HW5_IDS_572_NY_SB_JONA

Navya Yadagiri 674788385, Sayali Bonawale 656488690 ,Jona 651224838

4/25/2022

Problem Statement:

- Champo carpets is looking for a cost-efficient way of selecting appropriate sample designs that could generate maximum revenue for the organization.
- Identify the customer segments and their tastes and past preferences and trends to lead towards better conversion rate.
- To identify the most important customer and the most important products and find a way to connect the two using suitable attributes from data and appropriate analytical models
- Identify ideal set if samples to customers and help them increase the conversion rate.
- **Challenges**: Low conversion rate of sample carpets sent by them to their customers.
- The process of selection of Champ carpets samples designs were done in various ways and the process itself is costly and elaborate.

Importing the library

```
library(lubridate)

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union

library(skimr)

## Warning: package 'skimr' was built under R version 4.1.3

library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
#library(devtools)
library(tidyverse)

## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.5     v stringr 1.4.0
## v tidyr   1.1.4     v forcats 0.5.1
## v readr   2.0.2

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x lubridate::as.difftime() masks base::as.difftime()
## x lubridate::date()        masks base::date()
## x dplyr::filter()          masks stats::filter()
## x lubridate::intersect()   masks base::intersect()
## x dplyr::lag()             masks stats::lag()
## x lubridate::setdiff()     masks base::setdiff()
## x lubridate::union()       masks base::union()

library(psych)

##
## Attaching package: 'psych'

## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha

library(randomForest)

## Warning: package 'randomForest' was built under R version 4.1.3

## randomForest 4.7-1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:psych':
##
##     outlier

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
library("tidycomm")

## Warning: package 'tidycomm' was built under R version 4.1.3

##
## Attaching package: 'tidycomm'

## The following object is masked from 'package:psych':
##
##     describe

library(visdat)

## Warning: package 'visdat' was built under R version 4.1.3

library("funModeling")

## Warning: package 'funModeling' was built under R version 4.1.3

## Loading required package: Hmisc

## Loading required package: lattice

## Loading required package: survival

## Loading required package: Formula

##
## Attaching package: 'Hmisc'

## The following object is masked from 'package:tidycomm':
##
##     describe

## The following object is masked from 'package:psych':
##
##     describe

## The following objects are masked from 'package:dplyr':
##
##     src, summarize

## The following objects are masked from 'package:base':
##
##     format.pval, units

## funModeling v.1.9.4 :)
## Examples and tutorials at livebook.datascienceheroes.com
##  / Now in Spanish: librovivodecienciadedatos.ai

library("Hmisc")
library("rpart")
library("caret")
```

```
##
## Attaching package: 'caret'

## The following object is masked from 'package:survival':
##
##     cluster

## The following object is masked from 'package:purrr':
##
##     lift

library("rpart.plot")
```

Import the dataset and Performing basic Exploratory Data Analysis

```
#Importing data that has order and sample data
original_dataset <- readxl::read_excel("Champo Carpets.xlsx", sheet = "Raw
Data-Order and Sample");

# There are 16 columns
colnames(original_dataset)

##  [1] "OrderType"       "OrderCategory"   "CustomerCode"    "CountryName"
##  [5] "CustomerOrderNo" "Custorderdate"   "UnitName"        "QtyRequired"
##  [9] "TotalArea"       "Amount"          "ITEM_NAME"       "QualityName"
## [13] "DesignName"      "ColorName"       "AreaFt"          "ShapeName"

summary(original_dataset)

##   OrderType          OrderCategory      CustomerCode       CountryName
##  Length:18955       Length:18955       Length:18955       Length:18955
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##  CustomerOrderNo    Custorderdate                     UnitName
##  Length:18955       Min.   :2017-01-16 00:00:00   Length:18955
##  Class :character   1st Qu.:2018-02-27 00:00:00   Class :character
##  Mode  :character   Median :2018-12-01 00:00:00   Mode  :character
##                     Mean   :2018-10-18 15:28:02
##                     3rd Qu.:2019-07-05 00:00:00
##                     Max.   :2020-02-14 00:00:00
##   QtyRequired        TotalArea           Amount             ITEM_NAME
##  Min.   :   1.00   Min.   :   0.04   Min.   :     0.0   Length:18955
##  1st Qu.:   1.00   1st Qu.:   4.00   1st Qu.:     0.0   Class :character
##  Median :   4.00   Median :  15.00   Median :   200.6   Mode  :character
##  Mean   :  31.42   Mean   :  36.15   Mean   :  1657.6
##  3rd Qu.:  13.00   3rd Qu.:  54.00   3rd Qu.:   977.1
##  Max.   :6400.00   Max.   :1024.00   Max.   :599719.7
##  QualityName         DesignName         ColorName              AreaFt
##  Length:18955       Length:18955       Length:18955       Min.   :  0.4444
```

```
##  Class :character   Class :character   Class :character   1st Qu.:  8.4375
##  Mode  :character   Mode  :character   Mode  :character   Median : 35.0000
##                                                           Mean   : 44.4695
##                                                           3rd Qu.: 64.7361
##                                                           Max.   :645.7222
##   ShapeName
##  Length:18955
##  Class :character
##  Mode  :character
##
##
##

#The customer order no is assigned a character data type, which in ideal case
should be an integer
#original_dataset$CustomerOrderNo
glimpse(original_dataset)

## Rows: 18,955
## Columns: 16
## $ OrderType       <chr> "Area Wise", "Area Wise", "Area Wise", "Area
Wise", "A~
## $ OrderCategory   <chr> "Order", "Order", "Order", "Order", "Order",
"Order", ~
## $ CustomerCode    <chr> "H-1", "H-1", "H-1", "H-1", "H-1", "H-1", "H-1",
"H-1"~
## $ CountryName     <chr> "USA", "USA", "USA", "USA", "USA", "USA", "USA",
"USA"~
## $ CustomerOrderNo <chr> "1873354.0", "1873354.0", "1873354.0",
"1918436.0", "1~
## $ Custorderdate   <dttm> 2017-01-16, 2017-01-16, 2017-01-16, 2017-02-01,
2017-~
## $ UnitName        <chr> "Ft", "Ft", "Ft", "Ft", "Ft", "Ft", "Ft", "Ft",
"Ft", ~
## $ QtyRequired     <dbl> 2, 2, 2, 5, 5, 4, 6, 16, 2, 4, 2, 8, 2, 2, 5, 2,
5, 2,~
## $ TotalArea       <dbl> 6.00, 9.00, 54.00, 54.00, 71.25, 71.25, 128.25,
128.25~
## $ Amount          <dbl> 12.00, 18.00, 108.00, 270.00, 356.25, 285.00,
769.50, ~
## $ ITEM_NAME       <chr> "HAND TUFTED", "HAND TUFTED", "HAND TUFTED", "HAND
TUF~
## $ QualityName     <chr> "TUFTED 30C HARD TWIST", "TUFTED 30C HARD TWIST",
"TUF~
## $ DesignName      <chr> "OLD LONDON [3715]", "OLD LONDON [3715]", "OLD
LONDON ~
## $ ColorName       <chr> "BEIGE", "BEIGE", "BEIGE", "BEIGE", "BEIGE",
"BEIGE", ~
## $ AreaFt          <dbl> 6.00, 9.00, 54.00, 54.00, 71.25, 71.25, 128.25,
128.25~
```

```
## $ ShapeName       <chr> "REC", "REC", "REC", "REC", "REC", "REC", "REC",
"REC"~

#As we can see there are in total 9 missing values in the column CountryName
and CustomerOrderNo
#describe(original_dataset)
sum(is.na(original_dataset))

## [1] 9

#Making the copy of the original dataset
dataset <- data.frame(original_dataset)

#Identifying categorical and numerical variables in the main dataset
colsCategorical <- c(1:4,7,11:15)

# Some of the variables need to be converted from character to categorical
dataset[colsCategorical] <- lapply(dataset[colsCategorical], as.factor)

#Filtering out data containing only ORDER DATA from the original dataset
#order_only_data <- dataset %>% filter(dataset, OrderCategory == Order)

###---ORDER DATA--------
Order_only_data <- readxl::read_excel("Champo Carpets.xlsx", sheet = "Data
Order ONLY");
head(Order_only_data)

## # A tibble: 6 x 12
##   CustomerCode CountryName QtyRequired TotalArea Amount ITEM_NAME
QualityName
##   <chr>        <chr>             <dbl>     <dbl>  <dbl> <chr>       <chr>
## 1 H-1          USA                   6     128.    770. HAND TUFTED TUFTED
30C ~
## 2 H-1          USA                   6     117     702  HAND TUFTED TUFTED
60C
## 3 H-1          USA                   7      88     616  HAND TUFTED TUFTED
60C
## 4 H-1          USA                   7      88     616  HAND TUFTED TUFTED
60C
## 5 H-1          USA                   5     117     585  HAND TUFTED TUFTED
60C
## 6 H-1          USA                   6      71.2   428. HAND TUFTED TUFTED
60C
## # ... with 5 more variables: DesignName <chr>, ColorName <chr>,
## #   ShapeName <chr>, AreaFt <dbl>, AreaMtr <dbl>

#This data has an addition column - Customer code - telling what type of
cutomer segment it is
```

```
#There are around 12 columns in the order only data
colnames(Order_only_data)

##  [1] "CustomerCode" "CountryName"  "QtyRequired"  "TotalArea"    "Amount"
##  [6] "ITEM_NAME"    "QualityName"  "DesignName"   "ColorName"
"ShapeName"
## [11] "AreaFt"       "AreaMtr"

summary(Order_only_data)

##  CustomerCode       CountryName         QtyRequired        TotalArea
##  Length:13135       Length:13135       Min.   :   1.00   Min.   :   0.04
##  Class :character   Class :character   1st Qu.:   3.00   1st Qu.:   5.80
##  Mode  :character   Mode  :character   Median :   8.00   Median :  24.00
##                                        Mean   :  44.46   Mean   :  44.73
##                                        3rd Qu.:  20.00   3rd Qu.:  80.00
##                                        Max.   :6400.00   Max.   :1024.00
##      Amount           ITEM_NAME          QualityName         DesignName
##  Min.   :     0.0   Length:13135       Length:13135       Length:13135
##  1st Qu.:   163.2   Class :character   Class :character   Class :character
##  Median :   590.6   Mode  :character   Mode  :character   Mode  :character
##  Mean   :  2392.0
##  3rd Qu.:  1540.0
##  Max.   :599719.7
##   ColorName          ShapeName             AreaFt           AreaMtr
##  Length:13135       Length:13135       Min.   :  0.4444   Min.   : 0.040
##  Class :character   Class :character   1st Qu.: 15.0000   1st Qu.: 1.350
##  Mode  :character   Mode  :character   Median : 40.0000   Median : 3.600
##                                        Mean   : 54.6224   Mean   : 4.952
##                                        3rd Qu.: 80.0000   3rd Qu.: 7.200
##                                        Max.   :645.7222   Max.   :60.000

#There are few variables that needs their datatypes to be changed
categorical_variablesOD <- c(1,2,6,7,8,9,10)
Order_only_data[categorical_variablesOD] <-
lapply(Order_only_data[categorical_variablesOD], as.factor)

glimpse(Order_only_data)

## Rows: 13,135
## Columns: 12
## $ CustomerCode <fct> H-1, H-1, H-1, H-1, H-1, H-1, H-1, H-1, H-1, H-1, H-
1, H-~
## $ CountryName  <fct> USA, USA, USA, USA, USA, USA, USA, USA, USA, USA,
USA, US~
## $ QtyRequired  <dbl> 6, 6, 7, 7, 5, 6, 35, 5, 4, 7, 6, 4, 2, 2, 2, 2, 2,
2, 2,~
## $ TotalArea    <dbl> 128.2500, 117.0000, 88.0000, 88.0000, 117.0000,
71.2500, ~
## $ Amount       <dbl> 769.500, 702.000, 616.000, 616.000, 585.000, 427.500,
393~
```

```
## $ ITEM_NAME    <fct> HAND TUFTED, HAND TUFTED, HAND TUFTED, HAND TUFTED,
HAND ~
## $ QualityName  <fct> TUFTED 30C HARD TWIST, TUFTED 60C, TUFTED 60C, TUFTED
60C~
## $ DesignName   <fct> OLD LONDON [3715], DUDLEY [9012], WEMBLY [CC-206],
SYMPHO~
## $ ColorName    <fct> GREEN/IVORY, BEIGE, BEIGE/SAGE, CHARCOAL, NAVY/BEIGE,
BRO~
## $ ShapeName    <fct> REC, REC, REC, REC, REC, REC, REC, REC, REC, REC,
REC, RE~
## $ AreaFt       <dbl> 128.2500, 117.0000, 88.0000, 88.0000, 117.0000,
71.2500, ~
## $ AreaMtr      <dbl> 11.5425, 10.5300, 7.9200, 7.9200, 10.5300, 6.4125,
1.0125~
```

```
sum(is.na(Order_only_data))
```

```
## [1] 0
```

```
####----SAMPLE DATA -----

#The sample data contains predict varable whether the sample has converted to
an order or not

#If its 1 - Converted
#If its 0 - Not Converted

sample_only_dataset <- readxl::read_excel("Champo Carpets.xlsx", sheet =
"Data on Sample ONLY");
names(sample_only_dataset)[names(sample_only_dataset) == 'Order Conversion']
<- "Order_Conversion"
names(sample_only_dataset)[names(sample_only_dataset) == 'Hand Tufted'] <-
"Hand_Tufted"
names(sample_only_dataset)[names(sample_only_dataset) == 'Double Back'] <-
"Double_Back"
names(sample_only_dataset)[names(sample_only_dataset) == 'Hand Woven'] <-
"Hand_Woven"

sample_only_dataset$Order_Conversion <-
as.factor(sample_only_dataset$Order_Conversion)
sample_only_dataset$CustomerCode <-
as.factor(sample_only_dataset$CustomerCode)
sample_only_dataset$ShapeName <- as.factor(sample_only_dataset$ShapeName)

sample_random <- sample_only_dataset[sample(nrow(sample_only_dataset)),]
sample_random <- subset (sample_random, select = -c(USA, UK, Italy, Belgium,
Romania, Australia, India))

#Add Column Corresponding to the countries
sample_only_dataset$Poland<-
```

```
ifelse(sample_only_dataset$CountryName=="POLAND",1,0)
sample_only_dataset$Brazil<-
ifelse(sample_only_dataset$CountryName=="BRAZIL",1,0)
sample_only_dataset$Canada<-
ifelse(sample_only_dataset$CountryName=="CANADA",1,0)
sample_only_dataset$Israel<-
ifelse(sample_only_dataset$CountryName=="ISRAEL",1,0)
sample_only_dataset$China<-
ifelse(sample_only_dataset$CountryName=="CHINA",1,0)
sample_only_dataset$South_Africa<-
ifelse(sample_only_dataset$CountryName=="SOUTH AFRICA",1,0)
sample_only_dataset$UAE<-ifelse(sample_only_dataset$CountryName=="UAE",1,0)

sample_only_dataset$USA<-
ifelse(sample_only_dataset$CountryName=="POLAND",1,0)
sample_only_dataset$UK<-ifelse(sample_only_dataset$CountryName=="BRAZIL",1,0)
sample_only_dataset$Italy<-
ifelse(sample_only_dataset$CountryName=="CANADA",1,0)
sample_only_dataset$Belgium<-
ifelse(sample_only_dataset$CountryName=="ISRAEL",1,0)
sample_only_dataset$Romania<-
ifelse(sample_only_dataset$CountryName=="CHINA",1,0)
sample_only_dataset$Australia<-ifelse(sample_only_dataset$CountryName=="SOUTH
AFRICA",1,0)
sample_only_dataset$India<-ifelse(sample_only_dataset$CountryName=="UAE",1,0)

levels(sample_only_dataset$Order_Conversion) <- c("Not
Converted","Converted")
```

Performing Univariant Analysis on the raw and Order data:

```
#Describing the categorical variables
#describe_cat(original_dataset)

attach(dataset)

#Identifying the different kinds of orders in each country and for each
customer segments

levels(CustomerCode)

##  [1] "A-11" "A-6"  "A-9"  "B-2"  "B-3"  "B-4"  "C-1"  "C-2"  "C-3"  "CC"
## [11] "CTS"  "DR"   "E-2"  "F-1"  "F-2"  "F-6"  "G-1"  "G-4"  "H-1"  "H-2"
## [21] "I-2"  "JL"   "K-2"  "K-3"  "L-2"  "L-3"  "L-4"  "L-5"  "M-1"  "M-2"
## [31] "N-1"  "P-4"  "P-5"  "PC"   "PD"   "R-4"  "RC"   "S-2"  "S-3"  "T-2"
## [41] "T-4"  "T-5"  "T-6"  "T-9"  "TGT"  "V-1"

sum(is.na(original_dataset))

## [1] 9
```

```
colnames(original_dataset)

##  [1] "OrderType"       "OrderCategory"   "CustomerCode"    "CountryName"
##  [5] "CustomerOrderNo" "Custorderdate"   "UnitName"        "QtyRequired"
##  [9] "TotalArea"       "Amount"          "ITEM_NAME"       "QualityName"
## [13] "DesignName"      "ColorName"       "AreaFt"          "ShapeName"

levels(OrderCategory)

## [1] "Order"  "Sample"

#Display the different customer codes in each country for both order
categories
```

Balance and Unbalanced data

There are 4651 instances with no conversion and 1169 with conversion, we can clearly see that the data set is an unbalanced data, and we can combat using the following 3 techniques:

1. Under- Sampling

2.Over -Sampling

3. SMOTE

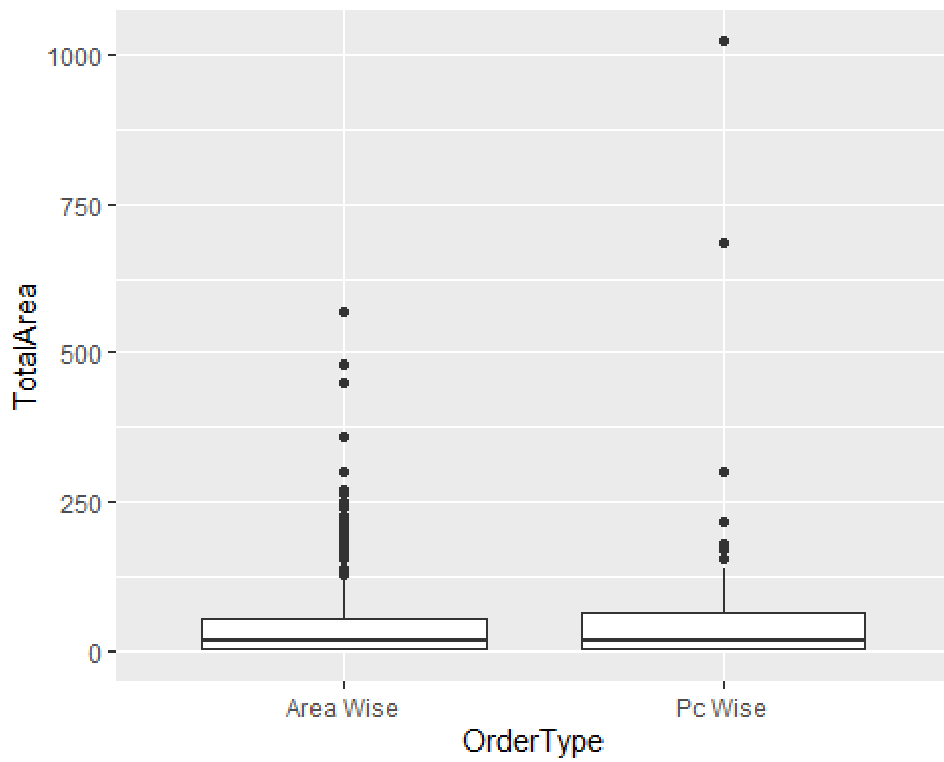
#install.packages("ROSE")

```
library(ROSE)

## Warning: package 'ROSE' was built under R version 4.1.3

## Loaded ROSE 0.0-4

##We are balancing the data using Over-sampling technique
balanced_sample_dataset <- ovun.sample(Order_Conversion~., data =
sample_only_dataset, method = "over", N = 9000)$data

summary(balanced_sample_dataset$Order_Conversion)

## Not Converted     Converted
##          4651          4349
```

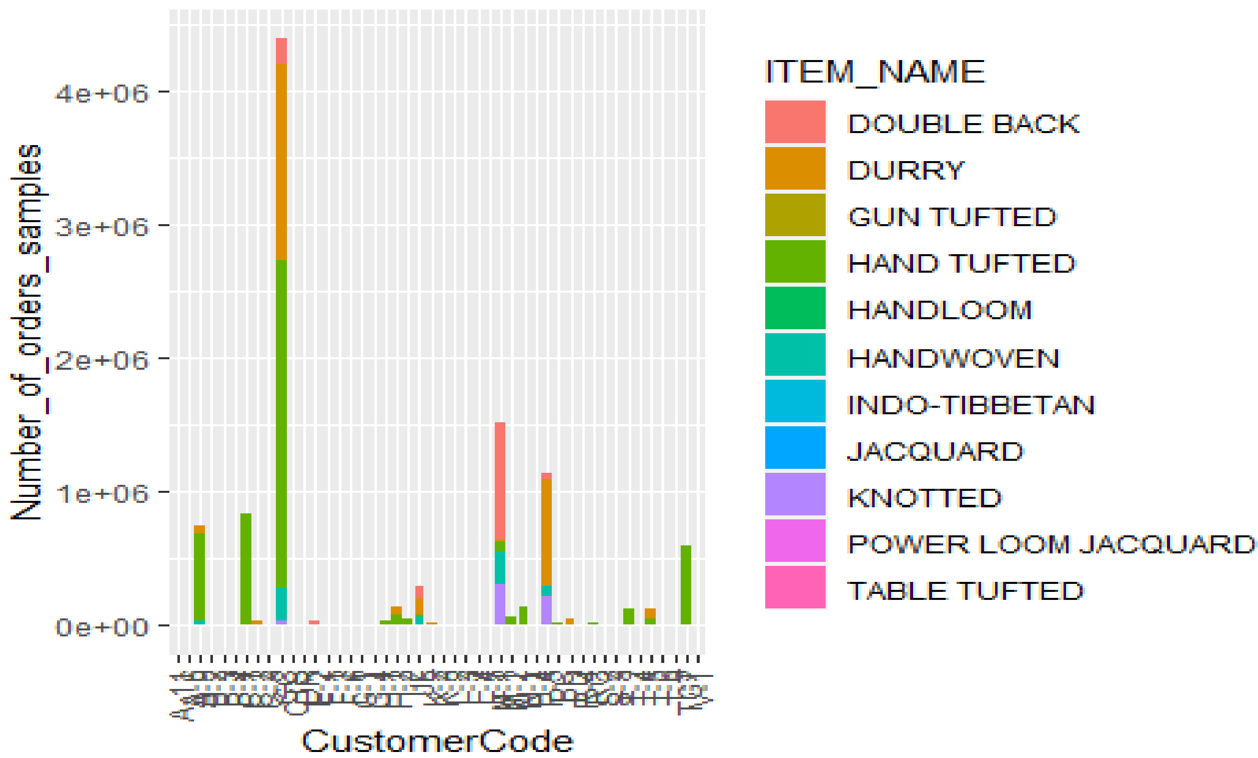
Q1. With the help of data visualization, provide key insights using exploratory data analysis.

Basic Visualizations

```
#Box plot for Numerical valriables:
ggplot(data = dataset, aes(x = OrderType ,y = TotalArea)) + geom_boxplot()
```



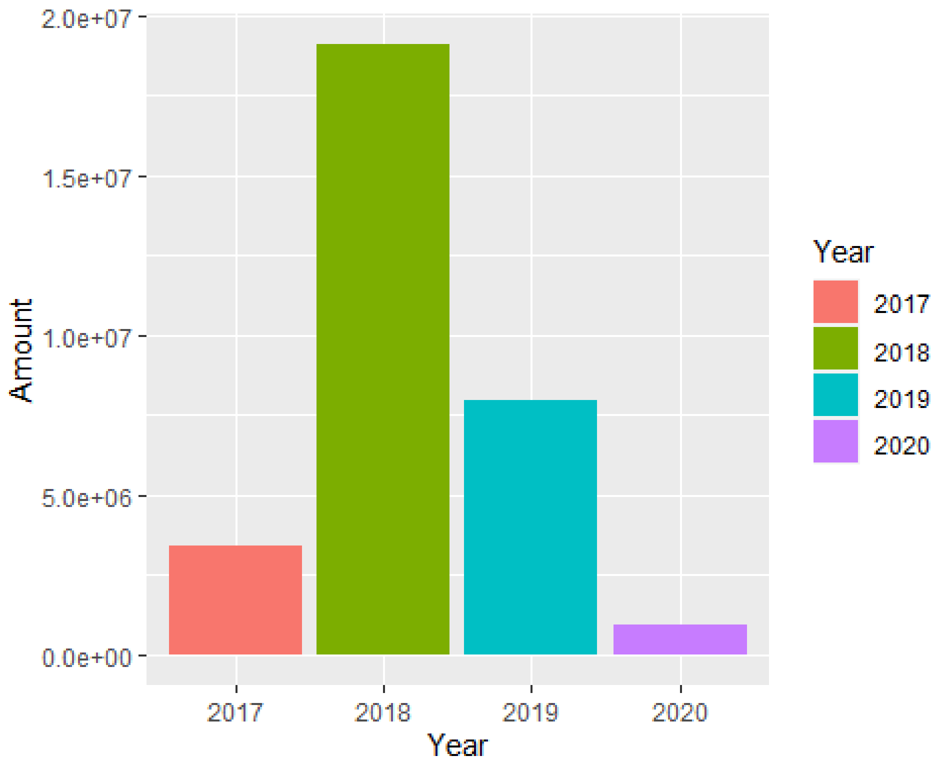

```
#There are few outliers in the total area of area and piece wise dataset in
the main dataset, but since we are using Sample only data for machine
learning model model

#Removing blank values in ITEM_NAME in the raw data set
dataset <- dataset[!(dataset$ITEM_NAME =="-"),]


#Distribution of customer code

dataset %>% select(CustomerCode,ITEM_NAME) %>%
group_by(CustomerCode, ITEM_NAME) %>%
mutate(Number_of_orders_samples = n()) %>%
ggplot(aes(fill = ITEM_NAME, x = CustomerCode, y =
Number_of_orders_samples))+
geom_bar(position = "stack", stat = "identity") + scale_x_discrete(guide =
guide_axis(angle = 90))
```




```
#The graph clearly shows the customer codes whose orders and sample request
are high and their frequent item names

#From the graph below we can say the Frequent Customer groups are as follows:
#1.CC - Hand Tufted and Durry and Jacquard
#2.M-1 - Double Back
#3.P-5 - Durry
#4.C-1 - Hand Tufted
#5.A-9 - Hand Tufted
#6.TGT -Hand Tufted

# And in all their purchase the HAND TUFTED is common, and in group M-1 their
maximum requests are Double Back

#The total revenue generated in all the years
attach(dataset)

## The following objects are masked from dataset (pos = 4):
##
##     Amount, AreaFt, ColorName, CountryName, CustomerCode,
##     CustomerOrderNo, Custorderdate, DesignName, ITEM_NAME,
##     OrderCategory, OrderType, QtyRequired, QualityName, ShapeName,
##     TotalArea, UnitName

dataset$Year <- format(dataset$Custorderdate, format = "%Y")

head(dataset$Year)
```

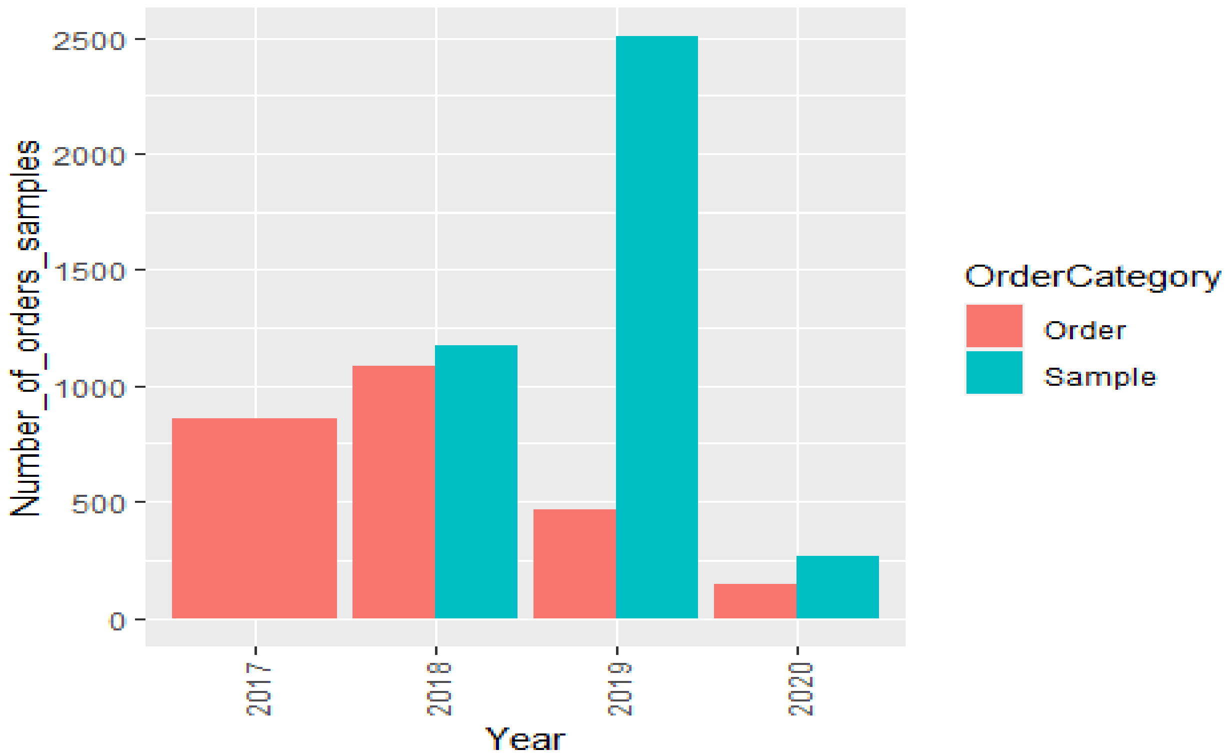
```
## [1] "2017" "2017" "2017" "2017" "2017" "2017"

unique(dataset$Year)

## [1] "2017" "2018" "2019" "2020"

ggplot(dataset, aes(x=Year, y=Amount, fill=Year)) +
geom_bar(stat = "identity")
```

Total Revenue over the Years from 2017-2020

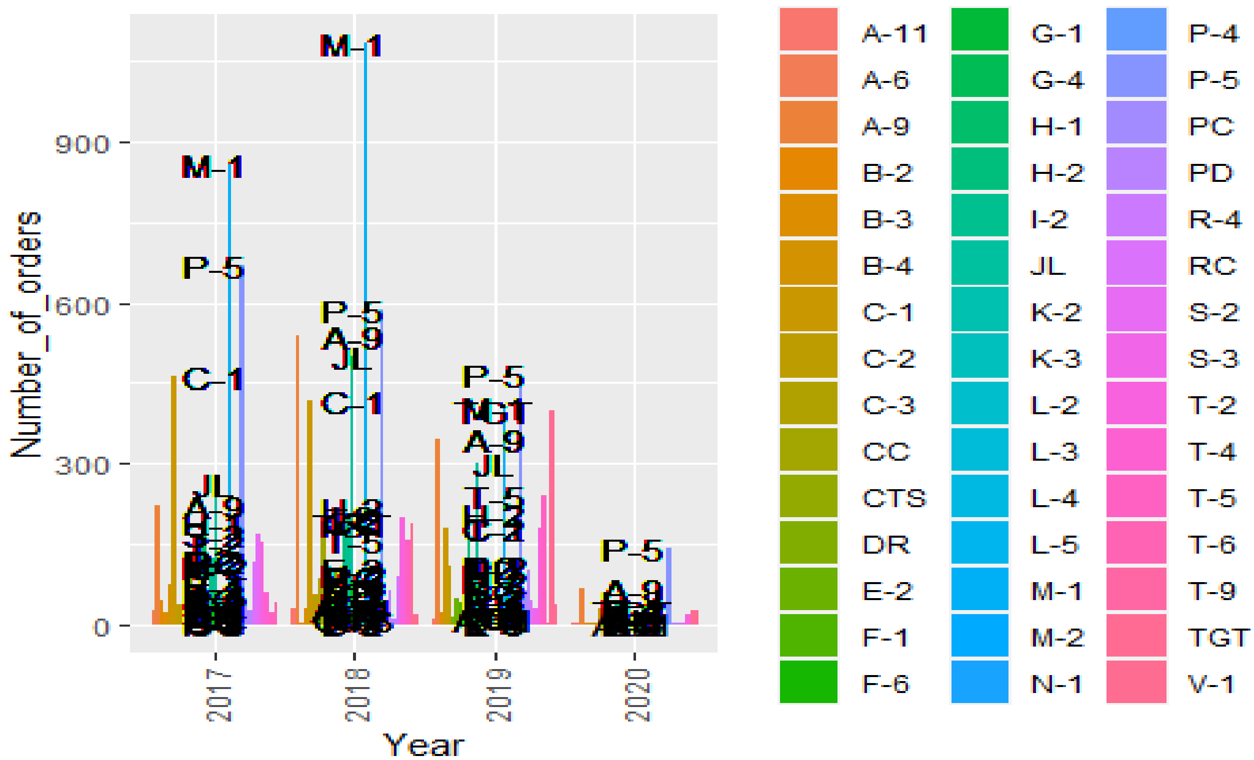



```
#As we can see the maximum revenue was generated in the year 2018 and then in the year 2019

#Identifying the number of orders in each year

dataset %>% select(CustomerCode,Year, OrderCategory) %>%
group_by(CustomerCode, Year, OrderCategory) %>%
mutate(Number_of_orders_samples = n()) %>%
ggplot(aes(fill = OrderCategory, x = Year, y = Number_of_orders_samples))+
geom_bar(position = "dodge", stat = "identity") + scale_x_discrete(guide =
guide_axis(angle = 90))
```

```
Distribution of Order and Sample Category Over the years
```

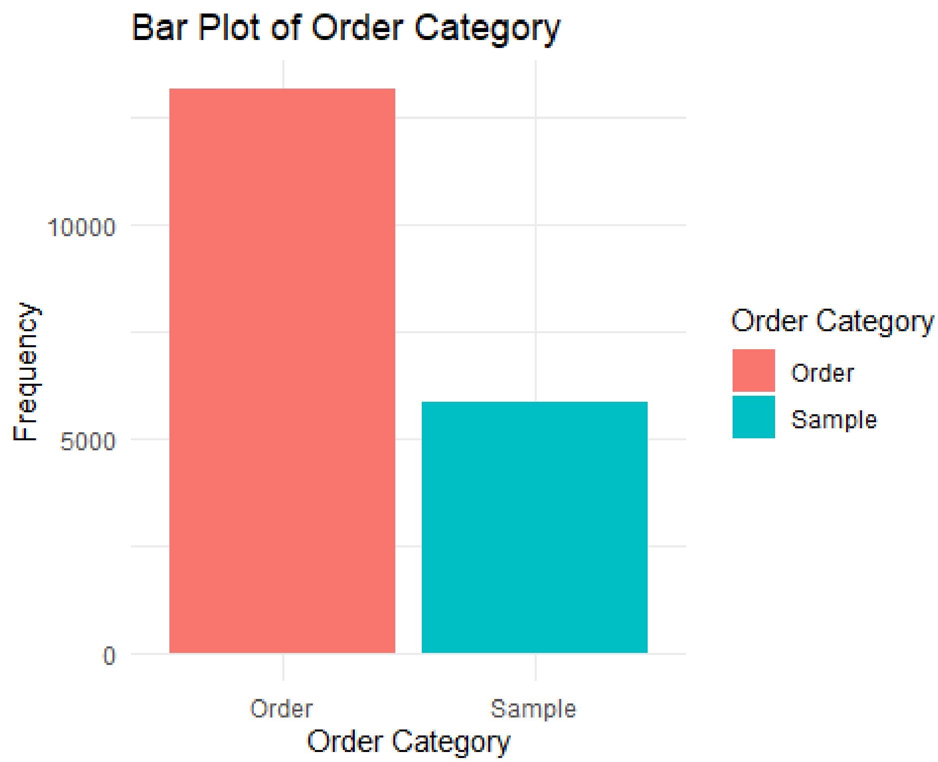



```
#As we can see the the sample requests were higher for the year 2019, 2017
has the highest orders

dataset %>% select(CustomerCode,Year, OrderCategory) %>%
group_by(CustomerCode, Year, OrderCategory) %>%
filter(OrderCategory == "Order") %>%
mutate(Number_of_orders = n()) %>%
ggplot(aes(fill = CustomerCode, x = Year, y = Number_of_orders))+
geom_bar(position = "dodge", stat = "identity") + scale_x_discrete(guide =
guide_axis(angle = 90)) +
geom_text(aes(label = CustomerCode))
```



Bar Plot of Order Category

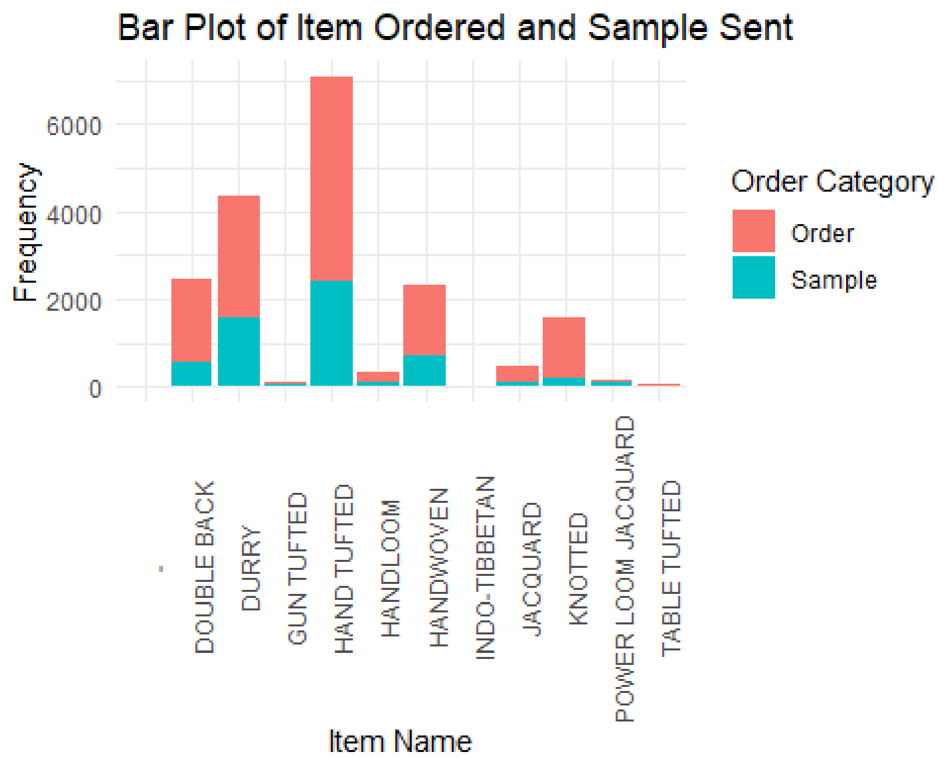
```
ggplot(data=original_dataset, aes(x=as.factor(OrderCategory),
fill=as.factor(OrderCategory)))+
  geom_bar()+
  theme_minimal()+ ggtitle("Bar Plot of Order Category") +
  xlab("Order Category") + ylab("Frequency")+
  labs(fill = "Order Category")
```




```
table(original_dataset$OrderCategory)

##
##  Order Sample
##  13135   5820
```

Bar Plot of Item Name

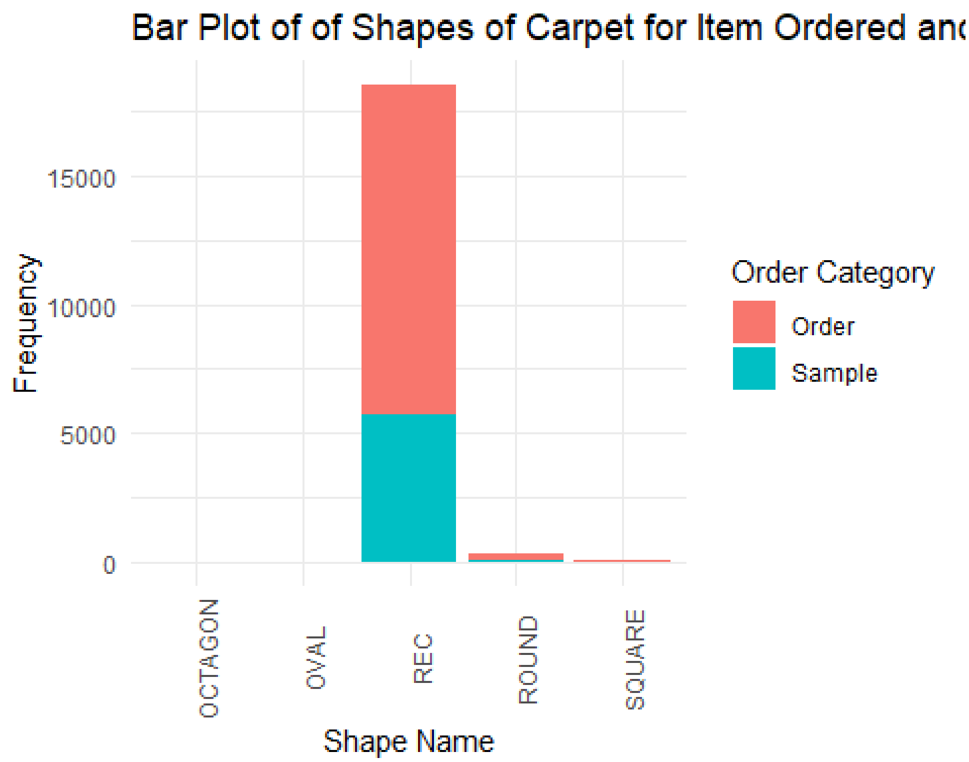
```
ggplot(data=original_dataset, aes(x=as.factor(ITEM_NAME),
fill=as.factor(OrderCategory)))+
  geom_bar()+
  theme_minimal()+ ggtitle("Bar Plot of Item Ordered and Sample Sent") +
  xlab("Item Name") + ylab("Frequency")+
  labs(fill = "Order Category")+theme(axis.text.x = element_text(angle = 90))
```




```
original_dataset %>% group_by(ITEM_NAME)%>%tally()

## # A tibble: 12 x 2
##    ITEM_NAME              n
##    <chr>              <int>
##  1 -                      4
##  2 DOUBLE BACK         2474
##  3 DURRY               4355
##  4 GUN TUFTED            91
##  5 HAND TUFTED         7095
##  6 HANDLOOM             357
##  7 HANDWOVEN           2330
##  8 INDO-TIBBETAN         11
##  9 JACQUARD             477
## 10 KNOTTED             1575
## 11 POWER LOOM JACQUARD  144
## 12 TABLE TUFTED          42
```

Bar Plot of Shape Name

```
ggplot(data=original_dataset, aes(x=as.factor(ShapeName),
fill=as.factor(OrderCategory)))+
  geom_bar()+
  theme_minimal()+ ggtitle("Bar Plot of of Shapes of Carpet for Item Ordered
and Sample Sent") +
  xlab("Shape Name") + ylab("Frequency")+
  labs(fill = "Order Category")+theme(axis.text.x = element_text(angle = 90))
```

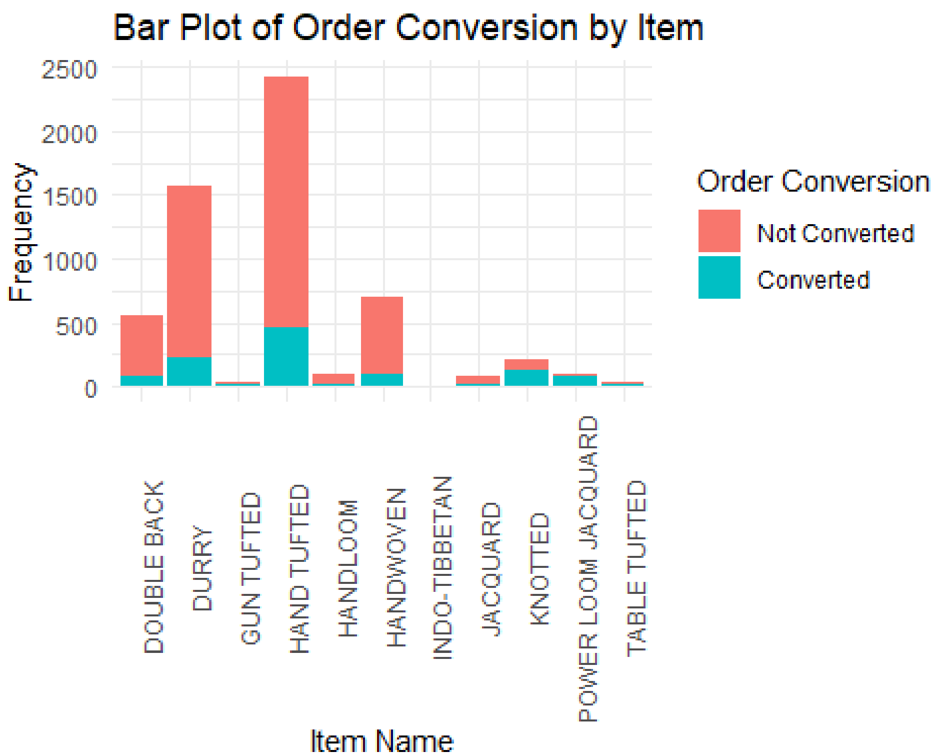



```
original_dataset %>% group_by(ShapeName,OrderCategory)%>%tally()

## # A tibble: 8 x 3
## # Groups:   ShapeName [5]
##   ShapeName OrderCategory     n
##   <chr>     <chr>         <int>
## 1 OCTAGON   Order             2
## 2 OVAL      Order             1
## 3 REC       Order         12777
## 4 REC       Sample         5741
## 5 ROUND     Order           305
## 6 ROUND     Sample           57
## 7 SQUARE    Order            50
## 8 SQUARE    Sample           22
```

Most of the Orders received by Champo Carpets are of Rectangular Size, and for this size only most of the samples sent.

Bar Plot of Order Conversion by Item name.

```
ggplot(data=sample_only_dataset, aes(x=as.factor(ITEM_NAME),
fill=Order_Conversion))+
  geom_bar()+
  theme_minimal()+ ggtitle("Bar Plot of Order Conversion by Item") +
  xlab("Item Name") + ylab("Frequency")+
  labs(fill = "Order Conversion")+theme(axis.text.x = element_text(angle =
90))
```

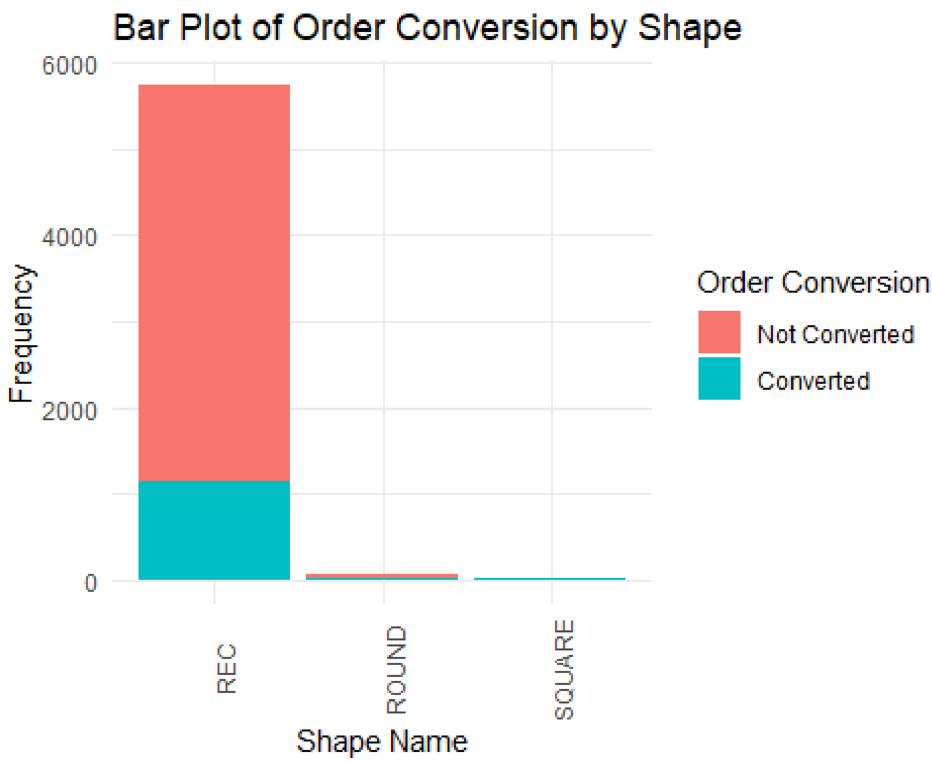



```
sample_only_dataset %>% group_by(ITEM_NAME,Order_Conversion)%>%tally()
```

```
## # A tibble: 21 x 3
## # Groups:   ITEM_NAME [11]
##    ITEM_NAME   Order_Conversion     n
##    <chr>       <fct>            <int>
##  1 DOUBLE BACK Not Converted      477
##  2 DOUBLE BACK Converted           77
##  3 DURRY       Not Converted     1333
##  4 DURRY       Converted          230
##  5 GUN TUFTED  Not Converted       20
##  6 GUN TUFTED  Converted           17
##  7 HAND TUFTED Not Converted     1967
##  8 HAND TUFTED Converted          458
##  9 HANDLOOM    Not Converted       79
## 10 HANDLOOM    Converted           24
## # ... with 11 more rows
```

Bar Plot of Order Conversion by Shape

```
ggplot(data=sample_only_dataset, aes(x=as.factor(ShapeName),
fill=Order_Conversion))+
  geom_bar()+
  theme_minimal()+ ggtitle("Bar Plot of Order Conversion by Shape") +
  xlab("Shape Name") + ylab("Frequency")+
  labs(fill = "Order Conversion")+theme(axis.text.x = element_text(angle =
90))
```

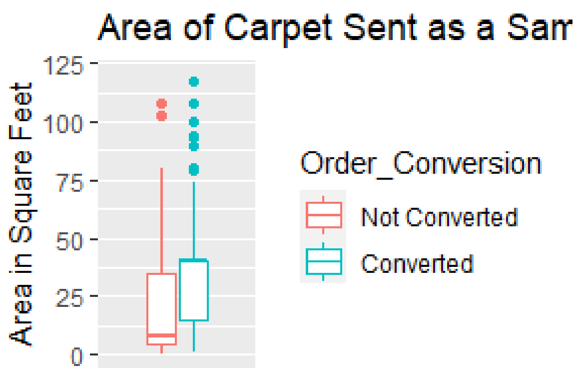



```
sample_only_dataset %>% group_by(ShapeName,Order_Conversion)%>%tally()

## # A tibble: 6 x 3
## # Groups:   ShapeName [3]
##   ShapeName Order_Conversion     n
##   <fct>     <fct>            <int>
## 1 REC       Not Converted     4598
## 2 REC       Converted         1143
## 3 ROUND     Not Converted       40
## 4 ROUND     Converted           17
## 5 SQUARE    Not Converted       13
## 6 SQUARE    Converted            9
```

Most of the Orders received by Champo Carpets are of Rectangular Size, and for this size only most of the samples sent.

Box Plot of Area of Sample Sent by Order Conversion

```
sample_only_dataset%>%ggplot() +
  geom_boxplot(aes(y = AreaFt, color=Order_Conversion)) +
  scale_x_discrete( ) +
  ylim(0,120)+
  labs(title = "Area of Carpet Sent as a Sample in Square Feet", y = "Area in Square Feet")
```




```
summary(sample_only_dataset$AreaFt)

##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##   0.6667   6.0000  11.0000  21.5558  39.8125 480.0000
```

Key Insights from Data Visualization

The key insights from the data visualization is that the conversion rate of the sample sent to the order received is $\frac{1169}{(1169+4651)} = \frac{1169}{5820} = 20.09\%$. The conversion rate of Rectangular Carpet is $\frac{1143}{(1143+4598)} = 19.91\%$. The maximum number of carpet sent as a sample were of Shape Rectangle, the conversion rate is $\frac{458}{1967+458} = 18.89\%$. Likewise, the conversion rate of Durry and Handwoven are 17.25% and 15.76%, respectively.The distribution of Carpet Area in square Feet is skewed and consist of many outliers. The distribution of Area for Order Conversion 1 is Left Skewed and the distribution of Area for Order Conversion 0 is right skewed. Thus, the box and whisker plot reveals that the conversion rate of sample carpet sent is higher for the carpet with higher Area. The major challenges which Champo carpet faces is the lower conversion rate and the project envision to reveal possible solution for improving the conversion rate for the sample sent.

Solution Approach

The Champo Carpet faces the issue of low conversion rate of the sample sent to the order received. The initial analysis post exploratory data analysis is the binary classification of Order Conversion through Machine Learning Classification techniques. The technique includes Logistic Regression Modeling, Random Forest Classification, Decision Tree Modeling, Neural Networking.The modeling helps identify the best and appropriate classifier for classifying the order conversion based on the features of sample carpet sent. The next stage of analysis is to perform unsupervised leaning methods including the clustering techniques and market basket analysis to identify the various cluster and to develop a recommend system in the end to take the lead forward. Finally to advice Champo Carpet what and how can the company improve the order conversion rate.

Q2. What kind of analytics and machine learning algorithms (e.g. classification, regression, clustering, recommender systems and etc.) can be used by Champo Carpets to solve their problems and in general for value creation? Justify your choices.

For our data analysis we have considered the "Champo Carpets excel dataset" and since the target response is 1, 0 indicating the order conversion, which is a binary target variable, which is basically a classification problem, so we decided to use the following decision models:

- Decision Tree
- Random Forest
- Logistic regression

Since we were asked to identify the most popular customer codes and most frequently bought products, we have used the following unsupervised learning:

- K-means
- Hierarchical clustering
- Neural Networks

Under K-Means clustering we are able to identify all the customer code/ customer segments whose purchase behaviour is similar.

Logistic regression

```
Sample_onlyData_LM <- readxl::read_excel("Champo Carpets.xlsx", sheet = "Data
on Sample ONLY");

names(Sample_onlyData_LM)[names(Sample_onlyData_LM) == 'Order Conversion'] <-
"Order_Conversion"

colnames(Sample_onlyData_LM)

##  [1] "CustomerCode"     "CountryName"      "USA"              "UK"
##  [5] "Italy"            "Belgium"          "Romania"          "Australia"
##  [9] "India"            "QtyRequired"      "ITEM_NAME"        "Hand
Tufted"
## [13] "Durry"            "Double Back"      "Hand Woven"       "Knotted"
## [17] "Jacquard"         "Handloom"         "Other"            "ShapeName"
## [21] "REC"              "Round"            "Square"           "AreaFt"
## [25] "Order_Conversion"

colnames_Sample_only_Data <- c(1:9,11:23,25)

set.seed(9090)

Sample_onlyData_LM[colnames_Sample_only_Data] <-
lapply(Sample_onlyData_LM[colnames_Sample_only_Data], as.factor)
```

```
Sample_onlyData_LM <- subset(Sample_onlyData_LM,select = -c(3:9,12:19,21:23))

colnames(Sample_onlyData_LM)

## [1] "CustomerCode"     "CountryName"      "QtyRequired"      "ITEM_NAME"
## [5] "ShapeName"        "AreaFt"           "Order_Conversion"

indx <- sample(2, nrow(Sample_onlyData_LM), replace = T, prob = c(0.8, 0.2))
train <- Sample_onlyData_LM[indx == 1, ]
test <- Sample_onlyData_LM[indx == 2, ]

glimpse(Sample_onlyData_LM)

## Rows: 5,820
## Columns: 7
## $ CustomerCode     <fct> CC, M-1, M-1, M-1, M-1, CC, CC, M-1, M-1, CC, CC,
CC,~
## $ CountryName      <fct> INDIA, USA, USA, USA, USA, INDIA, INDIA, USA,
USA, IN~
## $ QtyRequired      <dbl> 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 25, 1, 2, 35,
35, 35~
## $ ITEM_NAME        <fct> HAND TUFTED, HAND TUFTED, HAND TUFTED, HAND
TUFTED, H~
## $ ShapeName        <fct> REC, REC, REC, REC, REC, REC, REC, REC, REC, REC,
REC~
## $ AreaFt           <dbl> 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 40.0,
108.0~
## $ Order_Conversion <fct> 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0,
0, 0,~

attach(Sample_onlyData_LM)

## The following objects are masked from dataset (pos = 3):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName

## The following objects are masked from dataset (pos = 5):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName

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

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

```
##     Min       1Q   Median       3Q      Max
## -2.7954  -0.5722  -0.2624  -0.1724   3.0299
##
## Coefficients: (12 not defined because of singularities)
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 -2.377e+00  1.321e+00  -1.800 0.071921 .
## CustomerCodeA-9              1.944e-01  1.322e+00   0.147 0.883031
## CustomerCodeB-3             -1.404e+01  1.455e+03  -0.010 0.992304
## CustomerCodeC-1             -3.372e-02  1.379e+00  -0.024 0.980499
## CustomerCodeC-2              4.926e-02  1.355e+00   0.036 0.970999
## CustomerCodeCC              -1.913e+00  1.311e+00  -1.460 0.144315
## CustomerCodeCTS             -1.477e+01  4.859e+02  -0.030 0.975744
## CustomerCodeE-2              3.325e+00  1.506e+00   2.207 0.027309 *
## CustomerCodeF-1              1.795e-01  1.381e+00   0.130 0.896607
## CustomerCodeF-2             -6.173e-01  1.801e+00  -0.343 0.731756
## CustomerCodeF-6              1.693e+01  7.166e+02   0.024 0.981150
## CustomerCodeH-2             -4.883e-01  1.324e+00  -0.369 0.712162
## CustomerCodeI-2              1.116e+00  1.417e+00   0.788 0.430970
## CustomerCodeJL               1.814e+00  1.361e+00   1.332 0.182718
## CustomerCodeK-2             -1.410e+01  7.277e+02  -0.019 0.984539
## CustomerCodeK-3             -1.342e+01  1.455e+03  -0.009 0.992643
## CustomerCodeL-3             -1.394e+00  1.847e+00  -0.755 0.450195
## CustomerCodeL-4             -1.609e+01  1.029e+03  -0.016 0.987529
## CustomerCodeL-5             -1.452e+00  1.513e+00  -0.960 0.337260
## CustomerCodeM-1             -9.976e-01  1.347e+00  -0.740 0.459015
## CustomerCodeM-2             -5.087e-02  1.367e+00  -0.037 0.970323
## CustomerCodeN-1             -2.165e+00  1.364e+00  -1.587 0.112472
## CustomerCodeP-4             -2.660e+00  1.431e+00  -1.858 0.063132 .
## CustomerCodeP-5              2.333e-01  1.354e+00   0.172 0.863250
## CustomerCodePC              -1.780e+01  6.227e+02  -0.029 0.977200
## CustomerCodePD               4.246e+00  1.385e+00   3.065 0.002178 **
## CustomerCodeRC              -2.348e+00  1.653e+00  -1.421 0.155317
## CustomerCodeS-3             -3.936e-01  1.350e+00  -0.292 0.770615
## CustomerCodeT-2             -2.462e+00  1.438e+00  -1.711 0.086997 .
## CustomerCodeT-4              7.026e-01  2.071e+00   0.339 0.734454
## CustomerCodeT-5             -8.450e-01  1.328e+00  -0.636 0.524510
## CustomerCodeTGT             -1.521e+00  1.339e+00  -1.136 0.255973
## CustomerCodeV-1             -2.856e+00  1.751e+00  -1.631 0.102854
## CountryNameBELGIUM                  NA         NA      NA       NA
## CountryNameBRAZIL                   NA         NA      NA       NA
## CountryNameCANADA                   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                      2.569e-02  7.507e-03   3.422 0.000622 ***
## ITEM_NAMEDURRY                   5.442e-01  2.237e-01   2.433 0.014971 *
## ITEM_NAMEGUN TUFTED              3.127e+00  4.394e-01   7.115 1.12e-12 ***
## ITEM_NAMEHAND TUFTED             3.965e-01  2.107e-01   1.882 0.059894 .
## ITEM_NAMEHANDLOOM                3.265e-01  3.783e-01   0.863 0.388136
## ITEM_NAMEHANDWOVEN              -4.196e-01  2.772e-01  -1.513 0.130163
## ITEM_NAMEINDO-TIBBETAN           1.850e+01  8.403e+02   0.022 0.982438
## ITEM_NAMEJACQUARD                2.517e-01  4.356e-01   0.578 0.563411
## ITEM_NAMEKNOTTED                 3.195e+00  2.807e-01  11.382  < 2e-16 ***
## ITEM_NAMEPOWER LOOM JACQUARD     5.733e+00  4.414e-01  12.988  < 2e-16 ***
## ITEM_NAMETABLE TUFTED            3.843e+00  5.305e-01   7.244 4.35e-13 ***
## ShapeNameROUND                   8.571e-01  4.176e-01   2.052 0.040122 *
## ShapeNameSQUARE                  1.501e+00  6.855e-01   2.190 0.028529 *
## AreaFt                           5.714e-02  2.781e-03  20.545  < 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: 4591.5  on 4578  degrees of freedom
## Residual deviance: 2830.5  on 4532  degrees of freedom
## AIC: 2924.5
##
## Number of Fisher Scoring iterations: 14
```

```
#Pred <- predict(logitModel, newdata = test, type = "response")
#Pred
```

VARIABLE SELECTION

```
#Class <- ifelse(Pred >= 0.5, 1, 0)
#Class

#Performing the Variable Selection with Forward Selction

#Constructing a full glm model with all the variables

full <-  glm(Order_Conversion~., data = train, family = "binomial")

null <- glm(Order_Conversion~1, data = train, family = "binomial")

#Forward Selection
step(null, scope = list(lower = null, upper = full), direction = "forward")
```

```
## Start:  AIC=4593.54
## Order_Conversion ~ 1
##
##                 Df Deviance    AIC
## + CustomerCode 32   3999.0 4065.0
## + AreaFt        1   4118.6 4122.6
## + CountryName  12   4159.2 4185.2
```

```
## + ITEM_NAME    10   4172.2 4194.2
## + QtyRequired   1   4578.4 4582.4
## + ShapeName     2   4585.1 4591.1
## <none>              4591.5 4593.5
##
## Step:  AIC=4064.98
## Order_Conversion ~ CustomerCode
##
##               Df Deviance    AIC
## + ITEM_NAME   10   3428.9 3514.9
## + AreaFt       1   3480.8 3548.8
## + QtyRequired  1   3986.3 4054.3
## + ShapeName    2   3989.9 4059.9
## <none>             3999.0 4065.0
##
## Step:  AIC=3514.87
## Order_Conversion ~ CustomerCode + ITEM_NAME
##
##               Df Deviance    AIC
## + AreaFt       1   2848.2 2936.2
## + QtyRequired  1   3413.7 3501.7
## <none>             3428.9 3514.9
## + ShapeName    2   3426.0 3516.0
##
## Step:  AIC=2936.2
## Order_Conversion ~ CustomerCode + ITEM_NAME + AreaFt
##
##               Df Deviance    AIC
## + QtyRequired  1   2838.3 2928.3
## + ShapeName    2   2840.3 2932.3
## <none>             2848.2 2936.2
##
## Step:  AIC=2928.26
## Order_Conversion ~ CustomerCode + ITEM_NAME + AreaFt + QtyRequired
##
##             Df Deviance    AIC
## + ShapeName  2   2830.5 2924.5
## <none>           2838.3 2928.3
##
## Step:  AIC=2924.48
## Order_Conversion ~ CustomerCode + ITEM_NAME + AreaFt + QtyRequired +
##     ShapeName
##
##        Df Deviance    AIC
## <none>      2830.5 2924.5

##
## Call:  glm(formula = Order_Conversion ~ CustomerCode + ITEM_NAME + AreaFt
+
##     QtyRequired + ShapeName, family = "binomial", data = train)
```

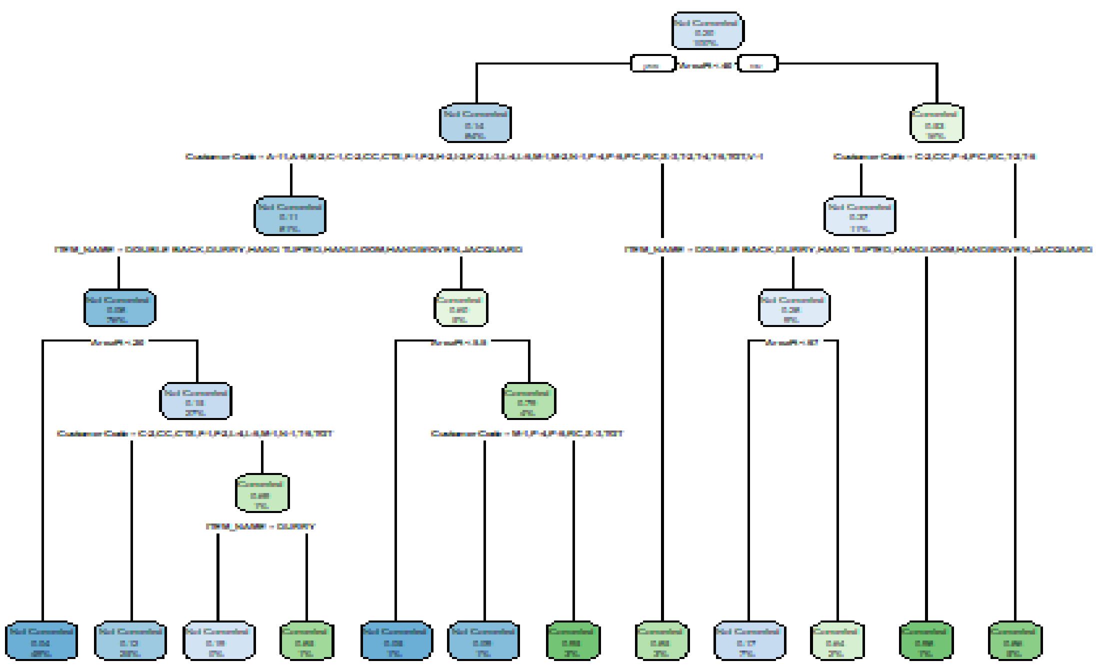
```
##
## Coefficients:
##                 (Intercept)              CustomerCodeA-9
##                    -2.37659                      0.19443
##             CustomerCodeB-3              CustomerCodeC-1
##                   -14.03774                     -0.03372
##             CustomerCodeC-2               CustomerCodeCC
##                     0.04925                     -1.91325
##             CustomerCodeCTS              CustomerCodeE-2
##                   -14.77232                      3.32450
##             CustomerCodeF-1              CustomerCodeF-2
##                     0.17949                     -0.61725
##             CustomerCodeF-6              CustomerCodeH-2
##                    16.93149                     -0.48834
##             CustomerCodeI-2               CustomerCodeJL
##                     1.11564                      1.81403
##             CustomerCodeK-2              CustomerCodeK-3
##                   -14.10222                    -13.41979
##             CustomerCodeL-3              CustomerCodeL-4
##                    -1.39434                    -16.08575
##             CustomerCodeL-5              CustomerCodeM-1
##                    -1.45216                     -0.99756
##             CustomerCodeM-2              CustomerCodeN-1
##                    -0.05087                     -2.16516
##             CustomerCodeP-4              CustomerCodeP-5
##                    -2.66006                      0.23326
##              CustomerCodePC               CustomerCodePD
##                   -17.79681                      4.24572
##              CustomerCodeRC              CustomerCodeS-3
##                    -2.34849                     -0.39364
##             CustomerCodeT-2              CustomerCodeT-4
##                    -2.46167                      0.70265
##             CustomerCodeT-5               CustomerCodeTGT
##                    -0.84503                     -1.52065
##             CustomerCodeV-1                ITEM_NAMEDURRY
##                    -2.85622                      0.54420
##         ITEM_NAMEGUN TUFTED         ITEM_NAMEHAND TUFTED
##                     3.12656                      0.39647
##           ITEM_NAMEHANDLOOM           ITEM_NAMEHANDWOVEN
##                     0.32652                     -0.41957
##      ITEM_NAMEINDO-TIBBETAN            ITEM_NAMEJACQUARD
##                    18.49613                      0.25167
##            ITEM_NAMEKNOTTED  ITEM_NAMEPOWER LOOM JACQUARD
##                     3.19467                      5.73267
##       ITEM_NAMETABLE TUFTED                       AreaFt
##                     3.84319                      0.05714
##                 QtyRequired               ShapeNameROUND
##                     0.02569                      0.85712
##             ShapeNameSQUARE
##                     1.50117
```

```
##
## Degrees of Freedom: 4578 Total (i.e. Null);  4532 Residual
## Null Deviance:       4592
## Residual Deviance: 2830  AIC: 2924

#The AIC should be lower to have a better accurate model, so from our
dataset, we can see that the formula with best variables is AreaFt and
CustomerCode , and we should exclude all the variables beyond <none>

#And in the next step, we can see that the AIC is decreased further to
4049.71 with customer code variables to the formula, and in the next further
step we see that the AIC is decreased further to 3490.45 with addition to
Area Ft, and with all the variables added we have the AIC reduced to 2872.26
```

Applying Decision Trees for the Sample Only data (Unbalanced data)

```
set.seed(1234)
nrow(sample_only_dataset)

## [1] 5820

#There are no Null values
sum(is.na(sample_only_dataset))

## [1] 0

indx <- sample(2, nrow(sample_only_dataset), replace = TRUE, prob =
c(0.8,0.2))
#Splitting the train and test data
train <- sample_only_dataset[indx == 1, ]
test <- sample_only_dataset[indx == 2,]

sum(is.na(train))

## [1] 0

sum(is.na(sample_only_dataset))

## [1] 0

nrow(train)/nrow(test) #4:1

## [1] 3.982877

attach(sample_only_dataset)

## The following objects are masked from Sample_onlyData_LM:
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, Order_Conversion,
##     QtyRequired, ShapeName

## The following objects are masked from dataset (pos = 4):
##
```

```
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName

## The following objects are masked from dataset (pos = 6):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName

formula = Order_Conversion ~ .

mytree <- rpart(formula, data = train)

rpart.plot(mytree)
```

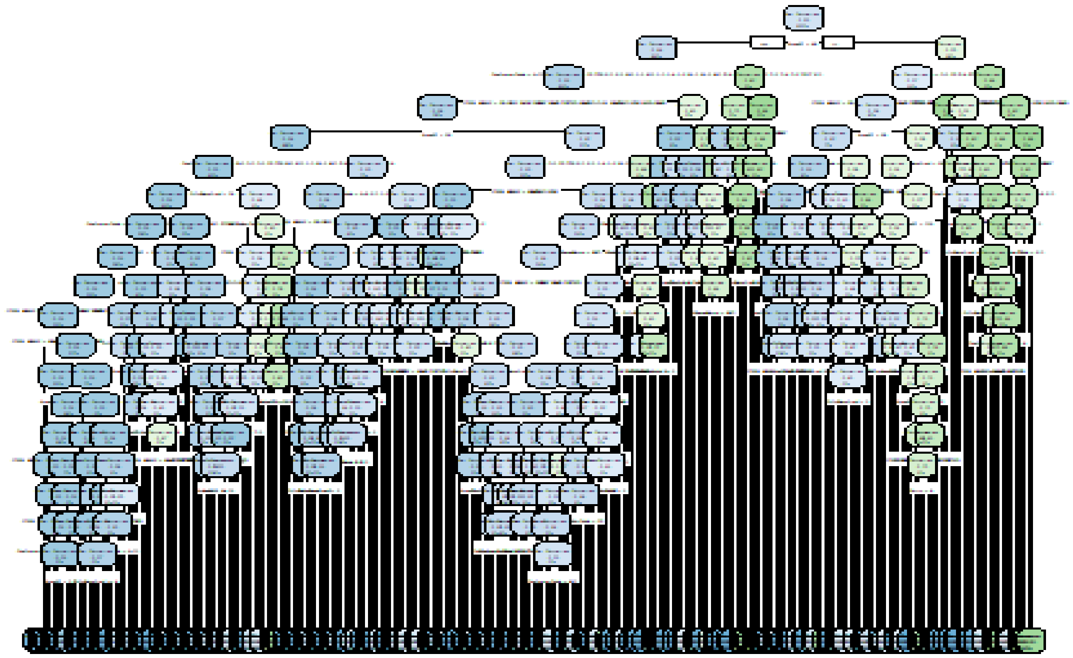


```
tree_pred_train <- predict(mytree, train, type = "class")

train_Error <- mean(tree_pred_train != train$Order_Conversion)

train_Error # <- The train error is 8.7%

## [1] 0.08748925

sum(is.na(tree_pred_train))

## [1] 0

testPred <- predict(mytree, newdata = test, type = "class")
```

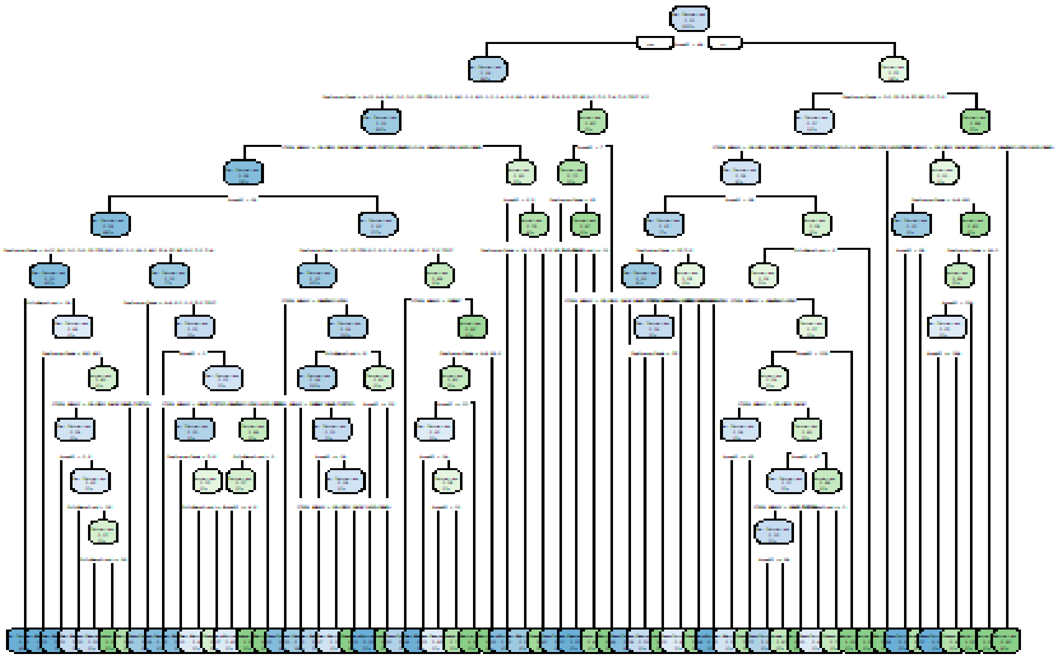
```
##Checking the test error:
mean(testPred != test$Order_Conversion)

## [1] 0.1001712

##The test error is 10%

confusionMatrix(testPred, test$Order_Conversion)

## Confusion Matrix and Statistics
##
##                Reference
## Prediction      Not Converted Converted
##   Not Converted           911        86
##   Converted                31       140
##
##                Accuracy : 0.8998
##                  95% CI : (0.8812, 0.9165)
##     No Information Rate : 0.8065
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6463
##
##  Mcnemar's Test P-Value : 5.966e-07
##
##             Sensitivity : 0.9671
##             Specificity : 0.6195
##          Pos Pred Value : 0.9137
##          Neg Pred Value : 0.8187
##              Prevalence : 0.8065
##          Detection Rate : 0.7800
##    Detection Prevalence : 0.8536
##       Balanced Accuracy : 0.7933
##
##        'Positive' Class : Not Converted
##
```

Tuning the Hyper parameters /Changing the parameters of the decision tree (Unbalanced data)

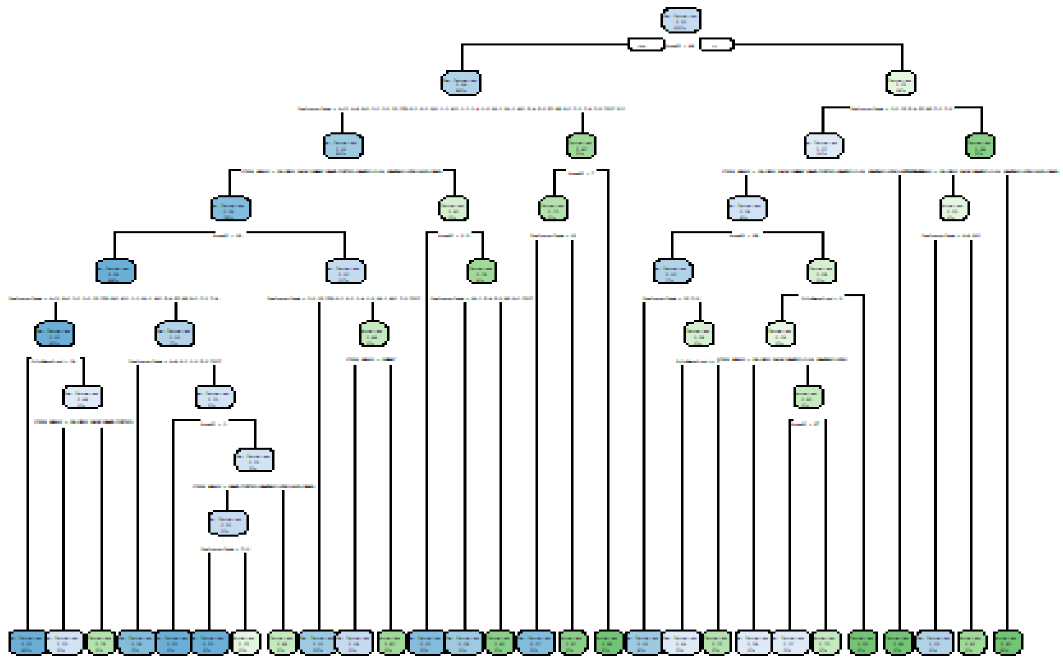
```
Constructing the entire decision tree

myTree1 <- rpart(formula, train, parms = list(split = "information"),control
= rpart.control(minbucket = 0,minsplit = 0, cp = -1))

rpart.plot(myTree1)

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



```
## Tuning the decision tree with different parameter values

myTree1 <- rpart(formula, train, parms = list(split = "information"),control
= rpart.control(minbucket = 3,minsplit = 5, cp = 0.001))

rpart.plot(myTree1)

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

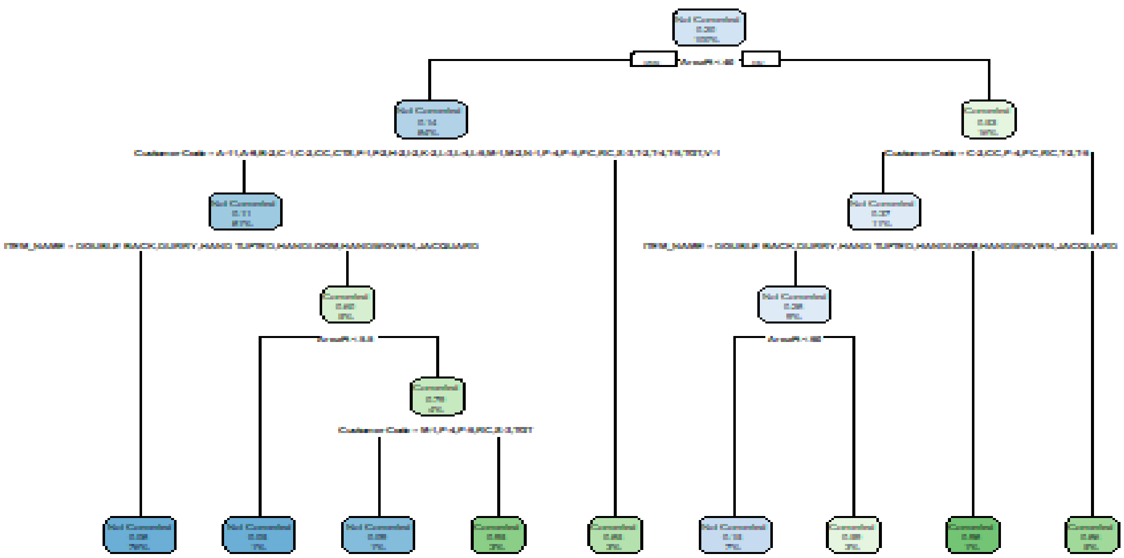


```r
myTree2 <- rpart(formula, train, parms = list(split = "information"),control
= rpart.control(minbucket = 10,minsplit = 5, cp = 0.001))

rpart.plot(myTree2)

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



```
tree2_pred_train <- predict(myTree2, train, type = "class")

train2_Error <- mean(tree2_pred_train != train$Order_Conversion)

train2_Error # <- The train error is 7.4%

## [1] 0.07459157

sum(is.na(tree_pred_train))

## [1] 0

testPred2 <- predict(myTree2, newdata = test, type = "class")

##Checking the test error:
mean(testPred != test$Order_Conversion)

## [1] 0.1001712

##The test error is 10%


##Hypertuning the parameters in the decision Tree
##1. Trying to construct with minbucket = 25 and Minsplit 25 and cp = 0.015

myTree3 <- rpart(formula, train, parms = list(split = "information"),control
= rpart.control(minbucket = 25,minsplit = 25, cp = 0.015))
```

```
rpart.plot(myTree3)
```



```
#Predicting with Train data
tree3_pred_train <- predict(myTree2, train, type = "class")

train3_Error <- mean(tree3_pred_train != train$Order_Conversion)

train3_Error # <- The train error is 7.4%

## [1] 0.07459157

testPred2 <- predict(myTree2, newdata = test, type = "class")

#Checking the test error:
mean(testPred != test$Order_Conversion)

## [1] 0.1001712

confusionMatrix(testPred2, test$Order_Conversion)

## Confusion Matrix and Statistics
##
##                  Reference
## Prediction      Not Converted Converted
##   Not Converted           912        64
##   Converted                30       162
```

```
##
##                Accuracy : 0.9195
##                  95% CI : (0.9024, 0.9345)
##     No Information Rate : 0.8065
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7265
##
##  Mcnemar's Test P-Value : 0.0006648
##
##             Sensitivity : 0.9682
##             Specificity : 0.7168
##          Pos Pred Value : 0.9344
##          Neg Pred Value : 0.8438
##              Prevalence : 0.8065
##          Detection Rate : 0.7808
##    Detection Prevalence : 0.8356
##       Balanced Accuracy : 0.8425
##
##        'Positive' Class : Not Converted
##
```

##Accuracy is 91.95%

The decision tree has the highest accuracy of 91% with the following hyperparameter values **minbucket = 10,minsplit = 5, cp = 0.001, and the sensitivity or the recall is 96%.**

###Random Forest for Unbalanced data

```
ntree <- 100
set.seed(123)

random_forest_data <- sample_random[sample(nrow(sample_random)),]
#Removing some dummy variables

random_forest_data <- subset (random_forest_data, select = -c(3:10))

str(random_forest_data)

## tibble [5,820 x 10] (S3: tbl_df/tbl/data.frame)
##  $ CustomerCode    : Factor w/ 34 levels "A-11","A-9","B-2",..: 7 23 7 23
7 7 7 7 7 7 ...
##  $ CountryName     : chr [1:5820] "INDIA" "USA" "INDIA" "USA" ...
##  $ Handloom        : num [1:5820] 0 0 0 0 0 0 0 0 0 0 ...
##  $ Other           : num [1:5820] 0 0 0 0 0 0 0 0 0 0 ...
##  $ ShapeName       : Factor w/ 3 levels "REC","ROUND",..: 1 1 1 1 1 1 1 1
1 1 ...
##  $ REC             : num [1:5820] 1 1 1 1 1 1 1 1 1 1 ...
##  $ Round           : num [1:5820] 0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ Square          : num [1:5820] 0 0 0 0 0 0 0 0 0 0 ...
##  $ AreaFt          : num [1:5820] 40 8.12 8.44 35 4 ...
##  $ Order_Conversion: Factor w/ 2 levels "0","1": 2 1 1 1 1 1 1 1 1 1 ...
```

```
colnames(random_forest_data)
```

```
##  [1] "CustomerCode"     "CountryName"      "Handloom"         "Other"
##  [5] "ShapeName"        "REC"              "Round"            "Square"
##  [9] "AreaFt"           "Order_Conversion"
```

```
random_forest_data$CustomerCode <- as.factor(random_forest_data$CustomerCode)
random_forest_data$CountryName <- as.factor(random_forest_data$CountryName)

myFormula = Order_Conversion~ .

##Building random forest model
rf <- randomForest(myFormula, data = random_forest_data, mtry =
sqrt(ncol(random_forest_data)-1), ntree = 100, proximity = T, importance = T)

print(rf)
```

```
##
## Call:
##  randomForest(formula = myFormula, data = random_forest_data,      mtry =
sqrt(ncol(random_forest_data) - 1), ntree = 100, 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: 10.02%
## Confusion matrix:
##      0   1 class.error
## 0 4527 124  0.02666093
## 1  459 710  0.39264328
```

```
##The function ran random forest classification, the no,of variables tried at
each split is 3. The OOB estimate of error rate is 10.1%

#Assigning the importance for each variable
#rf$importance
importance(rf, type = 1)
```

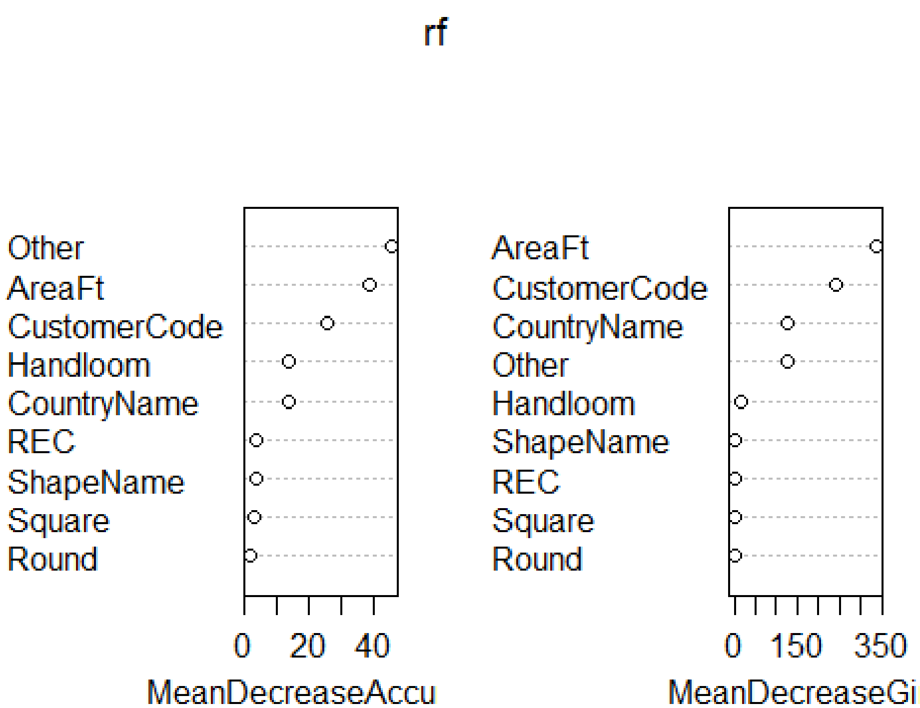
```
##              MeanDecreaseAccuracy
## CustomerCode            25.718920
## CountryName             13.840188
## Handloom                13.883364
## Other                   46.160407
## ShapeName                3.801021
## REC                      3.874198
## Round                    1.428337
```

```
## Square                         3.176847
## AreaFt                        38.937300
```

```
importance(rf, type = 2)
```

```
##                MeanDecreaseGini
## CustomerCode         242.304035
## CountryName          125.984756
## Handloom              14.302359
## Other                125.652317
## ShapeName              3.384845
## REC                    2.656725
## Round                  1.201849
## Square                 1.421301
## AreaFt               339.817252
```

```
varImpPlot(rf)
```



As the above graphs display the importance of the variables based on the MeanDecrease Accuracy and MeanDecreaseGini, and we can clearly state that AreaFt, CustomerCode, and Country Name are on top most important variables.

```
#The OOB error rate is 10%
```

```
rf$err.rate[ntree,1]
```

```
##       OOB
## 0.1001718
```

```r
#rf$predicted

# Confusion matrix
Confusion_Matrix_Random <- table(rf$predicted,
random_forest_data$Order_Conversion, dnn = c("Predicted", "Actual"))
Confusion_Matrix_Random
```

```
##          Actual
## Predicted    0    1
##         0 4527  459
##         1  124  710
```

```r
library(caret)
#confusionMatrix(rf$predicted, random_forest_data$Order_Conversion, positive
= "Converted")
```

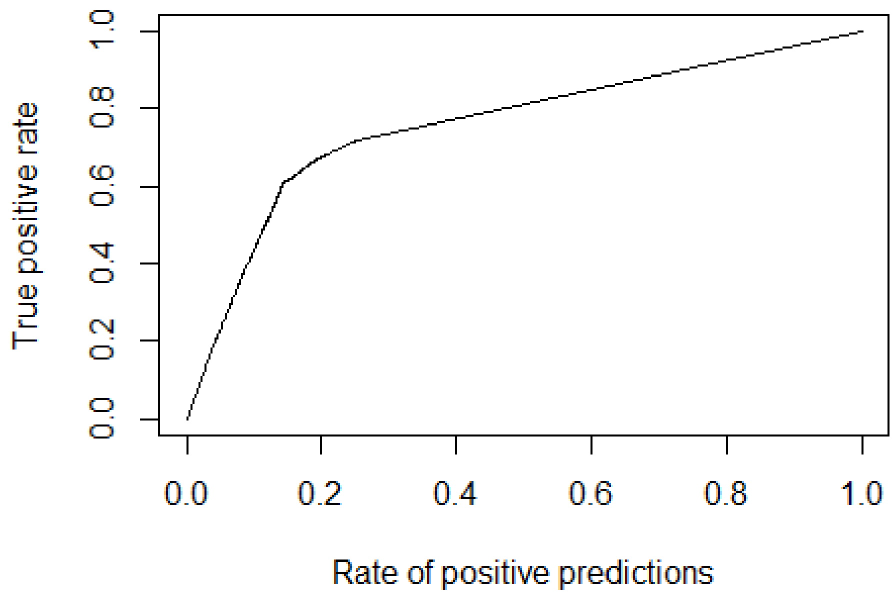
Drawing evaluation charts

```r
library(ROCR)
pred <- prediction(rf$votes[, 2],random_forest_data$Order_Conversion)
```

Gain Chart

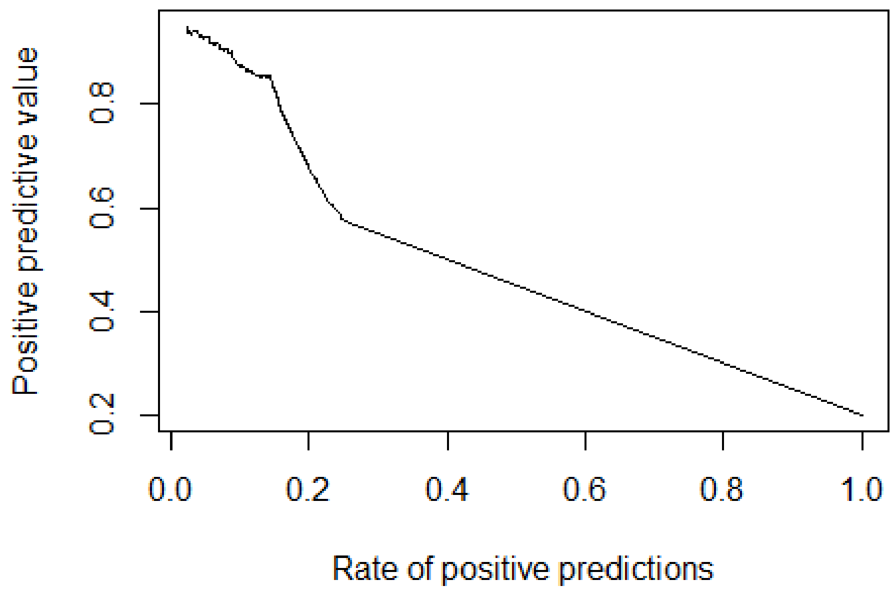
###Gain chart presents the percentage of captured positive responses as a function of selected percentage of a sample. ####Which is actually in our case

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



Response Chart

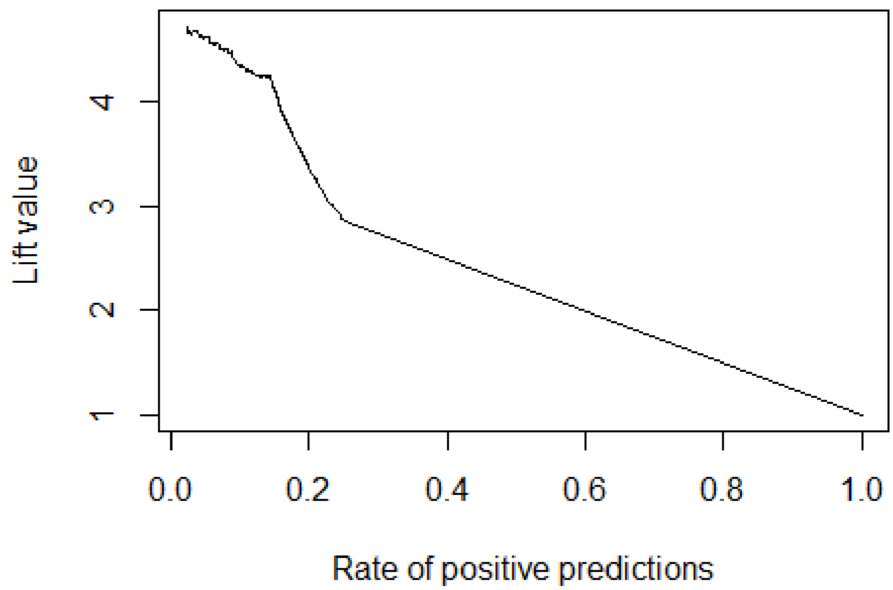
```
perf <- performance(pred, "ppv", "rpp")
plot(perf)
```



Lift Chart

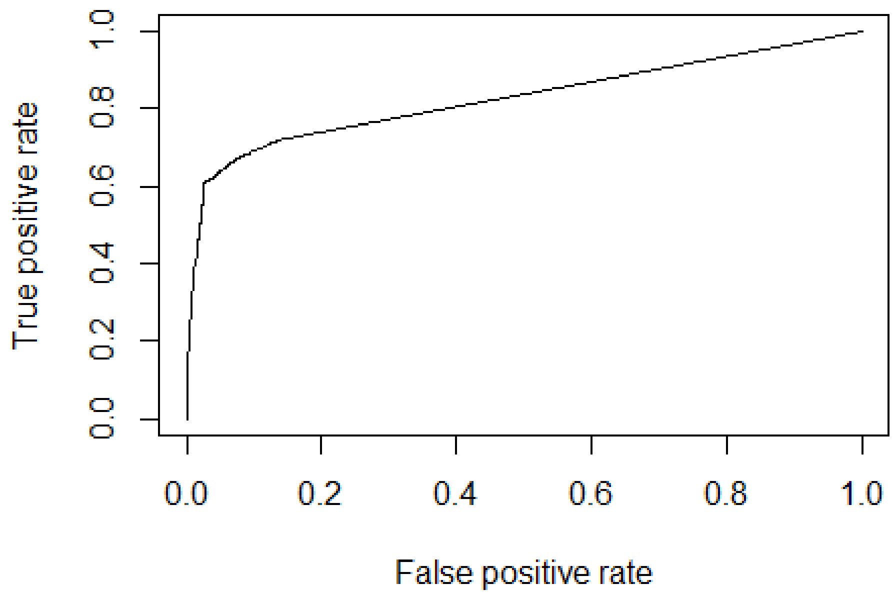
###The lift chart measures effectiveness of our predictive classification model comparing it with the baseline model.

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



ROC Curve - We can conclude that we have a smaller false alarm and also has higher recall,captures more retained(positve)

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



auc

##Since the AUC is 0.86 and the graph clearly shows the the model is accurate and a good model

```
auc <- performance(pred, "auc")
auc

## A performance instance
##   'Area under the ROC curve'

auc <- unlist(slot(auc, "y.values"))
auc

## [1] 0.8262461

## The AUC is 0.832, since its close to 1, we can say that the model is
accurate and good model
```

Performing cross validation for the Sample only dataset

##Cross validation for Decision Tree

```
glimpse(sample_only_dataset)

## Rows: 5,820
## Columns: 32
```

```
## $ CustomerCode     <fct> CC, M-1, M-1, M-1, M-1, CC, CC, M-1, M-1, CC, CC,
CC,~
## $ CountryName      <chr> "INDIA", "USA", "USA", "USA", "USA", "INDIA",
"INDIA"~
## $ USA              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ UK               <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ Italy            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ Belgium          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ Romania          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ Australia        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ India            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ QtyRequired      <dbl> 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 25, 1, 2, 35,
35, 35~
## $ ITEM_NAME        <chr> "HAND TUFTED", "HAND TUFTED", "HAND TUFTED",
"HAND TU~
## $ Hand_Tufted      <dbl> 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0,
0, 0,~
## $ Durry            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ Double_Back      <dbl> 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1,~
## $ Hand_Woven       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ Knotted          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1, 0,~
## $ Jacquard         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ Handloom         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ Other            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ ShapeName        <fct> REC, REC, REC, REC, REC, REC, REC, REC, REC, REC,
REC~
## $ REC              <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1,~
## $ Round            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ Square           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ AreaFt           <dbl> 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 80.0, 40.0,
108.0~
## $ Order_Conversion <fct> Converted, Converted, Converted, Converted,
Converted~
```

```
## $ Poland          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ Brazil          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ Canada          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ Israel          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ China           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ South_Africa    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~
## $ UAE             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,~

## Dividing the data set in k folds
attach(sample_only_dataset)

## The following objects are masked from sample_only_dataset (pos = 4):
##
##     AreaFt, Australia, Belgium, Brazil, Canada, China, CountryName,
##     CustomerCode, Double_Back, Durry, Hand_Tufted, Hand_Woven,
##     Handloom, India, Israel, Italy, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, Poland, QtyRequired, REC, Romania, Round,
##     ShapeName, South_Africa, Square, UAE, UK, USA

## The following objects are masked from Sample_onlyData_LM:
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, Order_Conversion,
##     QtyRequired, ShapeName

## The following objects are masked from dataset (pos = 6):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName

## The following objects are masked from dataset (pos = 8):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName

sample_only_dataset_CV <-
sample_only_dataset[sample(nrow(sample_only_dataset)),] # randomized the
position of the instances

attach(sample_only_dataset_CV)

## The following objects are masked from sample_only_dataset (pos = 3):
##
##     AreaFt, Australia, Belgium, Brazil, Canada, China, CountryName,
##     CustomerCode, Double_Back, Durry, Hand_Tufted, Hand_Woven,
```

```
##     Handloom, India, Israel, Italy, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, Poland, QtyRequired, REC, Romania, Round,
##     ShapeName, South_Africa, Square, UAE, UK, USA

## The following objects are masked from sample_only_dataset (pos = 5):
##
##     AreaFt, Australia, Belgium, Brazil, Canada, China, CountryName,
##     CustomerCode, Double_Back, Durry, Hand_Tufted, Hand_Woven,
##     Handloom, India, Israel, Italy, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, Poland, QtyRequired, REC, Romania, Round,
##     ShapeName, South_Africa, Square, UAE, UK, USA

## The following objects are masked from Sample_onlyData_LM:
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, Order_Conversion,
##     QtyRequired, ShapeName

## The following objects are masked from dataset (pos = 7):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName

## The following objects are masked from dataset (pos = 9):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName

k <- 10

nmethod <- 2
# This signifies the number of ML models we will be running on the K Fold
instances, since we are using only 1 ML model, we put it 1
folds <- cut(seq(1,nrow(sample_only_dataset_CV)), breaks = k, labels = FALSE)

model.err<- matrix(-1,k,nmethod,dimnames =
list(paste0("Fold",1:k),c("Decision Tree","Logistic Reg"))) #Here we are
creating a matrix to record each ML Error rate with different k values

#Since the first field is data, and currently we don't have any value we put
it as -1
#Removing the customer code in the dataset

sample_only_dataset_CV$CustomerCode <-
as.factor(sample_only_dataset_CV$CustomerCode)
sample_only_dataset_CV$CountryName <-
as.factor(sample_only_dataset_CV$CountryName)
attach(sample_only_dataset_CV)

## The following objects are masked from sample_only_dataset_CV (pos = 3):
##
##     AreaFt, Australia, Belgium, Brazil, Canada, China, CountryName,
```

```
##     CustomerCode, Double_Back, Durry, Hand_Tufted, Hand_Woven,
##     Handloom, India, Israel, Italy, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, Poland, QtyRequired, REC, Romania, Round,
##     ShapeName, South_Africa, Square, UAE, UK, USA

## The following objects are masked from sample_only_dataset (pos = 4):
##
##     AreaFt, Australia, Belgium, Brazil, Canada, China, CountryName,
##     CustomerCode, Double_Back, Durry, Hand_Tufted, Hand_Woven,
##     Handloom, India, Israel, Italy, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, Poland, QtyRequired, REC, Romania, Round,
##     ShapeName, South_Africa, Square, UAE, UK, USA

## The following objects are masked from sample_only_dataset (pos = 6):
##
##     AreaFt, Australia, Belgium, Brazil, Canada, China, CountryName,
##     CustomerCode, Double_Back, Durry, Hand_Tufted, Hand_Woven,
##     Handloom, India, Israel, Italy, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, Poland, QtyRequired, REC, Romania, Round,
##     ShapeName, South_Africa, Square, UAE, UK, USA

## The following objects are masked from Sample_onlyData_LM:
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, Order_Conversion,
##     QtyRequired, ShapeName

## The following objects are masked from dataset (pos = 8):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName

## The following objects are masked from dataset (pos = 10):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName
```

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

  formula = Order_Conversion~.

  #Creation of Decision Tree
  my_Tree <- rpart(formula, data = train)
  pred_class <- predict(my_Tree, newdata = test, type = "class")

  #Checking the test error:
  model.err[i][1] <- mean(pred_class != Order_Conversion)
```

```
  ##Random Forest
  rf <- randomForest(formula, data = random_forest_data, mtry =
sqrt(ncol(random_forest_data)-1), ntree = 50, proximity = T, importance = T)

  model.err[i][2] <- mean(rf$predicted !=
random_forest_data$Order_Conversion)

}

## Warning in model.err[i] <- `*vtmp*`: number of items to replace is not a
## multiple of replacement length

## Warning in model.err[i] <- `*vtmp*`: number of items to replace is not a
## multiple of replacement length

## Warning in model.err[i] <- `*vtmp*`: number of items to replace is not a
## multiple of replacement length

## Warning in model.err[i] <- `*vtmp*`: number of items to replace is not a
## multiple of replacement length

## Warning in model.err[i] <- `*vtmp*`: number of items to replace is not a
## multiple of replacement length

## Warning in model.err[i] <- `*vtmp*`: number of items to replace is not a
## multiple of replacement length

## Warning in model.err[i] <- `*vtmp*`: number of items to replace is not a
## multiple of replacement length

## Warning in model.err[i] <- `*vtmp*`: number of items to replace is not a
## multiple of replacement length

## Warning in model.err[i] <- `*vtmp*`: number of items to replace is not a
## multiple of replacement length

## Warning in model.err[i] <- `*vtmp*`: number of items to replace is not a
## multiple of replacement length

mean(model.err)

## [1] -0.3634708

model.err

##        Decision Tree Logistic Reg
## Fold1      0.2771478           -1
## Fold2      0.2565292           -1
## Fold3      0.2871134           -1
## Fold4      0.2726804           -1
```

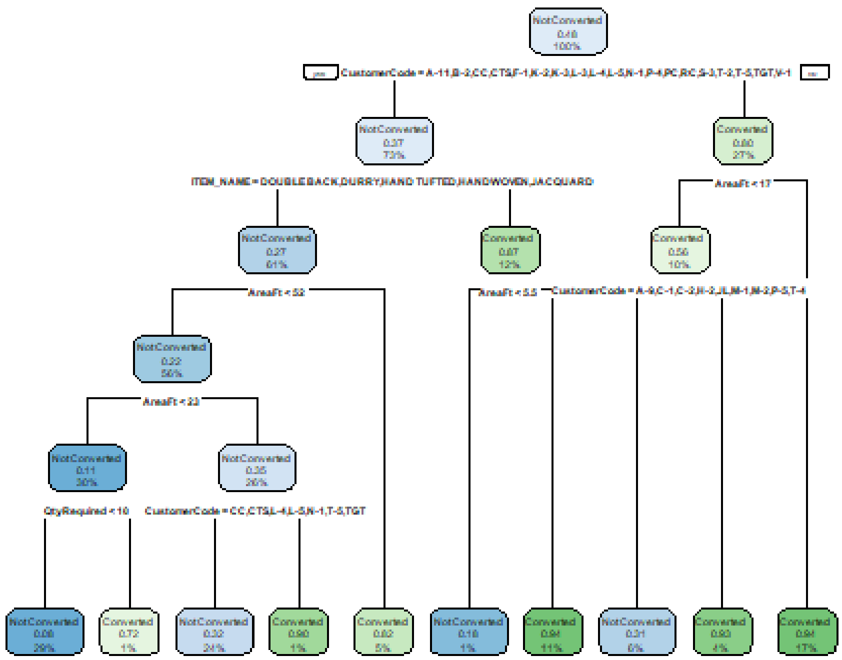
```
## Fold5      0.2874570          -1
## Fold6      0.2689003          -1
## Fold7      0.2678694          -1
## Fold8      0.2630584          -1
## Fold9      0.2682131          -1
## Fold10     0.2816151          -1
```

##BALANCED DATA ###Applying Decision Trees for Balanced data

```
set.seed(1234)
nrow(balanced_sample_dataset)

## [1] 9000

#There are no Null values
sum(is.na(balanced_sample_dataset))

## [1] 0

indx1 <- sample(2, nrow(balanced_sample_dataset), replace = TRUE, prob =
c(0.8,0.2))

#Splitting the train and test data
train1 <- balanced_sample_dataset[indx == 1, ]
test1 <- balanced_sample_dataset[indx == 2,]

sum(is.na(train1))

## [1] 0

sum(is.na(balanced_sample_dataset))

## [1] 0

nrow(train1)/nrow(test1) #4:1

## [1] 3.980631

attach(balanced_sample_dataset)

## The following objects are masked from sample_only_dataset_CV (pos = 3):
##
##     AreaFt, Australia, Belgium, Brazil, Canada, China, CountryName,
##     CustomerCode, Double_Back, Durry, Hand_Tufted, Hand_Woven,
##     Handloom, India, Israel, Italy, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, Poland, QtyRequired, REC, Romania, Round,
##     ShapeName, South_Africa, Square, UAE, UK, USA

## The following objects are masked from sample_only_dataset_CV (pos = 4):
##
##     AreaFt, Australia, Belgium, Brazil, Canada, China, CountryName,
##     CustomerCode, Double_Back, Durry, Hand_Tufted, Hand_Woven,
##     Handloom, India, Israel, Italy, ITEM_NAME, Jacquard, Knotted,
```

```
##     Order_Conversion, Other, Poland, QtyRequired, REC, Romania, Round,
##     ShapeName, South_Africa, Square, UAE, UK, USA

## The following objects are masked from sample_only_dataset (pos = 5):
##
##     AreaFt, Australia, Belgium, Brazil, Canada, China, CountryName,
##     CustomerCode, Double_Back, Durry, Hand_Tufted, Hand_Woven,
##     Handloom, India, Israel, Italy, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, Poland, QtyRequired, REC, Romania, Round,
##     ShapeName, South_Africa, Square, UAE, UK, USA

## The following objects are masked from sample_only_dataset (pos = 7):
##
##     AreaFt, Australia, Belgium, Brazil, Canada, China, CountryName,
##     CustomerCode, Double_Back, Durry, Hand_Tufted, Hand_Woven,
##     Handloom, India, Israel, Italy, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, Poland, QtyRequired, REC, Romania, Round,
##     ShapeName, South_Africa, Square, UAE, UK, USA

## The following objects are masked from Sample_onlyData_LM:
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, Order_Conversion,
##     QtyRequired, ShapeName

## The following objects are masked from dataset (pos = 9):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName

## The following objects are masked from dataset (pos = 11):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName
```

```
formula1 = Order_Conversion ~ .

mytree1 <- rpart(formula1, data = train1)

rpart.plot(mytree1)
```

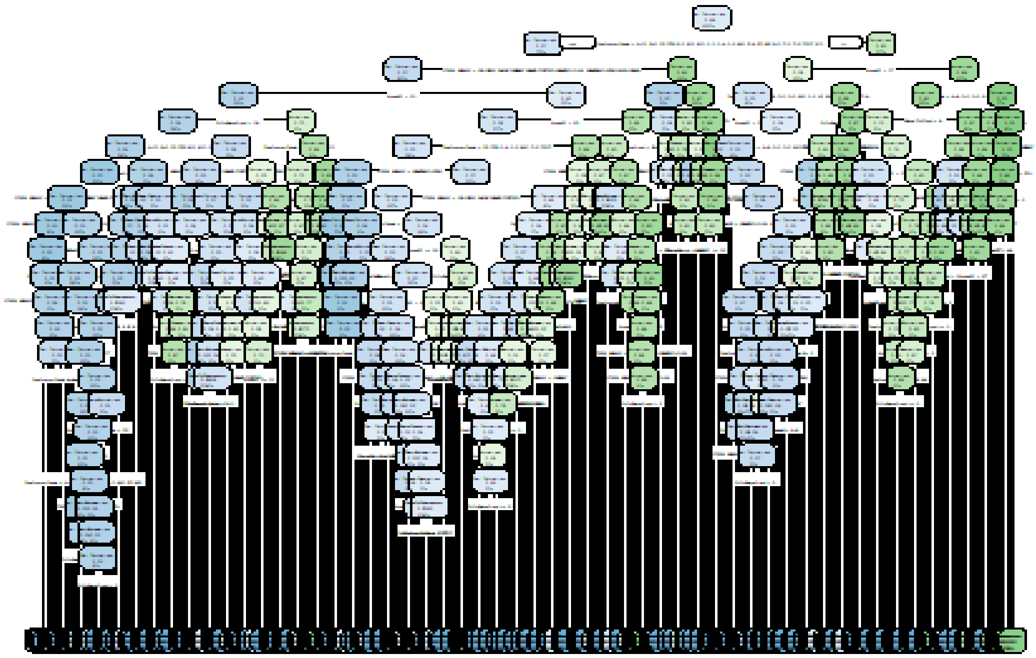


```
tree_pred_train1 <- predict(mytree1, train1, type = "class")

train_Error1 <- mean(tree_pred_train1 != train1$Order_Conversion)

train_Error1 # <- The train error is 17%

## [1] 0.1548728

sum(is.na(tree_pred_train1))

## [1] 0

testPred1 <- predict(mytree1, newdata = test1, type = "class")

#Checking the test error:
mean(testPred1 != test1$Order_Conversion)

## [1] 0.1615938

confusionMatrix(testPred1, test1$Order_Conversion)

## Confusion Matrix and Statistics
##
##                Reference
## Prediction      Not Converted Converted
##   Not Converted           875       223
##   Converted                69       640
##
```

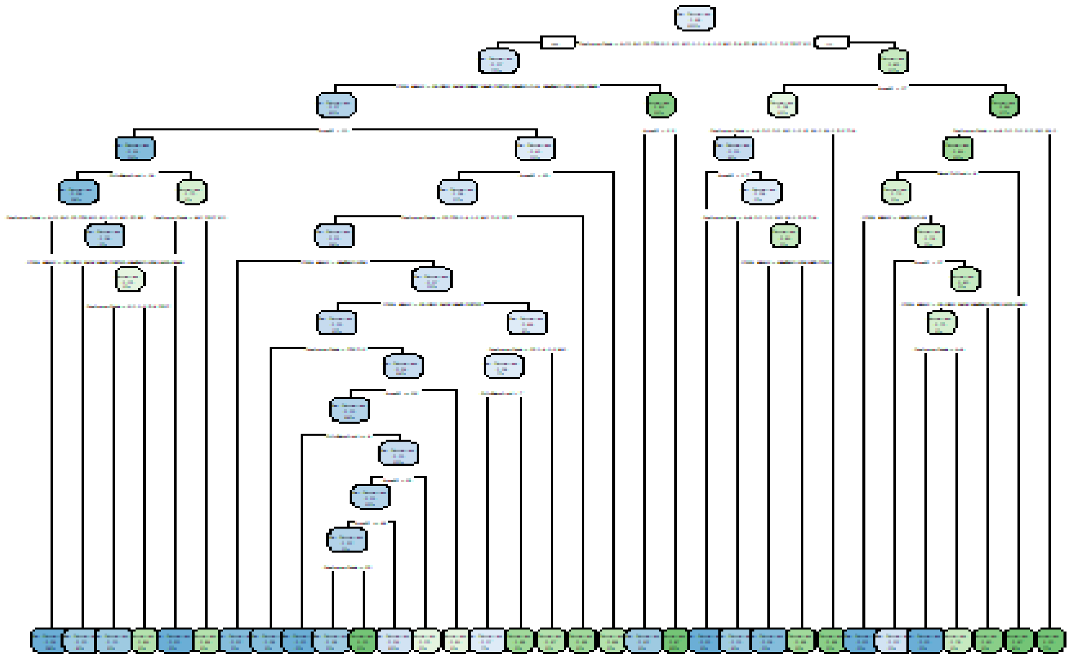
```
##                Accuracy : 0.8384
##                  95% CI : (0.8206, 0.8551)
##     No Information Rate : 0.5224
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6737
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9269
##             Specificity : 0.7416
##          Pos Pred Value : 0.7969
##          Neg Pred Value : 0.9027
##              Prevalence : 0.5224
##          Detection Rate : 0.4842
##    Detection Prevalence : 0.6076
##       Balanced Accuracy : 0.8343
##
##        'Positive' Class : Not Converted
##

#The test error is 17.9%

As we can see the accuracy is 83% for a balanced data for decision tree, and
less than the one with unbalanced data taking the not converted as the
positive class
```

Tuning the Hyper parameters /Changing the parameters of the decision tree (Balanced data)

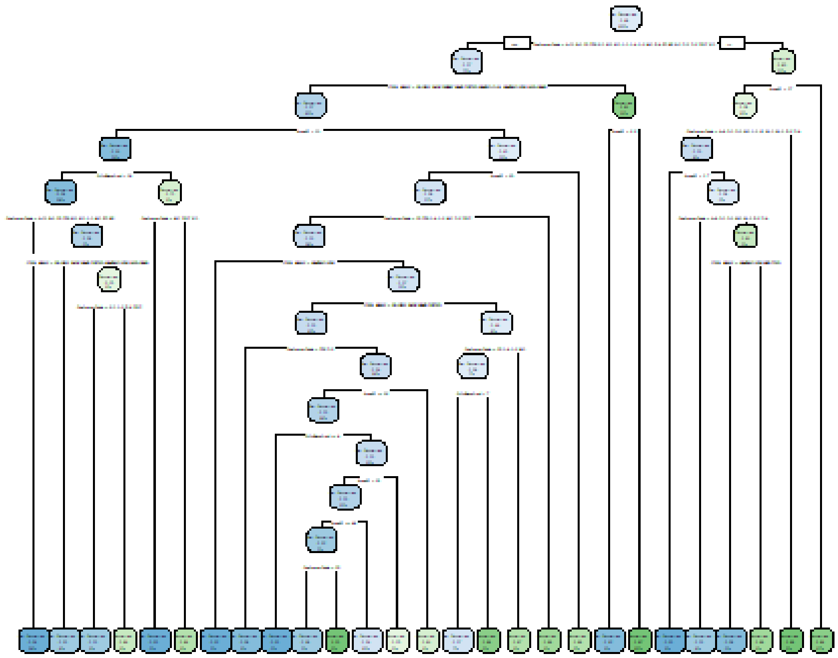
```
myTreeb1 <- rpart(formula1, train1, parms = list(split =
"information"),control = rpart.control(minbucket = 0,minsplit = 0, cp = -1))

rpart.plot(myTreeb1)

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



```
## Tuning the decision tree with different parameter values

myTreeb1 <- rpart(formula1, train1, parms = list(split =
"information"),control = rpart.control(minbucket = 3,minsplit = 5, cp =
0.001))

rpart.plot(myTreeb1)

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



```
myTreeb2 <- rpart(formula1, train1, parms = list(split =
"information"),control = rpart.control(minbucket = 10,minsplit = 5, cp =
0.001))

rpart.plot(myTreeb2)

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



```
tree2_pred_trainb1 <- predict(myTreeb2, train1, type = "class")

train2_Error_b2 <- mean(tree2_pred_trainb1 != train1$Order_Conversion)

train2_Error_b2 # <- The train error is 12.66%

## [1] 0.1290143

sum(is.na(tree_pred_train1))

## [1] 0

testPredb2 <- predict(myTreeb2, newdata = test1, type = "class")

#Checking the test error:
mean(testPredb2 != test$Order_Conversion)

## Warning in `!=.default`(testPredb2, test$Order_Conversion): longer object
length
## is not a multiple of shorter object length

## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple
of
## shorter object length

## [1] 0.4537908
```

###Neural network model using nnet for Balanced dataset####

```
sample_balanced_demo <-
balanced_sample_dataset[sample(nrow(balanced_sample_dataset)),]
sample_balanced_demo <- subset (sample_balanced_demo, select = -c(USA, UK,
Italy, Belgium, Romania, Australia, India))
sample_balanced_nn <-
sample_balanced_demo[sample(nrow(sample_balanced_demo)),]
colnames(sample_balanced_nn)

##  [1] "CustomerCode"     "CountryName"      "QtyRequired"      "ITEM_NAME"
##  [5] "Hand_Tufted"      "Durry"            "Double_Back"      "Hand_Woven"
##  [9] "Knotted"          "Jacquard"         "Handloom"         "Other"
## [13] "ShapeName"        "REC"              "Round"            "Square"
## [17] "AreaFt"           "Order_Conversion" "Poland"           "Brazil"
## [21] "Canada"           "Israel"           "China"
"South_Africa"
## [25] "UAE"

##To check NA values
table(is.na(sample_balanced_nn))

##
##  FALSE
## 225000

lapply(sample_balanced_nn, function(x) { length(which(is.na(x)))})

## $CustomerCode
## [1] 0
##
## $CountryName
## [1] 0
##
## $QtyRequired
## [1] 0
##
## $ITEM_NAME
## [1] 0
##
## $Hand_Tufted
## [1] 0
##
## $Durry
## [1] 0
##
## $Double_Back
## [1] 0
##
## $Hand_Woven
## [1] 0
```

```
##
## $Knotted
## [1] 0
##
## $Jacquard
## [1] 0
##
## $Handloom
## [1] 0
##
## $Other
## [1] 0
##
## $ShapeName
## [1] 0
##
## $REC
## [1] 0
##
## $Round
## [1] 0
##
## $Square
## [1] 0
##
## $AreaFt
## [1] 0
##
## $Order_Conversion
## [1] 0
##
## $Poland
## [1] 0
##
## $Brazil
## [1] 0
##
## $Canada
## [1] 0
##
## $Israel
## [1] 0
##
## $China
## [1] 0
##
## $South_Africa
## [1] 0
##
```

```
## $UAE
## [1] 0
```

```r
#There are few variables that needs their datatypes to be changed
attach(sample_balanced_nn)
```

```
## The following objects are masked from balanced_sample_dataset:
## 
##     AreaFt, Brazil, Canada, China, CountryName, CustomerCode,
##     Double_Back, Durry, Hand_Tufted, Hand_Woven, Handloom, Israel,
##     ITEM_NAME, Jacquard, Knotted, Order_Conversion, Other, Poland,
##     QtyRequired, REC, Round, ShapeName, South_Africa, Square, UAE

## The following objects are masked from sample_only_dataset_CV (pos = 4):
## 
##     AreaFt, Brazil, Canada, China, CountryName, CustomerCode,
##     Double_Back, Durry, Hand_Tufted, Hand_Woven, Handloom, Israel,
##     ITEM_NAME, Jacquard, Knotted, Order_Conversion, Other, Poland,
##     QtyRequired, REC, Round, ShapeName, South_Africa, Square, UAE

## The following objects are masked from sample_only_dataset_CV (pos = 5):
## 
##     AreaFt, Brazil, Canada, China, CountryName, CustomerCode,
##     Double_Back, Durry, Hand_Tufted, Hand_Woven, Handloom, Israel,
##     ITEM_NAME, Jacquard, Knotted, Order_Conversion, Other, Poland,
##     QtyRequired, REC, Round, ShapeName, South_Africa, Square, UAE

## The following objects are masked from sample_only_dataset (pos = 6):
## 
##     AreaFt, Brazil, Canada, China, CountryName, CustomerCode,
##     Double_Back, Durry, Hand_Tufted, Hand_Woven, Handloom, Israel,
##     ITEM_NAME, Jacquard, Knotted, Order_Conversion, Other, Poland,
##     QtyRequired, REC, Round, ShapeName, South_Africa, Square, UAE

## The following objects are masked from sample_only_dataset (pos = 8):
## 
##     AreaFt, Brazil, Canada, China, CountryName, CustomerCode,
##     Double_Back, Durry, Hand_Tufted, Hand_Woven, Handloom, Israel,
##     ITEM_NAME, Jacquard, Knotted, Order_Conversion, Other, Poland,
##     QtyRequired, REC, Round, ShapeName, South_Africa, Square, UAE

## The following objects are masked from Sample_onlyData_LM:
## 
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, Order_Conversion,
##     QtyRequired, ShapeName

## The following objects are masked from dataset (pos = 10):
## 
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName
```

```
## The following objects are masked from dataset (pos = 12):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName
```

```
sample_balanced_nn$CustomerCode<-
as.numeric(factor(as.matrix(sample_balanced_nn$CustomerCode)))
sample_balanced_nn$CountryName<-
as.numeric(factor(as.matrix(sample_balanced_nn$CountryName)))
sample_balanced_nn$ITEM_NAME<-
as.numeric(factor(as.matrix(sample_balanced_nn$ITEM_NAME)))
sample_balanced_nn$ShapeName<-
as.numeric(factor(as.matrix(sample_balanced_nn$ShapeName)))

nrow(sample_balanced_nn)
```

```
## [1] 9000
```

```
head(sample_balanced_nn)
```

```
##      CustomerCode CountryName QtyRequired ITEM_NAME Hand_Tufted Durry
## 819            27           2           3         2           0     1
## 4799            7           6           1         9           0     0
## 1812            7           6           1         2           0     1
## 6285           32          13           1         2           0     1
## 410             7           6           1         4           1     0
## 2792            7           6           1         2           0     1
##      Double_Back Hand_Woven Knotted Jacquard Handloom Other ShapeName REC
Round
## 819            0          0       0        0        0     0         1   1
0
## 4799           0          0       1        0        0     0         1   1
0
## 1812           0          0       0        0        0     0         1   1
0
## 6285           0          0       0        0        0     0         1   1
0
## 410            0          0       0        0        0     0         1   1
0
## 2792           0          0       0        0        0     0         1   1
0
##      Square  AreaFt Order_Conversion Poland Brazil Canada Israel China
## 819       0  2.0000    Not Converted      0      0      0      0     0
## 4799      0  6.0000        Converted      0      0      0      0     0
## 1812      0 24.0000    Not Converted      0      0      0      0     0
## 6285      0  8.4375        Converted      0      0      0      0     0
## 410       0  4.0000    Not Converted      0      0      0      0     0
## 2792      0  6.0000    Not Converted      0      0      0      0     0
##      South_Africa UAE
## 819             0   0
## 4799            0   0
```

```
## 1812            0   0
## 6285            0   0
## 410             0   0
## 2792            0   0

##Normalize data before training a neural network###
###myscale() function uses min-max transformation to normalize variable x
myscale_bnn <- function(x)
{
  (x - min(x)) / (max(x) - min(x))
}

sample_balanced_nn <- sample_balanced_nn %>% mutate_if(is.numeric,
myscale_bnn)

###Splitting the normalized data into train and test set
set.seed(1234)
indx_b <- sample(2, nrow(sample_balanced_nn), replace = T, prob = c(0.7,0.3))
trainb <- sample_balanced_nn[indx_b == 1,]
testb <- sample_balanced_nn[indx_b == 2,]


###Using nnet function to build neural network model
attach(sample_balanced_nn)

## The following objects are masked from sample_balanced_nn (pos = 3):
##
##     AreaFt, Brazil, Canada, China, CountryName, CustomerCode,
##     Double_Back, Durry, Hand_Tufted, Hand_Woven, Handloom, Israel,
##     ITEM_NAME, Jacquard, Knotted, Order_Conversion, Other, Poland,
##     QtyRequired, REC, Round, ShapeName, South_Africa, Square, UAE

## The following objects are masked from balanced_sample_dataset:
##
##     AreaFt, Brazil, Canada, China, CountryName, CustomerCode,
##     Double_Back, Durry, Hand_Tufted, Hand_Woven, Handloom, Israel,
##     ITEM_NAME, Jacquard, Knotted, Order_Conversion, Other, Poland,
##     QtyRequired, REC, Round, ShapeName, South_Africa, Square, UAE

## The following objects are masked from sample_only_dataset_CV (pos = 5):
##
##     AreaFt, Brazil, Canada, China, CountryName, CustomerCode,
##     Double_Back, Durry, Hand_Tufted, Hand_Woven, Handloom, Israel,
##     ITEM_NAME, Jacquard, Knotted, Order_Conversion, Other, Poland,
##     QtyRequired, REC, Round, ShapeName, South_Africa, Square, UAE

## The following objects are masked from sample_only_dataset_CV (pos = 6):
##
##     AreaFt, Brazil, Canada, China, CountryName, CustomerCode,
##     Double_Back, Durry, Hand_Tufted, Hand_Woven, Handloom, Israel,
```

```
##     ITEM_NAME, Jacquard, Knotted, Order_Conversion, Other, Poland,
##     QtyRequired, REC, Round, ShapeName, South_Africa, Square, UAE

## The following objects are masked from sample_only_dataset (pos = 7):
##
##     AreaFt, Brazil, Canada, China, CountryName, CustomerCode,
##     Double_Back, Durry, Hand_Tufted, Hand_Woven, Handloom, Israel,
##     ITEM_NAME, Jacquard, Knotted, Order_Conversion, Other, Poland,
##     QtyRequired, REC, Round, ShapeName, South_Africa, Square, UAE

## The following objects are masked from sample_only_dataset (pos = 9):
##
##     AreaFt, Brazil, Canada, China, CountryName, CustomerCode,
##     Double_Back, Durry, Hand_Tufted, Hand_Woven, Handloom, Israel,
##     ITEM_NAME, Jacquard, Knotted, Order_Conversion, Other, Poland,
##     QtyRequired, REC, Round, ShapeName, South_Africa, Square, UAE

## The following objects are masked from Sample_onlyData_LM:
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, Order_Conversion,
##     QtyRequired, ShapeName

## The following objects are masked from dataset (pos = 11):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName

## The following objects are masked from dataset (pos = 13):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName
```

```
library(nnet)
```

```
## Warning: package 'nnet' was built under R version 4.1.3
```

```
nnModel_b <- nnet(Order_Conversion ~., data = trainb, linout = FALSE, size =
10, hidden =3, decay = 0.01, maxit = 1000)
```

```
## # weights:  261
## initial  value 6358.601780
## iter  10 value 3369.805705
## iter  20 value 2745.045107
## iter  30 value 2544.767858
## iter  40 value 2418.663219
## iter  50 value 2326.799261
## iter  60 value 2279.284291
## iter  70 value 2247.757329
## iter  80 value 2220.721194
## iter  90 value 2204.345534
## iter 100 value 2179.076436
## iter 110 value 2162.585824
```

```
## iter 120 value 2152.483659
## iter 130 value 2143.751760
## iter 140 value 2133.901139
## iter 150 value 2120.579527
## iter 160 value 2110.110766
## iter 170 value 2100.670735
## iter 180 value 2094.610805
## iter 190 value 2087.297675
## iter 200 value 2079.335869
## iter 210 value 2067.549318
## iter 220 value 2058.344806
## iter 230 value 2051.779280
## iter 240 value 2046.440000
## iter 250 value 2041.799954
## iter 260 value 2037.559861
## iter 270 value 2033.098521
## iter 280 value 2030.206453
## iter 290 value 2026.953774
## iter 300 value 2023.859623
## iter 310 value 2021.280979
## iter 320 value 2018.806619
## iter 330 value 2016.261150
## iter 340 value 2014.086814
## iter 350 value 2011.290164
## iter 360 value 2008.487782
## iter 370 value 2005.688900
## iter 380 value 2001.258161
## iter 390 value 1997.745981
## iter 400 value 1995.504974
## iter 410 value 1993.344289
## iter 420 value 1991.727977
## iter 430 value 1990.448384
## iter 440 value 1989.331289
## iter 450 value 1988.008157
## iter 460 value 1985.986561
## iter 470 value 1982.064934
## iter 480 value 1978.750187
## iter 490 value 1975.900958
## iter 500 value 1973.631592
## iter 510 value 1972.252743
## iter 520 value 1971.292261
## iter 530 value 1970.822322
## iter 540 value 1970.319179
## iter 550 value 1969.785205
## iter 560 value 1968.792275
## iter 570 value 1966.925448
## iter 580 value 1965.626911
## iter 590 value 1964.530057
## iter 600 value 1963.889048
## iter 610 value 1963.534418
```

```
## iter 620 value 1962.952581
## iter 630 value 1961.886659
## iter 640 value 1960.126137
## iter 650 value 1958.166727
## iter 660 value 1956.473777
## iter 670 value 1955.195581
## iter 680 value 1954.308908
## iter 690 value 1953.900452
## iter 700 value 1953.552907
## iter 710 value 1953.359797
## iter 720 value 1953.254882
## iter 730 value 1953.182803
## iter 740 value 1952.951027
## iter 750 value 1952.409391
## iter 760 value 1951.530698
## iter 770 value 1950.913353
## iter 780 value 1950.596381
## iter 790 value 1950.470765
## iter 800 value 1950.391560
## iter 810 value 1950.300033
## iter 820 value 1950.121837
## iter 830 value 1949.382342
## iter 840 value 1948.519263
## iter 850 value 1948.004725
## iter 860 value 1947.675924
## iter 870 value 1947.226089
## iter 880 value 1947.000854
## iter 890 value 1946.802539
## iter 900 value 1946.681714
## iter 910 value 1946.610827
## iter 920 value 1946.544067
## iter 930 value 1946.497090
## iter 940 value 1946.431133
## iter 950 value 1946.305138
## iter 960 value 1946.110080
## iter 970 value 1945.873162
## iter 980 value 1945.659955
## iter 990 value 1945.453261
## iter1000 value 1945.129833
## final  value 1945.129833
## stopped after 1000 iterations
```

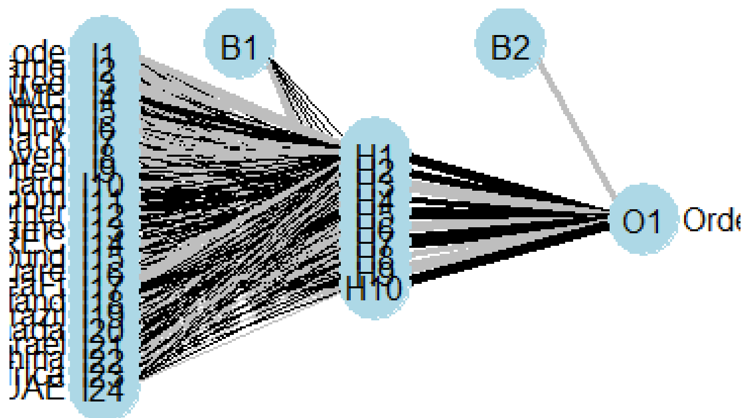
```
summary(nnModel_b)
```

```
## a 24-10-1 network with 261 weights
## options were - entropy fitting  decay=0.01
##   b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1  i8->h1
i9->h1
##    3.89  -17.54   -5.28    0.77    9.28    7.22   -7.13   -0.35   -6.50
0.21
```

```
## i10->h1 i11->h1 i12->h1 i13->h1 i14->h1 i15->h1 i16->h1 i17->h1 i18->h1 
i19->h1 
##    3.62    1.94    4.88    1.36    1.96    1.13    0.80   -9.32    0.01 
0.01 
## i20->h1 i21->h1 i22->h1 i23->h1 i24->h1 
##    0.26    0.08    0.00   -2.00    0.00 
##   b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2 
i9->h2 
##    0.98    0.40   -5.38   12.30    3.68   -0.02    3.58   -0.43   -0.90 
-1.05 
## i10->h2 i11->h2 i12->h2 i13->h2 i14->h2 i15->h2 i16->h2 i17->h2 i18->h2 
i19->h2 
##   -2.65    1.23    1.22    0.27   -0.37    2.16   -0.81   -0.72    0.08 
0.09 
## i20->h2 i21->h2 i22->h2 i23->h2 i24->h2 
##   -1.08    0.90    0.00   -0.05    0.00 
##   b->h3  i1->h3  i2->h3  i3->h3  i4->h3  i5->h3  i6->h3  i7->h3  i8->h3 
i9->h3 
##    1.63   -7.23   -2.56  -24.41   -0.78   -2.95   -1.53   -1.97   -0.21 
1.69 
## i10->h3 i11->h3 i12->h3 i13->h3 i14->h3 i15->h3 i16->h3 i17->h3 i18->h3 
i19->h3 
##   -0.12    1.33    5.40   -0.60    2.19    0.08   -0.64   -9.95    0.05 
0.38 
## i20->h3 i21->h3 i22->h3 i23->h3 i24->h3 
##   -1.45    0.06    0.00   -2.43    0.00 
##   b->h4  i1->h4  i2->h4  i3->h4  i4->h4  i5->h4  i6->h4  i7->h4  i8->h4 
i9->h4 
##    0.15    0.38   -0.07   -4.33   -2.45   -1.52   -0.10    1.74    2.80 
-4.12 
## i10->h4 i11->h4 i12->h4 i13->h4 i14->h4 i15->h4 i16->h4 i17->h4 i18->h4 
i19->h4 
##   -2.00    2.31    1.05    0.01    0.87   -1.45    0.73  -15.87    0.26 
-0.14 
## i20->h4 i21->h4 i22->h4 i23->h4 i24->h4 
##    0.39    1.65    0.00    0.34    0.00 
##   b->h5  i1->h5  i2->h5  i3->h5  i4->h5  i5->h5  i6->h5  i7->h5  i8->h5 
i9->h5 
##   -4.08    4.81    1.87   -8.68    1.14   -5.36    0.64    5.73   -2.89 
1.24 
## i10->h5 i11->h5 i12->h5 i13->h5 i14->h5 i15->h5 i16->h5 i17->h5 i18->h5 
i19->h5 
##    0.18   -3.41   -0.21    1.08   -5.76    1.21    0.47   12.71    0.75 
-0.01 
## i20->h5 i21->h5 i22->h5 i23->h5 i24->h5 
##    0.59   -0.03    0.00    0.22    0.00 
##   b->h6  i1->h6  i2->h6  i3->h6  i4->h6  i5->h6  i6->h6  i7->h6  i8->h6 
i9->h6 
##    1.62    5.76   -5.80    8.46   -2.99    4.17   -1.07    0.81   -6.06 
2.62
```

```
## i10->h6 i11->h6 i12->h6 i13->h6 i14->h6 i15->h6 i16->h6 i17->h6 i18->h6
i19->h6
##   -2.31    1.96    1.48    1.28   -1.03    2.73   -0.10  -15.91   -0.21
0.42
## i20->h6 i21->h6 i22->h6 i23->h6 i24->h6
##   -1.95   -0.34    0.00   -0.15    0.00
##   b->h7  i1->h7  i2->h7  i3->h7  i4->h7  i5->h7  i6->h7  i7->h7  i8->h7
i9->h7
##   -5.13    7.07    2.91  -19.17   -1.84    1.32    2.00   -6.36   -0.14
1.07
## i10->h7 i11->h7 i12->h7 i13->h7 i14->h7 i15->h7 i16->h7 i17->h7 i18->h7
i19->h7
##    0.95    0.01   -3.96   -1.38   -2.69   -2.13   -0.31    3.99   -0.01
-0.39
## i20->h7 i21->h7 i22->h7 i23->h7 i24->h7
##    0.74    0.34    0.00   -0.92    0.00
##   b->h8  i1->h8  i2->h8  i3->h8  i4->h8  i5->h8  i6->h8  i7->h8  i8->h8
i9->h8
##   -2.76  -18.55   10.80   10.38   -4.60    1.08  -11.72    0.64   -0.61
0.45
## i10->h8 i11->h8 i12->h8 i13->h8 i14->h8 i15->h8 i16->h8 i17->h8 i18->h8
i19->h8
##    1.96    2.32    3.14    0.07   -1.73   -2.21    1.18   24.66   -0.28
-0.01
## i20->h8 i21->h8 i22->h8 i23->h8 i24->h8
##   -1.05    0.09    0.00    0.43    0.00
##   b->h9  i1->h9  i2->h9  i3->h9  i4->h9  i5->h9  i6->h9  i7->h9  i8->h9
i9->h9
##   -7.41    8.37    7.12    2.76    9.01   -9.61    3.25    1.84    0.69
-1.65
## i10->h9 i11->h9 i12->h9 i13->h9 i14->h9 i15->h9 i16->h9 i17->h9 i18->h9
i19->h9
##    1.16   -4.26    1.16   -0.68   -6.74    0.01   -0.69  -10.67    0.02
0.00
## i20->h9 i21->h9 i22->h9 i23->h9 i24->h9
##    0.55   -0.04    0.00    0.01    0.00
##   b->h10  i1->h10  i2->h10  i3->h10  i4->h10  i5->h10  i6->h10  i7->h10
##    -5.64    -8.96    12.56     7.93     1.01    -5.09    -4.17    -7.09
##  i8->h10  i9->h10 i10->h10 i11->h10 i12->h10 i13->h10 i14->h10 i15->h10
##     3.67     6.70    -1.12    -4.95     6.41     0.16    -4.75    -2.10
## i16->h10 i17->h10 i18->h10 i19->h10 i20->h10 i21->h10 i22->h10 i23->h10
##     1.21    13.36     0.00     0.00    -0.01     0.00     0.00     0.40
## i24->h10
##     0.00
##   b->o  h1->o  h2->o  h3->o  h4->o  h5->o  h6->o  h7->o  h8->o  h9->o h10-
>o
## -11.49  19.75  13.46 -24.18  11.92  23.54 -17.25  25.23 -10.03 -24.08
20.14
```

```r
##You are using wts to get the best weights found and fitted.values to get
the fitted values on training data
# nnModel_b$wts
# nnModel_b$fitted.values

##To draw nnet model
library(NeuralNetTools)

## Warning: package 'NeuralNetTools' was built under R version 4.1.3

plotnet(nnModel_b)
```




```r
##Neural network model used to predict test instances
nn.preds = predict(nnModel_b, testb)

##Notice we still have results between 0 and 1 that are more like
probabilities of belonging to each class. To get the predicted classes we can
use change the type argument.

nn.preds = as.factor(predict(nnModel_b, testb, type = "class"))

##Confusion Matrix
ConfMatrix_b <- table(nn.preds, testb$Order_Conversion, dnn =
c("predicted","actual"))
print(ConfMatrix_b)
```

```
##                 actual
## predicted       Not Converted Converted
##   Converted               115      1042
##   Not Converted          1240       264

##Check performance of neural network model
error_metric = function(ConfMatrix_b)
{
TN = ConfMatrix_b[1,1]
TP = ConfMatrix_b[2,2]
FN = ConfMatrix_b[1,2]
FP = ConfMatrix_b[2,1]
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("precision" = precision,
"recall" = recall,
"falsepositiverate" = falsePositiveRate,
"falsenegativerate" = falseNegativeRate,
"error" = error)
return(modelPerf)
}
outPutlist_b <- error_metric(ConfMatrix_b)
library(plyr)

## ------------------------------------------------------------------------------
----

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first,
then dplyr:
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------------
----

##
## Attaching package: 'plyr'

## The following objects are masked from 'package:Hmisc':
##
##     is.discrete, summarize

## The following object is masked from 'package:purrr':
##
##     compact

## The following objects are masked from 'package:dplyr':
##
```

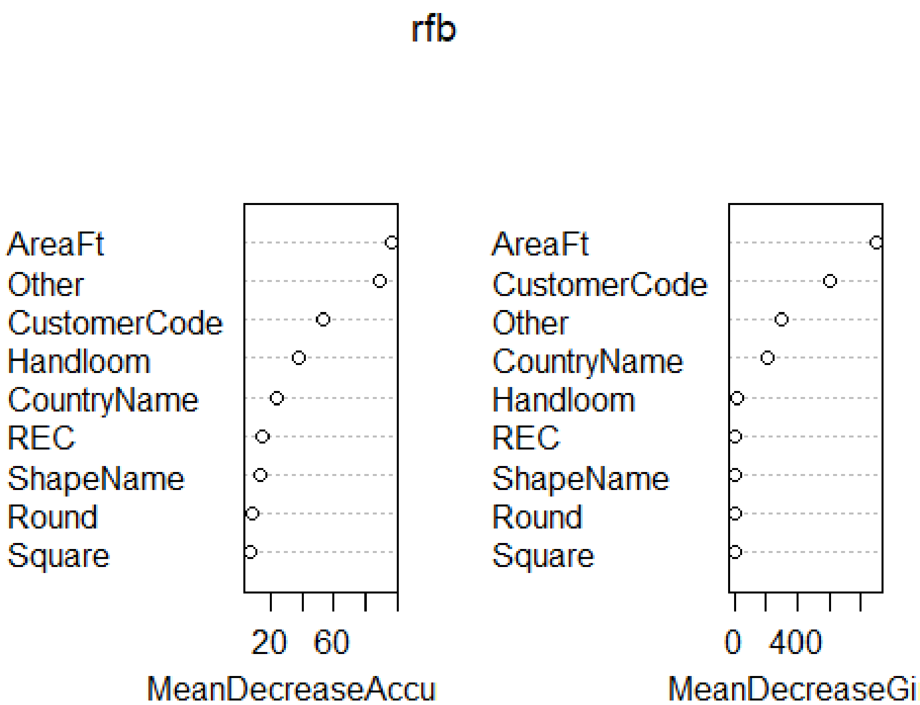
```
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

df_b <- ldply(outPutlist_b, data.frame)
setNames(df_b, c("", "Values"))

##                        Values
## 1         precision 0.1755319
## 2            recall 0.2021440
## 3 falsepositiverate 0.9151292
## 4 falsenegativerate 0.7978560
## 5             error 0.8575723
```

###Random Forest for Balanced data

```
ntreeb <- 100
set.seed(123)

random_forest_data_bal <-
sample_balanced_demo[sample(nrow(sample_balanced_demo)),]
random_forest_data_bal <- subset (random_forest_data_bal, select = -c(3:10))
random_forest_data_bal <- subset (random_forest_data_bal, select = -c(11:17))


str(random_forest_data_bal)

## 'data.frame':    9000 obs. of  10 variables:
##  $ CustomerCode    : Factor w/ 34 levels "A-11","A-9","B-2",..: 13 6 7 24
32 23 7 13 7 7 ...
##  $ CountryName     : chr  "USA" "USA" "INDIA" "USA" ...
##  $ Handloom        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Other           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ ShapeName       : Factor w/ 3 levels "REC","ROUND",..: 1 1 1 1 1 1 1 1
1 1 ...
##  $ REC             : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Round           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Square          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ AreaFt          : num  80 6 24 24 40 ...
##  $ Order_Conversion: Factor w/ 2 levels "Not Converted",..: 2 1 1 1 2 1 1
2 2 2 ...

colnames(random_forest_data_bal)

##  [1] "CustomerCode"     "CountryName"      "Handloom"         "Other"
##  [5] "ShapeName"        "REC"              "Round"            "Square"
##  [9] "AreaFt"           "Order_Conversion"

myFormula = Order_Conversion~ .

##Building random forest model
rfb <- randomForest(myFormula, data = random_forest_data_bal, mtry =
```

```r
sqrt(ncol(random_forest_data_bal)-1), ntreeb =100 , proximity = T, importance
= T)

print(rfb)

##
## Call:
##  randomForest(formula = myFormula, data = random_forest_data_bal,
mtry = sqrt(ncol(random_forest_data_bal) - 1), ntreeb = 100,      proximity =
T, importance = T)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 18.3%
## Confusion matrix:
##               Not Converted Converted class.error
## Not Converted          4141       510   0.1096538
## Converted              1137      3212   0.2614394

##The function ran random forest classification, the no,of variables tried at
each split is 3. The OOB estimate of error rate is 10.1%

#Assigning the importance for each variable
#rf$importance
importance(rfb, type = 1)

##              MeanDecreaseAccuracy
## CustomerCode            52.717082
## CountryName             24.418004
## Handloom                38.271580
## Other                   89.241593
## ShapeName               14.201693
## REC                     15.423730
## Round                    8.248709
## Square                   7.017468
## AreaFt                  97.086648

importance(rfb, type = 2)

##              MeanDecreaseGini
## CustomerCode       609.419427
## CountryName        212.519151
## Handloom            24.559541
## Other              293.867218
## ShapeName            5.194767
## REC                  5.252575
## Round                2.488582
## Square               1.940188
## AreaFt             903.196367
```

```
varImpPlot(rfb)
```




```
rfb$err.rate[ntreeb,1]

##   OOB
## 0.184

The OOB Error rate is 18% for a balanced data, a little higher than the
unbalanced data.

#rf$predicted

# Confusion matrix
Confusion_Matrix_Random_bal <- table(rfb$predicted,
random_forest_data_bal$Order_Conversion, dnn = c("Predicted", "Actual"))
Confusion_Matrix_Random_bal

##                Actual
## Predicted        Not Converted Converted
##   Not Converted           4141      1137
##   Converted                510      3212

library(caret)
#confusionMatrix(rf$predicted, random_forest_data$Order_Conversion, positive
= "Converted")
```

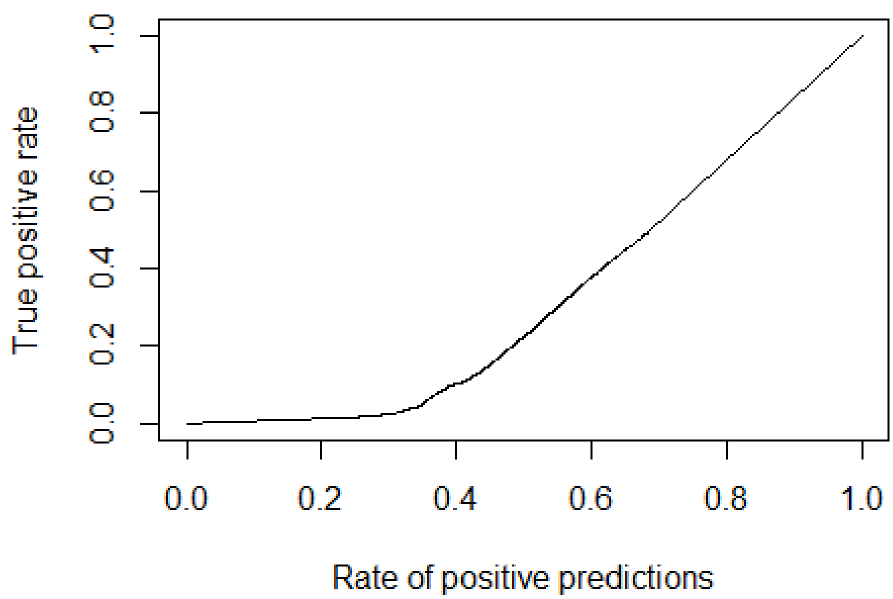
Drawing evaluation charts

```
library(ROCR)
pred1 <- prediction(rfb$votes[, 2],random_forest_data_bal$Order_Conversion)
```

Gain Chart

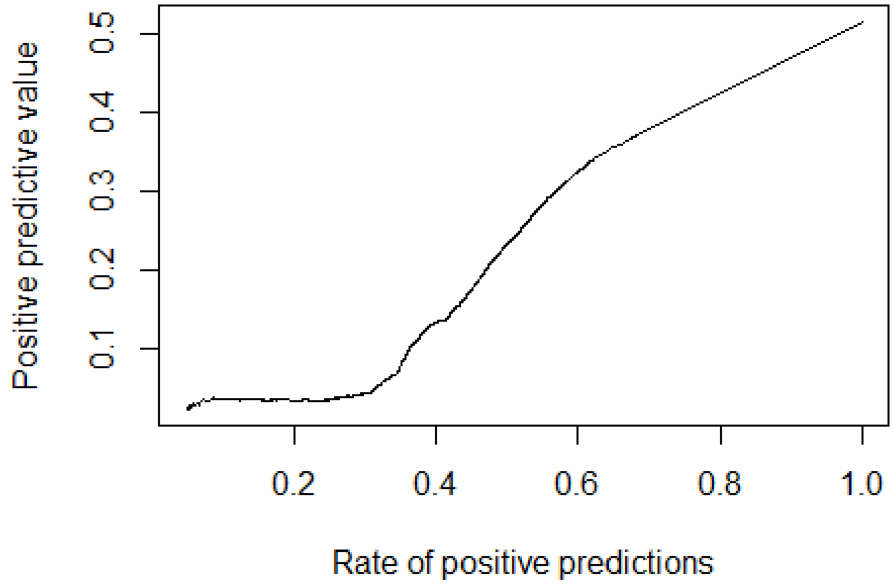
###Gain chart presents the percentage of captured positive responses as a function of selected percentage of a sample. ####Which is actually in our case

```
perf1 <- performance(pred1, "tpr", "rpp")
plot(perf1)
```



Response Chart

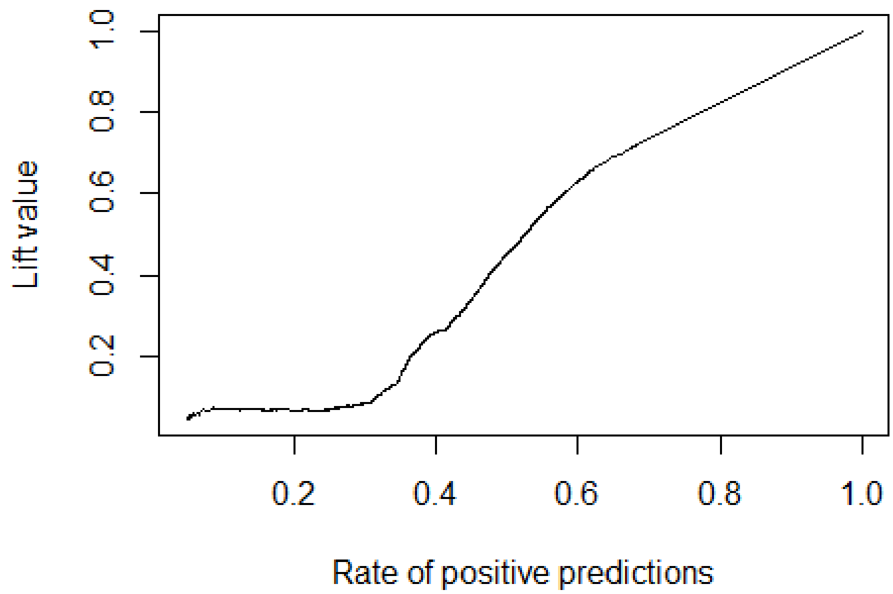
```
perf1 <- performance(pred1, "ppv", "rpp")
plot(perf1)
```



Lift Chart

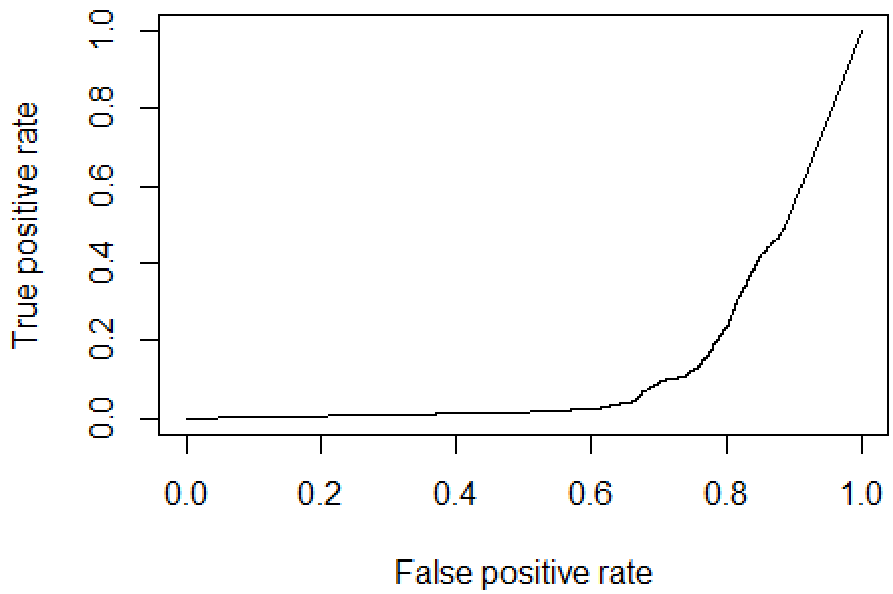
###The lift chart measures effectiveness of our predictive classification model comparing it with the baseline model.

```
perf1 <- performance(pred1, "lift", "rpp")
plot(perf1)
```



ROC Curve - We can conclude that we have a smaller false alarm and also has higher recall,captures more retained(positve)

```
perf1 <- performance(pred1, "tpr", "fpr")
plot(perf1)
```



auc

##Since the AUC is 0.86 and the graph clearly shows the the model is accurate and a good model

```
auc1 <- performance(pred1, "auc")
auc1

## A performance instance
##   'Area under the ROC curve'

auc1 <- unlist(slot(auc1, "y.values"))
auc1

## [1] 0.1426869
```

Q.4) Discuss the data strategy for building customer segmentation using clustering. What are the benefits Champo Carpets can expect from clustering? Hint: Data strategy should clearly identify the data that should be used and how it should be used, including any feature engineering that may be performed before the model building.

The data strategy for customer segmentation and clustering is to group the data based on purchased quantity, amount, total sales, individual item sales. This data helps identify the potential customers within the buyers.

Customer segmentation, by definition, means dividing the customers based on the significant features that helps company improves they target sales and reduces expenses. The champo carpet can benefit from customer segmentation and clustering by identifying the group or cluster of potential customer who would respond to the particular offers provided by the carpet company. Additionally, it can help them identify the customer group which is having high spending on carpets and can channel plans for holiday sales through promotional and marketing offers.

So using unsupervised learning machine learning models like K-Means or Hierarchical clustering we would be able to identify customer segments that have similar purchase patterns and in this way we will be able to target right customers and with right products and focus marketing of products to customers with actual interest of the product.

I our case we have considered K-Means clustering to identify customer groups and combine similar purchase pattern customer segments.

Q.5) Discuss clustering algorithms that can be used for segmenting Champo Carpets's customers. Please justify your choices. Discuss what distance and similarity measures is suitable in this case.

Customer segmentation is often used by companies to divide customer based on common attributes or patterns to market their services/products to the different groups effectively. Customer segmentation can be done through K-Means Clustering. The k-means clustering randomly selects K-data points or centroids, and assign each data point to its nearest centroid, by calculating the ***Euclidian distance*** between all points to all centroids. Calculate the mean of each centroid and shift the centroid in middle of the assigned data points. We have used the elbow method to identify the best cluster which groups similar customer purchase. By also taking the average silhouette measure, we can obtain the number of clusters and determine how similar is the customers purchase pattern.

We can also use Hierarchical Clustering; it generates a plot called Dendrogram which indicates the observations are “grouped together”

K-means clustering

```
cluster_dataset <- readxl::read_excel("Champo Carpets.xlsx", sheet = "Data
for Clustering");
colnames(cluster_dataset)

##  [1] "Row Labels"         "Sum of QtyRequired" "Sum of TotalArea"
##  [4] "Sum of Amount"      "DURRY"              "HANDLOOM"
##  [7] "DOUBLE BACK"        "JACQUARD"           "HAND TUFTED"
## [10] "HAND WOVEN"         "KNOTTED"            "GUN TUFTED"
## [13] "Powerloom Jacquard" "INDO TEBETAN"
```

```
#describe(cluster_dataset)
attach(cluster_dataset)
# head(cluster_dataset, n = 3)
#str(cluster_dataset)
glimpse(cluster_dataset)

## Rows: 45
## Columns: 14
## $ `Row Labels`         <chr> "A-11", "A-6", "A-9", "B-2", "B-3", "B-4",
"C-1",~
## $ `Sum of QtyRequired` <dbl> 2466, 131, 18923, 624, 464, 692, 5137, 55172,
156~
## $ `Sum of TotalArea`   <dbl> 139.5900, 2086.0000, 53625.6544, 202.8987,
8451.5~
## $ `Sum of Amount`      <dbl> 185404.1000, 6247.4600, 1592079.7900,
14811.1591,~
## $ DURRY                <dbl> 1021, 0, 3585, 581, 0, 80, 288, 37042, 1240,
4, 0~
## $ HANDLOOM             <dbl> 1445, 0, 0, 0, 0, 102, 0, 0, 0, 30, 0, 0, 0,
0, 0~
## $ `DOUBLE BACK`        <dbl> 0, 25, 175, 0, 459, 0, 0, 0, 0, 3, 0, 16,
348, 64~
## $ JACQUARD             <dbl> 0, 106, 714, 2, 5, 0, 0, 0, 0, 0, 0, 6, 151,
0, 0~
## $ `HAND TUFTED`        <dbl> 0, 0, 11716, 0, 0, 510, 4176, 3816, 326,
5021, 56~
## $ `HAND WOVEN`         <dbl> 0, 0, 2116, 41, 0, 0, 220, 14314, 0, 0, 0, 0,
51,~
## $ KNOTTED              <dbl> 0, 0, 617, 0, 0, 0, 453, 0, 0, 0, 0, 114, 18,
0, ~
## $ `GUN TUFTED`         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 19, 0, 0, 0, 0, 0,
0, ~
## $ `Powerloom Jacquard` <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0~
## $ `INDO TEBETAN`       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0~

summary(cluster_dataset)

##   Row Labels        Sum of QtyRequired Sum of TotalArea    Sum of Amount
##  Length:45          Min.   :     2     Min.   :     1.35   Min.   :
329
##  Class :character   1st Qu.:   565     1st Qu.:   376.77   1st Qu.:
39701
##  Mode  :character   Median :  1566     Median :  2120.00   Median :
116778
##                     Mean   : 12978     Mean   : 13056.59   Mean   :
698210
##                     3rd Qu.: 11146     3rd Qu.:  8451.56   3rd Qu.:
426626
```

```
##                     Max.   :183206     Max.   :209725.22   Max.   
:11341053  
##      DURRY            HANDLOOM        DOUBLE BACK        JACQUARD     
##  Min.   :     0   Min.   :   0.0   Min.   :   0.0   Min.   :  0.00  
##  1st Qu.:     0   1st Qu.:   0.0   1st Qu.:   0.0   1st Qu.:  0.00  
##  Median :   289   Median :   0.0   Median :   0.0   Median :  0.00  
##  Mean   :  7103   Mean   : 185.5   Mean   : 407.9   Mean   : 89.42  
##  3rd Qu.:  1560   3rd Qu.:   0.0   3rd Qu.: 175.0   3rd Qu.: 72.00  
##  Max.   :139618   Max.   :3673.0   Max.   :5439.0   Max.   :714.00  
##   HAND TUFTED      HAND WOVEN          KNOTTED          GUN TUFTED    
##  Min.   :    0   Min.   :    0.0   Min.   :   0.0   Min.   :  0.000  
##  1st Qu.:    0   1st Qu.:    0.0   1st Qu.:   0.0   1st Qu.:  0.000  
##  Median :  510   Median :    0.0   Median :   0.0   Median :  0.000  
##  Mean   : 3651   Mean   :  867.7   Mean   : 365.8   Mean   :  8.133  
##  3rd Qu.: 3544   3rd Qu.:  269.0   3rd Qu.:  18.0   3rd Qu.:  0.000  
##  Max.   :60685   Max.   :14314.0   Max.   :9502.0   Max.   :195.000  
##  Powerloom Jacquard  INDO TEBETAN   
##  Min.   :   0.0     Min.   : 0.0000  
##  1st Qu.:   0.0     1st Qu.: 0.0000  
##  Median :   0.0     Median : 0.0000  
##  Mean   : 216.7     Mean   : 0.7111  
##  3rd Qu.:   0.0     3rd Qu.: 0.0000  
##  Max.   :9753.0     Max.   :20.0000
```

```r
##Changing the cluster dataset datatypes
cluster_dataset <- data.frame(cluster_dataset)
cluster_dataset$Row.Labels <- as.factor(cluster_dataset$Row.Labels)
# cluster_dataset$Sum.of.QtyRequired <- 
as.numeric(cluster_dataset$Sum.of.QtyRequired)
# cluster_dataset$Sum.of.TotalArea <- 
as.numeric(cluster_dataset$Sum.of.TotalArea)
# cluster_dataset$Sum.of.Amount <- as.numeric(cluster_dataset$Sum.of.Amount)
# cluster_dataset$DURRY <- as.numeric(cluster_dataset$DURRY)
# cluster_dataset$HANDLOOM <- as.numeric(cluster_dataset$HANDLOOM)
# cluster_dataset$DOUBLE.BACK <- as.numeric(cluster_dataset$DOUBLE.BACK)
# cluster_dataset$JACQUARD <- as.numeric(cluster_dataset$JACQUARD)
# cluster_dataset$HAND.TUFTED <- as.numeric(cluster_dataset$HAND.TUFTED)
# cluster_dataset$HAND.WOVEN <- as.numeric(cluster_dataset$HAND.WOVEN)
# cluster_dataset$KNOTTED <- as.numeric(cluster_dataset$KNOTTED)
# cluster_dataset$GUN.TUFTED <- as.numeric(cluster_dataset$GUN.TUFTED)
# cluster_dataset$Powerloom.Jacquard <- 
as.numeric(cluster_dataset$Powerloom.Jacquard)
# cluster_dataset$INDO.TEBETAN <- as.numeric(cluster_dataset$INDO.TEBETAN)

glimpse(cluster_dataset)
```

```
## Rows: 45
## Columns: 14
## $ Row.Labels         <fct> A-11, A-6, A-9, B-2, B-3, B-4, C-1, C-2, C-3, 
CC, C~
```

```
## $ Sum.of.QtyRequired <dbl> 2466, 131, 18923, 624, 464, 692, 5137, 55172,
1566,~
## $ Sum.of.TotalArea   <dbl> 139.5900, 2086.0000, 53625.6544, 202.8987,
8451.562~
## $ Sum.of.Amount      <dbl> 185404.1000, 6247.4600, 1592079.7900,
14811.1591, 5~
## $ DURRY              <dbl> 1021, 0, 3585, 581, 0, 80, 288, 37042, 1240, 4,
0, ~
## $ HANDLOOM           <dbl> 1445, 0, 0, 0, 0, 102, 0, 0, 0, 30, 0, 0, 0, 0,
0, ~
## $ DOUBLE.BACK        <dbl> 0, 25, 175, 0, 459, 0, 0, 0, 0, 3, 0, 16, 348,
64, ~
## $ JACQUARD           <dbl> 0, 106, 714, 2, 5, 0, 0, 0, 0, 0, 0, 6, 151, 0,
0, ~
## $ HAND.TUFTED        <dbl> 0, 0, 11716, 0, 0, 510, 4176, 3816, 326, 5021,
565,~
## $ HAND.WOVEN         <dbl> 0, 0, 2116, 41, 0, 0, 220, 14314, 0, 0, 0, 0,
51, 0~
## $ KNOTTED            <dbl> 0, 0, 617, 0, 0, 0, 453, 0, 0, 0, 0, 114, 18,
0, 0,~
## $ GUN.TUFTED         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 19, 0, 0, 0, 0, 0,
0, 0,~
## $ Powerloom.Jacquard <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, ~
## $ INDO.TEBETAN       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, ~

### Check if any missing values in the dataset
sum(is.na(cluster_dataset[,]))

## [1] 0

###There are no missing values in the dataset

###We use min max transformation to normalize instances:
#We are scaling because we want our data to be weighted to have more a
balanced data so that it doesnot affect the euclidean distance
library(dplyr)
myscale <- function(x) {
(x - min(x)) / (max(x) - min(x))
}
cluster_data <- cluster_dataset %>%
  mutate_if(is.numeric, myscale)

#describe(cluster_data)
attach(cluster_data)

## The following objects are masked from cluster_dataset:
##
##     DURRY, HANDLOOM, JACQUARD, KNOTTED
```

```r
cluster_data <- select(cluster_data, 2:14)

table(is.na(cluster_data))

##
## FALSE
##   585

###k-means
### To create graphs of the clusters generated with the kmeans function
library(factoextra)

## Warning: package 'factoextra' was built under R version 4.1.3

## Welcome! Want to learn more? See two factoextra-related books at
https://goo.gl/ve3WBa

km6 <- kmeans(cluster_data, centers = 6, nstart = 100)
km7 <- kmeans(cluster_data, centers = 7, nstart = 100)
km8 <- kmeans(cluster_data, centers = 8, nstart = 100)
km9 <- kmeans(cluster_data, centers = 9, nstart = 100)

print(km9)

## K-means clustering with 9 clusters of sizes 1, 32, 1, 4, 2, 1, 1, 2, 1
##
## Cluster means:
##   Sum.of.QtyRequired Sum.of.TotalArea Sum.of.Amount       DURRY   HANDLOOM
## 1         1.00000000      0.092998692    0.33546999 1.000000000 1.00000000
## 2         0.02059623      0.023717593    0.01060837 0.018318958 0.01818166
## 3         0.26402808      0.379858715    0.27036982 0.186200920 0.03757147
## 4         0.04962228      0.077753992    0.05907750 0.012460786 0.05363463
## 5         0.04533198      0.007526745    0.04263264 0.036703720 0.07187585
## 6         0.08211065      0.179421548    1.00000000 0.000000000 0.00000000
## 7         0.39784066      0.004378617    0.08368485 0.087402770 0.00000000
## 8         0.20203980      0.029772362    0.12292176 0.151670988 0.00000000
## 9         0.09086592      1.000000000    0.17278132 0.002950909 0.29539886
##   DOUBLE.BACK   JACQUARD HAND.TUFTED  HAND.WOVEN    KNOTTED  GUN.TUFTED
## 1  0.00000000 0.77030812  0.43852682 0.209585022 0.00000000 0.000000000
## 2  0.01255975 0.02976190  0.01337851 0.007645487 0.00294017 0.003044872
## 3  0.86247472 0.49439776  0.03875752 0.373061339 1.00000000 0.000000000
## 4  0.03718514 0.73529412  0.08618687 0.065355596 0.02691539 0.038461538
## 5  0.15223387 0.03501401  0.02171871 0.033289088 0.02041675 0.312820513
## 6  0.00000000 0.00000000  0.24791958 0.000000000 0.00000000 0.000000000
## 7  0.00000000 0.00000000  1.00000000 0.000000000 0.00000000 0.000000000
## 8  0.32864497 0.16176471  0.06064102 0.678496577 0.05398863 0.000000000
## 9  1.00000000 0.08403361  0.04444261 0.215523264 0.38160387 1.000000000
##   Powerloom.Jacquard INDO.TEBETAN
## 1                  1          0.0
## 2                  0          0.0
## 3                  0          0.0
```

```
## 4                    0          0.0
## 5                    0          0.8
## 6                    0          0.0
## 7                    0          0.0
## 8                    0          0.0
## 9                    0          0.0
##
## Clustering vector:
##  [1] 2 2 4 2 2 2 2 8 2 2 2 2 2 2 2 2 2 2 1 4 8 2 2 2 2 2 2 9 4 7 2 3 2 5 2
4 2 2
## [39] 5 2 2 2 2 6 2
##
## Within cluster sum of squares by cluster:
## [1] 0.0000000 0.6520855 0.0000000 0.2361929 0.2978700 0.0000000 0.0000000
## [8] 0.5272849 0.0000000
##  (between_SS / total_SS =  90.3 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
"tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
km6.clusters <- km6$cluster
km7.clusters <- km7$cluster
km8.clusters <- km8$cluster
km9.clusters <- km9$cluster
rownames(cluster_data) <-
paste(cluster_dataset$Row.Labels,1:dim(cluster_dataset)[1], sep = "-")
p6 <- fviz_cluster(list(data = cluster_data, cluster = km6.clusters))
p7 <- fviz_cluster(list(data = cluster_data, cluster = km7.clusters))
p8 <- fviz_cluster(list(data = cluster_data, cluster = km8.clusters))
p9 <- fviz_cluster(list(data = cluster_data, cluster = km9.clusters))
```

```
km8$cluster
```

```
##  [1] 1 1 3 1 1 1 1 5 1 1 1 1 1 1 1 1 1 1 6 3 5 1 1 1 1 1 1 4 3 2 1 7 1 8 1
3 1 1
## [39] 8 1 1 1 1 2 1
```

```
km8$centers
```

```
##   Sum.of.QtyRequired Sum.of.TotalArea Sum.of.Amount       DURRY   HANDLOOM
## 1         0.02059623      0.023717593    0.01060837 0.018318958 0.01818166
## 2         0.23997566      0.091900083    0.54184243 0.043701385 0.00000000
## 3         0.04962228      0.077753992    0.05907750 0.012460786 0.05363463
## 4         0.09086592      1.000000000    0.17278132 0.002950909 0.29539886
## 5         0.20203980      0.029772362    0.12292176 0.151670988 0.00000000
## 6         1.00000000      0.092998692    0.33546999 1.000000000 1.00000000
## 7         0.26402808      0.379858715    0.27036982 0.186200920 0.03757147
## 8         0.04533198      0.007526745    0.04263264 0.036703720 0.07187585
```

```
##   DOUBLE.BACK   JACQUARD HAND.TUFTED  HAND.WOVEN    KNOTTED  GUN.TUFTED
## 1  0.01255975 0.02976190  0.01337851 0.007645487 0.00294017 0.003044872
## 2  0.00000000 0.00000000  0.62395979 0.000000000 0.00000000 0.000000000
## 3  0.03718514 0.73529412  0.08618687 0.065355596 0.02691539 0.038461538
## 4  1.00000000 0.08403361  0.04444261 0.215523264 0.38160387 1.000000000
## 5  0.32864497 0.16176471  0.06064102 0.678496577 0.05398863 0.000000000
## 6  0.00000000 0.77030812  0.43852682 0.209585022 0.00000000 0.000000000
## 7  0.86247472 0.49439776  0.03875752 0.373061339 1.00000000 0.000000000
## 8  0.15223387 0.03501401  0.02171871 0.033289088 0.02041675 0.312820513
##   Powerloom.Jacquard INDO.TEBETAN
## 1                  0          0.0
## 2                  0          0.0
## 3                  0          0.0
## 4                  0          0.0
## 5                  0          0.0
## 6                  1          0.0
## 7                  0          0.0
## 8                  0          0.8
```

```
km8$withinss
```

```
## [1] 0.6520855 0.7716116 0.2361929 0.0000000 0.5272849 0.0000000 0.0000000
## [8] 0.2978700
```

```
km8$betweenss
```

```
## [1] 15.21135
```

```
km8$size
```

```
## [1] 32  2  4  1  2  1  1  2
```

```
##Determines and visualizes the optimal number of clusters
# plots to compare
# p1 <- fviz_cluster(km6, geom = "point", data = cluster_data) + ggtitle("k = 
2")
# p2 <- fviz_cluster(km7, geom = "point", data = cluster_data) + ggtitle("k = 
3")
# p3 <- fviz_cluster(km8, geom = "point", data = cluster_data) + ggtitle("k = 
4")
# p4 <- fviz_cluster(km9, geom = "point", data = cluster_data) + ggtitle("k = 
5")

library(gridExtra)
```

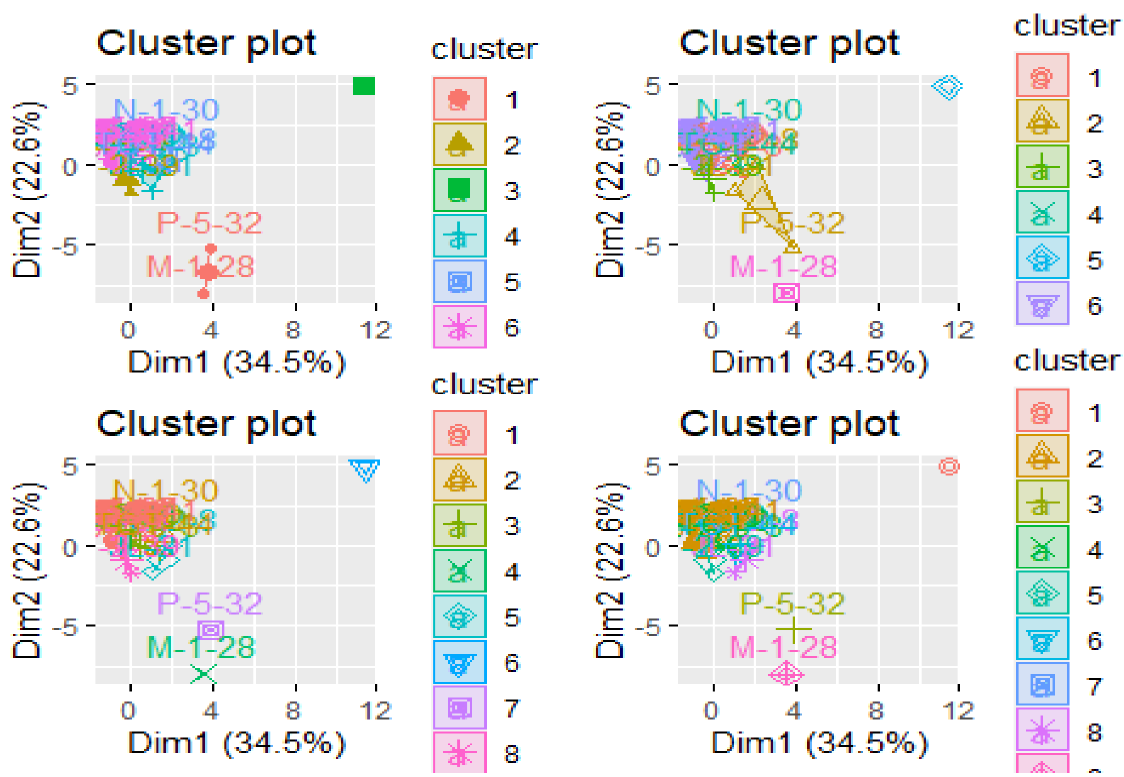
```
## 
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:randomForest':
## 
##     combine
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

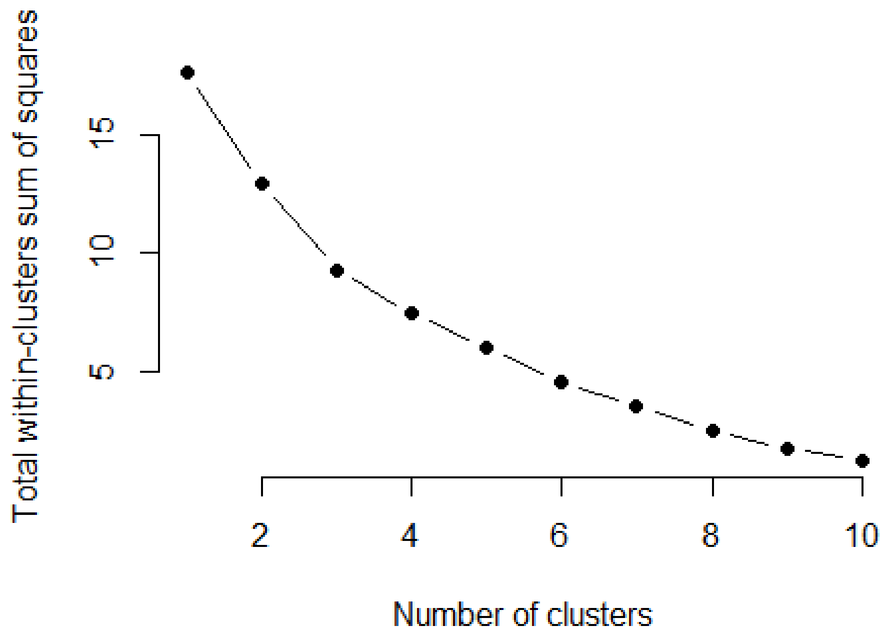
```
grid.arrange(p6, p7, p8, p9, nrow = 2)
```

From the above components of clusters the clusters (k) = 9, we see that we tried with different k values has <u>(between_SS / total_SS = 90.3 %)</u> which is higher than the other k values signifying its is the optimal number of clusters

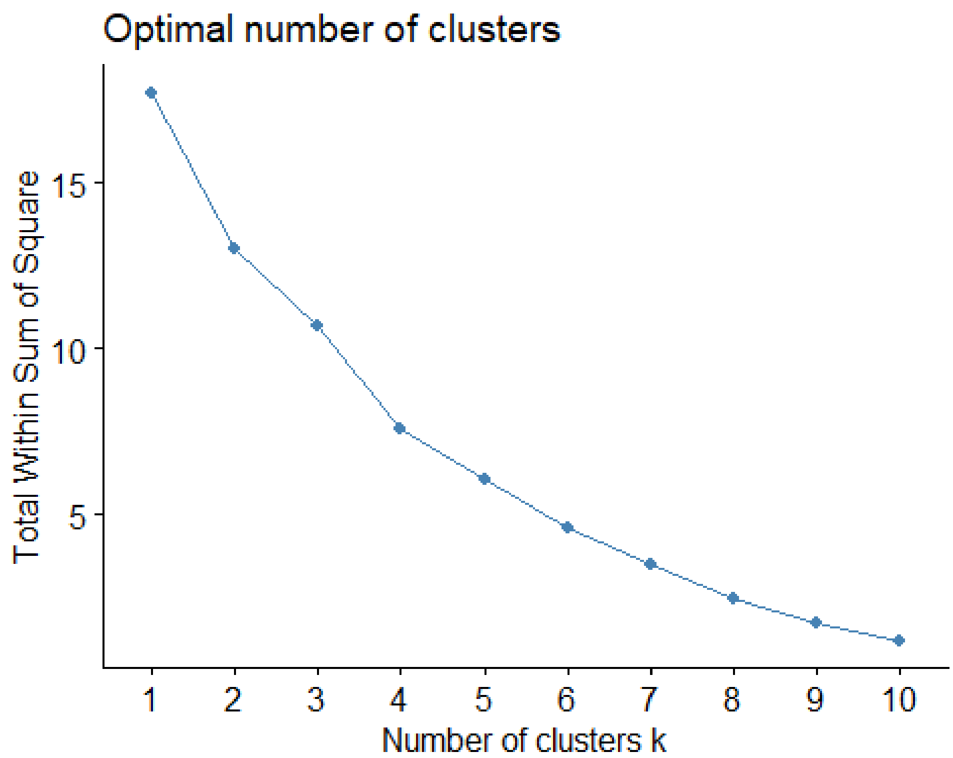



```
##To determine the number of clusters we use elbow method
set.seed(123)
# function to compute total within-cluster sum of square

wss <- function(k) {
  kmeans(cluster_data, centers = k, nstart = 100)$tot.withinss
}
# Compute and plot wss for k = 1 to k = 15
k.values <- 1:10

# extract wss for 2-15 clusters
library(tidyverse)
wss_values <- map_dbl(k.values, wss)
plot(k.values, wss_values, type="b", pch = 19, frame = FALSE, xlab="Number of
clusters", ylab="Total within-clusters sum of squares")
```

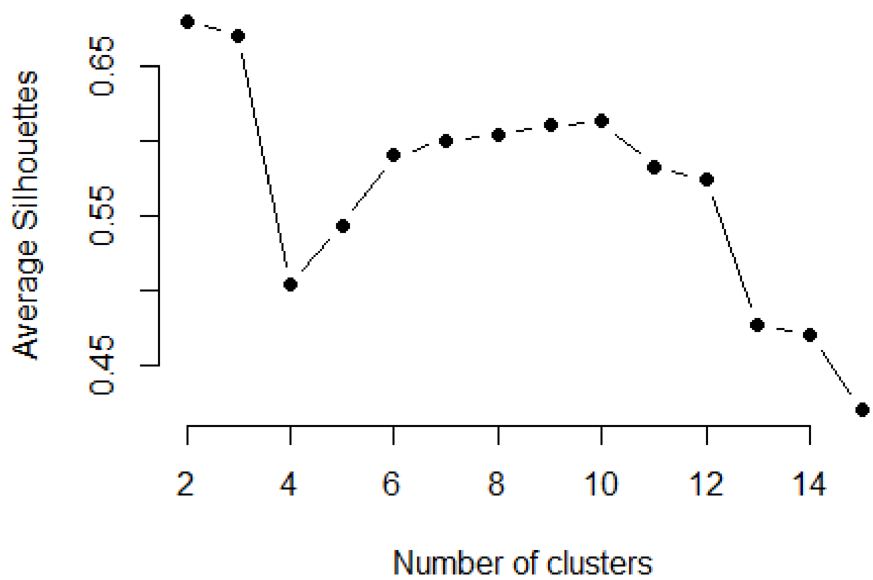
```
We can see from the Elbow Method, the optimal number of clusters range
between 5-10
```



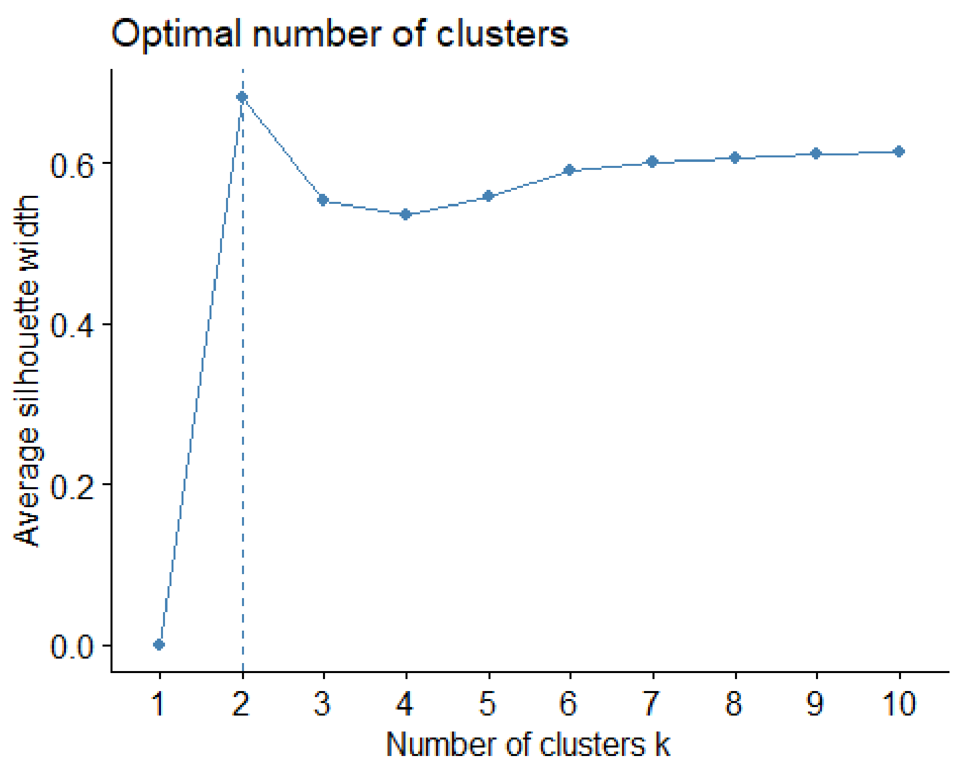

```
##To get the screen plot, we can also use the "fviz_nbclust" function.
##The graph shows the number of clusters that are optimal and it is between
5-10
set.seed(123)
fviz_nbclust(cluster_data, kmeans, method = "wss")
```



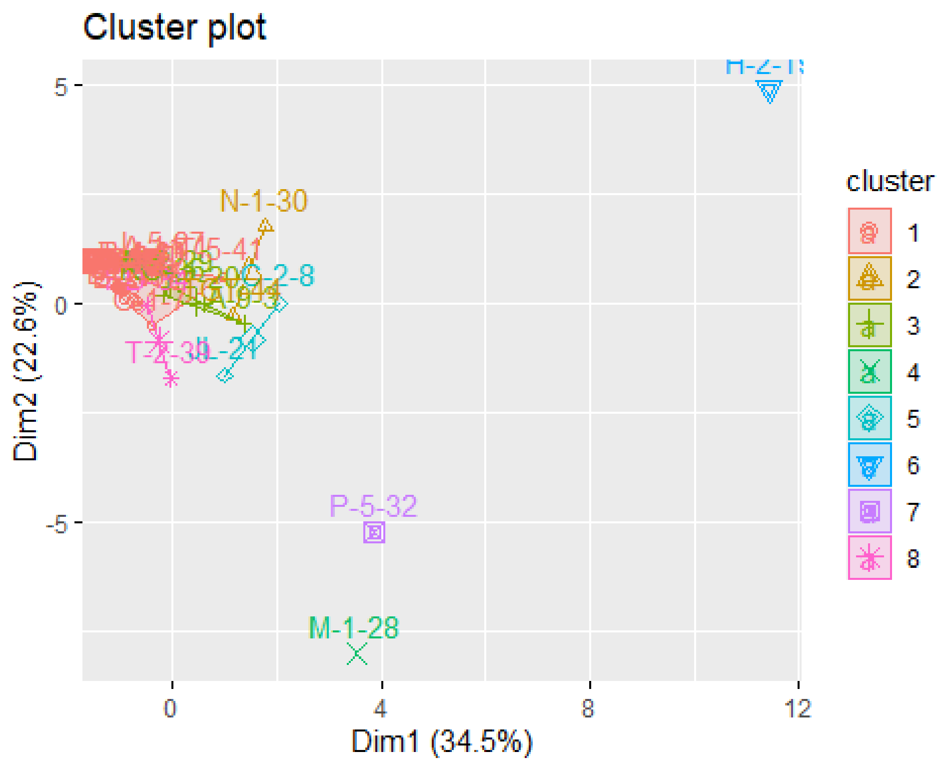

```r
# function to compute average silhouette for k clusters
library(cluster)
avgsil <- function(k) {
kmModel <- kmeans(cluster_data, centers = k, nstart = 100)
ss <- silhouette(kmModel$cluster, dist(cluster_data))
mean(ss[, 3])
}
# Compute and plot wss for k = 2 to k = 10
k.values <- 2:15
# extract avg silhouette for 2-15 clusters
avgsil_values <- map_dbl(k.values, avgsil)
plot(k.values, avgsil_values, type = "b", pch = 19, frame = FALSE, xlab =
"Number of clusters", ylab = "Average Silhouettes")
```




```
##Similar to the elbow method, the "average silhoutete method" can be found
in fviz_nbclust function.
fviz_nbclust(cluster_data, kmeans, method = "silhouette")
```



```
#Considering the number of clusters as 8
fviz_cluster(km8, data = cluster_data)
```




```
###After we finalize our kmeans model, we can extract the clusters and do
some descriptive analysis at each cluster. For example:
cluster_data %>%
mutate(Cluster = km8$cluster) %>%
group_by(Cluster) %>%
summarise_all("mean")

## # A tibble: 8 x 14
##   Cluster Sum.of.QtyRequired Sum.of.TotalArea Sum.of.Amount   DURRY
HANDLOOM
##     <int>              <dbl>            <dbl>          <dbl>   <dbl>
<dbl>
## 1       1             0.0206           0.0237         0.0106 0.0183
0.0182
## 2       2             0.240            0.0919         0.542  0.0437     0
## 3       3             0.0496           0.0778         0.0591 0.0125
0.0536
## 4       4             0.0909           1              0.173  0.00295
0.295
## 5       5             0.202            0.0298         0.123  0.152      0
## 6       6             1                0.0930         0.335  1          1
## 7       7             0.264            0.380          0.270  0.186
0.0376
## 8       8             0.0453           0.00753        0.0426 0.0367
```

```
0.0719
## # ... 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>
```

###Neural network model using nnet for unbalanced data####

```
sample_dataset_nn <- sample_random[sample(nrow(sample_random)),]
colnames(sample_dataset_nn)
```

```
##  [1] "CustomerCode"     "CountryName"      "QtyRequired"      "ITEM_NAME"
##  [5] "Hand_Tufted"      "Durry"            "Double_Back"      "Hand_Woven"
##  [9] "Knotted"          "Jacquard"         "Handloom"         "Other"
## [13] "ShapeName"        "REC"              "Round"            "Square"
## [17] "AreaFt"           "Order_Conversion"
```

```
##To check NA values
table(is.na(sample_dataset_nn))
```

```
##
##  FALSE
## 104760
```

```
lapply(sample_dataset_nn, function(x) { length(which(is.na(x)))})
```

```
## $CustomerCode
## [1] 0
##
## $CountryName
## [1] 0
##
## $QtyRequired
## [1] 0
##
## $ITEM_NAME
## [1] 0
##
## $Hand_Tufted
## [1] 0
##
## $Durry
## [1] 0
##
## $Double_Back
## [1] 0
##
## $Hand_Woven
## [1] 0
##
## $Knotted
## [1] 0
```

```
##
## $Jacquard
## [1] 0
##
## $Handloom
## [1] 0
##
## $Other
## [1] 0
##
## $ShapeName
## [1] 0
##
## $REC
## [1] 0
##
## $Round
## [1] 0
##
## $Square
## [1] 0
##
## $AreaFt
## [1] 0
##
## $Order_Conversion
## [1] 0
```

```
#There are few variables that needs their datatypes to be changed
# categorical_variablesSD <- c(1,2,6,7,8,9)
# sample_dataset[categorical_variablesSD] <-
lapply(sample_dataset[categorical_variablesSD], as.factor)
attach(sample_dataset_nn)
```

```
## The following objects are masked from sample_balanced_nn (pos = 11):
##
##     AreaFt, CountryName, CustomerCode, Double_Back, Durry, Hand_Tufted,
##     Hand_Woven, Handloom, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, QtyRequired, REC, Round, ShapeName, Square
```

```
## The following objects are masked from sample_balanced_nn (pos = 12):
##
##     AreaFt, CountryName, CustomerCode, Double_Back, Durry, Hand_Tufted,
##     Hand_Woven, Handloom, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, QtyRequired, REC, Round, ShapeName, Square
```

```
## The following objects are masked from balanced_sample_dataset:
##
##     AreaFt, CountryName, CustomerCode, Double_Back, Durry, Hand_Tufted,
##     Hand_Woven, Handloom, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, QtyRequired, REC, Round, ShapeName, Square
```

```
## The following objects are masked from sample_only_dataset_CV (pos = 14):
##
##     AreaFt, CountryName, CustomerCode, Double_Back, Durry, Hand_Tufted,
##     Hand_Woven, Handloom, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, QtyRequired, REC, Round, ShapeName, Square

## The following objects are masked from sample_only_dataset_CV (pos = 15):
##
##     AreaFt, CountryName, CustomerCode, Double_Back, Durry, Hand_Tufted,
##     Hand_Woven, Handloom, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, QtyRequired, REC, Round, ShapeName, Square

## The following objects are masked from sample_only_dataset (pos = 16):
##
##     AreaFt, CountryName, CustomerCode, Double_Back, Durry, Hand_Tufted,
##     Hand_Woven, Handloom, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, QtyRequired, REC, Round, ShapeName, Square

## The following objects are masked from sample_only_dataset (pos = 18):
##
##     AreaFt, CountryName, CustomerCode, Double_Back, Durry, Hand_Tufted,
##     Hand_Woven, Handloom, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, QtyRequired, REC, Round, ShapeName, Square

## The following objects are masked from Sample_onlyData_LM:
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, Order_Conversion,
##     QtyRequired, ShapeName

## The following objects are masked from dataset (pos = 20):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName

## The following objects are masked from dataset (pos = 22):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName

sample_dataset_nn$CustomerCode<-
as.numeric(factor(as.matrix(sample_dataset_nn$CustomerCode)))
sample_dataset_nn$CountryName<-
as.numeric(factor(as.matrix(sample_dataset_nn$CountryName)))
sample_dataset_nn$ITEM_NAME<-
as.numeric(factor(as.matrix(sample_dataset_nn$ITEM_NAME)))
sample_dataset_nn$ShapeName<-
as.numeric(factor(as.matrix(sample_dataset_nn$ShapeName)))

nrow(sample_dataset_nn)

## [1] 5820
```

```
head(sample_dataset_nn)

## # A tibble: 6 x 18
##   CustomerCode CountryName QtyRequired ITEM_NAME Hand_Tufted Durry
Double_Back
##          <dbl>       <dbl>       <dbl>     <dbl>       <dbl> <dbl>
<dbl>
## 1            7           6           1         9           0     0
0
## 2           23          14           3         4           1     0
0
## 3            7           6           1         2           0     1
0
## 4           23          14           1         4           1     0
0
## 5            7           6           1         1           0     0
1
## 6            7           6           1         4           1     0
0
## # ... with 11 more variables: Hand_Woven <dbl>, Knotted <dbl>, Jacquard
<dbl>,
## #   Handloom <dbl>, Other <dbl>, ShapeName <dbl>, REC <dbl>, Round <dbl>,
## #   Square <dbl>, AreaFt <dbl>, Order_Conversion <fct>

##Normalize data before training a neural network###
###myscale() function uses min-max transformation to normalize variable x
myscale <- function(x)
{
  (x - min(x)) / (max(x) - min(x))
}

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

###Splitting the normalized data into train and test set
set.seed(1234)
indx <- sample(2, nrow(sample_dataset_nn), replace = T, prob = c(0.7,0.3))
train <- sample_dataset_nn[indx == 1,]
test <- sample_dataset_nn[indx == 2,]


###Using nnet function to build neural network model
attach(sample_dataset_nn)

## The following objects are masked from sample_dataset_nn (pos = 3):
##
##     AreaFt, CountryName, CustomerCode, Double_Back, Durry, Hand_Tufted,
##     Hand_Woven, Handloom, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, QtyRequired, REC, Round, ShapeName, Square
```

```
## The following objects are masked from sample_balanced_nn (pos = 12):
##
##     AreaFt, CountryName, CustomerCode, Double_Back, Durry, Hand_Tufted,
##     Hand_Woven, Handloom, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, QtyRequired, REC, Round, ShapeName, Square

## The following objects are masked from sample_balanced_nn (pos = 13):
##
##     AreaFt, CountryName, CustomerCode, Double_Back, Durry, Hand_Tufted,
##     Hand_Woven, Handloom, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, QtyRequired, REC, Round, ShapeName, Square

## The following objects are masked from balanced_sample_dataset:
##
##     AreaFt, CountryName, CustomerCode, Double_Back, Durry, Hand_Tufted,
##     Hand_Woven, Handloom, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, QtyRequired, REC, Round, ShapeName, Square

## The following objects are masked from sample_only_dataset_CV (pos = 15):
##
##     AreaFt, CountryName, CustomerCode, Double_Back, Durry, Hand_Tufted,
##     Hand_Woven, Handloom, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, QtyRequired, REC, Round, ShapeName, Square

## The following objects are masked from sample_only_dataset_CV (pos = 16):
##
##     AreaFt, CountryName, CustomerCode, Double_Back, Durry, Hand_Tufted,
##     Hand_Woven, Handloom, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, QtyRequired, REC, Round, ShapeName, Square

## The following objects are masked from sample_only_dataset (pos = 17):
##
##     AreaFt, CountryName, CustomerCode, Double_Back, Durry, Hand_Tufted,
##     Hand_Woven, Handloom, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, QtyRequired, REC, Round, ShapeName, Square

## The following objects are masked from sample_only_dataset (pos = 19):
##
##     AreaFt, CountryName, CustomerCode, Double_Back, Durry, Hand_Tufted,
##     Hand_Woven, Handloom, ITEM_NAME, Jacquard, Knotted,
##     Order_Conversion, Other, QtyRequired, REC, Round, ShapeName, Square

## The following objects are masked from Sample_onlyData_LM:
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, Order_Conversion,
##     QtyRequired, ShapeName

## The following objects are masked from dataset (pos = 21):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName
```

```
## The following objects are masked from dataset (pos = 23):
##
##     AreaFt, CountryName, CustomerCode, ITEM_NAME, QtyRequired,
##     ShapeName
```

```r
library(nnet)
nnModel <- nnet(Order_Conversion ~., data = train, linout = FALSE, size = 10,
hidden =3, decay = 0.01, maxit = 1000)
```

```
## # weights:  191
## initial  value 4123.941136
## iter  10 value 1721.064396
## iter  20 value 1389.305585
## iter  30 value 1240.778416
## iter  40 value 1184.805834
## iter  50 value 1159.949555
## iter  60 value 1114.814105
## iter  70 value 1080.165819
## iter  80 value 1061.111137
## iter  90 value 1047.028769
## iter 100 value 1034.275600
## iter 110 value 1022.872139
## iter 120 value 1016.006889
## iter 130 value 1010.886271
## iter 140 value 1005.485936
## iter 150 value 1000.715772
## iter 160 value 997.158820
## iter 170 value 994.672243
## iter 180 value 993.199176
## iter 190 value 991.779869
## iter 200 value 990.166825
## iter 210 value 988.982646
## iter 220 value 988.090268
## iter 230 value 987.016808
## iter 240 value 986.302377
## iter 250 value 985.612222
## iter 260 value 985.077001
## iter 270 value 984.465896
## iter 280 value 983.884284
## iter 290 value 982.901638
## iter 300 value 981.052378
## iter 310 value 979.593928
## iter 320 value 978.768442
## iter 330 value 978.224310
## iter 340 value 977.660760
## iter 350 value 977.049898
## iter 360 value 976.361021
## iter 370 value 975.796779
## iter 380 value 975.335805
## iter 390 value 974.510273
```

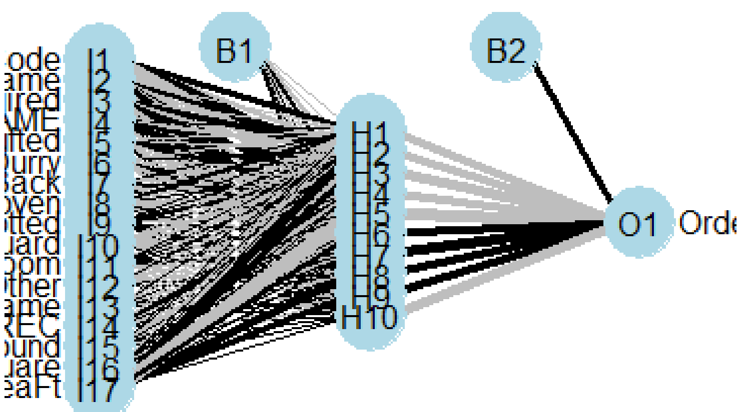
```
## iter 400 value 973.911819
## iter 410 value 973.418786
## iter 420 value 972.606519
## iter 430 value 971.632617
## iter 440 value 970.106067
## iter 450 value 968.451169
## iter 460 value 966.485211
## iter 470 value 965.368052
## iter 480 value 964.785365
## iter 490 value 964.505484
## iter 500 value 964.299641
## iter 510 value 964.049171
## iter 520 value 963.777046
## iter 530 value 963.249162
## iter 540 value 962.630370
## iter 550 value 962.146365
## iter 560 value 961.797228
## iter 570 value 961.477090
## iter 580 value 961.330730
## iter 590 value 961.176391
## iter 600 value 961.093592
## iter 610 value 960.984982
## iter 620 value 960.873817
## iter 630 value 960.810831
## iter 640 value 960.746997
## iter 650 value 960.688680
## iter 660 value 960.651681
## iter 670 value 960.631321
## iter 680 value 960.619668
## iter 690 value 960.612155
## iter 700 value 960.608551
## iter 710 value 960.604128
## iter 720 value 960.599499
## iter 730 value 960.597594
## iter 740 value 960.597063
## iter 750 value 960.596354
## iter 760 value 960.595735
## iter 770 value 960.595431
## final  value 960.595237
## converged
```

```
summary(nnModel)
```

```
## a 17-10-1 network with 191 weights
## options were - entropy fitting  decay=0.01
##   b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1  i8->h1
i9->h1
##   -2.12    8.62   -3.51   -1.20    0.31   -4.07   -0.25    0.25    4.32
3.78
## i10->h1 i11->h1 i12->h1 i13->h1 i14->h1 i15->h1 i16->h1 i17->h1
```

```
##   -3.26   -1.23   -1.65   -0.71   -2.47    2.13   -1.78   -2.31
##   b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2
i9->h2
##   -0.47   -6.55    5.74    8.82   -4.63   -0.87    0.33   -0.41   -0.44
0.83
## i10->h2 i11->h2 i12->h2 i13->h2 i14->h2 i15->h2 i16->h2 i17->h2
##   -3.87    2.28    1.68    1.25   -1.74    0.03    1.24   13.09
##   b->h3  i1->h3  i2->h3  i3->h3  i4->h3  i5->h3  i6->h3  i7->h3  i8->h3
i9->h3
##   -4.36   21.95   -3.84   -5.02    5.33    4.17   -3.13   -3.80    0.69
5.06
## i10->h3 i11->h3 i12->h3 i13->h3 i14->h3 i15->h3 i16->h3 i17->h3
##   -6.79    2.23   -2.79   -1.64   -0.79   -3.86    0.29    9.86
##   b->h4  i1->h4  i2->h4  i3->h4  i4->h4  i5->h4  i6->h4  i7->h4  i8->h4
i9->h4
##    5.73  -18.34   -7.33    6.35    1.44    5.65   -1.91    2.67   -6.56
2.89
## i10->h4 i11->h4 i12->h4 i13->h4 i14->h4 i15->h4 i16->h4 i17->h4
##   -0.24    3.37   -0.14   -1.98   10.10   -4.77    0.41    0.74
##   b->h5  i1->h5  i2->h5  i3->h5  i4->h5  i5->h5  i6->h5  i7->h5  i8->h5
i9->h5
##   -2.09   -6.12    6.74    3.86    4.37   -1.36    1.24    2.15   -1.71
-0.52
## i10->h5 i11->h5 i12->h5 i13->h5 i14->h5 i15->h5 i16->h5 i17->h5
##   -1.49    0.18   -0.57    0.78   -2.95    0.15    0.71  -24.40
##   b->h6  i1->h6  i2->h6  i3->h6  i4->h6  i5->h6  i6->h6  i7->h6  i8->h6
i9->h6
##    2.05   -0.49   -9.16   14.19    0.29   -0.31    5.89   -1.60   -2.49
3.51
## i10->h6 i11->h6 i12->h6 i13->h6 i14->h6 i15->h6 i16->h6 i17->h6
##   -3.14    2.99   -2.82   -0.86    3.72   -1.61   -0.06    4.47
##   b->h7  i1->h7  i2->h7  i3->h7  i4->h7  i5->h7  i6->h7  i7->h7  i8->h7
i9->h7
##   -0.96   -7.97   10.05   -4.64    0.55   -1.02   -5.51   -1.19    2.81
1.75
## i10->h7 i11->h7 i12->h7 i13->h7 i14->h7 i15->h7 i16->h7 i17->h7
##   -4.41    6.85   -0.22    2.02   -3.17    0.37    1.84    2.41
##   b->h8  i1->h8  i2->h8  i3->h8  i4->h8  i5->h8  i6->h8  i7->h8  i8->h8
i9->h8
##    2.25   -0.74   -7.71    0.27    2.00   -0.36    0.23   -1.78    1.87
4.37
## i10->h8 i11->h8 i12->h8 i13->h8 i14->h8 i15->h8 i16->h8 i17->h8
##   -1.74   -0.62    0.28    2.03   -1.24    2.93    0.56   14.19
##   b->h9  i1->h9  i2->h9  i3->h9  i4->h9  i5->h9  i6->h9  i7->h9  i8->h9
i9->h9
##   -2.00   -8.32    4.07   -4.30   -2.13    3.48   -2.12    3.66   -6.34
1.81
## i10->h9 i11->h9 i12->h9 i13->h9 i14->h9 i15->h9 i16->h9 i17->h9
##   -2.28   -2.76    2.55   -1.71    1.35   -3.28   -0.06   11.23
##   b->h10  i1->h10  i2->h10  i3->h10  i4->h10  i5->h10  i6->h10  i7->h10
```

```
##     4.58   -11.99    -6.55   -17.08    -4.80     0.28    -1.03     1.16
##  i8->h10  i9->h10 i10->h10 i11->h10 i12->h10 i13->h10 i14->h10 i15->h10
##     1.26     2.44     0.23    -0.02     0.25     1.35     2.01     2.42
## i16->h10 i17->h10
##     0.14     3.61
##   b->o  h1->o  h2->o  h3->o  h4->o  h5->o  h6->o  h7->o  h8->o  h9->o h10-
>o
##  10.87 -12.83 -12.52 -17.03 -15.69 -22.54  14.99  18.08  14.33  15.82 -
19.86

##You are using wts to get the best weights found and fitted.values to get
the fitted values on training data
# nnModel$wts
# nnModel$fitted.values

##To draw nnet model
library(NeuralNetTools)
plotnet(nnModel)
```




```
##Neural network model used to predict test instances
nn.preds = predict(nnModel, test)

##Notice we still have results between 0 and 1 that are more like
probabilities of belonging to each class. To get the predicted classes we can
use change the type argument.

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

```
##Confusion Matrix
ConfMatrix <- table(nn.preds, test$Order_Conversion, dnn =
c("predicted","actual"))
print(ConfMatrix)
```

```
##          actual
## predicted    0    1
##         0 1318  110
##         1   34  252
```

```
##Check performance of neural network model
error_metric = function(ConfMatrix)
{
TN = ConfMatrix[1,1]
TP = ConfMatrix[2,2]
FN = ConfMatrix[1,2]
FP = ConfMatrix[2,1]
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("precision" = precision,
"recall" = recall,
"falsepositiverate" = falsePositiveRate,
"falsenegativerate" = falseNegativeRate,
"error" = error)
return(modelPerf)
}
outPutlist <- error_metric(ConfMatrix)
library(plyr)
df <- ldply(outPutlist, data.frame)
setNames(df, c("", "Values"))
```

```
##                        Values
## 1         precision 0.88111888
## 2            recall 0.69613260
## 3 falsepositiverate 0.02514793
## 4 falsenegativerate 0.30386740
## 5             error 0.08401400
```

5. Recommendation Dataset

```
# Check the dataset
recom_df <- readxl::read_excel("Champo Carpets.xlsx", sheet = "Data for
Recommendation");
```

```
## New names:
## * Customer -> Customer...1
## * `` -> ...22
```

```
## * `` -> ...23
## * Customer -> Customer...24
```

```
#view(recom_df)

# The data shows column and its transpose included together. Let us split the columns and there transpose into separate dataframe.

recom_df1<-recom_df[,1:21]
recom_df2<-recom_df[,24:44]

# Rename the first column as Customer
names(recom_df1)[1]<-"Customer"
names(recom_df2)[1]<-"Customer"

# Check Missing
sum(is.na(recom_df1))
```

```
## [1] 0
```

```
sum(is.na(recom_df2))
```

```
## [1] 0
```

```
# Display names
names(recom_df1)
```

```
##  [1] "Customer"     "Hand Tufted"  "Double Wowen" "Durry"        "Double Back"
##  [6] "Knotted"      "Jacquared"    "Handloom"     "Other"        "Rectangle"
## [11] "Square"       "Round"        "Purple"       "Gray"         "Navy"
## [16] "PINK"         "BLUE"         "BLUSH PINK"   "NEUTRAL"      "TAN"
## [21] "NAVY"
```

```
# Display dimension
dim(recom_df1)
```

```
## [1] 20 21
```

```
# Display Structure
#str(recom_df1)
```

User-Based Collaborative Filtering: Correlation Based Similarity

Calculate Correlation between Customer Features

```
# Calculate Correlation of Each pair of Feature

a<-corrr::correlate(recom_df2[2:ncol(recom_df2)], method = "pearson",use="pairwise.complete.obs")
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```

```r
# Replace NA's to perfect correlation value i.e., 1
a[is.na(a)]<-1

# For each row find the Second best correlation as the best is 1 assigned for
Na's.
for(i in 2:ncol(a)) {
  # Select Column of CustomerCodes
  x=a[,i]
  # Find Second Best Correlation
  y=max(x[x != max(x)])
  #Store the Nearest neighbor in the dataframe
  recom_df1[i-1,'Nearest Neighbor']=a[which(x == y),1]
}
```

Define Recommendation System Based on Nearest Neighbor

```r
getRecommendation=function(CustomerCode){
  RowIndexCustomer=which(recom_df1$Customer == CustomerCode)
  CustomerOrder=recom_df1[RowIndexCustomer,]
  NeighborOrder=recom_df1[recom_df1$Customer %in%
recom_df1[RowIndexCustomer,ncol(recom_df1)],]
  Item=list()
  Shape=list()
  Color=list()

  for(i in 2:ncol(recom_df1)-1) {

    x=CustomerOrder[1,i]
    y=NeighborOrder[1,i]

    if(x==0 && y>0)
      if(i<10)
       Item[[length(Item)+1]] = names(CustomerOrder)[i]
      else if(i<13){
        Shape[[length(Shape)+1]] = names(CustomerOrder)[i]
      }
    else{
      Color[[length(Color)+1]] = names(CustomerOrder)[i]
    }}
  if(length(Item)>0){
    print(sprintf("Recommended Item: %s",paste(Item, collapse = ", ") ))
    }
  if(length(Shape)>0){
    print(sprintf("Recommended Shape: %s",paste(Shape, collapse = ", ")))
  }

  if(length(Color)>0){
```

```
    print(sprintf("Recommended Color: %s",paste(Color, collapse = ", ")))
  }
}
```

Display Recommendation for Customers

```
# For Customer Code H-2
getRecommendation('H-2')

## [1] "Recommended Item: Double Back"
## [1] "Recommended Color: Gray, PINK, BLUSH PINK"

# For Customer Code T-5
getRecommendation('T-5')

## [1] "Recommended Item: Knotted, Other"
## [1] "Recommended Shape: Square"
```

Recommendation to Champo Carpet.

Considering clusters (k) = 4

The Champo Carpet do face the issue of low conversion rate. Using the cluster analysis results, Cluster 4 represents a group of customers with high Quantity Requirements.

The Cluster 4 group requires mostly Durry with significantly higher Quantity and higher amount of Handloom, Double Back, Hand tuffed, and knotted items. It is only Customer Group that orders PowerLoom Jackguard.The Company can target the group for a combining offers of Durry and other mentioned items, to enhances their purchasing experience.

Cluster 3 is the customer group of large size items. The total area of the carpet order is highest for the group. The cluster more frequently orders large size durry, double back, and handwoven items.

Cluster 2 is unique group of customers that only purchases Handtuffed Items of moderate and large sizes. The total amount paid by the customer group is highest comparative to the other groups. The Cluster 2 appears to be rich class of customers with likeness of Hand Tuffed Product. The Cluster can be give discounted offers for other items in combination to the Hand Tuffed Products.

Cluster 1 is a cluster of customer who prefer small size items. The purchase power is moderately low and mostly orders Durry. Likewise, Cluster 6 is a group of customers with moderate Quantity requirements of Durry and Hand Tuffed Products. The company can target the customer based on their order preference.

Using the recommendation System of Filtering, the customer can be filtered and recommended the different types of items. Based on the recommended filtering system, Customer H-2 can be recommended with Double Back item and Color of the item the customer usually purchased to Grey, Pink and Bluish Pink.

Likewise, Customer T-5 can be recommended with item Knotted and Square shape of the items the customer usually purchases.