IDS572 – Data Mining - HW5

Vineeth Subramanian (668611450), **Akshaya Sri Subramanian Asha** (661054263), **Disha Pandey** (676191227)

Importing Libraries

```
library(readxl)
library(scales)
library(dplyr)
library(tidyverse)
library(ggplot2)
library(NeuralNetTools)
library(ipred)
library(nnet)
library(DataExplorer)
library(reshape2)
library(tracerer)
library(rpart)
library(rpart.plot)
library(caret)
library(Metrics)
library(knitr)
library(ROCR)
library(randomForest)
library(caret)
library(ROSE)
library(adabag)
library(purrr)
library(clustMixType)
library(factoextra)
```

Importing Dataset Champo Carpets - Raw Data Order and Sample, Data on Sample ONLY sheet, Data for Clustering and Data for Recommendation

```
champo_raw <- read_excel("Champo_Carpets.xlsx", sheet =2)
champo_sample_only <- read_excel("Champo_Carpets.xlsx", sheet =4)
champo_cluster <- read_excel("Champo_Carpets.xlsx", sheet =6)
champo_rec <- read_excel("Champo_Carpets.xlsx", sheet =5)
```

We will be using the Raw Data Order and Sample sheet for all Exploratory Data Analysis and Data on Sample ONLY sheet for Model Classification and Evaluation. However, we will pre process and perform Data cleaning on both the sheets. For K means, we will be using the Data for Clustering sheet and for recommendations we will be using the Data for Recommendation sheet.

Data Cleaning *Removing Unwanted variables in Raw Data Order and Sample sheet*

```r
#Data Cleaning
#Removing Unwanted variables in Raw Data Order and Sample sheet
champo_raw$CustomerOrderNo <- NULL
champo_raw$UnitName <- NULL
```

The Customer Order No and Unit Name have no inlfuence on determining the Order Conversion rate and hence we have disregarded the variables.

Removing Unwanted variables in Data on sample ONLY

```r
champo_sample_only$USA <- NULL
champo_sample_only$UK <- NULL
champo_sample_only$Italy <- NULL
champo_sample_only$Belgium <- NULL
champo_sample_only$Romania <- NULL
champo_sample_only$Australia <- NULL
champo_sample_only$India <- NULL
champo_sample_only$`Hand Tufted`<- NULL
champo_sample_only$Durry <- NULL
champo_sample_only$`Double Back`<- NULL
champo_sample_only$`Hand Woven` <- NULL
champo_sample_only$Knotted <- NULL
champo_sample_only$Jacquard <- NULL
champo_sample_only$Handloom <- NULL
champo_sample_only$Other <- NULL
champo_sample_only$REC <- NULL
champo_sample_only$Round <- NULL
champo_sample_only$Square <- NULL
```

The attributues of the countries we have removed were simply repetitive binary values of the already mentioned CountryName, ITEM_NAME and ShapeName variables and hence we have disregarded the variables.

Handling ITEM_NAME attribute

```r
champo_raw$ITEM_NAME[champo_raw$ITEM_NAME == "INDO-TIBBETAN"] <- "INDO TIBBET
AN"
champo_raw <- champo_raw[champo_raw$ITEM_NAME != "-",]
champo_sample_only$ITEM_NAME[champo_sample_only$ITEM_NAME == "INDO-TIBBETAN"]
<- "INDO TIBBETAN"
```

The ITEM_NAME attribute in both the sheets (Raw Data Order and Sample & Data on Sample ONLY) had few rows that were hypenated values. Hence, we have handled those rows with actual true values.

Converting data type to Categorical in Raw Data Order and Sample sheet & Data on Sample ONLY sheet

```r
champo_raw_names <- c('OrderType','OrderCategory','CustomerCode',
                      'CountryName','ITEM_NAME','QualityName',
                      'DesignName','ColorName','ShapeName')
```

```
champo_raw[,champo_raw_names] <- lapply(champo_raw[,champo_raw_names],factor)
champo_sample_only_names <- c('CustomerCode','CountryName','ITEM_NAME',
                              'ShapeName','Order Conversion')
champo_sample_only[,champo_sample_only_names] <-
  lapply(champo_sample_only[,champo_sample_only_names],factor)
```

Renaming the Target Variable in Data on Sample ONLY

```
#colnames(champo_sample_only) <- c("Order Conversion", "Order_Conversion")
colnames(champo_sample_only)[7] <- "Order_Conversion"
#colnames(champo_cluster[1]) <- "Rowlabels"
```

Missing Value Analysis - Raw Data Order and Sample & Data on Sample ONLY

```
colSums(is.na(champo_raw))

##     OrderType OrderCategory  CustomerCode   CountryName Custorderdate
##             0             0             0             0             0
##   QtyRequired     TotalArea        Amount     ITEM_NAME   QualityName
##             0             0             0             0             0
##    DesignName     ColorName     ShapeName        AreaFt
##             0             0             0             0

colSums(is.na(champo_sample_only))

##     CustomerCode      CountryName      QtyRequired        ITEM_NAME
##                0                0                0                0
##        ShapeName           AreaFt Order_Conversion
##                0                0                0
```

Summary Statistics of Dataset variables - Raw Data Order and Sample & Data on Sample ONLY

```
print(paste("Summary Statustics of Raw Data Order and Sample"))

## [1] "Summary Statustics of Raw Data Order and Sample"

summary(champo_raw)

##      OrderType      OrderCategory   CustomerCode   CountryName
##  Area Wise:14202   Order :13131   CC     :4135   USA    :10624
##  Pc Wise  : 4749   Sample: 5820   M-1    :2498   INDIA  : 4135
##                                   P-5    :1930   UK     : 1692
##                                   A-9    :1395   ITALY  :  596
##                                   JL     :1126   ROMANIA:  456
##                                   C-1    :1097   BELGIUM:  346
##                                   (Other):6770   (Other): 1102
##  Custorderdate                  QtyRequired        TotalArea
##  Min.   :2017-01-16 00:00:00   Min.   :   1.00   Min.   :   0.04
##  1st Qu.:2018-02-27 00:00:00   1st Qu.:   1.00   1st Qu.:   4.00
##  Median :2018-12-01 00:00:00   Median :   4.00   Median :  15.00
##  Mean   :2018-10-18 18:28:42   Mean   :  31.42   Mean   :  36.15
```

```
##  3rd Qu.:2019-07-05 00:00:00   3rd Qu.:  13.00   3rd Qu.:  54.00
##  Max.   :2020-02-14 00:00:00   Max.   :6400.00   Max.   :1024.00
##
##      Amount                ITEM_NAME                        QualityName
##  Min.   :     0.0   HAND TUFTED:7095   TUFTED 60C                  : 1319
##  1st Qu.:     0.0   DURRY      :4355   TUFTED 60C ALL LOOP         :  862
##  Median :   201.0   DOUBLE BACK:2474   TUFTED 60C+VISC 2/16 5PLY  :  840
##  Mean   :  1657.9   HANDWOVEN  :2330   TUFTED 60C LOOP/CUT         :  614
##  3rd Qu.:   979.4   KNOTTED    :1575   D.B. LILEN 2/8+VISCOSE 5PLY:  613
##  Max.   :599719.7   JACQUARD   : 477   D.B. 60C 2PLY+LEFA VISCOSE :  459
##                     (Other)    : 645   (Other)                     :14244
##             DesignName      ColorName        ShapeName          AreaFt
##  PLAIN             :  819   GREY   : 1334   OCTAGON:    2   Min.   :  0.44
44
##  HOMER             :  459   MULTI  : 1254   OVAL   :    1   1st Qu.:  8.43
75
##  TEXTURE LOOP      :  437   BLUE   : 1014   REC    :18514   Median : 35.00
00
##  ELOQ GARDEN [8517]:  350   SILVER :  742   ROUND  :  362   Mean   : 44.47
10
##  MODASA            :  236   BEIGE  :  648   SQUARE :   72   3rd Qu.: 64.73
61
##  DOUBLE DIAMOND    :  225   NAVY   :  580                   Max.   :645.72
22
##  (Other)           :16425   (Other):13379
```

```
print(paste("Summary Statistics of Data on Sample ONLY"))
```

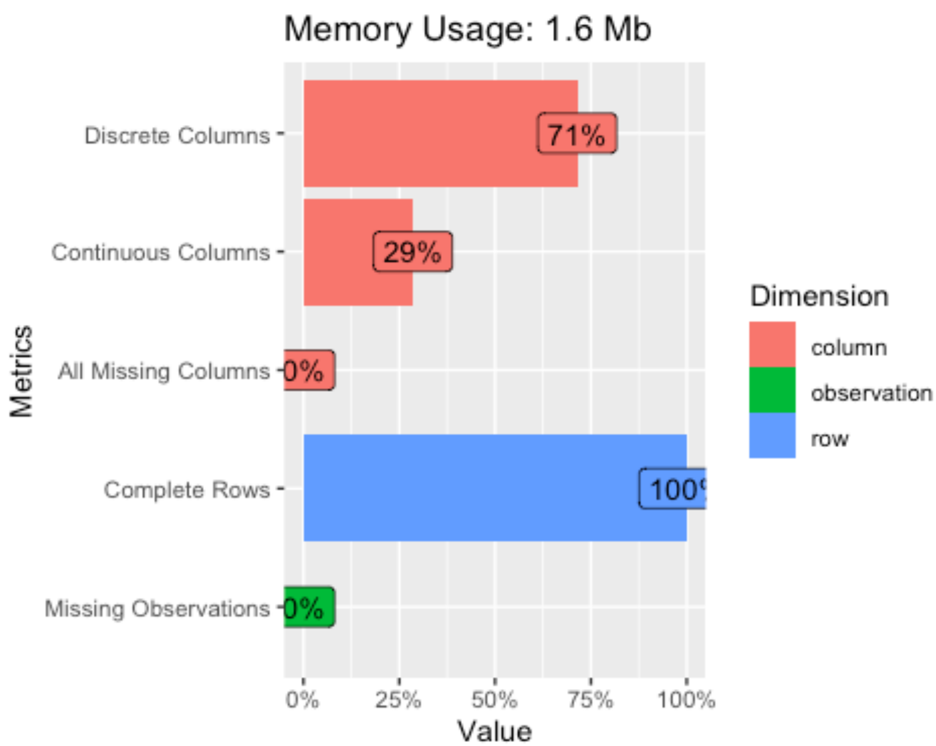
```
## [1] "Summary Statistics of Data on Sample ONLY"
```

```
summary(champo_sample_only)
```

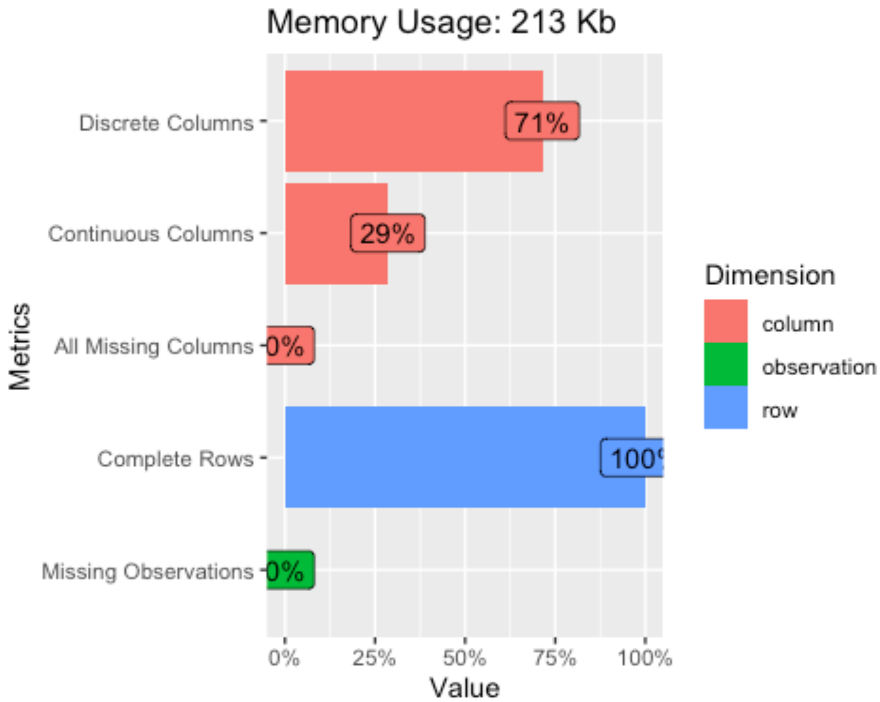
```
##   CustomerCode   CountryName    QtyRequired           ITEM_NAME
##  CC      :3941   INDIA   :3941   Min.   :  1.000   HAND TUFTED:2425
##  N-1     : 232   USA     :1430   1st Qu.:  1.000   DURRY      :1563
##  A-9     : 222   UK      : 203   Median :  1.000   HANDWOVEN  : 705
##  H-2     : 185   BELGIUM: 132   Mean   :  1.975   DOUBLE BACK: 554
##  TGT     : 176   ITALY  :  45   3rd Qu.:  1.000   KNOTTED    : 217
##  T-5     : 148   ROMANIA:  20   Max.   :200.000   HANDLOOM   : 103
##  (Other): 916   (Other):  49                      (Other)    : 253
##   ShapeName        AreaFt          Order_Conversion
##  REC   :5741   Min.   :  0.6667   0:4651
##  ROUND :  57   1st Qu.:  6.0000   1:1169
##  SQUARE:  22   Median : 11.0000
##                Mean   : 21.5558
##                3rd Qu.: 39.8125
##                Max.   :480.0000
##
```

Data Quality Check for Raw Data Order and Sample & Data on Sample ONLY

```
print(paste("Raw Data Order and Sample"))
data_qual <- t(introduce(champo_raw))
colnames(data_qual)<- "Values"
data_qual
plot_intro(champo_raw)
```

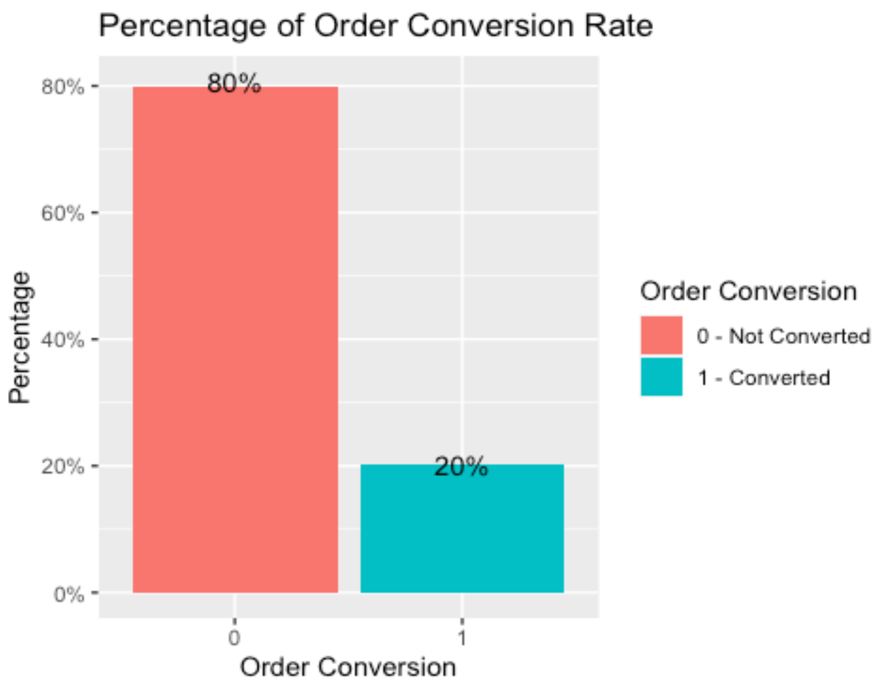



```
print(paste("Data on Sample ONLY"))

## [1] "Data on Sample ONLY"

data_qual <- t(introduce(champo_sample_only))
colnames(data_qual)<- "Values"
data_qual
plot_intro(champo_sample_only)
```



Exploratory Data Analysis ***Computing Proportion of Order Conversion Date***

```
Response <- champo_sample_only %>%
  dplyr::count(Order_Conversion) %>%
  mutate(perc = n / nrow(champo_sample_only))

Response %>%
 ggplot(aes(x=Order_Conversion,y= perc,fill=as.factor(Order_Conversion))) +
 geom_bar(stat="identity") +
 labs(title="Percentage of Order Conversion Rate") +
  xlab("Order Conversion") +
  ylab("Percentage") +
  geom_text(aes(label=scales::percent(perc)),
            position = position_stack(vjust = 1.01))+
  scale_y_continuous(labels = scales::percent)+
  scale_fill_manual(values = c("1" = "green", "0" = "red")) +
  scale_fill_discrete(name="Order Conversion",labels =
                          c("0" = "0 - Not Converted", "1" = "1 - Converted"))
```

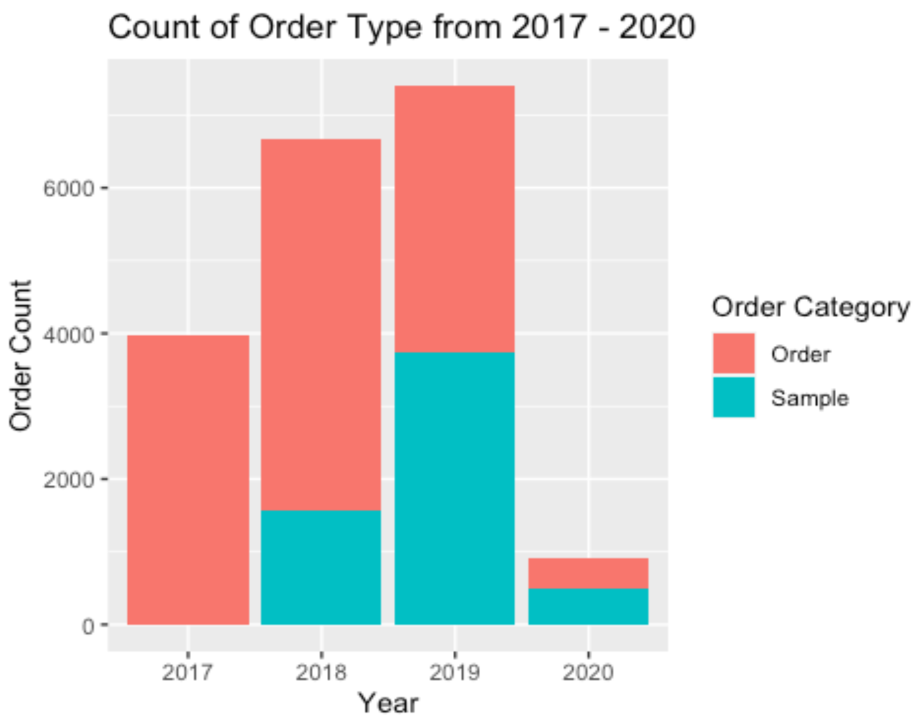


We visualize the outcome of Order Conversion Rate to be biased towards higher proportion of No Conversion Rate. About 80% of the 5820 customers do not revert back as potential customers for Champo Carpets and only 20% of the customer base show potential good conversion rate.Therefore, we can say that the data are unbalanced.

Problem 1

Useful Visualizations for Key Insights Order Count Charts - Computing the Count of Orders across all the years

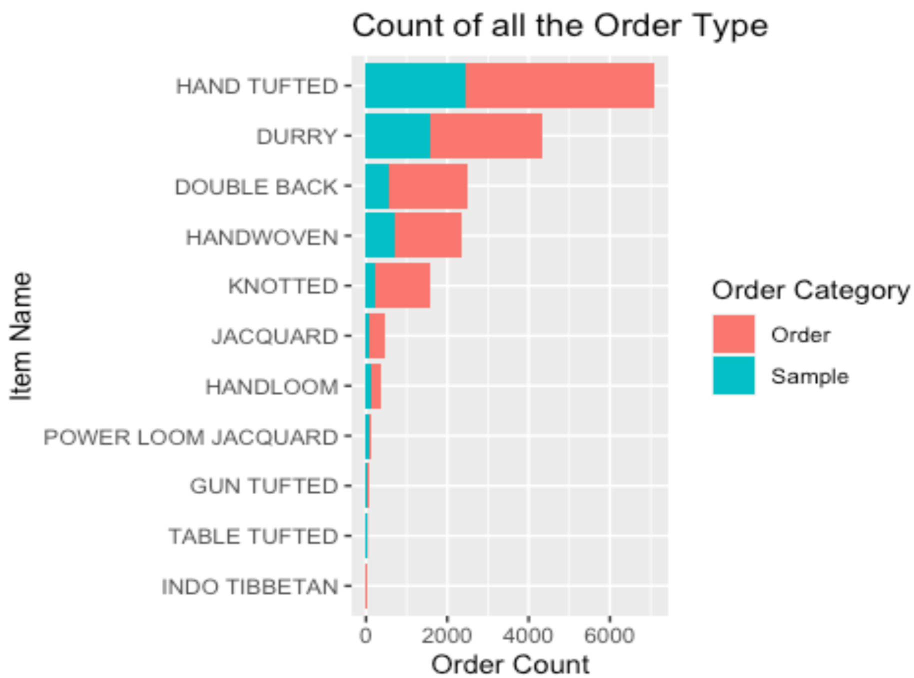
```
champo_raw$Custorderdate <- as.Date(champo_raw$Custorderdate)
champo_raw <- mutate(champo_raw, OrderYear = format(champo_raw$Custorderdate,
format = "%Y"))

champo_raw %>%
  select(OrderYear, OrderCategory) %>%
  group_by(OrderYear, OrderCategory) %>%
  mutate(Order_Count = n()) %>%
  unique() %>%
  ggplot(aes(fill = OrderCategory, x =OrderYear, y = Order_Count)) +
  geom_bar(position = "stack", stat = "identity") +
  ggtitle("Count of Order Type from 2017 - 2020") + labs(fill = "Order Catego
ry")+ xlab("Year") + ylab("Order Count")
```



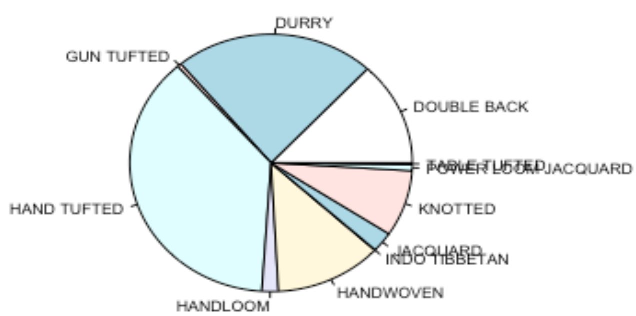
From the bar plot we can infer that there are more orders in the year 2017 and 2018. However, as we move from 2018 to 2020 there is a significant amount of samples. The total order quantity is very less in the year 2020 as compared to previous years.

Order & Sample - Computing the Count of Orders by Item Types

```
champo_raw %>%
  select(ITEM_NAME, OrderCategory) %>%
  group_by(ITEM_NAME, OrderCategory) %>%
  mutate(Order_Count = n()) %>%
  unique() %>%
  ggplot(aes(fill = OrderCategory, x = reorder(ITEM_NAME,Order_Count), y = Or
der_Count)) +
  geom_bar(position = "stack", stat = "identity") +
  ggtitle("Count of all the Order Type") + coord_flip() + labs(fill = "Order
Category") + xlab("Item Name") + ylab("Order Count")
```



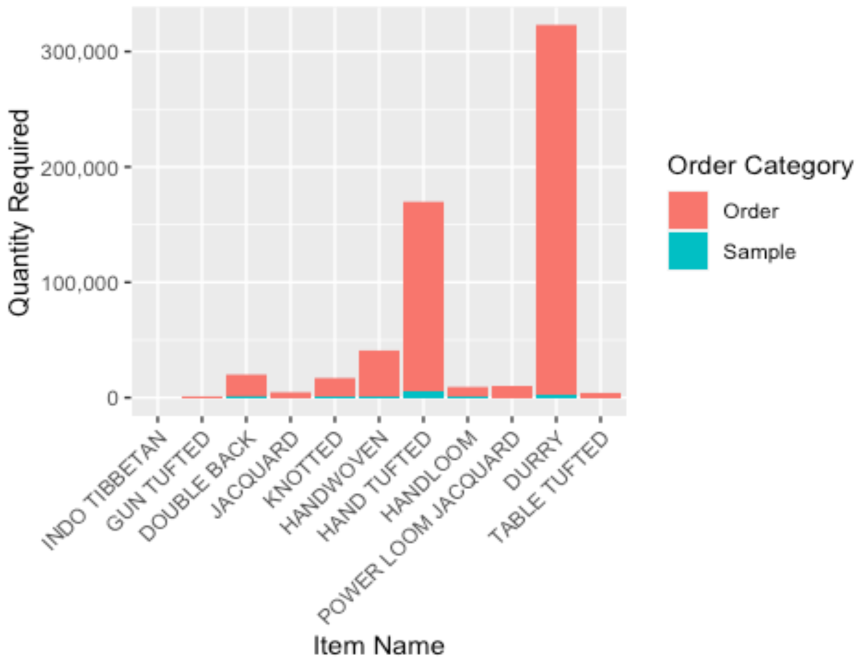

```
#Pie Chart
proportion_table <- table(champo_raw$ITEM_NAME)
pie(proportion_table, cex = 0.55)
```



We can infer that the highest count of Item ordered is of the "Hand Tufted" type, followed by "Durry", "Double Back" and "Handwoven". The least ordered item is of the "Indo Tibetan" type. We observe a significant trend of decrease in both the Order and Sample categories. Only "Power Loom Jacquard" and "Table Tufted" Item types had samples were highere than orders.

Counting the Quantity Required by Item Type

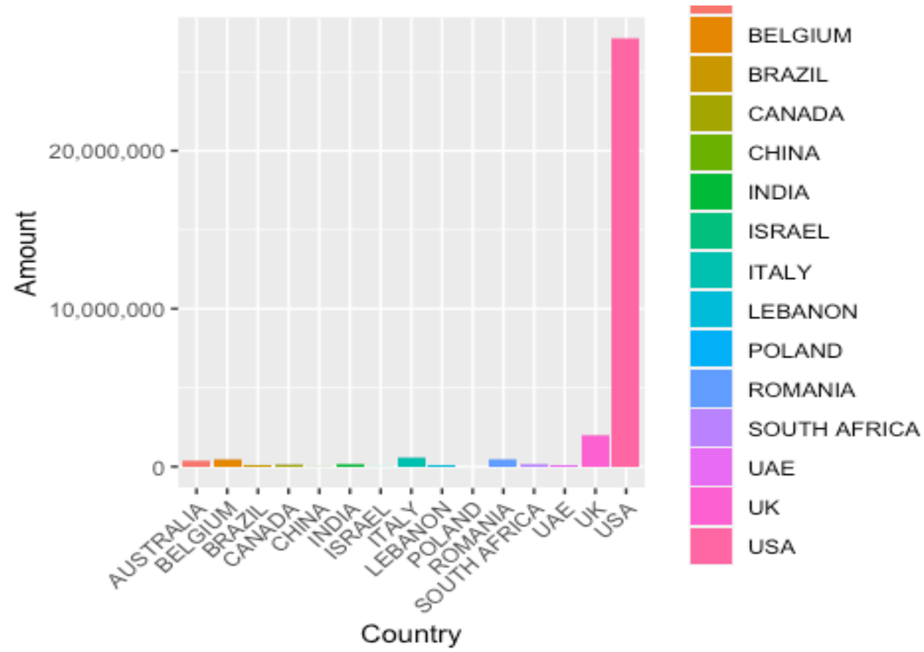
```
champo_raw %>%
  ggplot( aes(x= reorder(ITEM_NAME, QtyRequired),y=QtyRequired, fill= OrderCa
tegory)) +
  geom_bar(stat ="identity") +
  theme(axis.text.x=element_text (angle =45, hjust =1)) + labs(fill = "Order
Category") +
  scale_y_continuous(labels=comma) + xlab("Item Name") + ylab("Quantity Requi
red")
```



We can infer from the bar plot that, highest order quantity is of the "Durry" Item type, followed by "Hand Tufted" and "Handwoven". We observe that there is significantly less demand of the samples as compared to the total required quantity.

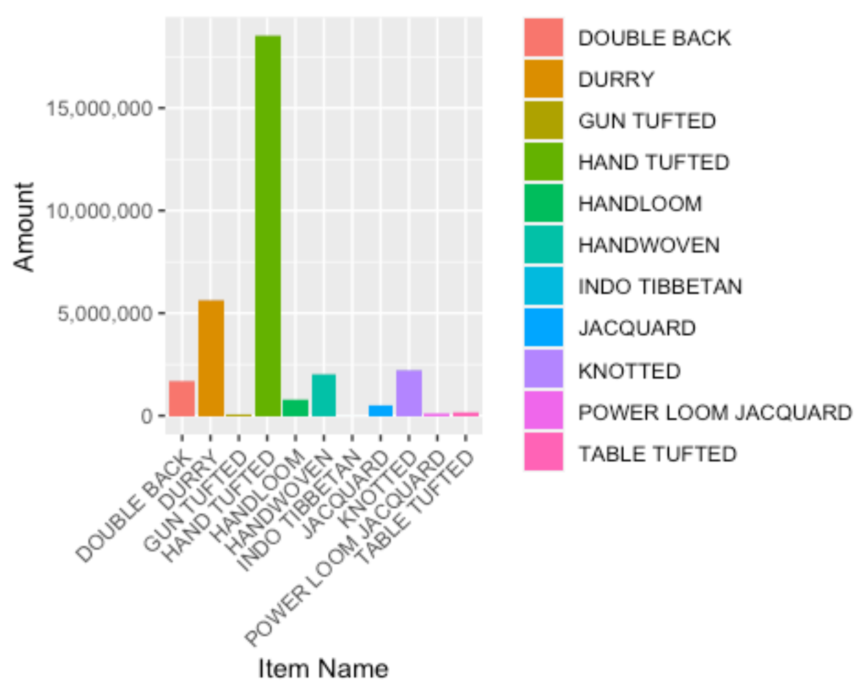
Revenue - Revenue computation by Countries

```
champo_raw %>%
  group_by(CountryName) %>%
  ggplot(aes(x= CountryName , y = Amount, fill= CountryName)) +
  geom_col() +
  theme(axis.text.x=element_text (angle =45, hjust =1)) +
  scale_y_continuous(labels=comma)  + labs(fill = "Country")+ xlab("Country")
+ ylab("Amount")
```



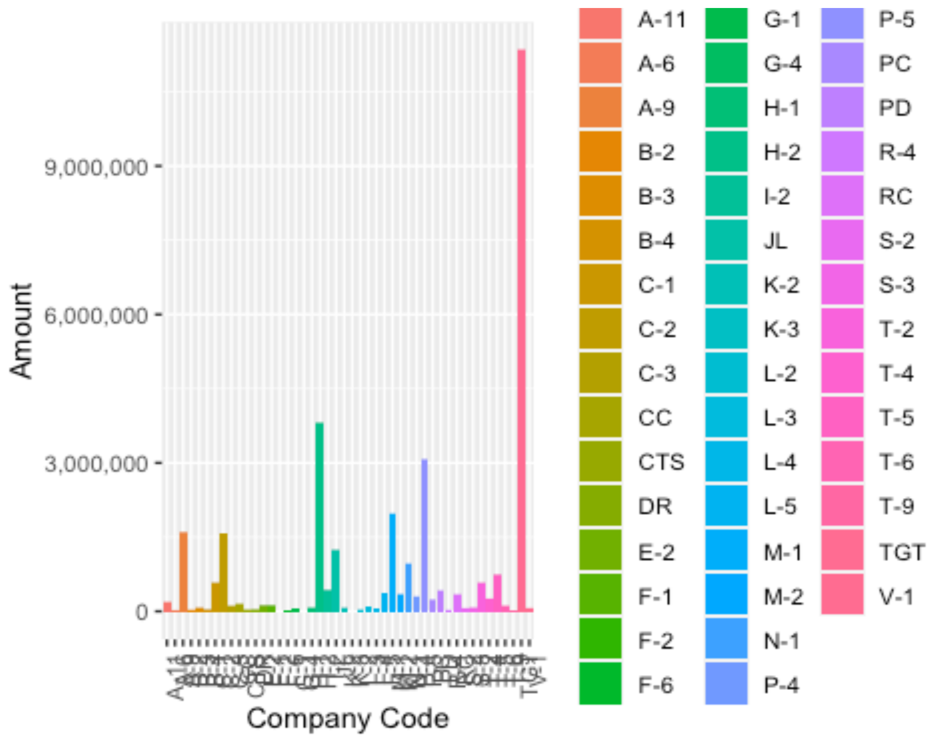
Revenue computation by Item Type

```
champo_raw %>%
  group_by(ITEM_NAME) %>%
  ggplot(aes(x= ITEM_NAME , y = Amount, fill= ITEM_NAME)) +
  geom_col() +
  theme(axis.text.x=element_text (angle =45, hjust =1)) +
  scale_y_continuous(labels=comma)+ labs(fill = "Item Name")+ xlab("Item Name
") + ylab("Amount")
```



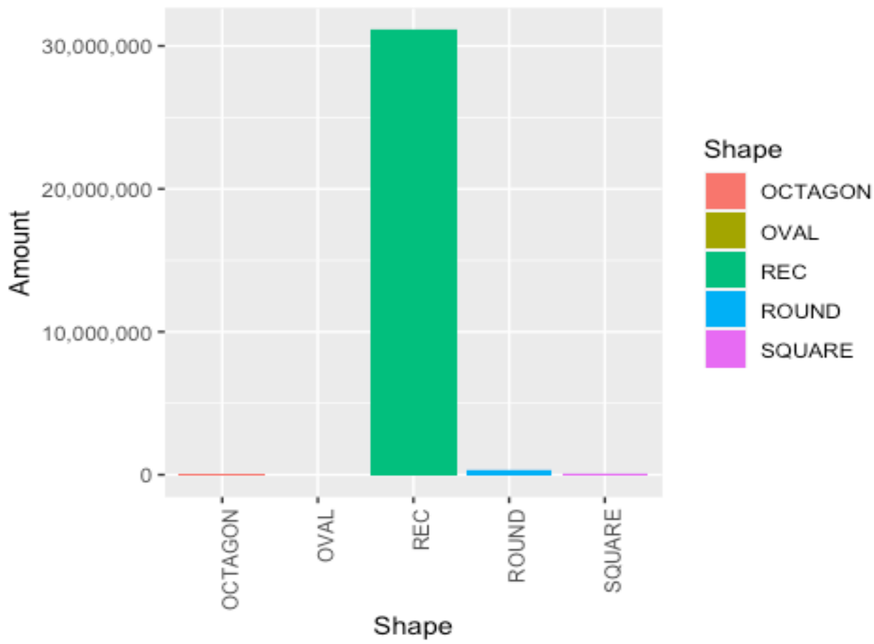
Revenue computation by Companies

```
champo_raw %>%
  group_by(CustomerCode) %>%
  ggplot(aes(x= CustomerCode , y = Amount, fill= CustomerCode)) +
  geom_col() +
  theme(axis.text.x=element_text (angle =90, hjust =1)) +
  scale_y_continuous(labels=comma) + labs(fill = "Country Names")+ labs(fill
= "Company Code")+ xlab("Company Code") + ylab("Amount")
```



Revenue computation by Shape Type

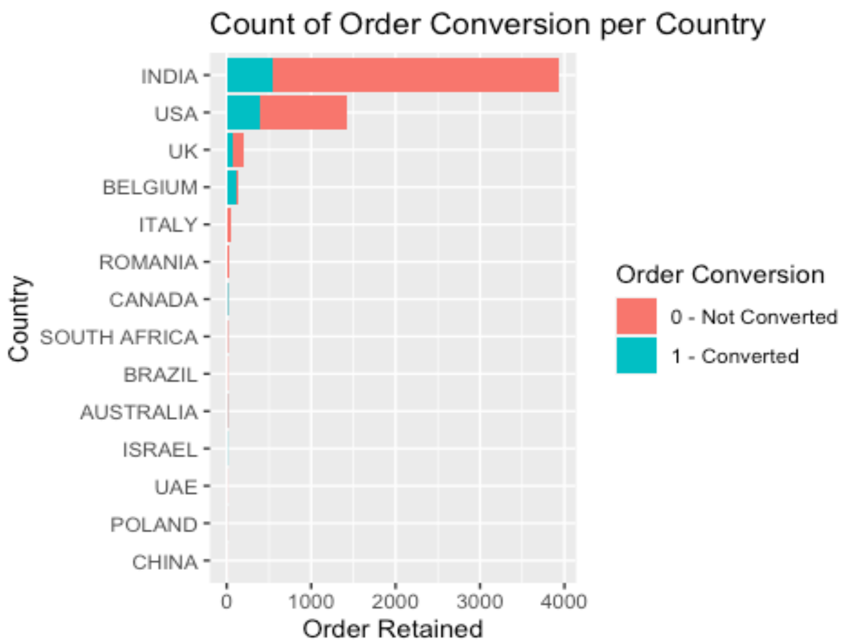
```
champo_raw %>%
  group_by(ShapeName) %>%
  ggplot(aes(x= ShapeName , y = Amount, fill= ShapeName)) +
  geom_col() +
  theme(axis.text.x=element_text (angle =90, hjust =1)) +
  scale_y_continuous(labels=comma) + labs(fill = "Shape") + xlab("Shape") + y
lab("Amount")
```



From the above Bar Plots we can infer that USA generates significantly higher revenue as compared to any other country. Also, the highest revenue generating Item type is “Hand Tufted”, followed by “Durry”, “Knotted” and “Handwoven” item types and the least revenue is obtained by “Gun Tufted” and “Indo Tibetan”. Similarly, a significantly higher revenue is gained by the sales to TGT company, followed by H-2 and P-5. Rectangular shaped carpets gives the highest revenue, followed by round and square carpets respectively.

Order Conversion - Computing Count of Order Conversion per Country

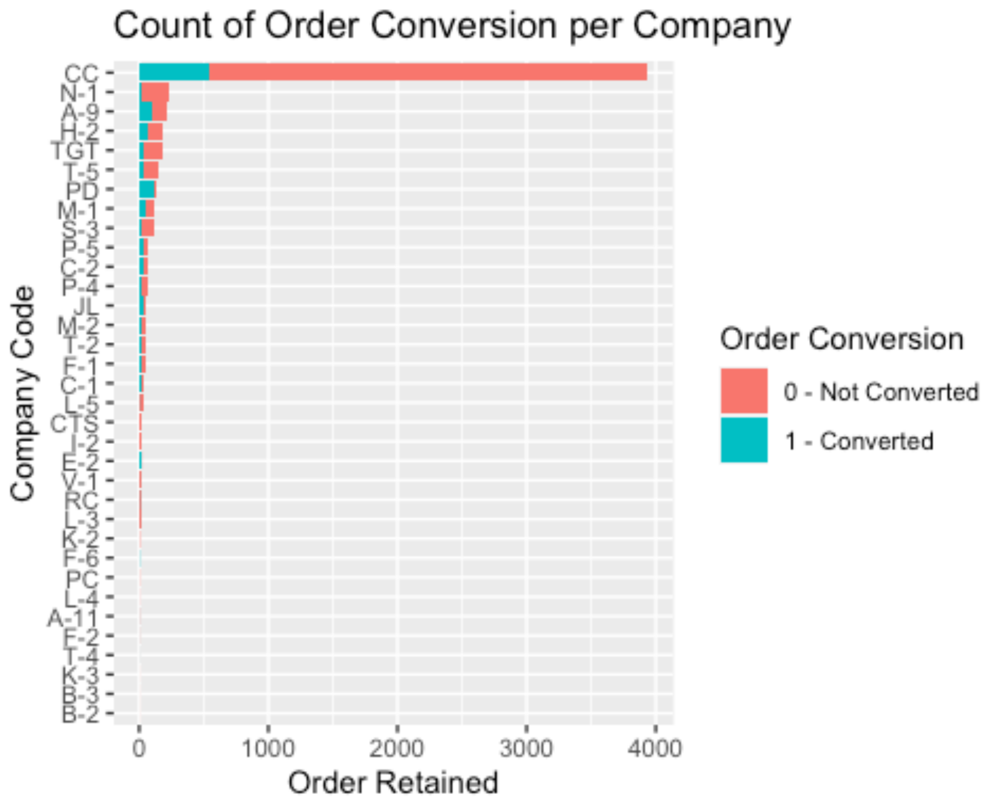
```
champo_sample_only %>%
  select(CountryName, Order_Conversion) %>%
  group_by(CountryName, Order_Conversion) %>%
  mutate(Order_Retained = n()) %>%
  unique() %>%
  ggplot(aes(fill = Order_Conversion, x = reorder(CountryName, Order_Retained
), y = Order_Retained)) +
  geom_bar(position = "stack", stat = "identity") +
  ggtitle("Count of Order Conversion per Country") + coord_flip()+ labs(fill
= "Country Names") +
  scale_fill_discrete(name="Order Conversion",labels = c("0" = "0 - Not Conve
rted", "1" = "1 - Converted"))+ ylab("Order Retained") + xlab("Country")
```



From the bar plot we can infer that, India has the highest order conversion count as compared to all other countries, followed by USA and UK. Belgium an Canada has the highest retain rate as compared to the not converted orders.

Computing Count of Order Conversion per Company

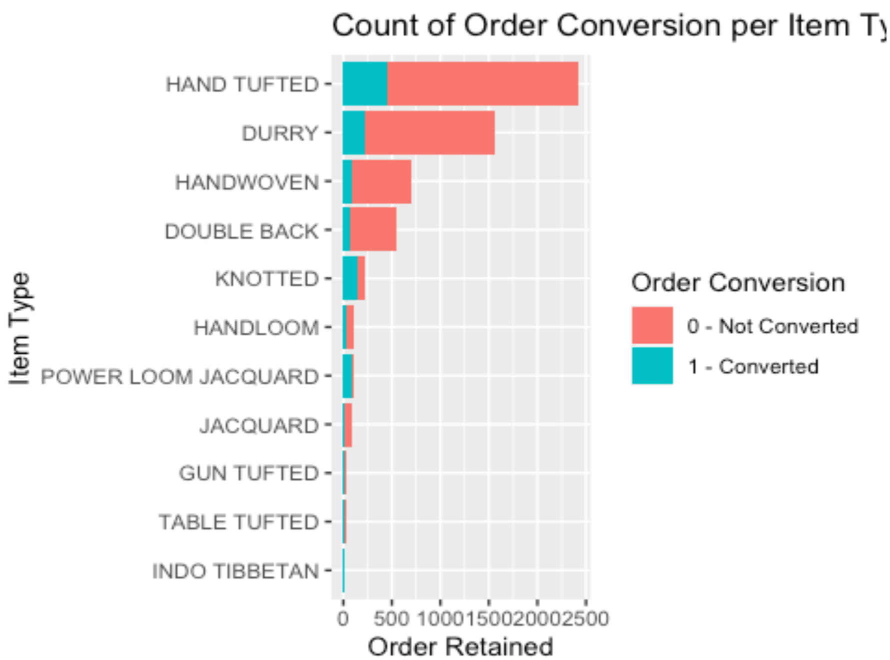
```
champo_sample_only %>%
  select(CustomerCode, Order_Conversion) %>%
  group_by(CustomerCode, Order_Conversion) %>%
  mutate(Order_Retained = n()) %>%
  unique() %>%
  ggplot(aes(fill = Order_Conversion, x = reorder(CustomerCode, Order_Retaine
d), y = Order_Retained)) +
  geom_bar(position = "stack", stat = "identity") +
  ggtitle("Count of Order Conversion per Company") + coord_flip() +
  scale_fill_discrete(name="Order Conversion",labels = c("0" = "0 - Not Conve
rted", "1" = "1 - Converted")) + ylab("Order Retained") + xlab("Company Code"
)
```



Similarly, CC company has a significantly higher order conversion than any other company. However, we can observe that "PD", "JL", "E-2" and "F-6" companies have the highest sample retain rate as compared to the not converted orders.

Computing Count of Order Conversion per Item Type

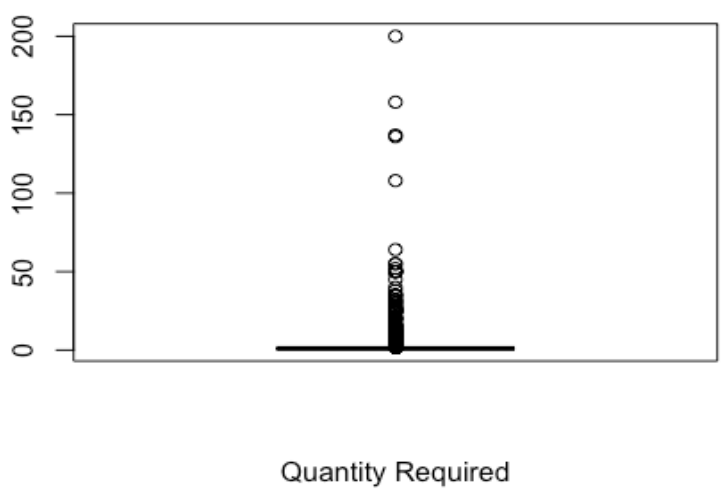
```
champo_sample_only %>%
  select(ITEM_NAME, Order_Conversion) %>%
  group_by(ITEM_NAME, Order_Conversion) %>%
  mutate(Order_Retained = n()) %>%
  unique() %>%
  ggplot(aes(fill = Order_Conversion, x = reorder(ITEM_NAME, Order_Retained),
y = Order_Retained)) +
  geom_bar(position = "stack", stat = "identity") +
  ggtitle("Count of Order Conversion per Item Type") + coord_flip()+
  scale_fill_discrete(name="Order Conversion",labels = c("0" = "0 - Not Conve
rted", "1" = "1 - Converted")) + ylab("Order Retained") + xlab("Item Type")
```



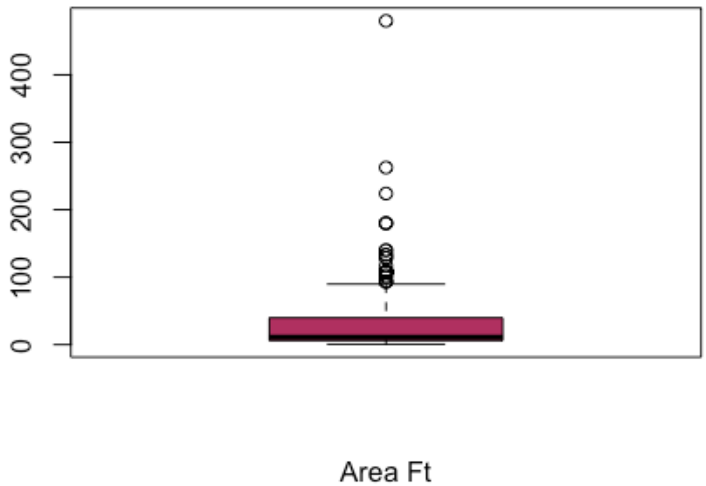
Hand Tufted" has the highest order conversion and is significantly less in terms of sample retain rate, followed by "Durry", "Hand Woven" and "Double Back". Whereas, "Knotted" and "Power Loom Jacquard" Item types have the highest sample retain rates as compared to other item types.

Analysis of Continuous Variables in Data on Sample ONLY sheet

```
boxplot(champo_sample_only$QtyRequired, col = "maroon", xlab="Quantity Requir
ed")
```

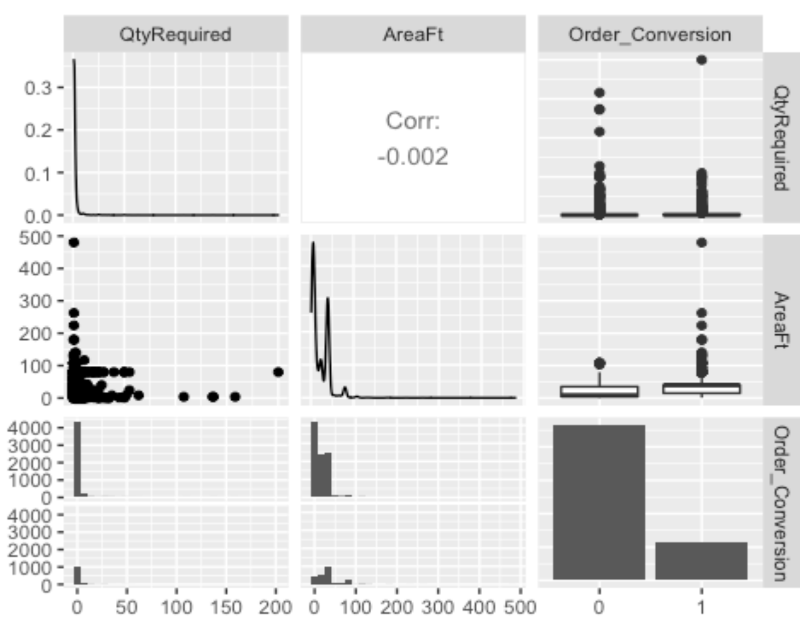



```
boxplot(champo_sample_only$AreaFt, col = "maroon", xlab="Area Ft")
```



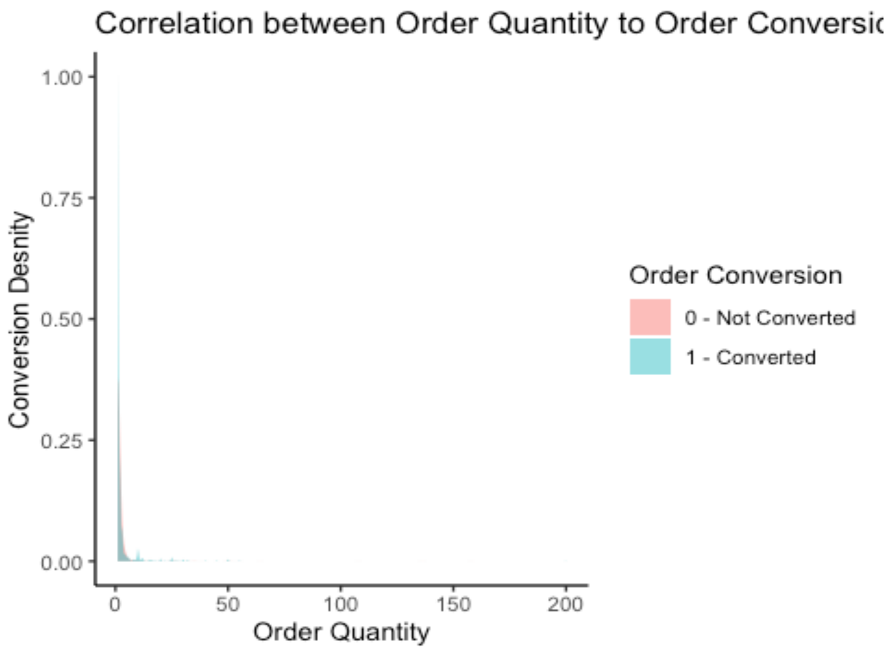
Correlation of Continuous Variables in Data on Sample ONLY sheet

```
library(GGally)
corr <- champo_sample_only %>%
  select(QtyRequired,AreaFt,Order_Conversion)
ggpairs(corr)
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



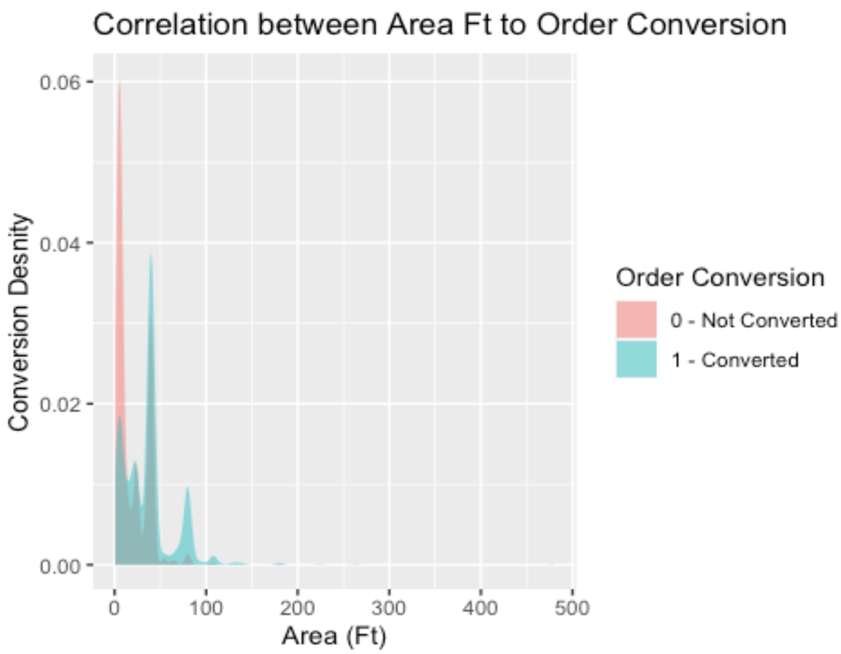
Computing the correlation between numerical variable to target variable

```
#By Order Quantity
ggplot(champo_sample_only)+
  geom_density(aes(x=QtyRequired,fill=`Order_Conversion`),alpha=0.5, color =
NA) + theme_classic() +
 scale_fill_discrete(name="Order Conversion",labels = c("0" = "0 - Not Conver
ted", "1" = "1 - Converted"))+ ggtitle("Correlation between Order Quantity to
Order Conversion") + xlab("Order Quantity") + ylab("Conversion Desnity")
```



From the density plot we can infer that there is a higher density of Converted and Not converted orders when the required quantity is within 10. However, as we increase the Order Quantity, there are negligible converted and not converted orders.

```
#By AreaFt
ggplot(champo_sample_only)+
  geom_density(aes(x=AreaFt,fill=`Order_Conversion`),alpha=0.5, color = NA) +
  scale_fill_discrete(name="Order Conversion",labels = c("0" = "0 - Not Conve
rted", "1" = "1 - Converted")) + ggtitle("Correlation between Area Ft to Orde
r Conversion") + xlab("Area (Ft)") + ylab("Conversion Desnity")
```



From the density plot we can infer that there is a peak in the Not Converted orders within 50ft area. However, as we increase the area of carpets the order conversion starts decreasing moderately with some highs and lows, and eventually become negligible after 150ft area.

Analysis of Categorical Variables in Data on Sample ONLY sheet

```
chisq.test(champo_sample_only$CustomerCode, champo_sample_only$Order_Conversi
on, correct=FALSE)
## X-squared = 934.19, df = 33, p-value < 2.2e-16
chisq.test(champo_sample_only$CountryName, champo_sample_only$Order_Conversio
n, correct=FALSE)
## X-squared = 671.46, df = 13, p-value < 2.2e-16
chisq.test(champo_sample_only$ITEM_NAME, champo_sample_only$Order_Conversion,
correct=FALSE)
## X-squared = 679.04, df = 10, p-value < 2.2e-16
chisq.test(champo_sample_only$ShapeName, champo_sample_only$Order_Conversion,
correct=FALSE)

## X-squared = 9.4222, df = 2, p-value = 0.008995
```

Problem 2

Machine Learning Algorithms consist of Supervised Learning Algorithms and Unsupervised Learning Algorithms. Within Supervised Learning Algorithms, we have Classification & Regression and under Unsupervised Learning Algorithms, we have Clustering and Association methods.

Considering the Champo Carpets Case Study, the ultimate goal is to identify two constraints - the most important customers and the most important products and eventually find a way to connect the two using suitable attributes and analytical models, thereby helping Champo Carpets increase the conversion rate.

In order to determine the most important products, we can use Classification and Regression algorithms such as Decision Tree, Random Forest and Logistic Regression respectively.

Logistic Regression - Post model building, we can analyze variable/attribute importance using the varImp(logitModel, scale = FALSE) method. This will provide the significant variables with their corresponding importance score with respect to the binary target variable. Moreover, significance can also be determined through the hypothesis testing of p and alpha values. In the Champo Carpets case, this method will help in identifying the most important attributes that can be likely considered for analyzing the potential conversion rate and further perform deeper analysis on those attributes for future forecasting.

Also, using the important attributes of the model, the Akaike Information Criteria (AIC) can be used to compare and fit several regression models. The important attributes can be divided into predictor variables in say, 4 models, input the models into a list, use the aictab() function. The model with the lowest AIC value can be considered as the best fitting model and the corresponding attributes in that model can be chosen as the final highest weighted attributes that contribute towards order conversion.

For future forecasting, Champo Carpets can use the attributes identified using the method mentioned above to analyze their customer base and take actions accordingly.

Decision Tree - In general, Decision Trees lay out all possible outcomes and solutions. In Champo Carpets case, decision trees generated can greatly help in downsizing or expanding certain carpet ITEMS in specific Country. It will also help in changing pricing models for different product offerings, deciding which ITEM is prone to converge more customers etc.

With the tree generated, Champo Carpets can determine the likelihood or percentage of conversion through branching of the most important attributes.

Decision Trees can also be used to identify the important attributes that determine the conversion of samples sent to the customers. The print(tree_model$variable.importance) syntax of decision tree will help identify the attributes that majorly contribute or affect the target variable, Order_Conversion.

Decision trees do not provide the answer to the problem Champo carpets are facing but it will definitely help the management determine which alternative will yield the greatest conversion rate, given a particular choice point. In other words, trees yield an expected value based on which Champo Carpets can base their decision

Random Forest - The Random Forest model is the easiest algorithm to determine feature importance or contribution to the mode. The functions importance() and varImpPlot() are few ways to evaluate feature importance. The importance(rf, type = 1) method measures feature importance using MeanDecreaseAccuracy and importance(rf, type = 2) using MeanDecreaseGini. The MeanDecreaseGini is generally used to measure how much the model's accuracy decreases when a given variable is excluded or in other words, ut measures how much each attribute contributes to homogeneity of nodes and leaves. The higher the value, higher the importance of the attribute in the model. The MeanDecreaseAccuracy is used to determine the average decrease in accuracy by randomly permutating the feature values in OOB sample. The more the accuracy is rugged, the more important that attribute is for classification, or in other words, higher the value, higher the importance of attribute.

Considering this conceptual definition for the Champo Carpets case, Random Forests models can be developed to identify the most important attributes responsible for conversion rate.

In order to determine the most important customers, we can use Clustering algorithms such as K means and hierarchical Clustering and Neural Networks.

K means and Hierarchical Clustering - Clustering is based on the idea of grouping identical data points into groups or clusters based on similarity. Champo Carpets can perform customer segmentation based on demographic, geographical, psychographical and behavioral data from the clusters generated by the K means model as customer segmentation is a powerful metric to identify unsatisfied customers. Using this, Champo can profile the clusters (using profiling techniques) to better understand their customers and describe them using the cluster analysis variables.

Neural Networks - Neural Network algorithms are designed to recognize patterns in data, cluster and classify them for a final outcome. Champo Carpets can use Neural Networks to help forecast by extracting unseen features and defining relationships through modelling. Champo carpets can generate recommendations through the neural network model on the basis of customer behaviors. By customer behavior, we mean to say, monitor the customer base through customer preferences that demonstrate a positive conversion and determine the characteristics or attribute value corresponding to the same and model the network to choose similar customers for future conversion purposes. For example, training the model to find customers based on similarity of Country or Customer Code who are likely to convert and use this for forecasting future trends of customer preferences.

CHOICE OF METRIC - Considering the Champo Carpets Case Study, the True Positive, False Positive and False Negative are

True Positive - Number of customers who converted the samples sent to them as orders and contributed toards conversion rate False Positive - The number of customers who did not converted the samples sent to them as ordered but were predicted to contribute towards conversion rate False Negative - The number of customers who converted the samples sent to them as ordered and were predicted to not contribute towards conversion rate

We know that, Precision = TP/(TP+FP) and Recall = TP/(TP+FN)

We want Champo Carpetsto reduce FN and thereby increase Recall as predicting converted customers and not converted can lead to a huge loss for Champo Carpets when compared to Precision. Therefore, Recall is recommended to be chosen as the metric.

Problem 3

Balancing Data

Based on our visualizations, a major key insight is that the data is unbalanced. We have a higher percentage of the customer base who will not contribute towards the order conversion rate when compared to the positive outcome. A biased dataset will usually not represent the data/models use case accurately resulting in skewed outcomes, low accuracy levels, and analytical errors. Therefore, before moving onto developing our classification models, we will attempt to remove the biased characteristic of the Data by using balancing algorithms.

```
print("Before Balancing")

## [1] "Before Balancing"

summary(champo_sample_only$Order_Conversion)

##    0    1
## 4651 1169

balanced.data <- ovun.sample(Order_Conversion ~., data = champo_sample_only,
method = "over", N=8000)$data
print("After Balancing")

## [1] "After Balancing"

summary(balanced.data$Order_Conversion)

##    0    1
## 4651 3349
```

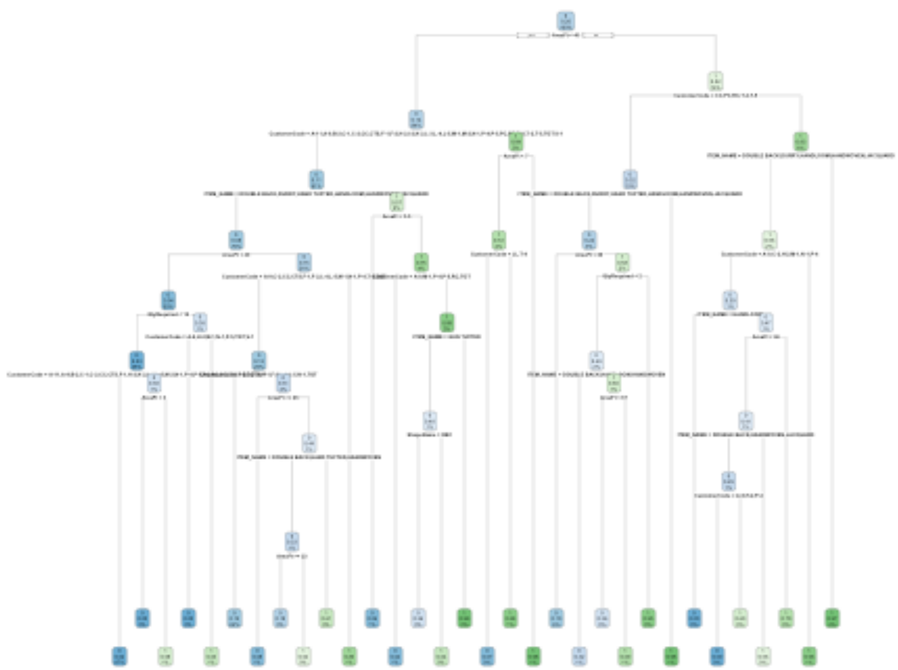
We will run all our models on both Balanced and Unbalanced Data. However, we will use our Balanced data to identify features that contribute towards conversion.

Model 1 - Decision Tree

Unbalanced Data

```
data_70_30_split <- champo_sample_only
set.seed(1346)

indx <- sample(2, nrow(data_70_30_split), replace= TRUE, prob = c(0.7, 0.3))

train <- data_70_30_split[indx == 1, ]
test <- data_70_30_split[indx == 2, ]
trainX <- train[-7]
testX <- test[-7]
```

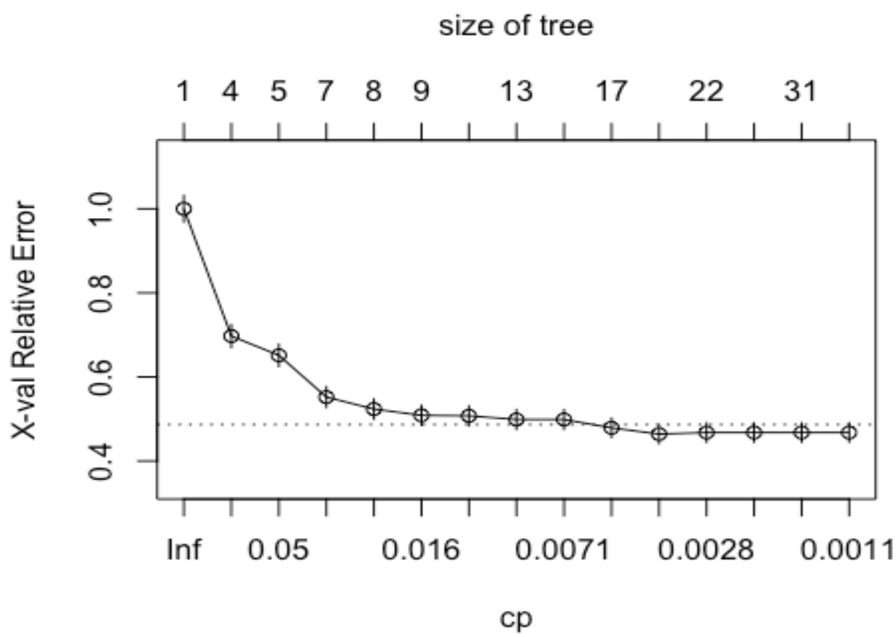
```
#tree_model <- rpart(response ~ ., train)
tree_model_unbalanced <- rpart(Order_Conversion ~ ., train, method = "class",
control = rpart.control(minsplit=20, minbucket=10, cp=0.001))
rpart.plot(tree_model_unbalanced)

## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```
#Depth of tree
nleaves <- length(unique(tree_model_unbalanced$where))
print(nleaves)

## [1] 32

#TRAIN DATA
#Determining Accuracy
train_preds <- predict(tree_model_unbalanced, trainX, type = "class")
train_confusionmatrix <- table(train_preds, train$Order_Conversion)
train_accuracy <- sum(diag(train_confusionmatrix))/sum(train_confusionmatrix)
print(train_confusionmatrix)

##
## train_preds    0    1
##           0 3181  218
##           1   89  588

print(paste("Training accuracy is ", round(train_accuracy,3), sep = ""))
```

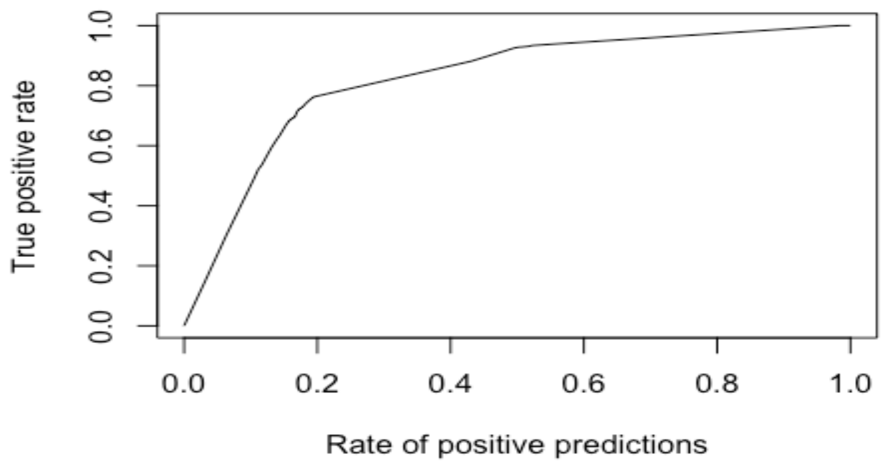
```
## [1] "Training accuracy is 0.925"

#Determining Recall
train_recall <- train_confusionmatrix[2,2]/(train_confusionmatrix[2,1] + trai
n_confusionmatrix[2,2])
print(paste("Training recall is ", round(train_recall,3), sep = ""))

## [1] "Training recall is 0.869"

#Determining Precision
train_precision <- train_confusionmatrix[2,2]/(train_confusionmatrix[1,2] + t
rain_confusionmatrix[2,2])
print(paste("Training precision is ", round(train_precision,3), sep = ""))

## [1] "Training precision is 0.73"

#TEST DATA
#Determining Accuracy
test_preds <- predict(tree_model_unbalanced, testX, type = "class")
test_confusionmatrix <- table(test_preds, test$Order_Conversion)
test_accuracy <- sum(diag(test_confusionmatrix))/sum(test_confusionmatrix)
print(test_confusionmatrix)

##
## test_preds    0    1
##          0 1336   99
##          1   45  264

print(paste("Test accuracy is ", round(test_accuracy,3), sep = ""))

## [1] "Test accuracy is 0.917"

#Determining Recall
test_recall <- test_confusionmatrix[2,2]/(test_confusionmatrix[2,1] + test_co
nfusionmatrix[2,2])
print(paste("Test recall is ", round(test_recall,3), sep = ""))

## [1] "Test recall is 0.854"

#Determining Precision
test_precision <- test_confusionmatrix[2,2]/(test_confusionmatrix[1,2] + test
_confusionmatrix[2,2])
print(paste("Test precision is ", round(test_precision,3), sep = ""))

## [1] "Test precision is 0.727"

#ERROR
#Determining Error of Train set
tree_pred_class <- predict(tree_model_unbalanced, train, type = "class")
trainerror <- mean(tree_pred_class != train$Order_Conversion)
print(paste("Training Error is ", round(trainerror,3), sep = ""))

## [1] "Training Error is 0.075"
```

```
#Determining Error of Test set
tree_pred_test <- predict(tree_model_unbalanced, test, type = "class")
testerror <- mean(tree_pred_test != test$Order_Conversion)
print(paste("Test Error is ", round(testerror,3), sep = ""))

## [1] "Test Error is 0.083"

#Determining best Cp value
plotcp(tree_model_unbalanced)
```

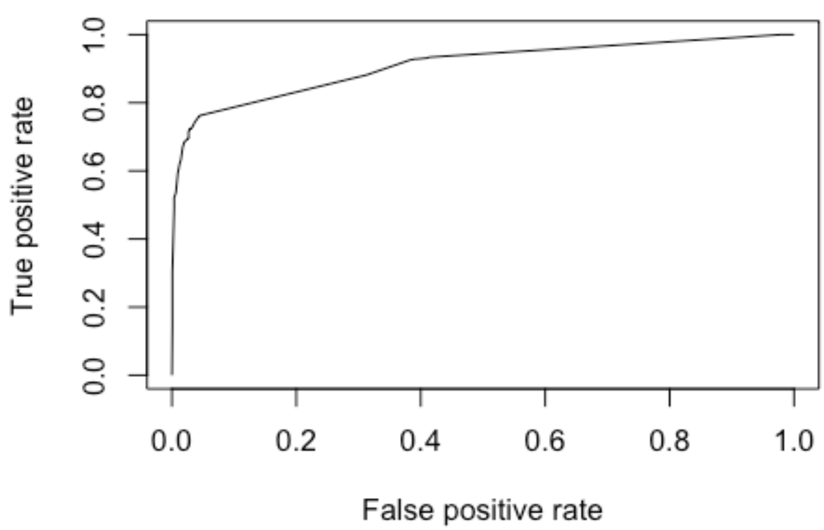



```
printcp(tree_model_unbalanced)
##            CP nsplit rel error  xerror     xstd
## 1  0.1033912      0   1.00000 1.00000 0.031549
## 2  0.0533499      3   0.68983 0.69727 0.027310
## 3  0.0465261      4   0.63648 0.65136 0.026534
## 4  0.0297767      6   0.54342 0.55211 0.024703
## 5  0.0210918      7   0.51365 0.52357 0.024132
## 6  0.0124069      8   0.49256 0.50868 0.023825
## 7  0.0080645     10   0.46774 0.50744 0.023799
## 8  0.0074442     12   0.45161 0.49876 0.023617
## 9  0.0068238     14   0.43672 0.49876 0.023617
## 10 0.0049628     16   0.42308 0.47891 0.023193
## 11 0.0031017     19   0.40819 0.46402 0.022867
## 12 0.0024814     21   0.40199 0.46774 0.022949
## 13 0.0016543     27   0.38710 0.46774 0.022949
## 14 0.0012407     30   0.38213 0.46774 0.022949
## 15 0.0010000     31   0.38089 0.46774 0.022949

#Evaluation Charts
pred_test <- predict(tree_model_unbalanced, newdata = test, type = "prob")
pred <- prediction(pred_test[, 2], test$Order_Conversion)
```

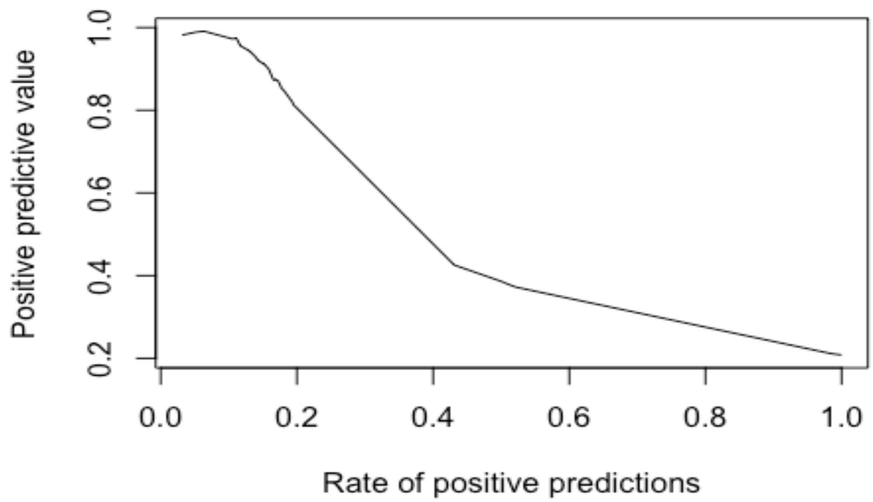
```r
#Gain Chart
perf <- performance(pred, "tpr", "rpp")
plot(perf)
```



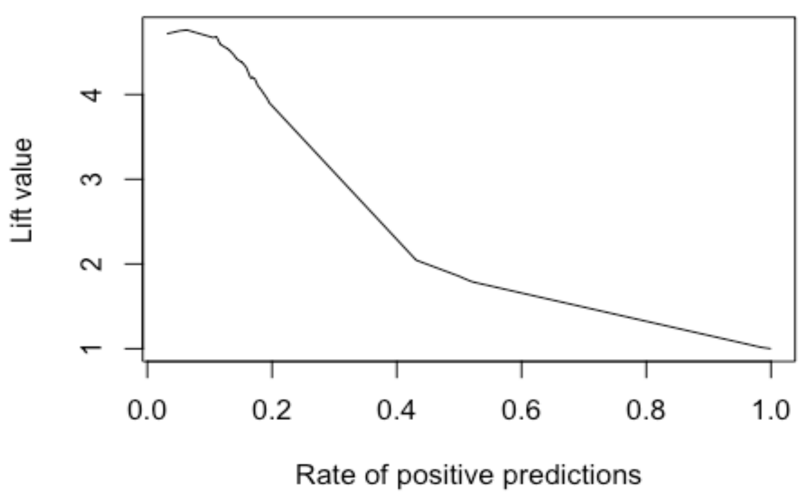

```r
#ROC Curve
perf <- performance(pred, "tpr", "fpr")
plot(perf)
```




```r
# Response Chart
perf <- performance(pred, "ppv", "rpp")
plot(perf)
```




```
# Lift Chart
perf <- performance(pred, "lift", "rpp")
plot(perf)
```



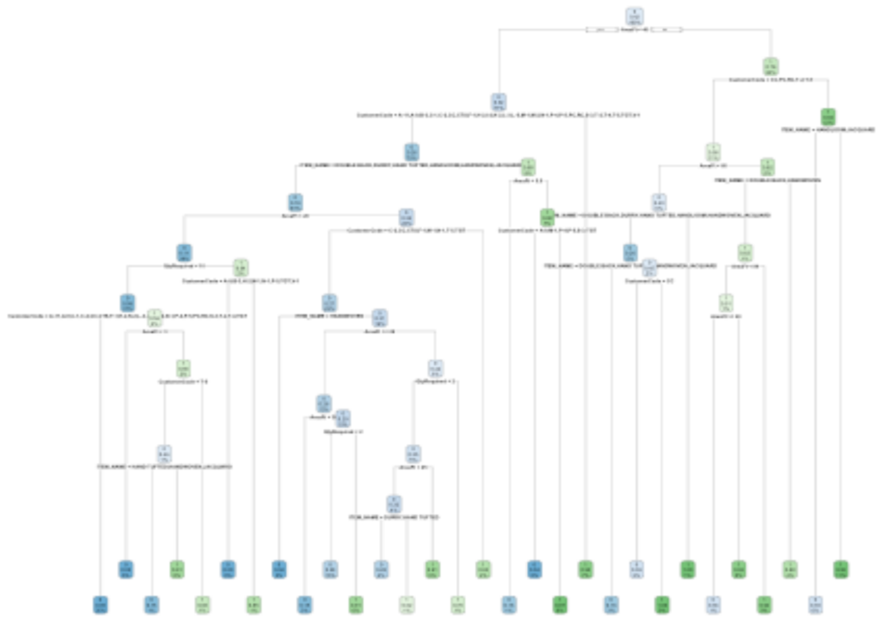

```
#Area Under Curve
auc <- unlist(slot(performance(pred, "auc"), "y.values"))
print(paste("The Area Under the Curve is ", auc))

## [1] "The Area Under the Curve is  0.908966233994211"

#CROSS VALIDATION
dt_cross <- data_70_30_split[sample(nrow(data_70_30_split)),]
```

```
k <- 5
nmethod <- 1
folds <- cut(seq(1,nrow(dt_cross)), breaks = k, labels = FALSE)
model.err <- matrix(-1, k, nmethod, dimnames = list(paste0("Fold", 1:k), c("D
ecision Tree Model")))

for (i in 1:k)
{
  testindexes <- which(folds == i, arr.ind = TRUE)
  test <- dt_cross[testindexes,]
  train <- dt_cross[-testindexes,]

tree_model <- rpart(Order_Conversion ~ ., train, method = "class", control =
rpart.control(minsplit=20, minbucket=10, cp= 0.001))
  predict_treemodel <- predict(tree_model, test, type = "class")
  model.err[i] <- mean(test$Order_Conversion!= predict_treemodel)
}
print(paste("The CV Error rate of Decision Tree after Cross Validation is ",r
ound(mean(model.err),3), sep = ""))

## [1] "The CV Error rate of Decision Tree after Cross Validation is 0.084"
```

For the Decision Tree of Unbalanced data, we chose the final minsplit and minbucket values to be 20 and 10 respectively as these were the values in which the model achieved highest accuracy and precision when a decision tree with cp = -1, minsplit = 0 and minbucket = 0 was constructed. From the cp plot, we found the lowest dip of cp value below the horizontal which was around 0.001.This is often a good indicator of the cp value. We also verified the cp value corresponding to the lowest dip in xerror value and obtained the corresponding cp value. Both the analysis fetched us a cp value of 0.001

Balanced Data

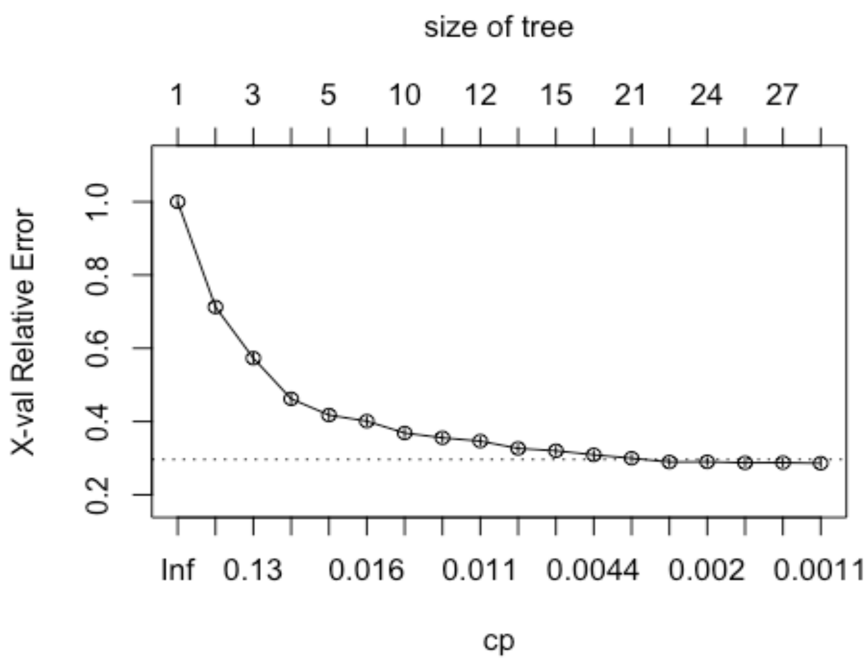
```
data_70_30_split <- balanced.data
set.seed(1346)

indx <- sample(2, nrow(data_70_30_split), replace= TRUE, prob = c(0.7, 0.3))

train <- data_70_30_split[indx == 1, ]
test <- data_70_30_split[indx == 2, ]
trainX <- train[-7]
testX <- test[-7]

#tree_model <- rpart(response ~ ., train)
tree_model_balanced <- rpart(Order_Conversion ~ ., train, method = "class", c
ontrol = rpart.control(minsplit=20, minbucket=10, cp=0.001))
rpart.plot(tree_model_balanced)

## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



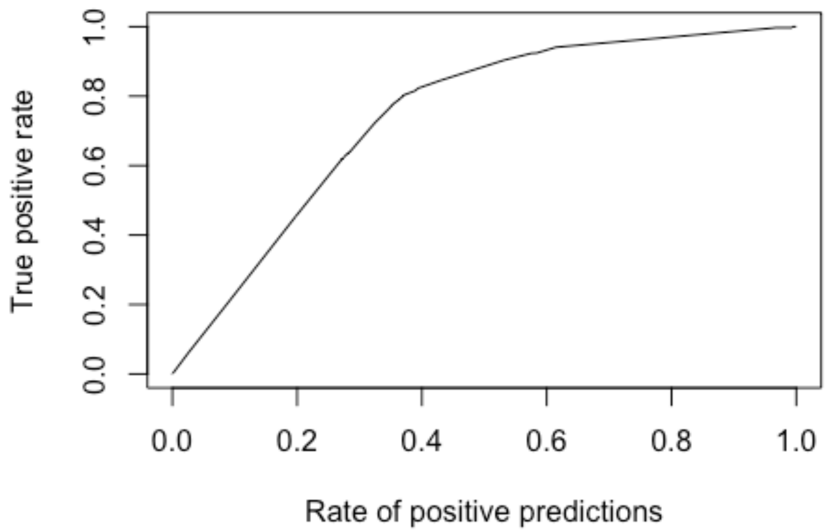
```
#Depth of tree
nleaves <- length(unique(tree_model_balanced$where))
print(nleaves)

## [1] 30

#TRAIN DATA
#Determining Accuracy
train_preds <- predict(tree_model_balanced, trainX, type = "class")
train_confusionmatrix <- table(train_preds, train$Order_Conversion)
train_accuracy <- sum(diag(train_confusionmatrix))/sum(train_confusionmatrix)
print(train_confusionmatrix)

##
## train_preds    0    1
##           0 3108  447
##           1  162 1902

print(paste("Training accuracy is ", round(train_accuracy,3), sep = ""))

## [1] "Training accuracy is 0.892"

#Determining Recall
train_recall <- train_confusionmatrix[2,2]/(train_confusionmatrix[2,1] + trai
n_confusionmatrix[2,2])
print(paste("Training recall is ", round(train_recall,3), sep = ""))

## [1] "Training recall is 0.922"
```

```
#Determining Precision
train_precision <- train_confusionmatrix[2,2]/(train_confusionmatrix[1,2] + t
rain_confusionmatrix[2,2])
print(paste("Training precision is ", round(train_precision,3), sep = ""))

## [1] "Training precision is 0.81"

#TEST DATA
#Determining Accuracy
test_preds <- predict(tree_model_balanced, testX, type = "class")
test_confusionmatrix <- table(test_preds, test$Order_Conversion)
test_accuracy <- sum(diag(test_confusionmatrix))/sum(test_confusionmatrix)
print(test_confusionmatrix)

##
## test_preds    0    1
##          0 1301  197
##          1   80  803

print(paste("Test accuracy is ", round(test_accuracy,3), sep = ""))

## [1] "Test accuracy is 0.884"

#Determining Recall
test_recall <- test_confusionmatrix[2,2]/(test_confusionmatrix[2,1] + test_co
nfusionmatrix[2,2])
print(paste("Test recall is ", round(test_recall,3), sep = ""))

## [1] "Test recall is 0.909"

#Determining Precision
test_precision <- test_confusionmatrix[2,2]/(test_confusionmatrix[1,2] + test
_confusionmatrix[2,2])
print(paste("Test precision is ", round(test_precision,3), sep = ""))

## [1] "Test precision is 0.803"

#ERROR
#Determining Error of Train set
tree_pred_class <- predict(tree_model_balanced, train, type = "class")
trainerror <- mean(tree_pred_class != train$Order_Conversion)
print(paste("Training Error is ", round(trainerror,3), sep = ""))

## [1] "Training Error is 0.108"

#Determining Error of Test set
tree_pred_test <- predict(tree_model_balanced, test, type = "class")
testerror <- mean(tree_pred_test != test$Order_Conversion)
print(paste("Test Error is ", round(testerror,3), sep = ""))

## [1] "Test Error is 0.116"
```

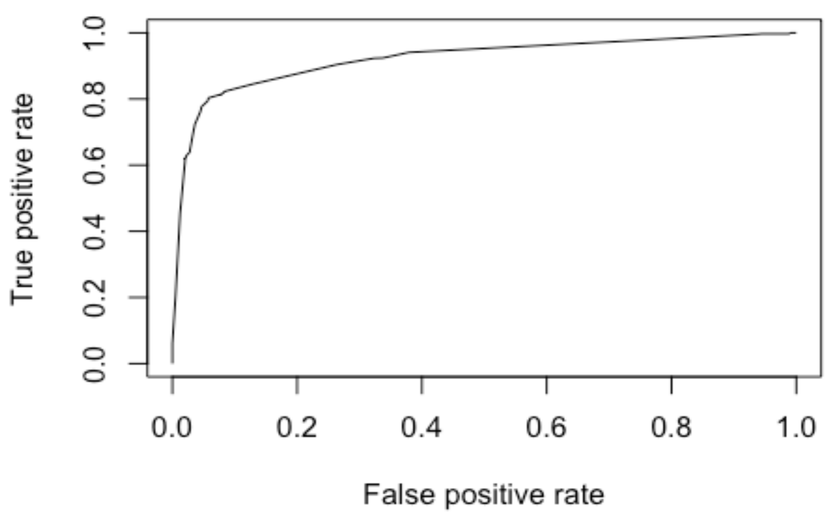
```
#Determining best Cp value
plotcp(tree_model_balanced)
```



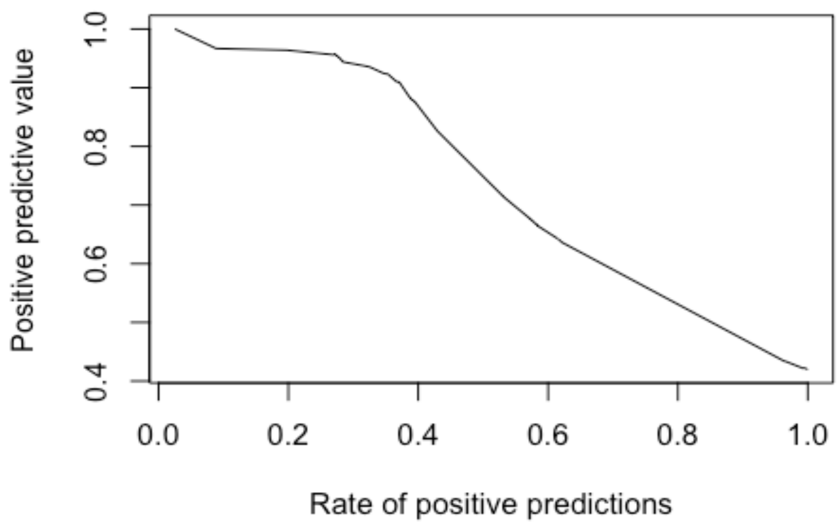

```
printcp(tree_model_balanced)
##           CP nsplit rel error  xerror     xstd
## 1  0.2873563      0   1.00000 1.00000 0.015740
## 2  0.1426139      1   0.71264 0.71264 0.014594
## 3  0.1111111      2   0.57003 0.57301 0.013620
## 4  0.0170285      3   0.45892 0.46190 0.012596
## 5  0.0163900      4   0.44189 0.41762 0.012114
## 6  0.0151128      6   0.40911 0.40102 0.011921
## 7  0.0114943      9   0.34951 0.36867 0.011522
## 8  0.0110685     10   0.33802 0.35547 0.011351
## 9  0.0106428     11   0.32695 0.34653 0.011232
## 10 0.0074500     12   0.31630 0.32652 0.010956
## 11 0.0051086     14   0.30140 0.32014 0.010865
## 12 0.0038314     16   0.29119 0.30907 0.010704
## 13 0.0029800     20   0.27586 0.29970 0.010564
## 14 0.0021286     21   0.27288 0.28948 0.010408
## 15 0.0019157     23   0.26862 0.28991 0.010414
## 16 0.0017029     25   0.26479 0.28736 0.010375
## 17 0.0012771     26   0.26309 0.28778 0.010381
## 18 0.0010000     29   0.25926 0.28608 0.010355

#Evaluation Charts
pred_test <- predict(tree_model_balanced, newdata = test, type = "prob")
pred <- prediction(pred_test[, 2], test$Order_Conversion)
#Gain Chart
perf <- performance(pred, "tpr", "rpp")
plot(perf)
```



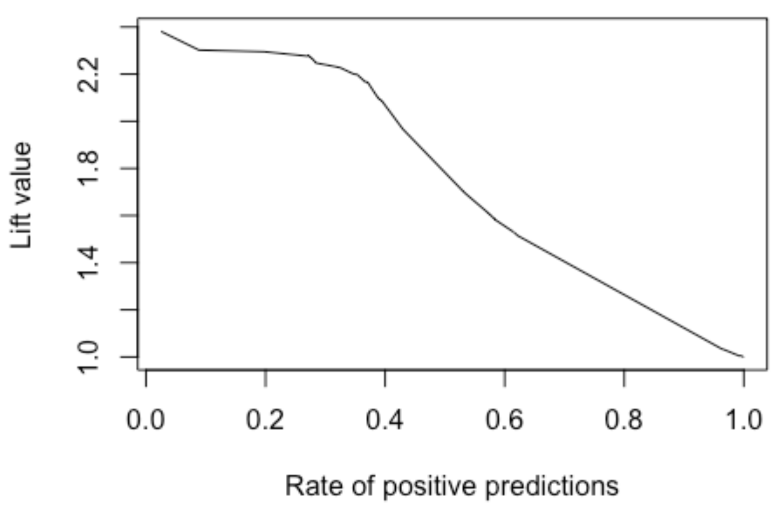

```r
#ROC Curve
perf <- performance(pred, "tpr", "fpr")
plot(perf)
```




```r
# Response Chart
perf <- performance(pred, "ppv", "rpp")
plot(perf)
```




```r
# Lift Chart
perf <- performance(pred, "lift", "rpp")
plot(perf)
```




```r
#Area Under Curve
auc <- unlist(slot(performance(pred, "auc"), "y.values"))
print(paste("The Area Under the Curve is ", auc))

## [1] "The Area Under the Curve is  0.920388848660391"
```

```
#CROSS VALIDATION
dt_cross <- data_70_30_split[sample(nrow(data_70_30_split)),]
k <- 5
nmethod <- 1
folds <- cut(seq(1,nrow(dt_cross)), breaks = k, labels = FALSE)
model.err <- matrix(-1, k, nmethod, dimnames = list(paste0("Fold", 1:k), c("D
ecision Tree Model")))

for (i in 1:k)
{
  testindexes <- which(folds == i, arr.ind = TRUE)
  test <- dt_cross[testindexes,]
  train <- dt_cross[-testindexes,]

tree_model <- rpart(Order_Conversion ~ ., train, method = "class", control =
rpart.control(minsplit=20, minbucket=10, cp= 0.001))
  predict_treemodel <- predict(tree_model, test, type = "class")
  model.err[i] <- mean(test$Order_Conversion!= predict_treemodel)
}
print(paste("The CV Error rate of Decision Tree after Cross Validation is ",r
ound(mean(model.err),3), sep = ""))

## [1] "The CV Error rate of Decision Tree after Cross Validation is 0.115"
```

For the Decision Tree of Balanced data, we chose the final minsplit and minbucket values to be 20 and 10 respectively as these were the values in which the model achieved highest accuracy and precision when a decision tree with cp = -1, minsplit = 0 and minbucket = 0 was constructed. From the cp plot, we found the lowest dip of cp value below the horizontal which was around 0.001.This is often a good indicator of the cp value. We also verified the cp value corresponding to the lowest dip in xerror value and obtained the corresponding cp value. Both the analysis fetched us a cp value of 0.001

Model Analysis - Identify Features

```
#Important variables in final balanced tree model
print(tree_model_balanced$variable.importance)

## CustomerCode       AreaFt    ITEM_NAME  CountryName  QtyRequired    ShapeN
ame
##    686.77649    656.39024    462.56106    385.10832    114.32896     14.34
472
```

From the Balanced Data Decision Tree, after performing variable importance, we attain the following attributes as the most important attributes –

1. AreaFt 2. CustomerCode 3. ITEM_NAME 4. CountryName 5. QtyRequired 6. ShapeName

Therefore, these are the variables that contributed the most towards the target variable, Order_Conversion

Balanced Vs. Unbalanced Data

	Decision Tree	
70/30 Split	Unbalanced Data	Balanced Data
Training Accuracy	92.5	88.5
Training Recall	86.9	92.2
Training Precision	73	79
Training Error	7.5	11.6
Test Accuracy	91.7	88.1
Test Recall	85.4	91.3
Test Precision	72.7	79.2
Test Error	8.3	12
Cross Validation	8.4	12.2
Area Under Curve	90.9	92

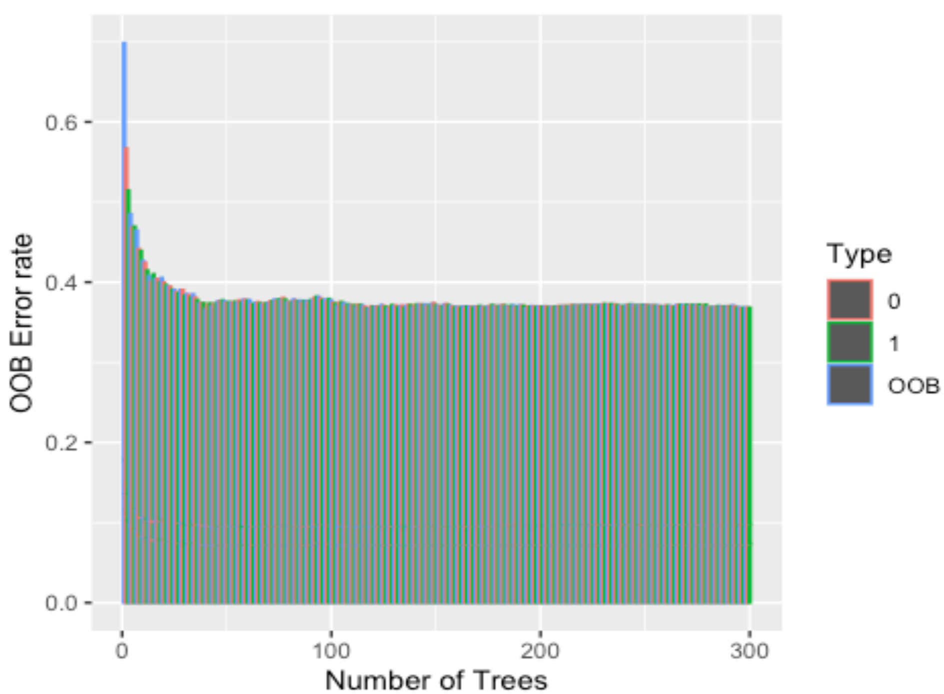
Considering recall as our metric, we witness that our Balanced Data performed better with a recall percentage of 91.3% in test data.

Model 2 - Random Forest

Unbalanced Data

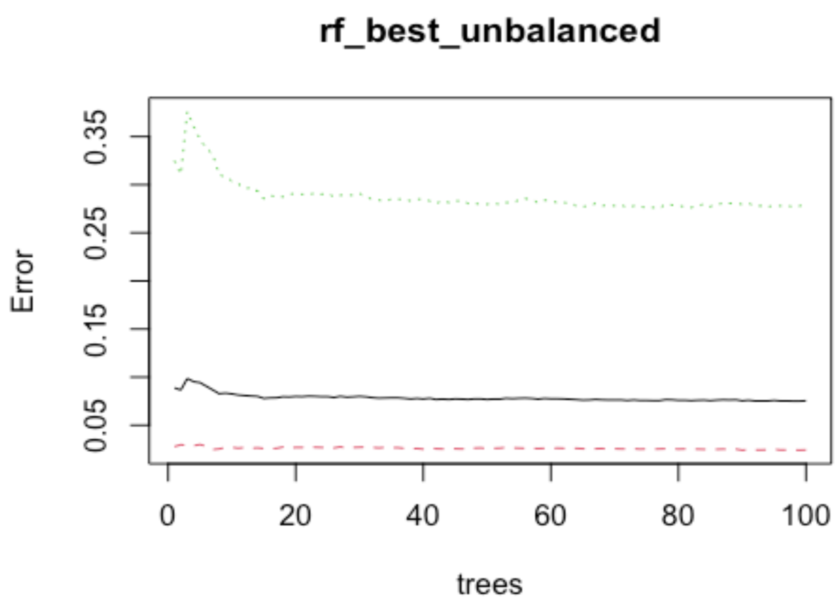
```
set.seed(1346)
rf_data <- champo_sample_only
rf <- randomForest(Order_Conversion ~ ., data= rf_data, ntree = 300, mtry = s
qrt(ncol(rf_data)-1), proximity = T, importance = T)
print(rf)

##
## Call:
##  randomForest(formula = Order_Conversion ~ ., data = rf_data,      ntree =
300, mtry = sqrt(ncol(rf_data) - 1), proximity = T,      importance = T)
##                Type of random forest: classification
##                      Number of trees: 300
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 7.37%
## Confusion matrix:
##      0   1 class.error
## 0 4538 113  0.02429585
## 1  316 853  0.27031651

#Determining best value of mtry using validation set
indx <- sample(2, nrow(rf_data), replace = T, prob= c(0.7,0.3))
Train <- rf_data[indx == 1,]
Validation <- rf_data[indx == 2,]
pr.err <- c()
for(mt in seq(1, ncol(Train)))
{
  rf_mtry <- randomForest(Order_Conversion ~., data = Train, ntree = 300, mtr
```

```
y = ifelse(mt == ncol(Train), mt -1, mt))
  pred <- predict(rf_mtry, newdata = Validation, type = "class")
  pr.err<- c(pr.err, mean(pred != Validation$Order_Conversion))
}
pr.err

## [1] 0.08178654 0.07598608 0.07946636 0.08004640 0.08004640 0.08062645 0.08
062645

bestmtry <- which.min(pr.err)
print(paste("The Best mtry is ", bestmtry))

## [1] "The Best mtry is  2"

#Determining best ntree
oob_err <- data.frame(trees = rep(1:nrow(rf$err.rate), times = 3), Type = rep
(c("OOB", "0", "1"), each = row(rf$err.rate)),
                                  error = c(rf$err.rate[,"OOB"], rf$err.rate[
,"0"], rf$err.rate[,"1"]))

## Warning in rep(c("OOB", "0", "1"), each = row(rf$err.rate)): first element
used
## of 'each' argument

ggplot(data = oob_err, aes(x = trees, y = error)) + geom_col(aes(color=Type))
+ xlab("Number of Trees") + ylab("OOB Error rate")
```



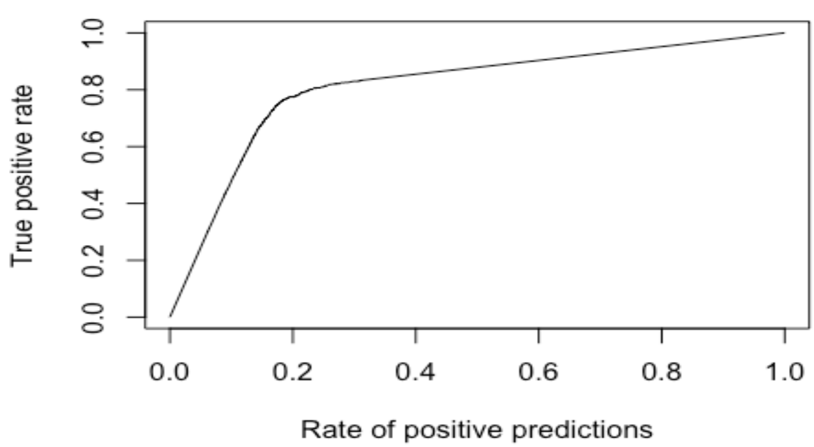

```
#Random Forest with best mtry and ntree
ntree = 100
rf_best_unbalanced <- randomForest(Order_Conversion ~ ., data= rf_data, ntree
```

```
= ntree, mtry = bestmtry, proximity = T, importance = T)
print(rf_best_unbalanced)

##
## Call:
##  randomForest(formula = Order_Conversion ~ ., data = rf_data,      ntree =
ntree, mtry = bestmtry, proximity = T, importance = T)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 7.54%
## Confusion matrix:
##      0   1 class.error
## 0 4538 113  0.02429585
## 1  326 843  0.27887083

plot(rf_best_unbalanced)
```

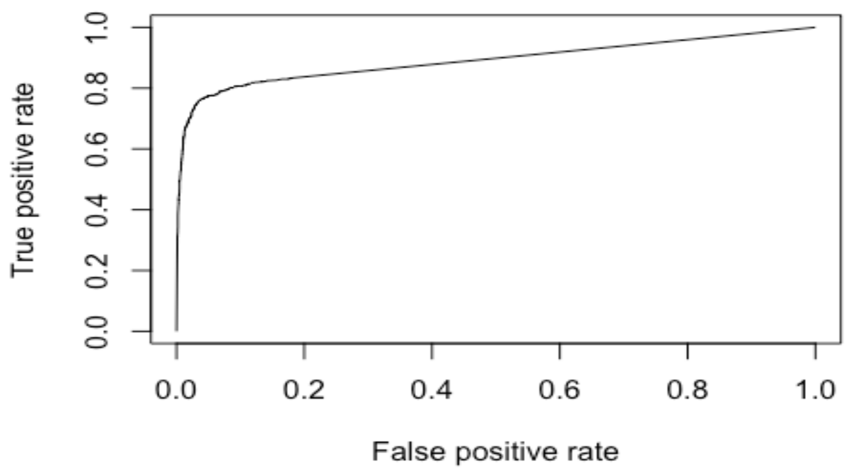



```
#Confusion Matrix
CM <- table(rf_best_unbalanced$predicted, rf_data$Order_Conversion, dnn = c("
Predicted", "Actual"))
error_metric = function(CM){
  TN = CM[1,1]
  TP = CM[2,2]
  FN = CM[1,2]
  FP = CM[2,1]
  accuracy = (TP+TN)/(TP+TN+FP+FN)
  recall = (TP)/(TP+FN)
  precision = (TP)/(TP+FP)
```

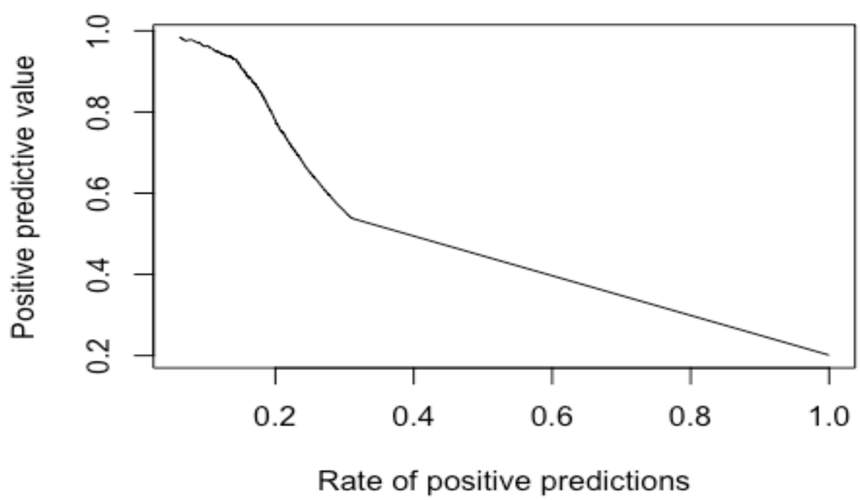
```
  falsePositiveRate = (FP)/(FP+TN)
  falseNegativeRate = (FN)/(FN+TP)
  error = (FP+FN)/(TP+TN+FP+FN)
  modelPerf <- list("accuracy" = accuracy,
                    "precision" = precision,
                    "recall" = recall,
                    "falsepositiverate" = falsePositiveRate,
                    "falsenegativerate" = falseNegativeRate,
                    "error" = error
                    )
  return(modelPerf)
}

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

##                          Values
## 1          accuracy 0.92457045
## 2         precision 0.88179916
## 3            recall 0.72112917
## 4 falsepositiverate 0.02429585
## 5 falsenegativerate 0.27887083
## 6             error 0.07542955

#Evaluation Charts
score <- rf_best_unbalanced$votes[,2]
pred <- prediction(score, rf_data$Order_Conversion)
#Gain Chart
perf <- performance(pred, "tpr", "rpp")
plot(perf)
```



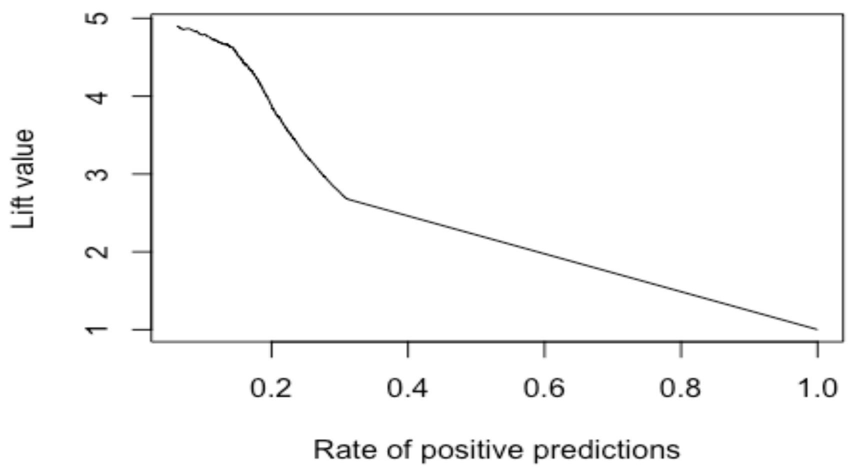
```r
#ROC Curve
perf <- performance(pred, "tpr", "fpr")
plot(perf)
```




```r
# Response Chart
perf <- performance(pred, "ppv", "rpp")
plot(perf)
```




```r
# Lift Chart
perf <- performance(pred, "lift", "rpp")
plot(perf)
```




```r
#Area Under Curve
auc <- unlist(slot(performance(pred, "auc"), "y.values"))
print(paste("The Area Under the Curve is ", auc))

## [1] "The Area Under the Curve is  0.89057266491068"

#CROSS Validation
rf_data <- rf_data[sample(nrow(rf_data)),]
k <- 10
nmethod <- 1
folds <- cut(seq(1,nrow(rf_data)), breaks = k, labels = FALSE)
model.err <- matrix(-1, k, nmethod, dimnames = list(paste0("Fold", 1:k), c("R
andom Forest Model")))

for (i in 1:k)
{
  testindexes <- which(folds == i, arr.ind = TRUE)
  test <- rf_data[testindexes,]
  train <- rf_data[-testindexes,]

rf_cross <- randomForest(Order_Conversion ~ ., data= rf_data, ntree = 100, mt
ry = bestmtry, proximity = T, importance = T)
  predict_treemodel <- predict(rf_cross, test, type = "class")
  model.err[i] <- mean(test$Order_Conversion!= predict_treemodel)
}
print(paste("The CV Error rate of Random Forest after Cross Validation is",me
an(model.err)))

## [1] "The CV Error rate of Random Forest after Cross Validation is 0.062027
4914089347"
```

Implemented Random Forest using ntree parameter as 300 and mtry as the square root of the number of variables. As a next step, to determine the best ntree and mtry values, a for loop was used to iterate over a range of values. Post this, we implemented the Random Forest model again with the best ntree= 100 and mtry=2 and achieved an OOB error rate of 7.54%

Balanced Data

```
set.seed(1346)
rf_data <- balanced.data
rf <- randomForest(Order_Conversion ~ ., data= rf_data, ntree = 300, mtry = s
qrt(ncol(rf_data)-1), proximity = T, importance = T)
print(rf)
```

```
##
## Call:
##  randomForest(formula = Order_Conversion ~ ., data = rf_data,      ntree =
300, mtry = sqrt(ncol(rf_data) - 1), proximity = T,      importance = T)
##                Type of random forest: classification
##                      Number of trees: 300
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 10.4%
## Confusion matrix:
##      0    1 class.error
## 0 4448  203  0.04364653
## 1  629 2720  0.18781726
```

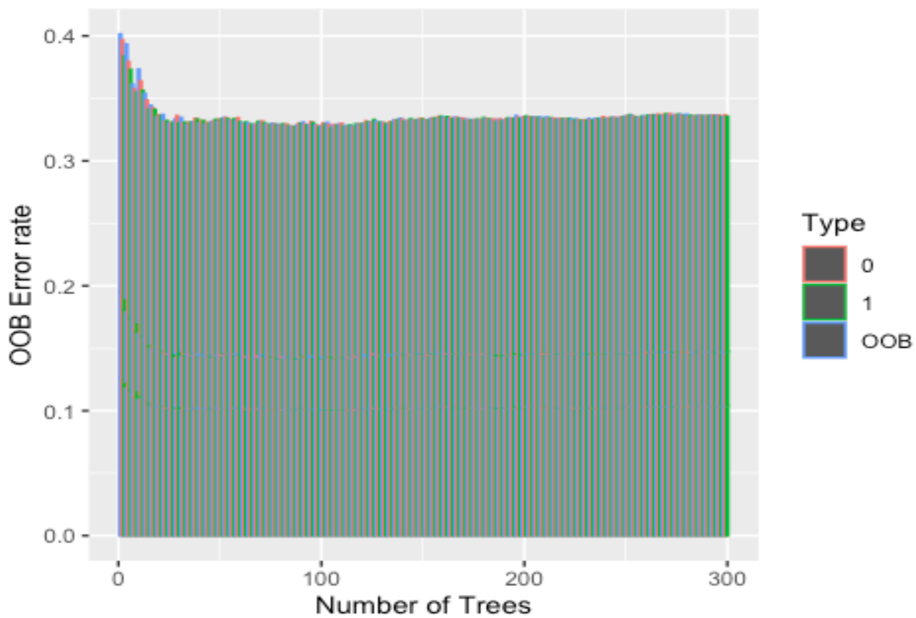
```
#Determining best value of mtry using validation set
indx <- sample(2, nrow(rf_data), replace = T, prob= c(0.7,0.3))
Train <- rf_data[indx == 1,]
Validation <- rf_data[indx == 2,]
pr.err <- c()
for(mt in seq(1, ncol(Train)))
{
  rf_mtry <- randomForest(Order_Conversion ~., data = Train, ntree = 300, mtr
y = ifelse(mt == ncol(Train), mt -1, mt))
  pred <- predict(rf_mtry, newdata = Validation, type = "class")
  pr.err<- c(pr.err, mean(pred != Validation$Order_Conversion))
}
pr.err
```

```
## [1] 0.11575031 0.10086813 0.09466722 0.09549401 0.09590740 0.09549401 0.09
632079
```

```
bestmtry <- which.min(pr.err)
print(paste("The Best mtry is ", bestmtry))
```

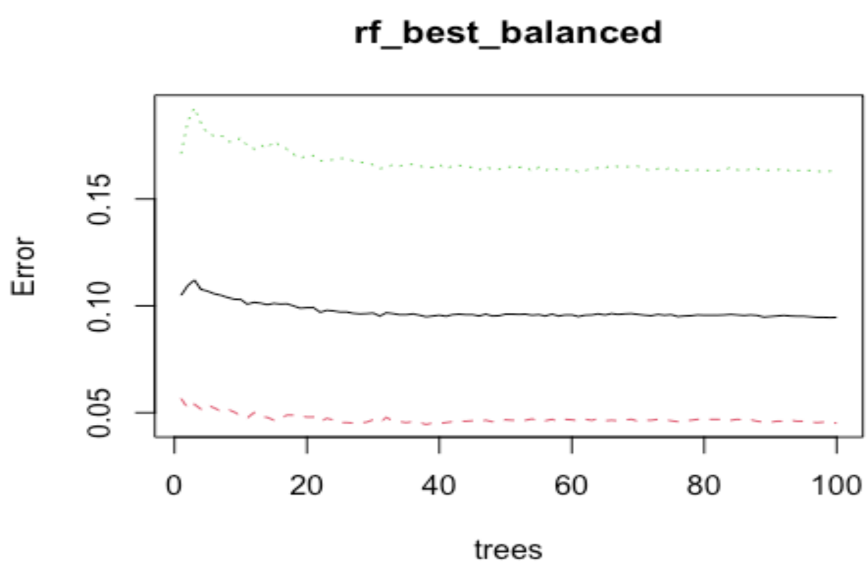
```
## [1] "The Best mtry is  3"
```

```
#Determining best ntree
oob_err <- data.frame(trees = rep(1:nrow(rf$err.rate), times = 3), Type = rep
(c("OOB", "0", "1"), each = row(rf$err.rate)),
                                    error = c(rf$err.rate[,"OOB"], rf$err.rate[
,"0"], rf$err.rate[,"1"]))

## Warning in rep(c("OOB", "0", "1"), each = row(rf$err.rate)): first element
used
## of 'each' argument

ggplot(data = oob_err, aes(x = trees, y = error)) + geom_col(aes(color=Type))
+ xlab("Number of Trees") + ylab("OOB Error rate")
```




```
#Random Forest with best mtry and ntree
ntree = 100
rf_best_balanced <- randomForest(Order_Conversion ~ ., data= rf_data, ntree =
ntree, mtry = bestmtry, proximity = T, importance = T)
print(rf_best_balanced)

##
## Call:
##  randomForest(formula = Order_Conversion ~ ., data = rf_data,      ntree =
ntree, mtry = bestmtry, proximity = T, importance = T)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 9.46%
## Confusion matrix:
##      0    1 class.error
## 0 4441  210  0.04515158
## 1  547 2802  0.16333234
```

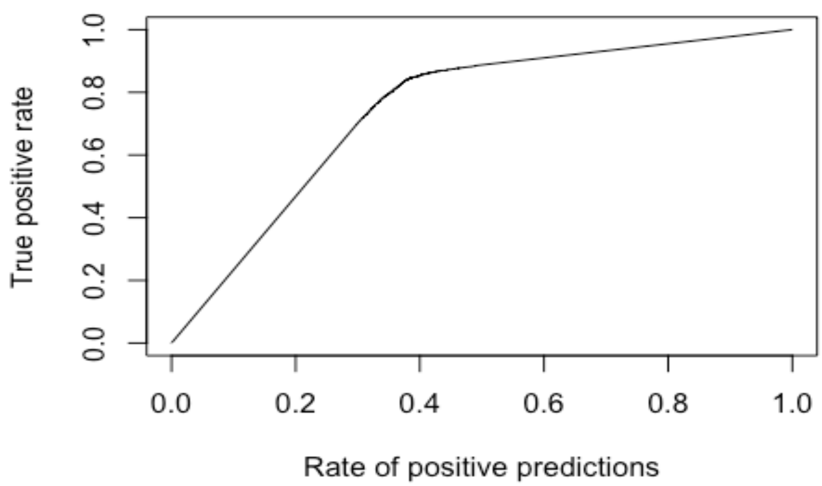
```
plot(rf_best_balanced)
```



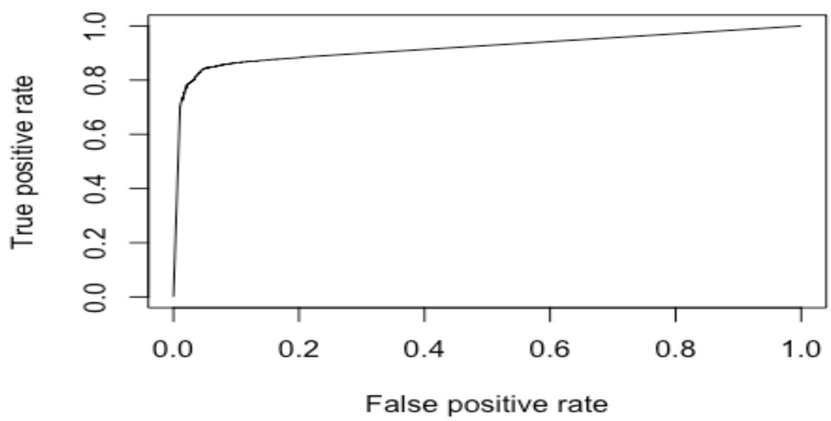

```
#Confusion Matrix
CM <- table(rf_best_balanced$predicted, rf_data$Order_Conversion, dnn = c("Pr
edicted", "Actual"))
error_metric = function(CM){
  TN = CM[1,1]
  TP = CM[2,2]
  FN = CM[1,2]
  FP = CM[2,1]
  accuracy = (TP+TN)/(TP+TN+FP+FN)
  recall = (TP)/(TP+FN)
  precision = (TP)/(TP+FP)
  falsePositiveRate = (FP)/(FP+TN)
  falseNegativeRate = (FN)/(FN+TP)
  error = (FP+FN)/(TP+TN+FP+FN)
  modelPerf <- list("accuracy" = accuracy,
                    "precision" = precision,
                    "recall" = recall,
                    "falsepositiverate" = falsePositiveRate,
                    "falsenegativerate" = falseNegativeRate,
                    "error" = error
                    )
  return(modelPerf)
}

outPutlist <- error_metric(CM)

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

```
##                        Values
## 1          accuracy 0.90537500
## 2         precision 0.93027888
## 3            recall 0.83666766
## 4 falsepositiverate 0.04515158
## 5 falsenegativerate 0.16333234
## 6             error 0.09462500
```

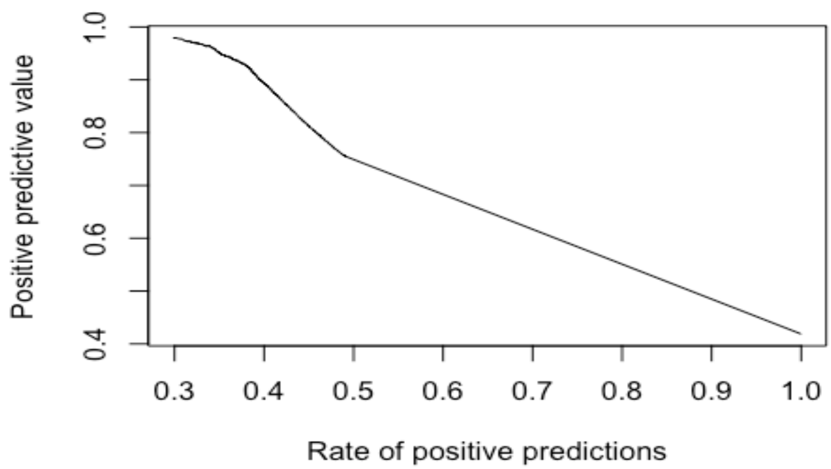
```
#Evaluation Charts
score <- rf_best_balanced$votes[,2]
pred <- prediction(score, rf_data$Order_Conversion)
#Gain Chart
perf <- performance(pred, "tpr", "rpp")
plot(perf)
```



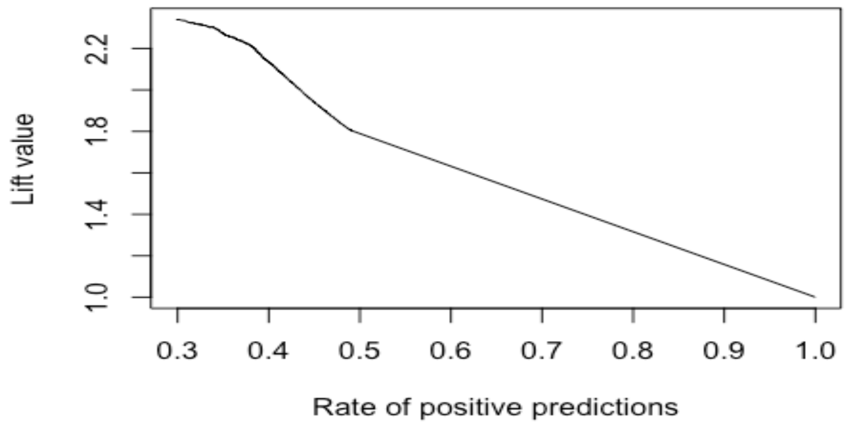

```
#ROC Curve
perf <- performance(pred, "tpr", "fpr")
plot(perf)
```




```
# Response Chart
perf <- performance(pred, "ppv", "rpp")
plot(perf)
```




```
# Lift Chart
perf <- performance(pred, "lift", "rpp")
plot(perf)
```



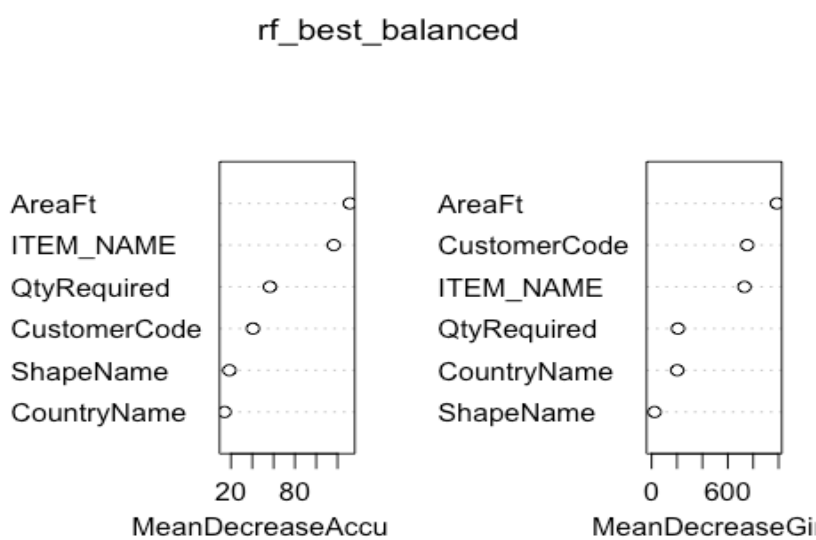

```
#Area Under Curve
auc <- unlist(slot(performance(pred, "auc"), "y.values"))
print(paste("The Area Under the Curve is ", auc))

## [1] "The Area Under the Curve is  0.918859536912696"

#CROSS Validation
rf_data <- rf_data[sample(nrow(rf_data)),]
k <- 10
nmethod <- 1
folds <- cut(seq(1,nrow(rf_data)), breaks = k, labels = FALSE)
model.err <- matrix(-1, k, nmethod, dimnames = list(paste0("Fold", 1:k), c("R
andom Forest Model")))

for (i in 1:k)
{
  testindexes <- which(folds == i, arr.ind = TRUE)
  test <- rf_data[testindexes,]
  train <- rf_data[-testindexes,]

rf_cross <- randomForest(Order_Conversion ~ ., data= rf_data, ntree = 100, mt
ry = bestmtry, proximity = T, importance = T)
  predict_treemodel <- predict(rf_cross, test, type = "class")
  model.err[i] <- mean(test$Order_Conversion!= predict_treemodel)
}
print(paste("The CV Error rate of Random Forest after Cross Validation is",me
an(model.err)))

## [1] "The CV Error rate of Random Forest after Cross Validation is 0.083"
```

Implemented Random Forest using ntree parameter as 300 and mtry as the square root of the number of variables. As a next step, to determine the best ntree and mtry values, a for loop was used to iterate over a range of values. Post this, we implemented the Random

Forest model again with the best ntree= 100 and mtry=4 and achieved an OOB error rate of 9.51%

Model Analysis - Identify Features

```
importance(rf_best_balanced, type = 1)

##              MeanDecreaseAccuracy
## CustomerCode             40.65557
## CountryName              14.03898
## QtyRequired              56.58150
## ITEM_NAME               116.55828
## ShapeName                18.52276
## AreaFt                  131.55868

importance(rf_best_balanced, type = 2)

##              MeanDecreaseGini
## CustomerCode        752.43712
## CountryName         201.64168
## QtyRequired         207.47768
## ITEM_NAME           730.97506
## ShapeName            23.32117
## AreaFt              985.89698

varImpPlot(rf_best_balanced)
```



From the Balanced Data Random Forest Model, after performing variable importance, we attain the following attributes as the most important attributes - **1. AreaFt 2. CustomerCode 3. ITEM_NAME 4. CountryName 5. QtyRequired 6. ShapeName**

Therefore, these are the variables that contributed the most towards the target variable, Order_Conversion

Balanced Vs. Unbalanced Data

Random Forest		
	Unbalanced Data	Balanced Data
Test Accuracy	92.4	89.9
Test Recall	72.1	83.1
Test Precision	88.1	91.9
Test Error	7.5	95.1
Cross Validation	6.2	8.7
Area Under Curve	89	91.1
OOB	7.5	10.7

Considering recall as our metric, we witness that our Balanced Data performed better with a recall percentage of 83.1% in test data.

Model 3 - Logistic Regression

Unbalanced Data

```
logit_data <- champo_sample_only

set.seed(1766)
indx <- sample(2, nrow(logit_data), replace = T, prob= c(0.7,0.3))
train <- logit_data[indx == 1,]
test <- logit_data[indx == 2,]
logitModel_unbalanced <- glm(Order_Conversion ~ ., data = train, family = "bi
nomial")
pred <- predict(logitModel_unbalanced, newdata = test, type = "response")clas
s <- as.factor(ifelse(pred >= 0.5, 1, 0))
#Confusion Matrix
CM <- table(class, test$Order_Conversion, dnn = c("Predicted", "Actual"))
error_metric = function(CM){
  TN = CM[1,1]
  TP = CM[2,2]
  FN = CM[1,2]
  FP = CM[2,1]
  accuracy = (TP+TN)/(TP+TN+FP+FN)
  recall = (TP)/(TP+FN)
  precision = (TP)/(TP+FP)
  falsePositiveRate = (FP)/(FP+TN)
  falseNegativeRate = (FN)/(FN+TP)
  error = (FP+FN)/(TP+TN+FP+FN)
  modelPerf <- list("accuracy" = accuracy,
                    "precision" = precision,
                    "recall" = recall,
                    "falsepositiverate" = falsePositiveRate,
                    "falsenegativerate" = falseNegativeRate,
                    "error" = error
```

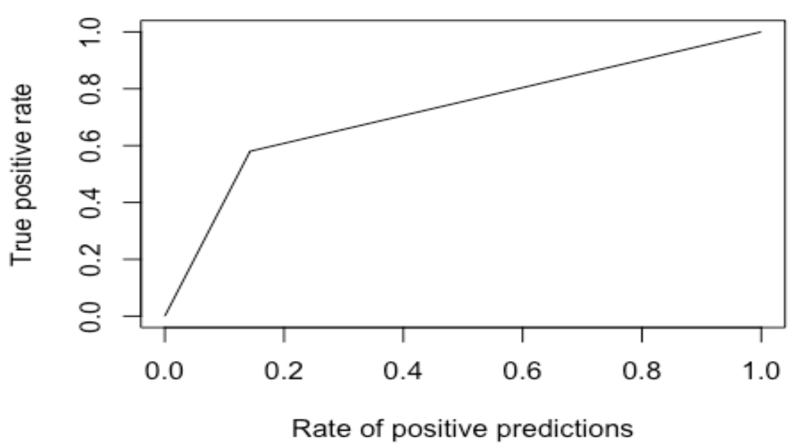
```
                    )
  return(modelPerf)
}
outPutlist <- error_metric(CM)
library(plyr)
df <- ldply(outPutlist, data.frame)
setNames(df,c("","Values"))
```

```
##                          Values
## 1          accuracy 0.88882587
## 2         precision 0.80555556
## 3            recall 0.58000000
## 4 falsepositiverate 0.03467799
## 5 falsenegativerate 0.42000000
## 6             error 0.11117413
```

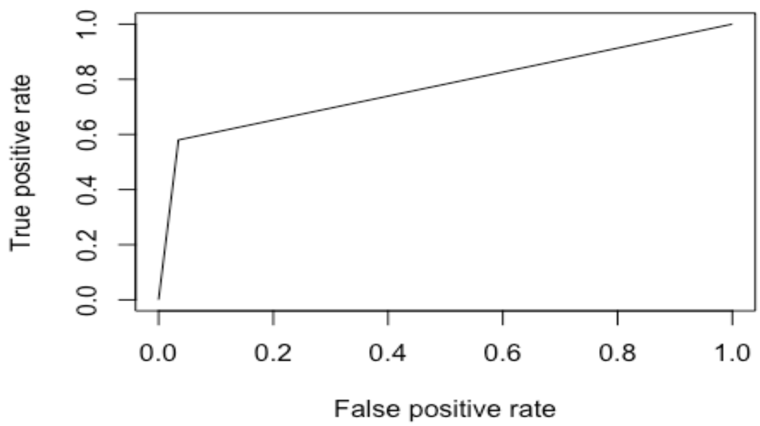
```
#Evaluation Charts
pred_test <- predict(logitModel_unbalanced, newdata = test, type = "response"
)
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
== :
## prediction from a rank-deficient fit may be misleading
```

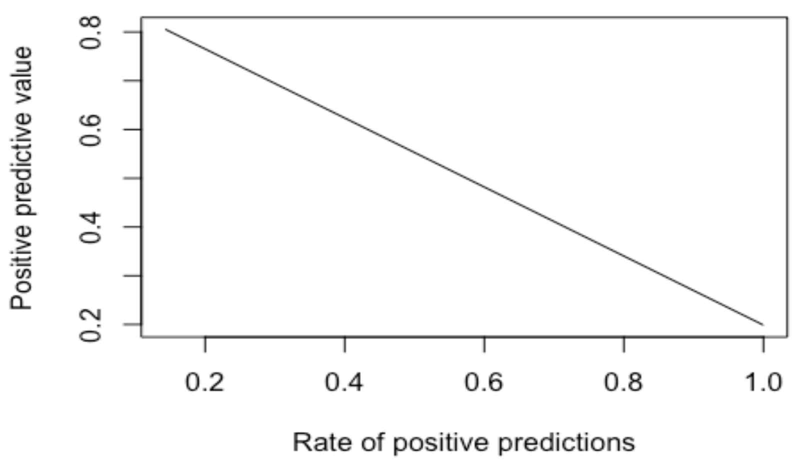
```
pred_threshold <- ifelse(pred_test > 0.5,1,0)
pred <- prediction(pred_threshold, test$Order_Conversion)
#Gain Chart
perf <- performance(pred, "tpr", "rpp")
plot(perf)
```



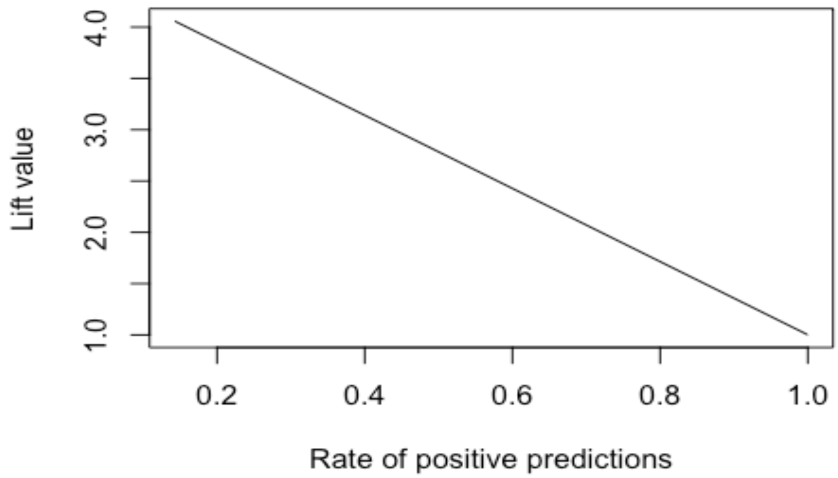

```
#ROC Curve
perf <- performance(pred, "tpr", "fpr")
plot(perf)
```




```
# Response Chart
perf <- performance(pred, "ppv", "rpp")
plot(perf)
```




```
# Lift Chart
perf <- performance(pred, "lift", "rpp")
plot(perf)
```

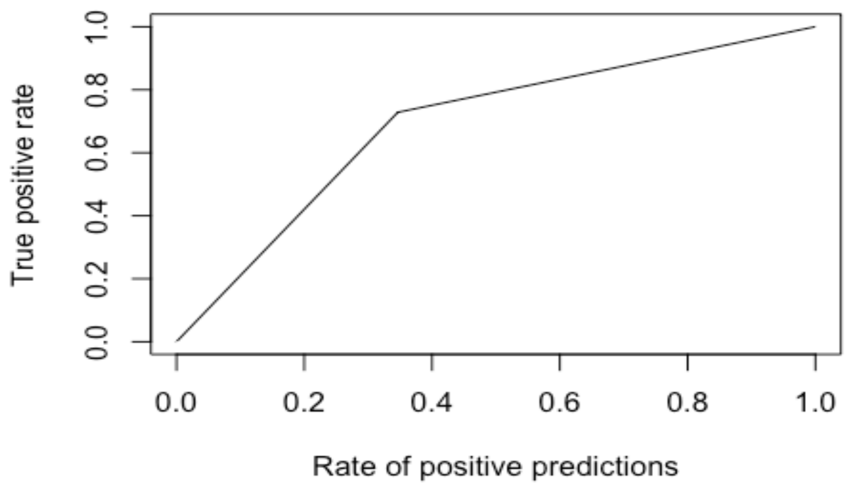



```
#Area Under Curve
auc <- unlist(slot(performance(pred, "auc"), "y.values"))
print(paste("The Area Under the Curve is ", auc))

## [1] "The Area Under the Curve is  0.772661004953999"
```

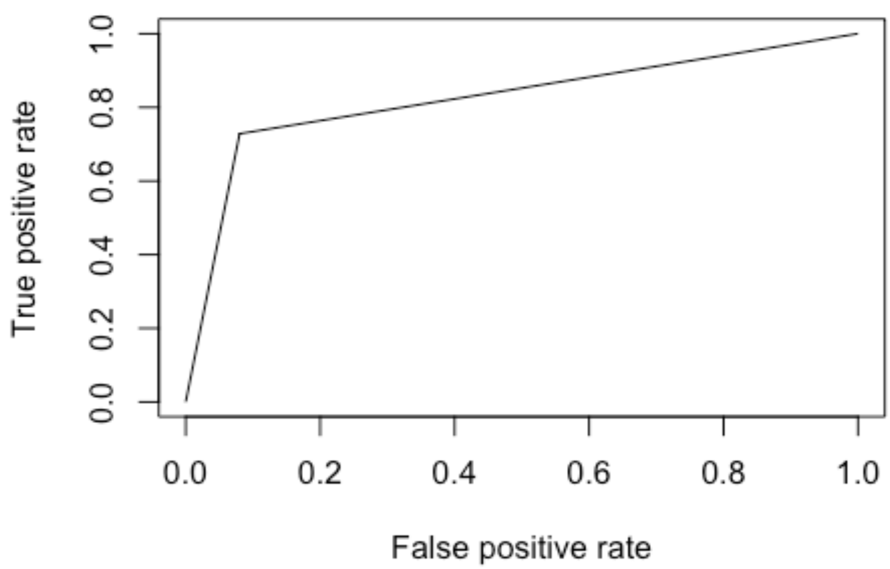
Balanced Data

```
logit_data <- balanced.data

set.seed(34)
indx <- sample(2, nrow(logit_data), replace = T, prob= c(0.7,0.3))
train <- logit_data[indx == 1,]
test <- logit_data[indx == 2,]

logitModel_balanced <- glm(Order_Conversion ~ ., data = train, family = "bino
mial")

pred <- predict(logitModel_balanced, newdata = test, type = "response")

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
== :
## prediction from a rank-deficient fit may be misleading

class <- as.factor(ifelse(pred >= 0.5, 1, 0))
#Confusion Matrix
CM <- table(class, test$Order_Conversion, dnn = c("Predicted", "Actual"))

error_metric = function(CM){
  TN = CM[1,1]
  TP = CM[2,2]
  FN = CM[1,2]
```

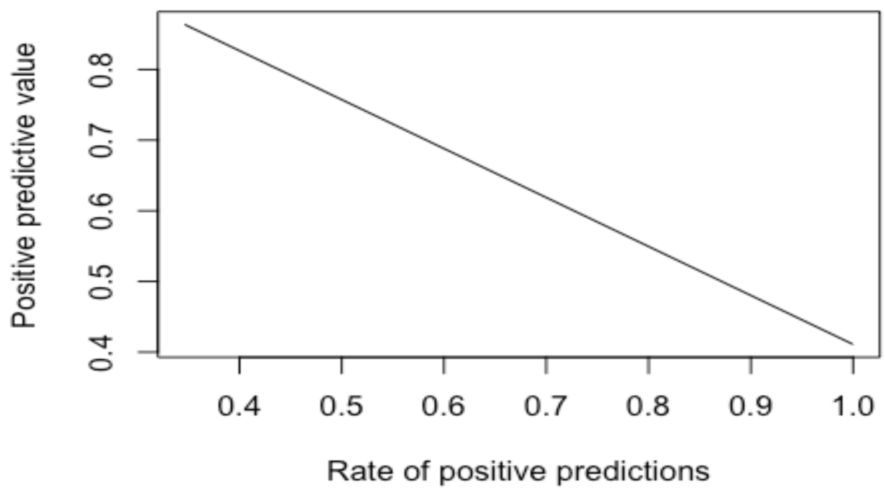
```
  FP = CM[2,1]
  accuracy = (TP+TN)/(TP+TN+FP+FN)
  recall = (TP)/(TP+FN)
  precision = (TP)/(TP+FP)
  falsePositiveRate = (FP)/(FP+TN)
  falseNegativeRate = (FN)/(FN+TP)
  error = (FP+FN)/(TP+TN+FP+FN)
  modelPerf <- list("accuracy" = accuracy,
                    "precision" = precision,
     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

outPutlist <- error_metric(CM)

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

##                        Values
## 1          accuracy 0.84120351
## 2         precision 0.86369119
## 3            recall 0.72838250
## 4 falsepositiverate 0.08014184
## 5 falsenegativerate 0.27161750
## 6             error 0.15879649

#Evaluation Charts
pred_test <- predict(logitModel_balanced, newdata = test, type = "response")

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type
== :
## prediction from a rank-deficient fit may be misleading

pred_threshold <- ifelse(pred_test > 0.5,1,0)
pred <- prediction(pred_threshold, test$Order_Conversion)
#Gain Chart
perf <- performance(pred, "tpr", "rpp")
plot(perf)
```



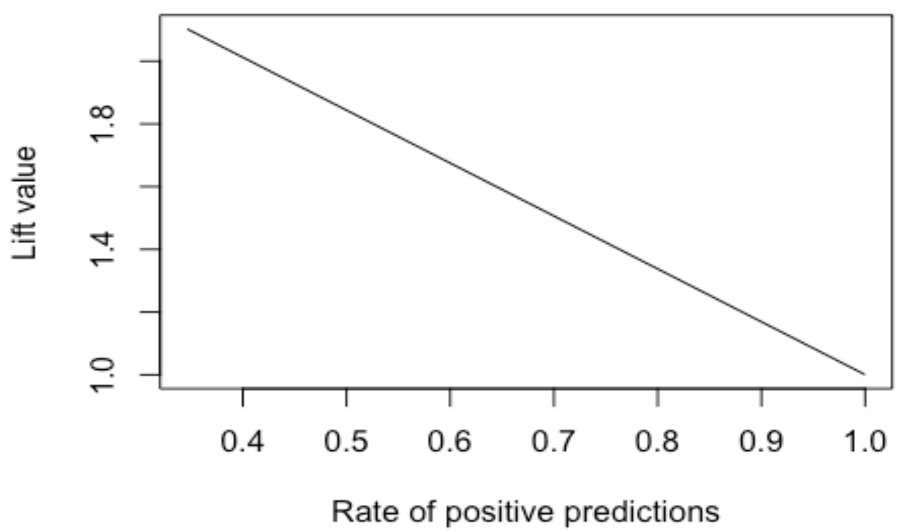

```r
#ROC Curve
perf <- performance(pred, "tpr", "fpr")
plot(perf)
```




```r
# Response Chart
perf <- performance(pred, "ppv", "rpp")
plot(perf)
```




```
# Lift Chart
perf <- performance(pred, "lift", "rpp")
plot(perf)
```




```
#Area Under Curve
auc <- unlist(slot(performance(pred, "auc"), "y.values"))
print(paste("The Area Under the Curve is ", auc))

## [1] "The Area Under the Curve is  0.824120329285802"
```

Model Analysis - Identify Features

```
summary(logitModel_balanced)

##
## Call:
## glm(formula = Order_Conversion ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.2456  -0.6634  -0.3187   0.5361   2.7482
##
## Coefficients: (13 not defined because of singularities)
##                                Estimate Std. Error z value Pr(>|z|)
## (Intercept)                  -8.987e-01  1.396e+00  -0.644 0.519560
## CustomerCodeA-9              -2.216e-01  1.402e+00  -0.158 0.874440
## CustomerCodeB-2              -1.540e+01  1.455e+03  -0.011 0.991560
## CustomerCodeB-3              -1.542e+01  1.455e+03  -0.011 0.991545
## CustomerCodeC-1              -3.775e-01  1.432e+00  -0.264 0.792088
## CustomerCodeC-2              -1.670e-01  1.422e+00  -0.117 0.906473
## CustomerCodeCC               -2.347e+00  1.394e+00  -1.684 0.092189 .
## CustomerCodeCTS              -1.624e+01  4.562e+02  -0.036 0.971595
## CustomerCodeE-2               2.993e+00  1.569e+00   1.907 0.056469 .
## CustomerCodeF-1              -4.692e-01  1.436e+00  -0.327 0.743832
## CustomerCodeF-2              -1.484e+00  1.716e+00  -0.865 0.387268
## CustomerCodeF-6               1.547e+01  4.102e+02   0.038 0.969907
## CustomerCodeH-2              -7.980e-01  1.403e+00  -0.569 0.569436
## CustomerCodeI-2               5.126e-01  1.490e+00   0.344 0.730858
## CustomerCodeJL                1.219e+00  1.425e+00   0.855 0.392565
## CustomerCodeK-2              -1.551e+01  1.455e+03  -0.011 0.991496
## CustomerCodeK-3              -1.498e+01  1.455e+03  -0.010 0.991790
## CustomerCodeL-3              -1.530e+01  6.509e+02  -0.024 0.981245
## CustomerCodeL-4              -1.758e+01  1.029e+03  -0.017 0.986370
## CustomerCodeL-5              -2.358e+00  1.542e+00  -1.529 0.126239
## CustomerCodeM-1              -5.079e-01  1.414e+00  -0.359 0.719424
## CustomerCodeM-2              -2.449e-01  1.429e+00  -0.171 0.863900
## CustomerCodeN-1              -2.950e+00  1.423e+00  -2.073 0.038153 *
## CustomerCodeP-4              -2.147e+00  1.450e+00  -1.481 0.138522
## CustomerCodeP-5               6.744e-02  1.419e+00   0.048 0.962097
## CustomerCodePC               -1.531e+01  1.029e+03  -0.015 0.988132
## CustomerCodePD                3.675e+00  1.447e+00   2.540 0.011076 *
## CustomerCodeRC               -2.189e+00  1.693e+00  -1.293 0.195871
## CustomerCodeS-3              -9.366e-01  1.413e+00  -0.663 0.507530
## CustomerCodeT-2              -3.737e+00  1.518e+00  -2.462 0.013833 *
## CustomerCodeT-4              -1.713e-01  1.964e+00  -0.087 0.930504
## CustomerCodeT-5              -1.119e+00  1.405e+00  -0.796 0.425821
## CustomerCodeTGT              -1.995e+00  1.409e+00  -1.415 0.156925
## CustomerCodeV-1              -3.444e+00  1.910e+00  -1.803 0.071390 .
## CountryNameBELGIUM                   NA         NA      NA       NA
## CountryNameBRAZIL                    NA         NA      NA       NA
```

```
## CountryNameCANADA                     NA         NA      NA       NA
## CountryNameCHINA                      NA         NA      NA       NA
## CountryNameINDIA                      NA         NA      NA       NA
## CountryNameISRAEL                     NA         NA      NA       NA
## CountryNameITALY                      NA         NA      NA       NA
## CountryNamePOLAND                     NA         NA      NA       NA
## CountryNameROMANIA                    NA         NA      NA       NA
## CountryNameSOUTH AFRICA               NA         NA      NA       NA
## CountryNameUAE                        NA         NA      NA       NA
## CountryNameUK                         NA         NA      NA       NA
## CountryNameUSA                        NA         NA      NA       NA
## QtyRequired                    3.828e-02  8.031e-03   4.766 1.88e-06 ***
## ITEM_NAMEDURRY                 4.539e-01  1.609e-01   2.821 0.004781 **
## ITEM_NAMEGUN TUFTED            2.770e+00  4.334e-01   6.391 1.65e-10 ***
## ITEM_NAMEHAND TUFTED           2.111e-01  1.552e-01   1.361 0.173593
## ITEM_NAMEHANDLOOM              6.250e-01  2.801e-01   2.231 0.025688 *
## ITEM_NAMEHANDWOVEN            -7.575e-01  2.098e-01  -3.610 0.000306 ***
## ITEM_NAMEINDO TIBBETAN         1.744e+01  5.146e+02   0.034 0.972961
## ITEM_NAMEJACQUARD             -1.077e-01  3.257e-01  -0.331 0.740825
## ITEM_NAMEKNOTTED               3.105e+00  2.212e-01  14.038  < 2e-16 ***
## ITEM_NAMEPOWER LOOM JACQUARD   5.821e+00  4.422e-01  13.165  < 2e-16 ***
## ITEM_NAMETABLE TUFTED          3.555e+00  4.465e-01   7.963 1.68e-15 ***
## ShapeNameROUND                 1.518e+00  3.139e-01   4.835 1.33e-06 ***
## ShapeNameSQUARE                2.330e+00  5.185e-01   4.493 7.01e-06 ***
## AreaFt                         5.889e-02  2.249e-03  26.184  < 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: 7635.8  on 5606  degrees of freedom
## Residual deviance: 4508.5  on 5559  degrees of freedom
## AIC: 4604.5
##
## Number of Fisher Scoring iterations: 14
```

```
varImp(logitModel_balanced, scale = F)
```

```
##                                 Overall
## CustomerCodeA-9              0.15802113
## CustomerCodeB-2              0.01057791
## CustomerCodeB-3              0.01059663
## CustomerCodeC-1              0.26359985
## CustomerCodeC-2              0.11748832
## CustomerCodeCC               1.68396404
## CustomerCodeCTS              0.03560821
## CustomerCodeE-2              1.90740155
## CustomerCodeF-1              0.32678360
## CustomerCodeF-2              0.86458282
## CustomerCodeF-6              0.03772541
```

```
## CustomerCodeH-2                  0.56888171
## CustomerCodeI-2                  0.34398479
## CustomerCodeJL                   0.85497466
## CustomerCodeK-2                  0.01065884
## CustomerCodeK-3                  0.01028963
## CustomerCodeL-3                  0.02350842
## CustomerCodeL-4                  0.01708306
## CustomerCodeL-5                  1.52910179
## CustomerCodeM-1                  0.35922917
## CustomerCodeM-2                  0.17141146
## CustomerCodeN-1                  2.07320531
## CustomerCodeP-4                  1.48131644
## CustomerCodeP-5                  0.04752242
## CustomerCodePC                   0.01487515
## CustomerCodePD                   2.54028444
## CustomerCodeRC                   1.29340592
## CustomerCodeS-3                  0.66268920
## CustomerCodeT-2                  2.46157404
## CustomerCodeT-4                  0.08721101
## CustomerCodeT-5                  0.79636316
## CustomerCodeTGT                  1.41548906
## CustomerCodeV-1                  1.80299161
## QtyRequired                      4.76626066
## ITEM_NAMEDURRY                   2.82140142
## ITEM_NAMEGUN TUFTED              6.39088187
## ITEM_NAMEHAND TUFTED             1.36074888
## ITEM_NAMEHANDLOOM                2.23089809
## ITEM_NAMEHANDWOVEN               3.60992954
## ITEM_NAMEINDO TIBBETAN           0.03389434
## ITEM_NAMEJACQUARD                0.33076130
## ITEM_NAMEKNOTTED                14.03814242
## ITEM_NAMEPOWER LOOM JACQUARD 13.16517328
## ITEM_NAMETABLE TUFTED            7.96292076
## ShapeNameROUND                   4.83453750
## ShapeNameSQUARE                  4.49329077
## AreaFt                          26.18378130
```

From the Balanced Data's Logistic Regression Model's summary statistics, we see that for the prescribed alpha values of 1%, 5% and 10%, only the following attributes are significant, that is, p value less than the alpha value - **1. AreaFt 2. ShapeName 3. ITEM_NAME 4. QtyRequired 5. CustomerCode**

Performing AIC with Forward Selection to determine the important features, we get

```
logit_data <- balanced.data

set.seed(34)
indx <- sample(2, nrow(logit_data), replace = T, prob= c(0.7,0.3))
train <- logit_data[indx == 1,]
test <- logit_data[indx == 2,]
```

```r
model1 <- logitModel_balanced <- glm(Order_Conversion ~ CountryName, data = t
rain, family = "binomial")
model2 <- logitModel_balanced <- glm(Order_Conversion ~ CountryName + ITEM_NA
ME, data = train, family = "binomial")
model3 <- logitModel_balanced <- glm(Order_Conversion ~ CountryName + ITEM_NA
ME + CustomerCode , data = train, family = "binomial")
model4 <- logitModel_balanced <- glm(Order_Conversion ~ CountryName + ITEM_NA
ME + CustomerCode + ShapeName, data = train, family = "binomial")
model5 <- logitModel_balanced <- glm(Order_Conversion ~ CountryName + ITEM_NA
ME + CustomerCode + ShapeName + AreaFt, data = train, family = "binomial")
model6 <- logitModel_balanced <- glm(Order_Conversion ~ CountryName + ITEM_NA
ME + CustomerCode + ShapeName + AreaFt + QtyRequired, data = train, family =
"binomial")

library(AICcmodavg)

##
## Attaching package: 'AICcmodavg'

## The following object is masked from 'package:randomForest':
##
##     importance

#define list of models
models <- list(model1, model2, model3, model4, model5, model6)

#specify model names
mod.names <- c('country', 'country.item', 'country.item.custcode','country.it
em.custcode.shape','country.item.custcode.shape.area','country.item.custcode.
shape.area.qty')

#calculate AIC of each model
aictab(cand.set = models, modnames = mod.names)

##
## Model selection based on AICc:
##
##                                     K    AICc Delta_AICc AICcWt Cum.Wt
## country.item.custcode.shape.area.qty 48 4605.32       0.00      1      1
## country.item.custcode.shape.area     47 4633.24      27.92      0      1
## country.item.custcode.shape          46 5703.61    1098.29      0      1
## country.item.custcode                44 5717.48    1112.16      0      1
## country.item                         24 6036.33    1431.01      0      1
## country                              14 6955.92    2350.60      0      1
##                                           LL
## country.item.custcode.shape.area.qty -2254.24
## country.item.custcode.shape.area     -2269.22
## country.item.custcode.shape          -2805.42
## country.item.custcode                -2814.38
```

```
## country.item                              -2994.06
## country                                   -3463.92
```

Generally, the model with the lowest AIC value is always listed first. From the output we can see that the model that has all 5 attributes has the lowest AIC value and is thus the best fitting model. Hence, using this method we can say all the 6 attributes listed can be considered as important features for predicting the target variable.

Balanced Vs. Unbalanced Data

Logistic Regression		
	Unbalanced Data	Balanced Data
Test Accuracy	88.8	83.8
Test Recall	58	72
Test Precision	80.5	86.3
Test Error	11.1	16.1
Area Under Curve	77.2	82

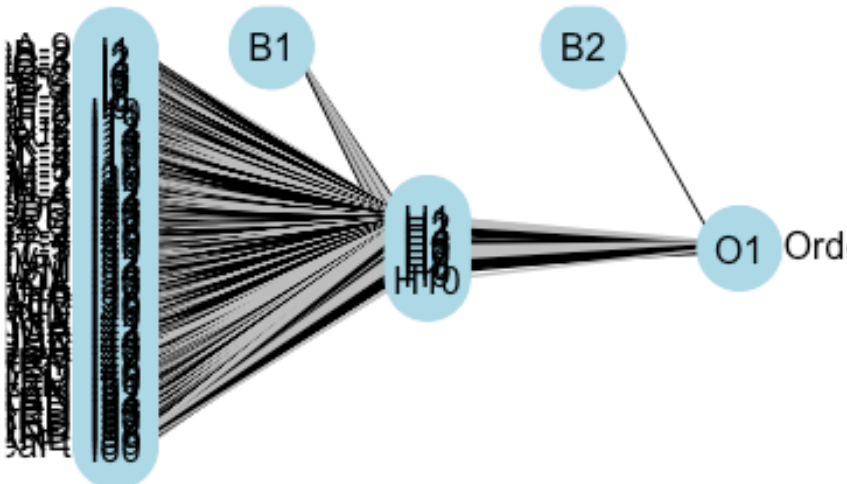
Considering recall as our metric, we witness that our Balanced Data performed better with a recall percentage of 72% in test data.

Model 4 - Neural Network

We will normalize both our Balanced and Unbalanced Data for performing Neural Network and K means.

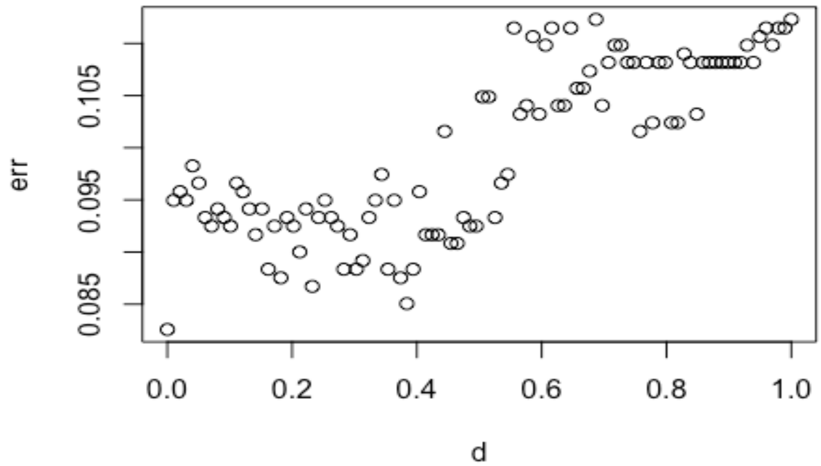
Unbalanced Data

```
set.seed(1346)
myscale <- function(x)
{
  (x - min(x))/(max(x) - min(x))
}
norm_data_unbalanced <- champo_sample_only %>%
  mutate_if(is.numeric, myscale)

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

nnModel_unbalanced <- nnet(Order_Conversion ~ ., data = train, linout = F, si
ze = 10, decay = 0.2, maxit = 3000)

plotnet(nnModel_unbalanced)
```

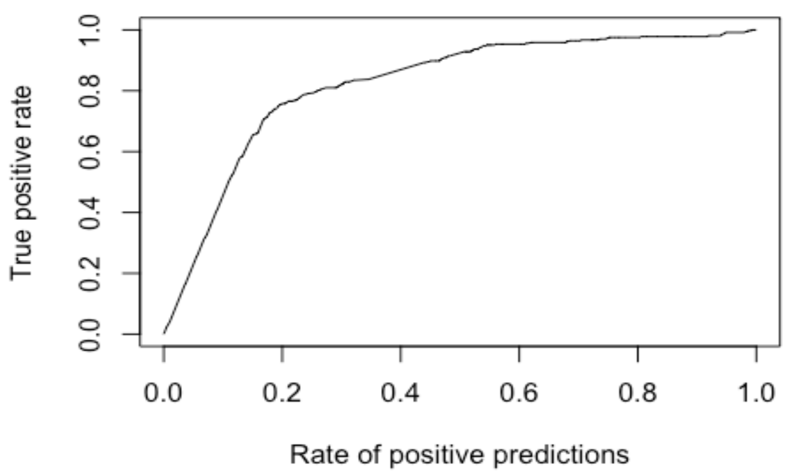



```
nn.preds <- as.factor(predict(nnModel_unbalanced, test, type = "class"))


#Confusion Matrix
CM <- table(nn.preds, test$Order_Conversion, dnn = c("Predicted", "Actual"))

error_metric = function(CM){
  TN = CM[1,1]
  TP = CM[2,2]
  FN = CM[1,2]
  FP = CM[2,1]
  accuracy = (TP+TN)/(TP+TN+FP+FN)
  recall = (TP)/(TP+FN)
  precision = (TP)/(TP+FP)
  falsePositiveRate = (FP)/(FP+TN)
  falseNegativeRate = (FN)/(FN+TP)
  error = (FP+FN)/(TP+TN+FP+FN)
  modelPerf <- list("Accuracy" = accuracy,
                    "Precision" = precision,
                    "Recall" = recall,
                    "Falsepositiverate" = falsePositiveRate,
                    "Falsenegativerate" = falseNegativeRate,
                    "Error" = error
```

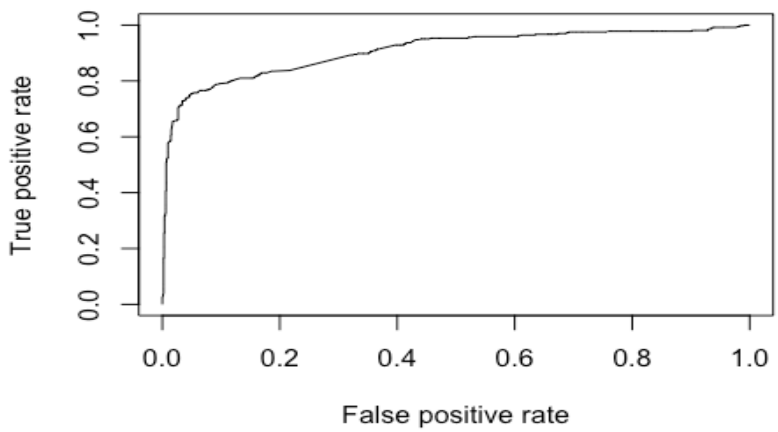
```
                    )
  return(modelPerf)
}

outPutlist <- error_metric(CM)

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

##                         Values
## 1          Accuracy 0.91685780
## 2         Precision 0.86577181
## 3            Recall 0.71074380
## 4 Falsepositiverate 0.02896452
## 5 Falsenegativerate 0.28925620
## 6             Error 0.08314220

#Determining Best Decay Parameter
set.seed(1346)
indx <- sample(2, nrow(train), replace = T, prob = c(0.7,0.3))
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 = 3000)
  pred.class <- predict(mymodel, newdata = validation, type = "class")
  err[k] <- mean(pred.class != validation$Order_Conversion)
  k <- k+1

} plot(d,err)
```



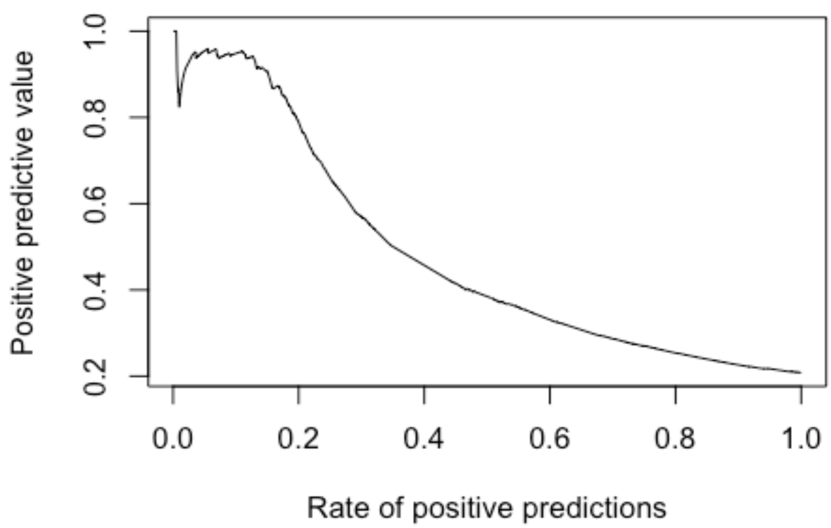

```
#Evaluation Charts
pred_test <- predict(nnModel_unbalanced, newdata = test)
pred <- prediction(pred_test, test$Order_Conversion)
#Gain Chart
perf <- performance(pred, "tpr", "rpp")
plot(perf)
```



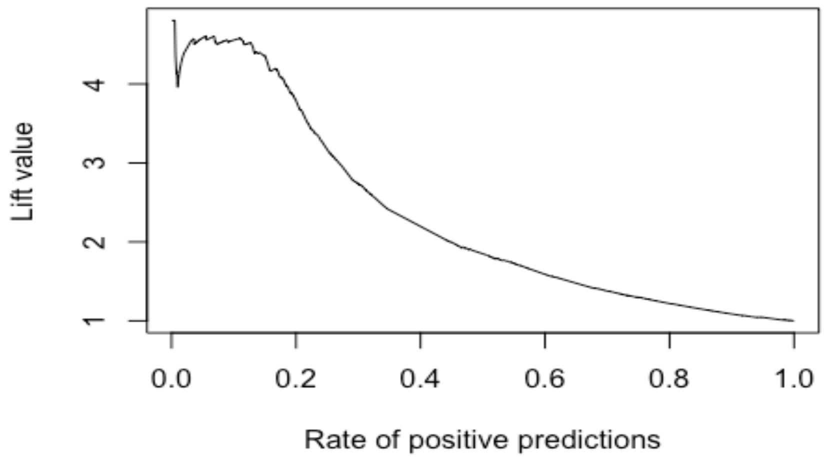

```
#ROC Curve
perf <- performance(pred, "tpr", "fpr")
plot(perf)
```




```r
# Response Chart
perf <- performance(pred, "ppv", "rpp")
plot(perf)
```



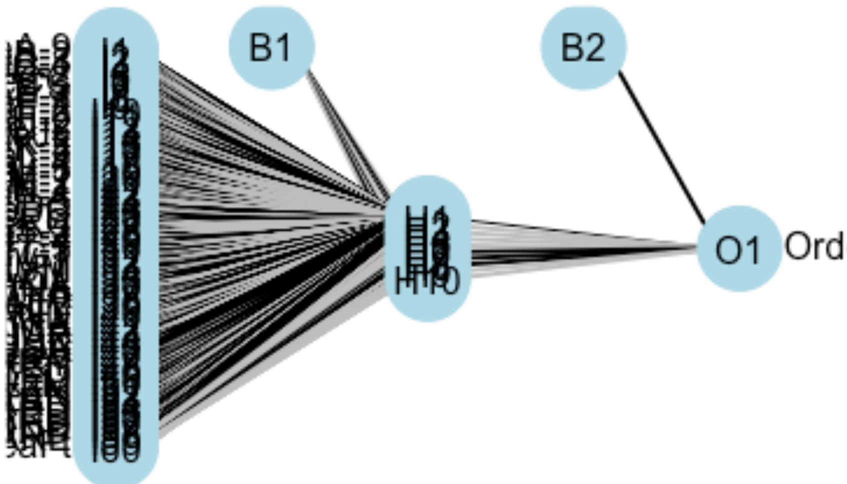

```r
# Lift Chart
perf <- performance(pred, "lift", "rpp")
plot(perf)
```




```
#Area Under Curve
auc <- unlist(slot(performance(pred, "auc"), "y.values"))
print(paste("The Area Under the Curve is ", auc))

## [1] "The Area Under the Curve is  0.908281019662761"
```

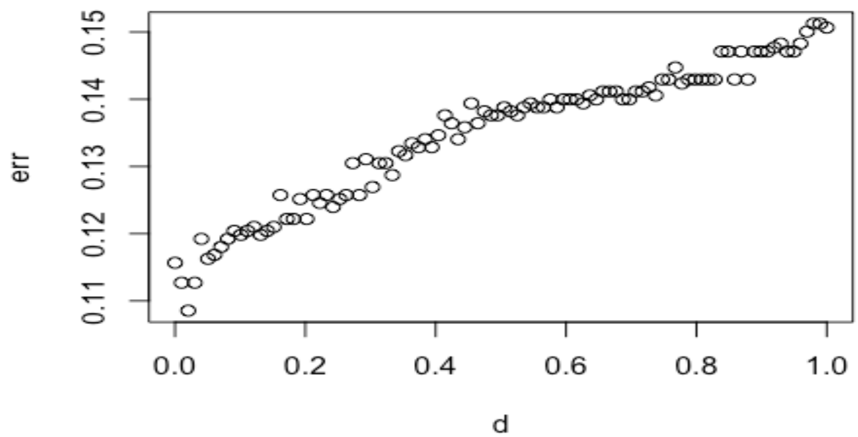
Balanced Data

```
set.seed(1346)
myscale <- function(x)
{
  (x - min(x))/(max(x) - min(x))
}
norm_data_balanced <- balanced.data %>%
  mutate_if(is.numeric, myscale)

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

nnModel_balanced <- nnet(Order_Conversion ~ ., data = train, linout = F, size
= 10, decay = 0., maxit = 3000)

plotnet(nnModel_balanced)
```

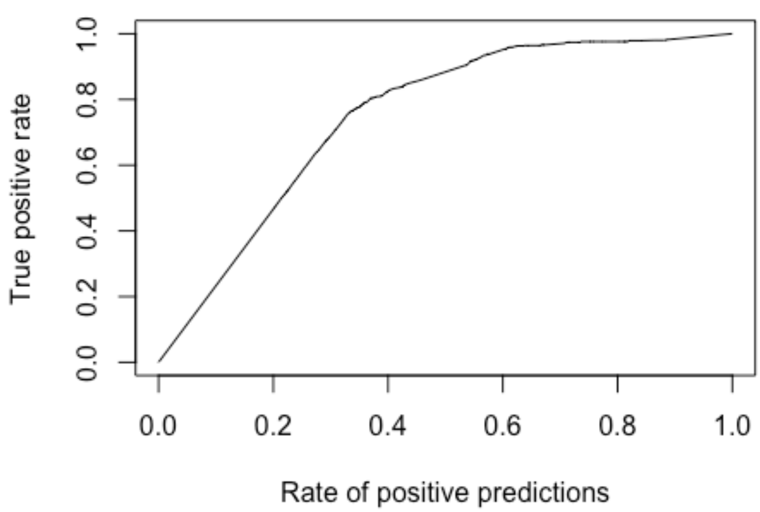



```
nn.preds <- as.factor(predict(nnModel_balanced, test, type = "class"))


#Confusion Matrix
CM <- table(nn.preds, test$Order_Conversion, dnn = c("Predicted", "Actual"))

error_metric = function(CM){
  TN = CM[1,1]
  TP = CM[2,2]
  FN = CM[1,2]
  FP = CM[2,1]
  accuracy = (TP+TN)/(TP+TN+FP+FN)
  recall = (TP)/(TP+FN)
  precision = (TP)/(TP+FP)
  falsePositiveRate = (FP)/(FP+TN)
  falseNegativeRate = (FN)/(FN+TP)
  error = (FP+FN)/(TP+TN+FP+FN)
  modelPerf <- list("Accuracy" = accuracy,
                    "Precision" = precision,
                    "Recall" = recall,
                    "Falsepositiverate" = falsePositiveRate,
                    "Falsenegativerate" = falseNegativeRate,
                    "Error" = error
```

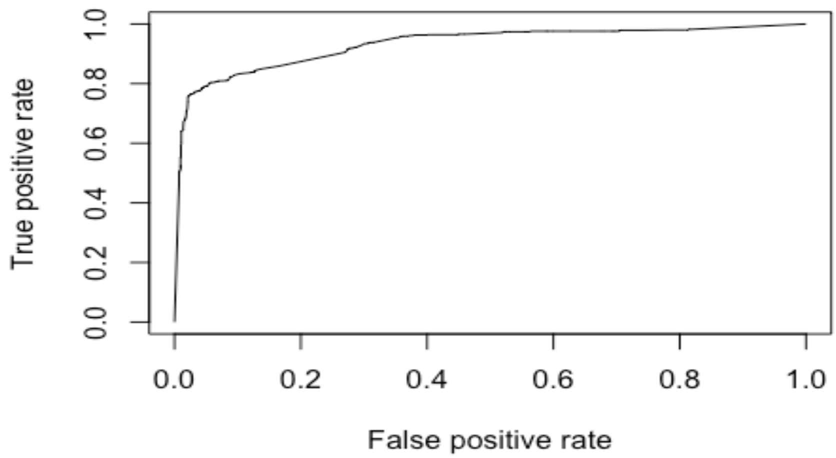
```r
                    )
  return(modelPerf)
}

outPutlist <- error_metric(CM)

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

##                          Values
## 1          Accuracy 0.88450231
## 2         Precision 0.92497069
## 3            Recall 0.78900000
## 4 Falsepositiverate 0.04634323
## 5 Falsenegativerate 0.21100000
## 6             Error 0.11549769

#Determining Best Decay Parameter
set.seed(1346)
indx <- sample(2, nrow(train), replace = T, prob = c(0.7,0.3))
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 = 3000)
  pred.class <- predict(mymodel, newdata = validation, type = "class")
  err[k] <- mean(pred.class != validation$Order_Conversion)
  k <- k+1

}

plot(d,err)
```



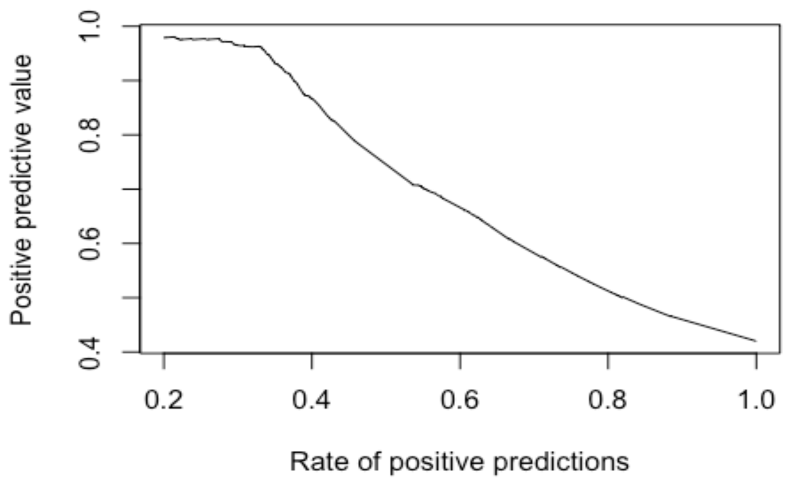

```r
#Evaluation Charts
pred_test <- predict(nnModel_balanced, newdata = test)
pred <- prediction(pred_test, test$Order_Conversion)
#Gain Chart
perf <- performance(pred, "tpr", "rpp")
plot(perf)
```



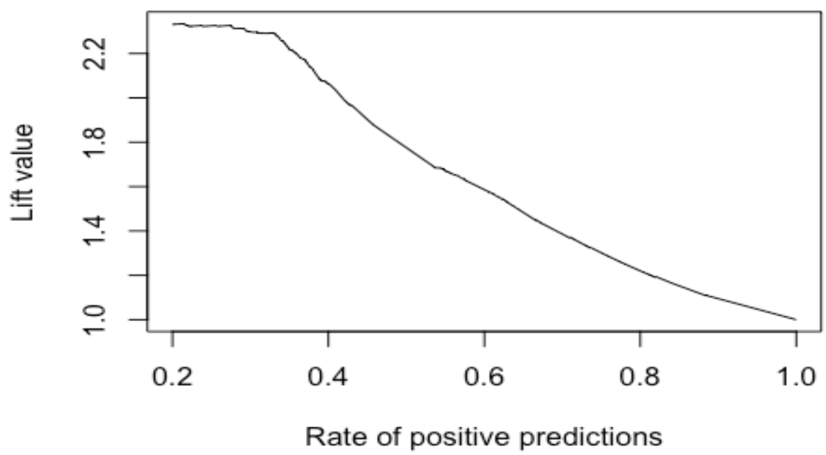

```r
#ROC Curve
perf <- performance(pred, "tpr", "fpr")
plot(perf)
```




```r
# Response Chart
perf <- performance(pred, "ppv", "rpp")
plot(perf)
```




```r
# Lift Chart
perf <- performance(pred, "lift", "rpp")
plot(perf)
```




```
#Area Under Curve
auc <- unlist(slot(performance(pred, "auc"), "y.values"))
print(paste("The Area Under the Curve is ", auc))

## [1] "The Area Under the Curve is  0.931295076031862"
```

Balanced Vs. Unbalanced Data

Neural Network		
	Unbalanced Data	**Balanced Data**
Test Accuracy	91.6	88.4
Test Recall	71	78.2
Test Precision	86.6	93.2
Test Error	8.31	11.5
Area Under Curve	90.8	94.2

Considering recall as our metric, we witness that our Balanced Data performed better with a recall percentage of 78.2% in test data.

Clustering Models such as K means and Hierarchial Clustering have been modelled for Problem 4,5 and 6. So, we will not repeat the code here.

Model Selection

Model Evaluation	
Model	Test Recall
Decision Tree	91.3
Random Forest	83.1
Logistic Regression	72
Neural Network	78.2

Decision Tree Model with 70-30 split is the best model to recommend to Champo Carpets as this yields the highest Recall (as we want to minimize FN as much as possible).

Problem 4

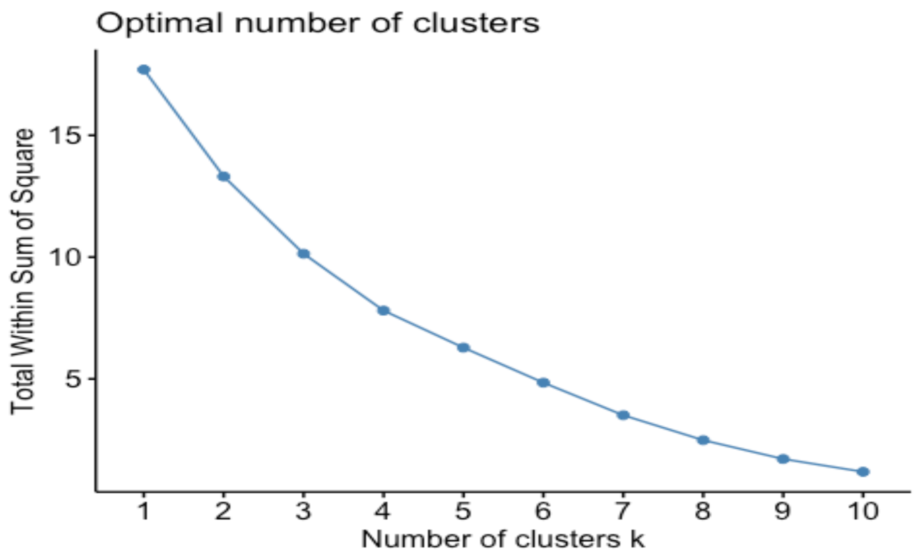
We have implemented Hierarchial Clustering to form clusters and dendograms for Problem 4 and 5. As K means has been used implemented for Problem 6, we have not repeated the code here.

Before proceeding with clustering, we will convert our row no.1 names to row labels

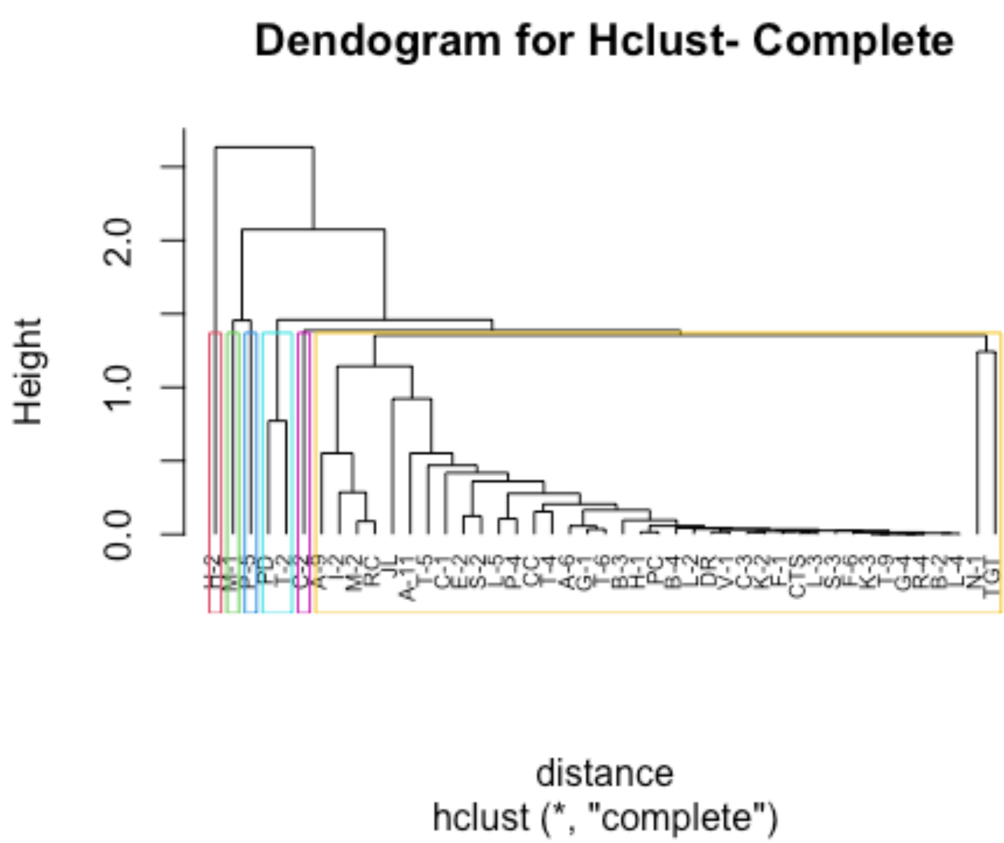
```
champo_cluster <- champo_cluster %>%
  remove_rownames %>%
  column_to_rownames(var = "Row Labels")
```

Hierarchical Clustering

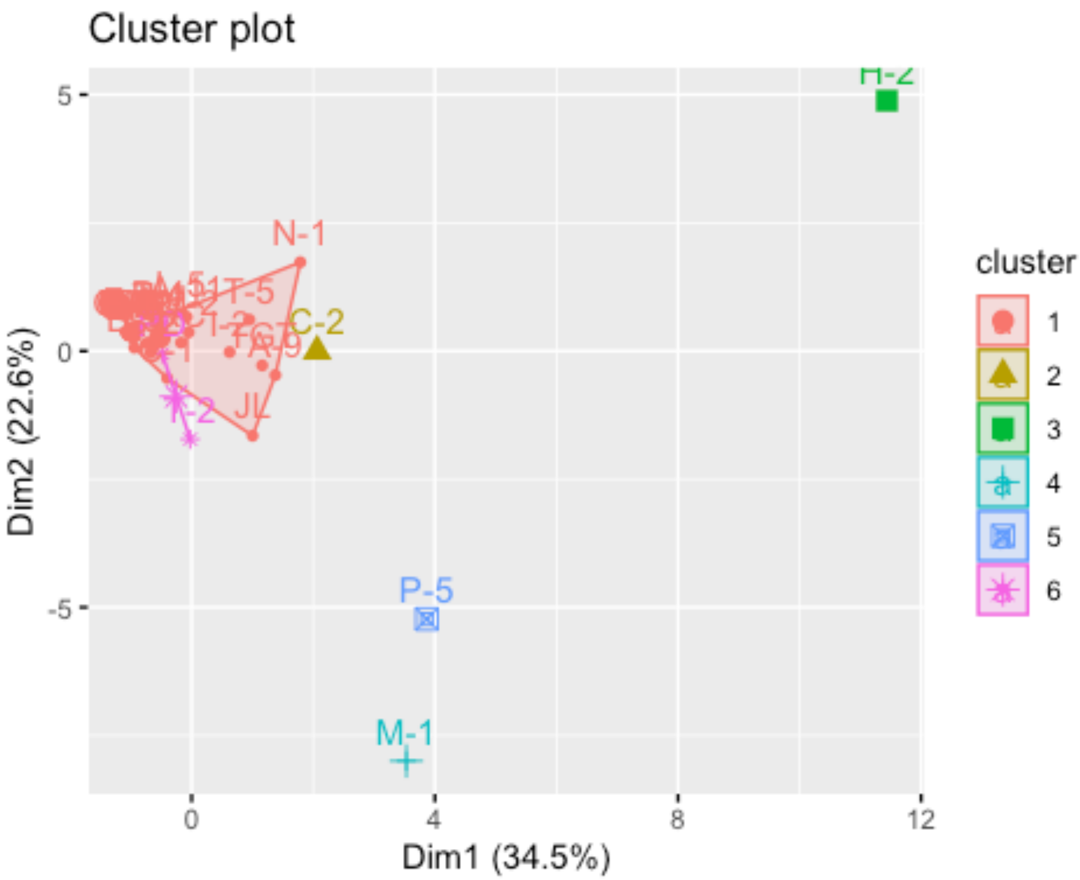
```
#Normalizing
myscale <- function(x)
{
  (x - min(x))/(max(x) - min(x))
}
cluster_data <- champo_cluster %>%
  mutate_if(is.numeric, myscale)
#Hierarchical Clustering
distance <- dist(cluster_data, method = "euclidean")
head(distance)

## [1] 0.4211715 1.1436707 0.3938603 0.4047369 0.3664961 0.5025482

h_complete <- hclust(distance, method = "complete")
#Checking the best K to plot the Hierarchical Clustering
fviz_nbclust(cluster_data, FUN = hcut, method = "wss")
```




```r
#Plotting the Hierarchical Clustering
plot(h_complete, cex= 0.6, hang = -2, main = "Dendogram for Hclust- Complete"
, labels = cluster_data$`Row Labels`)
rect.hclust(h_complete, k=6, border = 2:8)
```



```
#Cutting the tree
clusters <- cutree(h_complete, k=6)
table(clusters)

## clusters
##  1  2  3  4  5  6
## 39  1  1  1  1  2

#Visualizing the Result in a Scatter Plot
fviz_cluster(list(data = cluster_data, cluster = clusters))
```



Based on the results from above plot, it is evident that the hierarchical clustering can be used to segment customers into different clusters. The hierarchical clustering will find the shortest distance between points using Euclidean distance method. So that customers of similar purchase behavior or similar design type are being grouped together. Then using the Elbow method we can find the number of centroids. The graph shows us different trails on the number of centers that has been attempted, from which we can see that when k=6 or after 6 clusters there is no major difference.

Based on the cluster we can now segment the customers into 6 different categories.

Cluster	Country	Item	Characteristics
Cluster 1	India, USA	Hand Tufted	Rectangular Shapes
Cluster 2	USA	Hand Tufted, Handwoven, Dury	Multicolored Rectangular Shape
Cluster 3	USA	Double Back, Dury	Lighter shades of Nordic and wools
Cluster 4	USA	Knotted, Hand Tufted	Darker shades of Jute
Cluster 5	USA	Dury, Hand Tufted	Light shades of Flatwoven Cotton
Cluster 6	Belgium, Italy	Handwoven, Dury	Handwoven Rectangular shapes

Data Strategy –

1) We can group customers of similar purchase behavior based on the design type and send samples accordingly.

2) With the hierarchical clustering type it also easy to segment customers who would actually contribute towards the order conversion and the rate of sample testing being converted to an order is high.

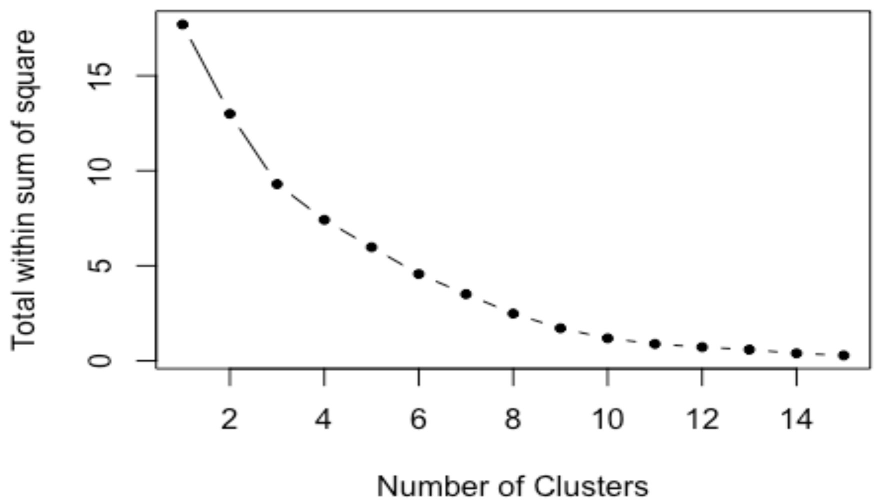
Problem 5

Hierarchical clustering would be a powerful method as it would be used to relate the points with the nearest distance and form clusters. Using euclidean distance it was easy to find the nearest point within each cluster. The complete link method has a stronger clustering structure than single, average or ward.D when we used to find the distance between the clusters.

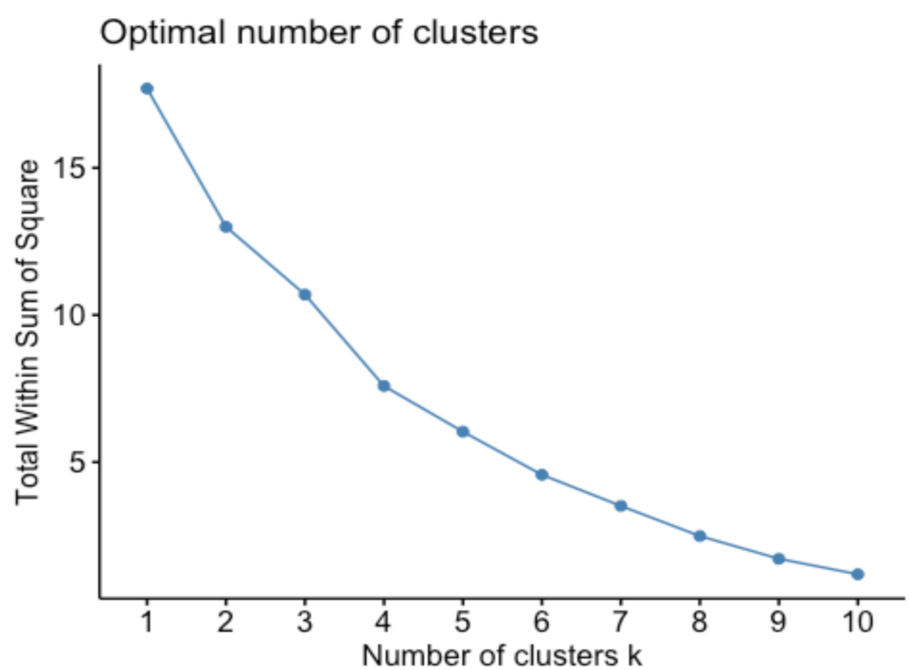
Problem 6

K means Clustering

```
#Normalizing
myscale <- function(x)
{
  (x - min(x))/(max(x) - min(x))
}
cluster_data <- champo_cluster %>%
  mutate_if(is.numeric, myscale)

set.seed(123)
 wss <- function(k) {
   kmeans(cluster_data, centers =  k, nstart = 100)$tot.withinss
 }
k.values <- 1:15
wss_values <- map_dbl(k.values, wss)
plot(k.values,wss_values, type = "b", xlab= "Number of Clusters", ylab = "Tot
al within sum of square",pch = 20, cex = 1)
```



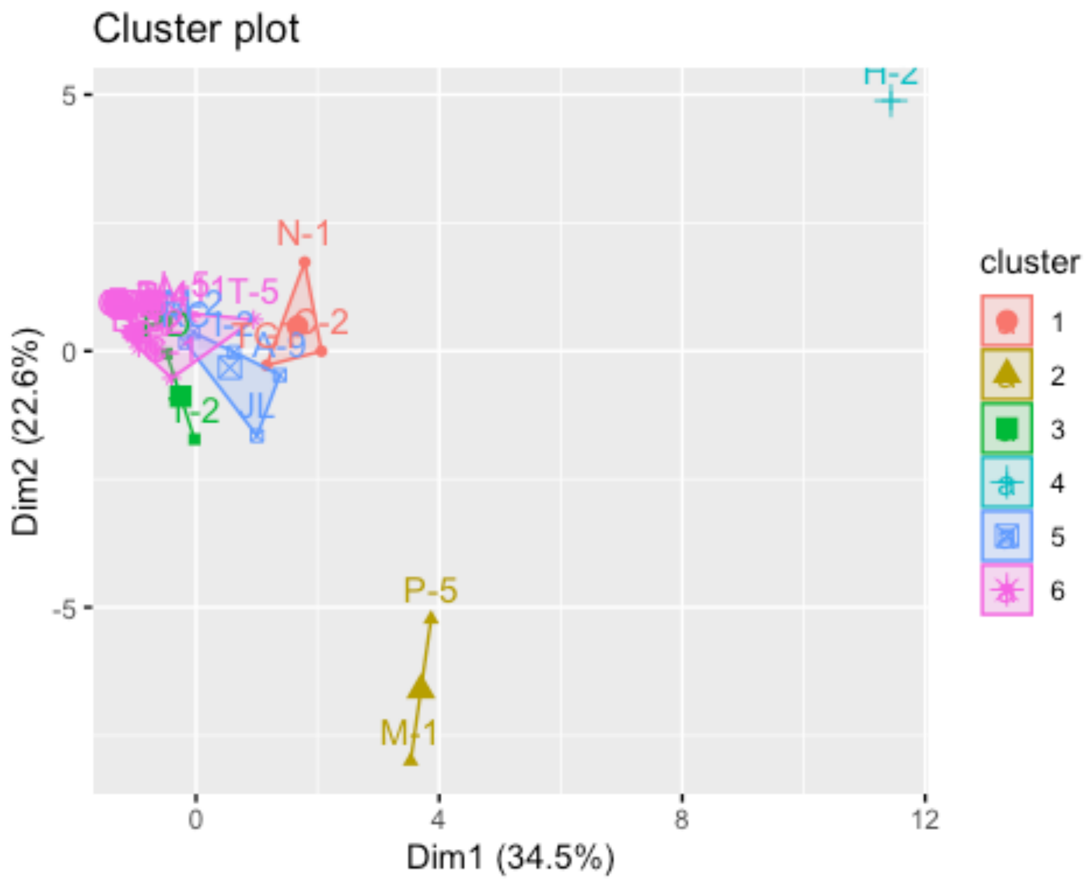

```
set.seed(123)
fviz_nbclust(cluster_data, kmeans, method = "wss")
```




```
(km6 <- kmeans(cluster_data, centers = 6, nstart = 100))

## K-means clustering with 6 clusters of sizes 3, 2, 2, 1, 5, 32
##
## Cluster means:
##   Sum of QtyRequired Sum of TotalArea Sum of Amount      DURRY   HANDLOOM
## 1         0.26036367      0.076379689    0.40698651 0.11757080 0.00000000
```

```
## 2         0.17744700      0.689929357    0.22157557 0.09457591 0.16648516
## 3         0.04533198      0.007526745    0.04263264 0.03670372 0.07187585
## 4         1.00000000      0.092998692    0.33546999 1.00000000 1.00000000
## 5         0.06028580      0.065044358    0.06897577 0.01757510 0.04290770
## 6         0.02059623      0.023717593    0.01060837 0.01831896 0.01818166
##   DOUBLE BACK   JACQUARD HAND TUFTED  HAND WOVEN    KNOTTED  GUN TUFTED
## 1  0.00000000 0.00000000  0.43693389 0.333333333 0.00000000 0.000000000
## 2  0.93123736 0.28921569  0.04160007 0.294292301 0.69080194 0.500000000
## 3  0.15223387 0.03501401  0.02171871 0.033289088 0.02041675 0.312820513
## 4  0.00000000 0.77030812  0.43852682 0.209585022 0.00000000 0.000000000
## 5  0.16120610 0.65294118  0.08062948 0.123683107 0.04312776 0.030769231
## 6  0.01255975 0.02976190  0.01337851 0.007645487 0.00294017 0.003044872
##   Powerloom Jacquard INDO TEBETAN
## 1                  0          0.0
## 2                  0          0.0
## 3                  0          0.8
## 4                  1          0.0
## 5                  0          0.0
## 6                  0          0.0
## 
## 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 
##    6    6    5    6    6    6    6    1    6    6    6    6    6    6    6 
6 
##  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 
##    6    6    4    5    5    6    6    6    6    6    6    2    5    1    6 
2 
##   PC   PD  R-4   RC  S-2  S-3  T-2  T-4  T-5  T-6  T-9  TGT  V-1 
##    6    3    6    5    6    6    3    6    6    6    6    1    6 
## 
## Within cluster sum of squares by cluster:
## [1] 1.7939465 1.0593578 0.2978700 0.0000000 0.7648458 0.6520855
##  (between_SS / total_SS =  74.2 %)
## 
## Available components:
## 
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withi
nss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
fviz_cluster(km6,data = cluster_data )
```




```
cluster_data %>%
   mutate(Cluster= km6$cluster) %>%
  group_by(Cluster) %>%
  summarise_all("mean")

## # A tibble: 6 × 14
##   Cluster `Sum of QtyRequired` `Sum of TotalArea` `Sum of Amount`  DURRY H
ANDLOOM
##     <int>                <dbl>              <dbl>           <dbl>  <dbl>
<dbl>
## 1       1               0.260              0.0764           0.407  0.118
0
## 2       2               0.177              0.690            0.222  0.0946
0.166
## 3       3               0.0453             0.00753          0.0426 0.0367
0.0719
## 4       4               1                  0.0930           0.335  1
1
## 5       5               0.0603             0.0650           0.0690 0.0176
0.0429
## 6       6               0.0206             0.0237           0.0106 0.0183
0.0182
## # … with 8 more variables: DOUBLE BACK <dbl>, JACQUARD <dbl>,
```

```
## #   HAND TUFTED <dbl>, HAND WOVEN <dbl>, KNOTTED <dbl>, GUN TUFTED <dbl>,
## #   Powerloom Jacquard <dbl>, INDO TEBETAN <dbl>
```

Optimal Clusters - Based on the results from the k-means clustering we find that the customers are being segmented into 6 different clusters as the optimal number of clusters was 6 using the elbow method.

Cluster 1 - C-2, N-1, TGT

Cluster 2 - M-1, P-5

Cluster 3 - PD, T-2

Cluster 4 - H-2

Cluster 5 - A-9, I-2, JL, M-2, RC

Cluster 6 - A-11, A-6, B-2, B-3, B-4, C-1, C-3, CC, CTS, DR, E-2, F-1, F-6, G-1, G-4, H-1, K-2, K-3,L-2, L-3, L-4, L-5, P-4, PC, R-4, S-2, S-3, T-4, T-5, T-6, T-9, V-1

Cluster Characteristics -

Cluster 1: The highest revenue being generated from this cluster is USA who choice of preference is Hand tufted, Durry, Hand woven, knotted and powerloom jacquard. Hand Tufted ordered by customer C-2 in large quantity of different shades led to major revenue, generation.

Cluster 2: The highest revenue generated from this cluster is USA preferring Double back, Durry, Knotted. Customers prefers lighter shades of Nordic and wools in Double back, Durry, Knotted.

Cluster 3: Majority of the customers are from Belgium and Italy who place orders in small quantity. Customers preference are Handwoven and Dury of rectangular shapes.

Cluster 4: The highest revenue is being generated from USA with preference of Dury and Hand tufted. Lighter shades of flatwoven cotton have a high preference by the customers.

Cluster 5: Majority of the customers are from Australia, Romania, UK with higher preference for Double back, Durry, Hand tufted and hand woven.

Cluster 6: Highest revenue generating customers are India and USA with a preference of Hand Tufted, Double back and Dury. Customers prefer handwoven rectangular shaped carpets.

Variable Significance - During the Cluster Character Analysis, we witnessed that the variables that contributed towards identifying similarity were

1. CustomerCode 2. ITEMNAME 3. DesignName 4. ShapeName 5. QtyRequired

6. CountryName 7. ColorName

Problem 7

We have implemented our own Collaborative Filtering techniqueas a recommender system using Cosine Similarity measure to find nearest neighbors.

Before proceeding with cosine similarity, we will convert our row no.1 names to row labels

```
champo_rec <- champo_rec %>%
  remove_rownames %>%
  column_to_rownames(var = "Customer")

#Cosine helper function
cluster_rec <- champo_rec
getcosine <- function(x,y)
{
  this.cosine <- sum(x*y) / (sqrt(sum(x*x)) * sqrt(sum(y*y)))
  return(this.cosine)
}

#Placeholer to store the data frame
cluster_data_similarity <- matrix(NA, nrow = ncol(cluster_rec), ncol = ncol(c
luster_rec), dimnames = list(colnames(cluster_rec), colnames(cluster_rec)))
#Filling the spaces with cosine similarity
for (i in 1:nrow(cluster_rec)) {
  for (j in 1:ncol(cluster_rec)) {
    cluster_data_similarity[i,j] <- getcosine(as.matrix(cluster_rec[i]), as.m
atrix(cluster_rec[j]))
  }
}
cluster_data_similarity <- as.data.frame(cluster_data_similarity)
# Getting the top 10 neighbors for each
cluster_data_neighbors <- matrix(NA, nrow = ncol(cluster_data_similarity), nc
ol=11, dimnames = list(colnames(cluster_data_similarity)))
#Finding the Neighbors
for (i in 1:ncol(cluster_rec))
  {
  cluster_data_neighbors[i,] <- (t(head(n=11,rownames(cluster_data_similarity
[order(cluster_data_similarity[,i], decreasing = TRUE),][i]))))

}
cluster_data_neighbors

##              [,1]           [,2]          [,3]          [,4]
## Hand Tufted  "Hand Tufted"  "BLUE"        "Navy"        "NAVY"
## Double Wowen "Double Wowen" "Rectangle"   "Durry"       "Double Back"
## Durry        "Durry"        "Rectangle"   "Round"       "Navy"
## Double Back  "Double Back"  "Knotted"     "Square"      "Double Wowen"
## Knotted      "Knotted"      "Square"      "Double Back" "Double Wowen"
## Jacquared    "Jacquared"    "NEUTRAL"     "Rectangle"   "PINK"
## Handloom     "Handloom"     "Round"       "Other"       "Durry"
## Other        "Other"        "Round"       "Durry"       "Navy"
```

```
## Rectangle    "Rectangle"    "Durry"        "BLUE"        "Navy"
## Square       "Square"       "Knotted"      "Double Back" "Double Wowen"
## Round        "Round"        "Other"        "Durry"       "Navy"
## Purple       "Purple"       "Jacquared"    "Double Back" "Double Wowen"
## Gray         "Gray"         "BLUSH PINK"   "PINK"        "Rectangle"
## Navy         "Navy"         "NAVY"         "Round"       "Rectangle"
## PINK         "PINK"         "BLUSH PINK"   "Gray"        "Jacquared"
## BLUE         "BLUE"         "Rectangle"    "Navy"        "NAVY"
## BLUSH PINK   "BLUSH PINK"   "PINK"         "Gray"        "Rectangle"
## NEUTRAL      "NEUTRAL"      "Jacquared"    "PINK"        "Hand Tufted"
## TAN          "TAN"          "Hand Tufted"  "Gray"        "Rectangle"
## NAVY         "Navy"         "NAVY"         "Round"       "Rectangle"
##              [,5]           [,6]            [,7]           [,8]
## Hand Tufted  "Rectangle"    "Durry"         "Round"        "Handloom"
## Double Wowen "Gray"         "Knotted"       "Jacquared"    "Square"
## Durry        "NAVY"         "Other"         "Handloom"     "BLUE"
## Double Back  "Jacquared"    "Gray"          "Rectangle"    "Purple"
## Knotted      "Jacquared"    "Rectangle"     "Gray"         "BLUE"
## Jacquared    "Durry"        "Handloom"      "Navy"         "NAVY"
## Handloom     "Navy"         "NAVY"          "Rectangle"    "BLUE"
## Other        "NAVY"         "Handloom"      "Rectangle"    "BLUE"
## Rectangle    "NAVY"         "Round"         "Other"        "Hand Tufted"
## Square       "Jacquared"    "Rectangle"     "BLUE"         "Gray"
## Round        "NAVY"         "Handloom"      "Rectangle"    "BLUE"
## Purple       "PINK"         "Handloom"      "Rectangle"    "Hand Tufted"
## Gray         "Double Wowen" "Durry"         "BLUE"         "Double Back"
## Navy         "Durry"        "BLUE"          "Other"        "Handloom"
## PINK         "NEUTRAL"      "Rectangle"     "Double Wowen" "Double Back"
## BLUE         "Hand Tufted"  "Durry"         "Round"        "Other"
## BLUSH PINK   "Durry"        "Double Wowen"  "Navy"         "NAVY"
## NEUTRAL      "Double Wowen" "Rectangle"     "Navy"         "NAVY"
## TAN          "Square"       "Double Wowen"  "Durry"        "Double Back"
## NAVY         "Durry"        "BLUE"          "Other"        "Handloom"
##              [,9]          [,10]          [,11]
## Hand Tufted  "Other"       "Jacquared"    "Double Wowen"
## Double Wowen "PINK"        "Navy"         "NAVY"
## Durry        "Jacquared"   "Hand Tufted"  "Double Wowen"
## Double Back  "PINK"        "Handloom"     "BLUE"
## Knotted      "Durry"       "Handloom"     "PINK"
## Jacquared    "Round"       "BLUE"         "Other"
## Handloom     "Jacquared"   "Hand Tufted"  "Double Wowen"
## Other        "Jacquared"   "Hand Tufted"  "Double Wowen"
## Rectangle    "Handloom"    "Double Wowen" "Jacquared"
## Square       "Durry"       "Handloom"     "Hand Tufted"
## Round        "Jacquared"   "Hand Tufted"  "Double Wowen"
## Purple       "Gray"        "Knotted"      "BLUE"
## Gray         "Knotted"     "Square"       "Jacquared"
## Navy         "Hand Tufted" "Jacquared"    "Double Wowen"
## PINK         "Purple"      "Durry"        "Hand Tufted"
## BLUE         "Handloom"    "Jacquared"    "Gray"
```

```
## BLUSH PINK   "BLUE"        "Jacquared"    "Hand Tufted"
## NEUTRAL      "Knotted"     "Purple"       "Gray"
## TAN          "Knotted"     "Jacquared"    "Handloom"
## NAVY         "Hand Tufted" "Jacquared"    "Double Wowen"
```

Based on an aggregate of client purchase history and the collaborative filtering approach. We used Cosine similarity to find the closest neighbors for each client. The following are our top three recommendations:

For consumers who want to buy hand tufted, we propose Dury, Handloom, or Jacquard in darker colors like blue or navy, in either rectange or circular shapes.

Customers who purchase Durry should also consider Handloom, Jacquard, or Handtufted in Navy or Blue, with a rectangular or circular form.

Customers looking for a double back should look for knotted, double woven, or jacquard in square or rectangle shapes and darker colors like gray or purple.

Problem 8

1. **Model Selection** - Champo Carpets can use Decision Tree Models to identify the important attributes that determine the conversion of samples sent to the customers. Decision trees do not provide the answer to the problem Champo carpets are facing but it will definitely help the management determine which alternative will yield the greatest conversion rate, given a particular choice point.From our Analysis using Decision Tree, Champo Carpets should focus more on the following variables as these have majority weightage to contribute towards conversion rate - 1. AreaFt 2. CustomerCode 3. ITEM_NAME 4. CountryName 5. QtyRequired 6. ShapeName

2. **Customer Segmentation** - Champo Carpets can use K means clustering model to understand the demographics of their customer base to better target segments that contribute more towards conversion rate.

Champo Carpets can group customers of similar purchase behavior based on the design type and send samples accordingly. With the hierarchical clustering type it also easy to segment customers who would actually contribute towards the order conversion and the rate of sample testing being converted to an order is high.

3. **Collaborative Filtering** - The rules generated through the Collaborative Filter technique using cosine similarity can greatly help Champo Carpets increase their customer conversion rate. The rules are -

For consumers who want to buy hand tufted, we propose Dury, Handloom, or Jacquard in darker colors like blue or navy, in either rectange or circular shapes. Customers who purchase Durry should also consider Handloom, Jacquard, or Handtufted in Navy or Blue, with a rectangular or circular form. Customers looking for a double back should look for knotted, double woven, or jacquard in square or rectangle shapes and darker colors like gray or purple.