DT/RF/LR/NN Evaluation with Clustering and Recommendation System

Karthick Sharan

4/30/2022

```
library(dplyr)
library(tidyverse)
library(readxl)
library(DataExplorer)
library(rpart)
library(rpart.plot)
library(data.table)
library(randomForest)
library(caret)
library(ROCR)
library(gridExtra)
library(GGally)      # Pair plots
library(ggcorrplot)
library(mice)        # Imputation
library(fastDummies)
library(ISLR)
library(standardize)  #Scaling
library(ROSE)  # Oversampling
library(e1071) # Oversampling
library(nnet) #neuralnet
library(NeuralNetTools)
library(arules) #Association
library(dlookr)
library(cluster)
library(factoextra)
```

1. Providing key insights using exploratory data analysis and visualisation

1.1 Data Exploration

- importing Data from Sample ONLY sheet *

```
dfm <- read_excel("C:/Users/rshara4/Documents/hw3/Champo Carpets.xlsx",
sheet=4)
head(dfm)

## # A tibble: 6 x 25
##   CustomerCode CountryName   USA    UK Italy Belgium Romania Australia
India
```

```
##   <chr>        <chr>       <dbl> <dbl> <dbl>   <dbl>   <dbl>     <dbl>
<dbl>
## 1 CC           INDIA           0     0     0       0       0         0
1
## 2 M-1          USA             1     0     0       0       0         0
0
## 3 M-1          USA             1     0     0       0       0         0
0
## 4 M-1          USA             1     0     0       0       0         0
0
## 5 M-1          USA             1     0     0       0       0         0
0
## 6 CC           INDIA           0     0     0       0       0         0
1
## # ... with 16 more variables: QtyRequired <dbl>, ITEM_NAME <chr>,
## #   Hand Tufted <dbl>, Durry <dbl>, Double Back <dbl>, Hand Woven <dbl>,
## #   Knotted <dbl>, Jacquard <dbl>, Handloom <dbl>, Other <dbl>,
## #   ShapeName <chr>, REC <dbl>, Round <dbl>, Square <dbl>, AreaFt <dbl>,
## #   Order Conversion <dbl>
```

- importing Data from RAW Data-Order and Sample sheet *

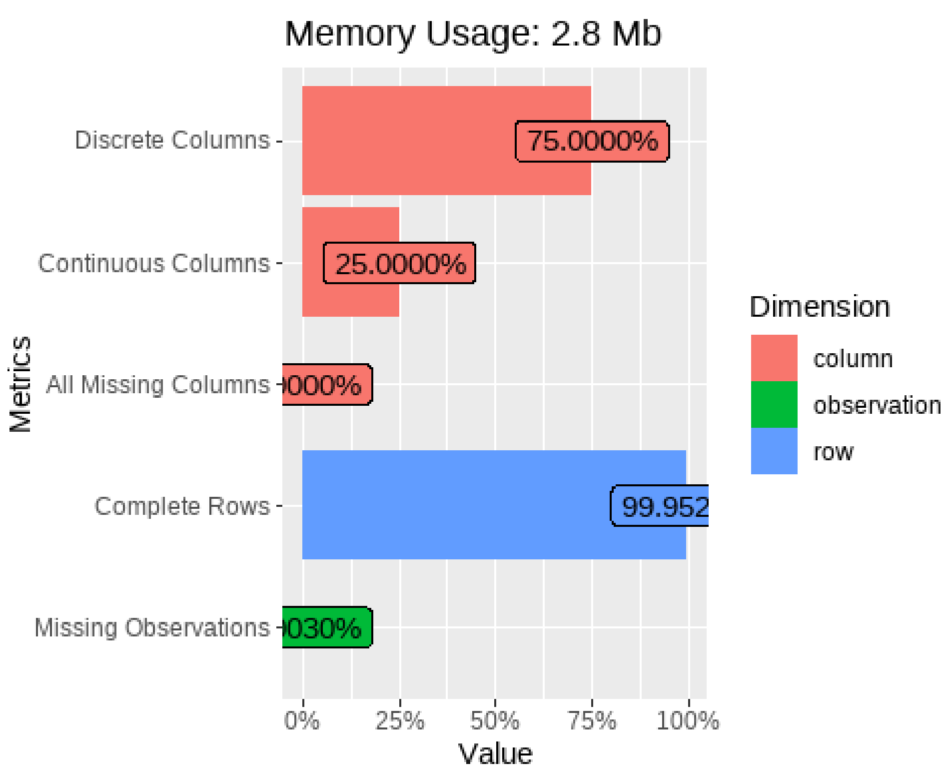
```
df <- read_excel("C:/Users/rshara4/Documents/hw3/Champo Carpets.xlsx",
sheet=2)
head(df)

## # A tibble: 6 x 16
##   OrderType OrderCategory CustomerCode CountryName CustomerOrderNo
##   <chr>     <chr>         <chr>        <chr>       <chr>
## 1 Area Wise Order         H-1          USA         1873354
## 2 Area Wise Order         H-1          USA         1873354
## 3 Area Wise Order         H-1          USA         1873354
## 4 Area Wise Order         H-1          USA         1918436
## 5 Area Wise Order         H-1          USA         1873354
## 6 Area Wise Order         H-1          USA         1918436
## # ... with 11 more variables: Custorderdate <dttm>, UnitName <chr>,
## #   QtyRequired <dbl>, TotalArea <dbl>, Amount <dbl>, ITEM_NAME <chr>,
## #   QualityName <chr>, DesignName <chr>, ColorName <chr>, ShapeName <chr>,
## #   AreaFt <dbl>

str(df)

## tibble [18,955 x 16] (S3: tbl_df/tbl/data.frame)
##  $ OrderType      : chr [1:18955] "Area Wise" "Area Wise" "Area Wise"
"Area Wise" ...
##  $ OrderCategory  : chr [1:18955] "Order" "Order" "Order" "Order" ...
##  $ CustomerCode   : chr [1:18955] "H-1" "H-1" "H-1" "H-1" ...
##  $ CountryName    : chr [1:18955] "USA" "USA" "USA" "USA" ...
##  $ CustomerOrderNo: chr [1:18955] "1873354" "1873354" "1873354" "1918436"
...
##  $ Custorderdate  : POSIXct[1:18955], format: "2017-01-16" "2017-01-16"
```

```
...
##  $ UnitName       : chr [1:18955] "Ft" "Ft" "Ft" "Ft" ...
##  $ QtyRequired    : num [1:18955] 2 2 2 5 5 4 6 16 2 4 ...
##  $ TotalArea      : num [1:18955] 6 9 54 54 71.2 ...
##  $ Amount         : num [1:18955] 12 18 108 270 356 ...
##  $ ITEM_NAME      : chr [1:18955] "HAND TUFTED" "HAND TUFTED" "HAND
TUFTED" "HAND TUFTED" ...
##  $ QualityName    : chr [1:18955] "TUFTED 30C HARD TWIST" "TUFTED 30C HARD
TWIST" "TUFTED 30C HARD TWIST" "TUFTED 30C HARD TWIST" ...
##  $ DesignName     : chr [1:18955] "OLD LONDON [3715]" "OLD LONDON [3715]"
"OLD LONDON [3715]" "OLD LONDON [3715]" ...
##  $ ColorName      : chr [1:18955] "BEIGE" "BEIGE" "BEIGE" "BEIGE" ...
##  $ ShapeName      : chr [1:18955] "REC" "REC" "REC" "REC" ...
##  $ AreaFt         : num [1:18955] 6 9 54 54 71.2 ...
```

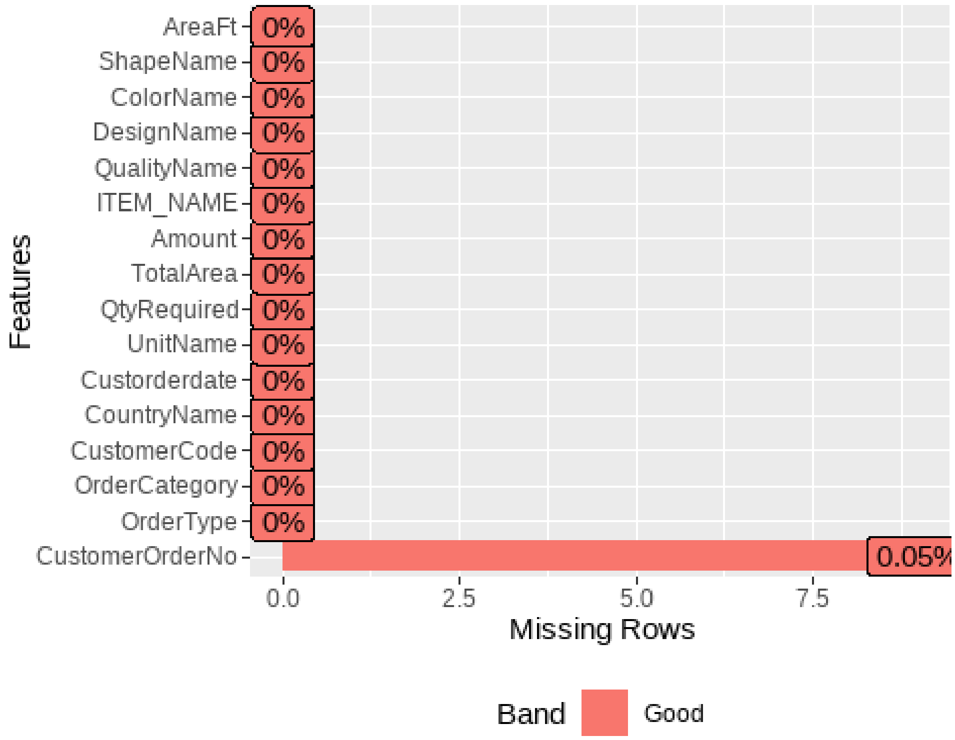
```
summary(df)
```

```
##   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          ShapeName
##  Length:18955       Length:18955       Length:18955       Length:18955
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##      AreaFt
##  Min.   :  0.4444
##  1st Qu.:  8.4375
##  Median : 35.0000
##  Mean   : 44.4695
```

```
##   3rd Qu.: 64.7361
##   Max.   :645.7222

# Reading dataset
col <- names(df)
introduce(df)

## # A tibble: 1 x 9
##    rows columns discrete_columns continuous_columns all_missing_columns
##   <int>   <int>            <int>              <int>               <int>
## 1 18955      16               12                  4                   0
## # ... with 4 more variables: total_missing_values <int>, complete_rows
<int>,
## #   total_observations <int>, memory_usage <dbl>

plot_intro(df)
```




```
plot_missing(df)
```

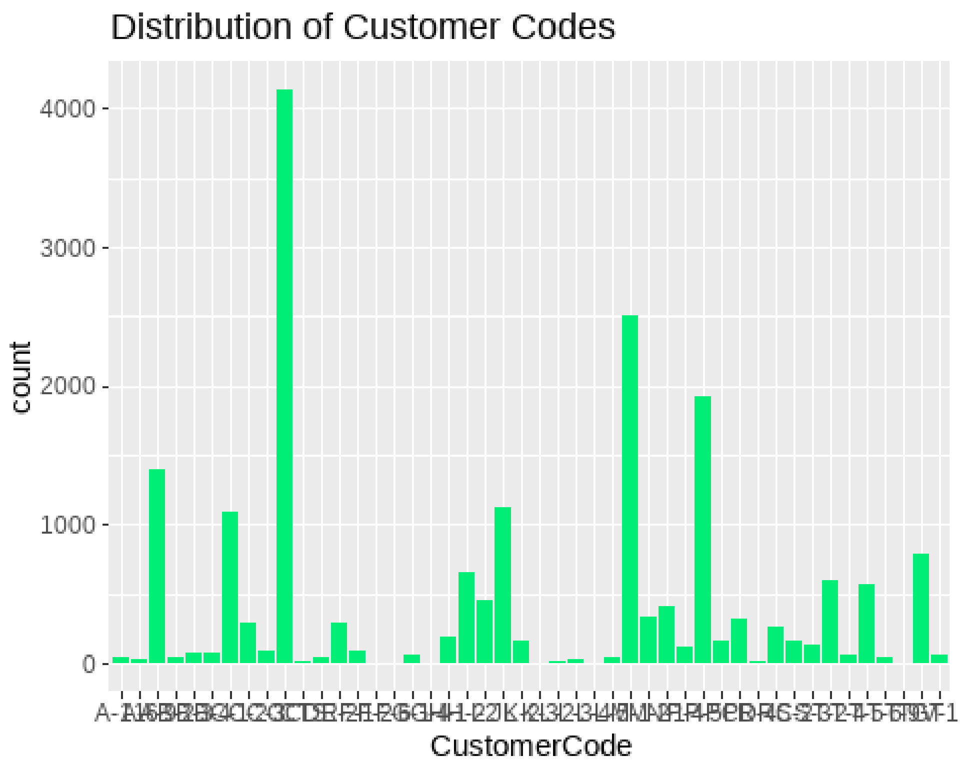


Feature Extraction

```
drop_col <- c("CustomerOrderNo","Custorderdate")


df$month <-  format(df$Custorderdate, "%m")
df$year <- format(df$Custorderdate, "%Y")

df <- select(df, -drop_col)

## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(drop_col)` instead of `drop_col` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

Feature selection is not being used as there is already only limited variables for exploration

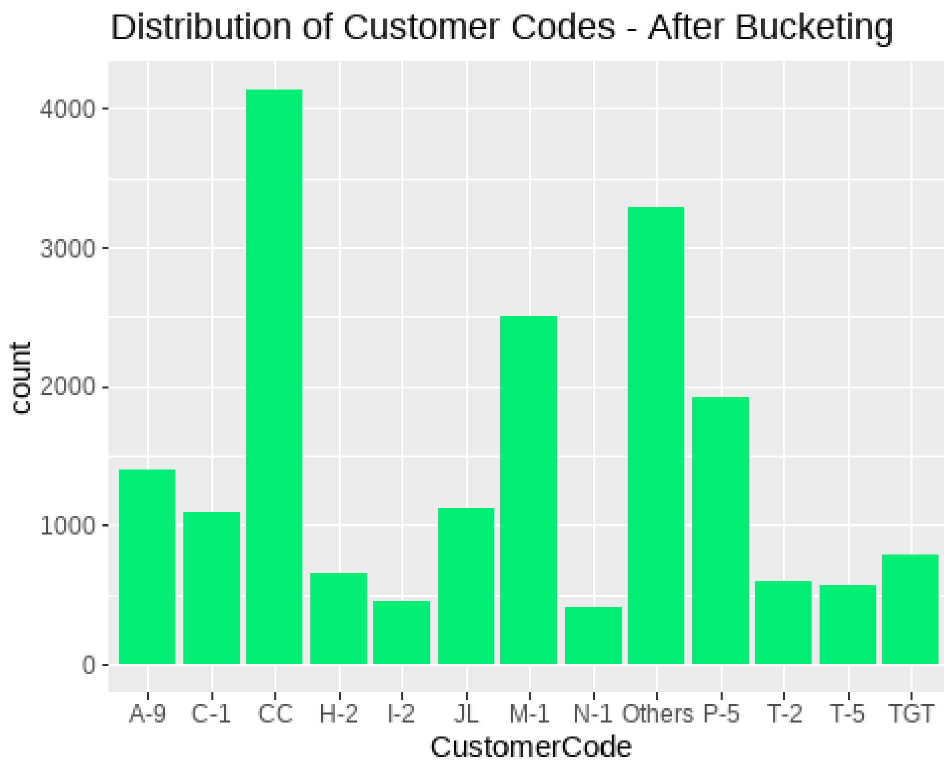
```
print(ggplot(df,aes_string(x='CustomerCode'))+geom_bar(fill='springgreen2')+
        ggtitle("Distribution of Customer Codes"))
```



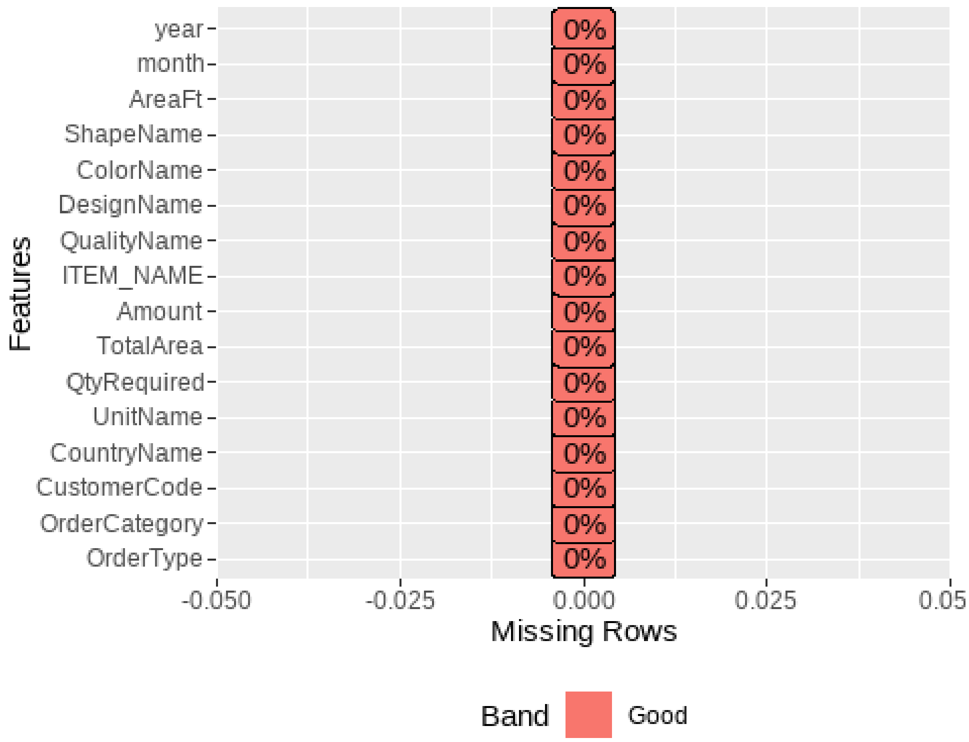

```
# Creating bucket in program code to reduce categories
sort(table(df['CustomerCode']), decreasing = TRUE)

##
##   CC  M-1  P-5  A-9   JL  C-1  TGT  H-2  T-2  T-5  I-2  N-1  M-2   PD  C-2
E-2
## 4135 2499 1930 1395 1128 1097  785  661  596  566  456  416  332  322  295
287
##   RC  H-1  S-2  K-2   PC  S-3  P-4  C-3  F-1  B-4  B-3  V-1  T-4  G-1  B-2
DR
##  265  193  168  165  155  138  112   87   87   75   73   64   59   56   48
46
## A-11  L-5  T-6  L-3  A-6  L-2  CTS  R-4  G-4  F-6  L-4  F-2  K-3  T-9
##   44   41   40   38   25   22   20   10    7    5    4    3    3    2

# selecting top categories
code <- c("CC","M-1","P-5","A-9","JL","C-1","TGT","H-2","T-2","T-5","I-2","N-
1")
df$CustomerCode <- ifelse(df$CustomerCode %in% code,
df$CustomerCode,"Others")

#converting other subgroups to "Others"
table(df['CustomerCode'])

##
##    A-9    C-1     CC    H-2    I-2     JL    M-1    N-1 Others    P-5
T-2
```

```
## 1395 1097 4135 661 456 1128 2499 416 3291 1930
596
##  T-5  TGT
##  566  785
```

```
print(ggplot(df,aes_string(x='CustomerCode'))+geom_bar(fill='springgreen2')+
        ggtitle("Distribution of Customer Codes - After Bucketing"))
```




```
plot_missing(df)
```



We notice that there are no missing values in our dataset

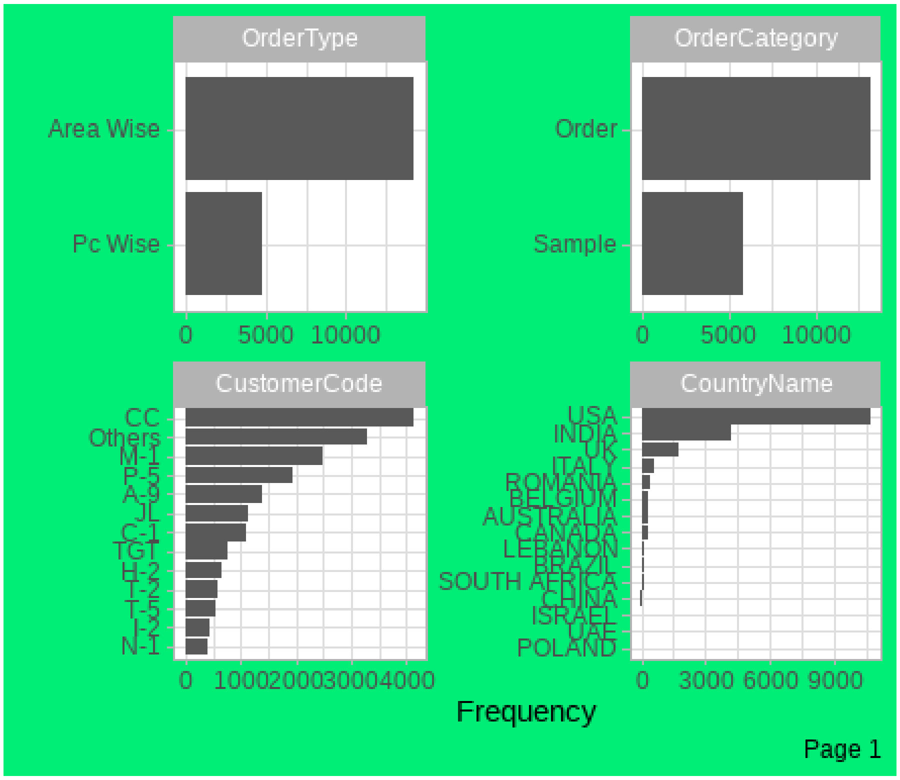
```
col <- names(df)
cat <- names(df[, sapply(df, class) == 'character'])
num <- names(select(df, -cat))

## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(cat)` instead of `cat` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.

cat # Categorical columns

##  [1] "OrderType"     "OrderCategory" "CustomerCode"  "CountryName"
##  [5] "UnitName"      "ITEM_NAME"     "QualityName"   "DesignName"
##  [9] "ColorName"     "ShapeName"     "month"         "year"

num # Numerical columns

## [1] "QtyRequired" "TotalArea"   "Amount"      "AreaFt"

df[cat] <- lapply(df[cat], factor)
str(df)

## tibble [18,955 x 16] (S3: tbl_df/tbl/data.frame)
##  $ OrderType    : Factor w/ 2 levels "Area Wise","Pc Wise": 1 1 1 1 1 1 1
1 1 1 ...
##  $ OrderCategory: Factor w/ 2 levels "Order","Sample": 1 1 1 1 1 1 1 1 1 1
```

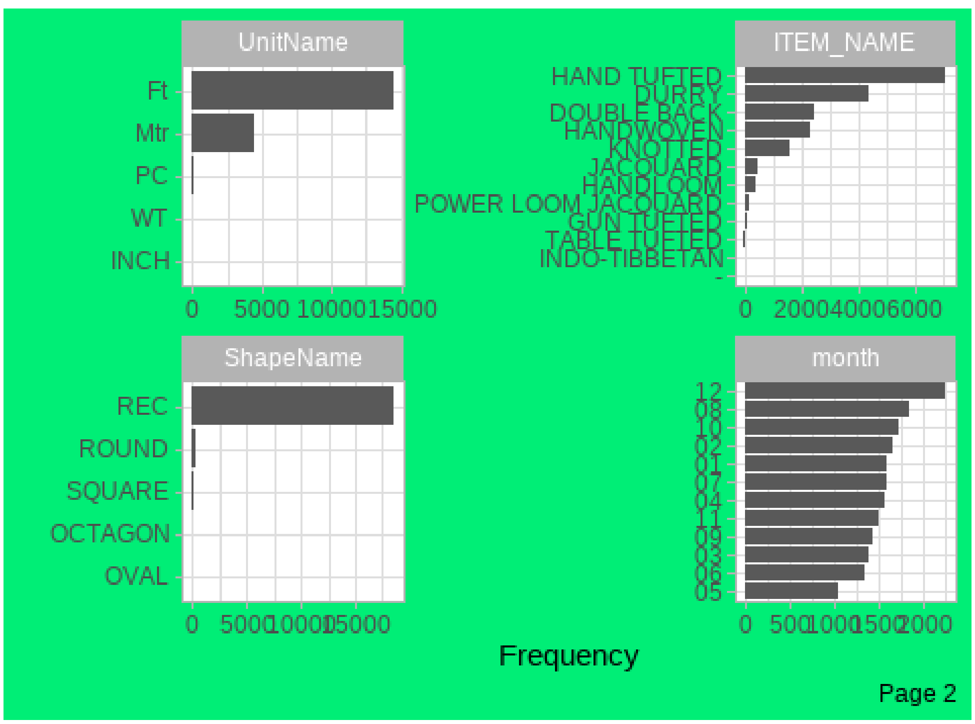
```
...
## $ CustomerCode : Factor w/ 13 levels "A-9","C-1","CC",..: 9 9 9 9 9 9 9 9
9 9 ...
## $ CountryName  : Factor w/ 15 levels "AUSTRALIA","BELGIUM",..: 15 15 15
15 15 15 15 15 15 15 ...
## $ UnitName     : Factor w/ 5 levels "Ft","INCH","Mtr",..: 1 1 1 1 1 1 1 1
1 1 ...
## $ QtyRequired  : num [1:18955] 2 2 2 5 5 4 6 16 2 4 ...
## $ TotalArea    : num [1:18955] 6 9 54 54 71.2 ...
## $ Amount       : num [1:18955] 12 18 108 270 356 ...
## $ ITEM_NAME    : Factor w/ 12 levels "-","DOUBLE BACK",..: 5 5 5 5 5 5 5
5 5 5 ...
## $ QualityName  : Factor w/ 382 levels "D.B 30C H/S LEFA VISCOSE+45C
WOOL",..: 300 300 300 300 300 300 300 300 300 300 ...
## $ DesignName   : Factor w/ 2254 levels "1 LOOP/1 CUT",..: 1793 1793 1793
1793 1793 1793 1793 1793 1793 1793 ...
## $ ColorName    : Factor w/ 815 levels "0620+18-1239",..: 125 125 125 125
125 125 385 385 385 385 ...
## $ ShapeName    : Factor w/ 5 levels "OCTAGON","OVAL",..: 3 3 3 3 3 3 3 3
4 4 ...
## $ AreaFt       : num [1:18955] 6 9 54 54 71.2 ...
## $ month        : Factor w/ 12 levels "01","02","03",..: 1 1 1 2 1 2 1 2 1
2 ...
## $ year         : Factor w/ 4 levels "2017","2018",..: 1 1 1 1 1 1 1 1 1 1
...
```

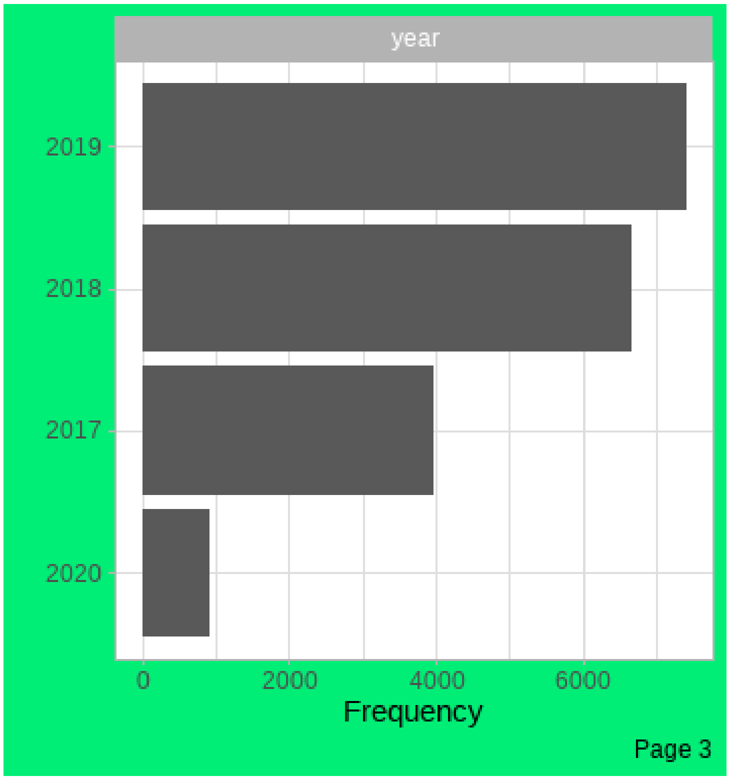
1.1.1 Exploring/Visulization of Categorical Data

```r
plot_bar(df, ncol = 2, nrow = 2, ggtheme = theme_light(),theme_config = list(
  "plot.background" = element_rect(fill = "springgreen2"),
  "aspect.ratio" = 1))
```

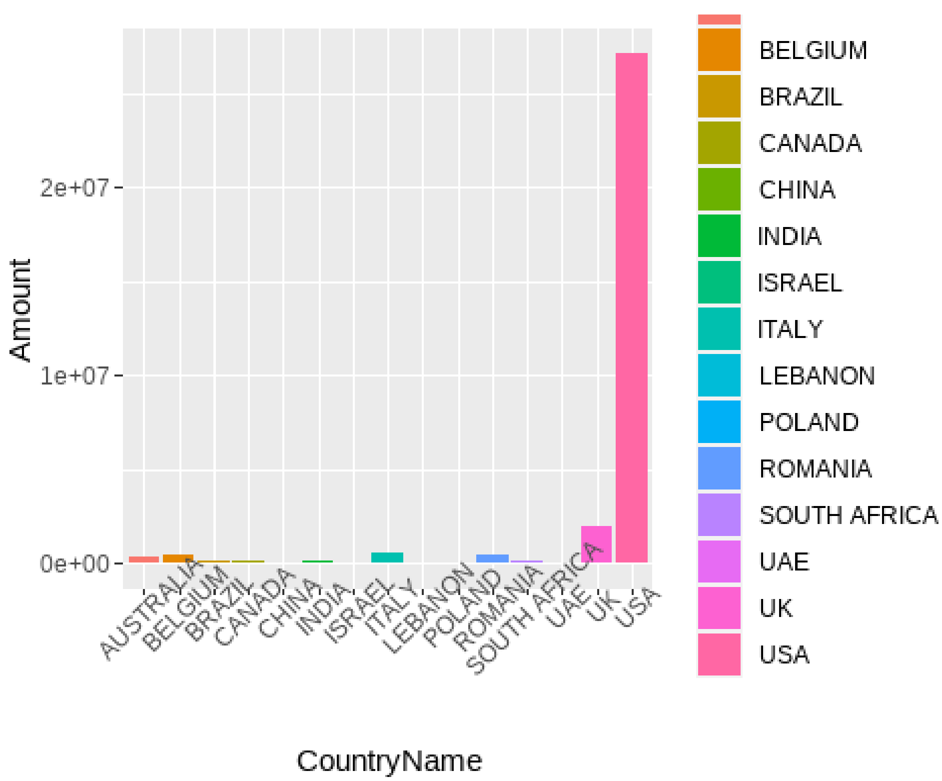
```
## 3 columns ignored with more than 50 categories.
## QualityName: 382 categories
## DesignName: 2254 categories
## ColorName: 815 categories
```


OrderType
Area Wise
Pc Wise
0
5000
10000
OrderCategory
Order
Sample
0
5000
10000
CustomerCode
CC
Others
M-1
P-5
A-9
JL
C-1
TGT
H-2
T-2
T-5
I-2
N-1
0
1000
2000
3000
4000
CountryName
USA
INDIA
UK
ITALY
ROMANIA
BELGIUM
AUSTRALIA
CANADA
LEBANON
BRAZIL
SOUTH AFRICA
CHINA
ISRAEL
UAE
POLAND
0
3000
6000
9000
Frequency
Page 1

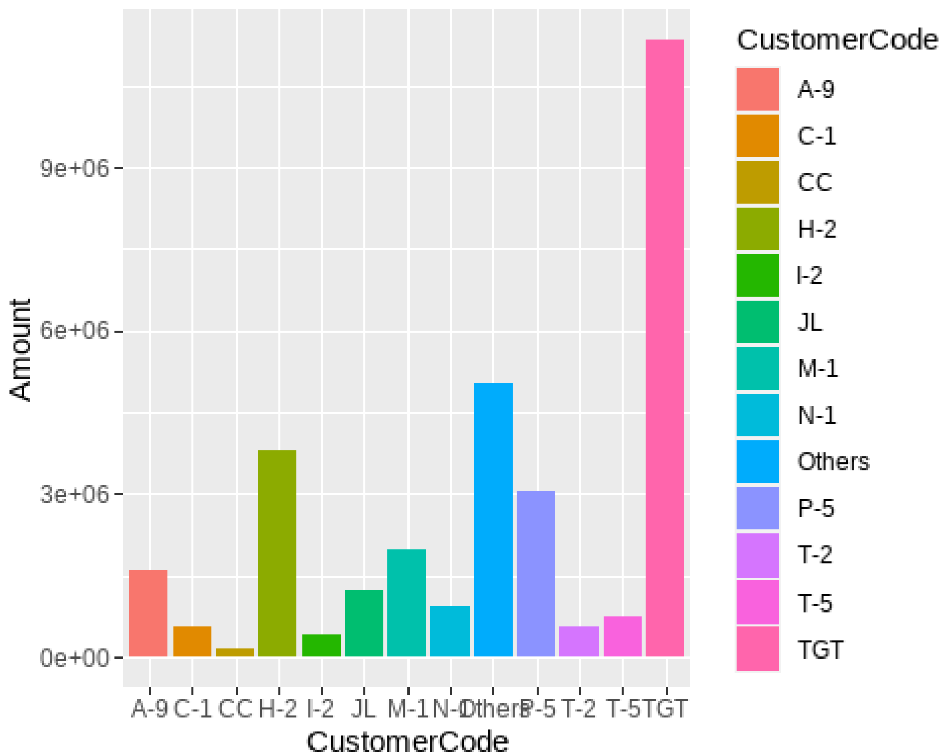

UnitName
Ft
Mtr
PC
WT
INCH
0
5000
10000
15000
ITEM_NAME
HAND TUFTED
DURRY
DOUBLE BACK
HANDWOVEN
KNOTTED
JACQUARD
HANDLOOM
POWER LOOM JACQUARD
GUN TUFTED
TABLE TUFTED
INDO-TIBBETAN
-
0
2000
4000
6000
ShapeName
REC
ROUND
SQUARE
OCTAGON
OVAL
0
5000
10000
15000
month
12
08
10
02
01
07
04
11
09
03
06
05
0
500
1000
1500
2000
Frequency
Page 2



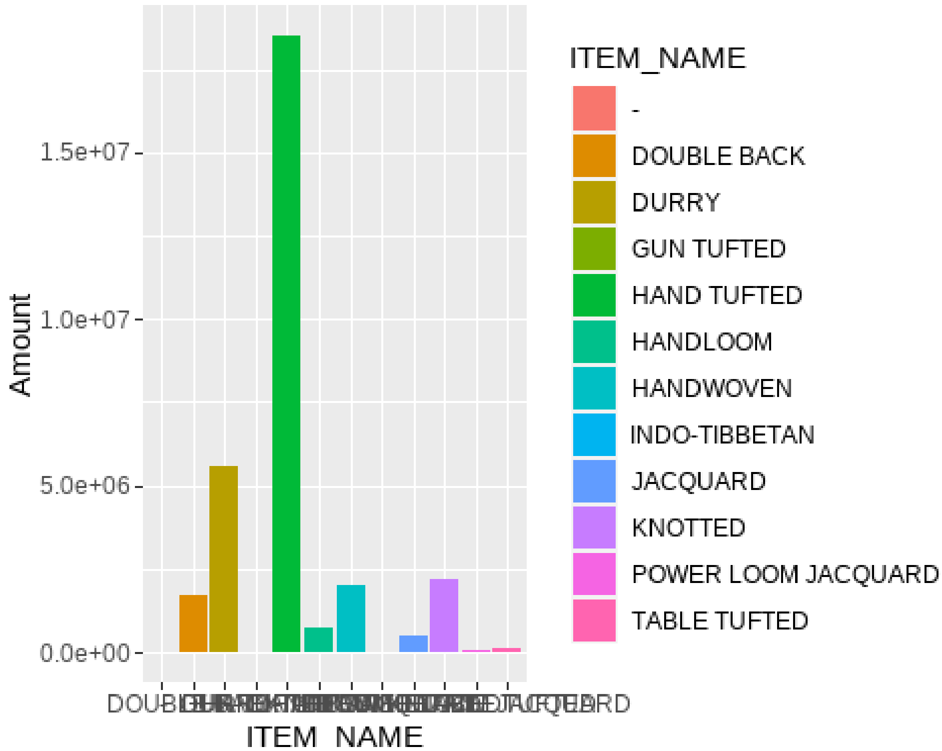

```r
# Visualizing total revenue earned from each country
ggplot(df,aes_string(x='CountryName', y='Amount', fill='CountryName'))+
  geom_bar(stat='identity') + theme(axis.text.x = element_text(angle = 45))
```



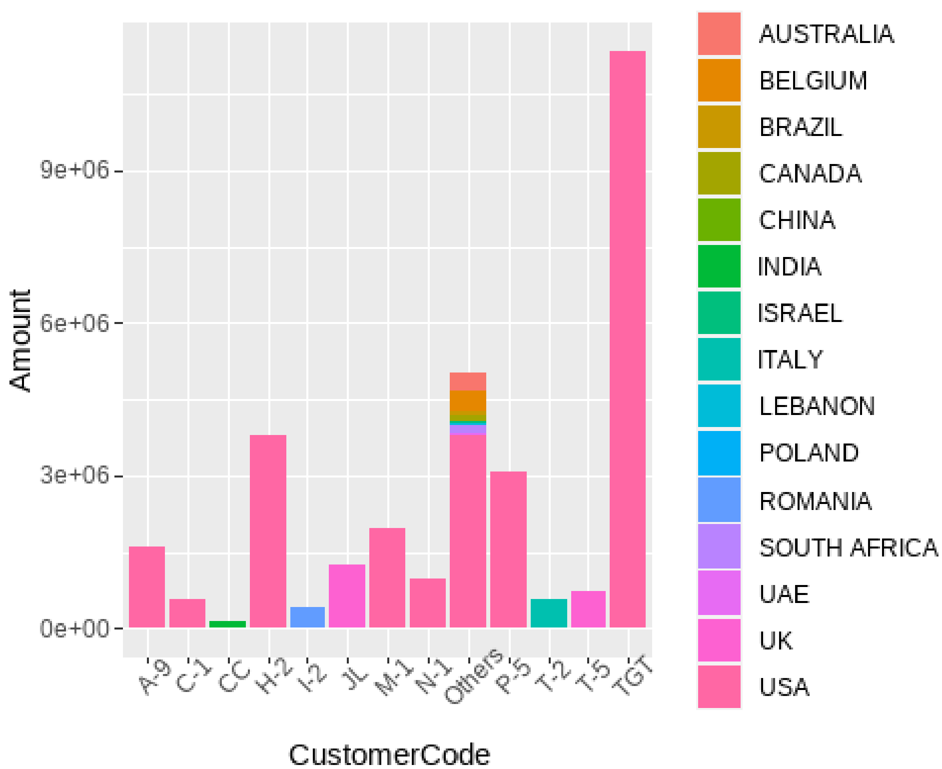
```r
ggplot(df,aes_string(x='CustomerCode', y='Amount', fill='CustomerCode'))+
  geom_bar(stat='identity')
```




```r
ggplot(df,aes_string(x='ITEM_NAME', y='Amount', fill='ITEM_NAME'))+
  geom_bar(stat='identity')
```




```
ggplot(df,aes_string(x='CustomerCode', y='Amount', fill='CountryName'))+
  geom_bar(stat='identity') + theme(axis.text.x = element_text(angle = 45))
```



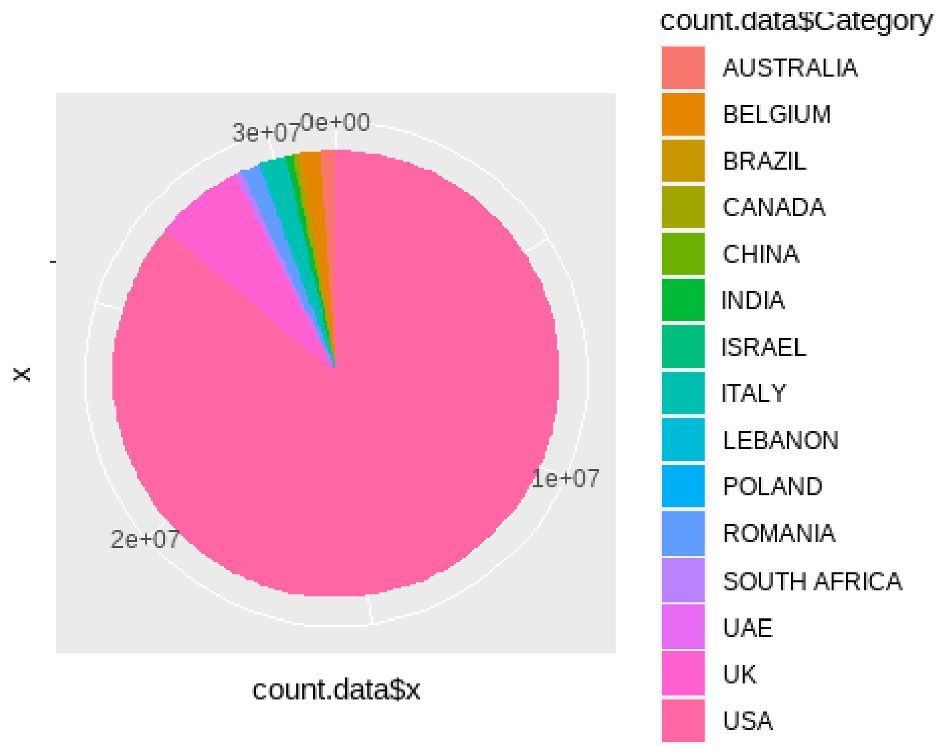
```r
count.data <- aggregate(df$Amount, by=list(Category=df$CountryName), FUN=sum)
count.data
```

```
##        Category           x
## 1     AUSTRALIA   356938.86
## 2       BELGIUM   426791.41
## 3        BRAZIL    59877.27
## 4        CANADA   116778.30
## 5         CHINA    24919.96
## 6         INDIA   147574.00
## 7        ISRAEL    17128.88
## 8         ITALY   563098.85
## 9       LEBANON    56742.73
## 10       POLAND        0.00
## 11      ROMANIA   426626.05
## 12 SOUTH AFRICA   130457.99
## 13          UAE    44234.00
## 14           UK  1965411.23
## 15          USA 27083224.09
```

```r
ggplot(count.data, aes(x="", y=count.data$x, fill=count.data$Category)) +
  geom_bar(stat="identity", width=1) +
  coord_polar("y", start=0)
```

```
## Warning: Use of `count.data$x` is discouraged. Use `x` instead.
```

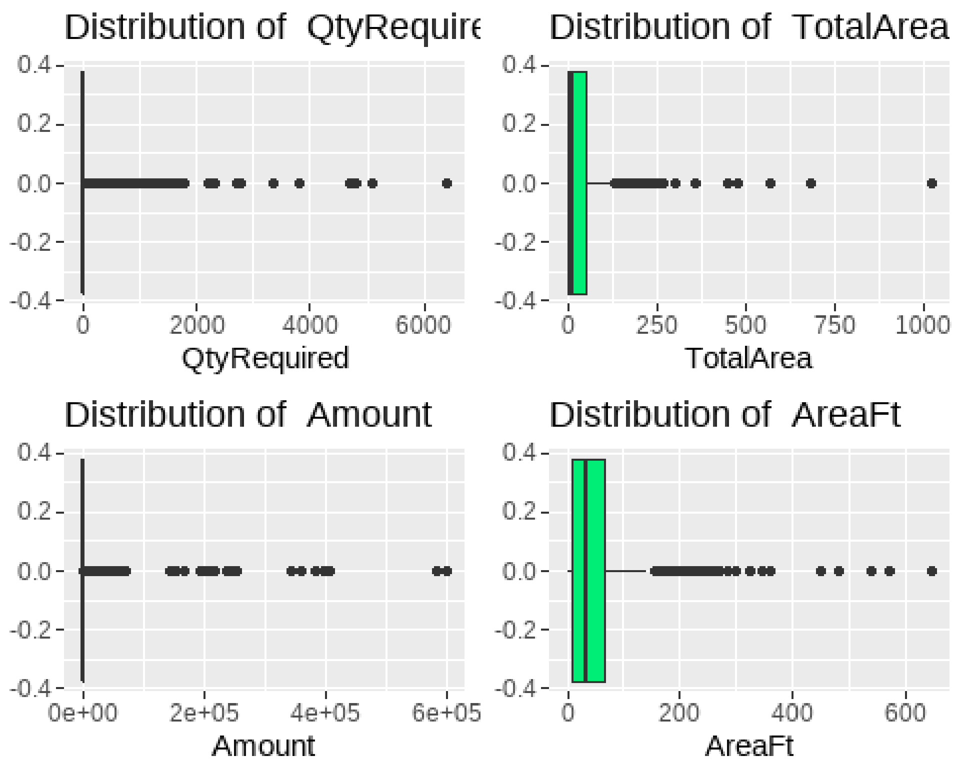
```
## Warning: Use of `count.data$Category` is discouraged. Use `Category`
instead.
```



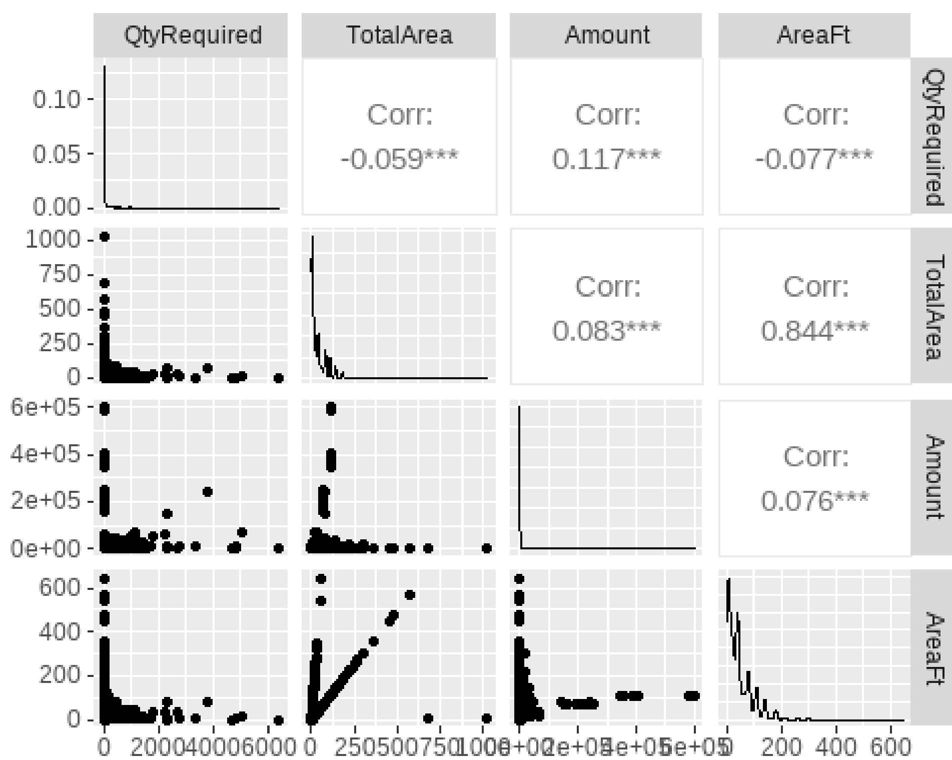
From the above plots we notice the following: * We have the most data for the year 2019, in the time period 2017-20 * There are twice as many orders as samples * Hand Tufted is the highest sold product by a large margin * USA seems to be the largest exported country with more than 75% share * TGT is the largest purchasing customer

1.1.2 Exploring/Visulization of Numerical Data

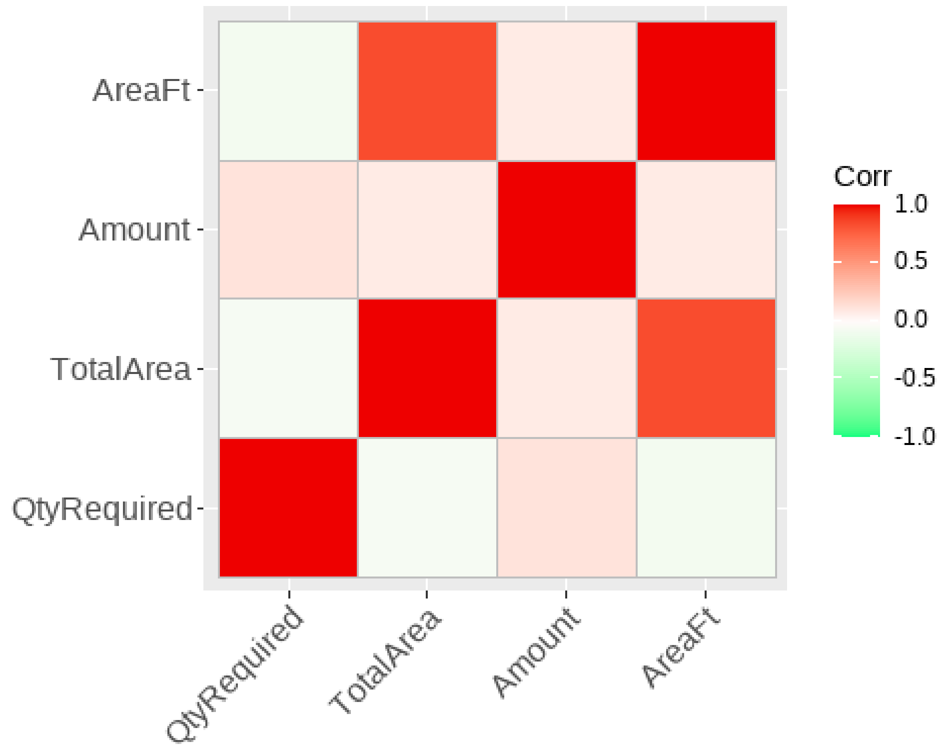
```
plot_list <- list()
n=1
for (i in num){
plot_list[[n]] <-
ggplot(df,aes_string(x=i))+geom_boxplot(fill='springgreen2')+
ggtitle(paste("Distribution of ",i))
n=n+1
}
grid.arrange(grobs=plot_list,ncol=2)
```




```r
# pair plot for input varibales
ggpairs(df[num], upper = list(continuous = wrap("cor", size = 4)))
```



```
ggcorrplot(cor(df[num]), ggtheme = 'theme_dark', show.legend = TRUE,
colors=c('springgreen1','snow1','red2'))
```



We can observe very high correlation between AreaFt and TotalArea, so to treat this multicollinearity we can simply remove one of the column since their corr value is very close to 1.

Dropping rows to teat multicollinearity

```
df <- select(df, -'TotalArea')

# Removing total area since we have AreaFt with response variable in the
sheet

# Filtering Orders and Samples
dfo <- filter(df, OrderCategory == 'Order')
dfs <- filter(df, OrderCategory == 'Sample')
```

1.2 Exploration of Sample Data

```
head(dfs)

## # A tibble: 6 x 15
##   OrderType OrderCategory CustomerCode CountryName UnitName QtyRequired
Amount
##   <fct>     <fct>         <fct>        <fct>       <fct>          <dbl>
<dbl>
## 1 Area Wise Sample        CC           INDIA       Ft                 1
0
```

```
## 2 Area Wise Sample        M-1          USA         Ft                 1
0
## 3 Area Wise Sample        M-1          USA         Ft                 2
0
## 4 Area Wise Sample        M-1          USA         Ft                 1
0
## 5 Area Wise Sample        M-1          USA         Ft                 1
0
## 6 Area Wise Sample        CC           INDIA       Ft                 1
0
## # ... with 8 more variables: ITEM_NAME <fct>, QualityName <fct>,
## #   DesignName <fct>, ColorName <fct>, ShapeName <fct>, AreaFt <dbl>,
## #   month <fct>, year <fct>
```

```
dfs$Target <- dfm$`Order Conversion`
str(dfs)
```

```
## tibble [5,820 x 16] (S3: tbl_df/tbl/data.frame)
##  $ OrderType    : Factor w/ 2 levels "Area Wise","Pc Wise": 1 1 1 1 1 1 1
1 1 1 ...
##  $ OrderCategory: Factor w/ 2 levels "Order","Sample": 2 2 2 2 2 2 2 2 2 2
...
##  $ CustomerCode : Factor w/ 13 levels "A-9","C-1","CC",..: 3 7 7 7 7 3 3 7
7 3 ...
##  $ CountryName  : Factor w/ 15 levels "AUSTRALIA","BELGIUM",..: 6 15 15 15
15 6 6 15 15 6 ...
##  $ UnitName     : Factor w/ 5 levels "Ft","INCH","Mtr",..: 1 1 1 1 1 1 1 1
1 1 ...
##  $ QtyRequired  : num [1:5820] 1 1 2 1 1 1 1 1 1 1 ...
##  $ Amount       : num [1:5820] 0 0 0 0 0 0 0 0 0 0 ...
##  $ ITEM_NAME    : Factor w/ 12 levels "-","DOUBLE BACK",..: 5 5 5 5 5 2 2
5 5 5 ...
##  $ QualityName  : Factor w/ 382 levels "D.B 30C H/S LEFA VISCOSE+45C
WOOL",..: 370 370 370 370 370 28 28 370 370 370 ...
##  $ DesignName   : Factor w/ 2254 levels "1 LOOP/1 CUT",..: 1156 1156 1149
1149 1427 1882 1882 1156 1156 1156 ...
##  $ ColorName    : Factor w/ 815 levels "0620+18-1239",..: 624 458 624 624
624 293 642 458 406 804 ...
##  $ ShapeName    : Factor w/ 5 levels "OCTAGON","OVAL",..: 3 3 3 3 3 3 3 3
3 3 ...
##  $ AreaFt       : num [1:5820] 80 80 80 80 80 80 80 40 108 54 ...
##  $ month        : Factor w/ 12 levels "01","02","03",..: 12 12 12 2 12 10
9 12 12 6 ...
##  $ year         : Factor w/ 4 levels "2017","2018",..: 2 2 2 3 2 2 2 2 2 2
...
##  $ Target       : num [1:5820] 1 1 1 1 1 1 1 1 0 1 ...
```

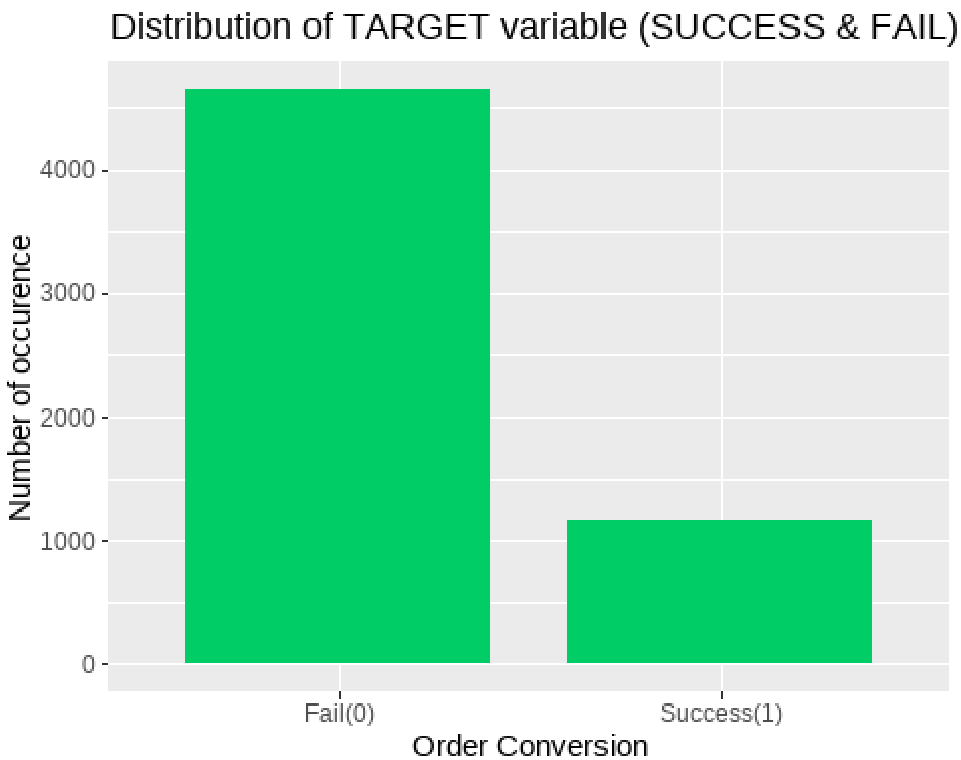
```
dfs[1010,]
```

```
## # A tibble: 1 x 16
##   OrderType OrderCategory CustomerCode CountryName UnitName QtyRequired
```

```
Amount
##   <fct>     <fct>         <fct>        <fct>       <fct>          <dbl>
<dbl>
## 1 Area Wise Sample        CC           INDIA       Ft                 1
0
## # ... with 9 more variables: ITEM_NAME <fct>, QualityName <fct>,
## #   DesignName <fct>, ColorName <fct>, ShapeName <fct>, AreaFt <dbl>,
## #   month <fct>, year <fct>, Target <dbl>

# We can verify with the provided data that this matches with the records
# for observation 1010 in Sample sheet

drop_col <- c("QualityName","DesignName","ColorName")
dfs <- select(dfs, -drop_col)

col <- names(dfs)
cat <- names(dfs[, sapply(dfs, class) == 'factor'])
num <- names(select(dfs, -cat))
```

Target Variable - Checking Balance

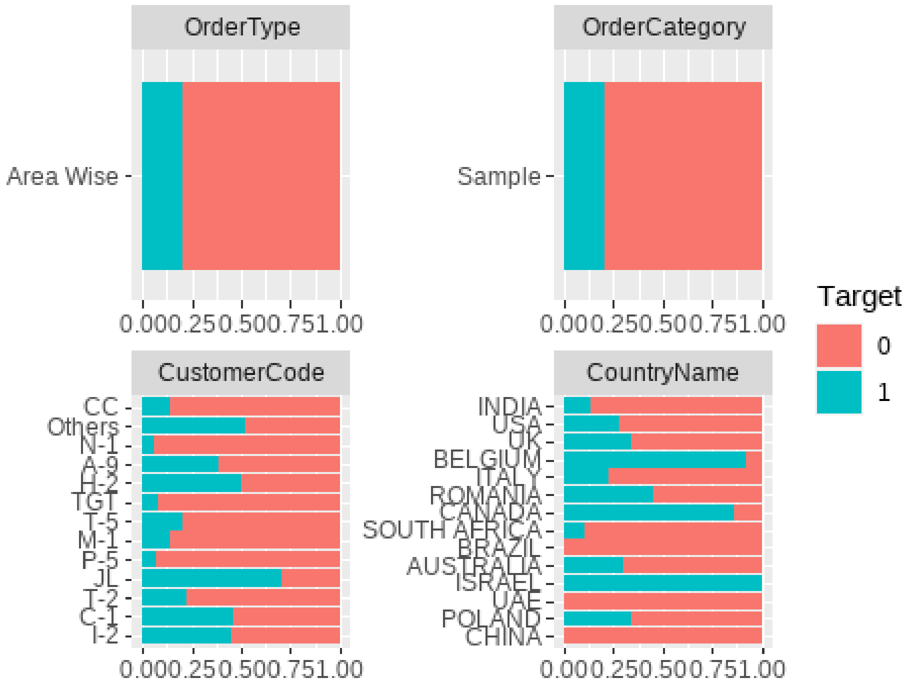
```
# Checking the category proportion of Target variable
df1 <- filter(dfs,dfs["Target"]==1)
df0 <- filter(dfs,dfs["Target"]==0)
print(paste("Count of Order Successful Conversion :",count(df1)))

## [1] "Count of Order Successful Conversion : 1169"

print(paste("Count of Order Conversion Failure :",count(df0)))

## [1] "Count of Order Conversion Failure : 4651"

df2 <- dfs
df2$Target <- as.factor(ifelse(df2$Target == 0, "Fail(0)", "Success(1)"))
ggplot(df2, aes(x=factor(Target)))+ geom_bar(stat="count",
width=0.8,fill='springgreen3')+
xlab('Order Conversion') + ylab('Number of occurence')+
ggtitle("Distribution of TARGET variable (SUCCESS & FAIL)")
```



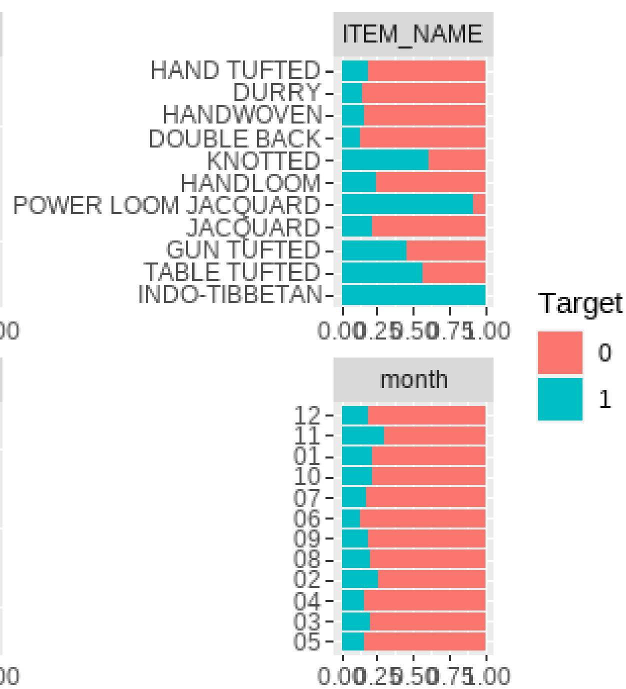
> Since the Target is Unbalanced we'll balance it before building a model.

1.2.1 Exploration of Categorical Variables

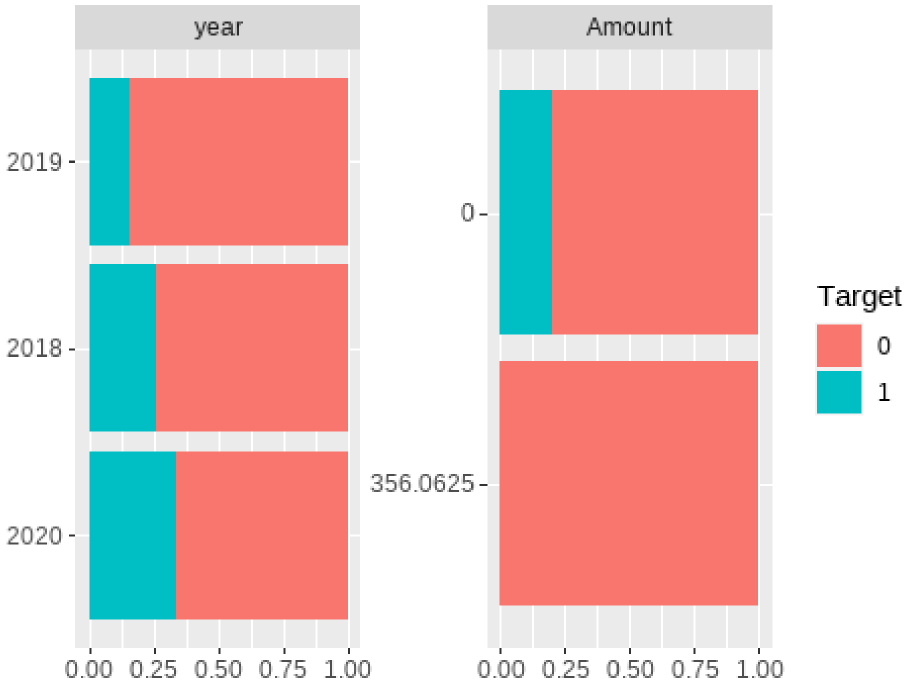
```
# Plotting all categorical variables with respect to Target variable

dfs$Target <- as.factor(dfs$Target)

plot_bar(dfs, by='Target', ncol = 2, nrow = 2)
```


OrderType
Area Wise
OrderCategory
Sample
CustomerCode
CC
Others
N-1
A-9
H-2
TGT
T-5
M-1
P-5
JL
T-2
C-1
I-2
CountryName
INDIA
USA
UK
BELGIUM
ITALY
ROMANIA
CANADA
SOUTH AFRICA
BRAZIL
AUSTRALIA
ISRAEL
UAE
POLAND
CHINA
0.00
0.25
0.50
0.75
1.00
Target
0
1

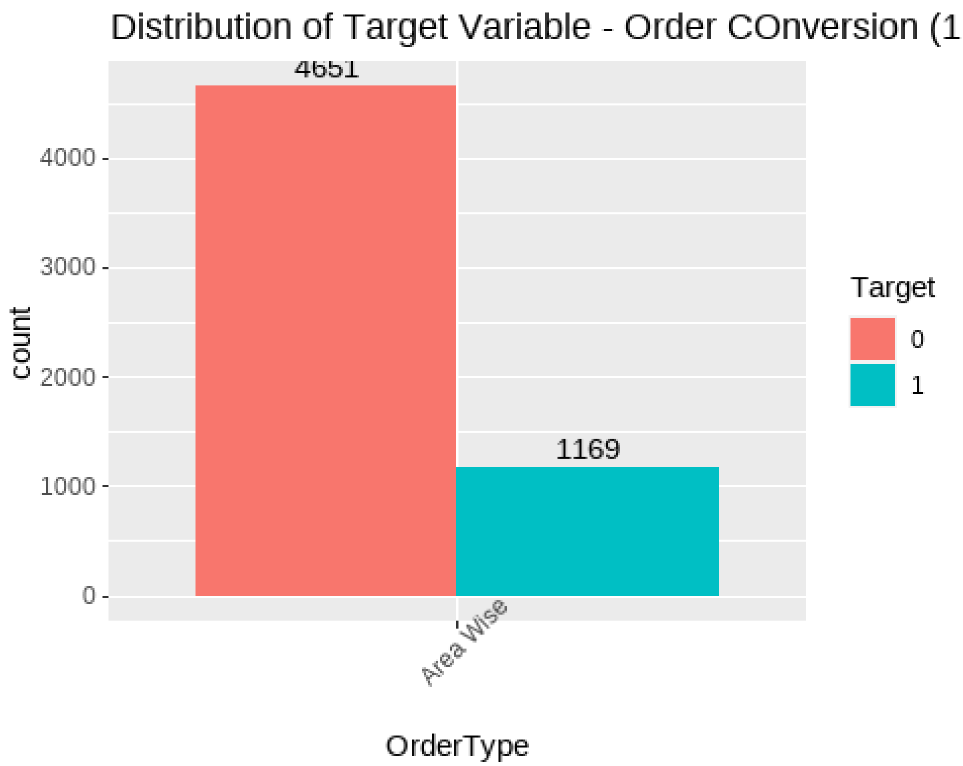
Page 1


UnitName
Ft
Mtr
ITEM_NAME
HAND TUFTED
DURRY
HANDWOVEN
DOUBLE BACK
KNOTTED
HANDLOOM
POWER LOOM JACQUARD
JACQUARD
GUN TUFTED
TABLE TUFTED
INDO-TIBBETAN
ShapeName
REC
ROUND
SQUARE
month
12
11
01
10
07
06
09
08
02
04
03
05
0.00
0.25
0.50
0.75
1.00
Target
0
1

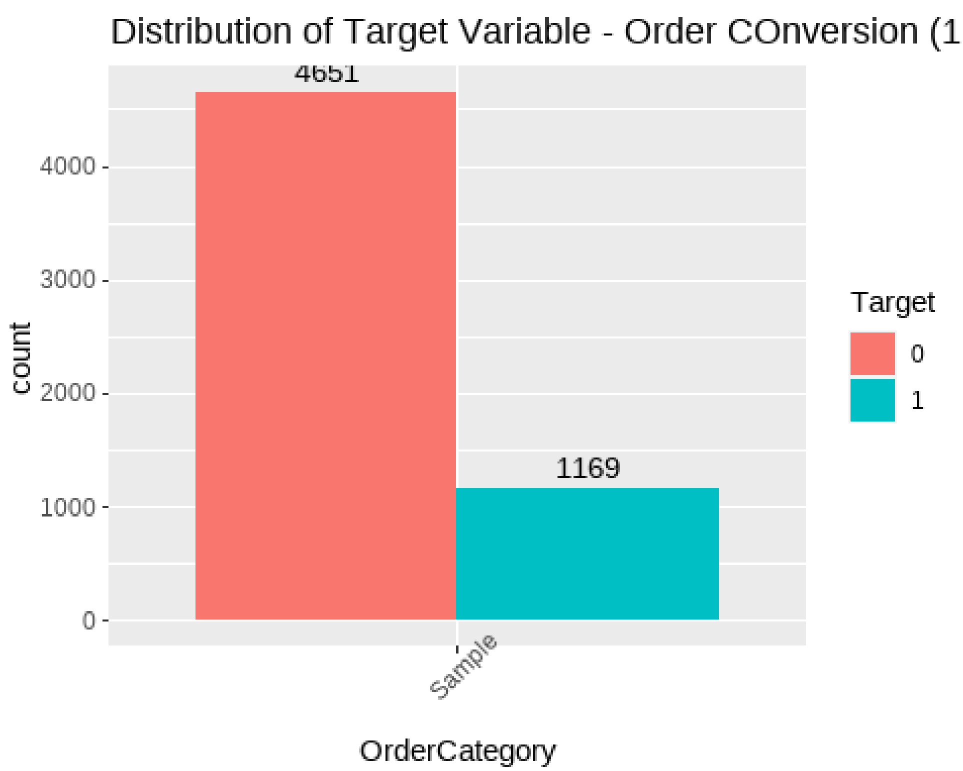
Page 2



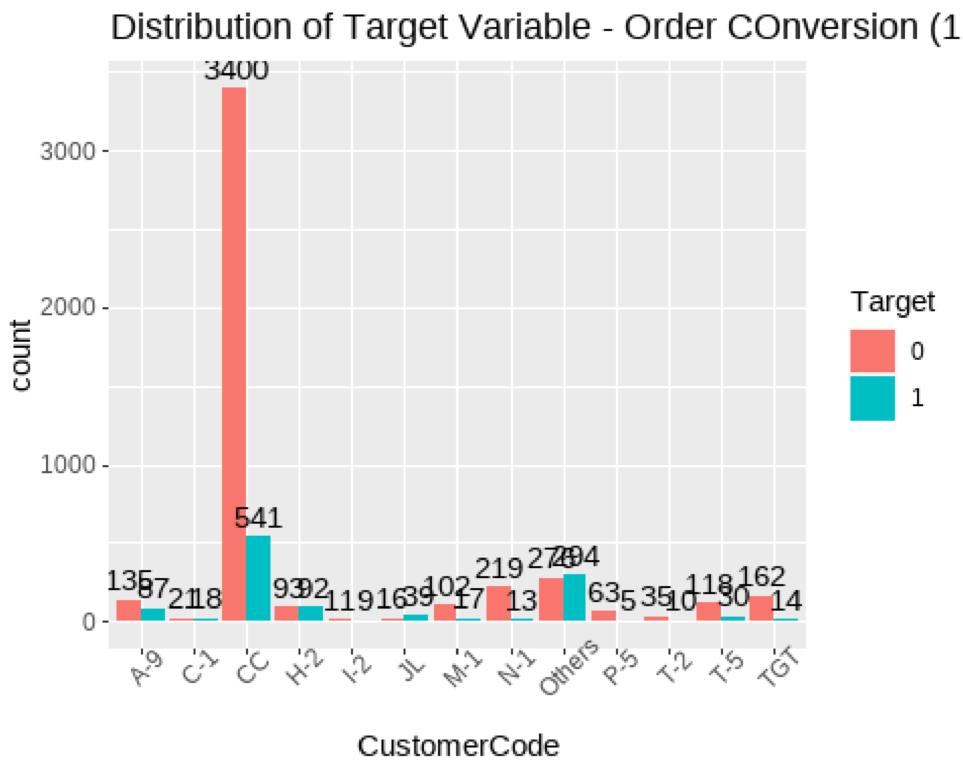

```
for (i in cat){
print(ggplot(dfs,aes_string(x=i, fill="Target"))+geom_bar(position="dodge")+
geom_text(stat='count', aes(label=..count..),position = position_dodge(0.9),
vjust=-0.5)+
ggtitle(paste("Distribution of Target Variable - Order COnversion (1 or 0)
in",i))+
theme(axis.text.x = element_text(angle = 45)))
}
```

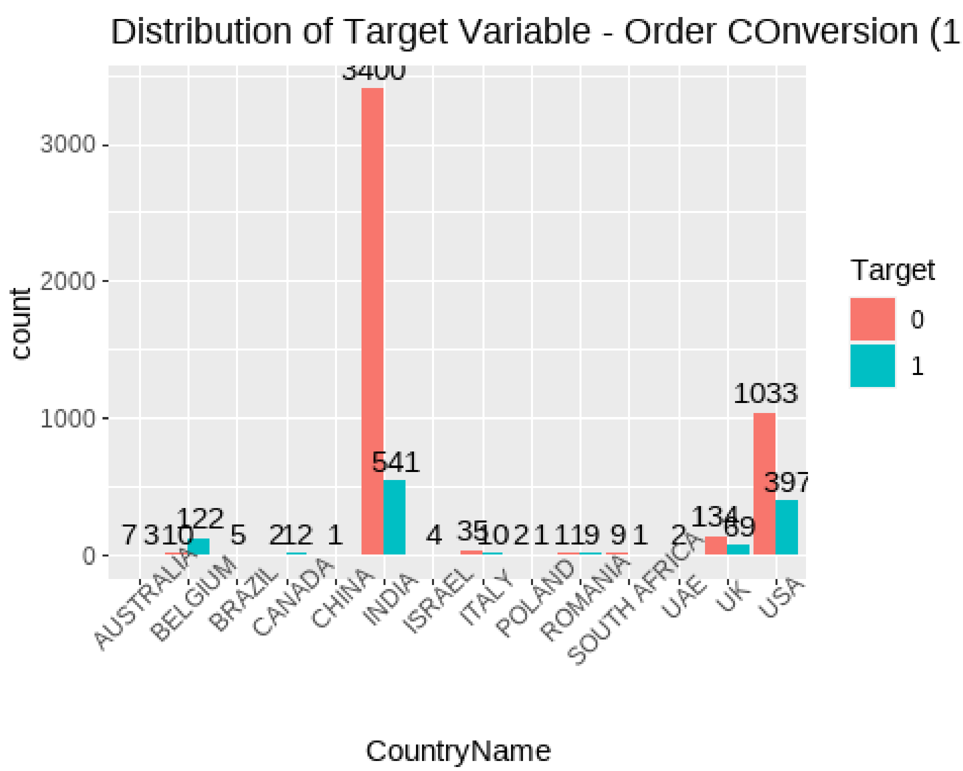


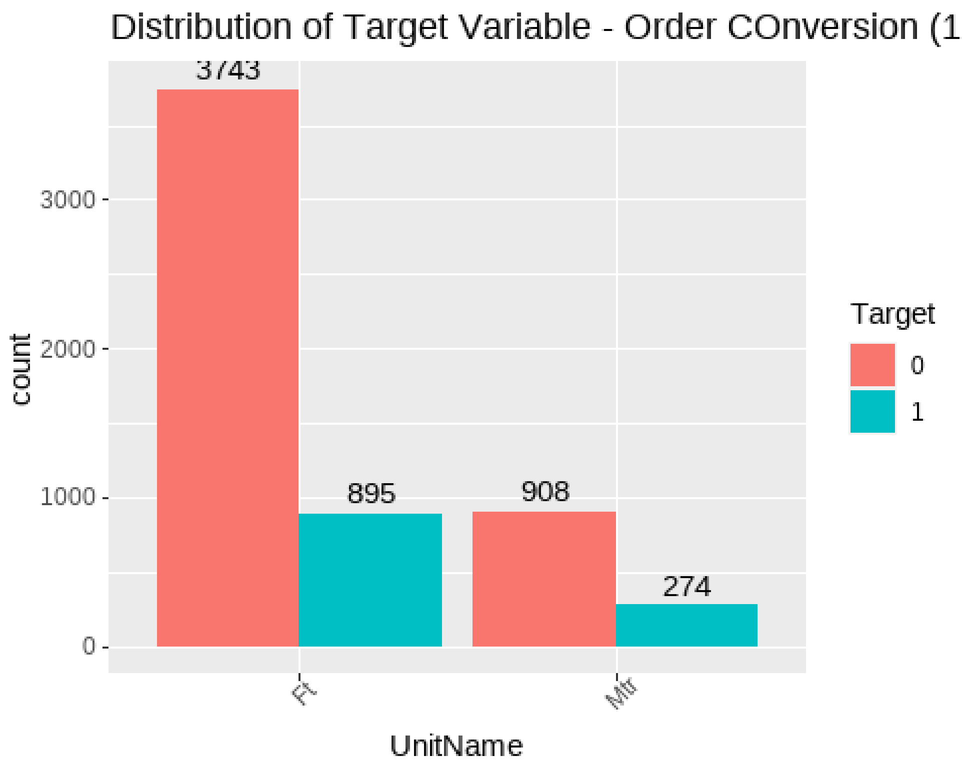
Distribution of Target Variable - Order COnversion (1
4651
1169
4000
3000
2000
1000
0
count
Area Wise
OrderType
Target
0
1

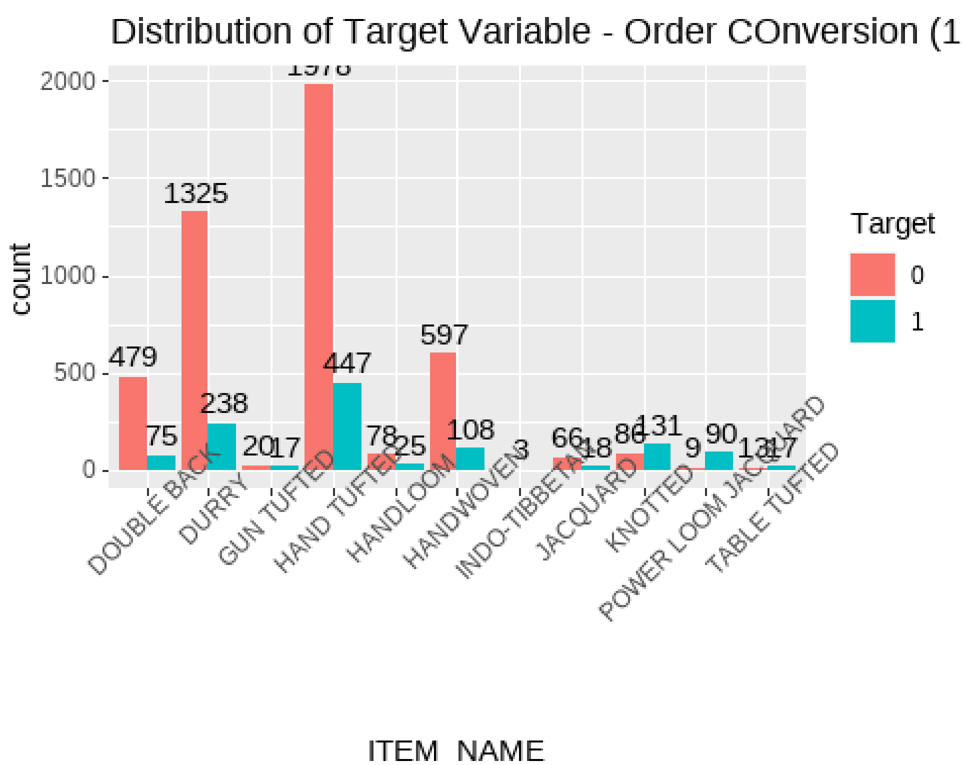


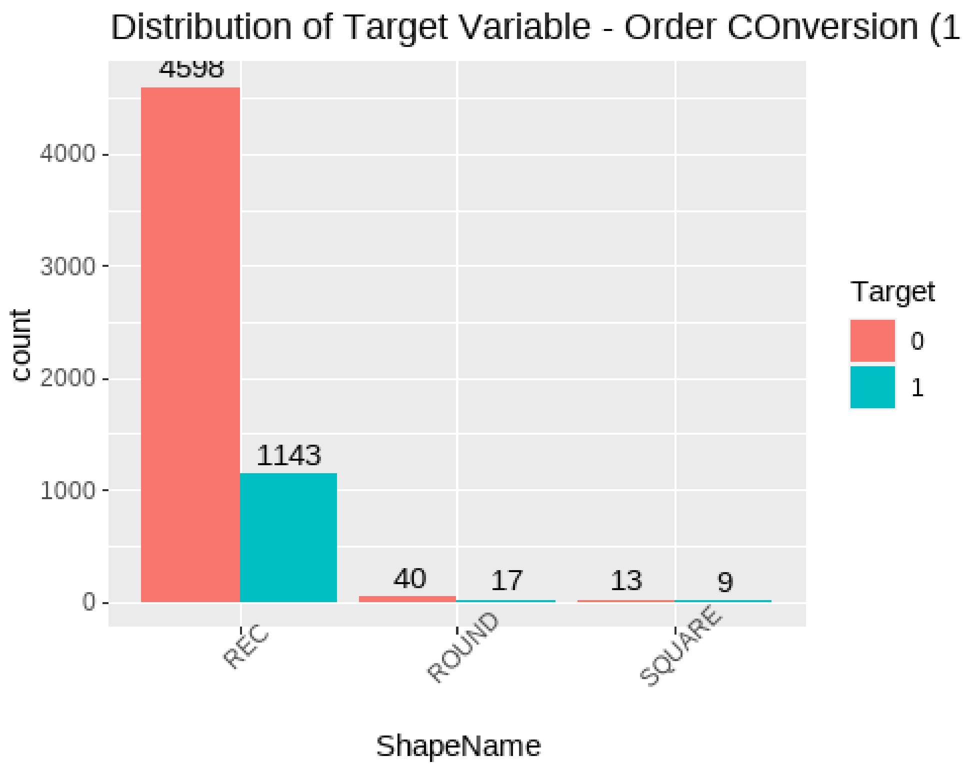
Distribution of Target Variable - Order COnversion (1
4651
1169
4000
3000
2000
1000
0
count
Sample
OrderCategory
Target
0
1


Distribution of Target Variable - Order COnversion (1
count
CustomerCode
Target
0
1
A-9
C-1
CC
H-2
I-2
JL
M-1
N-1
Others
P-5
T-2
T-5
TGT

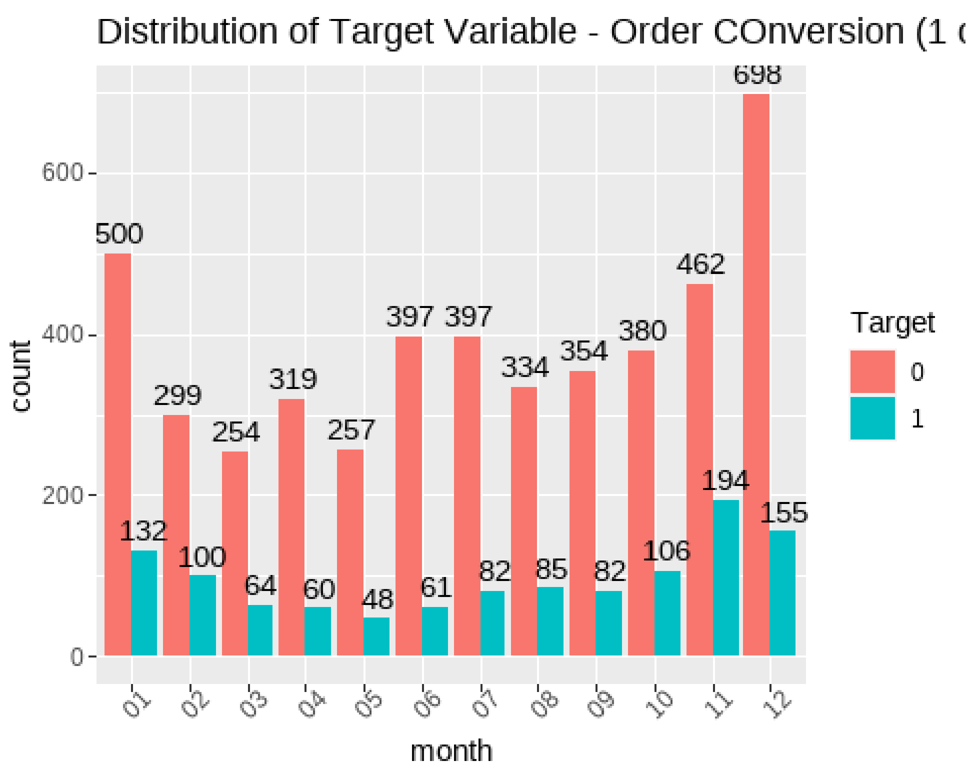

Distribution of Target Variable - Order COnversion (1
count
CountryName
Target
0
1
AUSTRALIA
BELGIUM
BRAZIL
CANADA
CHINA
INDIA
ISRAEL
ITALY
POLAND
ROMANIA
SOUTH AFRICA
UAE
UK
USA


Distribution of Target Variable - Order COnversion (1
3743
895
908
274
3000
2000
1000
0
count
Ft
Mtr
UnitName
Target
0
1

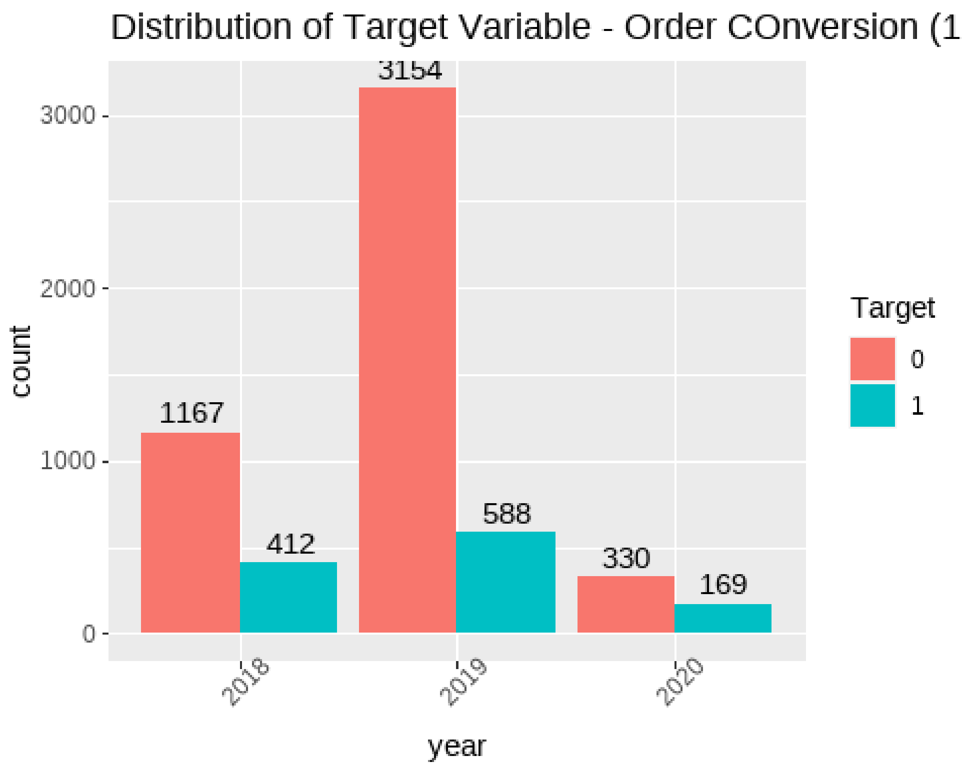

Distribution of Target Variable - Order COnversion (1
1325
479
75
238
20
17
447
78
25
597
108
3
66
18
86
131
9
90
13
17
2000
1500
1000
500
0
count
DOUBLE BACK
DURRY
GUN TUFTED
HAND TUFTED
HANDLOOM
HANDWOVEN
INDO-TIBBETAN
JACQUARD
KNOTTED
POWER LOOM JACQUARD
TABLE TUFTED
ITEM_NAME
Target
0
1



Distribution of Target Variable - Order COnversion (1
4598
4000
3000
2000
1000
0
1143
40
17
13
9
count
REC
ROUND
SQUARE
ShapeName
Target
0
1

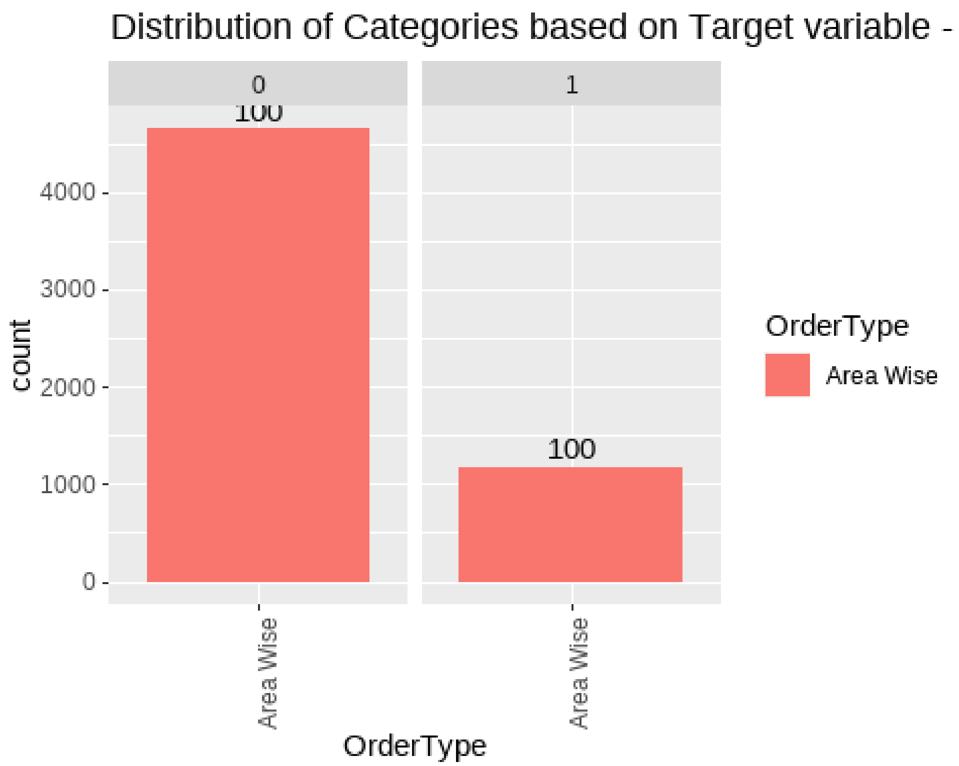


Distribution of Target Variable - Order COnversion (1 (
698
600
400
200
0
count
500
132
299
100
254
64
319
60
257
48
397
61
397
82
334
85
354
82
380
106
462
194
155
01
02
03
04
05
06
07
08
09
10
11
12
month
Target
0
1

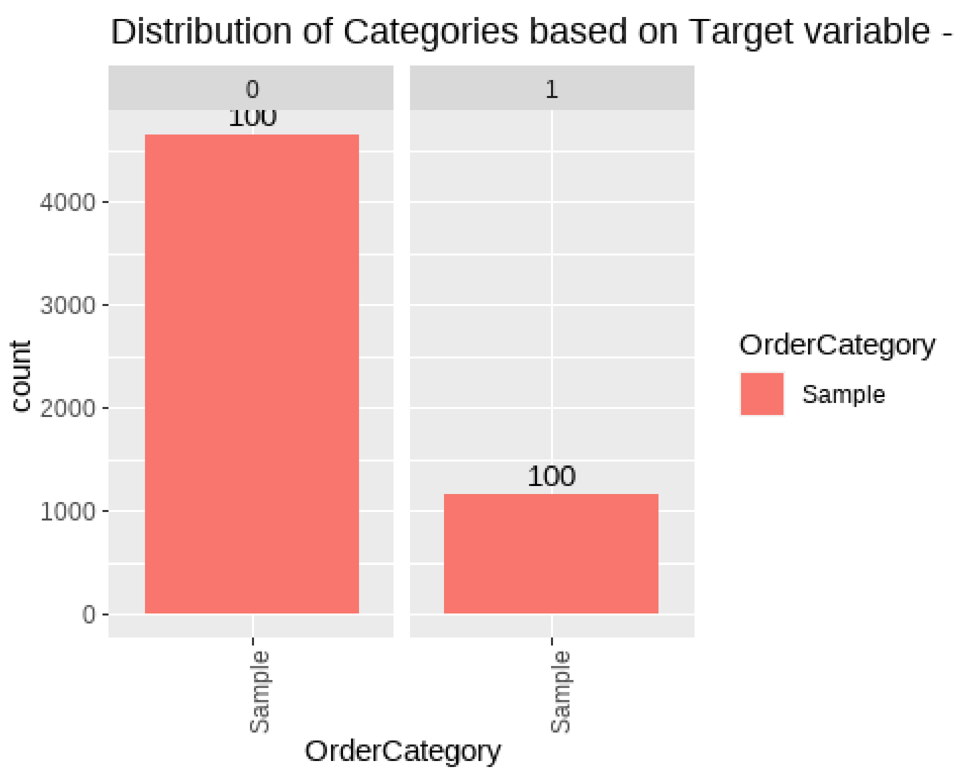


Distribution of input variable categories in each Target category

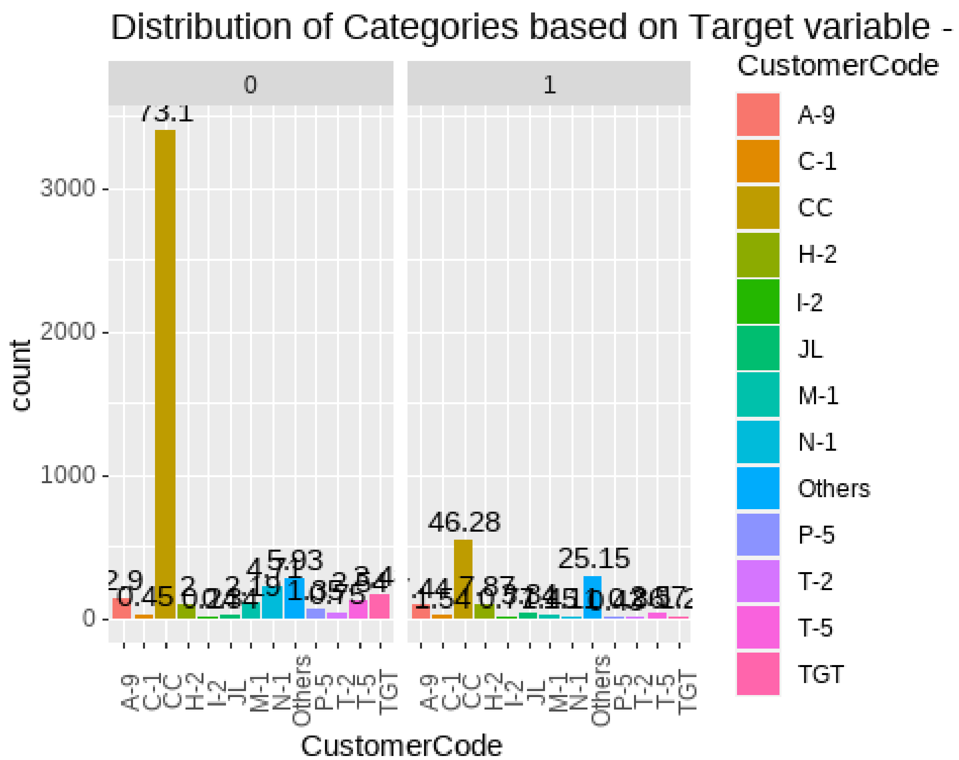
```
for (i in cat){
  if(i!='Target'){
    print(ggplot(dfs,aes_string(x=i, fill=i))+geom_bar(position="dodge")+
    geom_text(aes(label=round(after_stat(prop*100),2), group=1),stat='count',
    size=4,position = position_dodge(0.9), vjust=-0.4) + facet_wrap('Target')
    + ggtitle("Distribution of Categories based on Target variable - Order
Conversion (0 and 1)")+
    theme(axis.text.x = element_text(angle = 90)))
  }
}
```

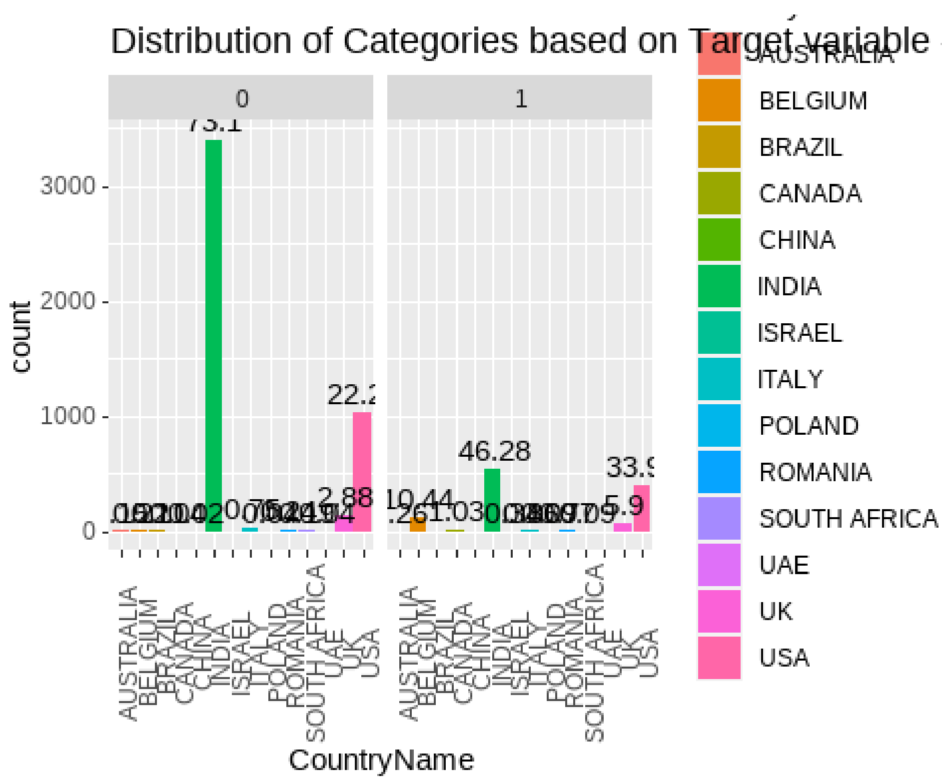


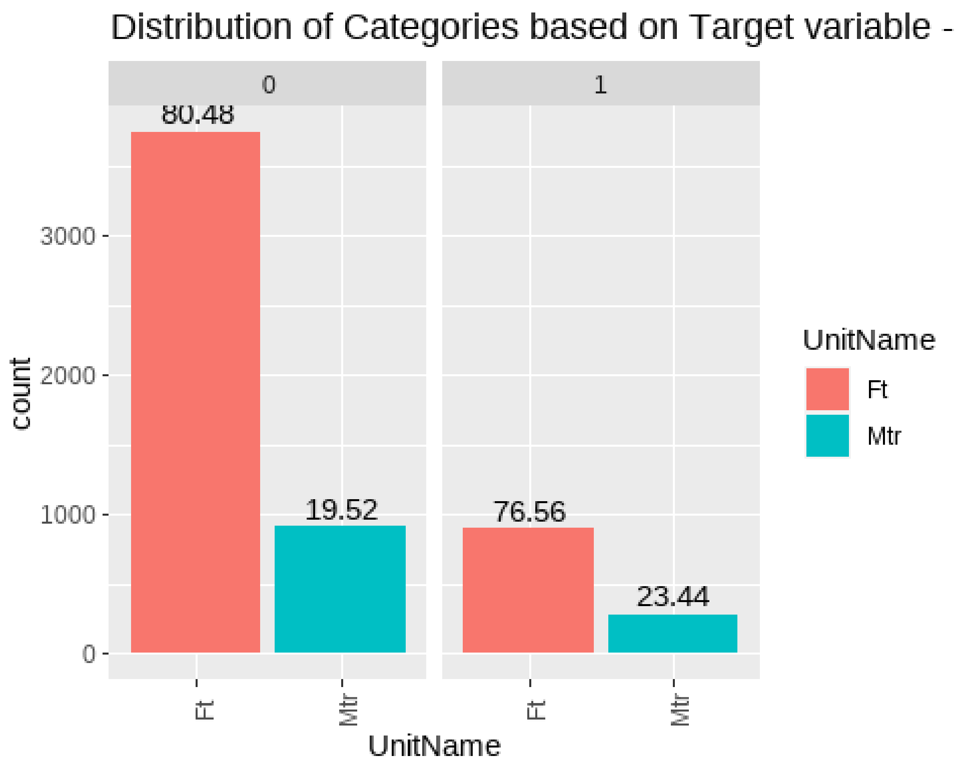
Distribution of Categories based on Target variable -
0
1
100
100
4000
3000
2000
1000
0
count
OrderType
Area Wise
Area Wise
Area Wise
OrderType

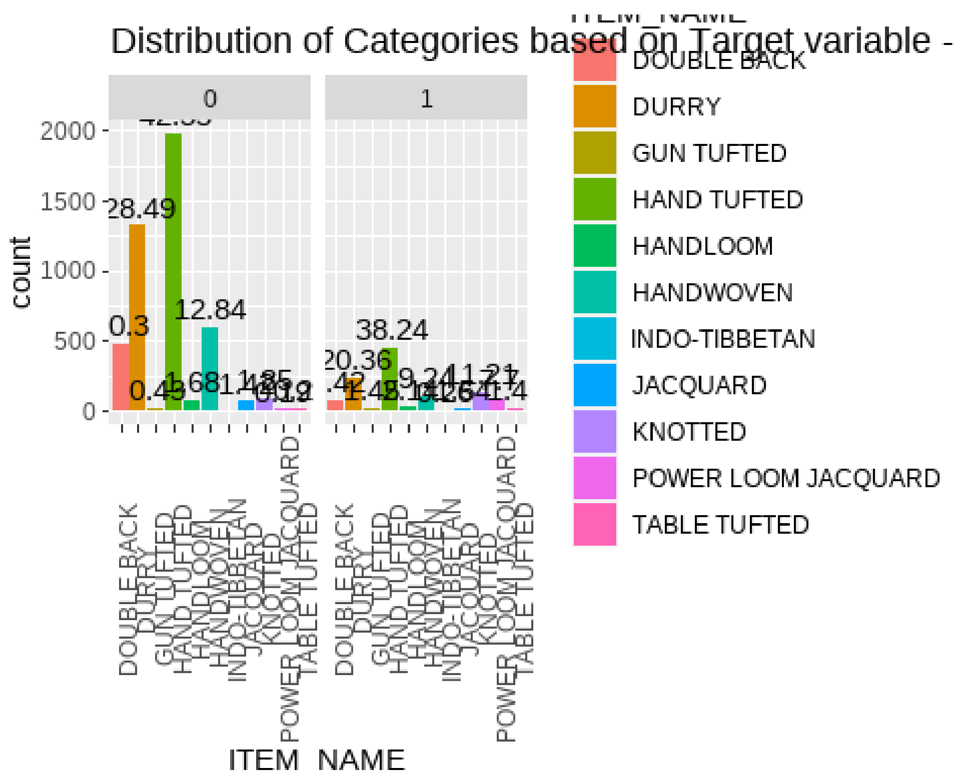


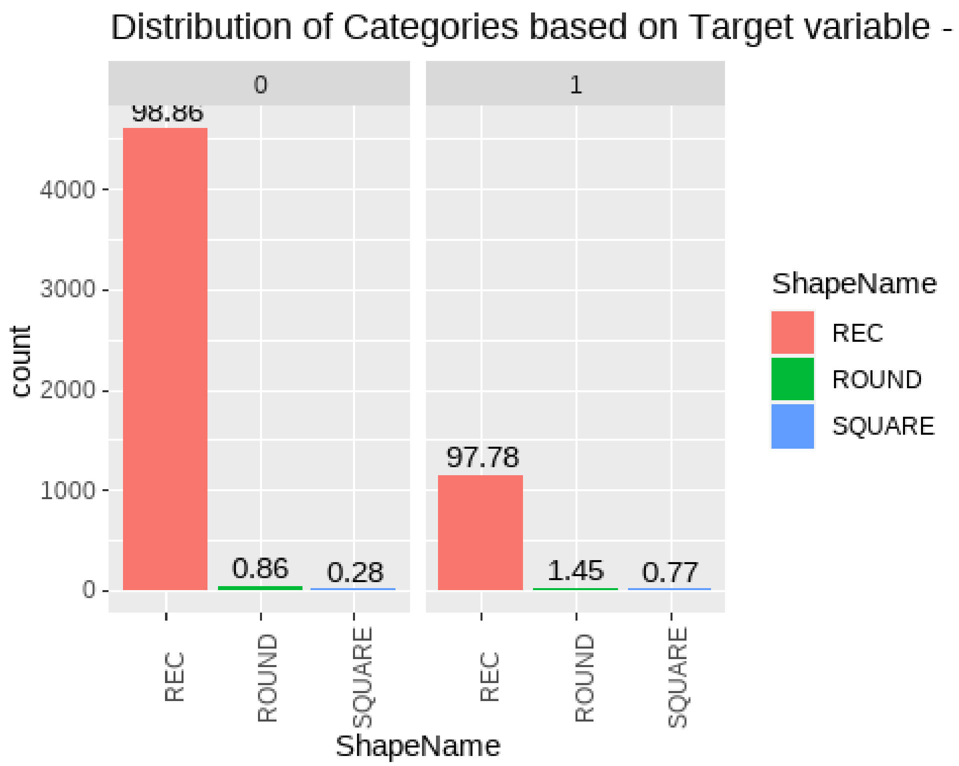
Distribution of Categories based on Target variable -
0
1
100
100
4000
3000
2000
1000
0
count
OrderCategory
Sample
Sample
Sample
OrderCategory

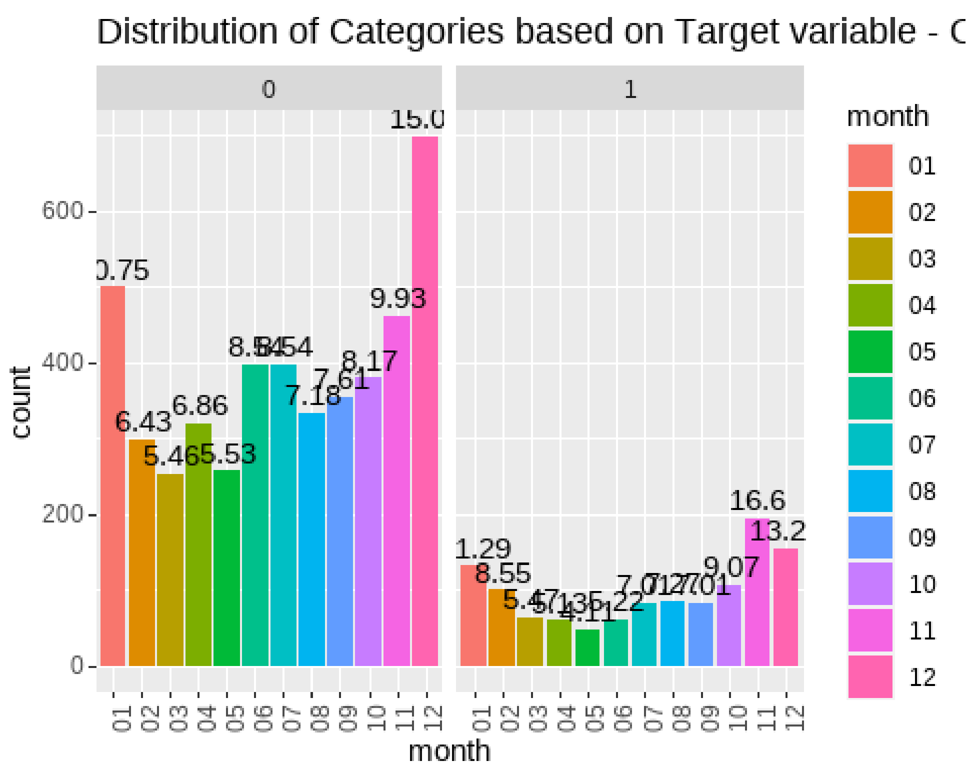

Distribution of Categories based on Target variable -
0
1
CustomerCode
A-9
C-1
CC
H-2
I-2
JL
M-1
N-1
Others
P-5
T-2
T-5
TGT
count
3000
2000
1000
0
73.1
46.28
25.15
CustomerCode

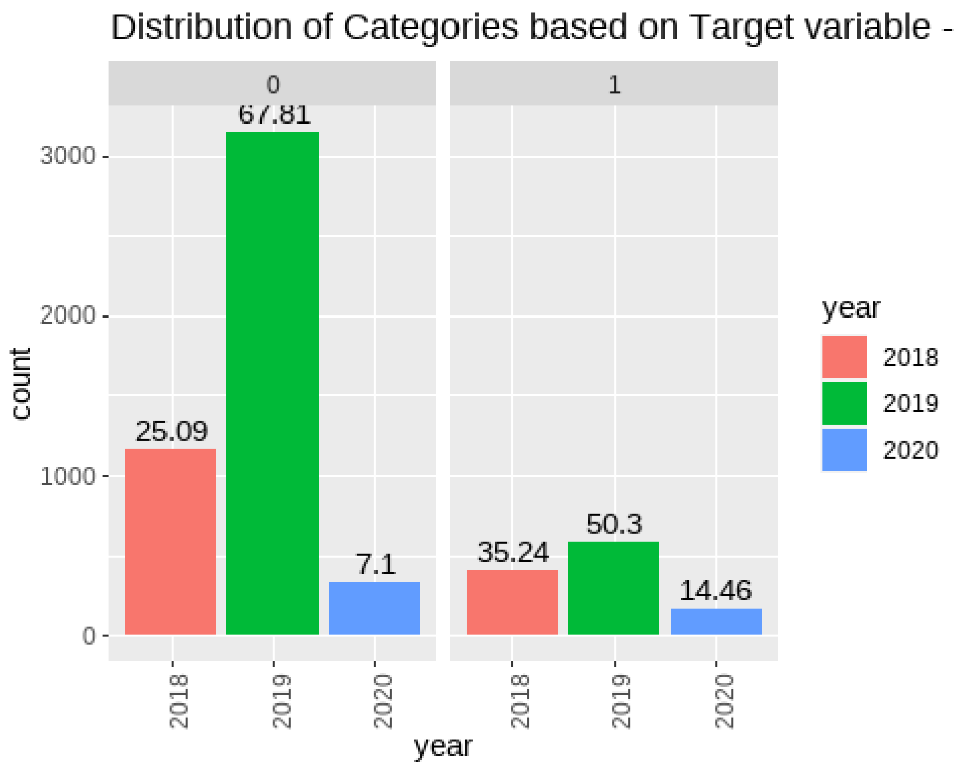

Distribution of Categories based on Target variable -
0
1
AUSTRALIA
BELGIUM
BRAZIL
CANADA
CHINA
INDIA
ISRAEL
ITALY
POLAND
ROMANIA
SOUTH AFRICA
UAE
UK
USA
count
3000
2000
1000
0
73.1
46.28
2.88
5.9
CountryName


Distribution of Categories based on Target variable -
0
1
80.48
19.52
76.56
23.44
3000
2000
1000
0
count
Ft
Mtr
UnitName
UnitName
Ft
Mtr


Distribution of Categories based on Target variable -
0
1
2000
1500
1000
500
0
count
28.49
0.3
12.84
38.24
20.36
ITEM_NAME
DOUBLE BACK
DURRY
GUN TUFTED
HAND TUFTED
HANDLOOM
HANDWOVEN
INDO-TIBBETAN
JACQUARD
KNOTTED
POWER LOOM JACQUARD
TABLE TUFTED


Distribution of Categories based on Target variable -
0
1
count
ShapeName
REC
ROUND
SQUARE
98.86
0.86
0.28
97.78
1.45
0.77

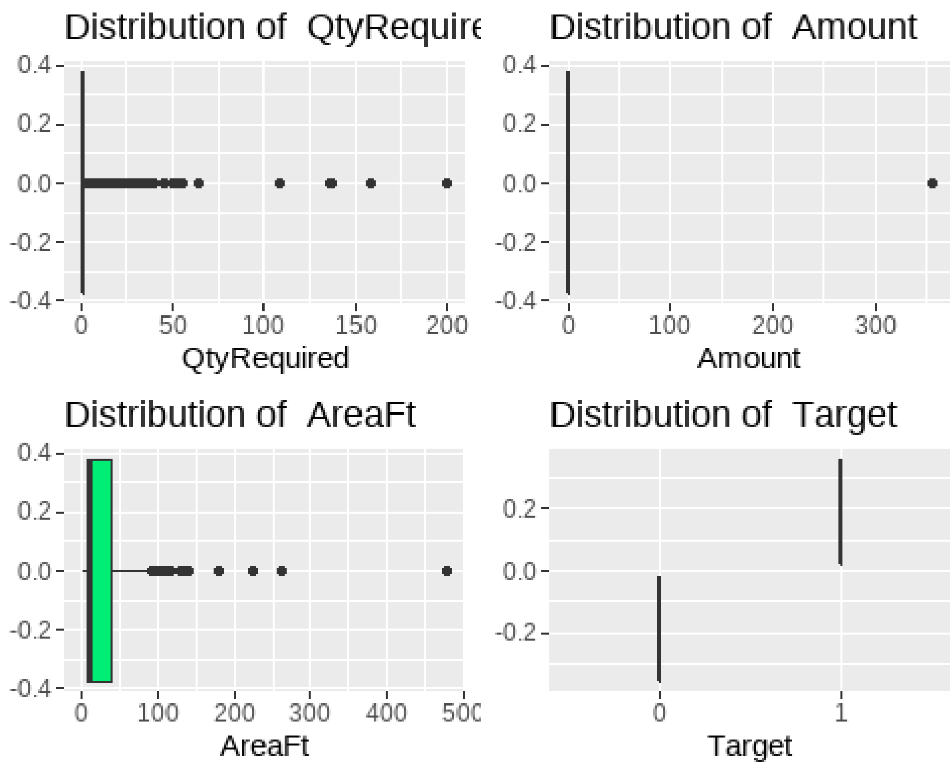

Distribution of Categories based on Target variable - C
0
1
count
month
15.0
9.93
8.17
7.61
7.18
6.86
6.43
5.53
16.6
13.2
1.29
8.55
9.07



The above plot differs from the ones before, since this not only shows the proportion of 1 and 0 in each variables as opposed to counts, but also scales the bar with counts.

1.2.2 EXPLORATION OF NUMERIC VARIABLES

```
num

## [1] "QtyRequired" "Amount"      "AreaFt"      "Target"

plot_list <- list()
n=1
for (i in num){
  plot_list[[n]] <-
ggplot(dfs,aes_string(x=i))+geom_boxplot(fill='springgreen2')+
  ggtitle(paste("Distribution of ",i))
  n=n+1
}
grid.arrange(grobs=plot_list,ncol=2)
```

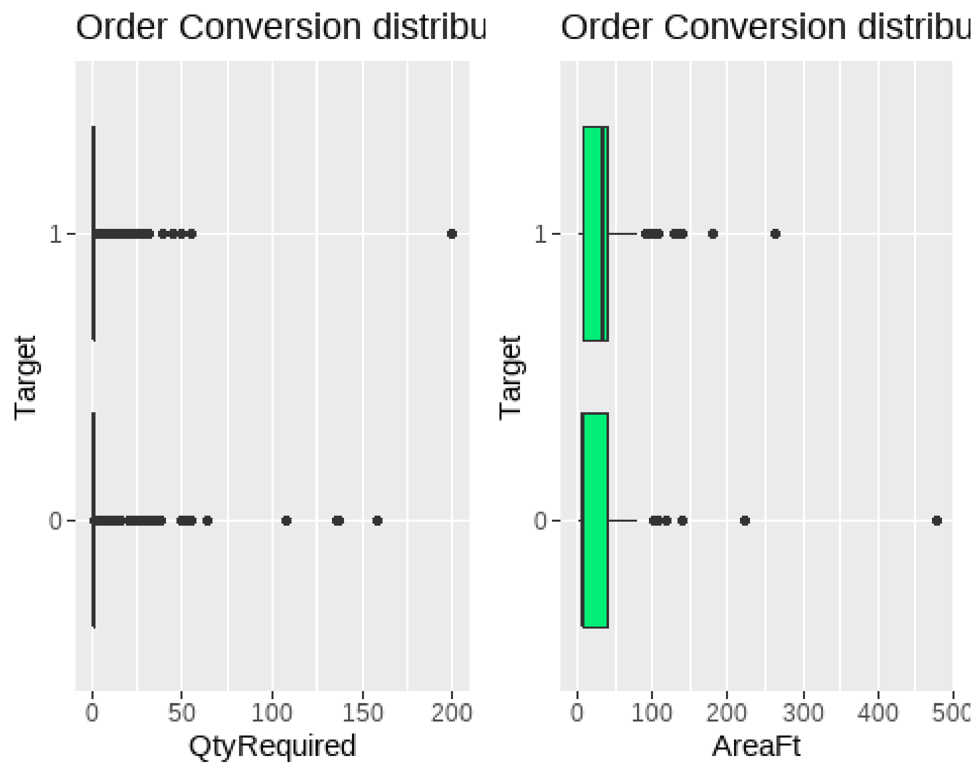


Since Amount is zero for almost all records, we can drop that column.

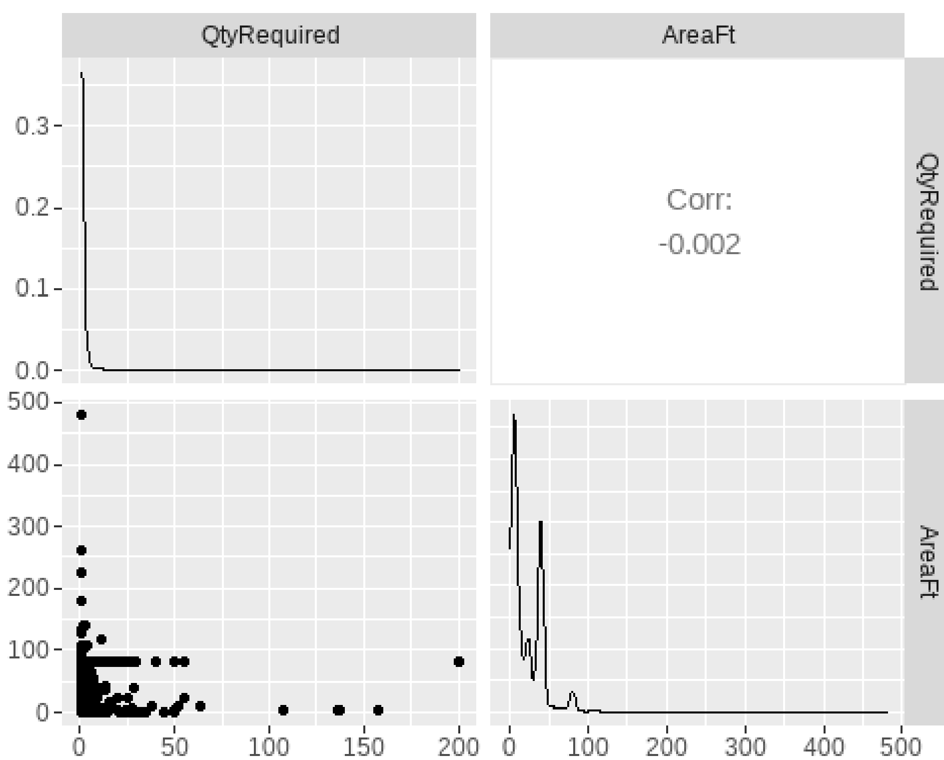
```
drop_col <- c("Amount")
dfs <- select(dfs, -drop_col)
n

## [1] 5

num <- num[-c(2,4)]
num

## [1] "QtyRequired" "AreaFt"
```

Plots to depict the distribution of Order Conversion with respect to numerical variables

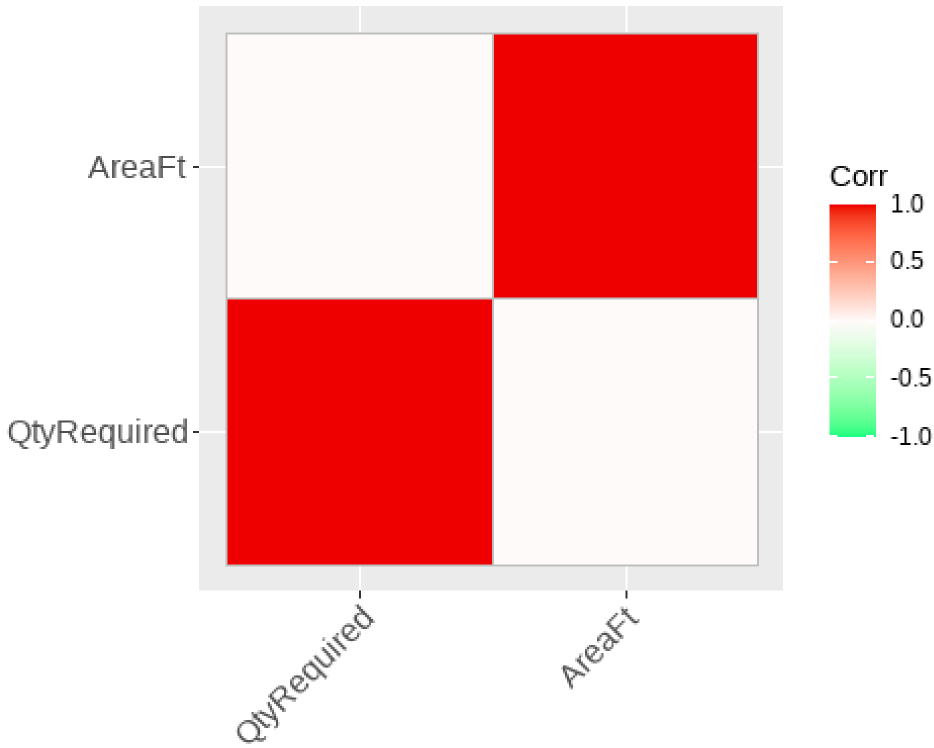
```
plot_list <- list()
n=1
for (i in num){

  plot_list[[n]] <- ggplot(dfs,aes_string(x=i, y="Target",
  group="Target"))+ geom_boxplot(fill="springgreen2")+
  ggtitle(paste("Order Conversion distribution in",i))
  n=n+1

}
grid.arrange(grobs=plot_list,ncol=2)
```




```r
# pair plot for input varibales
ggpairs(dfs[num], upper = list(continuous = wrap("cor", size = 4)))
```



```
ggcorrplot(cor(dfs[num]), ggtheme = 'theme_dark', show.legend = TRUE,
           colors=c('springgreen1','snow1','red2'))
```




```
introduce(dfs)

## # A tibble: 1 x 9
##    rows columns discrete_columns continuous_columns all_missing_columns
##   <int>   <int>            <int>              <int>               <int>
## 1  5820      12               10                  2                   0
## # ... with 4 more variables: total_missing_values <int>, complete_rows
<int>,
## #   total_observations <int>, memory_usage <dbl>
```

1.2 Preparing Dataset for Modelling

1.2.1 Adding features from raw data

```
head(dfm, 5)

## # A tibble: 5 x 25
##   CustomerCode CountryName   USA    UK Italy Belgium Romania Australia
India
##   <chr>        <chr>       <dbl> <dbl> <dbl>   <dbl>   <dbl>     <dbl>
<dbl>
## 1 CC           INDIA           0     0     0       0       0         0
1
## 2 M-1          USA             1     0     0       0       0         0
0
```

```
## 3 M-1          USA             1     0     0       0       0         0
0
## 4 M-1          USA             1     0     0       0       0         0
0
## 5 M-1          USA             1     0     0       0       0         0
0
## # ... with 16 more variables: QtyRequired <dbl>, ITEM_NAME <chr>,
## #   Hand Tufted <dbl>, Durry <dbl>, Double Back <dbl>, Hand Woven <dbl>,
## #   Knotted <dbl>, Jacquard <dbl>, Handloom <dbl>, Other <dbl>,
## #   ShapeName <chr>, REC <dbl>, Round <dbl>, Square <dbl>, AreaFt <dbl>,
## #   Order Conversion <dbl>

dfm$month <- dfs$month
dfm$year <- dfs$year
dfm$CustomerCode <- dfs$CustomerCode

X <- dummy_cols(dfm, select_columns = c("CustomerCode","year","month"),
                remove_first_dummy = TRUE )

drop_col <- c("CountryName","ITEM_NAME","ShapeName","CustomerCode","year",
              "month")
#dropping cols since we already have them encoded

X <- select(X, -drop_col)
X <- rename(X, Target=`Order Conversion`)
X <- rename(X, Hand_Tufted=`Hand Tufted`)
X <- rename(X, Double_Back=`Double Back`)
X <- rename(X, Hand_Woven = `Hand Woven`)
X <- rename(X, CustomerCode_C1 = 'CustomerCode_C-1')
X <- rename(X, CustomerCode_H2 = 'CustomerCode_H-2')
X <- rename(X, CustomerCode_I2 = 'CustomerCode_I-2')
X <- rename(X, CustomerCode_M1 = 'CustomerCode_M-1')
X <- rename(X, CustomerCode_N1 = 'CustomerCode_N-1')
X <- rename(X, CustomerCode_P5 = 'CustomerCode_P-5')
X <- rename(X, CustomerCode_T2 = 'CustomerCode_T-2')
X <- rename(X, CustomerCode_T5 = 'CustomerCode_T-5')
```

2. Analytics and Machine Learning Algorithms Used

- **Classification**
- We chose classification as our target variable is a binary variable
- **Logistic Regression** and **Non Tuned Decision Tree**
- We use these basic models to get a threshold and an idea about the model performance
- **Random Forest** and **Neural Network**

- We chose these as our best performing models and to compare and choose the ideal one for prediction

3. Developing ML models

3.1 PREPROCESSING & TRAIN TEST SPLIT

```
# Scaling Numeric (Non encoded variables)

X$AreaFt <- scale(X$AreaFt)
X$QtyRequired <- scale(X$QtyRequired)

X <- na.omit(X)
X

## # A tibble: 5,781 x 47
##       USA    UK Italy Belgium Romania Australia India QtyRequired[,1]
Hand_Tufted
##     <dbl> <dbl> <dbl>   <dbl>   <dbl>     <dbl> <dbl>           <dbl>
<dbl>
##  1     0     0     0       0       0         0     1          -0.172
1
##  2     1     0     0       0       0         0     0          -0.172
1
##  3     1     0     0       0       0         0     0           0.00441
1
##  4     1     0     0       0       0         0     0          -0.172
1
##  5     1     0     0       0       0         0     0          -0.172
1
##  6     0     0     0       0       0         0     1          -0.172
0
##  7     0     0     0       0       0         0     1          -0.172
0
##  8     1     0     0       0       0         0     0          -0.172
1
##  9     1     0     0       0       0         0     0          -0.172
1
## 10     0     0     0       0       0         0     1          -0.172
1
## # ... with 5,771 more rows, and 38 more variables: Durry <dbl>,
## #   Double_Back <dbl>, Hand_Woven <dbl>, Knotted <dbl>, Jacquard <dbl>,
## #   Handloom <dbl>, Other <dbl>, REC <dbl>, Round <dbl>, Square <dbl>,
## #   AreaFt <dbl[,1]>, Target <dbl>, CustomerCode_C1 <int>,
## #   CustomerCode_CC <int>, CustomerCode_H2 <int>, CustomerCode_I2 <int>,
## #   CustomerCode_JL <int>, CustomerCode_M1 <int>, CustomerCode_N1 <int>,
## #   CustomerCode_Others <int>, CustomerCode_P5 <int>, ...
```

```
set.seed(23)
indx <- sample(2, nrow(X), replace = T, prob = c(0.8, 0.2))
train <- X[indx == 1, ]
test <- X[indx == 2, ]

d_train <- list(dim(train))
d_test <- list(dim(test))
print(paste("Dimension of train data:",d_train))

## [1] "Dimension of train data: c(4621, 47)"

print(paste("Dimension of test data:",d_test))

## [1] "Dimension of test data: c(1160, 47)"
```

3.2 LOGISTIC REGRESSION

```
lr_model <- glm(Target~., data = train)
summary(lr_model)

##
## Call:
## glm(formula = Target ~ ., data = train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.06885  -0.16981  -0.04038   0.05158   1.10218
##
## Coefficients: (8 not defined because of singularities)
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)           0.954759   0.081023  11.784  < 2e-16 ***
## USA                   0.290201   0.024217  11.984  < 2e-16 ***
## UK                    0.087627   0.028612   3.063 0.002207 **
## Italy                 0.073547   0.056699   1.297 0.194647
## Belgium               0.944400   0.042928  22.000  < 2e-16 ***
## Romania               0.510324   0.077144   6.615 4.13e-11 ***
## Australia            -0.076973   0.105449  -0.730 0.465456
## India                       NA         NA      NA       NA
## QtyRequired           0.018427   0.004791   3.846 0.000122 ***
## Hand_Tufted          -0.660183   0.029102 -22.685  < 2e-16 ***
## Durry                -0.634687   0.029384 -21.600  < 2e-16 ***
## Double_Back          -0.676091   0.031771 -21.280  < 2e-16 ***
## Hand_Woven           -0.690416   0.031300 -22.058  < 2e-16 ***
## Knotted              -0.211862   0.037140  -5.704 1.24e-08 ***
## Jacquard             -0.743294   0.048608 -15.292  < 2e-16 ***
## Handloom             -0.660114   0.045116 -14.631  < 2e-16 ***
## Other                       NA         NA      NA       NA
## REC                  -0.218829   0.074887  -2.922 0.003494 **
## Round                -0.193800   0.088686  -2.185 0.028922 *
## Square                      NA         NA      NA       NA
## AreaFt                0.130610   0.004861  26.870  < 2e-16 ***
## CustomerCode_C1       0.004936   0.060030   0.082 0.934476
```

```
## CustomerCode_CC             NA         NA      NA       NA
## CustomerCode_H2      0.033544   0.035287   0.951 0.341855
## CustomerCode_I2             NA         NA      NA       NA
## CustomerCode_JL      0.424135   0.058452   7.256 4.65e-13 ***
## CustomerCode_M1     -0.237434   0.039932  -5.946 2.95e-09 ***
## CustomerCode_N1     -0.266267   0.033558  -7.935 2.64e-15 ***
## CustomerCode_Others -0.051953   0.029448  -1.764 0.077760 .
## CustomerCode_P5     -0.312560   0.048273  -6.475 1.05e-10 ***
## CustomerCode_T2             NA         NA      NA       NA
## CustomerCode_T5             NA         NA      NA       NA
## CustomerCode_TGT    -0.255801   0.035247  -7.257 4.61e-13 ***
## year_2018            0.106108   0.026855   3.951 7.90e-05 ***
## year_2019           -0.001847   0.023626  -0.078 0.937678
## year_2020                   NA         NA      NA       NA
## month_02            -0.056787   0.024158  -2.351 0.018783 *
## month_03            -0.034301   0.027646  -1.241 0.214766
## month_04            -0.030404   0.026825  -1.133 0.257109
## month_05             0.054959   0.028092   1.956 0.050480 .
## month_06             0.008401   0.025627   0.328 0.743056
## month_07             0.011794   0.025458   0.463 0.643186
## month_08             0.034161   0.026244   1.302 0.193098
## month_09             0.005047   0.026480   0.191 0.848840
## month_10            -0.023415   0.026169  -0.895 0.370969
## month_11            -0.011587   0.025007  -0.463 0.643129
## month_12            -0.066349   0.024083  -2.755 0.005893 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.09647039)
##
##     Null deviance: 739.24  on 4620  degrees of freedom
## Residual deviance: 442.03  on 4582  degrees of freedom
## AIC: 2348.4
##
## Number of Fisher Scoring iterations: 2
```

```
lr_pred <- predict(lr_model, newdata = test)
```

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

```
lr_class <- as.factor(ifelse(lr_pred >= 0.5, 1, 0))
```

```
actual <- as.factor(test$Target)
```

```
confusionMatrix(lr_class,actual)
```

```
## Confusion Matrix and Statistics
##
##             Reference
```

```
## Prediction   0   1
##          0 904 105
##          1  29 122
##
##                Accuracy : 0.8845
##                  95% CI : (0.8647, 0.9023)
##     No Information Rate : 0.8043
##     P-Value [Acc > NIR] : 1.697e-13
##
##                   Kappa : 0.5798
##
##  Mcnemar's Test P-Value : 9.232e-11
##
##             Sensitivity : 0.9689
##             Specificity : 0.5374
##          Pos Pred Value : 0.8959
##          Neg Pred Value : 0.8079
##              Prevalence : 0.8043
##          Detection Rate : 0.7793
##    Detection Prevalence : 0.8698
##       Balanced Accuracy : 0.7532
##
##        'Positive' Class : 0
##
```

```
cm_lr <- table(actual,lr_class, dnn = c("Actuals","Predicted"))
cm_lr
```

```
##        Predicted
## Actuals   0   1
##       0 904  29
##       1 105 122
```

```
# function to evaluate model with recall, precision and f-score :
metrics <- function(cm){
  print(paste("Test accuracy :", sum(diag(cm)) / sum(cm)))
  rc <- cm[2,2]/(cm[2,2]+cm[2,1])
  pr <- cm[2,2]/(cm[2,2]+cm[1,2])
  f <- 2*(pr*rc/(pr+rc))
  print(paste("Recall of 1 (Success) :", rc))
  print(paste("Precision of 1 (Success) :", pr))
  print(paste("f score of 1 (Success) :", f))
}

# function to display all performance metrics
full_metrics <- function(cm){
  print(paste("Test accuracy :", sum(diag(cm)) / sum(cm)))
  rc <- cm[2,2]/(cm[2,2]+cm[2,1])
  pr <- cm[2,2]/(cm[2,2]+cm[1,2])
  f <- 2*(pr*rc/(pr+rc))
  sp <- cm[1,1]/(cm[1,1]+cm[1,2])
```

```
  fpr <- cm[1,2]/(cm[1,1]+cm[1,2])
  fnr <- cm[2,1]/(cm[2,2]+cm[2,1])
  print(paste("Recall (Success) :", rc))
  print(paste("Precision (Success) :", pr))
  print(paste("F-score (Success) :", f))
  print(paste("Specificity (tnr) :",sp))
  print(paste("False positve rate:",fpr))
  print(paste("False negative rate:",fnr))
}

print("Performance of Logistisc Regression Model")

## [1] "Performance of Logistisc Regression Model"

metrics(cm_lr)

## [1] "Test accuracy : 0.88448275862069"
## [1] "Recall of 1 (Success) : 0.537444933920705"
## [1] "Precision of 1 (Success) : 0.80794701986755"
## [1] "f score of 1 (Success) : 0.645502645502645"
```

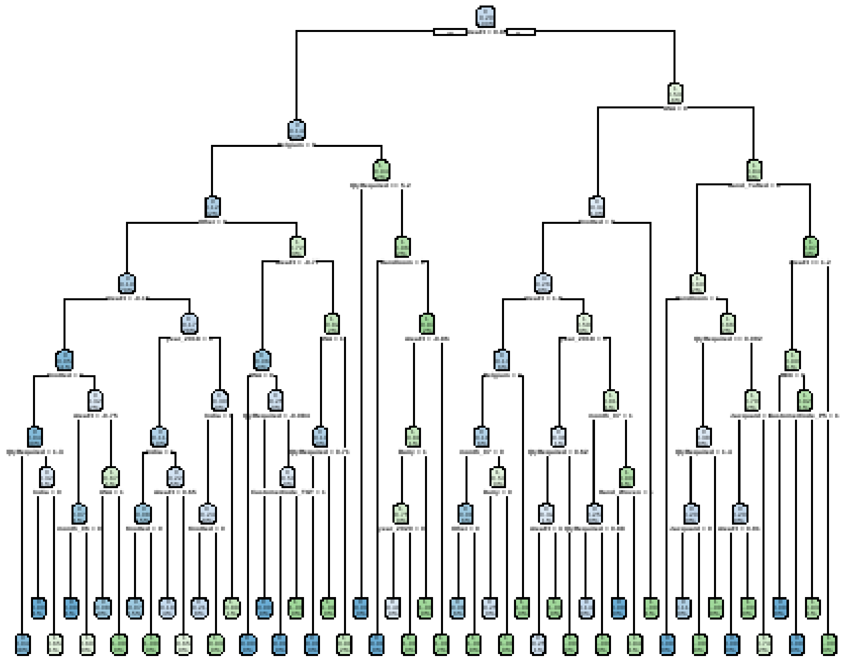
3.3 DECISION TREE MODEL

```
train_tree <- train
test_tree <- test

train_tree$Target <- factor(train_tree$Target)
test_tree$Target <- factor(test_tree$Target)

tree_model <- rpart(Target ~ ., train_tree, parms = list(split =
"information"),
  control = rpart.control(minbucket = 0, minsplit = 0,
  maxdepth = 7, cp =0))
print(tree_model)

## n= 4621
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 4621 924 0 (0.80004328 0.19995672)
##     2) AreaFt< 0.8515963 3884 535 0 (0.86225541 0.13774459)
##       4) Belgium< 0.5 3780 437 0 (0.88439153 0.11560847)
##         8) Other< 0.5 3658 349 0 (0.90459267 0.09540733)
##          16) AreaFt< -0.1585702 2355 124 0 (0.94734607 0.05265393)
##            32) Knotted< 0.5 2278  99 0 (0.95654083 0.04345917)
##              64) QtyRequired< 1.324108 2215  79 0 (0.96433409 0.03566591)
*
##              65) QtyRequired>=1.324108 63  20 0 (0.68253968 0.31746032)
##               130) India< 0.5 32   0 0 (1.00000000 0.00000000) *
##               131) India>=0.5 31  11 1 (0.35483871 0.64516129) *
##            33) Knotted>=0.5 77  25 0 (0.67532468 0.32467532)
```

```
##                66) AreaFt< -0.7451089 44   3 0 (0.93181818 0.06818182)
##                 132) month_06< 0.5 39   0 0 (1.00000000 0.00000000) *
##                 133) month_06>=0.5 5   2 1 (0.40000000 0.60000000) *
##                67) AreaFt>=-0.7451089 33  11 1 (0.33333333 0.66666667)
##                 134) USA>=0.5 12   1 0 (0.91666667 0.08333333) *
##                 135) USA< 0.5 21   0 1 (0.00000000 1.00000000) *
##            17) AreaFt>=-0.1585702 1303 225 0 (0.82732157 0.17267843)
##              34) year_2018< 0.5 899 102 0 (0.88654060 0.11345940)
##                68) India>=0.5 679  53 0 (0.92194404 0.07805596)
##                 136) Knotted< 0.5 674  48 0 (0.92878338 0.07121662) *
##                 137) Knotted>=0.5 5   0 1 (0.00000000 1.00000000) *
##                69) India< 0.5 220  49 0 (0.77727273 0.22272727)
##                 138) AreaFt< 0.6471136 203  38 0 (0.81280788 0.18719212) *
##                 139) AreaFt>=0.6471136 17   6 1 (0.35294118 0.64705882) *
##              35) year_2018>=0.5 404 123 0 (0.69554455 0.30445545)
##                70) India>=0.5 366  89 0 (0.75683060 0.24316940)
##                 140) Knotted< 0.5 351  75 0 (0.78632479 0.21367521) *
##                 141) Knotted>=0.5 15   1 1 (0.06666667 0.93333333) *
##                71) India< 0.5 38   4 1 (0.10526316 0.89473684) *
##           9) Other>=0.5 122  34 1 (0.27868852 0.72131148)
##            18) AreaFt< -0.7683126 16   1 0 (0.93750000 0.06250000)
##              36) USA< 0.5 12   0 0 (1.00000000 0.00000000) *
##              37) USA>=0.5 4   1 0 (0.75000000 0.25000000)
##                74) QtyRequired< -0.08356548 2   0 0 (1.00000000 0.00000000)
*
##                75) QtyRequired>=-0.08356548 2   1 0 (0.50000000 0.50000000)
##                 150) CustomerCode_TGT>=0.5 1   0 0 (1.00000000 0.00000000) *
##                 151) CustomerCode_TGT< 0.5 1   0 1 (0.00000000 1.00000000) *
##            19) AreaFt>=-0.7683126 106  19 1 (0.17924528 0.82075472)
##              38) USA>=0.5 10   1 0 (0.90000000 0.10000000)
##                76) QtyRequired< 0.7082507 9   0 0 (1.00000000 0.00000000) *
##                77) QtyRequired>=0.7082507 1   0 1 (0.00000000 1.00000000) *
##              39) USA< 0.5 96  10 1 (0.10416667 0.89583333) *
##         5) Belgium>=0.5 104   6 1 (0.05769231 0.94230769)
##          10) QtyRequired>=5.195209 2   0 0 (1.00000000 0.00000000) *
##          11) QtyRequired< 5.195209 102   4 1 (0.03921569 0.96078431)
##            22) Handloom>=0.5 1   0 0 (1.00000000 0.00000000) *
##            23) Handloom< 0.5 101   3 1 (0.02970297 0.97029703)
##              46) AreaFt< -0.8611274 28   3 1 (0.10714286 0.89285714)
##                92) Durry>=0.5 12   3 1 (0.25000000 0.75000000)
##                 184) year_2020< 0.5 5   2 0 (0.60000000 0.40000000) *
##                 185) year_2020>=0.5 7   0 1 (0.00000000 1.00000000) *
##                93) Durry< 0.5 16   0 1 (0.00000000 1.00000000) *
##              47) AreaFt>=-0.8611274 73   0 1 (0.00000000 1.00000000) *
##      3) AreaFt>=0.8515963 737 348 1 (0.47218453 0.52781547)
##        6) USA< 0.5 458 155 0 (0.66157205 0.33842795)
##          12) Knotted< 0.5 411 108 0 (0.73722628 0.26277372)
##            24) AreaFt< 1.793602 292  38 0 (0.86986301 0.13013699)
##              48) Belgium< 0.5 287  33 0 (0.88501742 0.11498258)
##                96) month_07< 0.5 268  23 0 (0.91417910 0.08582090)
```

```
##                 192) Other< 0.5 265  20 0 (0.92452830 0.07547170) *
##                 193) Other>=0.5 3   0 1 (0.00000000 1.00000000) *
##                97) month_07>=0.5 19   9 1 (0.47368421 0.52631579)
##                 194) Durry< 0.5 12   3 0 (0.75000000 0.25000000) *
##                 195) Durry>=0.5 7   0 1 (0.00000000 1.00000000) *
##              49) Belgium>=0.5 5   0 1 (0.00000000 1.00000000) *
##            25) AreaFt>=1.793602 119  49 1 (0.41176471 0.58823529)
##              50) year_2018< 0.5 61  23 0 (0.62295082 0.37704918)
##               100) QtyRequired< 0.6202711 56  18 0 (0.67857143 0.32142857)
##                 200) AreaFt< 3.037095 51  13 0 (0.74509804 0.25490196) *
##                 201) AreaFt>=3.037095 5   0 1 (0.00000000 1.00000000) *
##               101) QtyRequired>=0.6202711 5   0 1 (0.00000000 1.00000000) *
##              51) year_2018>=0.5 58  11 1 (0.18965517 0.81034483)
##               102) month_07>=0.5 8   2 0 (0.75000000 0.25000000)
##                 204) QtyRequired< 0.8842099 7   1 0 (0.85714286 0.14285714)
*
##                 205) QtyRequired>=0.8842099 1   0 1 (0.00000000 1.00000000)
*
##               103) month_07< 0.5 50   5 1 (0.10000000 0.90000000)
##                 206) Hand_Woven>=0.5 2   0 0 (1.00000000 0.00000000) *
##                 207) Hand_Woven< 0.5 48   3 1 (0.06250000 0.93750000) *
##          13) Knotted>=0.5 47   0 1 (0.00000000 1.00000000) *
##         7) USA>=0.5 279  45 1 (0.16129032 0.83870968)
##          14) Hand_Tufted< 0.5 98  39 1 (0.39795918 0.60204082)
##            28) Handloom>=0.5 8   0 0 (1.00000000 0.00000000) *
##            29) Handloom< 0.5 90  31 1 (0.34444444 0.65555556)
##              58) QtyRequired>=0.09239367 13   5 0 (0.61538462 0.38461538)
##               116) QtyRequired< 1.412087 10   2 0 (0.80000000 0.20000000)
##                 232) Jacquard< 0.5 9   1 0 (0.88888889 0.11111111) *
##                 233) Jacquard>=0.5 1   0 1 (0.00000000 1.00000000) *
##               117) QtyRequired>=1.412087 3   0 1 (0.00000000 1.00000000) *
##              59) QtyRequired< 0.09239367 77  23 1 (0.29870130 0.70129870)
##               118) Jacquard>=0.5 5   1 0 (0.80000000 0.20000000)
##                 236) AreaFt< 0.9102506 4   0 0 (1.00000000 0.00000000) *
##                 237) AreaFt>=0.9102506 1   0 1 (0.00000000 1.00000000) *
##               119) Jacquard< 0.5 72  19 1 (0.26388889 0.73611111) *
##          15) Hand_Tufted>=0.5 181   6 1 (0.03314917 0.96685083)
##            30) AreaFt>=1.180799 60   6 1 (0.10000000 0.90000000)
##              60) REC< 0.5 1   0 0 (1.00000000 0.00000000) *
##              61) REC>=0.5 59   5 1 (0.08474576 0.91525424)
##               122) CustomerCode_P5>=0.5 1   0 0 (1.00000000 0.00000000) *
##               123) CustomerCode_P5< 0.5 58   4 1 (0.06896552 0.93103448) *
##            31) AreaFt< 1.180799 121   0 1 (0.00000000 1.00000000) *
```

```
rpart.plot(tree_model)
```



```
pred_tree <- predict(tree_model, test_tree, type = "class")

cm_tree <- table(actual, pred_tree)
cm_tree

##         pred_tree
## actual    0   1
##       0 903  30
##       1  71 156

metrics(cm_tree)

## [1] "Test accuracy : 0.912931034482759"
## [1] "Recall of 1 (Success) : 0.687224669603524"
## [1] "Precision of 1 (Success) : 0.838709677419355"
## [1] "f score of 1 (Success) : 0.75544794188862"
```

3.3.1 Tuning Decision Tree

Pre-Pruning

```
bucket <- c(5,10,15)
split <- c(5,10,15)
for (i in bucket){
  for (j in split){
    print(paste("For bucket =",i,"and split =",j))
    tree_model2 <- rpart(Target ~ ., train_tree, parms = list(split =
"information"),
```

```
    control = rpart.control(minbucket = i, minsplit = j, maxdepth = 10, cp
=0))
    pred_test <- predict(tree_model2, test, type = "class")
    cm_test <- table(test$Target, pred_test)
    metrics(cm_test)
    writeLines("\n\n")
  }
}

## [1] "For bucket = 5 and split = 5"
## [1] "Test accuracy : 0.912068965517241"
## [1] "Recall of 1 (Success) : 0.700440528634361"
## [1] "Precision of 1 (Success) : 0.823834196891192"
## [1] "f score of 1 (Success) : 0.757142857142857"
##
##
##
## [1] "For bucket = 5 and split = 10"
## [1] "Test accuracy : 0.912068965517241"
## [1] "Recall of 1 (Success) : 0.700440528634361"
## [1] "Precision of 1 (Success) : 0.823834196891192"
## [1] "f score of 1 (Success) : 0.757142857142857"
##
##
##
## [1] "For bucket = 5 and split = 15"
## [1] "Test accuracy : 0.906896551724138"
## [1] "Recall of 1 (Success) : 0.704845814977974"
## [1] "Precision of 1 (Success) : 0.796019900497512"
## [1] "f score of 1 (Success) : 0.747663551401869"
##
##
##
## [1] "For bucket = 10 and split = 5"
## [1] "Test accuracy : 0.9"
## [1] "Recall of 1 (Success) : 0.687224669603524"
## [1] "Precision of 1 (Success) : 0.776119402985075"
## [1] "f score of 1 (Success) : 0.728971962616823"
##
##
##
## [1] "For bucket = 10 and split = 10"
## [1] "Test accuracy : 0.9"
## [1] "Recall of 1 (Success) : 0.687224669603524"
## [1] "Precision of 1 (Success) : 0.776119402985075"
## [1] "f score of 1 (Success) : 0.728971962616823"
##
##
##
## [1] "For bucket = 10 and split = 15"
```

```
## [1] "Test accuracy : 0.9"
## [1] "Recall of 1 (Success) : 0.687224669603524"
## [1] "Precision of 1 (Success) : 0.776119402985075"
## [1] "f score of 1 (Success) : 0.72897196261682З"
##
##
##
## [1] "For bucket = 15 and split = 5"
## [1] "Test accuracy : 0.897413793103448"
## [1] "Recall of 1 (Success) : 0.691629955947137"
## [1] "Precision of 1 (Success) : 0.762135922330097"
## [1] "f score of 1 (Success) : 0.725173210161663"
##
##
##
## [1] "For bucket = 15 and split = 10"
## [1] "Test accuracy : 0.897413793103448"
## [1] "Recall of 1 (Success) : 0.691629955947137"
## [1] "Precision of 1 (Success) : 0.762135922330097"
## [1] "f score of 1 (Success) : 0.725173210161663"
##
##
##
## [1] "For bucket = 15 and split = 15"
## [1] "Test accuracy : 0.897413793103448"
## [1] "Recall of 1 (Success) : 0.691629955947137"
## [1] "Precision of 1 (Success) : 0.762135922330097"
## [1] "f score of 1 (Success) : 0.725173210161663"
```

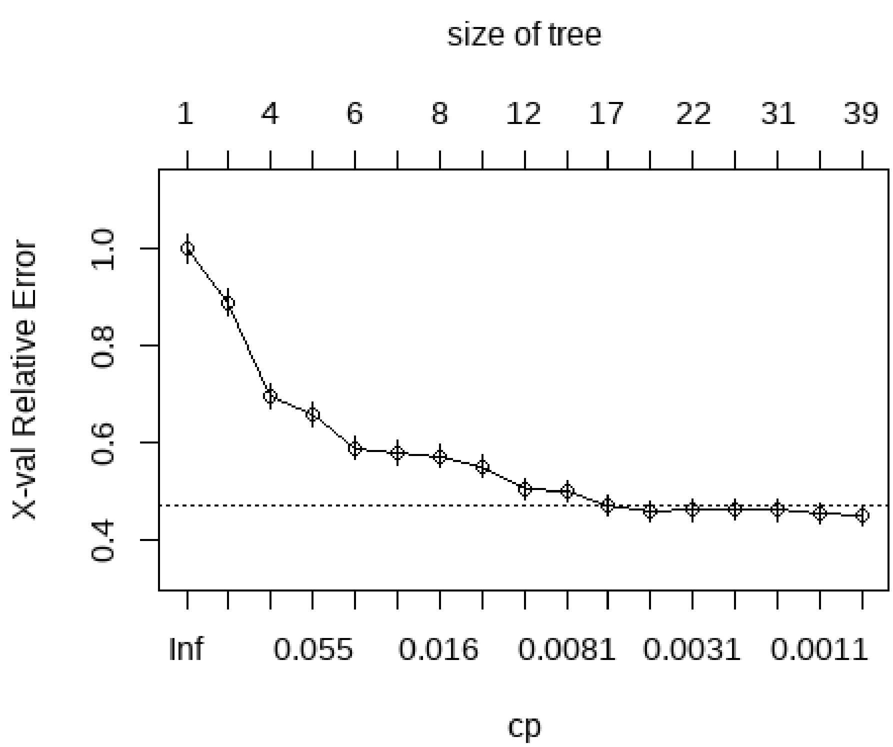
Best params for minbucket and minsplit are both 5.

Post-Pruning

```
# Determining best cp value for best minbucket and minsplit
tree_model_tune <- rpart(Target ~ ., train_tree, parms = list(split =
"gini"),
    control = rpart.control(minbucket = 10, minsplit = 10, cp=0))
printcp(tree_model_tune)
```

```
##
## Classification tree:
## rpart(formula = Target ~ ., data = train_tree, parms = list(split =
"gini"),
##     control = rpart.control(minbucket = 10, minsplit = 10, cp = 0))
##
## Variables actually used in tree construction:
##  [1] AreaFt              Belgium             CustomerCode_JL
##  [4] CustomerCode_Others CustomerCode_T5     Double_Back
##  [7] Durry               Hand_Tufted         India
## [10] Knotted             month_07            month_11
## [13] month_12            Other               QtyRequired
```

```
## [16] USA                  year_2018
##
## Root node error: 924/4621 = 0.19996
##
## n= 4621
##
##            CP nsplit rel error  xerror     xstd
## 1  0.1022727      0   1.00000 1.00000 0.029425
## 2  0.0995671      2   0.79545 0.88961 0.028134
## 3  0.0584416      3   0.69589 0.69589 0.025462
## 4  0.0508658      4   0.63745 0.65801 0.024868
## 5  0.0227273      5   0.58658 0.58983 0.023729
## 6  0.0162338      6   0.56385 0.58009 0.023558
## 7  0.0151515      7   0.54762 0.57251 0.023424
## 8  0.0146104      8   0.53247 0.55195 0.023053
## 9  0.0086580     11   0.48593 0.50433 0.022153
## 10 0.0075758     12   0.47727 0.50000 0.022069
## 11 0.0048701     16   0.44589 0.47186 0.021506
## 12 0.0036075     18   0.43615 0.45996 0.021260
## 13 0.0027056     21   0.42532 0.46104 0.021283
## 14 0.0021645     25   0.41450 0.46429 0.021350
## 15 0.0014430     30   0.40368 0.46104 0.021283
## 16 0.0008658     33   0.39935 0.45346 0.021125
## 17 0.0000000     38   0.39502 0.45130 0.021079

plotcp(tree_model_tune)
```



Best cp = 0.0007

```
cp <- 0.0007
prunedTree <- tree_model_tune <- rpart(Target ~ ., train_tree, parms =
list(split = "information"),
  control = rpart.control(minbucket = 5, minsplit = 5, cp=cp))
print(prunedTree)
```

```
## n= 4621
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##     1) root 4621 924 0 (0.80004328 0.19995672)
##       2) AreaFt< 0.8515963 3884 535 0 (0.86225541 0.13774459)
##         4) Belgium< 0.5 3780 437 0 (0.88439153 0.11560847)
##           8) Other< 0.5 3658 349 0 (0.90459267 0.09540733)
##            16) AreaFt< -0.1585702 2355 124 0 (0.94734607 0.05265393)
##              32) Knotted< 0.5 2278  99 0 (0.95654083 0.04345917)
##                64) QtyRequired< 1.324108 2215  79 0 (0.96433409
0.03566591)
##                 128) India>=0.5 1498  19 0 (0.98731642 0.01268358) *
##                 129) India< 0.5 717  60 0 (0.91631799 0.08368201)
##                   258) month_11< 0.5 637  42 0 (0.93406593 0.06593407)
##                     516) USA>=0.5 560  26 0 (0.95357143 0.04642857) *
##                     517) USA< 0.5 77  16 0 (0.79220779 0.20779221)
##                      1034) Hand_Woven>=0.5 20   0 0 (1.00000000
0.00000000) *
##                      1035) Hand_Woven< 0.5 57  16 0 (0.71929825
0.28070175)
##                        2070) AreaFt< -0.8430796 18   0 0 (1.00000000
0.00000000) *
##                        2071) AreaFt>=-0.8430796 39  16 0 (0.58974359
0.41025641)
##                          4142) Romania< 0.5 34  11 0 (0.67647059
0.32352941)
##                            8284) Hand_Tufted>=0.5 18   2 0 (0.88888889
0.11111111) *
##                            8285) Hand_Tufted< 0.5 16   7 1 (0.43750000
0.56250000)
##                             16570) year_2018>=0.5 5   1 0 (0.80000000
0.20000000) *
##                             16571) year_2018< 0.5 11   3 1 (0.27272727
0.72727273) *
##                          4143) Romania>=0.5 5   0 1 (0.00000000
1.00000000) *
##                   259) month_11>=0.5 80  18 0 (0.77500000 0.22500000)
##                     518) CustomerCode_Others< 0.5 58   8 0 (0.86206897
0.13793103) *
##                     519) CustomerCode_Others>=0.5 22  10 0 (0.54545455
```

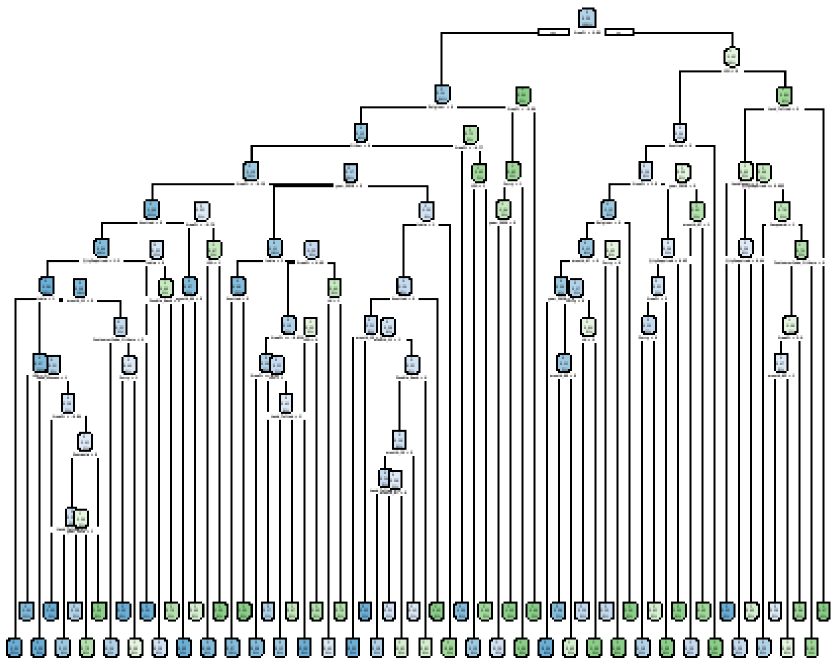
```
0.45454545)
##                        1038) Durry>=0.5 5   0 0 (1.00000000 0.00000000) *
##                        1039) Durry< 0.5 17   7 1 (0.41176471 0.58823529) *
##                  65) QtyRequired>=1.324108 63  20 0 (0.68253968 0.31746032)
##                   130) India< 0.5 32   0 0 (1.00000000 0.00000000) *
##                   131) India>=0.5 31  11 1 (0.35483871 0.64516129)
##                     262) Double_Back>=0.5 7   2 0 (0.71428571 0.28571429) *
##                     263) Double_Back< 0.5 24   6 1 (0.25000000 0.75000000) *
##                33) Knotted>=0.5 77  25 0 (0.67532468 0.32467532)
##                  66) AreaFt< -0.7451089 44   3 0 (0.93181818 0.06818182)
##                   132) month_06< 0.5 39   0 0 (1.00000000 0.00000000) *
##                   133) month_06>=0.5 5   2 1 (0.40000000 0.60000000) *
##                  67) AreaFt>=-0.7451089 33  11 1 (0.33333333 0.66666667)
##                   134) USA>=0.5 12   1 0 (0.91666667 0.08333333) *
##                   135) USA< 0.5 21   0 1 (0.00000000 1.00000000) *
##              17) AreaFt>=-0.1585702 1303 225 0 (0.82732157 0.17267843)
##                34) year_2018< 0.5 899 102 0 (0.88654060 0.11345940)
##                  68) India>=0.5 679  53 0 (0.92194404 0.07805596)
##                   136) Knotted< 0.5 674  48 0 (0.92878338 0.07121662) *
##                   137) Knotted>=0.5 5   0 1 (0.00000000 1.00000000) *
##                  69) India< 0.5 220  49 0 (0.77727273 0.22272727)
##                   138) AreaFt< 0.6471136 203  38 0 (0.81280788 0.18719212)
##                     276) AreaFt>=-0.01419208 181  26 0 (0.85635359
0.14364641)
##                       552) AreaFt>=0.5072463 99   8 0 (0.91919192
0.08080808) *
##                       553) AreaFt< 0.5072463 82  18 0 (0.78048780
0.21951220)
##                        1106) USA>=0.5 65  11 0 (0.83076923 0.16923077) *
##                        1107) USA< 0.5 17   7 0 (0.58823529 0.41176471)
##                          2214) Hand_Tufted>=0.5 8   1 0 (0.87500000
0.12500000) *
##                          2215) Hand_Tufted< 0.5 9   3 1 (0.33333333
0.66666667) *
##                     277) AreaFt< -0.01419208 22  10 1 (0.45454545
0.54545455)
##                       554) USA< 0.5 7   0 0 (1.00000000 0.00000000) *
##                       555) USA>=0.5 15   3 1 (0.20000000 0.80000000) *
##                   139) AreaFt>=0.6471136 17   6 1 (0.35294118 0.64705882)
##                     278) UK>=0.5 5   2 0 (0.60000000 0.40000000) *
##                     279) UK< 0.5 12   3 1 (0.25000000 0.75000000) *
##                35) year_2018>=0.5 404 123 0 (0.69554455 0.30445545)
##                  70) India>=0.5 366  89 0 (0.75683060 0.24316940)
##                   140) Knotted< 0.5 351  75 0 (0.78632479 0.21367521)
##                     280) month_12>=0.5 91   2 0 (0.97802198 0.02197802) *
##                     281) month_12< 0.5 260  73 0 (0.71923077 0.28076923)
##                       562) month_11>=0.5 33   1 0 (0.96969697 0.03030303) *
##                       563) month_11< 0.5 227  72 0 (0.68281938 0.31718062)
##                        1126) Double_Back< 0.5 209  61 0 (0.70813397
0.29186603)
```

```
##                          2252) month_10< 0.5 136  32 0 (0.76470588
0.23529412)
##                            4504) Hand_Tufted>=0.5 99  18 0 (0.81818182
0.18181818) *
##                            4505) Hand_Tufted< 0.5 37  14 0 (0.62162162
0.37837838)
##                              9010) month_07< 0.5 29   9 0 (0.68965517
0.31034483) *
##                              9011) month_07>=0.5 8   3 1 (0.37500000
0.62500000) *
##                          2253) month_10>=0.5 73  29 0 (0.60273973
0.39726027) *
##                        1127) Double_Back>=0.5 18   7 1 (0.38888889
0.61111111) *
##                  141) Knotted>=0.5 15   1 1 (0.06666667 0.93333333) *
##                 71) India< 0.5 38   4 1 (0.10526316 0.89473684) *
##            9) Other>=0.5 122  34 1 (0.27868852 0.72131148)
##             18) AreaFt< -0.7683126 16   1 0 (0.93750000 0.06250000) *
##             19) AreaFt>=-0.7683126 106  19 1 (0.17924528 0.82075472)
##               38) USA>=0.5 10   1 0 (0.90000000 0.10000000) *
##               39) USA< 0.5 96  10 1 (0.10416667 0.89583333) *
##         5) Belgium>=0.5 104   6 1 (0.05769231 0.94230769)
##          10) AreaFt< -0.8611274 30   5 1 (0.16666667 0.83333333)
##            20) Durry>=0.5 14   5 1 (0.35714286 0.64285714)
##              40) year_2020< 0.5 7   2 0 (0.71428571 0.28571429) *
##              41) year_2020>=0.5 7   0 1 (0.00000000 1.00000000) *
##            21) Durry< 0.5 16   0 1 (0.00000000 1.00000000) *
##          11) AreaFt>=-0.8611274 74   1 1 (0.01351351 0.98648649) *
##       3) AreaFt>=0.8515963 737 348 1 (0.47218453 0.52781547)
##         6) USA< 0.5 458 155 0 (0.66157205 0.33842795)
##          12) Knotted< 0.5 411 108 0 (0.73722628 0.26277372)
##            24) AreaFt< 1.793602 292  38 0 (0.86986301 0.13013699)
##              48) Belgium< 0.5 287  33 0 (0.88501742 0.11498258)
##                96) month_07< 0.5 268  23 0 (0.91417910 0.08582090)
##                 192) year_2018< 0.5 166   6 0 (0.96385542 0.03614458) *
##                 193) year_2018>=0.5 102  17 0 (0.83333333 0.16666667)
##                   386) Durry< 0.5 89  10 0 (0.88764045 0.11235955)
##                     772) month_05< 0.5 84   7 0 (0.91666667 0.08333333) *
##                     773) month_05>=0.5 5   2 1 (0.40000000 0.60000000) *
##                   387) Durry>=0.5 13   6 1 (0.46153846 0.53846154)
##                     774) UK< 0.5 8   2 0 (0.75000000 0.25000000) *
##                     775) UK>=0.5 5   0 1 (0.00000000 1.00000000) *
##                97) month_07>=0.5 19   9 1 (0.47368421 0.52631579)
##                 194) Durry< 0.5 12   3 0 (0.75000000 0.25000000) *
##                 195) Durry>=0.5 7   0 1 (0.00000000 1.00000000) *
##              49) Belgium>=0.5 5   0 1 (0.00000000 1.00000000) *
##            25) AreaFt>=1.793602 119  49 1 (0.41176471 0.58823529)
##              50) year_2018< 0.5 61  23 0 (0.62295082 0.37704918)
##               100) QtyRequired< 0.6202711 56  18 0 (0.67857143 0.32142857)
##                 200) AreaFt< 3.037095 51  13 0 (0.74509804 0.25490196)
```

```
##                     400) Durry< 0.5 43   8 0 (0.81395349 0.18604651) *
##                     401) Durry>=0.5 8   3 1 (0.37500000 0.62500000) *
##                   201) AreaFt>=3.037095 5   0 1 (0.00000000 1.00000000) *
##                 101) QtyRequired>=0.6202711 5   0 1 (0.00000000 1.00000000)
*
##                51) year_2018>=0.5 58  11 1 (0.18965517 0.81034483)
##                 102) month_07>=0.5 8   2 0 (0.75000000 0.25000000) *
##                 103) month_07< 0.5 50   5 1 (0.10000000 0.90000000) *
##            13) Knotted>=0.5 47   0 1 (0.00000000 1.00000000) *
##           7) USA>=0.5 279  45 1 (0.16129032 0.83870968)
##            14) Hand_Tufted< 0.5 98  39 1 (0.39795918 0.60204082)
##              28) Handloom>=0.5 8   0 0 (1.00000000 0.00000000) *
##              29) Handloom< 0.5 90  31 1 (0.34444444 0.65555556)
##                58) QtyRequired>=0.09239367 13   5 0 (0.61538462 0.38461538)
##                 116) QtyRequired< 0.444312 8   2 0 (0.75000000 0.25000000) *
##                 117) QtyRequired>=0.444312 5   2 1 (0.40000000 0.60000000) *
##                59) QtyRequired< 0.09239367 77  23 1 (0.29870130 0.70129870)
##                 118) Jacquard>=0.5 5   1 0 (0.80000000 0.20000000) *
##                 119) Jacquard< 0.5 72  19 1 (0.26388889 0.73611111)
##                   238) CustomerCode_Others< 0.5 24  10 1 (0.41666667
0.58333333)
##                     476) AreaFt< 2.643278 17   8 0 (0.52941176 0.47058824)
##                       952) month_02>=0.5 8   3 0 (0.62500000 0.37500000) *
##                       953) month_02< 0.5 9   4 1 (0.44444444 0.55555556) *
##                     477) AreaFt>=2.643278 7   1 1 (0.14285714 0.85714286) *
##                   239) CustomerCode_Others>=0.5 48   9 1 (0.18750000
0.81250000) *
##            15) Hand_Tufted>=0.5 181   6 1 (0.03314917 0.96685083) *
```

```
rpart.plot(prunedTree)
```

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



```r
# Checking classification metrics after post pruning
pred_test_prune <- predict(prunedTree, test_tree, type = "class")
cm_test_prune <- table(actual, pred_test_prune)
metrics(cm_test_prune)

## [1] "Test accuracy : 0.918103448275862"
## [1] "Recall of 1 (Success) : 0.731277533039648"
## [1] "Precision of 1 (Success) : 0.83"
## [1] "f score of 1 (Success) : 0.77751756440281"
```

While not much, there still seems to be a little improvement after pre and post pruning

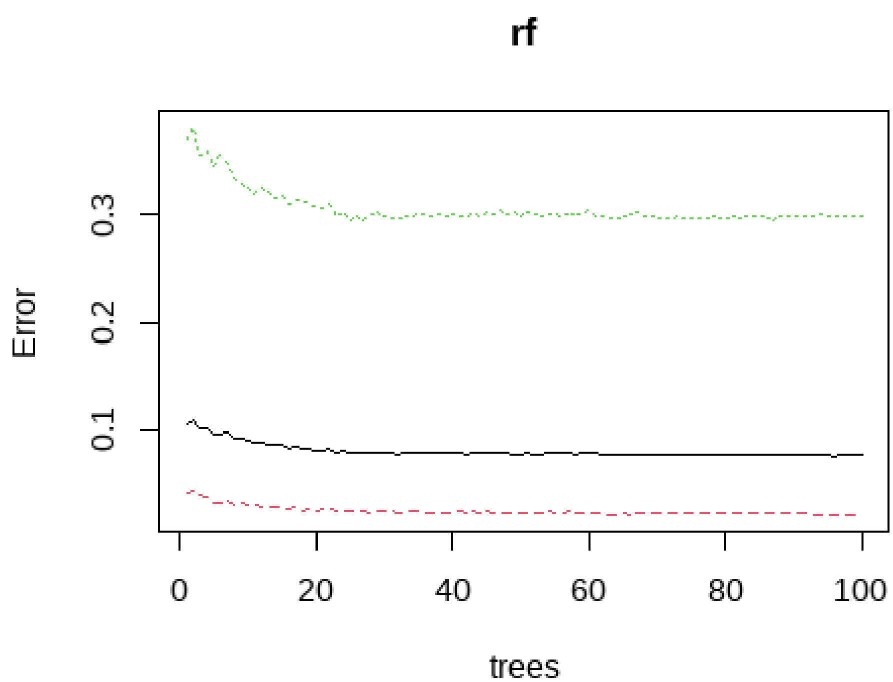
3.4 RANDOM FOREST MODEL

```r
#Random Forest Model

X_rf <- X
X_rf$Target <- as.factor(X_rf$Target)
# X_rf is same as X but with Target as a factor


rf <- randomForest(Target ~ ., data = X_rf, mtry = sqrt(ncol(train_tree)-1),
      ntree = 100)
print(rf)

##
## Call:
```

```
##  randomForest(formula = Target ~ ., data = X_rf, mtry = sqrt(ncol(train_tree) -      1), ntree = 100)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 7.73%
## Confusion matrix:
##      0   1 class.error
## 0 4526 104   0.0224622
## 1  343 808   0.2980017
```

OOB of 7.9% is not bad at all, let's check the plots.

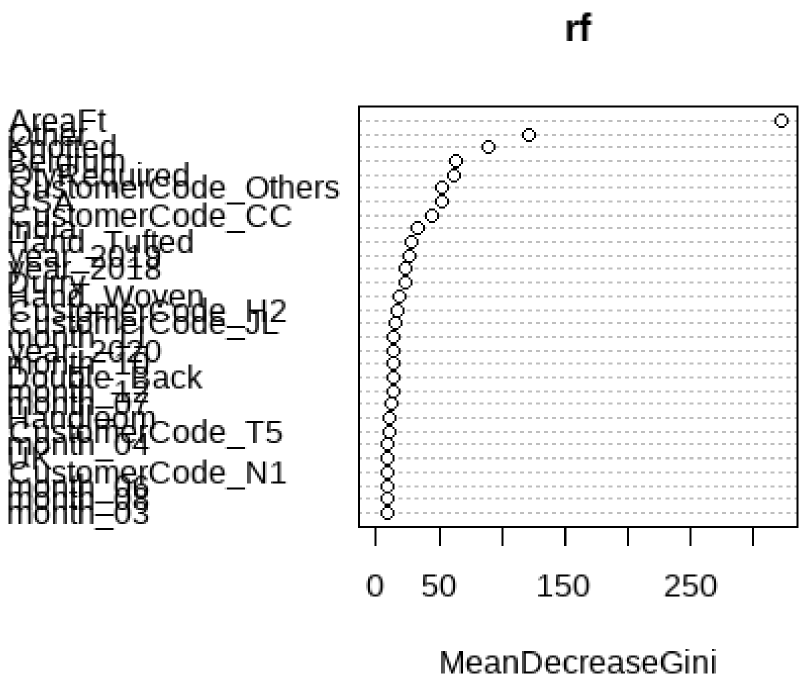
```
plot(rf)
```



3.4.1 Importanat Features

```
importance(rf, type = 2) #using mean decrease gini

##                       MeanDecreaseGini
## USA                          51.661714
## UK                            8.718797
## Italy                         2.344911
## Belgium                      63.693542
## Romania                       3.031090
## Australia                     1.217891
## India                        32.732725
```

```
## QtyRequired                61.598571
## Hand_Tufted                27.958256
## Durry                      23.255803
## Double_Back                13.243370
## Hand_Woven                 18.777610
## Knotted                    89.985209
## Jacquard                    7.462869
## Handloom                   10.497282
## Other                     122.170123
## REC                         4.647635
## Round                       2.901002
## Square                      3.063809
## AreaFt                    322.346377
## CustomerCode_C1             3.311976
## CustomerCode_CC            44.585855
## CustomerCode_H2            16.908460
## CustomerCode_I2             2.853310
## CustomerCode_JL            15.512053
## CustomerCode_M1             2.328480
## CustomerCode_N1             8.598904
## CustomerCode_Others        52.510240
## CustomerCode_P5             4.560096
## CustomerCode_T2             2.868948
## CustomerCode_T5             9.961701
## CustomerCode_TGT            6.138032
## year_2018                  24.069929
## year_2019                  26.455496
## year_2020                  13.653535
## month_02                    6.673053
## month_03                    8.243808
## month_04                    9.335747
## month_05                    7.819652
## month_06                    8.498973
## month_07                   11.505617
## month_08                    8.455099
## month_09                    7.495909
## month_10                   13.293865
## month_11                   13.982725
## month_12                   13.135195

varImpPlot(rf)
```



Let's ignore others for now since it's a culmination of multiple countries, so leaving it out, the important variables are : AreaFt, Knotted, Belgium, QtyRequired, USA, India, CustomerCodeCC etc.

3.4.2 Tuning RF for F-scpre

```
f_sc <- c()
for(mt in seq(6,ncol(train)-15)){
  rf1 <- randomForest(Target~., data = X_rf, ntree = 200, mtry = mt)
  pred_test_rf <- predict(rf1, test, type = "class")
  cm <- table(test$Target, pred_test_rf)
  rc <- cm[2,2]/(cm[2,2]+cm[2,1])
  pr <- cm[2,2]/(cm[2,2]+cm[1,2])
  fs <- 2*(pr*rc/(pr+rc))
  f_sc <- c(f_sc,fs)
  print(paste("F score for mtry -",mt," is: ",fs))
}

## [1] "F score for mtry - 6  is:  0.817307692307692"
## [1] "F score for mtry - 7  is:  0.8274231678487"
## [1] "F score for mtry - 8  is:  0.831353919239905"
## [1] "F score for mtry - 9  is:  0.845070422535211"
## [1] "F score for mtry - 10  is:  0.853828306264501"
## [1] "F score for mtry - 11  is:  0.851851851851852"
## [1] "F score for mtry - 12  is:  0.856470588235294"
## [1] "F score for mtry - 13  is:  0.862470862470862"
## [1] "F score for mtry - 14  is:  0.864485981308411"
## [1] "F score for mtry - 15  is:  0.85981308411215"
```

```
## [1] "F score for mtry - 16  is:  0.865116279069767"
## [1] "F score for mtry - 17  is:  0.869767441860465"
## [1] "F score for mtry - 18  is:  0.859154929577465"
## [1] "F score for mtry - 19  is:  0.871194379391101"
## [1] "F score for mtry - 20  is:  0.871794871794872"
## [1] "F score for mtry - 21  is:  0.873831775700935"
## [1] "F score for mtry - 22  is:  0.871194379391101"
## [1] "F score for mtry - 23  is:  0.873831775700935"
## [1] "F score for mtry - 24  is:  0.871194379391101"
## [1] "F score for mtry - 25  is:  0.873831775700935"
## [1] "F score for mtry - 26  is:  0.875878220140515"
## [1] "F score for mtry - 27  is:  0.877030162412993"
## [1] "F score for mtry - 28  is:  0.875"
## [1] "F score for mtry - 29  is:  0.879069767441861"
## [1] "F score for mtry - 30  is:  0.876456876456876"
## [1] "F score for mtry - 31  is:  0.877934272300469"
## [1] "F score for mtry - 32  is:  0.877030162412993"
```

```
bestmtry <- which.max(f_sc)+5
print(paste("Best value for mtry :",bestmtry))
```

```
## [1] "Best value for mtry : 29"
```

```
rf_tune <- randomForest(Target~., data = X_rf, ntree = 200, mtry = bestmtry)
pred_rf <- predict(rf_tune, test_tree, type = "class")
cm_rft <- table(actual, pred_rf, dnn = c('Actuals', 'Predicted'))
# CM generated for same test set so we can compare models
print(rf1)
```

```
##
## Call:
##  randomForest(formula = Target ~ ., data = X_rf, ntree = 200,      mtry =
mt)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 32
##
##         OOB estimate of  error rate: 7.8%
## Confusion matrix:
##      0   1 class.error
## 0 4482 148  0.03196544
## 1  303 848  0.26324935
```

```
metrics(cm_rft)
```

```
## [1] "Test accuracy : 0.955172413793103"
## [1] "Recall of 1 (Success) : 0.823788546255507"
## [1] "Precision of 1 (Success) : 0.939698492462312"
## [1] "f score of 1 (Success) : 0.877934272300469"
```

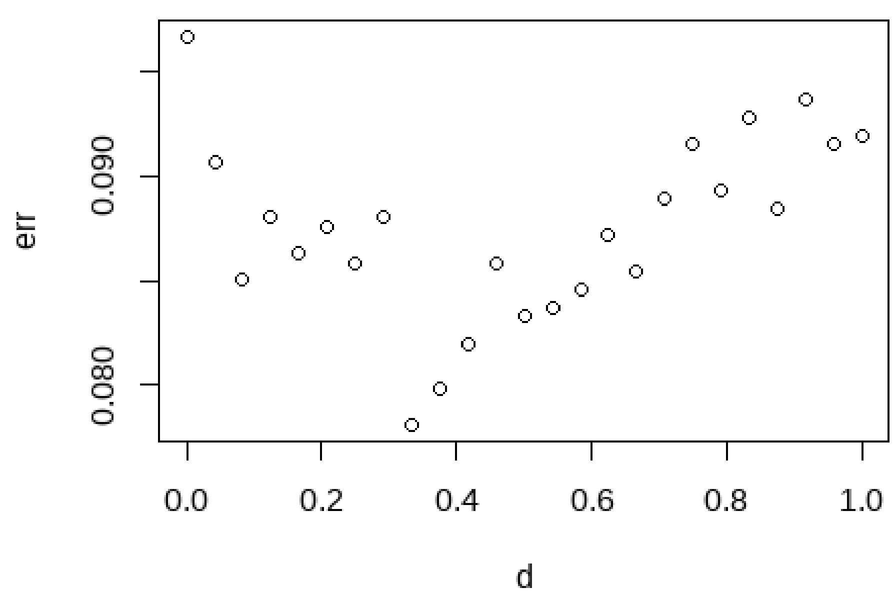
So far, RF has given the best results with 84% recall, and 0.877 f-score!

3.5 Neural Network

3.5.1 Identifying the best decay and size parameters

Best Decay

```
set.seed(156)
indx <- sample(2, nrow(train), replace = T, prob = c(0.5, 0.5))
train2 <- train[indx == 1, ]
validation <- train[indx == 2, ]
err <- vector("numeric", 25)
d <- seq(0.0001, 1, length.out=25)

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

plot(d, err)
```




```
table(d, err)

##               err
## d              0.0781182563659905 0.0798446266724212 0.0820025895554597
```

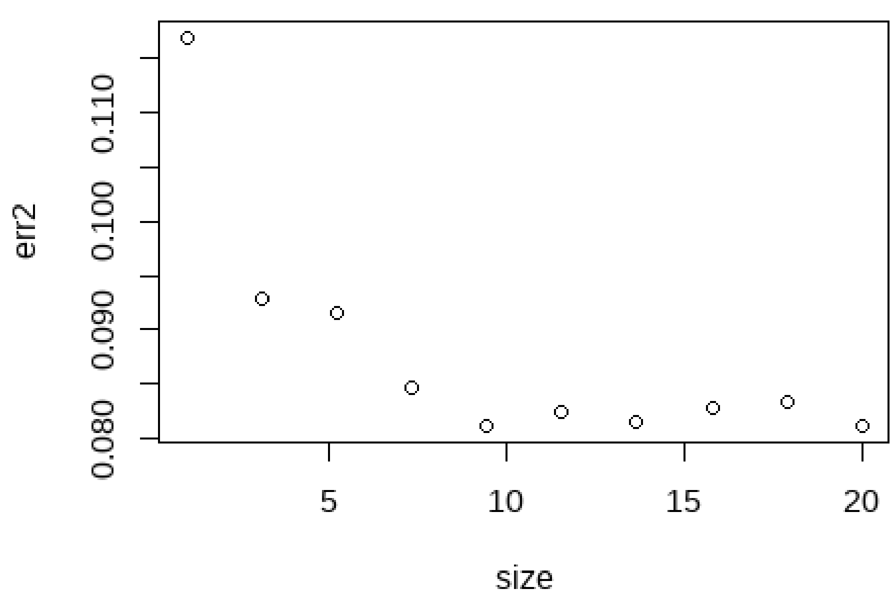
```
##   1e-04                     0                  0                  0
##   0.0417625                 0                  0                  0
##   0.083425                  0                  0                  0
##   0.1250875                 0                  0                  0
##   0.16675                   0                  0                  0
##   0.2084125                 0                  0                  0
##   0.250075                  0                  0                  0
##   0.2917375                 0                  0                  0
##   0.3334                    1                  0                  0
##   0.3750625                 0                  1                  0
##   0.416725                  0                  0                  1
##   0.4583875                 0                  0                  0
##   0.50005                   0                  0                  0
##   0.5417125                 0                  0                  0
##   0.583375                  0                  0                  0
##   0.6250375                 0                  0                  0
##   0.6667                    0                  0                  0
##   0.7083625                 0                  0                  0
##   0.750025                  0                  0                  0
##   0.7916875                 0                  0                  0
##   0.83335                   0                  0                  0
##   0.8750125                 0                  0                  0
##   0.916675                  0                  0                  0
##   0.9583375                 0                  0                  0
##   1                         0                  0                  0
##              err
## d             0.0832973672852827 0.0837289598618904 0.0845921450151057
##   1e-04                     0                  0                  0
##   0.0417625                 0                  0                  0
##   0.083425                  0                  0                  0
##   0.1250875                 0                  0                  0
##   0.16675                   0                  0                  0
##   0.2084125                 0                  0                  0
##   0.250075                  0                  0                  0
##   0.2917375                 0                  0                  0
##   0.3334                    0                  0                  0
##   0.3750625                 0                  0                  0
##   0.416725                  0                  0                  0
##   0.4583875                 0                  0                  0
##   0.50005                   1                  0                  0
##   0.5417125                 0                  1                  0
##   0.583375                  0                  0                  1
##   0.6250375                 0                  0                  0
##   0.6667                    0                  0                  0
##   0.7083625                 0                  0                  0
##   0.750025                  0                  0                  0
##   0.7916875                 0                  0                  0
##   0.83335                   0                  0                  0
##   0.8750125                 0                  0                  0
##   0.916675                  0                  0                  0
```

```
## 	0.9583375            0            0            0
## 	1                    0            0            0
```

We can conclude from the plot above that the best value for decay is 0.3334

Best Size

```
set.seed(5)
size <- seq(1, 20, length.out=10)
err2 <- vector("numeric", 10)
s = 1
for(i in size) {
mymodel <- nnet(as.factor(Target) ~., data = train2, decay = 0.3334, size =
i, maxit = 1000)
pred.class <- predict(mymodel, newdata = validation, type = "class")
err2[s] <- mean(pred.class != validation$Target)
s <- s +1
}

plot(size, err2)
```




```
table(size, err2)

##                     err2
## size                 0.0811394044022443 0.081570996978852 0.0824341821320673
##   1                                   0                 0                  0
##   3.11111111111111                    0                 0                  0
```

```
##   5.22222222222222                   0                  0                   0
##   7.33333333333333                   0                  0                   0
##   9.44444444444444                   1                  0                   0
##   11.5555555555556                   0                  0                   1
##   13.6666666666667                   0                  1                   0
##   15.7777777777778                   0                  0                   0
##   17.8888888888889                   0                  0                   0
##   20                                 1                  0                   0
##                      err2
## size                  0.082865774708675 0.0832973672852827 0.0845921450151057
##   1                                 0                  0                   0
##   3.11111111111111                  0                  0                   0
##   5.22222222222222                  0                  0                   0
##   7.33333333333333                  0                  0                   1
##   9.44444444444444                  0                  0                   0
##   11.5555555555556                  0                  0                   0
##   13.6666666666667                  0                  0                   0
##   15.7777777777778                  1                  0                   0
##   17.8888888888889                  0                  1                   0
##   20                                0                  0                   0
##                      err2
## size                  0.0914976262408287 0.0927924039706517 0.116961588260682
##   1                                  0                  0                  1
##   3.11111111111111                   0                  1                  0
##   5.22222222222222                   1                  0                  0
##   7.33333333333333                   0                  0                  0
##   9.44444444444444                   0                  0                  0
##   11.5555555555556                   0                  0                  0
##   13.6666666666667                   0                  0                  0
##   15.7777777777778                   0                  0                  0
##   17.8888888888889                   0                  0                  0
##   20                                 0                  0                  0
```

** The least error is when the size is 9. So we now have the best size and decay to proceed with our main model **

3.5.2 Building the best Newral Net Model

```
nnModel <- nnet(as.factor(Target) ~ ., data = train, linout = FALSE,
size = 9, decay = 0.3334, maxit = 1000)
```

```
## # weights:  433
## initial  value 2679.417124
## iter  10 value 1441.182570
## iter  20 value 1216.169954
## iter  30 value 1102.263063
## iter  40 value 1042.270715
## iter  50 value 1010.818603
## iter  60 value 987.945103
## iter  70 value 974.837606
## iter  80 value 968.773607
```

```
## iter  90 value 965.749063
## iter 100 value 964.382451
## iter 110 value 962.952029
## iter 120 value 962.025764
## iter 130 value 960.644691
## iter 140 value 958.315806
## iter 150 value 955.077118
## iter 160 value 952.245385
## iter 170 value 950.179593
## iter 180 value 949.468467
## iter 190 value 948.392241
## iter 200 value 947.109979
## iter 210 value 946.679481
## iter 220 value 946.387986
## iter 230 value 946.024044
## iter 240 value 945.570359
## iter 250 value 944.794970
## iter 260 value 944.516070
## iter 270 value 944.237794
## iter 280 value 943.800202
## iter 290 value 943.507114
## iter 300 value 943.253222
## iter 310 value 942.775079
## iter 320 value 942.090181
## iter 330 value 941.539799
## iter 340 value 941.087798
## iter 350 value 940.268416
## iter 360 value 939.822431
## iter 370 value 938.557676
## iter 380 value 937.755988
## iter 390 value 937.471730
## iter 400 value 936.949160
## iter 410 value 936.810082
## iter 420 value 936.779631
## iter 430 value 936.770447
## iter 440 value 936.766942
## iter 450 value 936.766066
## final  value 936.766045
## converged
```

```
summary(nnModel)
```

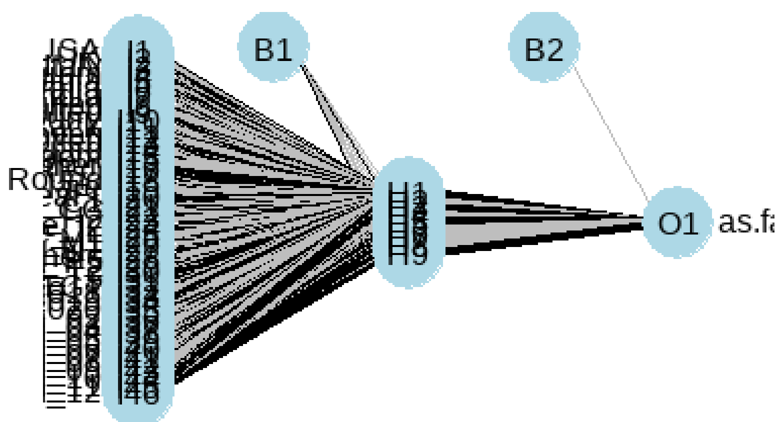
```
## a 46-9-1 network with 433 weights
## options were - entropy fitting  decay=0.3334
##   b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1  i8->h1 
i9->h1 
##   -0.74    0.43    0.75   -0.42   -0.68    0.22   -1.06   -1.04   -2.37 
0.75 
## i10->h1 i11->h1 i12->h1 i13->h1 i14->h1 i15->h1 i16->h1 i17->h1 i18->h1 
i19->h1
```

```
##   -1.10    0.09   -0.78    0.56   -0.13   -2.46    2.33    0.84   -0.37
-1.21
## i20->h1 i21->h1 i22->h1 i23->h1 i24->h1 i25->h1 i26->h1 i27->h1 i28->h1
i29->h1
##    0.60   -0.59   -1.04   -0.22    0.22   -0.12   -0.03   -0.62    0.32
0.00
## i30->h1 i31->h1 i32->h1 i33->h1 i34->h1 i35->h1 i36->h1 i37->h1 i38->h1
i39->h1
##   -0.42    0.87   -0.40   -1.40    1.32   -0.65   -0.72   -0.86   -1.54
-0.14
## i40->h1 i41->h1 i42->h1 i43->h1 i44->h1 i45->h1 i46->h1
##    0.33   -0.02    0.16   -1.80    0.35    2.12    0.65
##   b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2
i9->h2
##   -0.86   -0.31    1.00    0.07   -1.41   -0.65   -0.54    0.43    0.71
-0.24
## i10->h2 i11->h2 i12->h2 i13->h2 i14->h2 i15->h2 i16->h2 i17->h2 i18->h2
i19->h2
##    1.13   -1.35    1.06   -0.72   -1.29   -0.22    0.78    0.42   -0.67
-0.61
## i20->h2 i21->h2 i22->h2 i23->h2 i24->h2 i25->h2 i26->h2 i27->h2 i28->h2
i29->h2
##   -1.42   -0.24    0.43   -1.03   -0.65   -0.39    0.15   -0.34    0.11
-0.09
## i30->h2 i31->h2 i32->h2 i33->h2 i34->h2 i35->h2 i36->h2 i37->h2 i38->h2
i39->h2
##    0.07    1.39    0.90    0.10    0.63   -1.59    0.78    0.16    0.47
0.03
## i40->h2 i41->h2 i42->h2 i43->h2 i44->h2 i45->h2 i46->h2
##    1.41   -2.79    0.94   -0.25    0.17    1.26    1.40
##   b->h3  i1->h3  i2->h3  i3->h3  i4->h3  i5->h3  i6->h3  i7->h3  i8->h3
i9->h3
##    0.91   -2.54    0.72    0.09    1.11    0.49    0.67    1.04   -0.08
1.08
## i10->h3 i11->h3 i12->h3 i13->h3 i14->h3 i15->h3 i16->h3 i17->h3 i18->h3
i19->h3
##    0.48   -1.05    0.61    0.08   -0.09   -0.53    0.33    0.00    0.33
0.57
## i20->h3 i21->h3 i22->h3 i23->h3 i24->h3 i25->h3 i26->h3 i27->h3 i28->h3
i29->h3
##    2.74    0.28    1.04   -0.52    0.49   -1.01    0.51   -0.51   -0.15
0.45
## i30->h3 i31->h3 i32->h3 i33->h3 i34->h3 i35->h3 i36->h3 i37->h3 i38->h3
i39->h3
##    0.09    1.73    1.10    1.54   -0.55   -0.08    1.08    0.65    0.38
0.87
## i40->h3 i41->h3 i42->h3 i43->h3 i44->h3 i45->h3 i46->h3
##   -0.50   -0.21   -0.42    0.03   -0.40    0.49   -0.24
##   b->h4  i1->h4  i2->h4  i3->h4  i4->h4  i5->h4  i6->h4  i7->h4  i8->h4
i9->h4
```

```
##   -0.46    0.45    0.41    0.10   -0.43   -0.29    0.21   -0.70   -0.66
1.69
## i10->h4 i11->h4 i12->h4 i13->h4 i14->h4 i15->h4 i16->h4 i17->h4 i18->h4
i19->h4
##   -1.96   -0.37    0.80    0.65   -0.56    0.04   -0.76    0.59   -1.00
-0.06
## i20->h4 i21->h4 i22->h4 i23->h4 i24->h4 i25->h4 i26->h4 i27->h4 i28->h4
i29->h4
##    0.48    0.33   -0.70   -1.72   -0.29   -0.43   -0.39    1.15   -0.45
-0.61
## i30->h4 i31->h4 i32->h4 i33->h4 i34->h4 i35->h4 i36->h4 i37->h4 i38->h4
i39->h4
##    0.10    0.84    0.56   -1.52    0.69    0.36    0.83   -0.87    0.31
-0.85
## i40->h4 i41->h4 i42->h4 i43->h4 i44->h4 i45->h4 i46->h4
##   -0.17    1.81   -2.06   -0.19    0.06   -0.77   -0.04
##   b->h5  i1->h5  i2->h5  i3->h5  i4->h5  i5->h5  i6->h5  i7->h5  i8->h5
i9->h5
##   -0.41   -1.31   -0.88   -0.33    2.41   -0.37   -0.88    0.06    1.54
0.08
## i10->h5 i11->h5 i12->h5 i13->h5 i14->h5 i15->h5 i16->h5 i17->h5 i18->h5
i19->h5
##   -1.14   -1.78   -0.26    2.15    0.98   -2.01    1.59   -0.51    0.20
-0.09
## i20->h5 i21->h5 i22->h5 i23->h5 i24->h5 i25->h5 i26->h5 i27->h5 i28->h5
i29->h5
##    0.62    0.20    0.06   -0.82   -0.37   -0.11    0.28   -1.15   -1.06
-0.77
## i30->h5 i31->h5 i32->h5 i33->h5 i34->h5 i35->h5 i36->h5 i37->h5 i38->h5
i39->h5
##   -0.33   -0.77    0.12   -1.43    0.16    0.87   -0.79   -1.36    1.00
0.06
## i40->h5 i41->h5 i42->h5 i43->h5 i44->h5 i45->h5 i46->h5
##    0.77   -0.60   -0.18    0.59   -0.54    0.04    0.79
##   b->h6  i1->h6  i2->h6  i3->h6  i4->h6  i5->h6  i6->h6  i7->h6  i8->h6
i9->h6
##   -0.75   -1.52    0.36    0.31   -0.99    0.15   -0.37    0.96   -1.61
1.20
## i10->h6 i11->h6 i12->h6 i13->h6 i14->h6 i15->h6 i16->h6 i17->h6 i18->h6
i19->h6
##   -0.38    1.17    0.57   -2.27    1.86   -1.03   -1.86   -0.86    1.27
-1.15
## i20->h6 i21->h6 i22->h6 i23->h6 i24->h6 i25->h6 i26->h6 i27->h6 i28->h6
i29->h6
##   -0.34   -0.25    0.96   -0.71    0.15   -0.76    0.19   -0.52   -0.96
1.29
## i30->h6 i31->h6 i32->h6 i33->h6 i34->h6 i35->h6 i36->h6 i37->h6 i38->h6
i39->h6
##    0.31    1.12   -0.57   -1.60   -0.32    1.18   -1.06   -0.32    0.51
0.34
```

```
## i40->h6 i41->h6 i42->h6 i43->h6 i44->h6 i45->h6 i46->h6
##   -0.49    0.13    0.79    0.62   -0.83    0.83    0.89
##   b->h7  i1->h7  i2->h7  i3->h7  i4->h7  i5->h7  i6->h7  i7->h7  i8->h7
i9->h7
##   -0.49   -0.51   -0.19    0.54   -1.44    0.10    0.03    1.01    0.66
-0.93
## i10->h7 i11->h7 i12->h7 i13->h7 i14->h7 i15->h7 i16->h7 i17->h7 i18->h7
i19->h7
##   -1.01   -1.29   -0.57    0.91    1.22   -1.02    2.19    0.47   -0.54
-0.43
## i20->h7 i21->h7 i22->h7 i23->h7 i24->h7 i25->h7 i26->h7 i27->h7 i28->h7
i29->h7
##    0.48   -0.80    1.01    0.28    0.10   -0.38    0.48    0.46    0.44
0.93
## i30->h7 i31->h7 i32->h7 i33->h7 i34->h7 i35->h7 i36->h7 i37->h7 i38->h7
i39->h7
##    0.54    0.19    0.09   -2.20    0.78    0.93   -0.36   -0.42   -0.49
-0.09
## i40->h7 i41->h7 i42->h7 i43->h7 i44->h7 i45->h7 i46->h7
##    0.29    0.79    0.13   -1.26   -0.18    1.57    1.01
##   b->h8  i1->h8  i2->h8  i3->h8  i4->h8  i5->h8  i6->h8  i7->h8  i8->h8
i9->h8
##   -0.53   -0.10   -0.04    0.02    0.05   -0.57   -0.88    0.10    1.06
0.29
## i10->h8 i11->h8 i12->h8 i13->h8 i14->h8 i15->h8 i16->h8 i17->h8 i18->h8
i19->h8
##    0.39    0.28   -0.76   -0.14    0.11   -0.88    0.19    0.15   -0.35
-0.33
## i20->h8 i21->h8 i22->h8 i23->h8 i24->h8 i25->h8 i26->h8 i27->h8 i28->h8
i29->h8
##    2.27    0.10    0.10    1.01   -0.57   -1.20   -0.24    0.11   -0.78
-0.50
## i30->h8 i31->h8 i32->h8 i33->h8 i34->h8 i35->h8 i36->h8 i37->h8 i38->h8
i39->h8
##    0.02    1.17   -0.32    0.70   -0.86   -0.37    0.23    0.58    0.09
-0.86
## i40->h8 i41->h8 i42->h8 i43->h8 i44->h8 i45->h8 i46->h8
##   -0.66   -2.01   -1.54    0.06    1.35    1.75    0.49
##   b->h9  i1->h9  i2->h9  i3->h9  i4->h9  i5->h9  i6->h9  i7->h9  i8->h9
i9->h9
##    0.46   -0.47    0.92    0.02   -0.81    0.11   -0.37    0.69    0.73
0.42
## i10->h9 i11->h9 i12->h9 i13->h9 i14->h9 i15->h9 i16->h9 i17->h9 i18->h9
i19->h9
##   -0.45    0.18    1.60   -0.98   -0.78    0.32    0.17    0.00   -0.19
0.65
## i20->h9 i21->h9 i22->h9 i23->h9 i24->h9 i25->h9 i26->h9 i27->h9 i28->h9
i29->h9
##    2.50   -0.31    0.69   -0.55    0.11   -0.14   -0.51    1.05    0.14
0.32
```

```
## i30->h9 i31->h9 i32->h9 i33->h9 i34->h9 i35->h9 i36->h9 i37->h9 i38->h9
i39->h9
##    0.02    1.06   -0.08   -0.04    1.37   -0.87    0.50    0.34   -0.41
-0.47
## i40->h9 i41->h9 i42->h9 i43->h9 i44->h9 i45->h9 i46->h9
##   -0.52   -1.56   -0.55   -0.71   -0.09    1.57    1.35
##  b->o h1->o h2->o h3->o h4->o h5->o h6->o h7->o h8->o h9->o
## -0.29  5.54 -4.97  5.38 -5.30  7.03 -6.73 -5.14 -5.04  4.57

plotnet(nnModel)
```




```
nn.preds <- as.factor(predict(nnModel, test, type = "class"))
CM <- table(test$Target, nn.preds, dnn = c("actual","predicted"))
print(CM)

##       predicted
## actual   0   1
##      0 896  37
##      1  60 167

nn.preds

## Output has been removed



outPutlist <- full_metrics(CM)
```

```
## [1] "Test accuracy : 0.916379310344828"
## [1] "Recall (Success) : 0.73568281938326"
## [1] "Precision (Success) : 0.818627450980392"
## [1] "F-score (Success) : 0.774941995359629"
## [1] "Specificity (tnr) : 0.960342979635584"
## [1] "False positve rate: 0.0396570203644159"
## [1] "False negative rate: 0.26431718061674"
```

3.6 KFOLD Cross Validation for all Models (IMBALANCED)

```
X <- X[sample(nrow(X)), ] #Shufffling row
k <- 10
nmethod <- 4
folds <- cut(seq(1,nrow(X)), breaks=k, labels=FALSE)
model_fscore <- matrix(-1, k, nmethod, dimnames= list(paste0("Folds ", 1:k),

c("LogisticRegression","DecisionTree","RandomForest","NeuralNetwork"))
        )

for (i in 1:k){
  test_ind <- which(folds==i, arr.ind = TRUE)
  test_cv <- X[test_ind, ]
  train_cv <- X[-test_ind, ]

  train_tree_cv <- train_cv
  test_tree_cv <- test_cv
  train_tree_cv$Target <- as.factor(train_tree_cv$Target)
  test_tree_cv$Target <- as.factor(test_tree_cv$Target)

  #Logistoc
  CV_lr <- glm(Target~., data = train_cv)
  lr_pred <- predict(CV_lr, newdata = test_cv)
  lr_class <- as.factor(ifelse(lr_pred >= 0.5, 1, 0))
  cm_lr <- table(as.factor(test_cv$Target), lr_class)
  rc_lr <- cm_lr[2,2]/(cm_lr[2,2]+cm_lr[2,1])
  pr_lr <- cm_lr[2,2]/(cm_lr[2,2]+cm_lr[1,2])
  fs_lr <- 2*(pr_lr*rc_lr/(pr_lr+rc_lr))

  #DT
  CVtree <- rpart(Target ~ ., train_tree_cv, parms = list(split = "gini"),
    control = rpart.control(minbucket = 5, minsplit = 5, cp=cp))
  dt_pred <- predict(CVtree, test_tree_cv, type='class')
  cm_dt <- table(test_tree_cv$Target, dt_pred)
  rc_dt <- cm_dt[2,2]/(cm_dt[2,2]+cm_dt[2,1])
  pr_dt <- cm_dt[2,2]/(cm_dt[2,2]+cm_dt[1,2])
  fs_dt <- 2*(pr_dt*rc_dt/(pr_dt+rc_dt))

  #RF
  CVrf <- randomForest(Target~., data = train_tree_cv, ntree = 200, mtry =
bestmtry)
```

```
  rf_pred <- predict(CVrf, test_tree_cv, type='class')
  cm_rf <- table(test_tree_cv$Target, rf_pred)
  rc_rf <- cm_rf[2,2]/(cm_rf[2,2]+cm_rf[2,1])
  pr_rf <- cm_rf[2,2]/(cm_rf[2,2]+cm_rf[1,2])
  fs_rf <- 2*(pr_rf*rc_rf/(pr_rf+rc_rf))

  #Neural Net
  nnModel <- nnet(as.factor(Target) ~ ., data = train_cv, linout = FALSE,
                  size = 5, decay = 0.3334, maxit = 1000)
  nn.preds <- as.factor(predict(nnModel, test_cv, type = "class"))
  cm_nn <- table(as.factor(test_cv$Target), nn.preds)
  rc_nn <- cm_nn[2,2]/(cm_nn[2,2]+cm_nn[2,1])
  pr_nn <- cm_nn[2,2]/(cm_nn[2,2]+cm_nn[1,2])
  fs_nn <- 2*(pr_nn*rc_nn/(pr_nn+rc_nn))

  model_fscore[i,1] <- fs_lr
  model_fscore[i,2] <- fs_dt
  model_fscore[i,3] <- fs_rf
  model_fscore[i,4] <- fs_nn

}


print("K-Fold f-score for all models")

## [1] "K-Fold f-score for all models"

model_fscore

##            LogisticRegression DecisionTree RandomForest NeuralNetwork
## Folds 1             0.7058824    0.8416290    0.8401826     0.8161435
## Folds 2             0.6320755    0.7500000    0.7881356     0.7916667
## Folds 3             0.6478873    0.7983871    0.8000000     0.7804878
## Folds 4             0.5595238    0.7065217    0.7195767     0.7539267
## Folds 5             0.6321839    0.7500000    0.7802691     0.7192118
## Folds 6             0.5454545    0.8000000    0.8059701     0.7772021
## Folds 7             0.6595745    0.7870370    0.8235294     0.8055556
## Folds 8             0.5578947    0.7523810    0.7727273     0.7081340
## Folds 9             0.5614035    0.7342995    0.7414634     0.7106599
## Folds 10            0.5211268    0.7861272    0.7613636     0.7692308
```

As seen before RF gives the best f-score at 3 Folds!

```
print(paste("Mean fscore for Logical Regresssion: ",mean(model_fscore[,1])))

## [1] "Mean fscore for Logical Regresssion:  0.60230068858698"

print(paste("Mean fscore for Decision Tree: ",mean(model_fscore[,2])))

## [1] "Mean fscore for Decision Tree:  0.770638246913691"
```

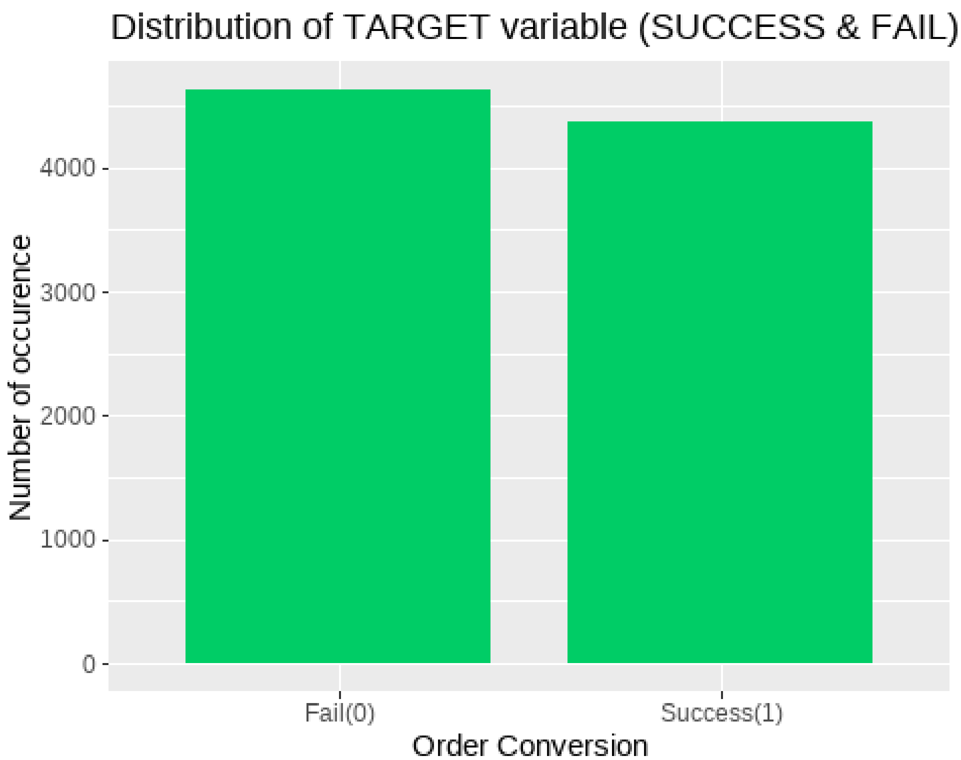
```
print(paste("Mean fscore for Random Forest: ",mean(model_fscore[,3])))

## [1] "Mean fscore for Random Forest:  0.783321790423834"

print(paste("Mean fscore for Neural Network: ",mean(model_fscore[,4])))

## [1] "Mean fscore for Neural Network:  0.763221876062755"
```

While mean f-score for DT, RF and Neural Networks is somewhat similar, we saw RF having the max value for f-score individually.

3.7 Balancing data using Over Sampling

We've decided to do oversampling since we have only ~5k records, so we don't want to reduce the training set by under sampling.

```
over_X <- ovun.sample(Target~., data = X, method = "over", N = 9000)$data
print("Proportion of Success(1) and Failure(0) after Balancing :")

## [1] "Proportion of Success(1) and Failure(0) after Balancing :"

table(over_X$Target)

##
##    0    1
## 4630 4370

df3 <- over_X
df3$Target <- as.factor(ifelse(df3$Target == 0, "Fail(0)", "Success(1)"))
ggplot(df3, aes(x=factor(Target)))+ geom_bar(stat="count",
width=0.8,fill='springgreen3')+
xlab('Order Conversion') + ylab('Number of occurence')+
ggtitle("Distribution of TARGET variable (SUCCESS & FAIL) after BALANCING")
```




```
over_X <- over_X[sample(nrow(over_X)), ] #Shufffling row
k <- 10
nmethod <- 4
folds <- cut(seq(1,nrow(over_X)), breaks=k, labels=FALSE)
model_fscore <- matrix(-1, k, nmethod, dimnames= list(paste0("Folds ", 1:k),

c("LogisticRegression","DecisionTree","RandomForest","NeuralNetwork"))
        )

for (i in 1:k){
  test_ind <- which(folds==i, arr.ind = TRUE)
  test_cv <- over_X[test_ind, ]
  train_cv <- over_X[-test_ind, ]

  train_tree_cv <- train_cv
  test_tree_cv <- test_cv
  train_tree_cv$Target <- as.factor(train_tree_cv$Target)
  test_tree_cv$Target <- as.factor(test_tree_cv$Target)

  #LR
  CV_lr <- glm(Target~., data = train_cv)
  lr_pred <- predict(CV_lr, newdata = test_cv)
  lr_class <- as.factor(ifelse(lr_pred >= 0.5, 1, 0))
  cm_lr <- table(as.factor(test_cv$Target), lr_class)
  rc_lr <- cm_lr[2,2]/(cm_lr[2,2]+cm_lr[2,1])
  pr_lr <- cm_lr[2,2]/(cm_lr[2,2]+cm_lr[1,2])
```

```
  fs_lr <- 2*(pr_lr*rc_lr/(pr_lr+rc_lr))

  #DT
  CVtree <- rpart(Target ~ ., train_tree_cv, parms = list(split = "gini"),
    control = rpart.control(minbucket = 5, minsplit = 5, cp=cp))
  dt_pred <- predict(CVtree, test_tree_cv, type='class')
  cm_dt <- table(test_tree_cv$Target, dt_pred)
  rc_dt <- cm_dt[2,2]/(cm_dt[2,2]+cm_dt[2,1])
  pr_dt <- cm_dt[2,2]/(cm_dt[2,2]+cm_dt[1,2])
  fs_dt <- 2*(pr_dt*rc_dt/(pr_dt+rc_dt))

  #RF
  CVrf <- randomForest(Target~., data = train_tree_cv, ntree = 200, mtry =
bestmtry)
  rf_pred <- predict(CVrf, test_tree_cv, type='class')
  cm_rf <- table(test_tree_cv$Target, rf_pred)
  rc_rf <- cm_rf[2,2]/(cm_rf[2,2]+cm_rf[2,1])
  pr_rf <- cm_rf[2,2]/(cm_rf[2,2]+cm_rf[1,2])
  fs_rf <- 2*(pr_rf*rc_rf/(pr_rf+rc_rf))

   #Neural Net
  nnModel <- nnet(as.factor(Target) ~ ., data = train_cv, linout = FALSE,
                  size = 5, decay = 0.3334, maxit = 1000)
  nn.preds <- as.factor(predict(nnModel, test_cv, type = "class"))
  cm_nn <- table(as.factor(test_cv$Target), nn.preds)
  rc_nn <- cm_nn[2,2]/(cm_nn[2,2]+cm_nn[2,1])
  pr_nn <- cm_nn[2,2]/(cm_nn[2,2]+cm_nn[1,2])
  fs_nn <- 2*(pr_nn*rc_nn/(pr_nn+rc_nn))

  model_fscore[i,1] <- fs_lr
  model_fscore[i,2] <- fs_dt
  model_fscore[i,3] <- fs_rf
  model_fscore[i,4] <- fs_nn

}



print('F-score for K-flods after Balancing Data')

## [1] "F-score for K-flods after Balancing Data"

model_fscore

##          LogisticRegression DecisionTree RandomForest NeuralNetwork
## Folds 1           0.8042204    0.8680947    0.9135255     0.8664422
## Folds 2           0.8037166    0.9115646    0.9353008     0.8794489
## Folds 3           0.8094170    0.8949079    0.9314775     0.8930131
## Folds 4           0.8110048    0.8967972    0.9404901     0.8769415
## Folds 5           0.8120805    0.9017467    0.9357602     0.9108696
```

```
## Folds 6          0.8122867    0.8970917    0.9328933     0.8811659
## Folds 7          0.8153310    0.8921023    0.9242591     0.8871508
## Folds 8          0.8038741    0.8784597    0.9103774     0.8592411
## Folds 9          0.8161329    0.9082462    0.9239501     0.9019608
## Folds 10         0.7841727    0.8922717    0.9172414     0.8628370
```

```
print(paste("Mean fscore for Logical Regresssion: ",mean(model_fscore[,1])))

## [1] "Mean fscore for Logical Regresssion:  0.807223668173878"

print(paste("Mean fscore for Decision Tree: ",mean(model_fscore[,2])))

## [1] "Mean fscore for Decision Tree:  0.894128274772933"

print(paste("Mean fscore for Random Forest: ",mean(model_fscore[,3])))

## [1] "Mean fscore for Random Forest:  0.926527520235542"

print(paste("Mean fscore for Neural Network: ",mean(model_fscore[,4])))

## [1] "Mean fscore for Neural Network:  0.881907094569292"
```

We can see an overall improvement in the model performance now, all models have a higher F-score after balancing, while LR gets the biggest boost from around 55% to 80% now.

Also, we can observe that RF performs much better than other models with an average F-score of 0.93.

3.8 Comparing ROC curves for different Models

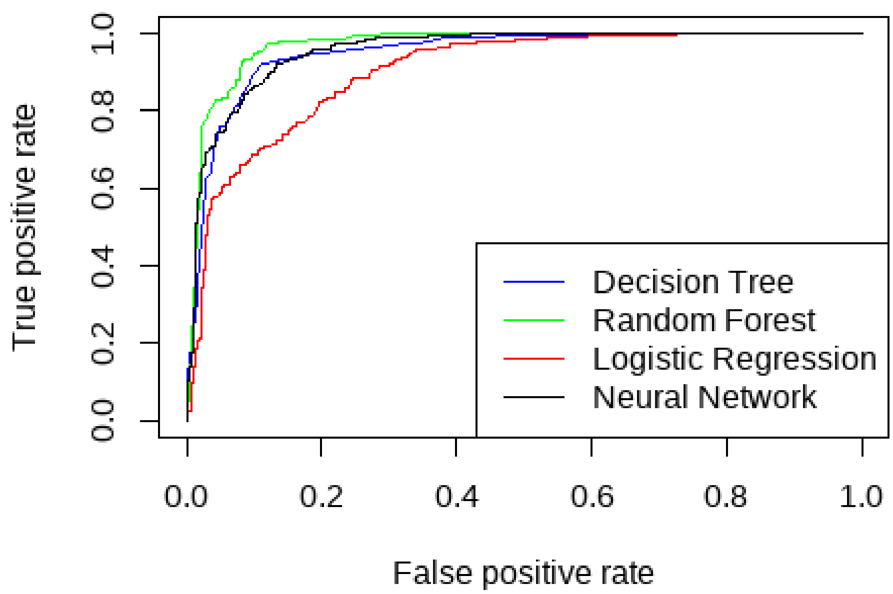
```
CV_lr <- glm(Target~., data = train_cv)
lr_score <- predict(CV_lr, newdata = test_cv)

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

lr_pred <- prediction(lr_score, test_cv$Target)

CVtree <- rpart(Target ~ ., train_tree_cv, parms = list(split = "gini"),
      control = rpart.control(minbucket = 5, minsplit = 5, cp=cp))
dt_score <- predict(CVtree, test_tree_cv, type='prob')
dt_pred <- prediction(dt_score[,2], test_cv$Target)

CVrf <- randomForest(Target~., data = train_tree_cv, ntree = 200, mtry =
bestmtry)
rf_score <- predict(CVrf, test_tree_cv, type='prob')
rf_pred <- prediction(rf_score[,2], test_cv$Target)
```

```r
nnModel <- nnet(as.factor(Target) ~ ., data = train_cv, linout = FALSE,
                  size = 5, decay = 0.3334, maxit = 1000)
```

```
## # weights:  241
## initial  value 5964.301664
## iter  10 value 3672.060080
## iter  20 value 3286.438081
## iter  30 value 2951.052795
## iter  40 value 2794.600465
## iter  50 value 2694.556433
## iter  60 value 2601.118764
## iter  70 value 2512.157958
## iter  80 value 2442.484739
## iter  90 value 2404.469969
## iter 100 value 2366.894754
## iter 110 value 2336.690466
## iter 120 value 2312.641885
## iter 130 value 2296.080244
## iter 140 value 2281.717240
## iter 150 value 2256.314227
## iter 160 value 2240.503770
## iter 170 value 2232.190108
## iter 180 value 2221.954491
## iter 190 value 2211.393254
## iter 200 value 2208.299479
## iter 210 value 2205.887334
## iter 220 value 2202.924017
## iter 230 value 2201.327677
## iter 240 value 2199.656258
## iter 250 value 2198.312188
## iter 260 value 2196.885932
## iter 270 value 2195.108927
## iter 280 value 2191.862962
## iter 290 value 2190.412623
## iter 300 value 2188.843956
## iter 310 value 2186.704752
## iter 320 value 2186.428082
## iter 330 value 2186.299491
## iter 340 value 2186.244171
## iter 350 value 2186.241531
## final  value 2186.241451
## converged
```

```r
nn.preds <- as.factor(predict(nnModel, test_cv))
nn_pred <- prediction(as.numeric(nn.preds), test_cv$Target)

perf_lr <- performance(lr_pred, "tpr","fpr")
perf_dt <- performance(dt_pred, "tpr", "fpr")
perf_rf <- performance(rf_pred, "tpr", "fpr")
perf_nn <- performance(nn_pred, "tpr", "fpr")
```

```r
plot(perf_dt, col="blue")
plot(perf_rf, add = TRUE, col="green")
plot(perf_lr, add = TRUE, col="red")
plot(perf_nn, add = TRUE, col="black")
legend("bottomright", c("Decision Tree", "Random Forest","Logistic
Regression","Neural Network"),
       lty=1, col = c("blue","green","red","black"))
```



From the ROC Curves we can conclude that our tuned Random Forest Model is performing better than our other model. This also coincides with our earlier findings. Hence, let's fix Random Forest as our final model and get the best cut-off points/threshold value.

```r
# Function to get cut-off points :
opt.cut <- function(perf){
  cut.ind <- mapply(FUN = function(x,y,p){
    d=(x-0)^2+(y-1)^2
    ind<- which(d==min(d))
    c(recall = y[[ind]], specificity = 1-x[[ind]],cutoff = p[[ind]])
    },
  perf@x.values, perf@y.values,perf@alpha.values)
}

print("Cut-off points :")
```

```
## [1] "Cut-off points :"

print(paste("tpr :",opt.cut(perf_rf)[1]))

## [1] "tpr : 0.927058823529412"

print(paste("fpr :",1-opt.cut(perf_rf)[2]))

## [1] "fpr : 0.0821052631578947"

print(paste("Best Threshold :",opt.cut(perf_rf)[3]))

## [1] "Best Threshold : 0.66"

bst_thr <- opt.cut(perf_rf)[3]
```

3.9 Final Model and Evaluation Charts

** We'll build a Random Forest model using the above threshold and parameters which we tuned earlier to create the best/final model **

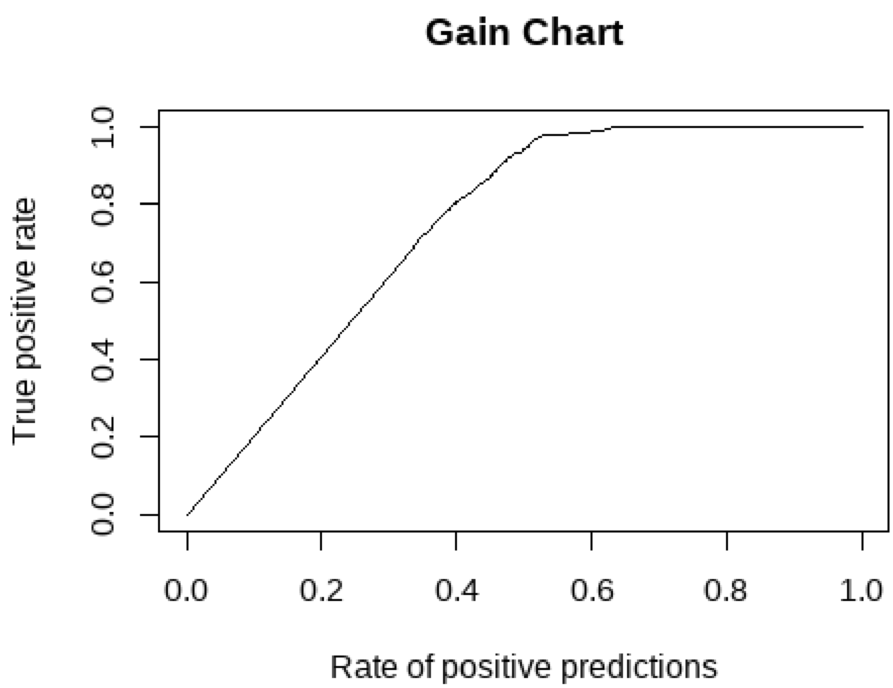
```
rf_final <- randomForest(Target~., data = train_tree_cv, ntree = 300, mtry =
bestmtry)
rf_final_score <- predict(rf_final, test_tree_cv, type='prob')
rf_final_class <- predict(rf_final, test_tree_cv, type='class')
rf_final_class_thresh <-
as.factor(ifelse(rf_final_score[,2]>=bst_thr,"1","0"))
rf_final_pred <- prediction(rf_final_score[,2], test_tree_cv$Target)
cm_rf_final_thresh <- table(test_tree_cv$Target, rf_final_class_thresh)
cm_rf_final <- table(test_tree_cv$Target, rf_final_class)


print("Confusion matrix and other performance metrics after implementing
threshold")

## [1] "Confusion matrix and other performance metrics after implementing
threshold"

cm_rf_final_thresh

##    rf_final_class_thresh
##       0   1
##   0 435  40
##   1  33 392

full_metrics(cm_rf_final_thresh)

## [1] "Test accuracy : 0.918888888888889"
## [1] "Recall (Success) : 0.922352941176471"
## [1] "Precision (Success) : 0.907407407407407"
## [1] "F-score (Success) : 0.914819136522754"
## [1] "Specificity (tnr) : 0.91578947368421"
```

```
## [1] "False positve rate: 0.0842105263157895"
## [1] "False negative rate: 0.0776470588235294"
```

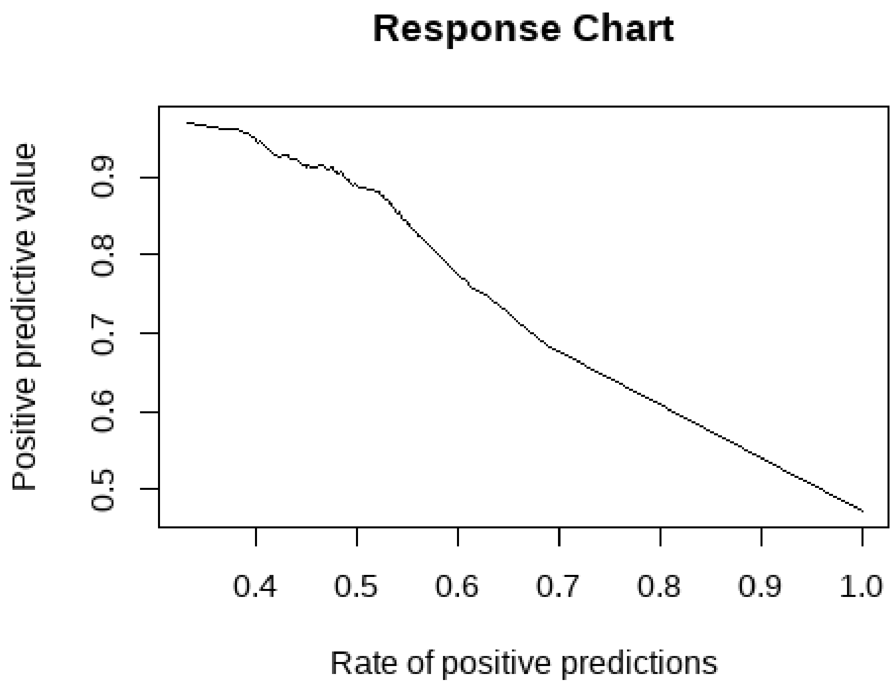
3.9.1 EVALUATION CHARTS

We'll plot the evaluation charts using the best model, which is RF in our case.

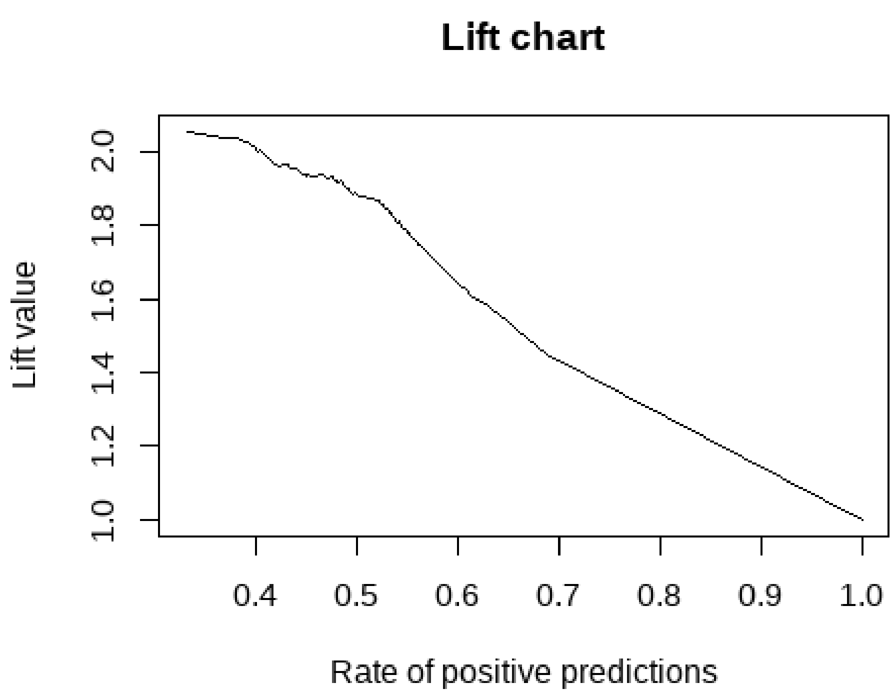
```r
# Gain Chart
perf <- performance(rf_final_pred, "tpr", "rpp")
plot(perf, main="Gain Chart")
```



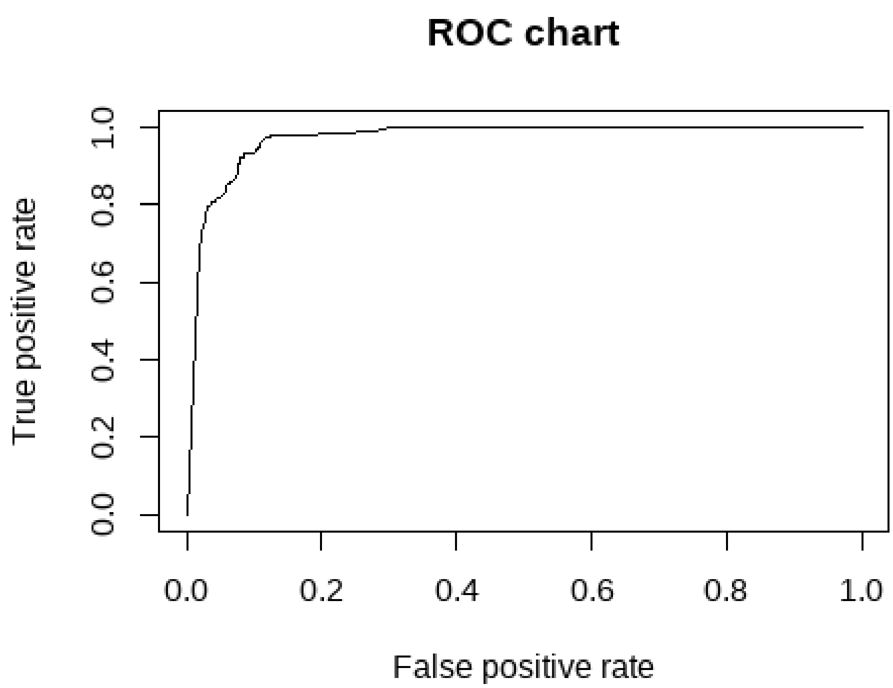

```r
# Response Chart
perf <- performance(rf_final_pred, "ppv", "rpp")
plot(perf, main='Response Chart')
```




```
# Lift Chart
perf <- performance(rf_final_pred, "lift", "rpp")
plot(perf, main='Lift chart')
```



```r
# ROC Curve
perf <- performance(rf_final_pred, "tpr", "fpr")
plot(perf, main='ROC chart')
```




```r
# auc
auc <- performance(rf_final_pred, "auc")
auc <- unlist(slot(auc, "y.values"))
print(paste0("ROC AUC Score for the Final Model is:",auc))

## [1] "ROC AUC Score for the Final Model is:0.9704"
```

4. Customer segmentation using Clustering

We'll perform clustering for the given data to identify which customers can be categorized into a single cluster and to segmet the customers. If customers are in the same cluster items can be recommended to them based on the buying of other customers within the same cluster.

We'll be using K-means, agglomerative and heirarhical clustering to identify clusters within the dataset.

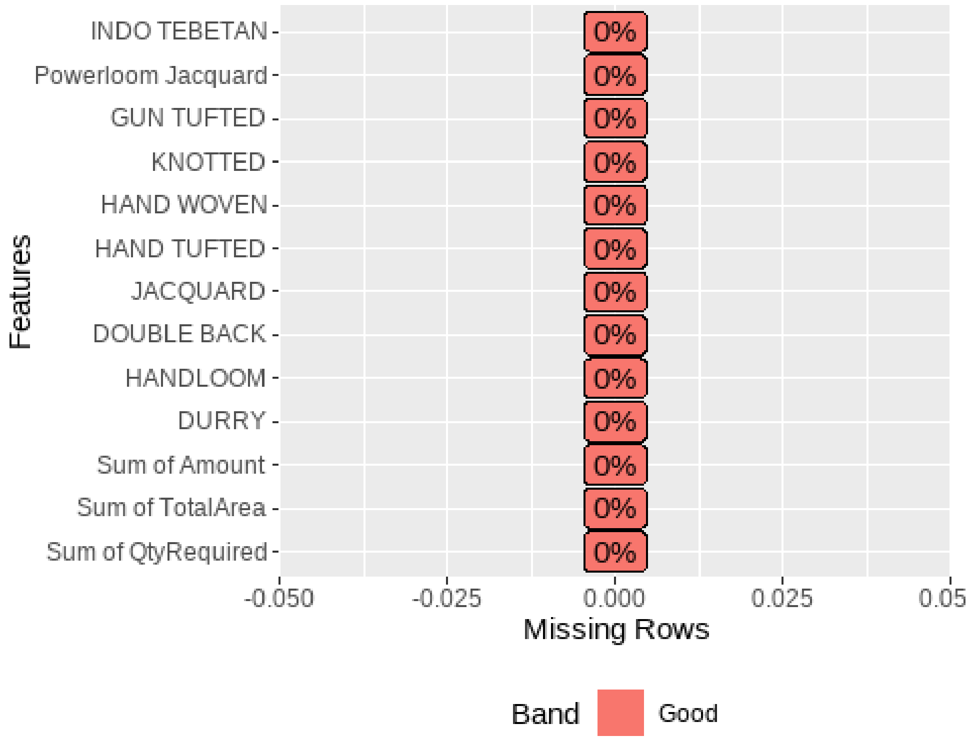
4.1 Preparing data for clustering

```r
dfc <- read_excel("C:/Users/rshara4/Documents/hw3/Champo Carpets.xlsx",
sheet=6)%>%as.data.frame()
rownames(dfc) <- dfc$`Row Labels`
dfc<-dfc[,-1] #removing customer column, since we've made it rowlabels
```

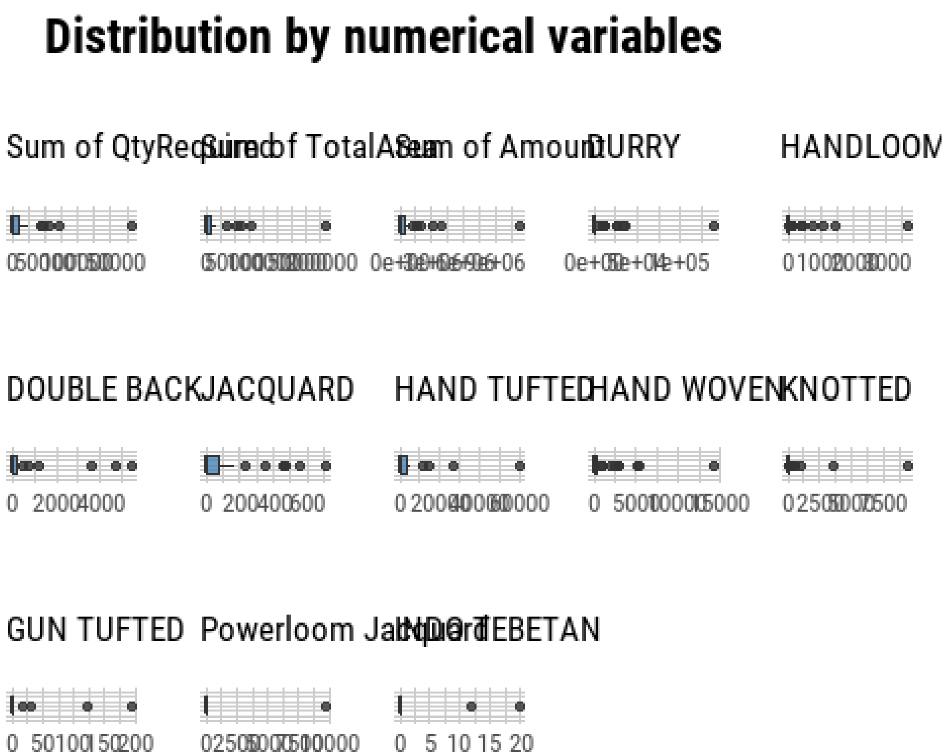
```
dfc <- na.omit(dfc)
dfc

##       Sum of QtyRequired Sum of TotalArea Sum of Amount  DURRY HANDLOOM
## A-11                2466         139.5900  1.854041e+05   1021     1445
## A-6                  131        2086.0000  6.247460e+03      0        0
## A-9                18923       53625.6544  1.592080e+06   3585        0
## B-2                  624         202.8987  1.481116e+04    581        0
## B-3                  464        8451.5625  5.862687e+04      0        0
## B-4                  692        3244.2500  2.624250e+04     80      102
## C-1                 5137       62763.0555  5.676207e+05    288        0
## C-2                55172        9510.0000  1.557123e+06  37042        0
## C-3                 1566        4016.0000  9.906235e+04   1240        0
## CC                  5077        7695.9930  1.475740e+05      4       30
## CTS                  565         420.0000  2.380000e+04      0        0
## DR                   149         305.9765  2.864812e+04      0        0
## E-2                  581       18878.0000  1.167783e+05     13        0
## F-1                 1158        2822.0000  1.168382e+05    288        0
## F-6                 1400           1.3500  1.680000e+04   1400        0
## G-1                  146        5348.0000  3.970124e+04      0        0
## G-4                  119          21.9352  3.288752e+02    119        0
## H-1                 1137        9327.0625  6.538379e+04     39        0
## H-2               183206       19505.3958  3.804801e+06 139618     3673
## I-2                 7501        1508.6320  4.266260e+05    978      788
## JL                 18861        2980.6500  1.231578e+06   5310        0
## K-2                  438        3852.0790  5.987727e+04    358        0
## K-3                    3          80.6666  4.099995e+02      0        0
## L-2                  313          81.9400  2.150349e+04      0        0
## L-3                  760        1721.0000  9.075675e+04      0        0
## L-4                  776           7.3600  4.423400e+04    776        0
## L-5                25840         210.0000  3.588900e+05  25840        0
## M-1                16649      209725.2220  1.959794e+06    412     1085
## M-2                 6926        8200.3959  3.342452e+05   1869        0
## N-1                72888         919.6505  9.493757e+05  12203        0
## P-4                16653        1834.0000  2.925444e+05  12900        0
## P-5                48373       79666.7905  3.066518e+06  25997      138
## PC                  1294        8781.0625  2.279496e+05      0        0
## PD                 11146         725.0137  4.045289e+05   9950      133
## R-4                  175          48.4000  1.010880e+04    175        0
## RC                  3022        1898.1906  3.282907e+05    527        0
## S-2                 1712         528.8725  5.674273e+04    289        0
## S-3                  604        1800.0000  6.136800e+04      0        0
## T-2                 5468        2434.7624  5.630988e+05    299      395
## T-4                 5677        2811.3750  2.382410e+05   1560      450
## T-5                42967        9221.3750  7.338330e+05  34651      110
## T-6                 1737        2120.0000  1.014880e+05      4        0
## T-9                    2          17.2800  7.589700e+02      0        0
## TGT                15045       37630.3318  1.134105e+07      0        0
## V-1                  447         376.7690  4.776128e+04    219        0
```

```
##       DOUBLE BACK JACQUARD HAND TUFTED HAND WOVEN KNOTTED GUN TUFTED
## A-11            0        0           0          0       0          0
## A-6            25      106           0          0       0          0
## A-9           175      714       11716       2116     617          0
## B-2             0        2           0         41       0          0
## B-3           459        5           0          0       0          0
## B-4             0        0         510          0       0          0
## C-1             0        0        4176        220     453          0
## C-2             0        0        3816      14314       0          0
## C-3             0        0         326          0       0          0
## CC              3        0        5021          0       0         19
## CTS             0        0         565          0       0          0
## DR             16        6          13          0     114          0
## E-2           348      151           0         51      18          0
## F-1            64        0         806          0       0          0
## F-6             0        0           0          0       0          0
## G-1            52       68           0         26       0          0
## G-4             0        0           0          0       0          0
## H-1             0        0        1077         18       0          0
## H-2             0      550       26612       3000       0          0
## I-2           410      456        3657       1126      56         30
## JL           3575      231        3544       5110    1026          0
## K-2             0        0           0          0      80          0
## K-3             3        0           0          0       0          0
## L-2           160        0         153          0       0          0
## L-3             0        0         760          0       0          0
## L-4             0        0           0          0       0          0
## L-5             0        0           0          0       0          0
## M-1          5439       60        2697       3085    3626        195
## M-2             0      471        4418        168       0          0
## N-1             0        0       60685          0       0          0
## P-4             0        0         133         56       0          0
## P-5          4691      353        2352       5340    9502          0
## PC              0        0        1294          0       0          0
## PD            414       50           0        191     388          0
## R-4             0        0           0          0       0          0
## RC            224      459        1130        332     350          0
## S-2           794      170         190        269       0          0
## S-3             0        0         326        278       0          0
## T-2          1242        0        2636        762       0        122
## T-4             0        0        3667          0       0          0
## T-5           262      100        5302       2542       0          0
## T-6             0       72        1661          0       0          0
## T-9             0        0           0          1       1          0
## TGT             0        0       15045          0       0          0
## V-1             0        0           0          0     228          0
##       Powerloom Jacquard INDO TEBETAN
## A-11                  0            0
## A-6                   0            0
## A-9                   0            0
```

```
## B-2                          0             0
## B-3                          0             0
## B-4                          0             0
## C-1                          0             0
## C-2                          0             0
## C-3                          0             0
## CC                           0             0
## CTS                          0             0
## DR                           0             0
## E-2                          0             0
## F-1                          0             0
## F-6                          0             0
## G-1                          0             0
## G-4                          0             0
## H-1                          0             0
## H-2                       9753             0
## I-2                          0             0
## JL                           0             0
## K-2                          0             0
## K-3                          0             0
## L-2                          0             0
## L-3                          0             0
## L-4                          0             0
## L-5                          0             0
## M-1                          0             0
## M-2                          0             0
## N-1                          0             0
## P-4                          0             0
## P-5                          0             0
## PC                           0             0
## PD                           0            20
## R-4                          0             0
## RC                           0             0
## S-2                          0             0
## S-3                          0             0
## T-2                          0            12
## T-4                          0             0
## T-5                          0             0
## T-6                          0             0
## T-9                          0             0
## TGT                          0             0
## V-1                          0             0

#Ceckinh Missing and Boxplots
plot_missing(dfc)
```



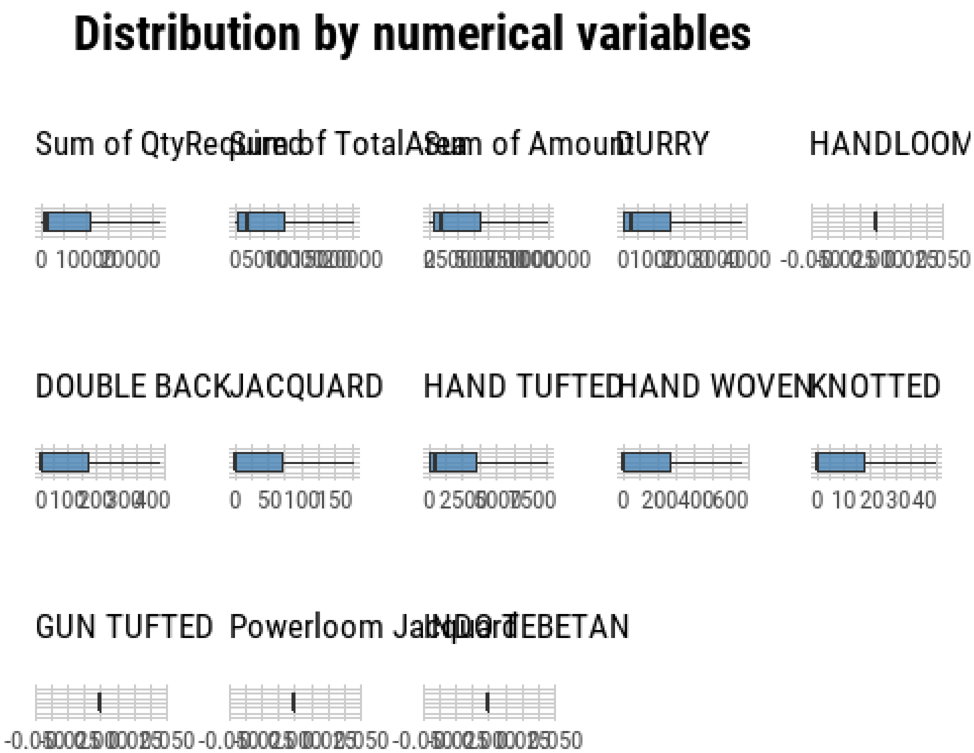

```
plot_box_numeric(dfc)
```



```r
#Checking Normality
normality(dfc)
```

```
## # A tibble: 13 x 4
##    vars                 statistic  p_value sample
##    <chr>                    <dbl>    <dbl>  <dbl>
##  1 Sum of QtyRequired       0.456 1.07e-11     45
##  2 Sum of TotalArea         0.402 2.65e-12     45
##  3 Sum of Amount            0.397 2.38e-12     45
##  4 DURRY                    0.350 7.54e-13     45
##  5 HANDLOOM                 0.348 7.20e-13     45
##  6 DOUBLE BACK              0.395 2.23e-12     45
##  7 JACQUARD                 0.585 4.55e-10     45
##  8 HAND TUFTED              0.390 1.98e-12     45
##  9 HAND WOVEN               0.410 3.29e-12     45
## 10 KNOTTED                  0.261 1.03e-13     45
## 11 GUN TUFTED               0.260 1.00e-13     45
## 12 Powerloom Jacquard       0.135 8.06e-15     45
## 13 INDO TEBETAN             0.214 3.82e-14     45
```

```r
# Capping Outliers since they'll affect during distance calculation

for (i in 1:13){
  #treating upperbound outlier
  ub <- quantile(dfc[[i]], 0.75) + IQR(dfc[[i]])*1.5
  dfc[[i]][dfc[[i]]>ub] <- ub

  #treating lower bound outliers
  lb <- quantile(dfc[[i]], 0.25) - IQR(dfc[[i]])*1.5
  dfc[[i]][dfc[[i]]<lb] <- lb
}

#Boxplots after treating Outliers
plot_box_numeric(dfc)
```




```r
#dropping columns with zero varianace after outlier treatment
dfc <- dfc[-c(5,11,12,13)]

dfc

##       Sum of QtyRequired Sum of TotalArea Sum of Amount DURRY DOUBLE BACK
## A-11              2466.0         139.5900   185404.1000  1021         0.0
## A-6                131.0        2086.0000     6247.4600     0        25.0
## A-9              18923.0       20563.7527  1007013.2610  3585       175.0
## B-2                624.0         202.8987    14811.1591   581         0.0
## B-3                464.0        8451.5625    58626.8650     0       437.5
## B-4                692.0        3244.2500    26242.5000    80         0.0
## C-1               5137.0       20563.7527   567620.7210   288         0.0
## C-2              27017.5        9510.0000  1007013.2610  3900         0.0
## C-3               1566.0        4016.0000    99062.3500  1240         0.0
## CC                5077.0        7695.9930   147574.0000     4         3.0
## CTS                565.0         420.0000    23800.0000     0         0.0
## DR                 149.0         305.9765    28648.1165     0        16.0
## E-2                581.0       18878.0000   116778.3000    13       348.0
## F-1               1158.0        2822.0000   116838.2000   288        64.0
## F-6               1400.0           1.3500    16800.0000  1400         0.0
## G-1                146.0        5348.0000    39701.2400     0        52.0
## G-4                119.0          21.9352      328.8752   119         0.0
## H-1               1137.0        9327.0625    65383.7950    39         0.0
## H-2              27017.5       19505.3958  1007013.2610  3900         0.0
## I-2               7501.0        1508.6320   426626.0484   978       410.0
## JL               18861.0        2980.6500  1007013.2610  3900       437.5
```

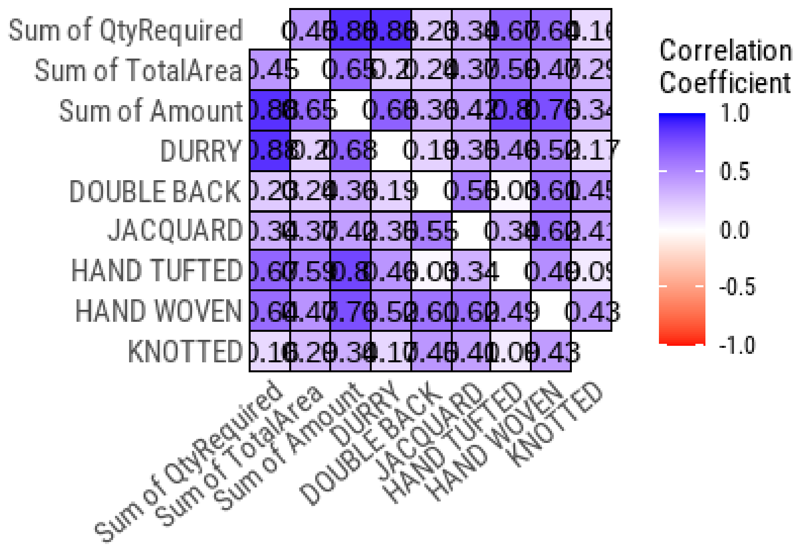
```
## K-2              438.0         3852.0790    59877.2660   358          0.0
## K-3                3.0           80.6666      409.9995     0          3.0
## L-2              313.0           81.9400    21503.4950     0        160.0
## L-3              760.0         1721.0000    90756.7500     0          0.0
## L-4              776.0            7.3600    44234.0000   776          0.0
## L-5            25840.0          210.0000   358890.0000  3900          0.0
## M-1            16649.0        20563.7527  1007013.2610   412        437.5
## M-2             6926.0         8200.3959   334245.2238  1869          0.0
## N-1            27017.5          919.6505   949375.6758  3900          0.0
## P-4            16653.0         1834.0000   292544.4500  3900          0.0
## P-5            27017.5        20563.7527  1007013.2610  3900        437.5
## PC              1294.0         8781.0625   227949.5550     0          0.0
## PD             11146.0          725.0137   404528.9455  3900        414.0
## R-4              175.0           48.4000    10108.8000   175          0.0
## RC              3022.0         1898.1906   328290.7475   527        224.0
## S-2             1712.0          528.8725    56742.7300   289        437.5
## S-3              604.0         1800.0000    61368.0000     0          0.0
## T-2             5468.0         2434.7624   563098.8478   299        437.5
## T-4             5677.0         2811.3750   238241.0000  1560          0.0
## T-5            27017.5         9221.3750   733832.9500  3900        262.0
## T-6             1737.0         2120.0000   101488.0000     4          0.0
## T-9                2.0           17.2800      758.9700     0          0.0
## TGT            15045.0        20563.7527  1007013.2610     0          0.0
## V-1              447.0          376.7690    47761.2800   219          0.0
##       JACQUARD HAND TUFTED HAND WOVEN KNOTTED
## A-11         0           0        0.0       0
## A-6        106           0        0.0       0
## A-9        180        8860      672.5      45
## B-2          2           0       41.0       0
## B-3          5           0        0.0       0
## B-4          0         510        0.0       0
## C-1          0        4176      220.0      45
## C-2          0        3816      672.5       0
## C-3          0         326        0.0       0
## CC           0        5021        0.0       0
## CTS          0         565        0.0       0
## DR           6          13        0.0      45
## E-2        151           0       51.0      18
## F-1          0         806        0.0       0
## F-6          0           0        0.0       0
## G-1         68           0       26.0       0
## G-4          0           0        0.0       0
## H-1          0        1077       18.0       0
## H-2        180        8860      672.5       0
## I-2        180        3657      672.5      45
## JL         180        3544      672.5      45
## K-2          0           0        0.0      45
## K-3          0           0        0.0       0
## L-2          0         153        0.0       0
## L-3          0         760        0.0       0
```

```
## L-4          0          0       0.0      0
## L-5          0          0       0.0      0
## M-1         60       2697     672.5     45
## M-2        180       4418     168.0      0
## N-1          0       8860       0.0      0
## P-4          0        133      56.0      0
## P-5        180       2352     672.5     45
## PC           0       1294       0.0      0
## PD          50          0     191.0     45
## R-4          0          0       0.0      0
## RC         180       1130     332.0     45
## S-2        170        190     269.0      0
## S-3          0        326     278.0      0
## T-2          0       2636     672.5      0
## T-4          0       3667       0.0      0
## T-5        100       5302     672.5      0
## T-6         72       1661       0.0      0
## T-9          0          0       1.0      1
## TGT          0       8860       0.0      0
## V-1          0          0       0.0     45
```

```
# Min-Max Scaling
min_max<-function(x){(x-min(x))/(max(x)-min(x))}

dfc_ns <- dfc
dfc<-dfc%>%
  mutate_if(is.numeric,min_max)
```

```
# Checking Multicollinearity
plot_correlate(dfc)
```

```
## Warning: 'plot_correlate' is deprecated.
## Use 'plot.correlate' instead.
## See help("Deprecated")
```



We can see that 'Sum of Amount' has high correlation with other variables so we can drop it and proceed with clustering.

5 Clustering algoeithns

We've decided to build K-means, agglomerative and heirarchical model to identify different clusters in the dataset.

For K-means we'll get the optimal cluster number with the help of scree plot and silhouette score, the elbow point in scree plot and highest sil score gives the best cluster.

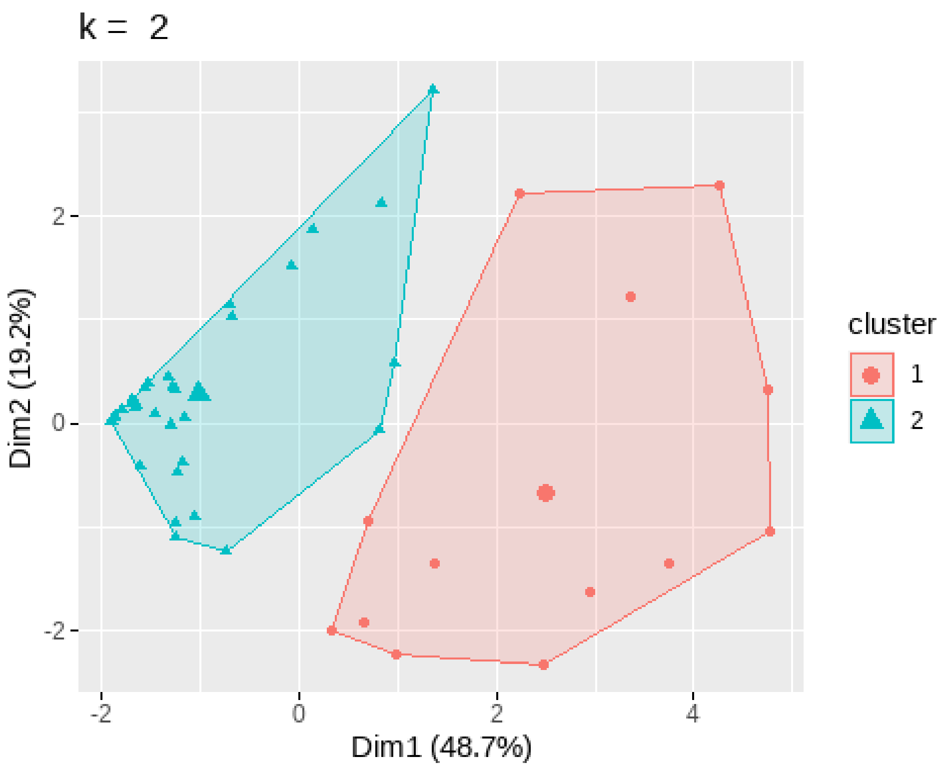
For agglomerative and heirarchical we'll use Ward metric, and divide the customers based on dendogram length/optimal value selected from kmeans.

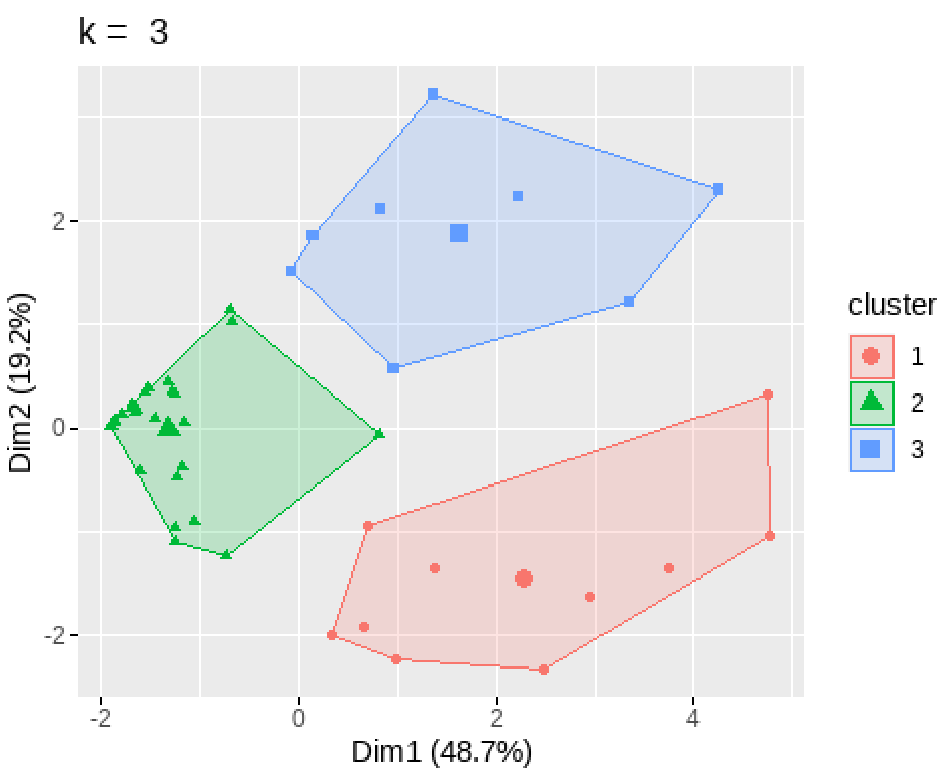
6 Building Clustering algorithms

6.1 K-Means

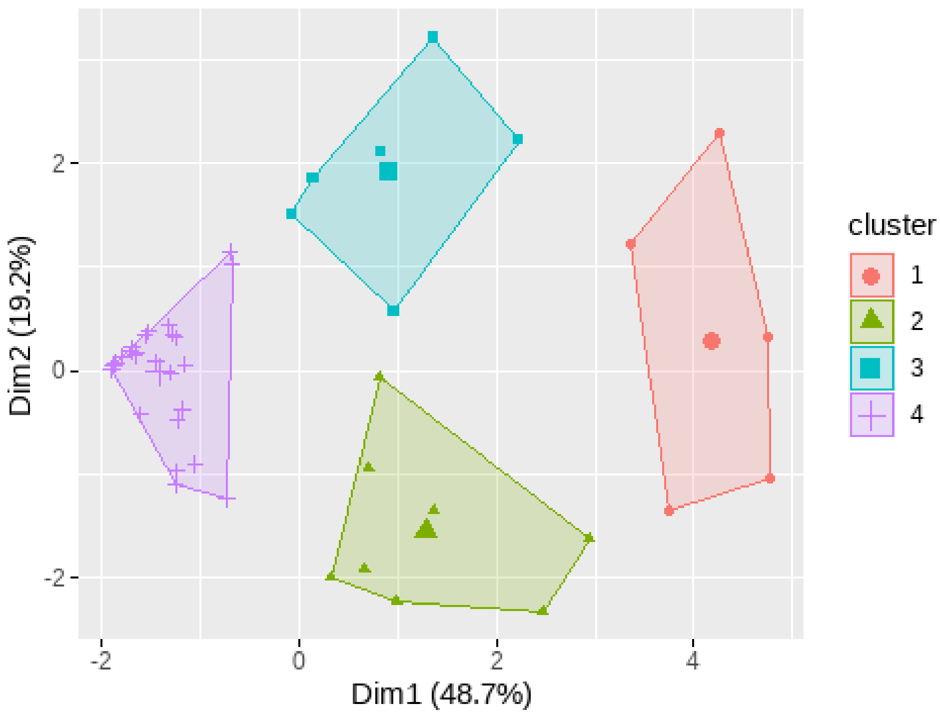
```
dfc <- dfc[,-3]  #dropping Sum of Amounts

k_range<-2:10
KM<-c()
for (i in k_range){
  km<-kmeans(dfc, center=i,nstart=100)
```

```
  KM<-append(KM,km)
  print(fviz_cluster(km,geom = "point",data=dfc)+ggtitle(paste("k = ",i)))
}
```



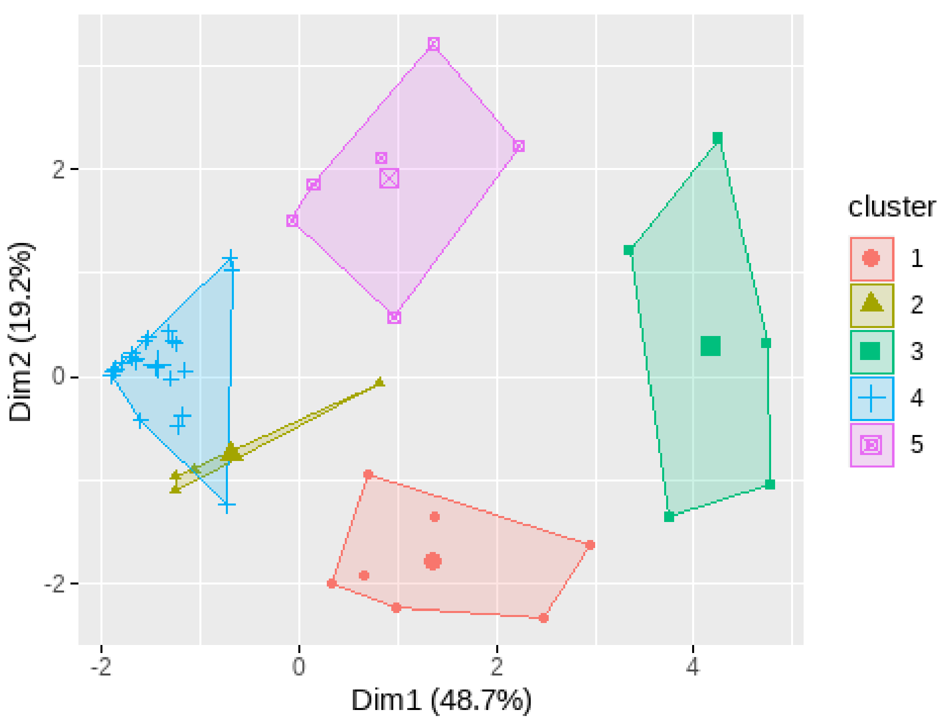


k = 4

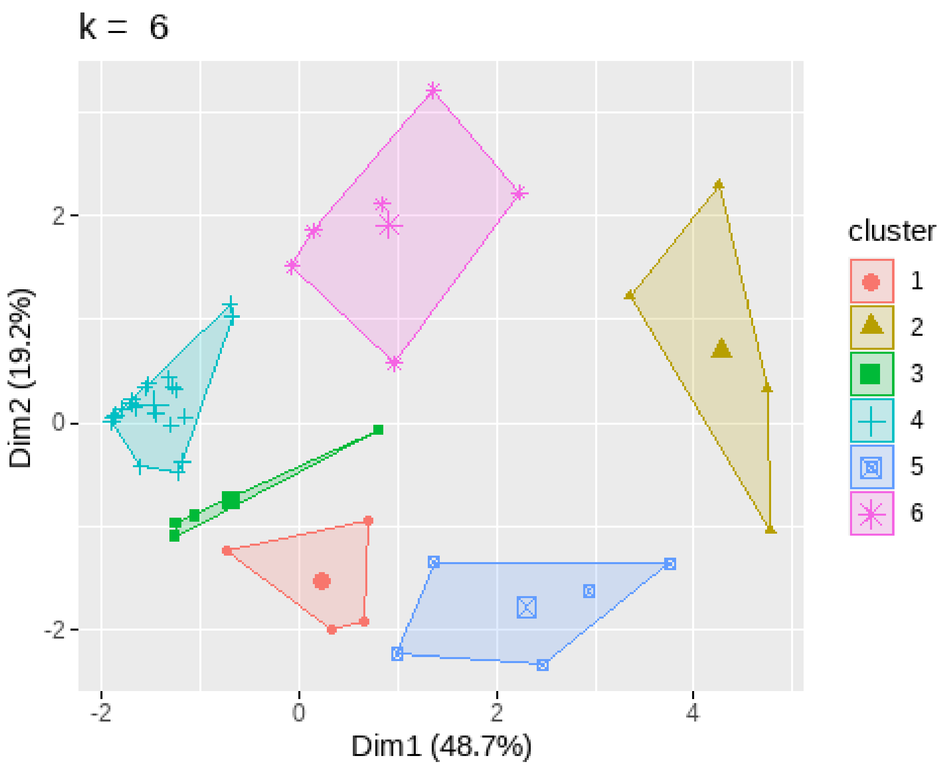


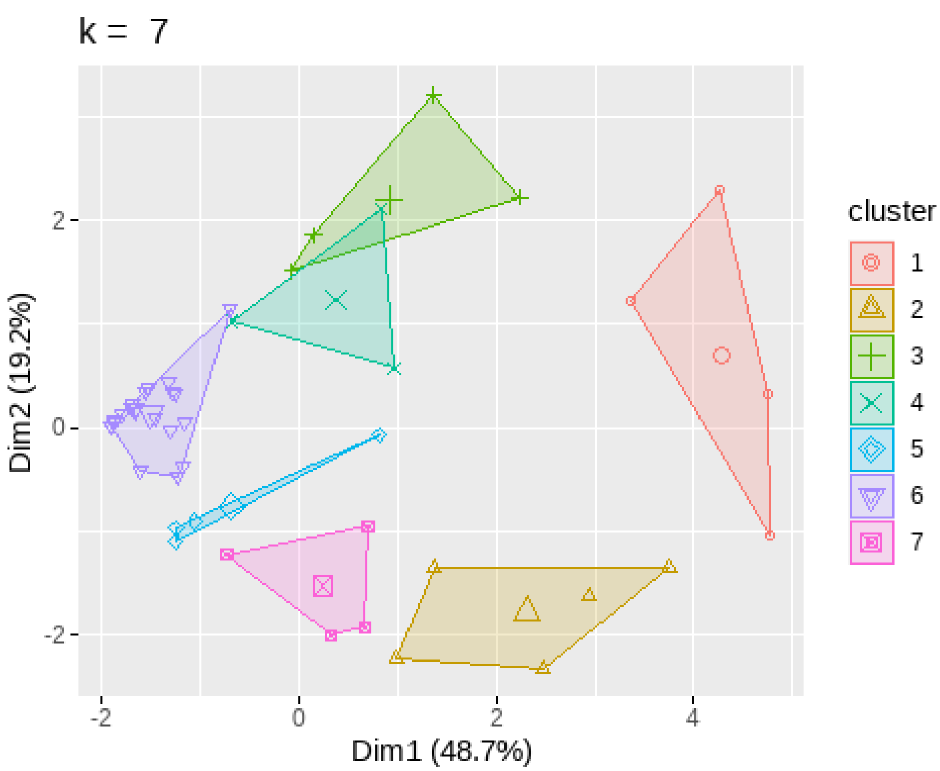
cluster
1
2
3
4
Dim1 (48.7%)
Dim2 (19.2%)
-2
0
2
4

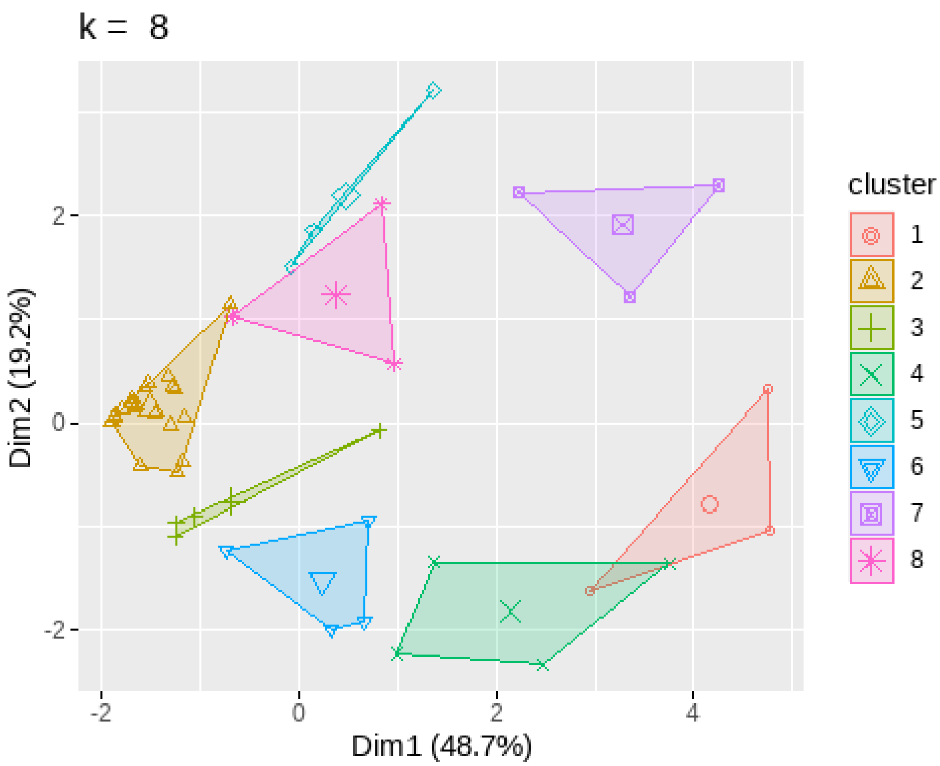
k = 5

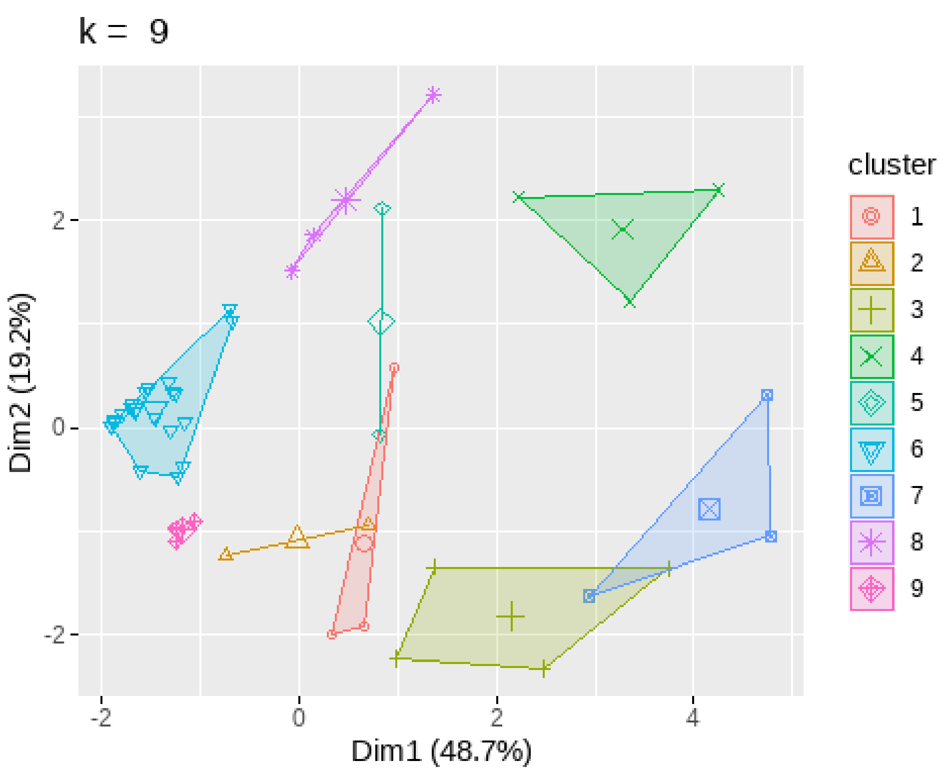


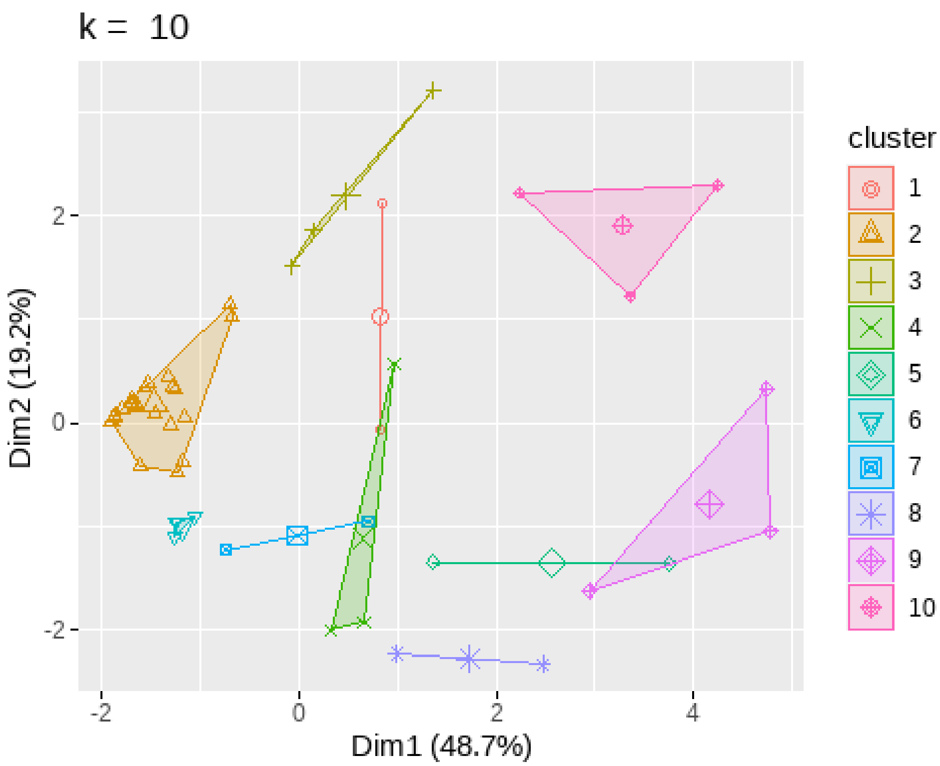
cluster
1
2
3
4
5
Dim1 (48.7%)
Dim2 (19.2%)
-2
0
2
4


k = 6
Dim2 (19.2%)
Dim1 (48.7%)
cluster
1
2
3
4
5
6
-2
0
2
4

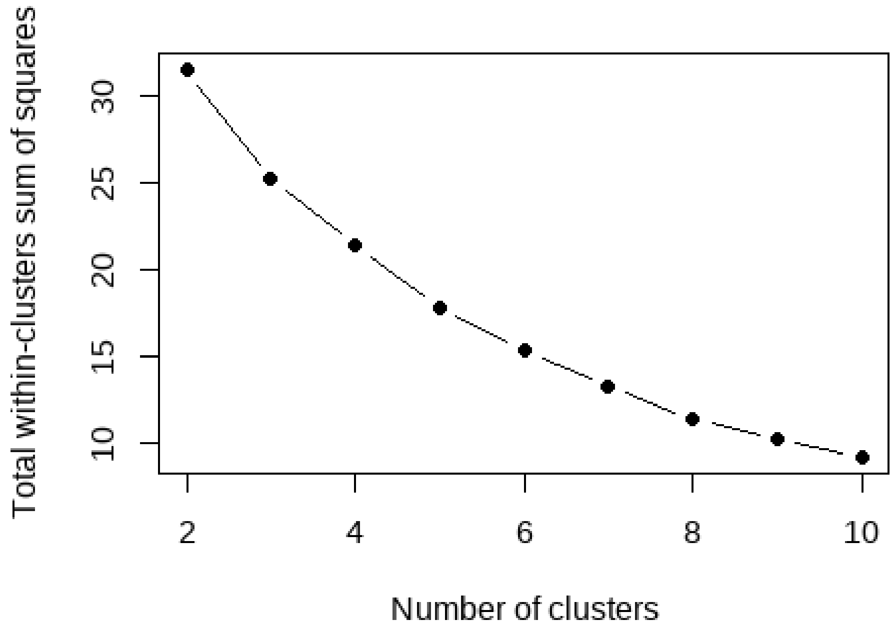

k = 7
Dim2 (19.2%)
Dim1 (48.7%)
cluster
1
2
3
4
5
6
7
-2
0
2
4


k = 8
Dim2 (19.2%)
Dim1 (48.7%)
-2
0
2
4
cluster
1
2
3
4
5
6
7
8

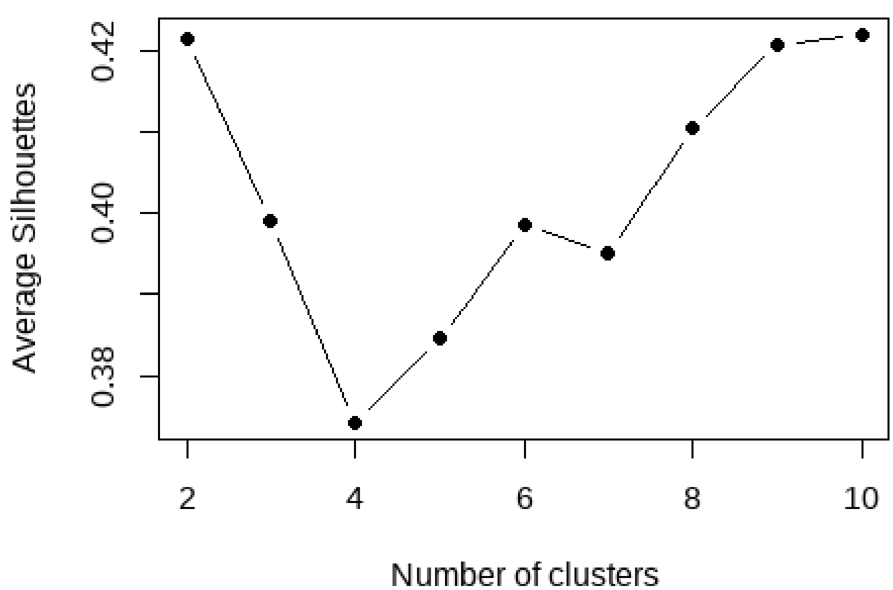

k = 9
Dim2 (19.2%)
Dim1 (48.7%)
-2
0
2
4
cluster
1
2
3
4
5
6
7
8
9



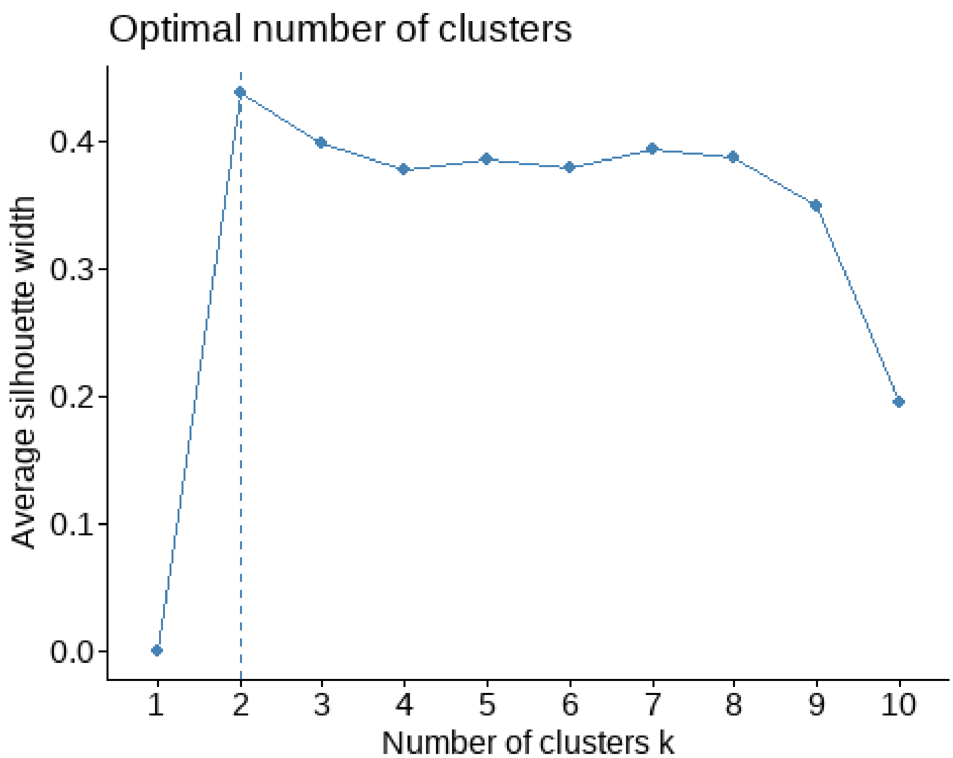

```
# Finding Optimal Clusters

wss<-function(k){kmeans(dfc,centers = k,nstart=100)$tot.withinss}
sil<-function(k){
  kmmodel<-kmeans(dfc,centers=k,nstart=100)
  s<-silhouette(kmmodel$cluster,dist(dfc))
  mean(s[,3])
}

kmmodel<-kmeans(dfc,centers=k,nstart=100)
s<-silhouette(kmmodel$cluster,dist(dfc))

WSS<-map_dbl(k_range,wss)
SIL<-map_dbl(k_range,sil)

plot(k_range, WSS,type="b",pch=19,xlab="Number of clusters",
ylab="Total within-clusters sum of squares")
```



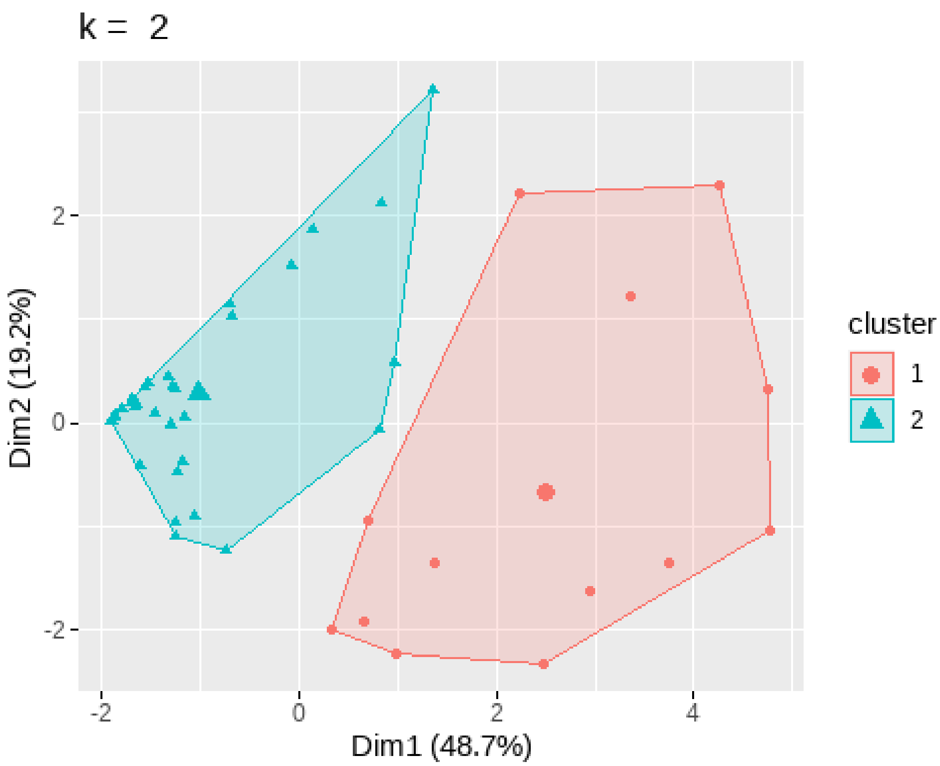

```
plot(k_range, SIL,type="b",pch=19,xlab = "Number of clusters",
ylab = "Average Silhouettes")
```

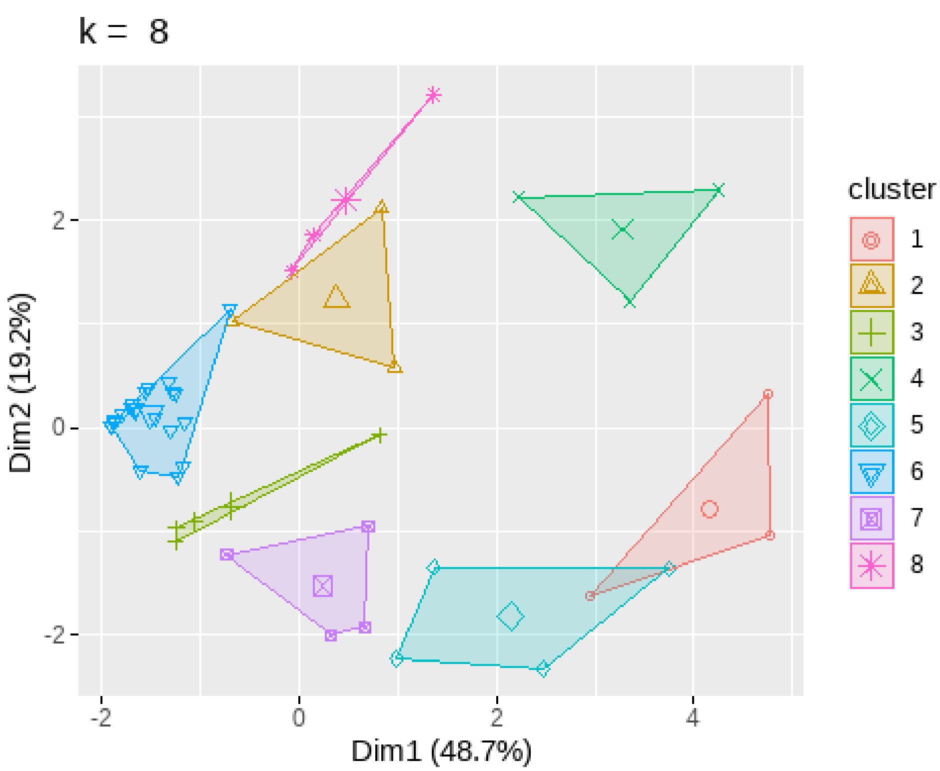


```
fviz_nbclust(dfc, kmeans, method = "silhouette")
```




```
for (i in c(2,8)){
  km<-kmeans(dfc, center=i,nstart=100)
  print(fviz_cluster(km,geom = "point",data=dfc)+ggtitle(paste("k = ",i)))
}
```






```
km8<-kmeans(dfc, center=8,nstart=100)
```

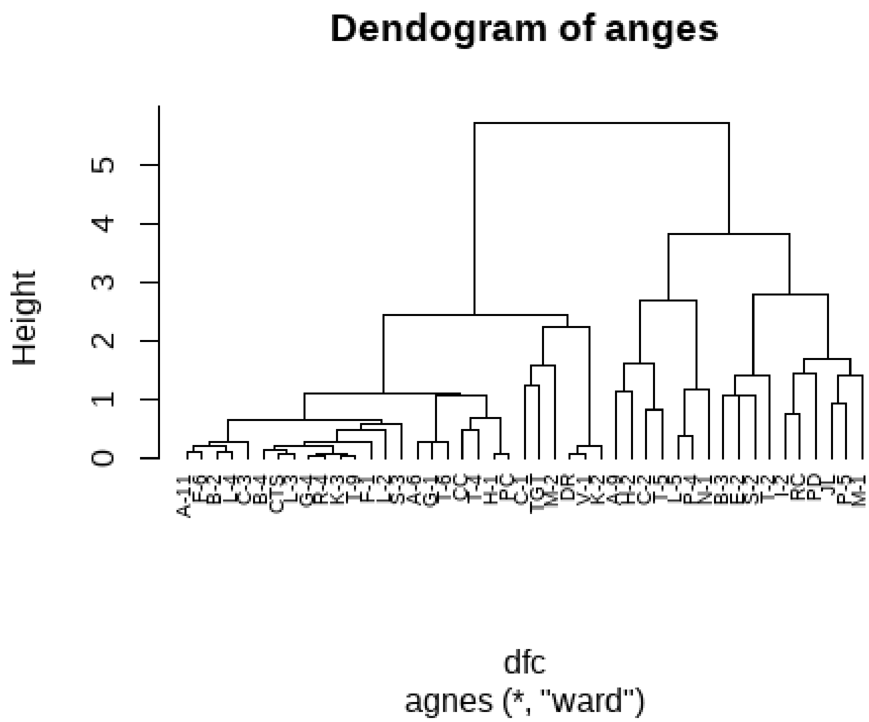
Based on our obtained Silhouette score 2 and 8 clusters seems to be the best option, the above plot display 2 and 8 clusters of the given data.

```
dfc_ns %>%
  mutate(Cluster = km8$cluster) %>%
  group_by(Cluster) %>%
  summarise_all("mean")

## # A tibble: 8 x 10
##   Cluster `Sum of QtyRequi~ `Sum of TotalAr~ `Sum of Amount` DURRY `DOUBLE
BACK`
##     <int>             <dbl>            <dbl>           <dbl> <dbl>
<dbl>
## 1       1             2056.            7573.         198812.  150.
415.
## 2       2             9016            12153.         496277.  624.
1
## 3       3            23170.             988.         533603. 3900
0
## 4       4            10132.            1778.         541615. 2326.
371.
## 5       5             1543.            6275.         175977.  216.
4
## 6       6             1016.            2148.          66259.  347.
14.5
## 7       7            27018.           12746.         915953. 3900
87.3
## 8       8            20863.           20564.        1007013. 2632.
350
## # ... with 4 more variables: JACQUARD <dbl>, HAND TUFTED <dbl>,
## #   HAND WOVEN <dbl>, KNOTTED <dbl>
```

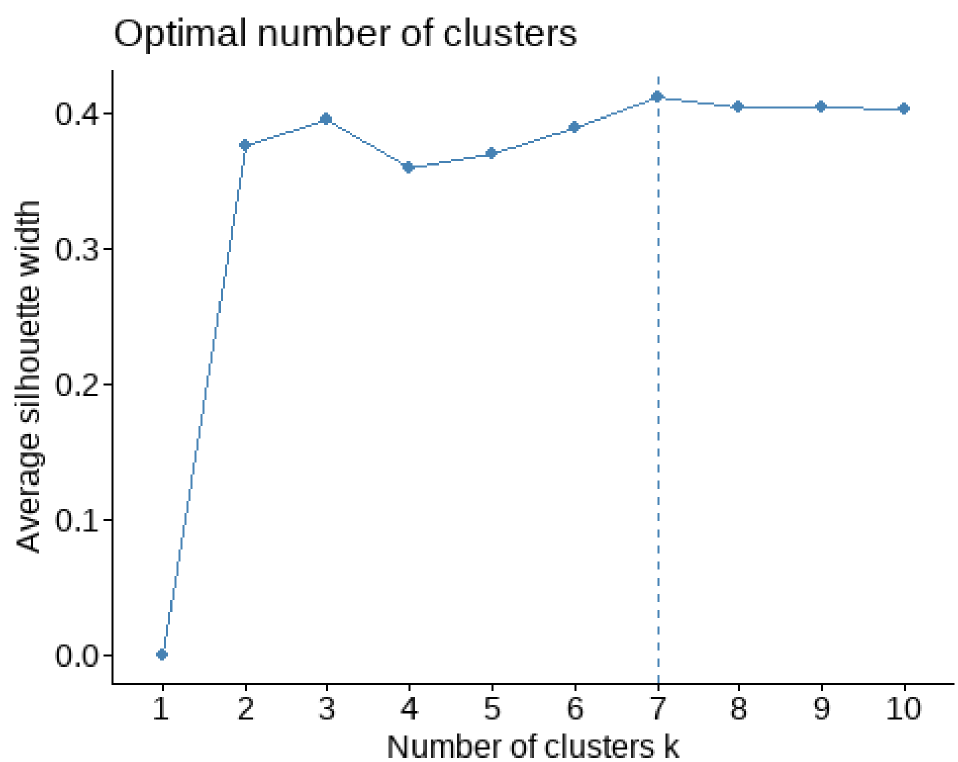
The above shows the mean for all variables for the 8 clusters. (Non scaled)

6.2 Agglomerative & Heirarchical

```
method<-c("average","single","complete","ward")

ac<-c()
for (i in 1:4){
  hc <- agnes(dfc, method=method[i])
  ac <- append(ac, hc$ac)
  print(c(method[i], hc$ac))
}

## [1] "average"           "0.730943305852239"
## [1] "single"            "0.594902765987097"
## [1] "complete"          "0.789118969442747"
## [1] "ward"              "0.902373846052513"

pltree(hc,cex=0.6,hang=-2,main="Dendogram of anges")
```

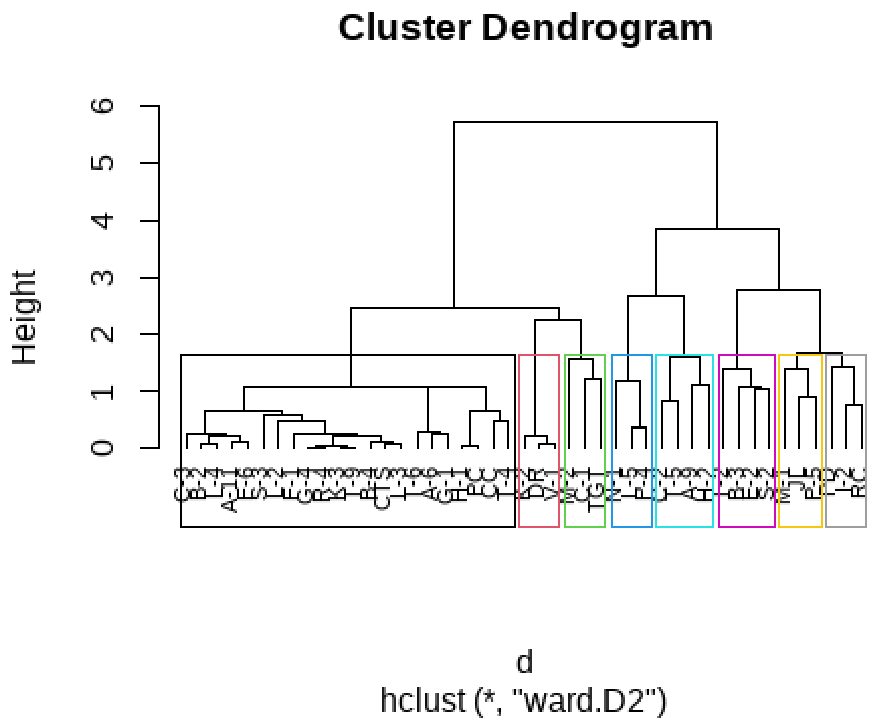



```
fviz_nbclust(dfc,FUN=hcut,method="silhouette")
```



We can observe that Ward gives us the best performance hence, we'll choose that.

```
d<-dist(dfc,method = "euclidean")

dfc_clust <- hclust(d, method = "ward.D2")

plot(dfc_clust, cex=0.7, hang=-2)
rect.hclust(dfc_clust, k=8, border=1:8)
```



We can see different clusters based on the above plot The observed cluseters are: Cluster 8 : PD, I-2, RC Cluster 7 : M-1, JL, P-5 Cluster 6 : T-2, B-3, E-2 , S-2 and so on.. Hence these customers have similar buying habbits.

7 Association

We used association to identify what items to reccomend to a person based on his past purchases.

```
tr <- read.transactions("C:/Users/rshara4/Downloads/AssociativeDataset.csv",
format = "basket", sep=",", skip = 0, cols = 1)

## Warning in asMethod(object): removing duplicated items in transactions

inspect(head(tr,5))

##     items          transactionID
## [1] {DURRY,
##      HANDLOOM}              A-11
## [2] {DOUBLE BACK,
```

```
##      JACQUARD}              A-6
## [3] {DOUBLE BACK,
##      DURRY,
##      HAND TUFTED,
##      HAND WOVEN,
##      JACQUARD,
##      KNOTTED}               A-9
## [4] {DURRY,
##      HAND WOVEN,
##      JACQUARD}              B-2
## [5] {DOUBLE BACK,
##      JACQUARD}              B-3
```

```
frequentItems <- eclat(tr,
parameter = list(supp=0.07, maxlen=15))
```

```
## Eclat
##
## parameter specification:
##  tidLists support minlen maxlen            target  ext
##     FALSE    0.07      1     15 frequent itemsets TRUE
##
## algorithmic control:
##  sparse sort verbose
##       7   -2    TRUE
##
## Absolute minimum support count: 3
##
## create itemset ...
## set transactions ...[10 item(s), 45 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating bit matrix ... [8 row(s), 45 column(s)] done [0.00s].
## writing  ... [127 set(s)] done [0.00s].
## Creating S4 object  ... done [0.00s].
```

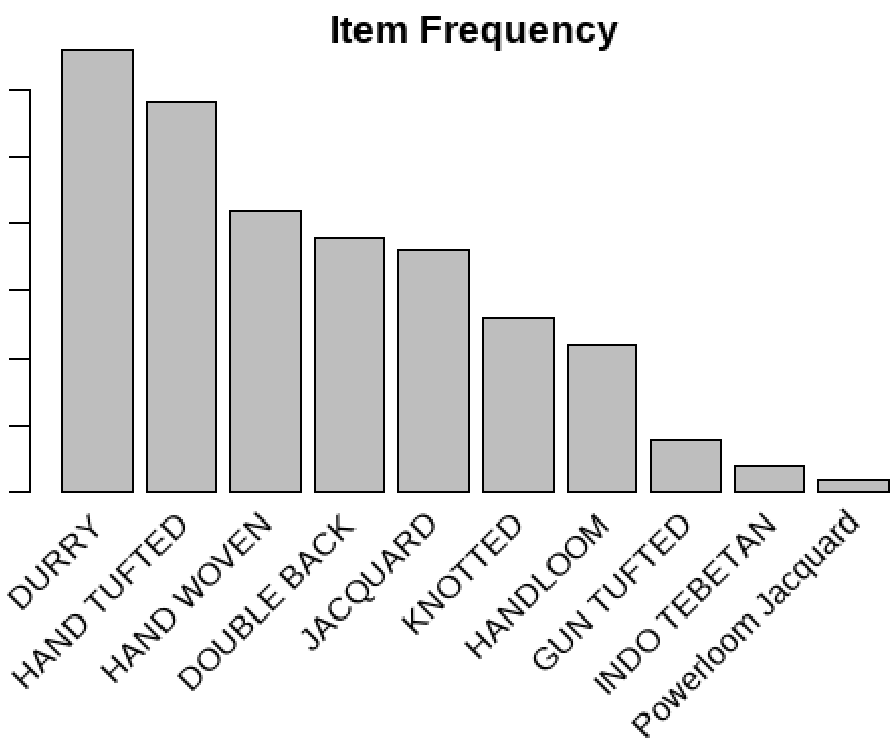
```
inspect(head(frequentItems,10))
```

```
##      items             support count
## [1]  {DOUBLE BACK,
##       DURRY,
##       GUN TUFTED,
##       HAND TUFTED,
##       HANDLOOM}     0.08888889     4
## [2]  {DOUBLE BACK,
##       DURRY,
##       GUN TUFTED,
##       HANDLOOM}     0.08888889     4
## [3]  {DOUBLE BACK,
##       GUN TUFTED,
##       HAND TUFTED,
##       HANDLOOM}     0.08888889     4
```

```
## [4]  {DURRY,
##       GUN TUFTED,
##       HAND TUFTED,
##       HANDLOOM}     0.08888889     4
## [5]  {DURRY,
##       GUN TUFTED,
##       HANDLOOM}     0.08888889     4
## [6]  {GUN TUFTED,
##       HAND TUFTED,
##       HANDLOOM}     0.08888889     4
## [7]  {DOUBLE BACK,
##       GUN TUFTED,
##       HANDLOOM}     0.08888889     4
## [8]  {DOUBLE BACK,
##       DURRY,
##       GUN TUFTED,
##       HAND TUFTED}  0.08888889     4
## [9]  {DOUBLE BACK,
##       DURRY,
##       GUN TUFTED}   0.08888889     4
## [10] {DOUBLE BACK,
##       GUN TUFTED,
##       HAND TUFTED}  0.08888889     4
```

We inspect all the transactions and find the most frequently bought together items with a minimum support of 0.07

```
par(mar=c(1,1,1,1))
itemFrequencyPlot(tr, topN=10, type="absolute",
main="Item Frequency")
```



Exploring all the association rules

```
rules <- apriori(tr, parameter = list(supp = 0.001, conf = 0.5, maxlen=3))

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.5    0.1    1 none FALSE            TRUE       5   0.001      1
##  maxlen target  ext
##       3  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 0
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[10 item(s), 45 transaction(s)] done [0.00s].
## sorting and recoding items ... [10 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3

## Warning in apriori(tr, parameter = list(supp = 0.001, conf = 0.5, maxlen =
3)):
## Mining stopped (maxlen reached). Only patterns up to a length of 3
returned!
```

```
##  done [0.00s].
## writing ... [249 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
#Sorting it based on highest lift
rules_lift <- sort (rules, by="lift", decreasing=TRUE)
inspect(rules_lift[1:10])
```

```
##      lhs                                rhs          support    confidence
## [1]  {HAND TUFTED, INDO TEBETAN}     => {GUN TUFTED} 0.02222222 1.0000000
## [2]  {DOUBLE BACK, HANDLOOM}         => {GUN TUFTED} 0.08888889 0.5714286
## [3]  {INDO TEBETAN}                  => {GUN TUFTED} 0.02222222 0.5000000
## [4]  {HANDLOOM, INDO TEBETAN}        => {GUN TUFTED} 0.02222222 0.5000000
## [5]  {DOUBLE BACK, INDO TEBETAN}     => {GUN TUFTED} 0.02222222 0.5000000
## [6]  {HAND WOVEN, INDO TEBETAN}      => {GUN TUFTED} 0.02222222 0.5000000
## [7]  {DURRY, INDO TEBETAN}           => {GUN TUFTED} 0.02222222 0.5000000
## [8]  {HANDLOOM, KNOTTED}             => {GUN TUFTED} 0.04444444 0.5000000
## [9]  {Powerloom Jacquard}            => {HANDLOOM}   0.02222222 1.0000000
## [10] {INDO TEBETAN}                  => {HANDLOOM}   0.04444444 1.0000000
##      coverage   lift      count
## [1]  0.02222222 11.250000 1
## [2]  0.15555556  6.428571 4
## [3]  0.04444444  5.625000 1
## [4]  0.04444444  5.625000 1
## [5]  0.04444444  5.625000 1
## [6]  0.04444444  5.625000 1
## [7]  0.04444444  5.625000 1
## [8]  0.08888889  5.625000 2
## [9]  0.02222222  4.090909 1
## [10] 0.04444444  4.090909 2
```

```r
rules_conf <- sort (rules, by="confidence", decreasing=TRUE)
# show the support, lift and confidence for all rules
inspect(rules_conf[1:10])
```

```
##      lhs                     rhs           support    confidence coverage
## [1]  {Powerloom Jacquard} => {HANDLOOM}    0.02222222 1
0.02222222
## [2]  {Powerloom Jacquard} => {JACQUARD}    0.02222222 1
0.02222222
## [3]  {Powerloom Jacquard} => {HAND WOVEN}  0.02222222 1
0.02222222
## [4]  {Powerloom Jacquard} => {HAND TUFTED} 0.02222222 1
0.02222222
## [5]  {Powerloom Jacquard} => {DURRY}       0.02222222 1
0.02222222
## [6]  {INDO TEBETAN}       => {HANDLOOM}    0.04444444 1
0.04444444
## [7]  {INDO TEBETAN}       => {DOUBLE BACK} 0.04444444 1
0.04444444
## [8]  {INDO TEBETAN}       => {HAND WOVEN}  0.04444444 1
```

```
0.04444444
## [9]  {INDO TEBETAN}       => {DURRY}       0.04444444 1
0.04444444
## [10] {GUN TUFTED}         => {HANDLOOM}    0.08888889 1
0.08888889
##      lift     count
## [1]  4.090909 1
## [2]  2.500000 1
## [3]  2.142857 1
## [4]  1.551724 1
## [5]  1.363636 1
## [6]  4.090909 2
## [7]  2.368421 2
## [8]  2.142857 2
## [9]  1.363636 2
## [10] 4.090909 4
```

Finding the rules related to given products

```
# Get rules that lead to buying 'Jacquard'
rules <- apriori (data=tr,
parameter=list(supp=0.001, conf=0.08),
appearance= list(default="lhs",rhs="JACQUARD"),control = list(verbose=F))

# 'high-confidence' rules
rules_conf <- sort (rules, by="confidence", decreasing=TRUE)
inspect(head(rules_conf))

##     lhs                                  rhs        support    confidence
## [1] {Powerloom Jacquard}              => {JACQUARD} 0.02222222 1
## [2] {HANDLOOM, Powerloom Jacquard}    => {JACQUARD} 0.02222222 1
## [3] {HAND WOVEN, Powerloom Jacquard}  => {JACQUARD} 0.02222222 1
## [4] {HAND TUFTED, Powerloom Jacquard} => {JACQUARD} 0.02222222 1
## [5] {DURRY, Powerloom Jacquard}       => {JACQUARD} 0.02222222 1
## [6] {INDO TEBETAN, KNOTTED}           => {JACQUARD} 0.02222222 1
##     coverage   lift count
## [1] 0.02222222 2.5  1
## [2] 0.02222222 2.5  1
## [3] 0.02222222 2.5  1
## [4] 0.02222222 2.5  1
## [5] 0.02222222 2.5  1
## [6] 0.02222222 2.5  1
```

Those who bought ‘Double Back’ also bought

```
rules <- apriori (data=tr, parameter=list (supp=0.001,conf = 0.15,minlen=2),
appearance = list(default = "rhs", lhs = "DOUBLE BACK"), control =
list(verbose=F))

#Listing the rules with highest lift for the condition
rules_conf <- sort (rules, by="confidence", decreasing=TRUE)
inspect(head(rules_conf))
```

```
##     lhs               rhs            support   confidence coverage  lift
## [1] {DOUBLE BACK} => {JACQUARD}    0.3111111 0.7368421  0.4222222
1.8421053
## [2] {DOUBLE BACK} => {HAND TUFTED} 0.2888889 0.6842105  0.4222222
1.0617060
## [3] {DOUBLE BACK} => {DURRY}       0.2888889 0.6842105  0.4222222
0.9330144
## [4] {DOUBLE BACK} => {HAND WOVEN}  0.2666667 0.6315789  0.4222222
1.3533835
## [5] {DOUBLE BACK} => {KNOTTED}     0.2000000 0.4736842  0.4222222
1.6396761
## [6] {DOUBLE BACK} => {HANDLOOM}    0.1555556 0.3684211  0.4222222
1.5071770
##     count
## [1] 14
## [2] 13
## [3] 13
## [4] 12
## [5]  9
## [6]  7
```

8 Reccomendation to Champo carpets

With the help of the models built the company can identify all the important attributes/variables which affect the sample conversion rate. Some Important variable are : AreaFT, QtyRequired, CountryName and cetrain ItemTypes.

With the help of clustering the company can gain knowledge about the segements or clusters present in the dataset. In other words it'll help identify the different types of customers with similar buying habits. This will enable them to form better strategies and focus more on those who are likely to convert.

The association rules which we devised are as follows: We have now converted the dataset into transactions for further analysis When buying Hand Tufted and Indo Tibetian, there is a 12 likelehood for buyers to also purchase Gun Tufted. Simlarly, there is a 4 time more likelehood of buying Handloom when Poweloom Jacquard or Indo Tibetian is purchased We see a high support for Handloom when Gun Tufted is purchased. Recommendation would be to target customers buying them with Handloom products We can infer from the suggestion methods what items can be marketed together. In the example in associative section, we have explored the best products that would likely sell when we sell Double back and Jacquard. The same can be extended for all the products that needs similar analysis.