generate maximum revenue for the organization.

In order to produce appropriate samples, we can implement a classification learning algorithm on the sample data which did/did not lead to a sale. This can be achieved through: - decision trees (with appropriate splitting, pruning parameters), further, as this problem would focus more on avoiding false positives (i.e, leads falsely marked as positive) as this would lead to high cost of sample production, therefore, precision of the model should be primary focus.

For example: Decision rules to find the features contributing toward conversion or non-conversion.

- regression models can help us analyse the important factors that contribute towards the sale of products, thus leading to focussed efforts by Champo Carpets to generate higher revenue.

2: Developing models such as clustering to identify customer preferences and recommendations systems.

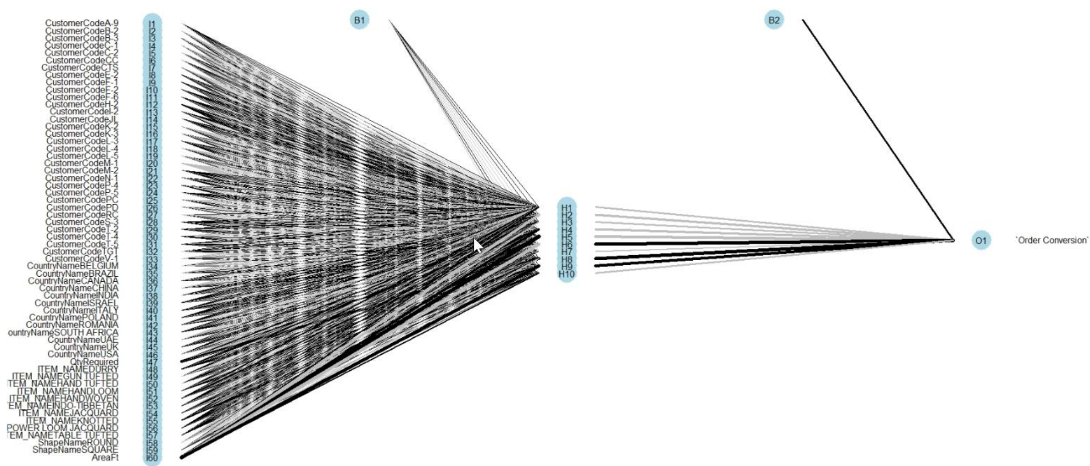
In order to solve the above problem, customer clustering could be designed based on past purchase patterns and cross recommendation of products within the same cluster could be performed. This will increase the likelihood of the customer of the same cluster with other similar customers to purchase the product. Also, find the correlated customers for product suggestions.

QUESTION 3

Building Models on Unbalanced Sample Data Set

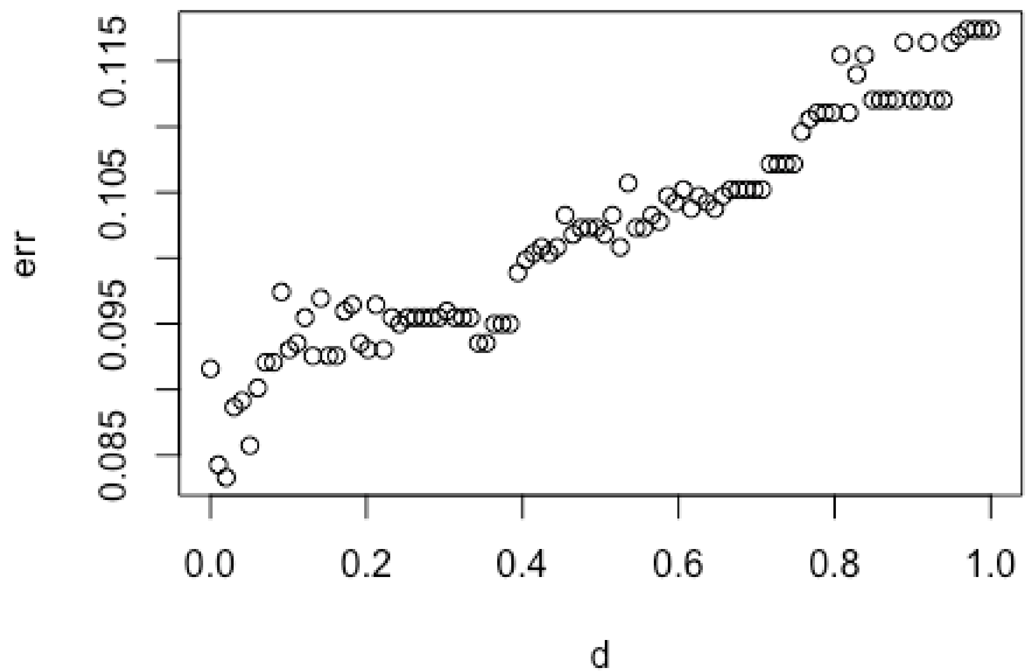
Neural Network on unbalanced Sample data set

```
#Normalizing all numerical variables
myscale <- function (x)
{
  (x - min(x))/ (max(x) - min(x))
}

neural_data <- sample_data %>% mutate_if(is.numeric, myscale)

#summary(neural_data)
#str(neural_data)
#view(neural_data)

#Partitioning data into train and test sets
set.seed(1234)
ind <- sample(2, nrow(neural_data), replace = T, prob = c(0.7,0.3))
train <- neural_data[ind == 1, ]
test <- neural_data[ind == 2, ]

knitr::include_graphics("NeuralPlot.jpeg")
```




```r
#nnModel$wts
#nnModel$fitted.values


#The darker lines shows that the weights corresponding to these nodes are
higher. i.e these  variables are more important.
#Based on the weights below are the important variables:
#AreaFt(20.99)
#QtyRequired (15.03)
#Item_NameHandTufted (7.07)
#CustomerCodeM-1(6.02)


#On test data
nn.preds <- predict(nnModel, test)
#nn.preds
nn.preds.class <- as.factor(predict(nnModel, test, type = "class"))
#nn.preds.class

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


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

##
## nn.preds.class  CONVERTED NOT CONVERTED
##   CONVERTED           218            34
##   NOT CONVERTED       109          1353

#Checking performance of neural network
error_metric = function(CM)
```

```
{
  TP =CM[1,1]
  TN =CM[2,2]
  FN =CM[1,2]
  FP =CM[2,1]
  recall = (TP) / (TP+FN)
  precision = (TP)/(TP+FP) #calculating precision of test data
  falsePositiveRate = (FP) / (FP+TN)
  falseNegativeRate = (FN) / (FN+TP)
  error =(FP+FN)/(TP+TN+FP+FN)
  modelPerf <- list("precision" = precision,
                    "recall" = recall,
                    "falsepositiverate" = falsePositiveRate,
                    "falsenegativerate" = falseNegativeRate,
                    "error" = error)
  return(modelPerf)
}

outPutlist <- error_metric(CM)

df <- ldply(outPutlist, data.frame)
setNames(df, c("", "Values"))

##                        Values
## 1         precision 0.66666667
## 2            recall 0.86507937
## 3 falsepositiverate 0.07455540
## 4 falsenegativerate 0.13492063
## 5             error 0.08343057

#Both precision and recall are not that good. Error rate is less. Our model
is not performing that well on unbalanced test data.
#precision = 66.66%
#recall = 86.1%
#Error rate = 8.4%


#Creating validation data set
set.seed(156)
indx <- sample(2, nrow(train), replace = T, prob = c(0.5, 0.5))
train2 <- train[indx == 1, ]
validation <- train[indx == 2, ]

plot(d, err)
```




```
#Choosing the d value for which the error is minimum. That is the best decay
value.
#From the graph it is 0.01
```

Decision trees construction on unbalanced Sample data set

```
#str(sample_data)
set.seed(96)

index <- sample(2, replace = T, nrow(sample_data), prob = c(0.7,0.3))

train <- sample_data[index== 1,]

test <- sample_data[index== 2,]

MyFormula = `Order Conversion` ~.

mytree_70_30_basic <- rpart(MyFormula, data=train)

#summary(mytree_70_30_basic)
#print(mytree_70_30_basic)
```

```r
rpart.plot(mytree_70_30_basic)

#From the decision tree, below are the important variables:
#CustomerCode
#ITEM_NAME
#AreaFt

#Predict function to predict the classes for the decision tree
mytree_70_30_basic for training data.
mytree_train_predict_70_30 <- predict(mytree_70_30_basic, data = train , type
= "class")

#Calculating the training error by comparing predicted classes with target
variable of original dataset.
mytree_train_error_70_30 <- mean(mytree_train_predict_70_30 != train$`Order
Conversion`)

mytree_train_error_70_30

## [1] 0.09249692

#Error on training data is 9.2%

table(train$`Order Conversion`)

##
##     CONVERTED NOT CONVERTED
##           818          3247

#Only 20% of the samples will be converted to orders. Majority of the samples
are not converted.

#Predict function to predict the classes for the decision tree mytree_70_30
for testing data.
mytree_test_predict_70_30 <- predict(mytree_70_30_basic, newdata = test, type
= "class")

#Calculating the testing error by comparing predicted classes with target
variable of original dataset.
mytree_test_error_70_30 <- mean(mytree_test_predict_70_30 != test$`Order
Conversion`)

mytree_test_error_70_30

## [1] 0.0957265

#Error on test data is 9.5%

#Calculating the performance of the model by finding the difference between
the test error & train data.
```

```
diff_70_30 = mytree_test_error_70_30 - mytree_train_error_70_30

diff_70_30

## [1] 0.003229571

#difference between training and test error is 0.3%


#APPLYING PARAMETER VALUES TO ARRIVE AT BETTER PERFORMANCE FOR 70-30 MODEL
#Creating vectors for minsplit and minbucket values to be used for different
combinations to test performance CP: 0.01 with least xerror of 0.4951100

msplt <- c(12,48,102)
mbckt <- c(4,16,34)

for (i in msplt)
{
  for (j in mbckt)
  {
    #Using rpart function to construct the decision tree based on training
data,split on gini.
    mytree_70_30 <- rpart(MyFormula, data = train, parms = list(split="gini")
,control = rpart.control (minsplit = i,minbucket = j,cp=0.01))

    #Predict function to predict the classes for the decision tree
mytree_70_30 for training data.
    mytree_train_predict_70_30 <- predict(mytree_70_30, data = train , type =
"class")


    #Calculating the training error by comparing predicted classes with
target variable of original dataset.
    mytree_train_error_70_30 <- mean(mytree_train_predict_70_30 !=
train$`Order Conversion`)

    #Predict function to predict the classes for the decision tree
mytree_70_30 for testing data.
    mytree_test_predict_70_30 <- predict(mytree_70_30, newdata = test, type =
"class")

    #Calculating the testing error by comparing predicted classes with target
variable of original dataset.
    mytree_test_error_70_30 <- mean(mytree_test_predict_70_30 != test$`Order
Conversion`)

    #Calculating the performance of the model by finding the difference
between the test error & train data.
    diff_70_30 = mytree_test_error_70_30 - mytree_train_error_70_30
```

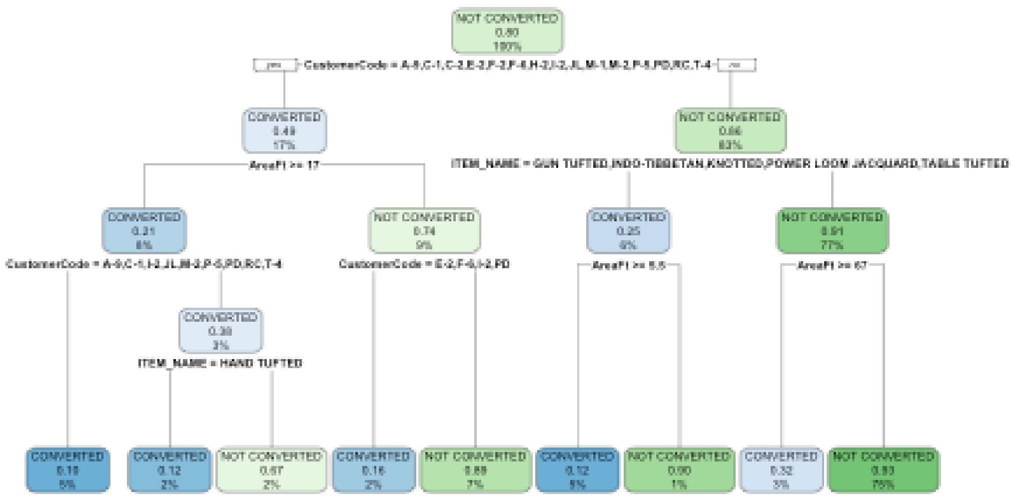
```
    #Confusion Matrix for 70:30 Split
    #As the sample costs are high, the precision should be high as samples
should only be sent for actual conversion not false positive

    cfmt <- table(mytree_train_predict_70_30,train$`Order Conversion`)
    print (cfmt)
    fp = cfmt[2,1]
    fn = cfmt[1,2]
    tn = cfmt[2,2]
    tp = cfmt[1,1]

    #Calculating precision by dividing true positive with the sum of true
positive and false positive.
    precision_train =  (tp)/(tp+fp)
    accuracymodel_train = (tp+tn)/(tp+tn+fp+fn)
    recall_train = (tp)/(tp+fn)
    fscore_train =
(2*(recall_train*precision_train))/(recall_train+precision_train)


    #Confusion matrix on test data
    cfmt <- table(mytree_test_predict_70_30,test$`Order Conversion`)
    print(cfmt)

    #Calculating precision by dividing true positive with the sum of true
positive and false positive.
    precision_test =  (tp)/(tp+fp)
    accuracymodel_test = (tp+tn)/(tp+tn+fp+fn)
    recall_test = (tp)/(tp+fn)
    fscore_test = (2*(recall_test*precision_test))/(recall_test +
precision_test)

    #Printing the values for train data error, test data error, performance
and other parameters.
    print(paste("Train data error: ", mytree_train_error_70_30))
    print(paste("Test data error: ", mytree_test_error_70_30))
    print(paste("Difference/performance", diff_70_30))
    print(paste("precision of training data: ", precision_train))
    print(paste("accuracy of training data: ", accuracymodel_train))
    print(paste("recall of training data: ", recall_train))
    print(paste("F-score of training data: ", fscore_train))
    print(paste("precision of test data: ", precision_test))
    print(paste("accuracy of test data: ", accuracymodel_test))
    print(paste("recall of test data: ", recall_test))
    print(paste("F-score of test data: ", fscore_test))
  }
}
```

```
##
## mytree_train_predict_70_30 CONVERTED NOT CONVERTED
##              CONVERTED           538            96
##              NOT CONVERTED       280          3151
##
## mytree_test_predict_70_30 CONVERTED NOT CONVERTED
##             CONVERTED           222            39
##             NOT CONVERTED       129          1365
## [1] "Train data error:  0.0924969249692497"
## [1] "Test data error:  0.0957264957264957"
## [1] "Difference/performance 0.00322957075724604"
## [1] "precision of training data:  0.657701711491443"
## [1] "accuracy of training data:  0.90750307503075"
## [1] "recall of training data:  0.848580441640379"
## [1] "F-score of training data:  0.741046831955923"
## [1] "precision of test data:  0.657701711491443"
## [1] "accuracy of test data:  0.90750307503075"
## [1] "recall of test data:  0.848580441640379"
## [1] "F-score of test data:  0.741046831955923"
##
## mytree_train_predict_70_30 CONVERTED NOT CONVERTED
##              CONVERTED           538            96
##              NOT CONVERTED       280          3151
##
## mytree_test_predict_70_30 CONVERTED NOT CONVERTED
##             CONVERTED           222            39
##             NOT CONVERTED       129          1365
## [1] "Train data error:  0.0924969249692497"
## [1] "Test data error:  0.0957264957264957"
## [1] "Difference/performance 0.00322957075724604"
## [1] "precision of training data:  0.657701711491443"
## [1] "accuracy of training data:  0.90750307503075"
## [1] "recall of training data:  0.848580441640379"
## [1] "F-score of training data:  0.741046831955923"
## [1] "precision of test data:  0.657701711491443"
## [1] "accuracy of test data:  0.90750307503075"
## [1] "recall of test data:  0.848580441640379"
## [1] "F-score of test data:  0.741046831955923"
##
## mytree_train_predict_70_30 CONVERTED NOT CONVERTED
##              CONVERTED           538            96
##              NOT CONVERTED       280          3151
##
## mytree_test_predict_70_30 CONVERTED NOT CONVERTED
##             CONVERTED           222            39
##             NOT CONVERTED       129          1365
## [1] "Train data error:  0.0924969249692497"
## [1] "Test data error:  0.0957264957264957"
## [1] "Difference/performance 0.00322957075724604"
## [1] "precision of training data:  0.657701711491443"
```

```
## [1] "accuracy of training data:  0.90750307503075"
## [1] "recall of training data:  0.848580441640379"
## [1] "F-score of training data:  0.741046831955923"
## [1] "precision of test data:  0.657701711491443"
## [1] "accuracy of test data:  0.90750307503075"
## [1] "recall of test data:  0.848580441640379"
## [1] "F-score of test data:  0.741046831955923"
##
## mytree_train_predict_70_30 CONVERTED NOT CONVERTED
##              CONVERTED           538            96
##              NOT CONVERTED       280          3151
##
## mytree_test_predict_70_30 CONVERTED NOT CONVERTED
##             CONVERTED           222            39
##             NOT CONVERTED       129          1365
## [1] "Train data error:  0.0924969249692497"
## [1] "Test data error:  0.0957264957264957"
## [1] "Difference/performance 0.00322957075724604"
## [1] "precision of training data:  0.657701711491443"
## [1] "accuracy of training data:  0.90750307503075"
## [1] "recall of training data:  0.848580441640379"
## [1] "F-score of training data:  0.741046831955923"
## [1] "precision of test data:  0.657701711491443"
## [1] "accuracy of test data:  0.90750307503075"
## [1] "recall of test data:  0.848580441640379"
## [1] "F-score of test data:  0.741046831955923"
##
## mytree_train_predict_70_30 CONVERTED NOT CONVERTED
##              CONVERTED           538            96
##              NOT CONVERTED       280          3151
##
## mytree_test_predict_70_30 CONVERTED NOT CONVERTED
##             CONVERTED           222            39
##             NOT CONVERTED       129          1365
## [1] "Train data error:  0.0924969249692497"
## [1] "Test data error:  0.0957264957264957"
## [1] "Difference/performance 0.00322957075724604"
## [1] "precision of training data:  0.657701711491443"
## [1] "accuracy of training data:  0.90750307503075"
## [1] "recall of training data:  0.848580441640379"
## [1] "F-score of training data:  0.741046831955923"
## [1] "precision of test data:  0.657701711491443"
## [1] "accuracy of test data:  0.90750307503075"
## [1] "recall of test data:  0.848580441640379"
## [1] "F-score of test data:  0.741046831955923"
##
## mytree_train_predict_70_30 CONVERTED NOT CONVERTED
##              CONVERTED           538            96
##              NOT CONVERTED       280          3151
##
```

```
## mytree_test_predict_70_30 CONVERTED NOT CONVERTED
##             CONVERTED           222            39
##             NOT CONVERTED       129          1365
## [1] "Train data error:  0.0924969249692497"
## [1] "Test data error:  0.0957264957264957"
## [1] "Difference/performance 0.00322957075724604"
## [1] "precision of training data:  0.657701711491443"
## [1] "accuracy of training data:  0.90750307503075"
## [1] "recall of training data:  0.848580441640379"
## [1] "F-score of training data:  0.741046831955923"
## [1] "precision of test data:  0.657701711491443"
## [1] "accuracy of test data:  0.90750307503075"
## [1] "recall of test data:  0.848580441640379"
## [1] "F-score of test data:  0.741046831955923"
##
## mytree_train_predict_70_30 CONVERTED NOT CONVERTED
##              CONVERTED           538            96
##              NOT CONVERTED       280          3151
##
## mytree_test_predict_70_30 CONVERTED NOT CONVERTED
##             CONVERTED           222            39
##             NOT CONVERTED       129          1365
## [1] "Train data error:  0.0924969249692497"
## [1] "Test data error:  0.0957264957264957"
## [1] "Difference/performance 0.00322957075724604"
## [1] "precision of training data:  0.657701711491443"
## [1] "accuracy of training data:  0.90750307503075"
## [1] "recall of training data:  0.848580441640379"
## [1] "F-score of training data:  0.741046831955923"
## [1] "precision of test data:  0.657701711491443"
## [1] "accuracy of test data:  0.90750307503075"
## [1] "recall of test data:  0.848580441640379"
## [1] "F-score of test data:  0.741046831955923"
##
## mytree_train_predict_70_30 CONVERTED NOT CONVERTED
##              CONVERTED           538            96
##              NOT CONVERTED       280          3151
##
## mytree_test_predict_70_30 CONVERTED NOT CONVERTED
##             CONVERTED           222            39
##             NOT CONVERTED       129          1365
## [1] "Train data error:  0.0924969249692497"
## [1] "Test data error:  0.0957264957264957"
## [1] "Difference/performance 0.00322957075724604"
## [1] "precision of training data:  0.657701711491443"
## [1] "accuracy of training data:  0.90750307503075"
## [1] "recall of training data:  0.848580441640379"
## [1] "F-score of training data:  0.741046831955923"
## [1] "precision of test data:  0.657701711491443"
## [1] "accuracy of test data:  0.90750307503075"
```

```
## [1] "recall of test data:  0.848580441640379"
## [1] "F-score of test data:  0.741046831955923"
##
## mytree_train_predict_70_30 CONVERTED NOT CONVERTED
##              CONVERTED           538            96
##              NOT CONVERTED       280          3151
##
## mytree_test_predict_70_30 CONVERTED NOT CONVERTED
##             CONVERTED           222            39
##             NOT CONVERTED       129          1365
## [1] "Train data error:  0.0924969249692497"
## [1] "Test data error:  0.0957264957264957"
## [1] "Difference/performance 0.00322957075724604"
## [1] "precision of training data:  0.657701711491443"
## [1] "accuracy of training data:  0.90750307503075"
## [1] "recall of training data:  0.848580441640379"
## [1] "F-score of training data:  0.741046831955923"
## [1] "precision of test data:  0.657701711491443"
## [1] "accuracy of test data:  0.90750307503075"
## [1] "recall of test data:  0.848580441640379"
## [1] "F-score of test data:  0.741046831955923"

#Performance has been the same irrespective of different values of msplt and mbckt
#Train error - 9.20%
#Test error - 9.50%
#Difference between Train and Test error - 0.30%
#Precision of Training Data - 65.70%
#Accuracy of Training Data - 90.70%
#Recall of Training Data - 84.80%
#F-Score of Training data - 74.10%
#Precision of Test Data - 65.70%
#Accuracy of Test Data - 90.70%
#Recall of Test Data - 84.80%
#F-Score of Test data - 74.10%


rpart.plot(mytree_70_30)
```



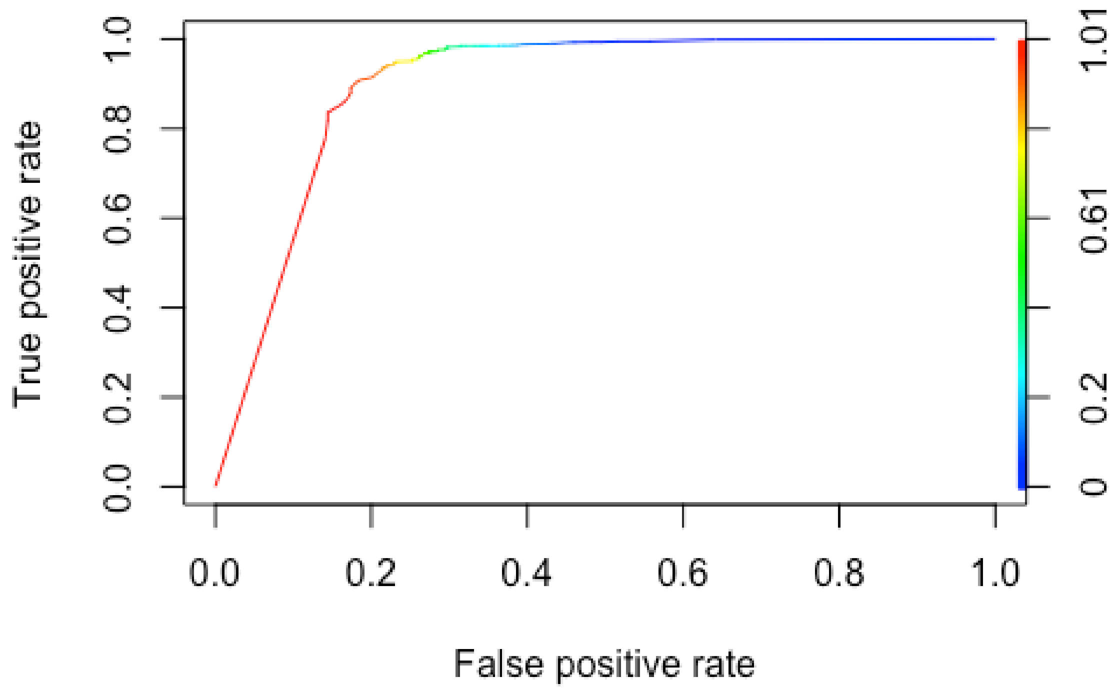

```
#Decision trees from the above plot:
#A strong decision rule from the above tree is:
#If CustomerCode is not equal to A-9, C-1, C-2, E-2, F-2, F-6, H-2, I-2, JL, M-1, M-2, P-5, PD, RC, T-4
#and ITEM_NAME is not equal to GUN TUFTED, INDO-TIBBETAN, KNOTTED, POWER LOOM JACQUARD, TABLE TUFTED
#and AreaFt<67 THEN NOT CONVERTED
```

Random Forest on unbalanced Sample data set

```
sample_data_rf <- sample_data
#str(sample_data_rf)


#Determining important variables
rf_imp <- randomForest(`Order Conversion` ~ ., data=sample_data_rf, mtry = sqrt(ncol(sample_data_rf)-1),
                       ntree = 100, proximity = T , importance = T)

importance(rf_imp, type = 2) #MeanDecreaseGini

##                MeanDecreaseGini
## CustomerCode        292.828871
```

```
## CountryName        163.890042
## QtyRequired         72.836117
## ITEM_NAME          326.393868
## ShapeName            9.675267
## AreaFt             408.919551

#Based on the MeanDecreaseGini values below are the variables in order of
importance:
#AreaFt
#ITEM_NAME
#CustomerCode
#CountryName


ind <- sample(2, nrow(sample_data_rf), replace = T, prob = c(0.7, 0.3))
Train <- sample_data_rf[ind == 1, ]
Validation <- sample_data_rf[ind == 2, ]
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 = Validation, type = "class")
  pr.err <- c(pr.err,mean(Validation$`Order Conversion` != predicted))
}


#Calculating Best mtry value.
bestmtry <- which.min(pr.err)
bestmtry  #3

## [1] 3

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: 3
##
##         OOB estimate of  error rate: 8.14%
## Confusion matrix:
##               CONVERTED NOT CONVERTED class.error
## CONVERTED           584           245  0.29553679
## NOT CONVERTED        89          3184  0.02719218
```

```r
#OOB estimate of  error rate: 8.16%


#Confusion Matrix
cfmt <- table(predicted,Validation$`Order Conversion`)
print(cfmt)

##
## predicted       CONVERTED NOT CONVERTED
##   CONVERTED           245            46
##   NOT CONVERTED        95          1332

fp = cfmt[2,1]
fn = cfmt[1,2]
tn = cfmt[2,2]
tp = cfmt[1,1]

#Calculating precision by dividing true positive with the sum of true
positive and false positive.
precision_test =  (tp)/(tp+fp)
accuracymodel_test = (tp+tn)/(tp+tn+fp+fn)
recall_test = (tp)/(tp+fn)
fscore_test = (2*(recall_test*precision_test))/(recall_test + precision_test)
print(paste("precision of test data: ", precision_test))

## [1] "precision of test data:  0.720588235294118"

print(paste("accuracy of test data: ", accuracymodel_test))

## [1] "accuracy of test data:  0.917927823050058"

print(paste("recall of test data: ", recall_test))

## [1] "recall of test data:  0.84192439862543"

print(paste("F-score of test data: ", fscore_test))

## [1] "F-score of test data:  0.776545166402536"

#Random Forest produces better recall of 84.1% on unbalanced data. Has
precision of 72.05%. F-Score is 77.6%

#Drawing evaluation chart - ROC Curve

predicteddtProb <- predict(rf1, newdata = Validation, type = "prob")[,2]
pred <- prediction(predicteddtProb, Validation$`Order Conversion`)
perf <- performance(pred, "tpr", "fpr")
plot(perf,colorize=TRUE)
```




```
#From the graph, the threshold value is 0.2
```

Logistic Regression on unbalanced Sample data set

```
logistic_data <- sample_data

set.seed(96)
#As we got NA values for CountryName, removing CustomerCode column
logistic_data <- subset(logistic_data,select = -c(CustomerCode))
ind <- sample(2, nrow(logistic_data), replace = T, prob = c(0.7, 0.3))
Train <- logistic_data[ind == 1, ]
Test <- logistic_data[ind == 2, ]

#Generalized linear Model
LogReg <- glm(`Order Conversion` ~., data = Train, family = "binomial")
summary(LogReg)

##
## Call:
## glm(formula = `Order Conversion` ~ ., family = "binomial", data = Train)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -3.0164   0.1985   0.2644   0.5428   3.0853
##
## Coefficients:
##                                Estimate Std. Error z value Pr(>|z|)
## (Intercept)                    3.267529   0.957990   3.411 0.000648 ***
## CountryNameBELGIUM            -5.410894   1.033173  -5.237 1.63e-07 ***
## CountryNameBRAZIL             11.783245 624.194539   0.019 0.984939
## CountryNameCANADA             -5.345177   1.423734  -3.754 0.000174 ***
## CountryNameINDIA               0.693522   0.937589   0.740 0.459490
## CountryNameISRAEL            -17.495537 509.578888  -0.034 0.972611
## CountryNameITALY               1.077176   1.164621   0.925 0.355010
## CountryNamePOLAND             -0.976773   1.707092  -0.572 0.567196
## CountryNameROMANIA            -3.138773   1.122463  -2.796 0.005169 **
## CountryNameSOUTH AFRICA       -0.226851   1.517621  -0.149 0.881177
## CountryNameUAE                13.733306 882.743882   0.016 0.987587
## CountryNameUK                 -1.054177   0.957136  -1.101 0.270728
## CountryNameUSA                -0.755750   0.939666  -0.804 0.421238
## QtyRequired                   -0.003943   0.007131  -0.553 0.580317
## ITEM_NAMEDURRY                -0.136126   0.222442  -0.612 0.540562
## ITEM_NAMEGUN TUFTED           -2.589782   0.490460  -5.280 1.29e-07 ***
## ITEM_NAMEHAND TUFTED           0.030406   0.203746   0.149 0.881369
## ITEM_NAMEHANDLOOM              0.170462   0.397894   0.428 0.668352
## ITEM_NAMEHANDWOVEN             0.754322   0.286305   2.635 0.008422 **
## ITEM_NAMEINDO-TIBBETAN       -17.340903 509.653061  -0.034 0.972857
## ITEM_NAMEJACQUARD              0.183197   0.470532   0.389 0.697024
## ITEM_NAMEKNOTTED              -2.976715   0.287446 -10.356  < 2e-16 ***
## ITEM_NAMEPOWER LOOM JACQUARD  -5.244358   0.452303 -11.595  < 2e-16 ***
## ITEM_NAMETABLE TUFTED         -2.707774   0.488827  -5.539 3.04e-08 ***
## ShapeNameROUND                -1.176272   0.426799  -2.756 0.005851 **
## ShapeNameSQUARE               -1.182599   0.658082  -1.797 0.072330 .
## AreaFt                        -0.057440   0.002836 -20.251  < 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: 4082.1  on 4064  degrees of freedom
## Residual deviance: 2589.5  on 4038  degrees of freedom
## AIC: 2643.5
##
## Number of Fisher Scoring iterations: 13

#Choosing important variables based on p-value. A p-value less than 0.05 is statistically significant.
#From our summary, below are the important variables:
#CountryNameBELGIUM
#CountryNameCANADA
#CountryNameROMANIA
#ITEM_NAMEGUN TUFTED
#ITEM_NAMEHANDWOVEN
```

```
#ITEM_NAMEKNOTTED
#ITEM_NAMEPOWER LOOM JACQUARD
#ITEM_NAMETABLE TUFTED
#ShapeNameROUND
#AreaFt


#Performance of the logistic regression model
#Residual variance has decreased when compared to Null deviance which shows
the quality of prediction has improved.
#Null deviance: 4090.4  on 4064  degrees of freedom
#Residual deviance: 2439.7  on 4004  degrees of freedom
```

Boosting on unbalanced Sample data set

Boosting on unbalanced data set takes a very long time to build the model. Hence, unable to print the results and commented the code.

```
#boosting_data <- sample_data
#set.seed(96)

#ind <- sample(2, nrow(boosting_data), replace = T, prob = c(0.7, 0.3))
#train <- boosting_data[ind == 1, ]
#test <- boosting_data[ind == 2, ]

#model = boosting(`Order Conversion` ~ ., data=train, boos=TRUE, mfinal=50)
#print(names(model))
#print(model$trees[1])

#pred = predict(model, test)
#print(pred$confusion)
#print(pred$error)

#result = data.frame(test$`Order Conversion`, pred$prob, pred$class)
#print(result)

#cross-validation method
#cvmodel = boosting.cv(`Order Conversion` ~ ., data=boosting_data, boos=TRUE,
mfinal=10, v=5)

#print(cvmodel[-1])
#print(data.frame(boosting_data$`Order Conversion`, cvmodel$class))
```

Balancing Sample Data Set

```
sample_data_balanced <- sample_data

#Renaming the 'Order Conversion' column to 'OrderConversion'
```

```r
colnames(sample_data_balanced)[which(names(sample_data_balanced) == "Order Conversion")] <- "OrderConversion"

#Summary of Data Before Balancing
summary(sample_data_balanced$OrderConversion)
```

```
##     CONVERTED NOT CONVERTED
##          1169          4651
```

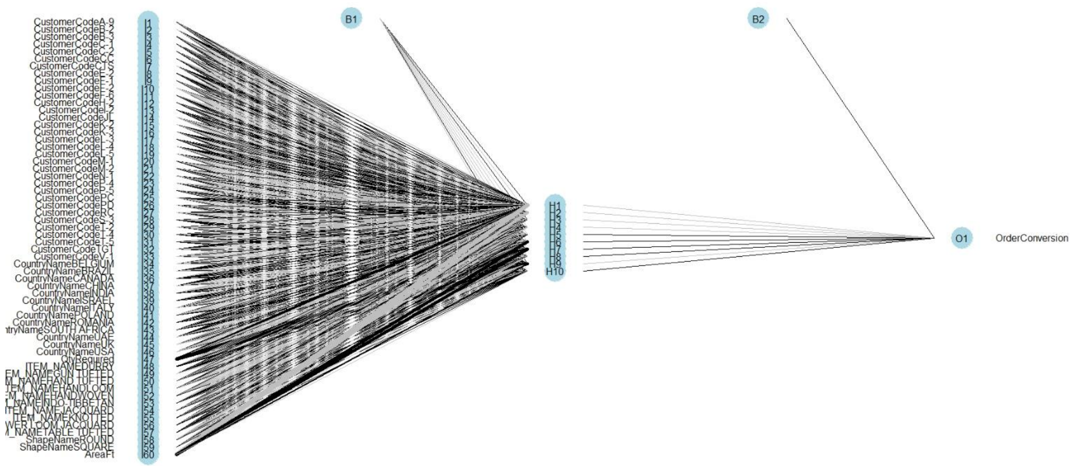
```r
#str(sample_data_balanced)

balanced.data <- ovun.sample(OrderConversion ~ ., data = sample_data_balanced, method = "over", N = 9305)$data

#Summary of Data After Balancing
summary(balanced.data$OrderConversion)
```

```
## NOT CONVERTED     CONVERTED
##          4651          4654
```

Building Models on Balanced Sample Data Set

Neural Network on Balanced Sample data set

```r
#Normalizing all numerical variables
myscale <- function (x)
{
  (x - min(x))/ (max(x) - min(x))
}

neural_data_balanced <- balanced.data %>% mutate_if(is.numeric, myscale)

#summary(neural_data_balanced)
#view(neural_data_balanced)

#Partitioning data into train and test sets
set.seed(1234)
ind <- sample(2, nrow(neural_data_balanced), replace = T, prob = c(0.7,0.3))
train <- neural_data_balanced[ind == 1, ]
test <- neural_data_balanced[ind == 2, ]

knitr::include_graphics("NeuralPlotBalanced.jpeg")
```

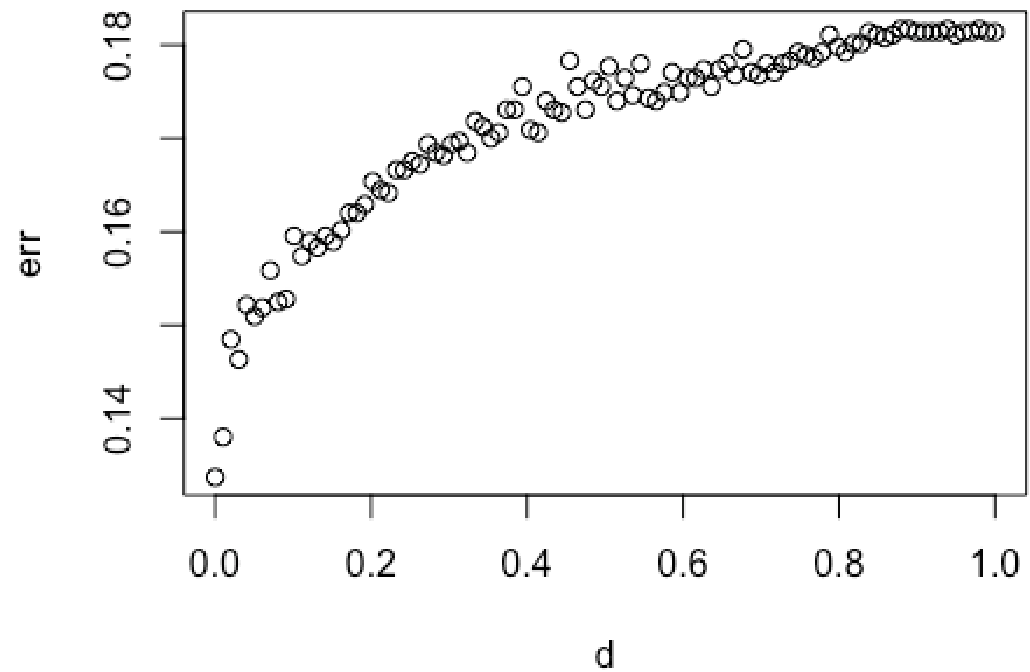



```r
#nnModel$wts
#nnModel$fitted.values

#summary(nnModel)
#The darker lines shows that the weights corresponding to these nodes are higher. i.e these  variables are more important.
#Based on the weights below are the important variables:
#AreaFt(18.57)
#QtyRequired (26.03)
#CustomerCodeM-2(4.82)

#On test data
nn.preds <- predict(nnModel, test)
#nn.preds
nn.preds.class <- as.factor(predict(nnModel, test, type = "class"))
#nn.preds.class

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


#confusion matrix
CM <- table(nn.preds.class, test$OrderConversion)
print(CM)
```

```
##
## nn.preds.class  NOT CONVERTED CONVERTED
##   CONVERTED                87      1102
##   NOT CONVERTED          1287       281
```

```r
#Checking performance of neural network
error_metric = function(CM)
{
```

```
  TN =CM[2,1]
  TP =CM[1,2]
  FP =CM[2,2]
  FN =CM[1,1]
  recall = (TP) / (TP+FN)
  precision = (TP)/(TP+FP) #calculating precision of test data
  falsePositiveRate = (FP) / (FP+TN)
  falseNegativeRate = (FN) / (FN+TP)
  error =(FP+FN)/(TP+TN+FP+FN)
  modelPerf <- list("precision" = precision,
                    "recall" = recall,
                    "falsepositiverate" = falsePositiveRate,
                    "falsenegativerate" = falseNegativeRate,
                    "error" = error)
  return(modelPerf)
}

outPutlist <- error_metric(CM)

df <- ldply(outPutlist, data.frame)
setNames(df, c("", "Values"))

##                         Values
## 1         precision 0.79681851
## 2            recall 0.92682927
## 3 falsepositiverate 0.17920918
## 4 falsenegativerate 0.07317073
## 5             error 0.13347842

#Both precision and recall values have increased. Our model is performing
better on balanced test data.
#precision = 81.4%
#recall = 93.52%
#Error rate = 12.1%


#Creating validation data set
set.seed(156)
indx <- sample(2, nrow(train), replace = T, prob = c(0.5, 0.5))
train2 <- train[indx == 1, ]
validation <- train[indx == 2, ]

plot(d, err)
```




```
#Choosing the d value for which the error is minimum. That is the best decay
value.
#From the graph it is 0
```

Decision trees construction on balanced Sample data set

```
#str(balanced.data)

dt_data_balanced <- balanced.data
set.seed(96)

index <- sample(2, replace = T, nrow(dt_data_balanced), prob = c(0.7,0.3))

train <- dt_data_balanced[index== 1,]

test <- dt_data_balanced[index== 2,]

MyFormula = OrderConversion ~.

mytree_70_30_basic <- rpart(MyFormula, data=train)

#summary(mytree_70_30_basic)
```

```
#print(mytree_70_30_basic)

rpart.plot(mytree_70_30_basic)
#From the decision tree, below are the important variables:
#CustomerCode
#ITEM_NAME
#AreaFt
#QtyRequired

#Predict function to predict the classes for the decision tree
mytree_70_30_basic for training data.
mytree_train_predict_70_30 <- predict(mytree_70_30_basic, data = train , type
= "class")

#Calculating the training error by comparing predicted classes with response
variable of original dataset.
mytree_train_error_70_30 <- mean(mytree_train_predict_70_30 !=
train$OrderConversion)

mytree_train_error_70_30

## [1] 0.1768255

#Error on balanced training data is 15.9%

table(train$OrderConversion)

##
## NOT CONVERTED     CONVERTED
##          3217          3247

#50% of the samples will be converted to orders on balanced data set

#Predict function to predict the classes for the decision tree mytree_70_30
for testing data.
mytree_test_predict_70_30 <- predict(mytree_70_30_basic, newdata = test, type
= "class")

#Calculating the testing error by comparing predicted classes with response
variable of original dataset.
mytree_test_error_70_30 <- mean(mytree_test_predict_70_30 !=
test$OrderConversion)

mytree_test_error_70_30

## [1] 0.1707145

#Error on balanced testing data is 16.1%
```

```
#Calculating the performance of the model by finding the difference between
the test error & train data.
diff_70_30 = mytree_test_error_70_30 - mytree_train_error_70_30

diff_70_30

## [1] -0.006110958

#Difference between training and testing balanced data is 0.15%


#APPLYING PARAMETER VALUES TO ARRIVE AT BETTER PERFORMANCE FOR 70-30 MODEL
#Creating vectors for minsplit and minbucket values to be used for different
combinations to test performance CP: 0.01 with least xerror of 0.3301212

msplt <- c(12,48,102)
mbckt <- c(4,16,34)

for (i in msplt)
{
  for (j in mbckt)
  {
    #Using rpart function to construct the decision tree based on training
data,split on gini.
    mytree_70_30 <- rpart(MyFormula, data = train, parms = list(split="gini")
,control = rpart.control (minsplit = i,minbucket = j,cp=0.01))

    #Predict function to predict the classes for the decision tree
mytree_70_30 for training data.
    mytree_train_predict_70_30 <- predict(mytree_70_30, data = train , type =
"class")


    #Calculating the training error by comparing predicted classes with
response variable of original dataset.
    mytree_train_error_70_30 <- mean(mytree_train_predict_70_30 !=
train$OrderConversion)

    #Predict function to predict the classes for the decision tree
mytree_70_30 for testing data.
    mytree_test_predict_70_30 <- predict(mytree_70_30, newdata = test, type =
"class")

    #Calculating the testing error by comparing predicted classes with
response variable of original dataset.
    mytree_test_error_70_30 <- mean(mytree_test_predict_70_30 !=
test$OrderConversion)

    #Calculating the performance of the model by finding the difference
```

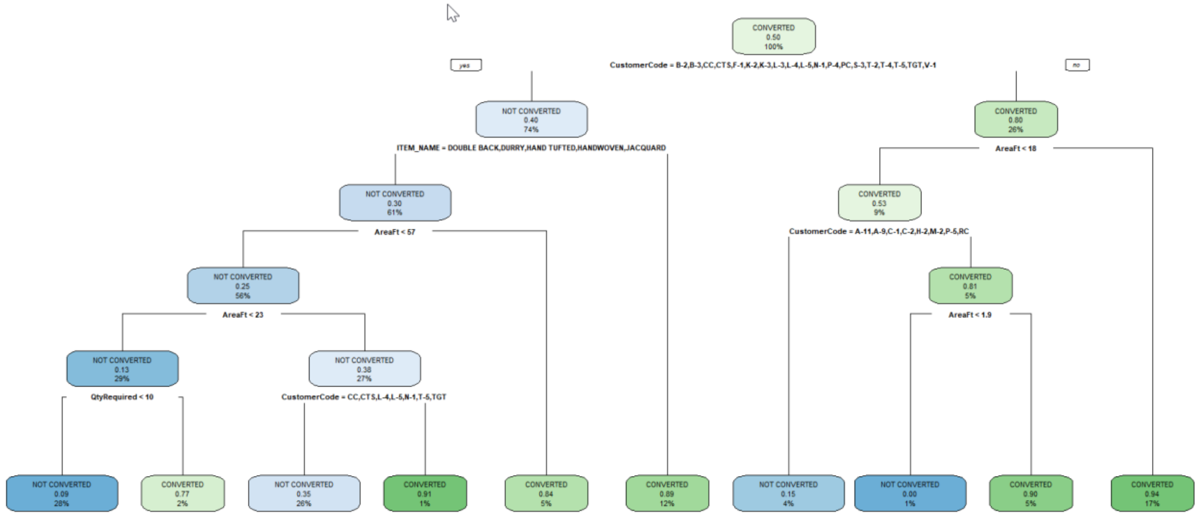
```
between the test error & train data.
    diff_70_30 = mytree_test_error_70_30 - mytree_train_error_70_30

    #Confusion Matrix for 70:30 Split
    #As the sample costs are high, the precision should be high as samples
should only be sent for actual conversion not false positive

    cfmt <- table(mytree_train_predict_70_30,train$OrderConversion)
    print (cfmt)
    fp = cfmt[2,1]
    fn = cfmt[1,2]
    tn = cfmt[2,2]
    tp = cfmt[1,1]

    #Calculating precision by dividing true positive with the sum of true
positive and false positive.
    precision_train =  (tp)/(tp+fp)
    accuracymodel_train = (tp+tn)/(tp+tn+fp+fn)
    recall_train = (tp)/(tp+fn)
    fscore_train =
(2*(recall_train*precision_train))/(recall_train+precision_train)

    #Confusion matrix on test data
    cfmt <- table(mytree_test_predict_70_30,test$OrderConversion)
    print(cfmt)

    #Calculating precision by dividing true positive with the sum of true
positive and false positive.
    precision_test =  (tp)/(tp+fp)
    accuracymodel_test = (tp+tn)/(tp+tn+fp+fn)
    recall_test = (tp)/(tp+fn)
    fscore_test = (2*(recall_test*precision_test))/(recall_test +
precision_test)

    #Printing the values for train data error, test data error, performance
and other parameters.
    print(paste("Train data error: ", mytree_train_error_70_30))
    print(paste("Test data error: ", mytree_test_error_70_30))
    print(paste("Difference/performance", diff_70_30))
    print(paste("precision of training data: ", precision_train))
    print(paste("accuracy of training data: ", accuracymodel_train))
    print(paste("recall of training data: ", recall_train))
    print(paste("F-score of training data: ", fscore_train))
    print(paste("precision of test data: ", precision_test))
    print(paste("accuracy of test data: ", accuracymodel_test))
    print(paste("recall of test data: ", recall_test))
    print(paste("F-score of test data: ", fscore_test))
  }
}
```

```
##
## mytree_train_predict_70_30 NOT CONVERTED CONVERTED
##              NOT CONVERTED          3042       968
##              CONVERTED               175      2279
##
## mytree_test_predict_70_30 NOT CONVERTED CONVERTED
##             NOT CONVERTED          1365       416
##             CONVERTED                69       991
## [1] "Train data error:  0.176825495049505"
## [1] "Test data error:  0.170714537134812"
## [1] "Difference/performance -0.00611095791469327"
## [1] "precision of training data:  0.94560149207336"
## [1] "accuracy of training data:  0.823174504950495"
## [1] "recall of training data:  0.75860349127182"
## [1] "F-score of training data:  0.841843088418431"
## [1] "precision of test data:  0.94560149207336"
## [1] "accuracy of test data:  0.823174504950495"
## [1] "recall of test data:  0.75860349127182"
## [1] "F-score of test data:  0.841843088418431"
##
## mytree_train_predict_70_30 NOT CONVERTED CONVERTED
##              NOT CONVERTED          3042       968
##              CONVERTED               175      2279
##
## mytree_test_predict_70_30 NOT CONVERTED CONVERTED
##             NOT CONVERTED          1365       416
##             CONVERTED                69       991
## [1] "Train data error:  0.176825495049505"
## [1] "Test data error:  0.170714537134812"
## [1] "Difference/performance -0.00611095791469327"
## [1] "precision of training data:  0.94560149207336"
## [1] "accuracy of training data:  0.823174504950495"
## [1] "recall of training data:  0.75860349127182"
## [1] "F-score of training data:  0.841843088418431"
## [1] "precision of test data:  0.94560149207336"
## [1] "accuracy of test data:  0.823174504950495"
## [1] "recall of test data:  0.75860349127182"
## [1] "F-score of test data:  0.841843088418431"
##
## mytree_train_predict_70_30 NOT CONVERTED CONVERTED
##              NOT CONVERTED          3042       968
##              CONVERTED               175      2279
##
## mytree_test_predict_70_30 NOT CONVERTED CONVERTED
##             NOT CONVERTED          1365       416
##             CONVERTED                69       991
## [1] "Train data error:  0.176825495049505"
## [1] "Test data error:  0.170714537134812"
## [1] "Difference/performance -0.00611095791469327"
## [1] "precision of training data:  0.94560149207336"
```

```
## [1] "accuracy of training data:  0.823174504950495"
## [1] "recall of training data:  0.75860349127182"
## [1] "F-score of training data:  0.841843088418431"
## [1] "precision of test data:  0.94560149207336"
## [1] "accuracy of test data:  0.823174504950495"
## [1] "recall of test data:  0.75860349127182"
## [1] "F-score of test data:  0.841843088418431"
##
## mytree_train_predict_70_30 NOT CONVERTED CONVERTED
##              NOT CONVERTED          3042       968
##              CONVERTED               175      2279
##
## mytree_test_predict_70_30 NOT CONVERTED CONVERTED
##             NOT CONVERTED          1365       416
##             CONVERTED                69       991
## [1] "Train data error:  0.176825495049505"
## [1] "Test data error:  0.170714537134812"
## [1] "Difference/performance -0.00611095791469327"
## [1] "precision of training data:  0.94560149207336"
## [1] "accuracy of training data:  0.823174504950495"
## [1] "recall of training data:  0.75860349127182"
## [1] "F-score of training data:  0.841843088418431"
## [1] "precision of test data:  0.94560149207336"
## [1] "accuracy of test data:  0.823174504950495"
## [1] "recall of test data:  0.75860349127182"
## [1] "F-score of test data:  0.841843088418431"
##
## mytree_train_predict_70_30 NOT CONVERTED CONVERTED
##              NOT CONVERTED          3042       968
##              CONVERTED               175      2279
##
## mytree_test_predict_70_30 NOT CONVERTED CONVERTED
##             NOT CONVERTED          1365       416
##             CONVERTED                69       991
## [1] "Train data error:  0.176825495049505"
## [1] "Test data error:  0.170714537134812"
## [1] "Difference/performance -0.00611095791469327"
## [1] "precision of training data:  0.94560149207336"
## [1] "accuracy of training data:  0.823174504950495"
## [1] "recall of training data:  0.75860349127182"
## [1] "F-score of training data:  0.841843088418431"
## [1] "precision of test data:  0.94560149207336"
## [1] "accuracy of test data:  0.823174504950495"
## [1] "recall of test data:  0.75860349127182"
## [1] "F-score of test data:  0.841843088418431"
##
## mytree_train_predict_70_30 NOT CONVERTED CONVERTED
##              NOT CONVERTED          3042       968
##              CONVERTED               175      2279
##
```

```
## mytree_test_predict_70_30 NOT CONVERTED CONVERTED
##             NOT CONVERTED          1365       416
##             CONVERTED                69       991
## [1] "Train data error:  0.176825495049505"
## [1] "Test data error:  0.170714537134812"
## [1] "Difference/performance -0.00611095791469327"
## [1] "precision of training data:  0.94560149207336"
## [1] "accuracy of training data:  0.823174504950495"
## [1] "recall of training data:  0.75860349127182"
## [1] "F-score of training data:  0.841843088418431"
## [1] "precision of test data:  0.94560149207336"
## [1] "accuracy of test data:  0.823174504950495"
## [1] "recall of test data:  0.75860349127182"
## [1] "F-score of test data:  0.841843088418431"
##
## mytree_train_predict_70_30 NOT CONVERTED CONVERTED
##              NOT CONVERTED          3042       968
##              CONVERTED               175      2279
##
## mytree_test_predict_70_30 NOT CONVERTED CONVERTED
##             NOT CONVERTED          1365       416
##             CONVERTED                69       991
## [1] "Train data error:  0.176825495049505"
## [1] "Test data error:  0.170714537134812"
## [1] "Difference/performance -0.00611095791469327"
## [1] "precision of training data:  0.94560149207336"
## [1] "accuracy of training data:  0.823174504950495"
## [1] "recall of training data:  0.75860349127182"
## [1] "F-score of training data:  0.841843088418431"
## [1] "precision of test data:  0.94560149207336"
## [1] "accuracy of test data:  0.823174504950495"
## [1] "recall of test data:  0.75860349127182"
## [1] "F-score of test data:  0.841843088418431"
##
## mytree_train_predict_70_30 NOT CONVERTED CONVERTED
##              NOT CONVERTED          3042       968
##              CONVERTED               175      2279
##
## mytree_test_predict_70_30 NOT CONVERTED CONVERTED
##             NOT CONVERTED          1365       416
##             CONVERTED                69       991
## [1] "Train data error:  0.176825495049505"
## [1] "Test data error:  0.170714537134812"
## [1] "Difference/performance -0.00611095791469327"
## [1] "precision of training data:  0.94560149207336"
## [1] "accuracy of training data:  0.823174504950495"
## [1] "recall of training data:  0.75860349127182"
## [1] "F-score of training data:  0.841843088418431"
## [1] "precision of test data:  0.94560149207336"
## [1] "accuracy of test data:  0.823174504950495"
```

```
## [1] "recall of test data:  0.75860349127182"
## [1] "F-score of test data:  0.841843088418431"
##
## mytree_train_predict_70_30 NOT CONVERTED CONVERTED
##              NOT CONVERTED          3042       968
##              CONVERTED               175      2279
##
## mytree_test_predict_70_30 NOT CONVERTED CONVERTED
##             NOT CONVERTED          1365       416
##             CONVERTED                69       991
## [1] "Train data error:  0.176825495049505"
## [1] "Test data error:  0.170714537134812"
## [1] "Difference/performance -0.00611095791469327"
## [1] "precision of training data:  0.94560149207336"
## [1] "accuracy of training data:  0.823174504950495"
## [1] "recall of training data:  0.75860349127182"
## [1] "F-score of training data:  0.841843088418431"
## [1] "precision of test data:  0.94560149207336"
## [1] "accuracy of test data:  0.823174504950495"
## [1] "recall of test data:  0.75860349127182"
## [1] "F-score of test data:  0.841843088418431"

#Performance has been the same irrespective of different values of msplt and
mbckt. But the performance has increased on balanced data.
#Train error - 15.90%
#Test error - 16.10%
#Difference between Train and Test error - 0.15%
#Precision of Training Data - 93.1%
#Accuracy of Training Data - 84%
#Recall of Training Data - 78.6%
#F-Score of Training data - 85.3%
#Precision of Test Data - 93.1%
#Accuracy of Test Data - 84%
#Recall of Test Data - 78.6%
#F-Score of Test data - 85.3%

rpart.plot(mytree_70_30)
```



```
#Decision trees from the above plot:
#A strong decision rule from the above tree is:


#If CustomerCode is equal to B-2, B-3 CC, CTS, F-1, K-2, K-3, L-3, L-4, L-5,
M-2, N-1, P-4, PC, S-3, T-2, T-5, TGT, V-1
#and ITEM_NAME is equal to DOUBLE BACK, DURRAY, HAND TUFTED, HANDLOOM,
HANDWOVEN, JACQUARD
#and AreaFt<23
#and QtyReequired < 10
#THEN NOT CONVERTED


#If CustomerCode is not equal to B-2, B-3 CC, CTS, F-1, K-2, K-3, L-3, L-4,
#L-5, M-2, N-1, P-4, PC, S-3, T-2, T-5, TGT, V-1 and AreaFt<18 THEN CONVERTED

#Comparing results of unbalanced data set and balanced data set for decision
trees.

knitr::include_graphics("DTPerformance.jpeg")
```

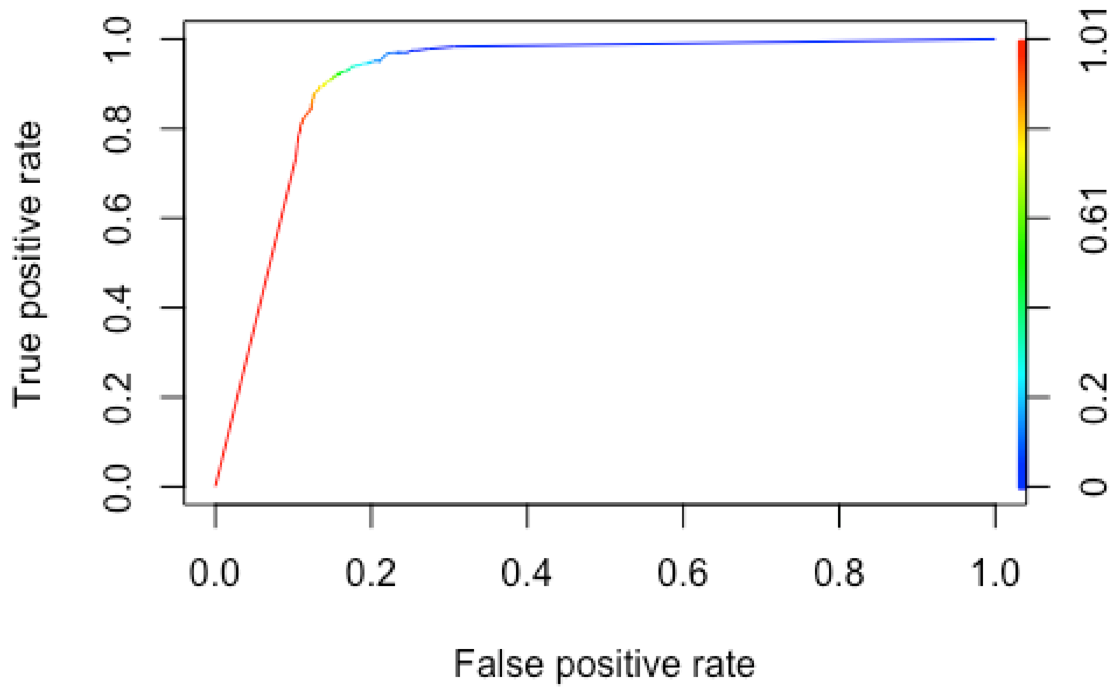
On Unbalanced data												
msplt	mbckt	Train error	Test error	Difference between Train and Test error	Precision of Training Data	Accuracy of Training Data	Recall of Training Data	F-Score of Training data	Precision of Test Data	Accuracy of Test Data	Recall of Test Data	F-Score of Test data
12	4	9.20%	9.50%	0.30%	65.70%	90.70%	84.80%	74.10%	65.70%	90.70%	84.80%	74.10%
	16	9.20%	9.50%	0.30%	65.70%	90.70%	84.80%	74.10%	65.70%	90.70%	84.80%	74.10%
	34	9.20%	9.50%	0.30%	65.70%	90.70%	84.80%	74.10%	65.70%	90.70%	84.80%	74.10%
48	4	9.20%	9.50%	0.30%	65.70%	90.70%	84.80%	74.10%	65.70%	90.70%	84.80%	74.10%
	16	9.20%	9.50%	0.30%	65.70%	90.70%	84.80%	74.10%	65.70%	90.70%	84.80%	74.10%
	34	9.20%	9.50%	0.30%	65.70%	90.70%	84.80%	74.10%	65.70%	90.70%	84.80%	74.10%
102	4	9.20%	9.50%	0.30%	65.70%	90.70%	84.80%	74.10%	65.70%	90.70%	84.80%	74.10%
	16	9.20%	9.50%	0.30%	65.70%	90.70%	84.80%	74.10%	65.70%	90.70%	84.80%	74.10%
	34	9.20%	9.50%	0.30%	65.70%	90.70%	84.80%	74.10%	65.70%	90.70%	84.80%	74.10%
On Balanced data												
12	4	15.90%	16.10%	0.15%	93.10%	84%	78.60%	85.30%	93.10%	84%	78.60%	85.30%
	16	15.90%	16.10%	0.15%	93.10%	84%	78.60%	85.30%	93.10%	84%	78.60%	85.30%
	34	15.90%	16.10%	0.15%	93.10%	84%	78.60%	85.30%	93.10%	84%	78.60%	85.30%
48	4	15.90%	16.10%	0.15%	93.10%	84%	78.60%	85.30%	93.10%	84%	78.60%	85.30%
	16	15.90%	16.10%	0.15%	93.10%	84%	78.60%	85.30%	93.10%	84%	78.60%	85.30%
	34	15.90%	16.10%	0.15%	93.10%	84%	78.60%	85.30%	93.10%	84%	78.60%	85.30%
102	4	15.90%	16.10%	0.15%	93.10%	84%	78.60%	85.30%	93.10%	84%	78.60%	85.30%
	16	15.90%	16.10%	0.15%	93.10%	84%	78.60%	85.30%	93.10%	84%	78.60%	85.30%
	34	15.90%	16.10%	0.15%	93.10%	84%	78.60%	85.30%	93.10%	84%	78.60%	85.30%

Random Forest on balanced Sample data set

```
sample_data_rf_balanced <- balanced.data
#str(sample_data_rf_balanced)

#Determining important variables
rf_imp <- randomForest(OrderConversion ~ ., data=sample_data_rf_balanced,
mtry = sqrt(ncol(sample_data_rf_balanced)-1),
                       ntree = 100, proximity = T , importance = T)

importance(rf_imp, type = 2) #MeanDecreaseGini

##              MeanDecreaseGini
## CustomerCode        687.77007
## CountryName         328.30934
## QtyRequired         236.18985
## ITEM_NAME           807.73662
## ShapeName            25.48503
## AreaFt             1017.62657

#Based on the MeanDecreaseGini values below are the variables in order of
importance:
#AreaFt
#ITEM_NAME
#CustomerCode
#CountryName

ind <- sample(2, nrow(sample_data_rf_balanced), replace = T, prob = c(0.7,
0.3))
Train <- sample_data_rf_balanced[ind == 1, ]
Validation <- sample_data_rf_balanced[ind == 2, ]
pr.err <- c()
```

```
for(mt in seq(1,ncol(Train))){
  rf1 <- randomForest(OrderConversion ~., data = Train, ntree = 100,
                      mtry = ifelse(mt == ncol(Train),
                                    mt-1, mt))
  predicted <- predict(rf1, newdata = Validation, type = "class")
  pr.err <- c(pr.err,mean(Validation$OrderConversion != predicted))
}

#Calculating Best mtry value.
bestmtry <- which.min(pr.err)
bestmtry  #4

## [1] 6

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

##
## Call:
##  randomForest(formula = OrderConversion ~ ., data = Train, ntree = 100,
mtry = bestmtry)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 10.91%
## Confusion matrix:
##               NOT CONVERTED CONVERTED class.error
## NOT CONVERTED          3007       227  0.07019171
## CONVERTED               484      2801  0.14733638

#OOB estimate of  error rate: 11.37%


#Confusion Matrix
cfmt <- table(predicted,Validation$OrderConversion)
print(cfmt)

##
## predicted       NOT CONVERTED CONVERTED
##   NOT CONVERTED          1307       218
##   CONVERTED               110      1151

fn =cfmt[2,1]
tp =cfmt[2,2]
fp =cfmt[1,2]
tn =cfmt[1,1]

#Calculating precision by dividing true positive with the sum of true
positive and false positive.
```

```r
precision_test =  (tp)/(tp+fp)
accuracymodel_test = (tp+tn)/(tp+tn+fp+fn)
recall_test = (tp)/(tp+fn)
fscore_test = (2*(recall_test*precision_test))/(recall_test + precision_test)
print(paste("precision of test data: ", precision_test))

## [1] "precision of test data:  0.840759678597516"

print(paste("accuracy of test data: ", accuracymodel_test))

## [1] "accuracy of test data:  0.882268485283561"

print(paste("recall of test data: ", recall_test))

## [1] "recall of test data:  0.912767644726408"

print(paste("F-score of test data: ", fscore_test))

## [1] "F-score of test data:  0.875285171102661"

#Random Forest produces better recall of 92.61% on unbalanced data. Has
precision of 82.98%. F-Score is 87.5%


#Drawing evaluation chart - ROC Curve

predicteddtProb <- predict(rf1, newdata = Validation, type = "prob")[,1]
pred <- prediction(predicteddtProb, Validation$OrderConversion)
perf <- performance(pred, "tpr", "fpr")
plot(perf,colorize=TRUE)
```




```
#From the graph, the threshold value is less than 0.2, which shows that the
performance has increased.
```

Logistic Regression on balanced Sample data set

```
logistic_data_balanced <- balanced.data

set.seed(96)

#As we got NA values for CountryName, removing CustomerCode column
logistic_data_balanced <- subset(logistic_data_balanced,select = -
c(CustomerCode))
ind <- sample(2, nrow(logistic_data_balanced), replace = T, prob = c(0.7,
0.3))
Train <- logistic_data_balanced[ind == 1, ]
Test <- logistic_data_balanced[ind == 2, ]

#Generalized linear Model
LogReg <- glm(OrderConversion ~., data = Train, family = "binomial")
#summary(LogReg)

#Choosing important variables based on p-value. A p-value less than 0.05 is
```

```
statistically significant.
#From our summary, below are the important variables:
#CountryNameBELGIUM
#CountryNameCANADA
#CountryNameINDIA
#CountryNameITALY
#QtyRequired
#ITEM_NAMEGUN TUFTED
#ITEM_NAMEHANDWOVEN
#ITEM_NAMEKNOTTED
#ITEM_NAMEPOWER LOOM JACQUARD
#ITEM_NAMETABLE TUFTED
#ShapeNameSQUARE
#AreaFt


#Performance of the logistic regression model
#Residual variance has decreased when compared to Null deviance which shows
the quality of prediction has improved.
#Null deviance: 8960.9  on 6463  degrees of freedom
#Residual deviance: 5577.0  on 6436  degrees of freedom
```

Boosting on balanced Sample data set

```
boosting_data_balanced <- balanced.data
set.seed(96)

#Splitting the data into train and test data
ind <- sample(2, nrow(boosting_data_balanced), replace = T, prob = c(0.7,
0.3))
train <- boosting_data_balanced[ind == 1, ]
test <- boosting_data_balanced[ind == 2, ]

#Using boosting function to build the model
model = boosting(OrderConversion ~ ., data=train, boos=TRUE, mfinal=50)

#Checking model properties
print(names(model))

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

print(model$trees[1])

## [[1]]
## n= 6464
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
```

```
##  1) root 6464 3163 NOT CONVERTED (0.5106745 0.4893255)
##    2) AreaFt< 39.90625 4855 1877 NOT CONVERTED (0.6133883 0.3866117)
##      4) CustomerCode=A-11,A-9,B-2,B-3,C-1,C-2,CC,CTS,F-1,F-2,H-2,K-2,L-4,L-5,M-2,N-1,P-4,P-5,RC,S-3,T-2,T-5,TGT,V-1 4290 1374 NOT CONVERTED (0.6797203 0.3202797)
##        8) ITEM_NAME=DOUBLE BACK,DURRY,HAND TUFTED,HANDLOOM,HANDWOVEN,JACQUARD 3778  920 NOT CONVERTED (0.7564849 0.2435151)
##         16) AreaFt< 15.875 2171  291 NOT CONVERTED (0.8659604 0.1340396)
##           32) QtyRequired< 9.5 2045  204 NOT CONVERTED (0.9002445 0.0997555) *
##           33) QtyRequired>=9.5 126   39 CONVERTED (0.3095238 0.6904762) *
##         17) AreaFt>=15.875 1607  629 NOT CONVERTED (0.6085874 0.3914126)
##           34) CustomerCode=C-2,CC,CTS,F-1,F-2,L-4,N-1,T-2,T-5,TGT 1478 519 NOT CONVERTED (0.6488498 0.3511502) *
##           35) CustomerCode=A-9,C-1,H-2,L-5,M-2,P-4,P-5,RC,S-3 129   19 CONVERTED (0.1472868 0.8527132) *
##        9) ITEM_NAME=GUN TUFTED,INDO-TIBBETAN,KNOTTED,POWER LOOM JACQUARD,TABLE TUFTED 512   58 CONVERTED (0.1132812 0.8867188) *
##      5) CustomerCode=E-2,F-6,I-2,JL,L-3,M-1,PD,T-4 565   62 CONVERTED (0.1097345 0.8902655) *
##    3) AreaFt>=39.90625 1609  323 CONVERTED (0.2007458 0.7992542)
##      6) CustomerCode=CC,P-4,T-2,T-5 819  288 CONVERTED (0.3516484 0.6483516)
##       12) ITEM_NAME=DOUBLE BACK,DURRY,HAND TUFTED,HANDLOOM,HANDWOVEN,JACQUARD 641  288 CONVERTED (0.4492980 0.5507020)
##         24) AreaFt< 57.1736 360  124 NOT CONVERTED (0.6555556 0.3444444) *
##         25) AreaFt>=57.1736 281   52 CONVERTED (0.1850534 0.8149466) *
##       13) ITEM_NAME=GUN TUFTED,KNOTTED,POWER LOOM JACQUARD,TABLE TUFTED 178    0 CONVERTED (0.0000000 1.0000000) *
##      7) CustomerCode=A-11,A-9,C-1,C-2,F-1,H-2,I-2,JL,M-1,M-2,N-1,P-5,PD,S-3,TGT 790   35 CONVERTED (0.0443038 0.9556962) *

#Predicting on test data
pred = predict(model, test)

#confusion matrix
print(pred$confusion)

##                Observed Class
## Predicted Class NOT CONVERTED CONVERTED
##   CONVERTED                90      1119
##   NOT CONVERTED          1344       288

#Error on the test data is 13.1%
print(pred$error)

## [1] 0.1330517

#probability of each class in the test data set
result = data.frame(test$OrderConversion, pred$prob, pred$class)
```

```
#print(result)

#cross-validation method
cvmodel = boosting.cv(OrderConversion ~ ., data=boosting_data_balanced,
boos=TRUE, mfinal=10, v=5)

## i:  1 Sun May  1 23:18:53 2022
## i:  2 Sun May  1 23:18:56 2022
## i:  3 Sun May  1 23:18:58 2022
## i:  4 Sun May  1 23:19:01 2022
## i:  5 Sun May  1 23:19:04 2022

print(cvmodel[-1])

## $confusion
##                Observed Class
## Predicted Class NOT CONVERTED CONVERTED
##   CONVERTED               386      3652
##   NOT CONVERTED          4265      1002
##
## $error
## [1] 0.1491671

#print(data.frame(boosting_data_balanced$OrderConversion, cvmodel$class))
#error from cross validation is 14.9
```

Selecting best models:

We are considering Precision as the performance measure for our models as we have to avoid False Positives as it would cost a lot to the company. Below are the observations made for different models:

Neural network: precision - 80% Recall - 92%

Decision Trees: precision - 93.10% recall - 78.60%

Random Forest: precision - 84% recall - 91%

As, Precision is high in Decision Trees. We are considering Decision trees as the best model

QUESTION 4

The data that we are using for clustering is 'Data for Clustering'. The first step involves converting all the variables to numerical values.

Clustering all the customers into certain groups based on their purchasing habits of products based on ITEM_NAME, Shape Name etc.. Then come up with different strategies for different groups (closeness between data points and distance) which can in turn be used for recommendation of sending appropriate samples to ensure conversion.

Use elbow method to come up with optimal number of clusters.

Feature engineering would involve selecting appropriate columns from the data set. We can use columns such as ‘Country Name’, ‘Shape’ and ‘Color’ from the raw data set to improve the clusters.

All of these are implemented in Question 6.

QUESTION 5

There are two clustering algorithms which can be used for segmenting Champo Carpets’s customers:

1. k-means clustering

We are choosing k-means clustering because of the following advantages. As our data set is large and has many variables, we use k-means as it performs better. Also, as we do not have a target variable (unsupervised), and do not know the group the clusters belong to, we use k-means.

2. hierarchical clustering

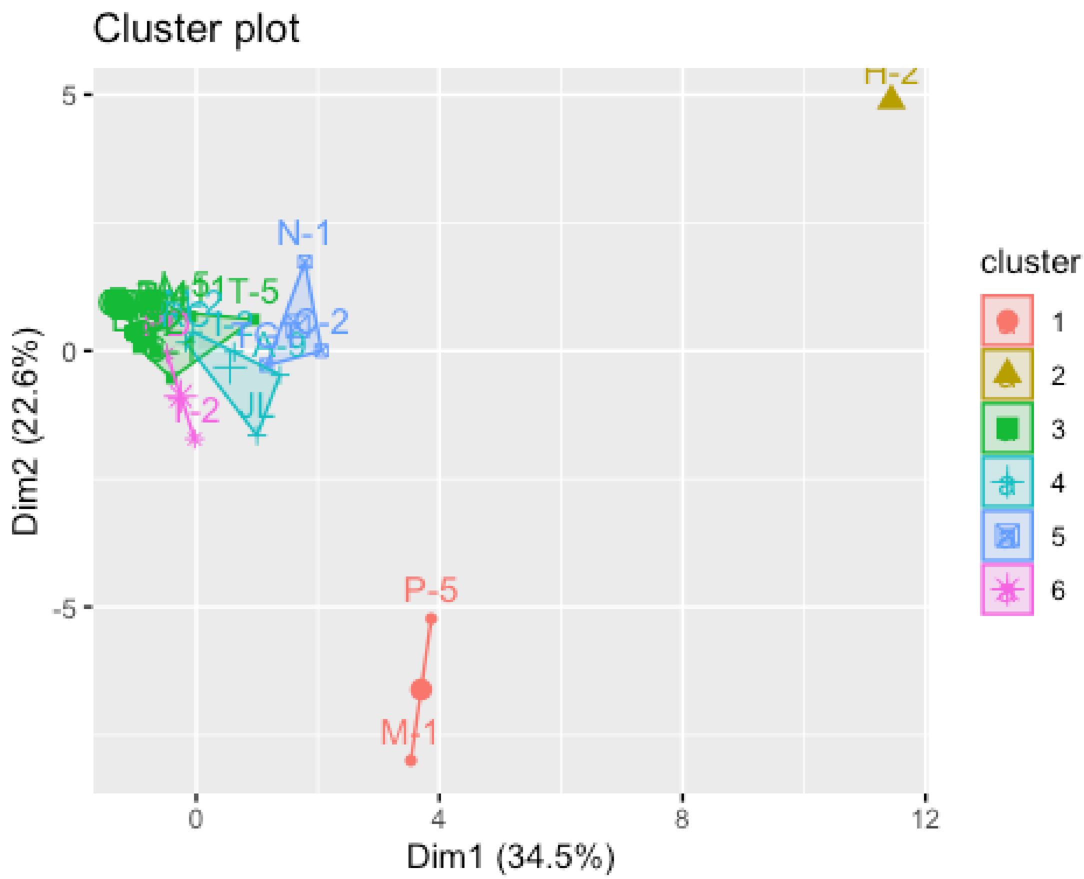
Euclidean distance measure is suitable in case of k-means clustering. i.e distance between the object and its cluster centroid.

QUESTION 6

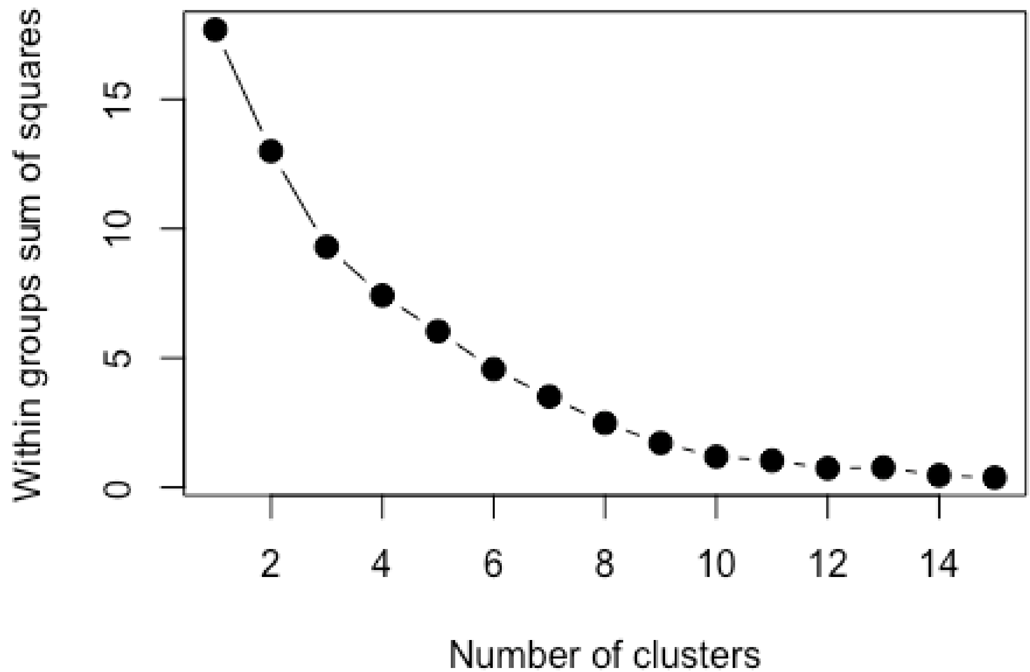
k-means clustering

```
cluster<-read_excel("Champo Carpets.xlsx", sheet=6)

df <- cluster

final <- df

final <- data.frame(column_to_rownames(final, var = "Row Labels"))

myscale <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}

final <- final %>% mutate_if(is.numeric, myscale)

set.seed(1234)

km <- kmeans(final, centers = 6, nstart = 100)

km
```

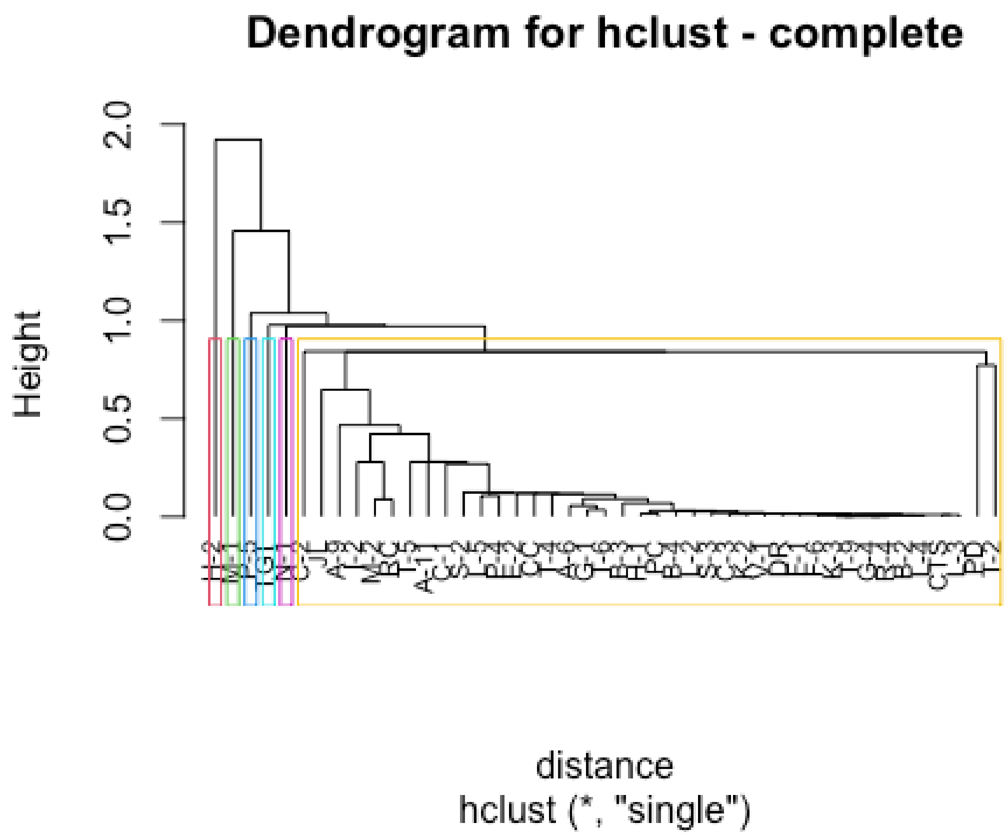
```
## K-means clustering with 6 clusters of sizes 2, 1, 32, 5, 3, 2
##
## Cluster means:
##   Sum.of.QtyRequired Sum.of.TotalArea Sum.of.Amount      DURRY   HANDLOOM
## 1         0.17744700      0.689929357    0.22157557 0.09457591 0.16648516
## 2         1.00000000      0.092998692    0.33546999 1.00000000 1.00000000
## 3         0.02059623      0.023717593    0.01060837 0.01831896 0.01818166
## 4         0.06028580      0.065044358    0.06897577 0.01757510 0.04290770
## 5         0.26036367      0.076379689    0.40698651 0.11757080 0.00000000
## 6         0.04533198      0.007526745    0.04263264 0.03670372 0.07187585
##   DOUBLE.BACK   JACQUARD HAND.TUFTED  HAND.WOVEN    KNOTTED  GUN.TUFTED
## 1  0.93123736 0.28921569  0.04160007 0.294292301 0.69080194 0.500000000
## 2  0.00000000 0.77030812  0.43852682 0.209585022 0.00000000 0.000000000
## 3  0.01255975 0.02976190  0.01337851 0.007645487 0.00294017 0.003044872
## 4  0.16120610 0.65294118  0.08062948 0.123683107 0.04312776 0.030769231
## 5  0.00000000 0.00000000  0.43693389 0.333333333 0.00000000 0.000000000
## 6  0.15223387 0.03501401  0.02171871 0.033289088 0.02041675 0.312820513
##   Powerloom.Jacquard INDO.TEBETAN
## 1                  0          0.0
## 2                  1          0.0
## 3                  0          0.0
## 4                  0          0.0
## 5                  0          0.0
## 6                  0          0.8
##
## Clustering vector:
## A-11  A-6  A-9  B-2  B-3  B-4  C-1  C-2  C-3   CC  CTS   DR  E-2  F-1  F-6
G-1
##    3    3    4    3    3    3    3    5    3    3    3    3    3    3    3
3
##  G-4  H-1  H-2  I-2   JL  K-2  K-3  L-2  L-3  L-4  L-5  M-1  M-2  N-1  P-4
P-5
##    3    3    2    4    4    3    3    3    3    3    3    1    4    5    3
1
##   PC   PD  R-4   RC  S-2  S-3  T-2  T-4  T-5  T-6  T-9  TGT  V-1
##    3    6    3    4    3    3    6    3    3    3    3    5    3
##
## Within cluster sum of squares by cluster:
## [1] 1.0593578 0.0000000 0.6520855 0.7648458 1.7939465 0.2978700
##  (between_SS / total_SS =  74.2 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
"tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"

#Accuracy of the model is 74.2%

fviz_cluster(km, data = final)
```




```
#For getting the summary of each cluster characteristics
cluster1 <- df[which( km$cluster == 1),]
#summary(cluster1)
cluster2 <- df[which( km$cluster == 2),]
#summary(cluster2)
cluster3 <- df[which( km$cluster == 3),]
#summary(cluster3)
cluster4 <- df[which( km$cluster == 4),]
#summary(cluster4)
cluster4 <- df[which( km$cluster == 5),]
#summary(cluster5)
cluster4 <- df[which( km$cluster == 6),]
#summary(cluster6)

mydata<-final
wss<-(nrow(mydata)-1)*sum(apply(mydata,2,var))
for(i in 1:15)
  wss[i]<-sum(kmeans(mydata, centers = i)$withinss)
plot(1:15,wss,type = "b",xlab = "Number of clusters", ylab = "Within groups
sum of squares", pch=20, cex=2)
```




```
#Hierarchical clustering method
distance <- dist(final, method = "euclidean")
hcomplete <- hclust(distance, method = "single")
plot(hcomplete, cex = 0.7, hang = -2, main = "Dendrogram for hclust - complete")


clusters <- cutree(hcomplete, k =6)

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



From the elbow graph in Question 6, we can observe that, after k=9, the graph flattens. Hence, we are not going beyond k=9. Also, choosing k=9 is not an optimal solution for our data set. Hence, we are going ahead with k=6.

Significant variables are Item Name, Country Name, Quantity Required. We included columns such as Shape from raw data while building the clustering model but the accuracy of the cluster decreased (from 74.2% to 47.5%).

Cluster Characteristics:

Cluster	Country Name	Item	Quantity Required
Cluster 1	USA	Durry, Handtuft	~72,888
Cluster 2	India, USA	Durry, Handtuft, Handloom, Indo Tibetian	~42,967
Cluster 3	Belgium, Italy	Durry, Double Black, Knotted	~11,146
Cluster 4	USA	Durry, Knotted, Handwoven	~48,373
Cluster 5	Romania, USA, UK, Australia	Handtuft, Handwoven	~18,923
Cluster 6	USA	Durry, Powerloom Jaquard	~1,83,206

QUESTION 7

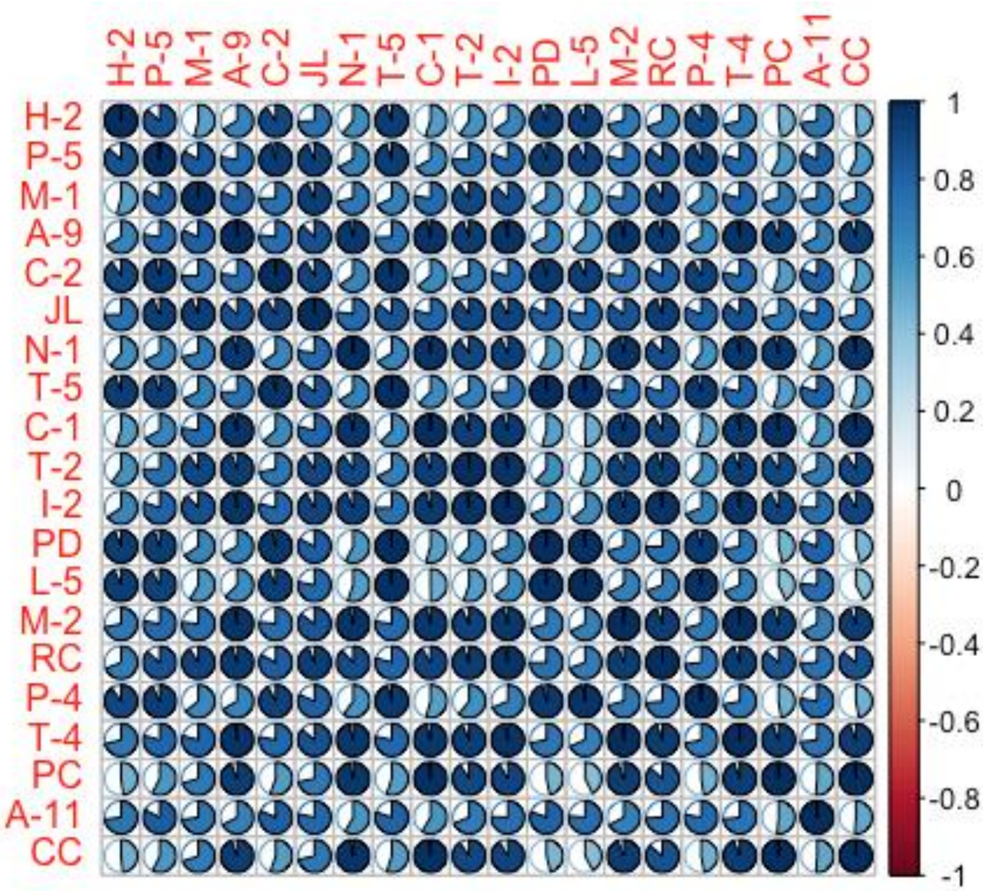
Recommender System

```
rec<-read_excel("Champo Carpets.xlsx", sheet=5)

## New names:
## • `Customer` -> `Customer...1`
## • `` -> `...22`
## • `` -> `...23`
## • `Customer` -> `Customer...24`

#View(rec)

rec_rbind <- rec[,c(1:21)]

rec_cbind <- rec[,-c(1:24)]

rec_rbind <- data.frame(column_to_rownames(rec_rbind, var = "Customer...1"))

#View(rec_rbind)
#View(rec_cbind)

dist(rec_rbind)

##             H-2        P-5        M-1        A-9        C-2         JL
## P-5  155166.504
## M-1  192085.610  41424.222
## A-9  187047.924  39824.292  12137.416
## C-2  143267.992  20326.567  54816.492  51505.000
## JL   186969.702  37109.834   6883.114  10097.962  49397.121
## N-1  154391.738  66759.762  82583.861  73702.547  67804.898  79683.946
## T-5  151158.056  16505.385  43702.158  39800.535  18387.174  38277.540
## C-1  199130.845  51744.336  13667.686  16583.060  63809.821  15943.230
## T-2  199265.243  51748.998  13236.822  17457.858  63809.301  15706.851
## I-2  197185.792  49436.290  11465.961  14791.275  61449.245  13412.778
## PD   189058.611  42212.633  13169.324  15933.738  53788.943  11345.992
## L-5  168577.921  25444.565  28871.212  27196.351  35308.615  23573.227
## M-2  197026.437  49900.902  12815.722  15124.421  61878.398  14362.326
## RC   200690.178  53359.689  15418.009  19774.196  65372.170  17802.678
## P-4  183047.307  36236.504  15810.795  16618.432  48195.110  11793.925
## T-4  197905.316  50651.274  13250.483  16142.247  62625.353  15093.045
## PC   202132.331  55106.120  17080.325  21181.617  67044.621  19536.146
## A-11 200810.953  53694.549  16183.390  20813.072  65621.108  18493.543
## CC   199197.940  52018.301  13946.079  16264.672  64023.855  16181.018
##             N-1        T-5        C-1        T-2        I-2         PD
## P-5
## M-1
## A-9
```

```
## C-2
## JL
## N-1
## T-5   68047.305
## C-1   89362.462  51137.128
## T-2   90451.683  51258.008   2195.150
## I-2   87923.180  48988.134   2821.416   2937.959
## PD    87009.452  40550.373  12221.264  11891.060  10535.528
## L-5   78282.938  19913.102  33992.017  33934.901  32013.082  22562.826
## M-2   87963.859  49054.953   2298.987   3231.146   2029.975  10445.147
## RC    92825.126  52660.985   3758.137   2865.750   5207.437  12589.450
## P-4   82700.145  34512.729  18772.025  18605.946  16797.166   8600.222
## T-4   89042.409  49812.946   1693.321   2400.185   2189.449  10711.854
## PC    93998.491  54245.666   4766.536   4195.277   6707.987  14106.646
## A-11  93850.175  52854.001   5193.006   4124.708   6244.604  12580.696
## CC    88804.778  51277.761   1162.109   2937.532   3227.615  12759.495
##             L-5        M-2         RC        P-4        T-4         PC
## P-5
## M-1
## A-9
## C-2
## JL
## N-1
## T-5
## C-1
## T-2
## I-2
## PD
## L-5
## M-2   31914.806
## RC    34897.939   5006.740
## P-4   16421.870  16804.912  19593.117
## T-4   32517.518   1393.740   3975.657  17335.009
## PC    36380.729   6309.220   1874.005  21103.698   5250.842
## A-11  34889.755   6198.854   2095.443  19669.307   5017.963   2489.556
## CC    34271.294   2557.038   4530.614  19097.750   2267.843   5305.061
##            A-11
## P-5
## M-1
## A-9
## C-2
## JL
## N-1
## T-5
## C-1
## T-2
## I-2
## PD
## L-5
## M-2
```

```
## RC
## P-4
## T-4
## PC
## A-11
## CC     5931.410
```

```
#Using Euclidean distance, we try to find the closes customer. For example,
lets take customer M-1 and the closest customer would be PC with distance of
1874.005.
#From the recommendation data set, we can see that, M-1 has bought Double
Wowen, Double Back, Knotted but PC has not bought. Hence, we can recommend
products these products to PC.

corr <- cor(rec_cbind)
corrplot(corr, method="pie")
```




```
# Using Pearson correlation, we find the highly correlated customers. For
example, take customer C-2, the closest customers would be P5 and PD.

#From the recommendation data set, we can see that, C-2 has bought Hand
Tufted but PD has not bought. Hence, we can recommend this product to PD.
```

```
m <- data.matrix(rec_cbind)
cosine(m)

##            H-2       P-5       M-1       A-9       C-2        JL       N-1
## H-2  1.0000000 0.8906404 0.6297218 0.7311610 0.9168037 0.7948763 0.6836669
## P-5  0.8906404 1.0000000 0.8629957 0.8159763 0.9645909 0.9461141 0.7313188
## M-1  0.6297218 0.8629957 1.0000000 0.8492928 0.7887762 0.9463881 0.7633831
## A-9  0.7311610 0.8159763 0.8492928 1.0000000 0.8060661 0.8976825 0.9720883
## C-2  0.9168037 0.9645909 0.7887762 0.8060661 1.0000000 0.9276485 0.7062480
## JL   0.7948763 0.9461141 0.9463881 0.8976825 0.9276485 1.0000000 0.8146264
## N-1  0.6836669 0.7313188 0.7633831 0.9720883 0.7062480 0.8146264 1.0000000
## T-5  0.9552671 0.9563769 0.7325080 0.7903624 0.9769163 0.8799629 0.7173661
## C-1  0.6202510 0.7162060 0.8061971 0.9725000 0.6877453 0.8202816 0.9767544
## T-2  0.6746653 0.7946883 0.9217630 0.9524451 0.7658427 0.9195475 0.9200747
## I-2  0.7218687 0.8391778 0.9000519 0.9805441 0.8230105 0.9336039 0.9432134
## PD   0.9437759 0.9527514 0.7067404 0.7214953 0.9671601 0.8499124 0.6360501
## L-5  0.9404205 0.9315627 0.6514787 0.6873931 0.9424290 0.8152258 0.6210414
## M-2  0.7702189 0.8176482 0.8084878 0.9800070 0.8046142 0.8780207 0.9751573
## RC   0.7524100 0.8890482 0.9267529 0.9625807 0.8560053 0.9507726 0.9035589
## P-4  0.9205331 0.9421586 0.7083314 0.7272964 0.9343588 0.8504401 0.6675017
## T-4  0.7596125 0.8259451 0.8256114 0.9800283 0.8128106 0.8833895 0.9691105
## PC   0.5531876 0.6327831 0.7446070 0.9441998 0.6122598 0.7558704 0.9674985
## A-11 0.7707996 0.8492960 0.7693828 0.7239152 0.8406001 0.8156180 0.6425968
## CC   0.5652085 0.6343337 0.7430103 0.9519192 0.6137220 0.7561831 0.9787965
##            T-5       C-1       T-2       I-2        PD       L-5       M-2
## H-2  0.9552671 0.6202510 0.6746653 0.7218687 0.9437759 0.9404205 0.7702189
## P-5  0.9563769 0.7162060 0.7946883 0.8391778 0.9527514 0.9315627 0.8176482
## M-1  0.7325080 0.8061971 0.9217630 0.9000519 0.7067404 0.6514787 0.8084878
## A-9  0.7903624 0.9725000 0.9524451 0.9805441 0.7214953 0.6873931 0.9800070
## C-2  0.9769163 0.6877453 0.7658427 0.8230105 0.9671601 0.9424290 0.8046142
## JL   0.8799629 0.8202816 0.9195475 0.9336039 0.8499124 0.8152258 0.8780207
## N-1  0.7173661 0.9767544 0.9200747 0.9432134 0.6360501 0.6210414 0.9751573
## T-5  1.0000000 0.6755994 0.7300434 0.7919567 0.9905981 0.9831843 0.8120585
## C-1  0.6755994 1.0000000 0.9445017 0.9569085 0.6015959 0.5635471 0.9698869
## T-2  0.7300434 0.9445017 1.0000000 0.9783418 0.6719103 0.6244356 0.9405392
## I-2  0.7919567 0.9569085 0.9783418 1.0000000 0.7323726 0.6951651 0.9701904
## PD   0.9905981 0.6015959 0.6719103 0.7323726 1.0000000 0.9882298 0.7481740
## L-5  0.9831843 0.5635471 0.6244356 0.6951651 0.9882298 1.0000000 0.7259485
## M-2  0.8120585 0.9698869 0.9405392 0.9701904 0.7481740 0.7259485 1.0000000
## RC   0.8305914 0.9248968 0.9537922 0.9788209 0.7839820 0.7427093 0.9527431
## P-4  0.9662806 0.6105119 0.6778108 0.7426128 0.9634781 0.9766897 0.7577039
## T-4  0.8182640 0.9703088 0.9450885 0.9758380 0.7591025 0.7313322 0.9927522
## PC   0.6115377 0.9892160 0.9172219 0.9247955 0.5287674 0.4982817 0.9498858
## A-11 0.8305632 0.6378565 0.7366032 0.7921164 0.8313280 0.7967757 0.7223619
## CC   0.6142462 0.9889018 0.9168246 0.9255301 0.5274079 0.4980094 0.9498663
##             RC       P-4       T-4        PC      A-11        CC
## H-2  0.7524100 0.9205331 0.7596125 0.5531876 0.7707996 0.5652085
## P-5  0.8890482 0.9421586 0.8259451 0.6327831 0.8492960 0.6343337
## M-1  0.9267529 0.7083314 0.8256114 0.7446070 0.7693828 0.7430103
## A-9  0.9625807 0.7272964 0.9800283 0.9441998 0.7239152 0.9519192
```

```
## C-2  0.8560053 0.9343588 0.8128106 0.6122598 0.8406001 0.6137220
## JL   0.9507726 0.8504401 0.8833895 0.7558704 0.8156180 0.7561831
## N-1  0.9035589 0.6675017 0.9691105 0.9674985 0.6425968 0.9787965
## T-5  0.8305914 0.9662806 0.8182640 0.6115377 0.8305632 0.6142462
## C-1  0.9248968 0.6105119 0.9703088 0.9892160 0.6378565 0.9889018
## T-2  0.9537922 0.6778108 0.9450885 0.9172219 0.7366032 0.9168246
## I-2  0.9788209 0.7426128 0.9758380 0.9247955 0.7921164 0.9255301
## PD   0.7839820 0.9634781 0.7591025 0.5287674 0.8313280 0.5274079
## L-5  0.7427093 0.9766897 0.7313322 0.4982817 0.7967757 0.4980094
## M-2  0.9527431 0.7577039 0.9927522 0.9498858 0.7223619 0.9498663
## RC   1.0000000 0.7827366 0.9516260 0.8799051 0.7818218 0.8789910
## P-4  0.7827366 1.0000000 0.7626373 0.5458218 0.8104879 0.5478642
## T-4  0.9516260 0.7626373 1.0000000 0.9496215 0.7688168 0.9484756
## PC   0.8799051 0.5458218 0.9496215 1.0000000 0.5728461 0.9963906
## A-11 0.7818218 0.8104879 0.7688168 0.5728461 1.0000000 0.5745827
## CC   0.8789910 0.5478642 0.9484756 0.9963906 0.5745827 1.0000000

# Using Cosine similarity, we find the highly correlated customers. For
example, take customer H-2, the closest customer would be PD.

#From the recommendation data set, we can see that, H-2 has bought Hand
Tufted but PD has not bought. Hence, we can recommend this product to PD.

# We can use any of the three methods to find out closest customers and come
up with different recommendation rules.
```

QUESTION 8

Final Recommendation

Below would be the final recommendation to Champo Carpets:

1. From the logistic regression model, we can identify the important variables affecting the conversion of samples to orders. We have identified the below as important variables:

CountryName

QtyRequired

ITEM_NAME

ShapeName

AreaFt

2. From the k-means clustering model, we can identify similar customer segments and cross recommend the products. We have 6 clusters in our model which were constructed based on item name.

3. From the recommendation system, we can come up with rules which can be used for product recommendation to customers.

Using Euclidean distance, we try to find the closes customer. For example, lets take customer M-1 and the closest customer would be PC with distance of 1874.005.

From the recommendation data set, we can see that, M-1 has bought Double Wowen, Double Back, Knotted but PC has not bought. Hence, we can recommend products these products to PC.

Using Pearson correlation, we find the highly correlated customers. For example, take customer C-2, the closest customers would be P5 and PD.

From the recommendation data set, we can see that, C-2 has bought Hand Tufted but PD has not bought. Hence, we can recommend this product to PD.

Using Cosine similarity, we find the highly correlated customers. For example, take customer H-2, the closest customer would be PD.

From the recommendation data set, we can see that, H-2 has bought Hand Tufted but PD has not bought. Hence, we can recommend this product to PD.

We can use any of the three methods to find out closest customers and come up with different recommendation rules.

4. Decision rules can be used to say whether the sample would be converted or not

Example:

If CustomerCode is equal to B-2, B-3 CC, CTS, F-1, K-2, K-3, L-3, L-4, L-5, M-2, N-1, P-4, PC, S-3, T-2, T-5, TGT, V-1 and ITEM_NAME is equal to DOUBLE BACK, DURRAY, HAND TUFTED, HANDLOOM, HANDWOVEN, JACQUARD and AreaFt<23 and QtyReequired < 10 THEN NOT CONVERTED

If CustomerCode is not equal to B-2, B-3 CC, CTS, F-1, K-2, K-3, L-3, L-4, L-5, M-2, N-1, P-4, PC, S-3, T-2, T-5, TGT, V-1 and AreaFt<18 THEN CONVERTED