Data Science Challenge - Capital One

Tarun Kateja

22 November 2019

```
rm(list = ls(all.names = T))
```

Loading required libraries.

```
library(dplyr)
library(tidyr)
library(RJSONIO)
library(jsonlite)
library(tidyverse)
library(data.table)
library(ggplot2)
library(gridExtra)
library(lubridate)
library(car)
library(caret)
library(ranger)
library(rpart)
library(rpart.plot)
library(ROSE)
library(ROCR)
library(pROC)
```

1) Data Downloading and Loading Environment

Downloading and loading the df in R environment programatically.

```
temp <- tempfile()
download.file("https://raw.githubusercontent.com/CapitalOneRecruiting/DS/master/transactions.zip",temp)

raw_df <- stream_in(unz(temp, "transactions.txt"), verbose = F)
```

2) Data Exploration

Dimensions of the raw df are as: 641914 observations and 29 features.

```
dim(raw_df)
```

```
## [1] 641914     29
```

Checking the types of variables available in the df.

```
var_type <- sapply(raw_df, class)

num_var <- colnames(raw_df[sapply(raw_df,is.numeric)])
cat_var <- colnames(raw_df[sapply(raw_df,is.character)])
log_var <- colnames(raw_df[sapply(raw_df,is.logical)])
```

There are total 29 variables in the raw df provided with 4 numerical variables and 22 categorical variables.

Calculating the missing rate (% of missing observations for each column).

```
raw_df[raw_df==""]<-NA
misvars <- data.frame(t(data.frame(map(raw_df, ~mean(is.na(.))))))
misvars <- setDT(misvars, keep.rownames = TRUE)[]
colnames(misvars) <- c("var_name", "missrate")
misvars[order(missrate, decreasing = TRUE),]
```

```
##                     var_name     missrate
##  1:               echoBuffer 1.0000000000
##  2:             merchantCity 1.0000000000
##  3:            merchantState 1.0000000000
##  4:              merchantZip 1.0000000000
##  5:            posOnPremises 1.0000000000
##  6:         recurringAuthInd 1.0000000000
##  7:               acqCountry 0.0060958322
##  8:             posEntryMode 0.0052109784
##  9:      merchantCountryCode 0.0009720928
## 10:          transactionType 0.0009175684
## 11:         posConditionCode 0.0004471004
## 12:            accountNumber 0.0000000000
## 13:               customerId 0.0000000000
## 14:              creditLimit 0.0000000000
## 15:           availableMoney 0.0000000000
## 16:      transactionDateTime 0.0000000000
## 17:        transactionAmount 0.0000000000
## 18:             merchantName 0.0000000000
## 19:     merchantCategoryCode 0.0000000000
## 20:           currentExpDate 0.0000000000
## 21:          accountOpenDate 0.0000000000
## 22:  dateOfLastAddressChange 0.0000000000
## 23:                  cardCVV 0.0000000000
## 24:               enteredCVV 0.0000000000
## 25:          cardLast4Digits 0.0000000000
## 26:                  isFraud 0.0000000000
## 27:           currentBalance 0.0000000000
## 28:              cardPresent 0.0000000000
## 29: expirationDateKeyInMatch 0.0000000000
##                     var_name     missrate
```

We can see there are 6 columns with 100% missing information and hence removing those!

Removing variables with 100% missing. And, creating df to work with…

```
drop <- misvars[misvars$missrate == 1,]$var_name
df <- raw_df[,!(names(raw_df)%in%drop)]
cat_var <- setdiff(cat_var, drop)
log_var <- setdiff(log_var, drop)
num_var <- setdiff(num_var, drop)
```

There are only few variables have missing data with < 1% missing observation. However, removing this small chunk might not be a good idea!

Let's check how many actual (isFraud == TRUE) are there for these variables.

```
df_check = df[!complete.cases(df),]
dim(df_check[df_check$isFraud == "TRUE",])
```

```
## [1] 410  23
```

There are 410 of these 8068 observations with missing, which are fraudulent. Which is approximately 5% of these observations. Knowing event rate (isFraud == TRUE) 1.7%. Its not good idea to drop these observations.

Basic statistics of numerical columns.

```
summary.data.frame(df[,num_var])
```

```
##   creditLimit    availableMoney  transactionAmount currentBalance
##  Min.   :  250   Min.   :-1245   Min.   :   0.00   Min.   :    0.0
##  1st Qu.: 5000   1st Qu.: 1115   1st Qu.:  32.32   1st Qu.:  502.4
##  Median : 7500   Median : 3578   Median :  85.80   Median : 2151.9
##  Mean   :10697   Mean   : 6653   Mean   : 135.16   Mean   : 4044.4
##  3rd Qu.:15000   3rd Qu.: 8169   3rd Qu.: 189.03   3rd Qu.: 5005.9
##  Max.   :50000   Max.   :50000   Max.   :1825.25   Max.   :47496.5
```

Looking at summary we can comment that most attributes are right skewed which we can further see in plots!

Number of unique values in each categorical column.

```
uniVal <- df[,cat_var] %>% summarise_each(funs(n_distinct))
```

```
## Warning: funs() is soft deprecated as of dplyr 0.8.0
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once per session.
```

```
uniVal
```

```
##   accountNumber customerId transactionDateTime merchantName acqCountry
## 1          5000       5000              635472         2493          5
##   merchantCountryCode posEntryMode posConditionCode merchantCategoryCode
## 1                   5            6                4                   19
##   currentExpDate accountOpenDate dateOfLastAddressChange cardCVV
## 1            165            1826                    2186     899
##   enteredCVV cardLast4Digits transactionType
## 1        980            5134               4
```

The columns which identify record uniquely will of-course be high. To note: 5 countries as acqCountry and merchantCountryCode Also, the card transactions are on 19 broad industries (merchantCategoryCode)

Lets check how many of above variables have less than 20 categories!

```
uniVal20 <- uniVal[,uniVal<20]
uniVal20
```

```
##   acqCountry merchantCountryCode posEntryMode posConditionCode
## 1          5                   5            6                4
##   merchantCategoryCode transactionType
## 1                   19               4
```

Frequency distribution of categorical columns with unique values less than 20.

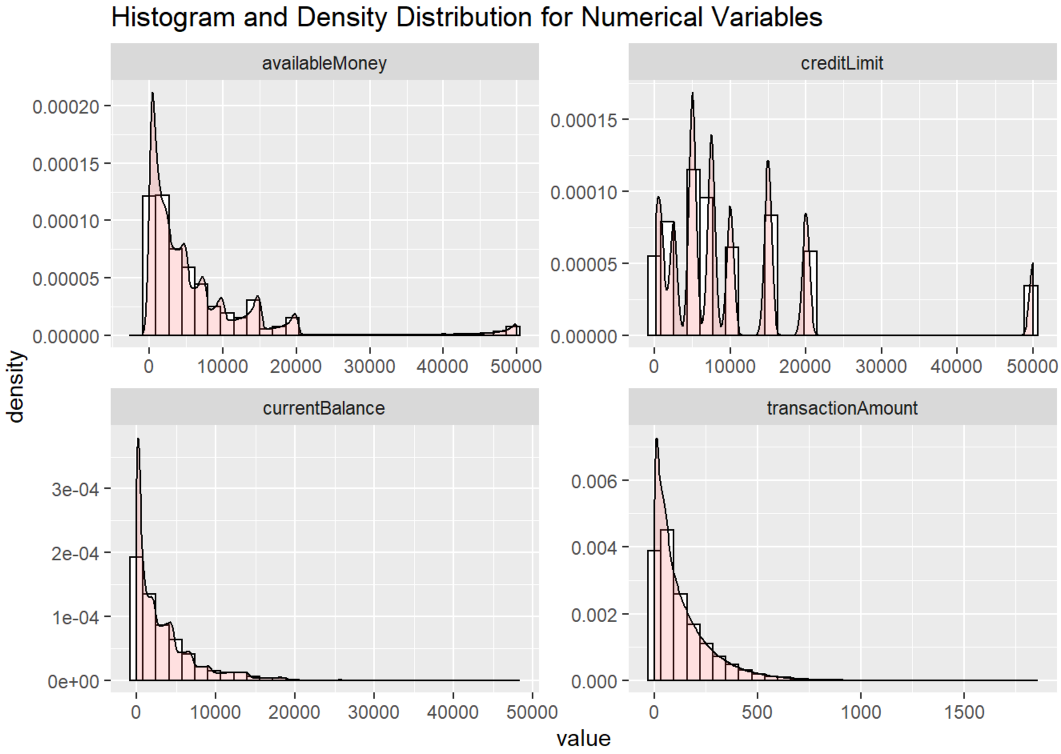
```
freq <-function(x){

  y <- data.frame(t(table(x)))
  y$Var1 <-NULL
  names(y) <- c("Levels","Freq")
  return(y)
}

freq_cat <- do.call(rbind, apply(df[ ,colnames(uniVal20)], 2, freq))
freq_cat
```

```
##                                           Levels   Freq
## acqCountry.1                                 CAN   1870
## acqCountry.2                                 MEX   2626
## acqCountry.3                                  PR   1202
## acqCountry.4                                  US 632303
## merchantCountryCode.1                        CAN   1874
## merchantCountryCode.2                        MEX   2636
## merchantCountryCode.3                         PR   1203
## merchantCountryCode.4                         US 635577
## posEntryMode.1                                02 160589
## posEntryMode.2                                05 255615
## posEntryMode.3                                09 193193
## posEntryMode.4                                80  12921
## posEntryMode.5                                90  16251
## posConditionCode.1                            01 514144
## posConditionCode.2                            08 121507
## posConditionCode.3                            99   5976
## merchantCategoryCode.1                   airline   9990
## merchantCategoryCode.2                      auto  10147
## merchantCategoryCode.3               cable/phone   1490
## merchantCategoryCode.4             entertainment  69138
## merchantCategoryCode.5                  fastfood 101196
## merchantCategoryCode.6                      food  68245
## merchantCategoryCode.7             food_delivery   4990
## merchantCategoryCode.8                      fuel  22566
## merchantCategoryCode.9                 furniture   7813
## merchantCategoryCode.10                      gym   2874
## merchantCategoryCode.11                   health  14344
## merchantCategoryCode.12                   hotels  22879
## merchantCategoryCode.13               mobileapps  14614
## merchantCategoryCode.14             online_gifts  33045
## merchantCategoryCode.15            online_retail 161469
## merchantCategoryCode.16 online_subscriptions  11247
## merchantCategoryCode.17            personal care  16917
## merchantCategoryCode.18                rideshare  50574
## merchantCategoryCode.19            subscriptions  18376
## transactionType.1       ADDRESS_VERIFICATION  16478
## transactionType.2                   PURCHASE 608685
## transactionType.3                   REVERSAL  16162
```

We can see majority is US in acqCountry. Therefore this data has many customers from US. Online retail, fastfood dominates in card transactions! Maybe they have maximum frauds! we will see in further sections.

Histogram and density distribution for numerical variables.

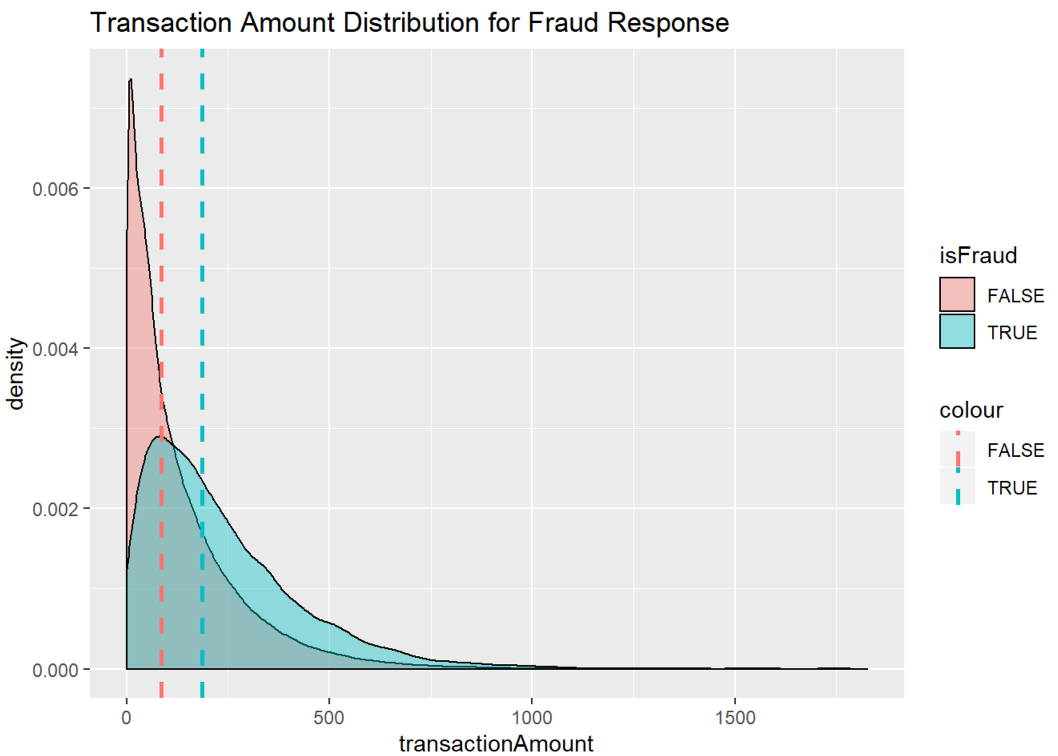
```
df[,num_var] %>% gather() %>% ggplot(aes(value)) + facet_wrap(~ key, scales = "free") +
  geom_histogram(aes(y=..density..), colour="black", fill="white") + geom_density(alpha=.2, fill="#FF6666")
+
  labs(title = paste("Histogram and Density Distribution for Numerical Variables"))
```



As we thought, the numericall columns are rightly skewed with median less than mean! It will be interesting to see Fraud behavior in this. Probably the fraudulent transactions will have higher mean and median than not fraudulent.

Plot of transaction amount distribution with respect to fraud nature!

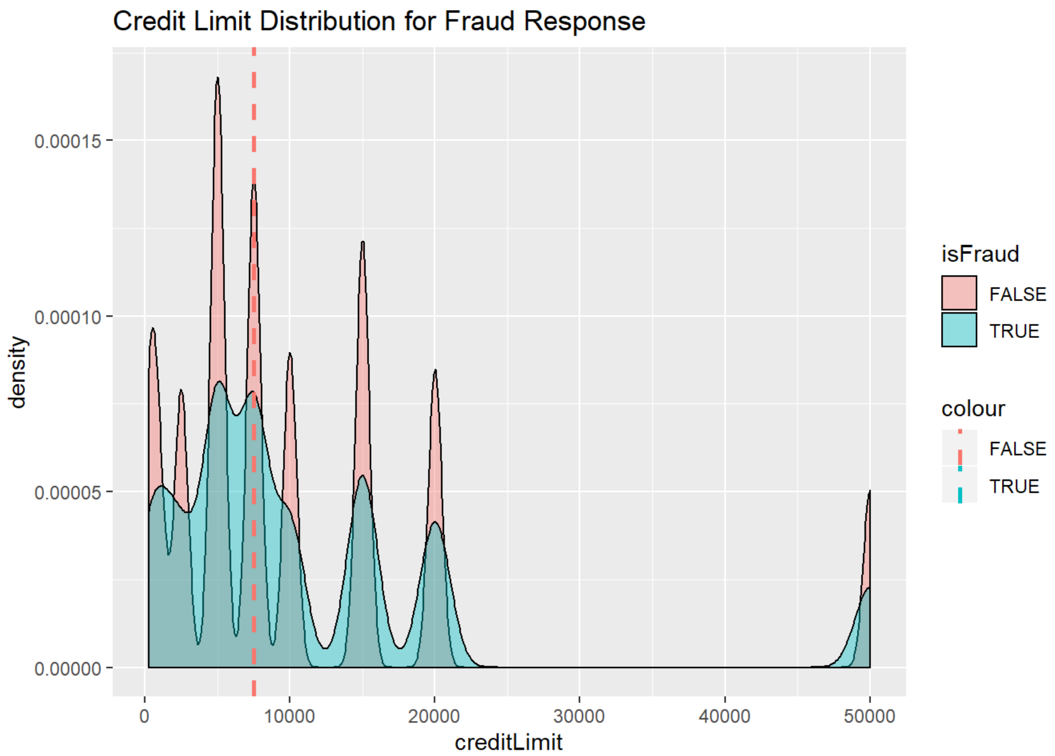
```
var_nplot <- c(num_var, "isFraud")

ggplot(data = df[,var_nplot], mapping = aes(x = transactionAmount)) + geom_density(aes(fill = isFraud), alpha = 0.4) +
  geom_vline(aes(xintercept = median(transactionAmount[isFraud == TRUE]), color = TRUE), linetype="dashed", size=1) +
  geom_vline(aes(xintercept = median(transactionAmount[isFraud == FALSE]), color = FALSE), linetype="dashed", size=1) +
  labs(title = "Transaction Amount Distribution for Fraud Response")
```



It is clear from above plot that median transaction amount for fraudulent transaction is higher than for not fraudulent transactions. And hence we can make a *Null Hypothesis* that transaction amount is not strong predictor for fraudulent nature. We can try to calculate p value for this test and for p value less than 0.5, we reject the Null Hypothesis saying transaction amount is strong predictor for fraudulent nature which is apparent from its distribution.

Plot of credit limit distribution with respect to fraud nature!

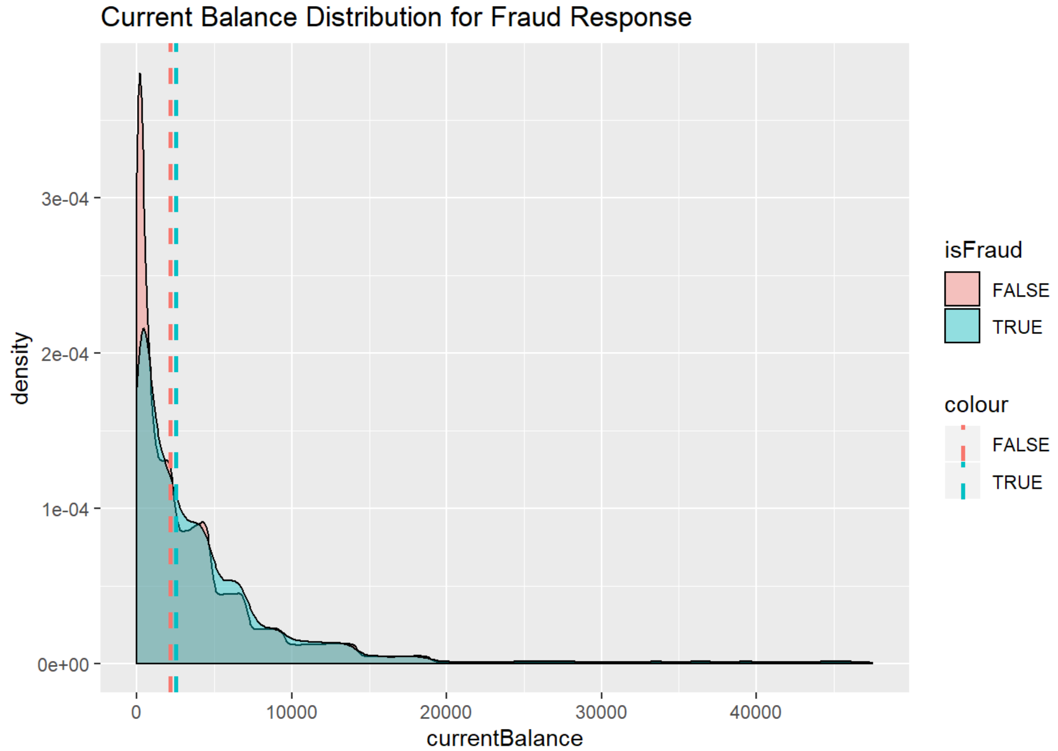
```
ggplot(data = df[,var_nplot], mapping = aes(x = creditLimit)) + geom_density(aes(fill = isFraud), alpha = 0.
4) +
  geom_vline(aes(xintercept = median(creditLimit[isFraud == TRUE]), color = TRUE), linetype="dashed", size=1
) +
  geom_vline(aes(xintercept = median(creditLimit[isFraud == FALSE]), color = FALSE), linetype="dashed", size
=1) +
  labs(title = "Credit Limit Distribution for Fraud Response")
```



This looks not so strong predictor!

Plot of current balance distribution with respect to fraud nature!

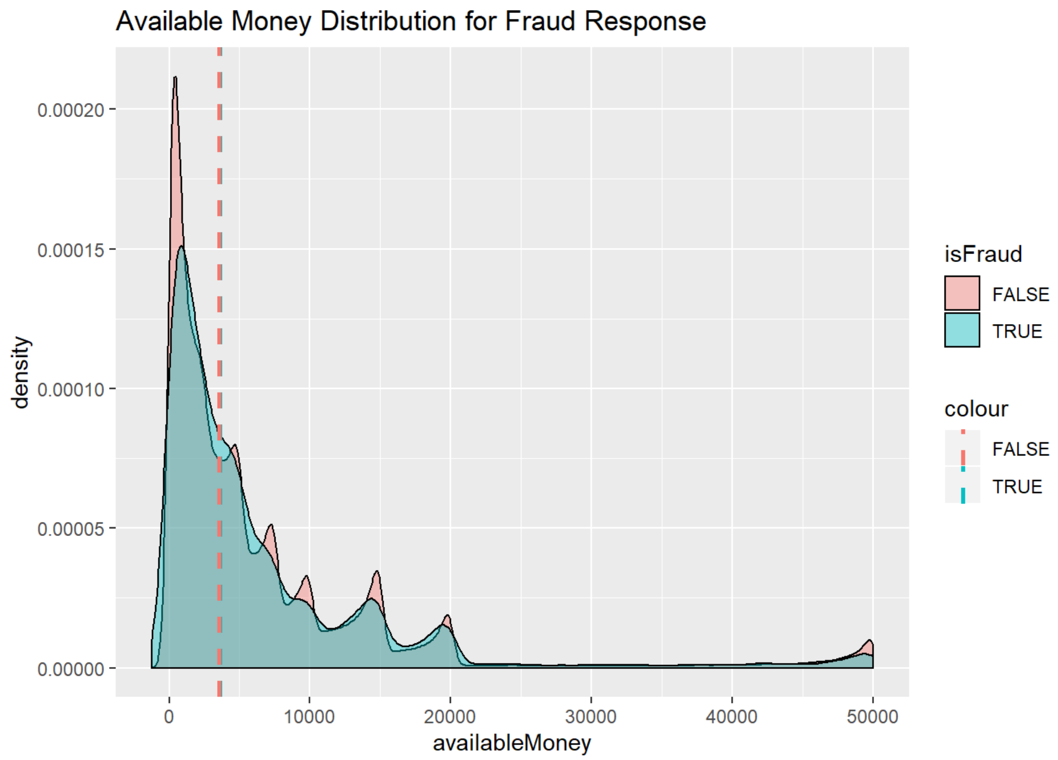
```
ggplot(data = df[,var_nplot], mapping = aes(x = currentBalance)) + geom_density(aes(fill = isFraud), alpha =
0.4) +
  geom_vline(aes(xintercept = median(currentBalance[isFraud == TRUE]), color = TRUE), linetype="dashed", siz
e=1) +
  geom_vline(aes(xintercept = median(currentBalance[isFraud == FALSE]), color = FALSE), linetype="dashed", s
ize=1) +
  labs(title = "Current Balance Distribution for Fraud Response")
```



This is not a strong predictor!

Plot of available money with respect to fraud nature!

```
ggplot(data = df[,var_nplot], mapping = aes(x = availableMoney)) + geom_density(aes(fill = isFraud), alpha =
0.4) +
  geom_vline(aes(xintercept = median(availableMoney[isFraud == TRUE]), color = TRUE), linetype="dashed", siz
e=1) +
  geom_vline(aes(xintercept = median(availableMoney[isFraud == FALSE]), color = FALSE), linetype="dashed", s
ize=1) +
  labs(title = "Available Money Distribution for Fraud Response")
```



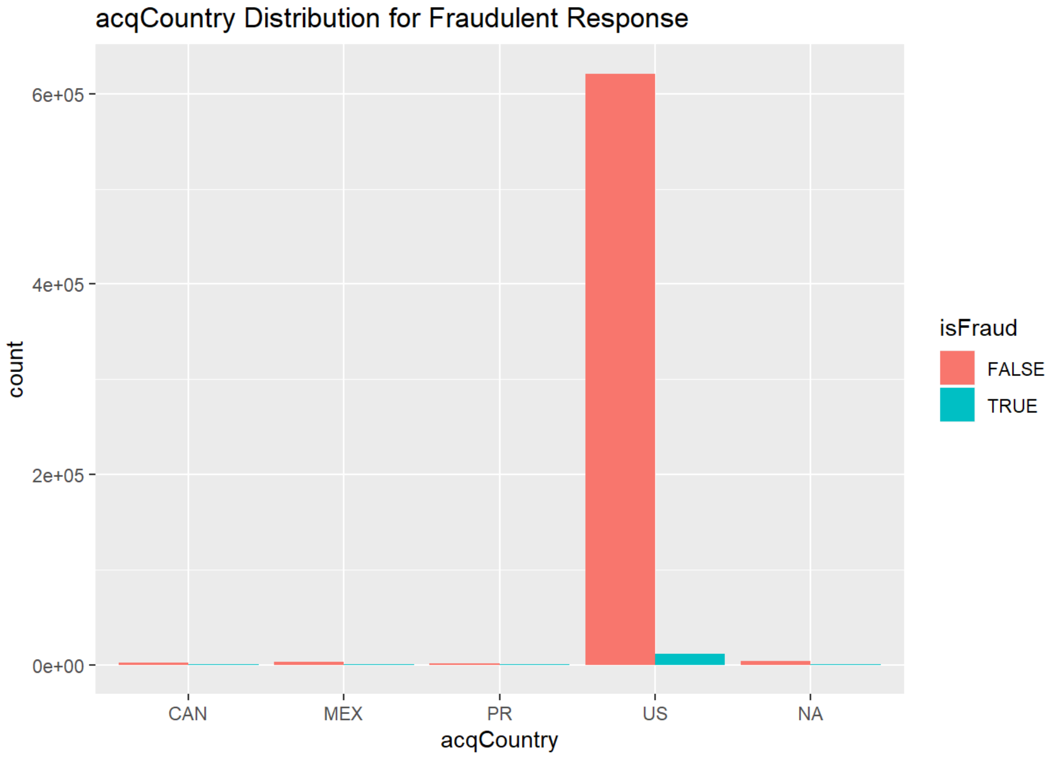
Lets check which categories have maximum fraud!

Bar plots for categorical columns with less than 20 unique values with respect to Fraud Response.

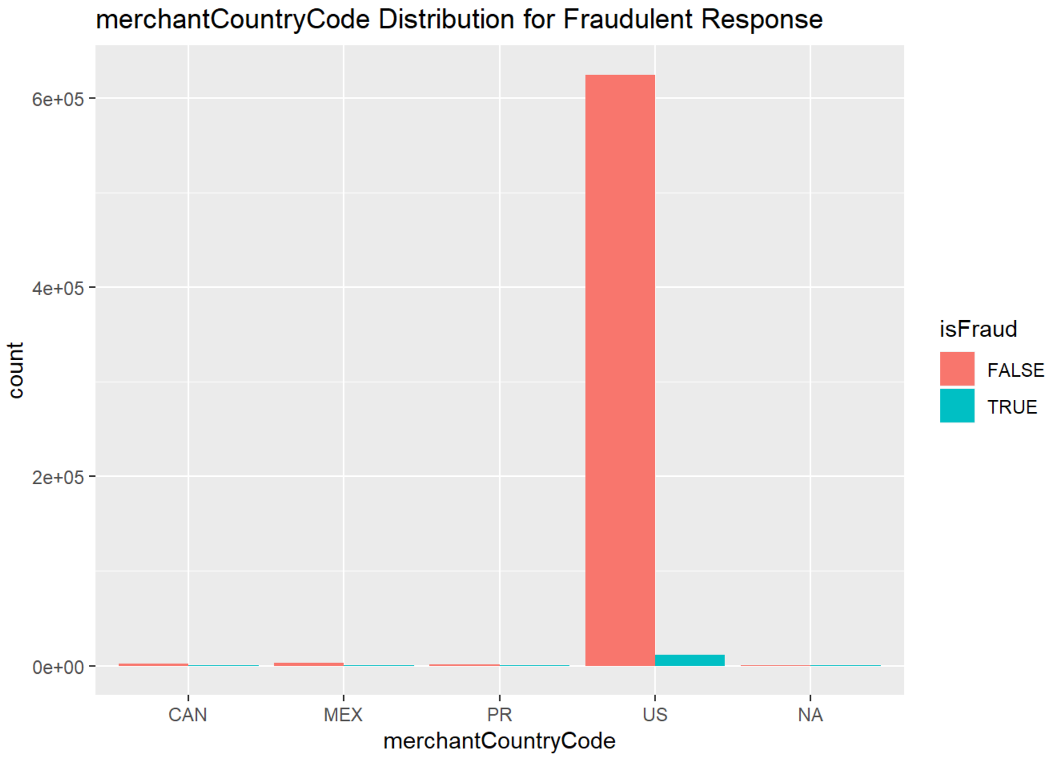
```
colnames(unival20)
```

```
## [1] "acqCountry"           "merchantCountryCode"  "posEntryMode"
## [4] "posConditionCode"     "merchantCategoryCode" "transactionType"
```

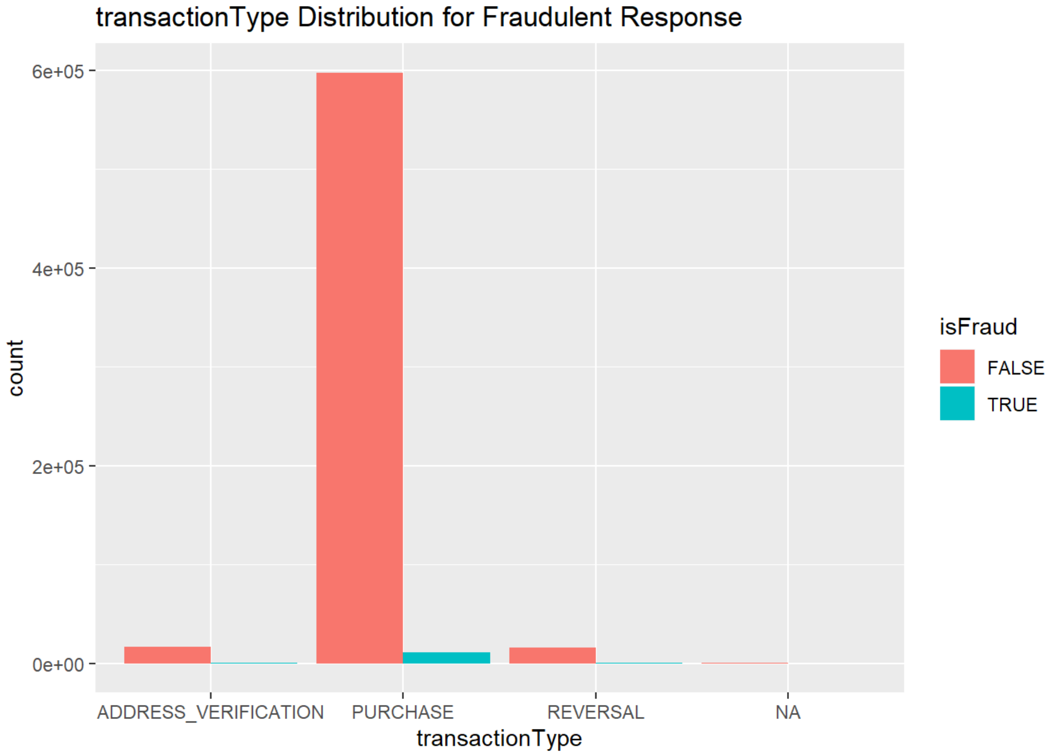
```
ggplot(df, aes(acqCountry, ..count..)) + geom_bar(aes(fill = isFraud), position = "dodge") +
labs(title = "acqCountry Distribution for Fraudulent Response")
```



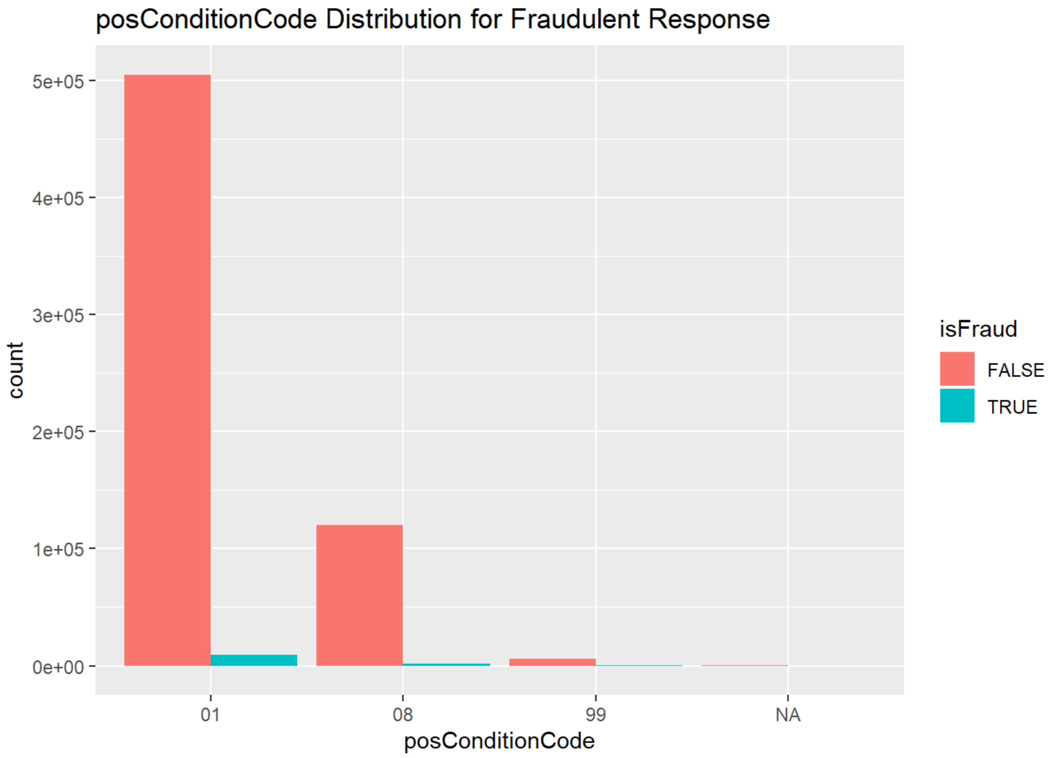

```
ggplot(df, aes(merchantCountryCode, ..count..)) + geom_bar(aes(fill = isFraud), position = "dodge") +
labs(title = "merchantCountryCode Distribution for Fraudulent Response")
```



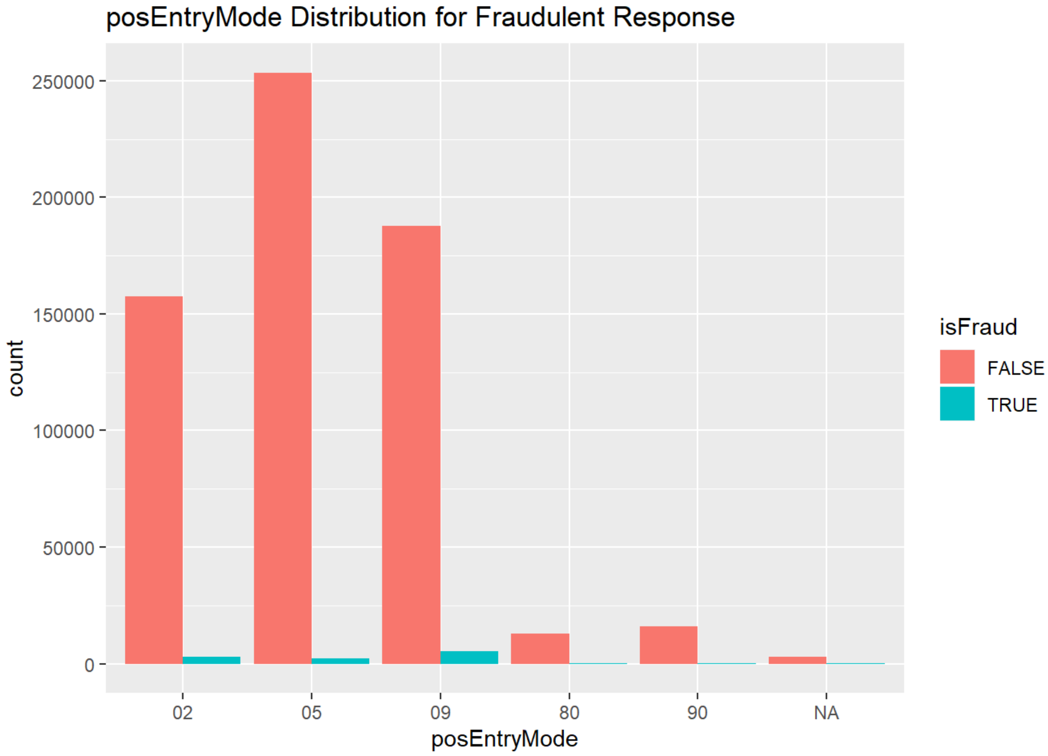

```
ggplot(df, aes(transactionType, ..count..)) + geom_bar(aes(fill = isFraud), position = "dodge") +
labs(title = "transactionType Distribution for Fraudulent Response")
```



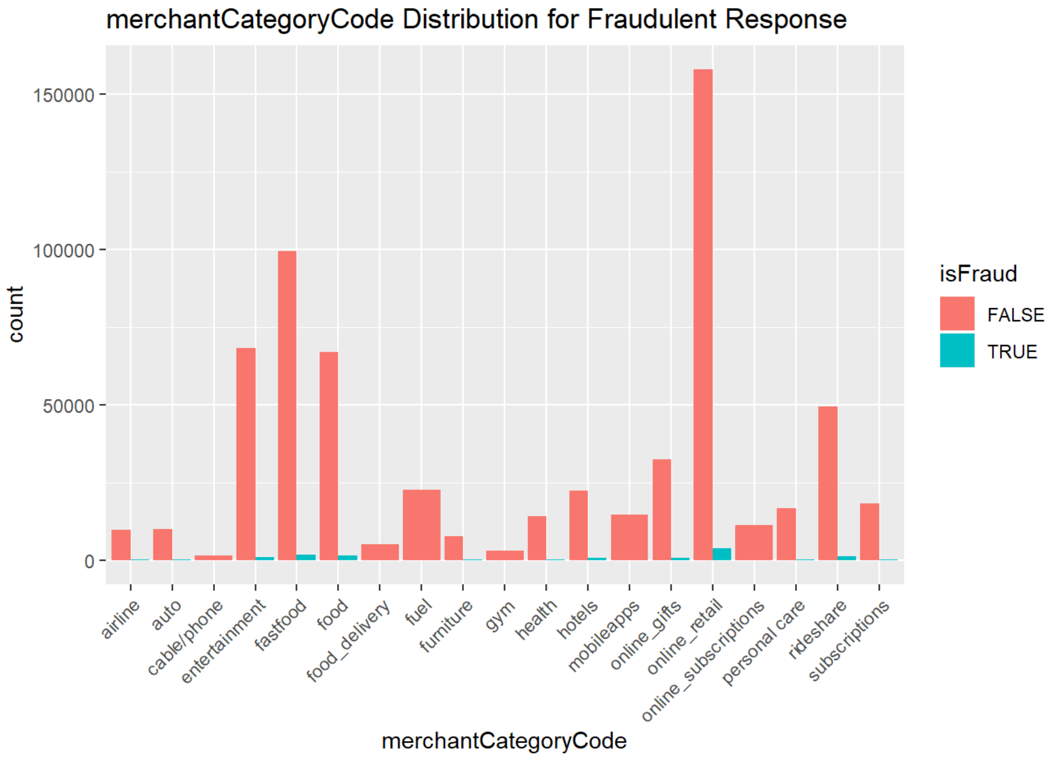

```
ggplot(df, aes(posConditionCode, ..count..)) + geom_bar(aes(fill = isFraud), position = "dodge") +
labs(title = "posConditionCode Distribution for Fraudulent Response")
```




```
ggplot(df, aes(posEntryMode, ..count..)) + geom_bar(aes(fill = isFraud), position = "dodge") +
labs(title = "posEntryMode Distribution for Fraudulent Response")
```




```
ggplot(df, aes(merchantCategoryCode, ..count..)) + geom_bar(aes(fill = isFraud), position = "dodge") +
labs(title = "merchantCategoryCode Distribution for Fraudulent Response") +
theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



It is visible from bar plot of Merchant Category Code that most fraudulent transactions are on online retail, fastfood and entertainment merchants as suspected.

3) Data Wrangling - Duplicate Transactions

Identifying duplicates, Reversal and Multi-swipe.

For reversed transactions, we can look at field transactionType = 'REVERSAL' The total amount of reversed transactions can be identified as below.

```
df %>% group_by(transactionType) %>% summarise_if(is.numeric, sum)
```

```
## # A tibble: 4 x 5
##   transactionType creditLimit availableMoney transactionAmou~
##   <chr>                 <dbl>          <dbl>            <dbl>
## 1 ADDRESS_VERIFI~   175391750      106548659.               0
## 2 PURCHASE         6509385000     4055922906.        84433897.
## 3 REVERSAL          175071250      103999183.         2242915.
## 4 <NA>                6841250        4073053.           85887.
## # ... with 1 more variable: currentBalance <dbl>
```

The total amount of reversed transactions is approximately $22.4 millions.

The total number of reversed transactions:

```
df %>% group_by(transactionType) %>% tally()
```

```
## # A tibble: 4 x 2
##   transactionType           n
##   <chr>                 <int>
## 1 ADDRESS_VERIFICATION  16478
## 2 PURCHASE             608685
## 3 REVERSAL              16162
## 4 <NA>                    589
```

There are 16162 reversed transactions which accounts for 2.52% of total transactions.

Let's check the median value of transaction amount for purchase and reversed transactions

```
df %>% group_by(transactionType) %>% summarise_if(is.numeric, median)
```

```
## # A tibble: 4 x 5
##   transactionType creditLimit availableMoney transactionAmou~
##   <chr>                 <dbl>          <dbl>            <dbl>
## 1 ADDRESS_VERIFI~        7500          3370.              0
## 2 PURCHASE               7500          3589.             89.6
## 3 REVERSAL               7500          3389.             91.3
## 4 <NA>                   7500          4009.            106.
## # ... with 1 more variable: currentBalance <dbl>
```

The median of reversed transactions is similar to that of purchase transactions at around $90

Let's work on multi-swipe!

As multi-swipe, are transactions where a vendor accidentally charges a customer's card multiple times within a **short time** span.

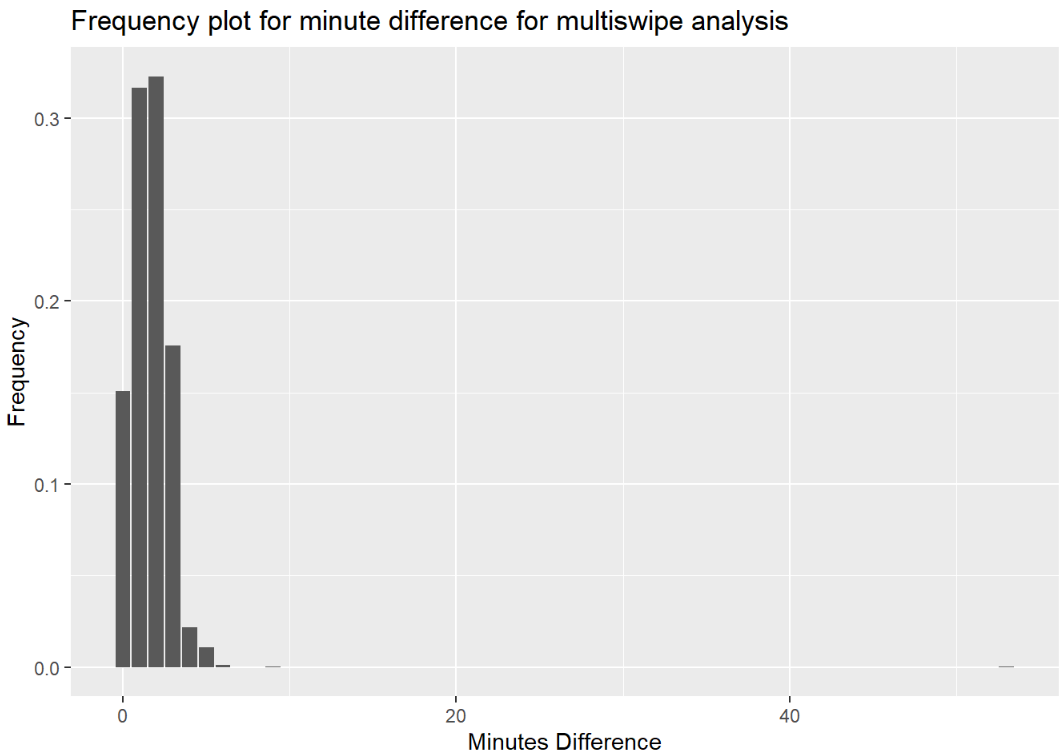
```
# Creating data without reversal transactions
df_ms <- df[!(df$transactionType == "REVERSAL"),]

# Breaking datetime variable to understand how short can time be for multi swipe!

df_ms$Date <- format(strptime(df_ms[,'transactionDateTime'],format = '%Y-%m-%dT%H:%M:%S'), "%Y-%m-%d")
df_ms$Hour <- format(strptime(df_ms[,'transactionDateTime'],format = '%Y-%m-%dT%H:%M:%S'), "%H")
df_ms$Minute <- format(strptime(df_ms[,'transactionDateTime'],format = '%Y-%m-%dT%H:%M:%S'), "%M")
df_ms$Second <- format(strptime(df_ms[,'transactionDateTime'],format = '%Y-%m-%dT%H:%M:%S'), "%S")
```

Assuming multiswipe is strictly of same amount. I believe, it will have same merchant name, same card last 4 digits, transaction amount, date and atleast hour! My intuittion is it should be few minutes but lets visualize it below.

Note: I have considered last 4 digits of credit card and not account number as one account can have multiple cards or supplimentary cards.

```
#Re-assigning dataframe
df2 <- df_ms

#Subsetting dataframe to calculate difference in minutes for multi swipe transactions. Accounting merchantNa
me, transactionAmount, cardLast4Digits, Date and Hour, assuming that multi-swipe transactions would occur no
t in more than few minutes.
df2 <- df_ms[,c('merchantName', 'transactionAmount', 'cardLast4Digits', 'Date', 'Hour', 'Minute')]

# Extracting duplicate rows
df2 <- df2[duplicated(df2[,c('merchantName', 'transactionAmount', 'cardLast4Digits', 'Date', 'Hour')]) | dup
licated(df2[,c('merchantName', 'transactionAmount', 'cardLast4Digits', 'Date', 'Hour')], fromLast=TRUE), ]

# Calculating difference in minutes for same transaction, on same date and with same merchant!
df3 <- df2 %>% group_by(merchantName, transactionAmount, cardLast4Digits, Date, Hour) %>%
 summarise(mindifference = max(as.numeric(Minute)) - min(as.numeric(Minute)))


ggplot(data = df3, aes(mindifference)) + geom_bar(aes(y = (..count..)/sum(..count..)))+
labs(title = paste("Frequency plot for minute difference for multiswipe analysis"), x = "Minutes Difference"
, y = "Frequency")
```

```
## Warning: Removed 1 rows containing non-finite values (stat_count).
```



As we can see most of the multi swpie transactions of same transaction amount, from same card, on same merchant and day differ only by few minutes and hence we can ascertain that these are multi swipe transactions!

There are couple of ways these can be removed. Since there is only one transaction with same above attributes but more than 10 mins difference. We can assume all observations with same merchant name, transaction amount, card last 4 digits, date and hour are multiswipe.

```
# Multi swipe duplicates
df_ms_new <- df_ms %>% group_by(merchantName, transactionAmount, cardLast4Digits, Date, Hour) %>% arrange(tr
ansactionDateTime) %>% filter(row_number() == 1)
# 619166

# Dropping one record with all NAs
df_ms_new <- df_ms_new[rowSums(is.na(df_ms_new)) != ncol(df_ms_new), ]
```

Checking for missing in new data created

```
misvars1 <- data.frame(t(data.frame(map(df_ms_new, ~mean(is.na(.))))))
misvars1 <- setDT(misvars1, keep.rownames = TRUE)[]
colnames(misvars1) <- c("var_name", "missrate")
misvars1[order(missrate, decreasing = TRUE),]
```

```
##                     var_name     missrate
##  1:               acqCountry 0.0060953058
##  2:             posEntryMode 0.0052344690
##  3:      merchantCountryCode 0.0009738923
##  4:         posConditionCode 0.0004506069
##  5:            accountNumber 0.0000000000
##  6:               customerId 0.0000000000
##  7:              creditLimit 0.0000000000
##  8:           availableMoney 0.0000000000
##  9:      transactionDateTime 0.0000000000
## 10:        transactionAmount 0.0000000000
## 11:             merchantName 0.0000000000
## 12:     merchantCategoryCode 0.0000000000
## 13:           currentExpDate 0.0000000000
## 14:          accountOpenDate 0.0000000000
## 15:  dateOfLastAddressChange 0.0000000000
## 16:                  cardCVV 0.0000000000
## 17:               enteredCVV 0.0000000000
## 18:          cardLast4Digits 0.0000000000
## 19:          transactionType 0.0000000000
## 20:                  isFraud 0.0000000000
## 21:           currentBalance 0.0000000000
## 22:              cardPresent 0.0000000000
## 23: expirationDateKeyInMatch 0.0000000000
## 24:                     Date 0.0000000000
## 25:                     Hour 0.0000000000
## 26:                   Minute 0.0000000000
## 27:                   Second 0.0000000000
##                     var_name     missrate
```

The missing rate is in the similar range to intial data, this verifies that operations we performed have not affecting missing data!

Second approach! This way we will assure that we have kept first transaction with respect to time as Normal.

Finally, Lets check how many transactions are multi-swipe by keeping first as ‘normal’ and considering rest at duplicates.

```
# converting minutes to seconds and creating secondnew
df_ms$SecondNew <- as.numeric(df_ms$Second) + (as.numeric(df_ms$Minute)*60)

df_dup <- df_ms[,c('merchantName', 'transactionAmount', 'cardLast4Digits', 'Date', 'Hour', 'SecondNew')]

df_ms_dup <- df_dup[duplicated(df_dup[,c('merchantName', 'transactionAmount', 'cardLast4Digits', 'Date', 'Ho
ur')]),]

df_ms_dup1 <- na.omit(df_ms_dup)
dim(df_ms_dup1)
```

```
## [1] 5998    6
```

There are approx 6000 multi swipe duplicate records.

```
median(df_ms_dup1$transactionAmount)
```

```
## [1] 99.745
```

```
sum(df_ms_dup1$transactionAmount)
```

```
## [1] 865249.5
```

Total multi swipe transaction amount is $865249.5 and Median multi swipe transaction amount is slightly higher at ~ $100.

Multi swipe can also be done with purpose of fraud!

```
indx_ms <- rownames(df_ms_dup1)

temp <- df_ms[rownames(df_ms) %in% indx_ms, ]
dim(temp[temp$isFraud=="TRUE", ])
```

```
## [1] 131  28
```

131 of 5998 multi swipe records are fraud (approximately 2.2%)

Creating final data with first as normal transaction!

```
df_final <- df_ms[!rownames(df_ms) %in% indx_ms, ]

# Removing expanded NA (All columns with NA which are around 589)
df_final <- df_final[rowSums(is.na(df_final)) != ncol(df_final), ]
```

Our final data after removing reversal and multi swipe transactions is df_final with 619165 observations. Note we have kept nulls in our data. Notice, the number of observations match with df_ms_new's number of observations where the duplicates were removed by sorting transactionDateTime and grouping by merchantName, transactionAmount, cardLast4Digits, Date, Hour.

While in df_final, we have identified duplicate rows at second levels and removing duplicates using index of duplicates and keeping first transaction in data as 'normal'. We will be using df_final for further analysis as this method can be trusted!

The missing rate in final data after removing the two types of duplicates.

```
misvars2 <- data.frame(t(data.frame(map(df_final, ~mean(is.na(.))))))
misvars2 <- setDT(misvars2, keep.rownames = TRUE)[]
colnames(misvars2) <- c("var_name", "missrate")
misvars2[order(missrate, decreasing = TRUE),]
```

```
##                    var_name     missrate
##  1:              acqCountry 0.0060953058
##  2:            posEntryMode 0.0052344690
##  3:     merchantCountryCode 0.0009738923
##  4:        posConditionCode 0.0004506069
##  5:           accountNumber 0.0000000000
##  6:              customerId 0.0000000000
##  7:             creditLimit 0.0000000000
##  8:          availableMoney 0.0000000000
##  9:     transactionDateTime 0.0000000000
## 10:       transactionAmount 0.0000000000
## 11:            merchantName 0.0000000000
## 12:    merchantCategoryCode 0.0000000000
## 13:          currentExpDate 0.0000000000
## 14:         accountOpenDate 0.0000000000
## 15:  dateOfLastAddressChange 0.0000000000
## 16:                 cardCVV 0.0000000000
## 17:              enteredCVV 0.0000000000
## 18:         cardLast4Digits 0.0000000000
## 19:         transactionType 0.0000000000
## 20:                 isFraud 0.0000000000
## 21:          currentBalance 0.0000000000
## 22:             cardPresent 0.0000000000
## 23: expirationDateKeyInMatch 0.0000000000
## 24:                    Date 0.0000000000
## 25:                    Hour 0.0000000000
## 26:                  Minute 0.0000000000
## 27:                  Second 0.0000000000
## 28:               SecondNew 0.0000000000
##                    var_name     missrate
```

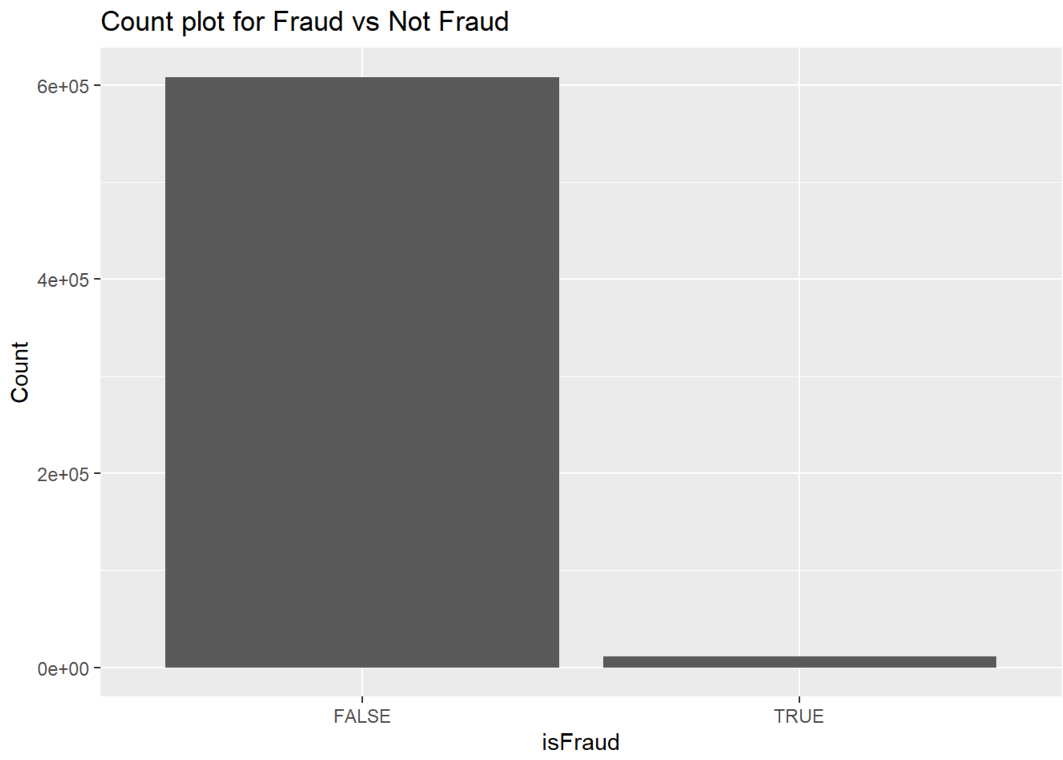
4) Data Preparation for Modeling

Modeling will require extensive data preparation. including handling missing values, identifying outliers if any, transforming variables for better understanding of model and model requirement and variable selecton.

Let's begin!

Before begining the modeling we must check the truth event rate (Fraud Rate), to have better idea about the model we need to build.

```
ggplot(data = df_final, aes(isFraud)) + geom_bar(aes(y = ..count..))+
labs(title = paste("Count plot for Fraud vs Not Fraud"), x = "isFraud", y = "Count")
```




```
event <- table(df_final$isFraud)
event
```

```
##
##   FALSE   TRUE
## 608299  10866
```

The event rate or truth rate of fraud nature is 1.75%. From this, is understandable that data is highly *imbalanced* and has to be treated carefully. My intuition is underbalancing will give good results given we have more than 600,000 observations which are not frauds and hence a sample of this can be nice representation of population.

The strategy here is to build a benchmark model with subset of raw variables first and then try out some new features with business sence to evaluate performance lift comparision with respect to benchmark model

As we have 28 features in our final data, which also include Date, Hour, Minute, Second, SecondNew extracted from transactionDateTime.

```
cat_var1 <- c(cat_var, colnames(df_final)[24:28])
log_var
```

```
## [1] "isFraud"                  "cardPresent"
## [3] "expirationDateKeyInMatch"
```

```
num_var
```

```
## [1] "creditLimit"      "availableMoney"    "transactionAmount"
## [4] "currentBalance"
```

```
cat_var1
```

```
##  [1] "accountNumber"          "customerId"
##  [3] "transactionDateTime"    "merchantName"
##  [5] "acqCountry"             "merchantCountryCode"
##  [7] "posEntryMode"           "posConditionCode"
##  [9] "merchantCategoryCode"   "currentExpDate"
## [11] "accountOpenDate"        "dateOfLastAddressChange"
## [13] "cardCVV"                "enteredCVV"
## [15] "cardLast4Digits"        "transactionType"
## [17] "Date"                   "Hour"
## [19] "Minute"                 "Second"
## [21] "SecondNew"
```

As the missing value variables are categorical, I would like to create a new category in each variable with name like 'missing'. This is better than treating missing value with mode here. As I am not exactly sure how data is collected and prepared. I am assuming most of missing value are 'NOT AT RANDOM'and hence keeping missing information here can be cruicial given higher event rate (%Fraud) for this population.

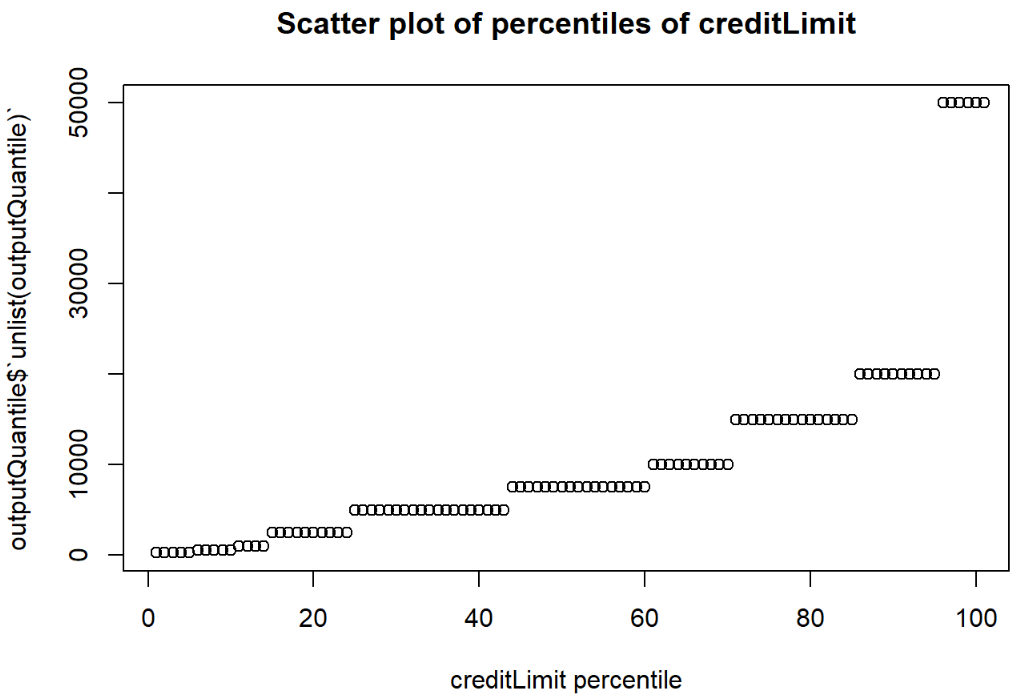
```
df_model <- df_final

df_model$acqCountry[which(is.na(df_model$acqCountry))] = "missing"
df_model$posEntryMode[which(is.na(df_model$posEntryMode))] = "missing"
df_model$merchantCountryCode[which(is.na(df_model$merchantCountryCode))] = "missing"
df_model$posConditionCode[which(is.na(df_model$posConditionCode))] = "missing"
```

Outlier detection for numerical variables. As there are only 4 numerical columns, I would like to use flooring and capping here. For example, to replace greater than 99th percentile value we can ceil the values with 99th percentile and values lower than 1st percentile can be floored with 1st percentile value.

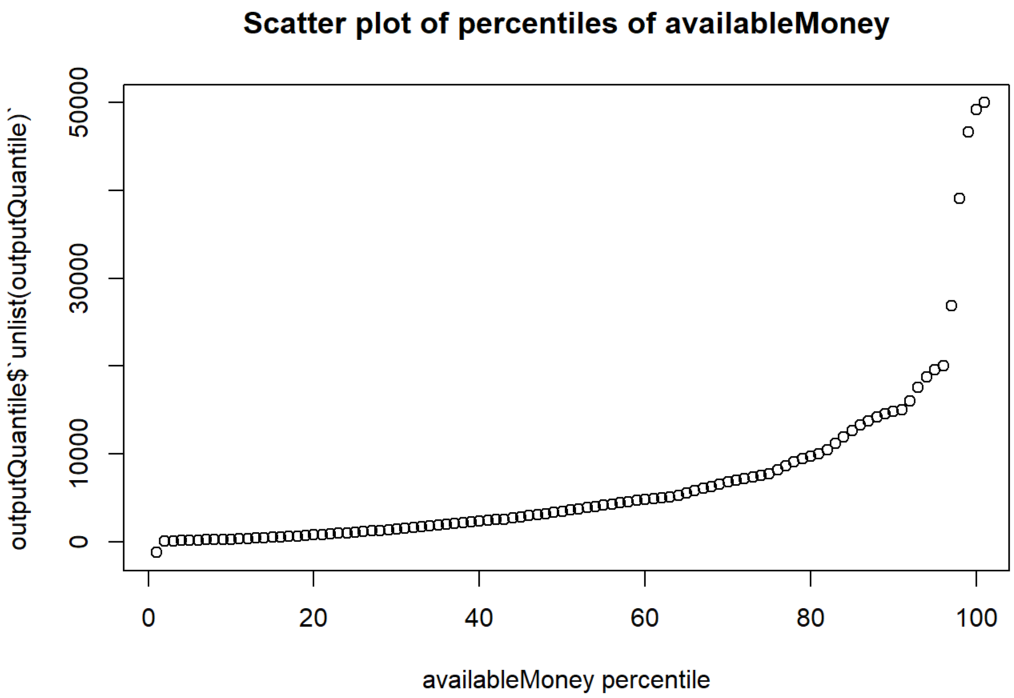
Lets check the scatter plot of each percentile value (1 to 100) for these variable to better strengthen idea of flooring and capping by 1st and 99th percentile.

```
for (var in num_var) {
  outputQuantile <- quantile(df_model[,as.character(var)], seq(0,1,by=0.01))
  str(paste("99th Percentile value for ", var, outputQuantile["99%"]))
  str(paste("1st Percentile value for ", var, outputQuantile["1%"]))
  outputQuantile = as.data.frame(unlist(outputQuantile))
  plot(outputQuantile$`unlist(outputQuantile)`, main=paste("Scatter plot of percentiles of", var),
       xlab=paste(var, "percentile"), xlim=c(1,100), col="Black")
}
```

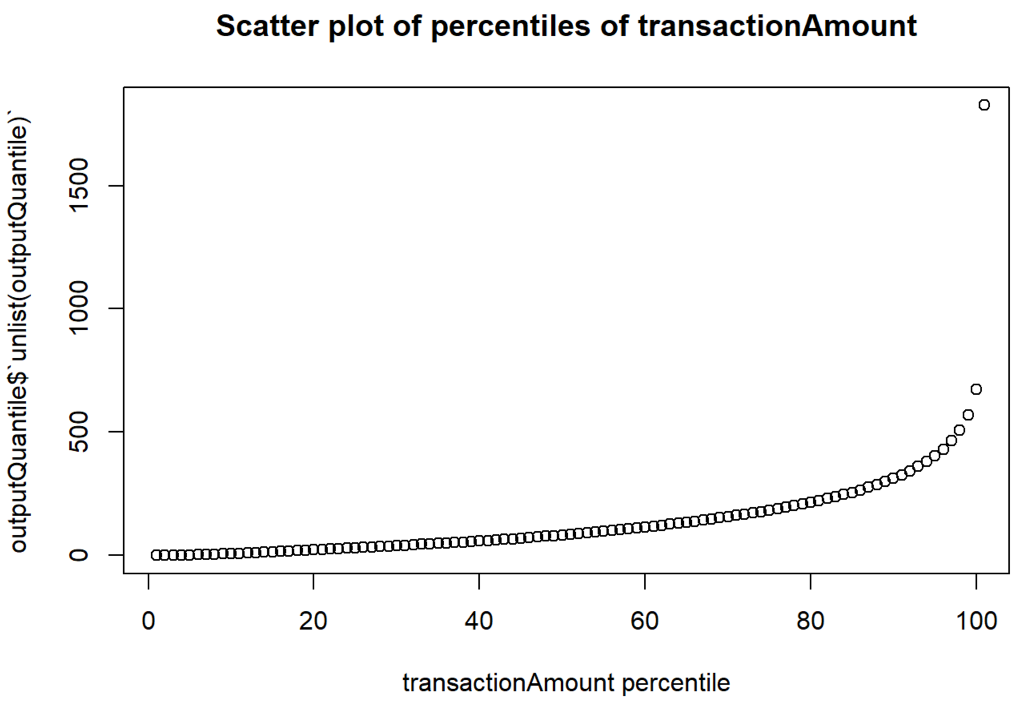
```
##  chr "99th Percentile value for  creditLimit 50000"
##  chr "1st Percentile value for  creditLimit 250"
```



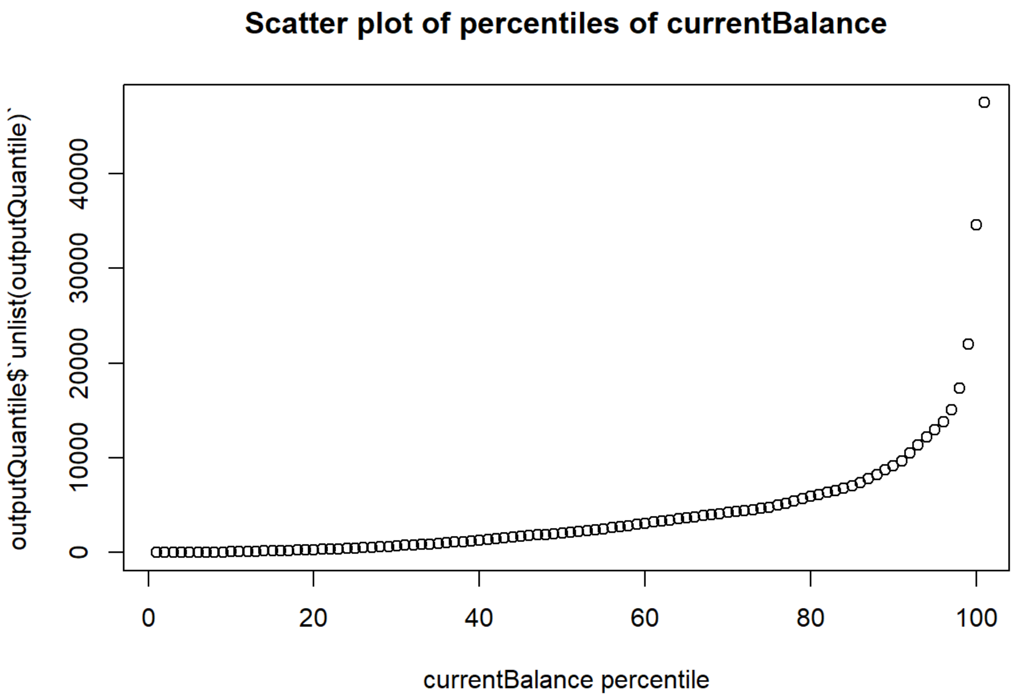

```
##  chr "99th Percentile value for  availableMoney 49233.5912"
##  chr "1st Percentile value for  availableMoney 40.3"
```




```
## chr "99th Percentile value for  transactionAmount 672.4672"
## chr "1st Percentile value for  transactionAmount 0"
```




```
## chr "99th Percentile value for  currentBalance 34600.3919999999"
## chr "1st Percentile value for  currentBalance 0"
```



As we can see from scatter plot and Right skewed histogram we plotted earlier for numerical variable suggest that capping 99th percentile makes sences (since rate of change or slope is going exponential after 95th percentile only). While flooring doesnot look a requirement here. Value above 99th percentile will be set at value of 99th percentile.

Capping the numerical variables.

```
#
df_model$creditLimit[df_model$creditLimit > 50000] <- 50000
df_model$availableMoney[df_model$availableMoney > 49233.59] <- 49233.59
df_model$transactionAmount[df_model$transactionAmount > 672.46] <- 672.46
df_model$currentBalance[df_model$currentBalance > 34600.39] <- 34600.39
```

Note: As given data is on snapshot of time, I would like to validate the models on out of time data as well forbetter comparison Creating training out of sample and out of time

As when the model will go into production, it will be scoring for future data (out of time), so its a good idea to not just evaluate on out of sample data (Ex. 30% in 70-30 split) but to also evaluate on Out of Time data.

Preparing training and validation data (Out of sample (oos) and Out of time (oot)).

```
df_model$Month <- format(strptime(df_model[,'transactionDateTime'],format = '%Y-%m-%dT%H:%M:%S'), "%m")

cat_var2 <- c(cat_var1, "Month")

set.seed(1234)
# Lets keep transactions in December for out of time validation
oot <- df_model[as.numeric(df_model$Month) == "12",] #Around 10% of complete data
df_model_f <- df_model[as.numeric(df_model$Month) != "12",]

# Out of sample and training by 30-70 split

nr<-nrow(df_model_f)
trnIndex<- sample(1:nr, size = round(0.7*nr), replace=FALSE)
Trn <- df_model_f[trnIndex, ]
oos <- df_model_f[-trnIndex, ]

# Lets check event rate in train, oos and oot
sum(df_final$isFraud) #10866 (1.7%)
```

```
## [1] 10866
```

```
sum(Trn$isFraud) #6983 (1.76%)
```

```
## [1] 6983
```

```
sum(oos$isFraud) #3024 (1.78%)
```

```
## [1] 3024
```

```
sum(oot$isFraud) #859 (1.56%)
```

```
## [1] 859
```

The label/event rate (% of Frauds) in Train, Out os Sample and Out of Time are in approximately same. (1.6-1.8%)

Let's think a little bit about modelling and as I mentioned earlier my first plan is to build a benchmark model to understand the power of any features I am planning to build. In benchmark model, I am planning to keep and remove features as following:

keep: creditLimit availableMoney transactionAmount acqCountry merchantCountryCode posEntryMode posConditionCode merchantCategoryCode (19 unique values, can be further clubbed to make lesser categories) accountOpenDate (Will calculate age of the account) transactionType (Will use to create binary variables for purchase and address verification) currentBalance cardPresent (binary will be created) expirationDateKeyInMatch (Definition not available; hence will keep as binary) cardCVV, enteredCVV (if same or not)

isFraud (Target/Label Variable)

remove: accountNumber, customerId (As these are identifiers or keys) transactionDateTime (Removing from benchmark but will incorporate in further models intelligently) merchantName (Too many distinct values (>2000) - check above data analysis section, and combining these will not make any sence) currentExpDate (Will be used in further models to create features) dateOfLastAddressChange (Will be used in further models to create features) cardLast4Digits (Keys but will be used to calculate number of credit cards in one account)

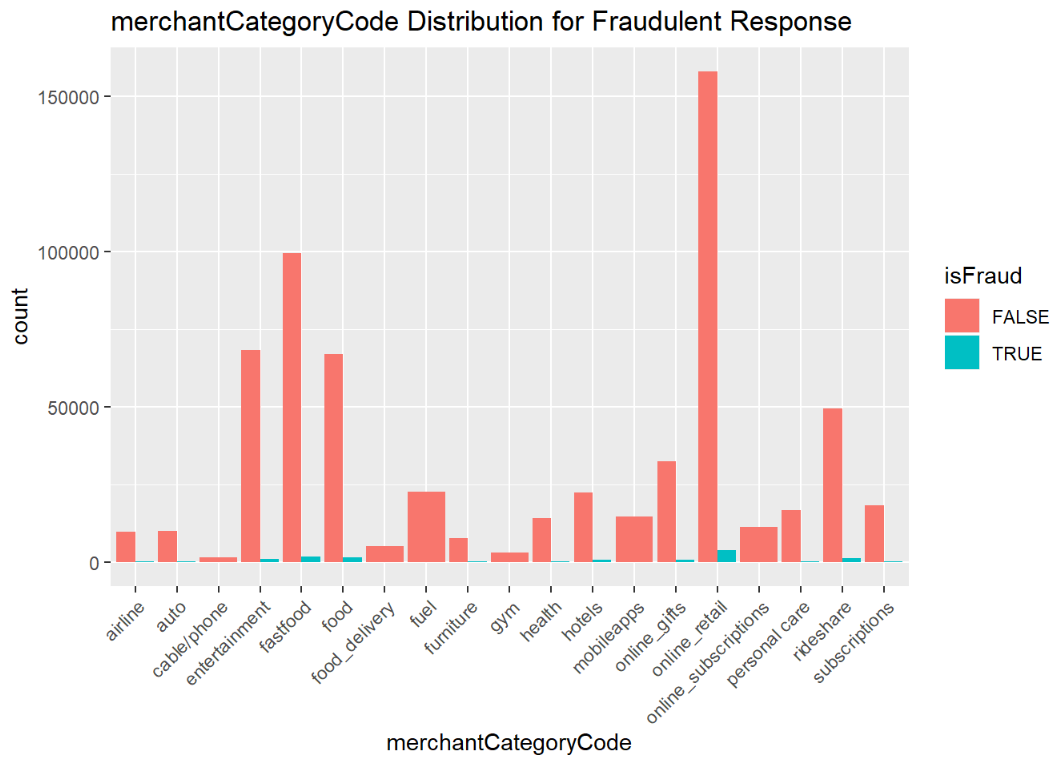
```
selected <- c('creditLimit', 'availableMoney', 'transactionAmount', 'acqCountry', 'merchantCountryCode', 'po
sEntryMode', 'posConditionCode', 'merchantCategoryCode', 'accountOpenDate', 'transactionType', 'currentBalan
ce', 'cardPresent', 'expirationDateKeyInMatch')
```

Let's combine some categories of merchant category code logically. This will help us reduce the sparsity in the data as encoding will form that many features.

```
# Lets first check, if we can club categories in merchantCategoryCode
table(df_model$merchantCategoryCode)
```

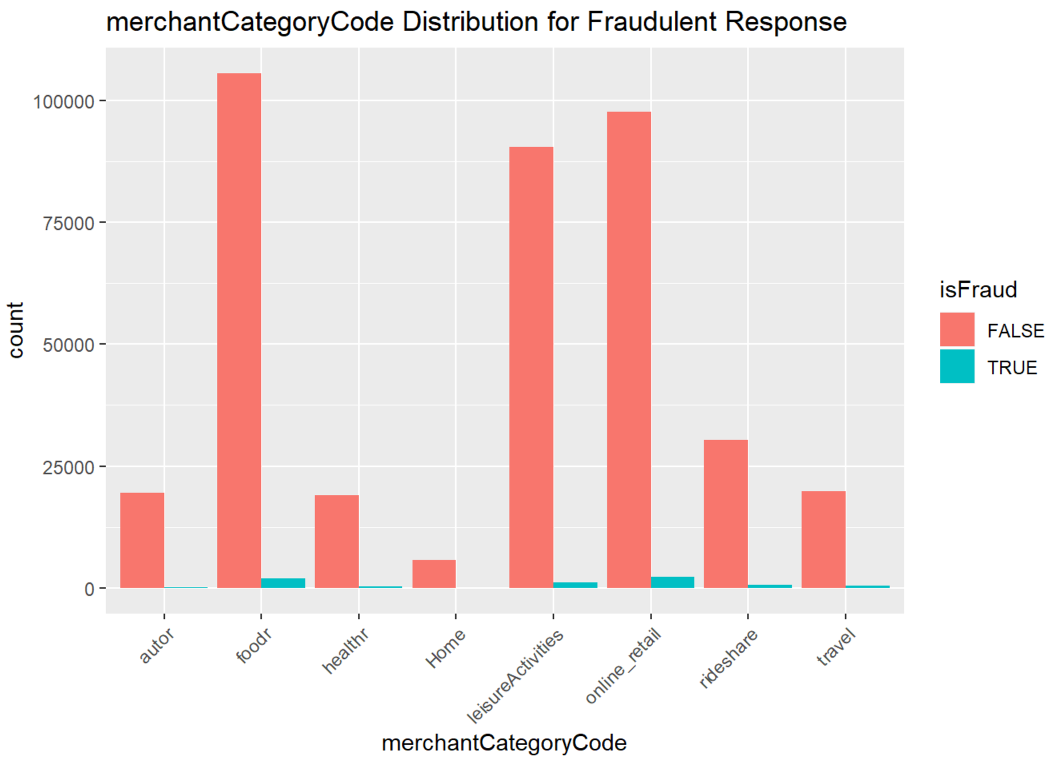
```
## 
##               airline                 auto          cable/phone 
##                  9579                 9749                 1490 
##         entertainment             fastfood                 food 
##                 66449                97251                65625 
##         food_delivery                 fuel            furniture 
##                  4990                22566                 7514 
##                   gym               health               hotels 
##                  2874                13782                21962 
##            mobileapps         online_gifts        online_retail 
##                 14614                31720               155221 
## online_subscriptions        personal care            rideshare 
##                 11247                16257                48600 
##         subscriptions 
##                 17675
```

```
ggplot(df, aes(merchantCategoryCode, ..count..)) + geom_bar(aes(fill = isFraud), position = "dodge") +
  labs(title = "merchantCategoryCode Distribution for Fraudulent Response") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



combining as follow (Based on intuition): fastfood, food, food delivery airline, hotel (travel and stay) personal care and health auto (unclear definition), fuel (auto expenses) rideshare online_retail cable/phone, furniture (Home expenses) entertainment, online_subscription, subscription, gym, mobileapps, online gifts (leisure and activities)

```
# Using car library to combine the categories of merchant category code variable
# We are clubbing 19 different merchant category codes in 8 different categories as below

Trn$merchantCategoryCode <- recode(Trn[, 'merchantCategoryCode'], "c('fastfood', 'food', 'food_delivery') =
'foodr'; c('airline', 'hotels') = 'travel'; c('personal care', 'health') = 'healthr'; c('auto', 'fuel') = 'a
utor';
c('airline', 'hotels') = 'travel'; c('cable/phone', 'furniture') = 'Home'; c('entertainment', 'online_subscr
iptions' , 'subscriptions', 'gym', 'mobileapps', 'online_gifts') = 'leisureActivities'")

oos$merchantCategoryCode <- recode(oos[, 'merchantCategoryCode'], "c('fastfood', 'food', 'food_delivery') =
'foodr'; c('airline', 'hotels') = 'travel'; c('personal care', 'health') = 'healthr'; c('auto', 'fuel') = 'a
utor';
c('airline', 'hotels') = 'travel'; c('cable/phone', 'furniture') = 'Home'; c('entertainment', 'online_subscr
iptions' , 'subscriptions', 'gym', 'mobileapps', 'online_gifts') = 'leisureActivities'")

oot$merchantCategoryCode <- recode(oot[, 'merchantCategoryCode'], "c('fastfood', 'food', 'food_delivery') =
'foodr'; c('airline', 'hotels') = 'travel'; c('personal care', 'health') = 'healthr'; c('auto', 'fuel') = 'a
utor';
c('airline', 'hotels') = 'travel'; c('cable/phone', 'furniture') = 'Home'; c('entertainment', 'online_subscr
iptions' , 'subscriptions', 'gym', 'mobileapps', 'online_gifts') = 'leisureActivities'")

ggplot(Trn, aes(merchantCategoryCode, ..count..)) + geom_bar(aes(fill = isFraud), position = "dodge") +
labs(title = "merchantCategoryCode Distribution for Fraudulent Response") +
theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



This looks good as we have now reduced merchant category to 8 from initial 19 and with fraud rates in line.

First New Features Introducing first new feature for benchmark model. Though my plan was to create features only in future versions. I believe age of account can be a very strong predictor and hence incorporating that in benchmark model as well.

Creating age of account variable using accountOpenDate age to expire in years (years left for expiry using currentExpDate)

```
monnb <- function(d) { lt <- as.POSIXlt(as.Date(d, origin="2000-01-01"));
lt$year*12 + lt$mon }
mondf <- function(d1, d2) { monnb(d2) - monnb(d1) }

Trn$ageMonths <- mondf(as.Date(Trn$accountOpenDate), as.Date(Trn$Date))
oos$ageMonths <- mondf(as.Date(oos$accountOpenDate), as.Date(oos$Date))
oot$ageMonths <- mondf(as.Date(oot$accountOpenDate), as.Date(oot$Date))

Trn$expireYear <- as.numeric(substr(Trn$currentExpDate, 4, 8)) - as.numeric(substr(Trn$Date, 1, 4))
oos$expireYear <- as.numeric(substr(oos$currentExpDate, 4, 8)) - as.numeric(substr(oos$Date, 1, 4))
oot$expireYear <- as.numeric(substr(oot$currentExpDate, 4, 8)) - as.numeric(substr(oot$Date, 1, 4))
```

Transforming some variables converting cardPresent, expirationDateKeyInMatch in binary

```
Trn$cardPresent <- ifelse(Trn$cardPresent=="TRUE", 1, 0)
oos$cardPresent <- ifelse(oos$cardPresent=="TRUE", 1, 0)
oot$cardPresent <- ifelse(oot$cardPresent=="TRUE", 1, 0)

Trn$expirationDateKeyInMatch <- ifelse(Trn$expirationDateKeyInMatch=="TRUE", 1, 0)
oos$expirationDateKeyInMatch <- ifelse(oos$expirationDateKeyInMatch=="TRUE", 1, 0)
oot$expirationDateKeyInMatch <- ifelse(oot$expirationDateKeyInMatch=="TRUE", 1, 0)
```

If ccv entered is actual cvv indicator

```
Trn$cvvInd <- ifelse(Trn$cardCVV==Trn$enteredCVV, 1, 0)
oos$cvvInd <- ifelse(oos$cardCVV==oos$enteredCVV, 1, 0)
oot$cvvInd <- ifelse(oot$cardCVV==oot$enteredCVV, 1, 0)
```

transactionType as if PURCHASE then 1 Else 0

```
Trn$transactionType <- ifelse(Trn$transactionType=="PURCHASE", 1, 0)
oos$transactionType <- ifelse(oos$transactionType=="PURCHASE", 1, 0)
oot$transactionType <- ifelse(oot$transactionType=="PURCHASE", 1, 0)
```

Lets create one hot encoded vectors for different categorical variables.

Trn_bm is created here!

```
#one hot encoding for catgeorical variables
varEncod <- c("acqCountry", "merchantCountryCode", "posEntryMode", "posConditionCode", "merchantCategoryCode")
varNum <- c("creditLimit", "availableMoney", "transactionAmount", "transactionType", "currentBalance", "cardPresent", "expirationDateKeyInMatch", "ageMonths", "expireYear", "cvvInd")
label <- "isFraud"


for (var in varEncod){
name_trn <- paste(var, "_cat_trn", sep = "")
name_oos <- paste(var, "_cat_oos", sep = "")
name_oot <- paste(var, "_cat_oot", sep = "")
assign(name_trn, unique(Trn[,var]))
assign(name_oos, unique(oos[,var]))
assign(name_oot, unique(oot[,var]))
}

acqCountry_trn_notoos <- acqCountry_cat_trn[which(!acqCountry_cat_trn %in% acqCountry_cat_oos)]
acqCountry_trn_notoot <- acqCountry_cat_trn[which(!acqCountry_cat_trn %in% acqCountry_cat_oot)]
acqCountry_oos_nottrn <- acqCountry_cat_oos[which(!acqCountry_cat_oos %in% acqCountry_cat_trn)]
acqCountry_oot_nottrn <- acqCountry_cat_oot[which(!acqCountry_cat_oot %in% acqCountry_cat_trn)]

merchantCountryCode_trn_notoos <- merchantCountryCode_cat_trn[which(!merchantCountryCode_cat_trn %in% merchantCountryCode_cat_oos)]
merchantCountryCode_trn_notoot <- merchantCountryCode_cat_trn[which(!merchantCountryCode_cat_trn %in% merchantCountryCode_cat_oot)]
merchantCountryCode_oos_nottrn <- merchantCountryCode_cat_oos[which(!merchantCountryCode_cat_oos %in% merchantCountryCode_cat_trn)]
merchantCountryCode_oot_nottrn <- merchantCountryCode_cat_oot[which(!merchantCountryCode_cat_oot %in% merchantCountryCode_cat_trn)]

posEntryMode_trn_notoos <- posEntryMode_cat_trn[which(!posEntryMode_cat_trn %in% posEntryMode_cat_oos)]
posEntryMode_trn_notoot <- posEntryMode_cat_trn[which(!posEntryMode_cat_trn %in% posEntryMode_cat_oot)]
posEntryMode_oos_nottrn <- posEntryMode_cat_oos[which(!posEntryMode_cat_oos %in% posEntryMode_cat_trn)]
posEntryMode_oot_nottrn <- posEntryMode_cat_oot[which(!posEntryMode_cat_oot %in% posEntryMode_cat_trn)]

posConditionCode_trn_notoos <- posConditionCode_cat_trn[which(!posConditionCode_cat_trn %in% posConditionCode_cat_oos)]
posConditionCode_trn_notoot <- posConditionCode_cat_trn[which(!posConditionCode_cat_trn %in% posConditionCode_cat_oot)]
posConditionCode_oos_nottrn <- posConditionCode_cat_oos[which(!posConditionCode_cat_oos %in% posConditionCode_cat_trn)]
posConditionCode_oot_nottrn <- posConditionCode_cat_oot[which(!posConditionCode_cat_oot %in% posConditionCode_cat_trn)]

merchantCategoryCode_trn_notoos <- merchantCategoryCode_cat_trn[which(!merchantCategoryCode_cat_trn %in% merchantCategoryCode_cat_oos)]
merchantCategoryCode_trn_notoot <- merchantCategoryCode_cat_trn[which(!merchantCategoryCode_cat_trn %in% merchantCategoryCode_cat_oot)]
merchantCategoryCode_oos_nottrn <- merchantCategoryCode_cat_oos[which(!merchantCategoryCode_cat_oos %in% merchantCategoryCode_cat_trn)]
merchantCategoryCode_oot_nottrn <- merchantCategoryCode_cat_oot[which(!merchantCategoryCode_cat_oot %in% merchantCategoryCode_cat_trn)]

# As all unique values for all categorical columns are available exhaustively in Training, Out of Sample and Out of Time (If not we would require additional treatment) - All above variables created are empty, hence safe for below one hot encoding of categorical variables

#One Hot for Trn
Trn_bm <- Trn[,varNum]

for (var in varEncod) {
for (unique_value in unique(Trn[,var])){
Trn_bm[paste(var, unique_value, sep = ".")] <- ifelse(Trn[,var]==unique_value,1,0)
}
}

# One Hot for oos
oos_bm <- oos[,varNum]

for (var in varEncod) {
for (unique_value in unique(oos[,var])){
oos_bm[paste(var, unique_value, sep = ".")] <- ifelse(oos[,var]==unique_value,1,0)
```

```
}
}

# One Hot for oot
oot_bm <- oot[,varNum]

for (var in varEncod) {
for (unique_value in unique(oot[,var])){
oot_bm[paste(var, unique_value, sep = ".")] <- ifelse(oot[,var]==unique_value,1,0)
}
}


# As all unique values for all categorical columns are available exhaustively in Training, Out of Sample and Out of Time (If not we would require additional treatment)
```

We made sure that categories are exhaustively represented or taken care in Train, OOS and OOT data This is cruicial for scoring or validation part.

Data for benchmark model is now ready!

```
# Creating a copy to save rework!
df_Trn <- Trn_bm
df_oos <- oos_bm
df_oot <- oot_bm
```

Variable Selection

I will be using Area Under the Curve (variant of Gini index) as a metric to evaluate the rank ordering power of variables. To Note: We should ideally use a method where we can also cover interactions between different available variables for better variable selection. But, I am only checking individual rank ordering power of each variable to select variables due to limited time. But this I would have surely tried given time (Like building Random Forest for variable selection as random forest gives every variable a chance while selecting variables with replacement for each tree builing in bagging.)

Also, I believe AUC method will give similar results to Null hypothesis method of linear regression. Which can also be used! In Null hypothesis of linear regression we try to identify statistically significant variables.

Lets check Gini/Area Under the Curve or Rank Order power of our Independend variables with respect to dependent variable.

```
Trn_bm$isFraud <- Trn$isFraud
oos_bm$isFraud <- oos$isFraud
oot_bm$isFraud <- oot$isFraud

Trn_bm$isFraud <- ifelse(Trn_bm$isFraud=="TRUE",1,0)
oos_bm$isFraud <- ifelse(oos_bm$isFraud=="TRUE",1,0)
oot_bm$isFraud <- ifelse(oot_bm$isFraud=="TRUE",1,0)

calc_auc <- function (actual, predicted)
{
r <- rank(predicted)
n_pos <- as.numeric (sum(actual == 1))
n_neg <- as.numeric (length(actual) - n_pos)
denom <- as.double (as.double (n_pos) * as.double(n_neg))
auc <- (sum(r[actual == 1]) - n_pos * (n_pos + 1)/2)/(denom)
auc
}

setDT(Trn_bm)

allnms <- colnames(Trn_bm)
allnms <- allnms[! allnms %in% c("isFraud")]

monkey = 1
actual <- ifelse (Trn_bm$isFraud==monkey,1,0)
aucDF <- Trn_bm[,allnms, with = F][,lapply(.SD, function (x) calc_auc (actual, x))]
aucDF <- as.data.frame (aucDF)
aucDF <- t (aucDF)
aucDF <- as.data.frame (aucDF)
aucDF$varName <- rownames (aucDF)
names (aucDF)[1] <- "auc"
aucDF$gini <- aucDF$auc
bigaucdf <- aucDF[, c("varName", "gini")]
rownames(bigaucdf) <- c(1:38)

bigaucdf$gini <- abs (bigaucdf$gini - 0.5)

# To understand which have maximum Gini
varSel_auc_0.001 <- bigaucdf$varName [which (bigaucdf$gini > 0.001)]
varSel_auc_0.007 <- bigaucdf$varName [which (bigaucdf$gini > 0.007)]
varSel_auc_0.03 <- bigaucdf$varName [which (bigaucdf$gini > 0.03)]
varSel_auc_0.01 <- bigaucdf$varName [which (bigaucdf$gini > 0.01)]
varSel_auc_0.05 <- bigaucdf$varName [which (bigaucdf$gini > 0.05)]
varSel_auc_0.08 <- bigaucdf$varName [which (bigaucdf$gin > 0.08)]

# Created multiple subset of variables based on area under the curve value (can also perform rank plots whic
h will give exactly same understanding on which predictors can perform best on given target variable)

bigaucdf <- bigaucdf[with(bigaucdf,order(-gini)),]
bigaucdf
```

```
##                                    varName         gini
## 3                        transactionAmount 1.884581e-01
## 23                         posEntryMode.05 1.088522e-01
## 22                         posEntryMode.09 8.991142e-02
## 36     merchantCategoryCode.online_retail 3.638185e-02
## 33 merchantCategoryCode.leisureActivities 3.620466e-02
## 6                              cardPresent 3.027042e-02
## 5                           currentBalance 2.943012e-02
## 28                      posConditionCode.08 2.219006e-02
## 1                              creditLimit 1.744614e-02
## 37              merchantCategoryCode.autor 1.659775e-02
## 27                      posConditionCode.01 1.653579e-02
## 38             merchantCategoryCode.travel 1.337761e-02
## 35          merchantCategoryCode.rideshare 1.015484e-02
## 4                          transactionType 9.388599e-03
## 21                         posEntryMode.02 9.355936e-03
## 25                    posEntryMode.missing 8.852770e-03
## 2                           availableMoney 7.713595e-03
## 8                                ageMonths 6.283249e-03
## 30                      posConditionCode.99 5.309580e-03
## 16                  merchantCountryCode.US 4.973047e-03
## 11                           acqCountry.US 4.631774e-03
## 34            merchantCategoryCode.healthr 3.741526e-03
## 10                                  cvvInd 3.464098e-03
## 9                               expireYear 3.212792e-03
## 19             merchantCountryCode.missing 3.102532e-03
## 12                      acqCountry.missing 2.964466e-03
## 31               merchantCategoryCode.Home 2.601937e-03
## 26                         posEntryMode.90 1.871609e-03
## 24                         posEntryMode.80 1.139545e-03
## 20                 merchantCountryCode.CAN 9.167888e-04
## 32              merchantCategoryCode.foodr 7.684355e-04
## 15                          acqCountry.CAN 7.071368e-04
## 13                          acqCountry.MEX 5.148092e-04
## 17                 merchantCountryCode.MEX 5.083649e-04
## 14                           acqCountry.PR 4.453616e-04
## 18                  merchantCountryCode.PR 4.453616e-04
## 29                 posConditionCode.missing 3.446928e-04
## 7                 expirationDateKeyInMatch 1.818218e-05
```

We can see transaction amount has maximum AUC, this makes sence as we saw a strong difference in distribution of transaction amount for fraud and not fraud behavior. Age month is at 18th number in this list, above many other variables.

Correlated Variable Reduction

Remove variables which are correlated, by keeping AUC/Gini in mind for each variable. Meaning, if variable 1 and variable 2 are correlated and variable 2 has higher AUC according to previous sections then keep variable 2 and remove variable 1

Keeping threshold of 80% correlation

```
df_Trn <- Trn_bm[,1:38]
df_Trn <- as.data.frame(df_Trn)

df_cor <- data.frame(matrix(NA, nrow = 394925, ncol = 38))
colnames(df_cor) <- bigaucdf$varName


for (i in 1:length(bigaucdf$varName)){
df_cor[,as.character(bigaucdf$varName[i])] <- df_Trn[,as.character(bigaucdf$varName[i])]
}

cor_inp <- sapply(df_cor, as.numeric)
cor_mat <- cor(cor_inp, use = "complete.obs")


rm_cor <- vector() #remove correlated
kp_cor <- vector() #keep correlated

for (i in 1:(length(bigaucdf$varName)-1)){
temp <- data.frame(cor_mat[i, (i+1):38])
colnames(temp) <- "cor"
temp$var <- rownames(cor_mat)[(i+1):38]
rownames(temp) <- c(1:(38-i))
var1 <- as.character(temp$var[temp$cor > 0.80])
rm_cor <- union(rm_cor, var1)
}

# using varSel_auc_0.001 (variables selected using AUC/gini method); removing correlated variables calculate
d above from this list

varf <- varSel_auc_0.001[! varSel_auc_0.001 %in% rm_cor]
varf <- c(varf, "isFraud")
varf
```

```
##  [1] "creditLimit"
##  [2] "transactionAmount"
##  [3] "transactionType"
##  [4] "currentBalance"
##  [5] "cardPresent"
##  [6] "ageMonths"
##  [7] "expireYear"
##  [8] "cvvInd"
##  [9] "acqCountry.missing"
## [10] "merchantCountryCode.US"
## [11] "merchantCountryCode.missing"
## [12] "posEntryMode.02"
## [13] "posEntryMode.09"
## [14] "posEntryMode.05"
## [15] "posEntryMode.80"
## [16] "posEntryMode.missing"
## [17] "posEntryMode.90"
## [18] "posConditionCode.01"
## [19] "posConditionCode.08"
## [20] "posConditionCode.99"
## [21] "merchantCategoryCode.Home"
## [22] "merchantCategoryCode.leisureActivities"
## [23] "merchantCategoryCode.healthr"
## [24] "merchantCategoryCode.rideshare"
## [25] "merchantCategoryCode.online_retail"
## [26] "merchantCategoryCode.autor"
## [27] "merchantCategoryCode.travel"
## [28] "isFraud"
```

```
# Finally we have 27 variables to build our benchmark model with after removing correlated and very low auc
value variables
```

Below 5 variables are correlated with other variables which have higher rank ordering power with respect to target variable.

```
rm_cor
```

```
## [1] "availableMoney"          "acqCountry.US"
## [3] "acqCountry.CAN"          "merchantCountryCode.MEX"
## [5] "merchantCountryCode.PR"
```

As the classes are highly imbalanced, it will be good idea to try different undersampled data to identify what fits our problem.

Note: I have tried different versions of undersampling, oversampling and balancing. Below are data, I want to keep in my final iteration to save model building time.

```
# library(ROSE)

Trn_bm %>% group_by(isFraud) %>% count()
```

```
## # A tibble: 2 x 2
## # Groups:   isFraud [2]
##   isFraud      n
##     <dbl>  <int>
## 1       0 387942
## 2       1   6983
```

```
#Undersampling majority class with 50%, 10% and 20% minority respectively (original is 1.7%)
Trn_u5 <- ovun.sample(isFraud ~., data=Trn_bm, na.action = na.pass, method = "under", p=0.5)$data
Trn_u5 %>% group_by(isFraud) %>% count()
```

```
## # A tibble: 2 x 2
## # Groups:   isFraud [2]
##   isFraud     n
##     <dbl> <int>
## 1       0  7020
## 2       1  6983
```

```
Trn_u1 <- ovun.sample(isFraud ~., data=Trn_bm, na.action = na.pass, method = "under", p=0.1)$data
Trn_u1 %>% group_by(isFraud) %>% count()
```

```
## # A tibble: 2 x 2
## # Groups:   isFraud [2]
##   isFraud     n
##     <dbl> <int>
## 1       0 62875
## 2       1  6983
```

```
Trn_u2 <- ovun.sample(isFraud ~., data=Trn_bm, na.action = na.pass, method = "under", p=0.2)$data
Trn_u2 %>% group_by(isFraud) %>% count()
```

```
## # A tibble: 2 x 2
## # Groups:   isFraud [2]
##   isFraud     n
##     <dbl> <int>
## 1       0 28033
## 2       1  6983
```

```
#balancing class with 50% each class
Trn_b <- ovun.sample(isFraud ~., data=Trn_bm, na.action = na.pass, method = "both", p=0.5)$data
Trn_b %>% group_by(isFraud) %>% count()
```

```
## # A tibble: 2 x 2
## # Groups:   isFraud [2]
##   isFraud      n
##     <dbl>  <int>
## 1       0 197326
## 2       1 197599
```

Lift calculation to measure decile performance (To evaluate model performance); if our model is able to capture information in top 2 percentile and top deciles population

```
lift_model <- function(actual,myPred,groups)
{
actual <- actual [order(myPred, decreasing=TRUE)]
myPred <- myPred [order (myPred, decreasing=TRUE)]
deciles <- rep (1:groups, each = length(actual)/groups)

deciles <- c(deciles, rep (groups, length (actual) - length(deciles)))
naiveAcc <- prop.table (table (actual))[2]
#naiveAcc<- naiveAcc/2
#tempDF <- data.frame (actual, deciles, myPred)
cumlifts <- NULL
for (j in 1:groups) {
cumlifts <- c(cumlifts, length (which (actual [deciles <= j]==1))/(naiveAcc * length (actual [deciles <= j])
))
}
return(cumlifts)
print(paste (cumlifts[1], cumlifts [5]))
}
```

5) Model Building

Note: I am using statistical tree based method for predicting the fraud. Starting with Decision Trees, I will build 3 benchmark models with different techniques, to choose a technique amoung Decision Trees, Random Forest and XGBoost

Also, I am writing nested loop to build final **hypertuned models** for above data selected.

Decision Trees

Benchmark Model #1 using Decision Trees using rpart!

I have written nested for loop to run model on each data shortlisted, hypertune model with minsplit (the minimum number of observations that must exist in a node in order for a split to be attempted), minbucket (the minimum number of observations that must exist in a node in order for a split to be attempted) and cp value (I am using built in feature of rpart to identify best value of cp in each iteration).

I have used 3 fold cross validation for hypertuning of parameters in final iterations (have tried 5 and 10 in previous iterations). We could use 10 or different numbers here but I have just used 5 to save computational time.

```
# In previous iterations the raw data with 1.7% event rate and 10% undersmapled data did not perform well an
d hence removing it from here and testing it only on below datasets

# Trn_u2 denotes 20% undersampled training data, Trn_u5 denotes 50% undersampled data and Trn_b denotes bala
nced data

data <- list(Trn_u2, Trn_u5, Trn_b)

rpart_oos <- list()
rpart_oot <- list()
rpart_confs <- list()
rpart_conft <- list()
rpart_lifts1 <- list()
rpart_liftt1 <- list()
rpart_lifts5 <- list()
rpart_liftt5 <- list()
rpart_aucs <- list()
rpart_auct <- list()
rpart_split <- list()
rpart_bucket <- list()
rpart_cp <- list()

for (l in 1:length(data)){
df <- as.data.frame(data[l])
Train <- df[,varf]
Train <- Train[sample(1:nrow(Train)),]

folds <- cut(seq(1,nrow(Train)),breaks=3,labels=FALSE)

# Initializing empty vector to store the accuracies
bestsplit <- vector()
bestbucket <- vector()
confusion.matrix <- matrix()
cpvalue <- vector()
auc1_cv <- vector()
```

```
auc2_cv <- vector()

# lets find best values of minsplit and minbucket and corresponding cp
# I have tried many values of minsplit and minbucket in earlier versions including c(5,10,15,20,25,30) and h
ence keeping which I believe will help and save time.
minsplit <- c(10,20,25)
minbucket <- c(10,20,25)

auc_1 <- matrix(nrow = length(minsplit), ncol = length(minbucket))
auc_2 <- matrix(nrow = length(minsplit), ncol = length(minbucket))
cp_final <- matrix(nrow = length(minsplit), ncol = length(minbucket))


for(k in 1:3){

testIndexes <- which(folds==k, arr.ind=TRUE)
data_test <- Train[testIndexes, ]
data_train <- Train[-testIndexes, ]

for(i in 1:length(minsplit)){

  for(j in 1:length(minbucket)){

    data_train$isFraud <- as.factor(data_train$isFraud)
    data_test$isFraud <- as.factor(data_test$isFraud)

    cf_rpart <- rpart(isFraud~., data = data_train, method = "class",
                      control = rpart.control(minsplit = minsplit[i], minbucket = minbucket[j], cp = 0.001)
)

    score <- predict(cf_rpart,data_test, type = "class")

    auc_1[i,j] <- calc_auc(data_test$isFraud, score)

    opt = which.min(cf_rpart$cptable[,"xerror"])
    cp = cf_rpart$cptable[opt, "CP"]
    cp_final[i,j] <- cp

    tree_prune = prune(cf_rpart, cp = cp)
    predTst=predict(tree_prune,data_test, type='class')

    auc_2[i,j] <- calc_auc(data_test$isFraud, predTst)
    }

  }

index <- which(auc_1 == max(auc_1, na.rm = T), arr.ind = T)
index <- index[1,]
bestsplit[k] <- minsplit[index[1]]
bestbucket[k] <- minbucket[index[2]]
cpvalue[k] <- cp_final[index[1],index[2]]
auc1_cv[k] <- mean(auc_1)
auc2_cv[k] <- mean(auc_2)

}

index <- which(auc1_cv == max(auc1_cv), arr.ind = T)

best_split_f <- bestsplit[index]
best_bucket_f <- bestbucket[index]
cpvalue <- cpvalue[index]

rpart_split[[l]] <- best_split_f
rpart_bucket[[l]] <- best_bucket_f
rpart_cp[[l]] <- cpvalue

# best_msplit <- which.max(cv_accuracy2)
# # min split should be 10 and corresponding cp
# best_cp <- cp_final[best_msplit]

# Lets train the final model
cf_rpart_final <- rpart(isFraud~., data = Train, method = "class", control = rpart.control(minsplit = best_s
plit_f, cp = cpvalue, minbucket = best_bucket_f))
```

```
plit_l, cp = cpvalue, minbucket = best_bucket_l))

save(cf_rpart_final, file = paste("C:/A_Work/UIC/Internship/Capital One/Data_Science_Challenge/rpart",
as.character(l) , ".rda", sep = "")) # Saving models for future use

score_final <- predict(cf_rpart_final, oos_bm[,varf], type='class')
score_final_oot <- predict(cf_rpart_final, oot_bm[,varf], type='class')

rpart_oos[[l]] <- score_final
rpart_oot[[l]] <- score_final_oot

confusion_dts <- table(pred = score_final, true=oos_bm$isFraud)
confusion_dtt <- table(pred = score_final_oot, true=oot_bm$isFraud)

rpart_confs[[l]] <- confusion_dts
rpart_conft[[l]] <- confusion_dtt

lifts1 <- lift_model(oos_bm$isFraud, score_final,50)[1]
liftt1 <- lift_model(oot_bm$isFraud, score_final_oot,50)[1]

lifts5 <- lift_model(oos_bm$isFraud, score_final,50)[5]
liftt5 <- lift_model(oot_bm$isFraud, score_final_oot,50)[5]

rpart_lifts1[[l]] <- lifts1
rpart_liftt1[[l]] <- liftt1

rpart_lifts5[[l]] <- lifts5
rpart_liftt5[[l]] <- liftt5

aucs <- calc_auc(oos_bm$isFraud, score_final)
auct <- calc_auc(oot_bm$isFraud, score_final_oot)

rpart_aucs[[l]] <- aucs
rpart_auct[[l]] <- auct

}

l <- which(rpart_aucs == max(unlist(rpart_aucs), na.rm=T), arr.ind = T)
l1 <- which(rpart_auct == max(unlist(rpart_auct), na.rm = T), arr.ind = T)

print(paste("According to AUC metric on OOS model", as.character(l), "is best"))
```

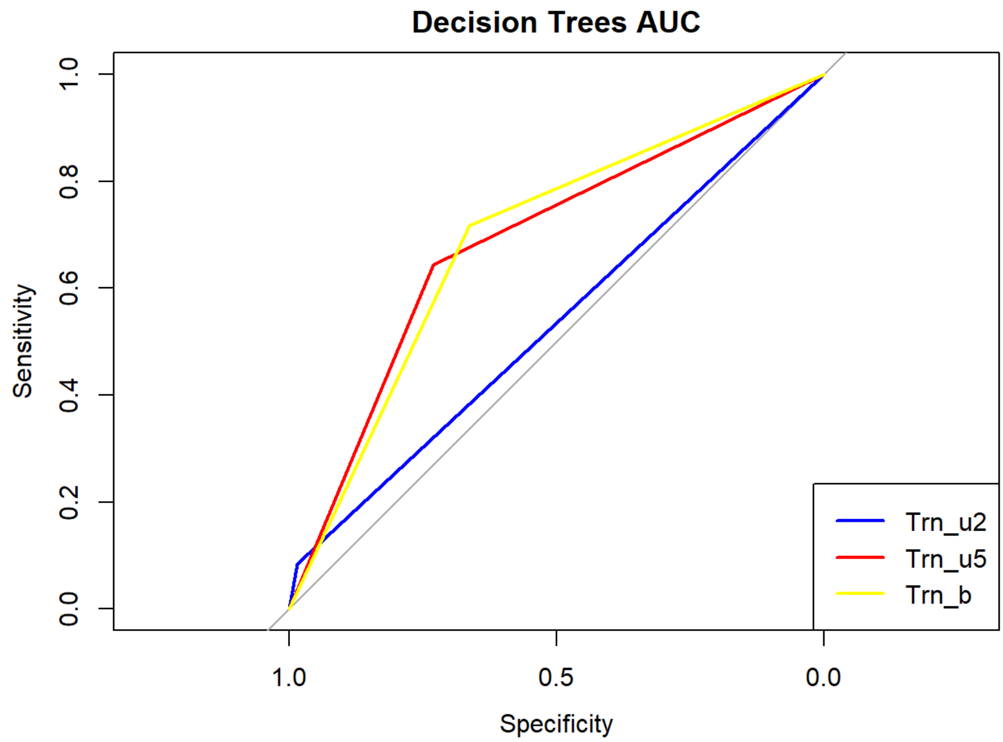
```
## [1] "According to AUC metric on OOS model 3 is best"
```

```
print(paste("According to AUC metric on oot model", as.character(l1), "is best"))
```

```
## [1] "According to AUC metric on oot model 3 is best"
```

Let’s Plot ROC Curves to visualize above decision tree results for out of time data

```
# library(pROC)

# OOS
plot(roc(as.numeric(oot_bm$isFraud), as.numeric(rpart_oot[[1]])),print.auc = F, col = "blue", main = "Decisi
on Trees AUC")
plot(roc(as.numeric(oot_bm$isFraud), as.numeric(rpart_oot[[2]])),print.auc = F,
     col = "red", add = TRUE, print.auc.y = .4)
plot(roc(as.numeric(oot_bm$isFraud), as.numeric(rpart_oot[[3]])),print.auc = F,
     col = "yellow", add = TRUE, print.auc.y = .4)
legend("bottomright", legend=c("Trn_u2", "Trn_u5", "Trn_b"), col=c("blue", "red", "yellow"), lwd=2)
```



As we can see area under the curve for Trn_b (Balanced data), Trn_u5 (50% undersampled data) is highest! and approximately same.

Confusion Matrix of DTs constructed for Out of Sample data. Model built on balanced sample showed great results at the cost of complexity. The true positives are 2156 and True negatives are 868!

```
rpart_confs
```

```
## [[1]]
##      true
## pred      0      1
##     0 162627   2698
##     1   3603    326
##
## [[2]]
##      true
## pred      0      1
##     0 117861   1108
##     1  48369   1916
##
## [[3]]
##      true
## pred      0      1
##     0 105753    868
##     1  60477   2156
```

we can see final tuned model on data 2 which is 50% undersampled data and data 3 which is balanced data has best predictions. There are more than 2000 frauds cases predicted correctly! Model on data 2 was quick as well due to size.

Out of Sample performance.
Using 50 % undersampled data we have identified 2022 of 3024 cases (66.86%) Using balanced data we have identified 2156 of 3024 cases (71.29%)

Lets check the confusion matrix for Out of Time (December Transactions kept separate for OOT Validation)! The true positive are higest for 50% undersampled data and balanced data at 578 and 616 respectively and cases of True negative are also low less than 300.

```
rpart_conft
```

```
## [[1]]
##      true
## pred     0     1
##    0 53267   786
##    1   860    73
##
## [[2]]
##      true
## pred     0     1
##    0 39558   306
##    1 14569   553
##
## [[3]]
##      true
## pred     0     1
##    0 35854   243
##    1 18273   616
```

Out of Time performance.

Using 50 % undersampled data we have identified 578 of 859 cases (67.28%) Using balanced data we have identified 616 of 859 cases (71.71%)

Area under the curve (Gini = 2*AUC -1)!

```
print(paste("Area under the curve for best decision tree with 50% undersampled data on out os sample validat
ion is", rpart_aucs[[l]]))
```

```
## [1] "Area under the curve for best decision tree with 50% undersampled data on out os sample validation i
s 0.674573883574966"
```

```
print(paste("Area under the curve for best decision tree with 50% undersampled data on out os time validatio
n is", rpart_auct[[l]]))
```

```
## [1] "Area under the curve for best decision tree with 50% undersampled data on out os time validation is
0.689759003170507"
```

Lift in top 2 percentile population

```
print(paste("lift in top 2 percentile population for best decision tree with 50% undersampled data on out os
sample validation is", rpart_lifts1[[l]]))
```

```
## [1] "lift in top 2 percentile population for best decision tree with 50% undersampled data on out os samp
le validation is 2.23219561088837"
```

```
print(paste("lift in top 2 percentile population for best decision tree with 50% undersampled data on out os
time validation is", rpart_liftt1[[l]]))
```

```
## [1] "lift in top 2 percentile population for best decision tree with 50% undersampled data on out os time
validation is 2.32981406527894"
```

Lift in top 10 percentile population

```
print(paste("lift in top 10 percentile (first decile) population for best decision tree with 50% undersample
d data on out os sample validation is", rpart_lifts5[[l]]))
```

```
## [1] "lift in top 10 percentile (first decile) population for best decision tree with 50% undersampled dat
a on out os sample validation is 1.88496518252796"
```

```
print(paste("lift in top 10 percentile (first decile) population for best decision tree with 50% undersample
d data on out os time validation is", rpart_liftt5[[l]]))
```

```
## [1] "lift in top 10 percentile (first decile) population for best decision tree with 50% undersampled dat
a on out os time validation is 2.09683265875105"
```

Best parameter for this decision tree built on 50% undersampled data and according to AUC are

```
rpart_bucket[[11]]
```

```
## [1] 10
```

```
rpart_split[[11]]
```

```
## [1] 10
```

```
rpart_cp[[11]]
```

```
## [1] 0.001
```

Variable Importance of Best Decision Tree Constructed using 50% undersampled data

```
benchmarkDT <- load(file = "C:/A_Work/UIC/Internship/Capital One/Data_Science_Challenge/rpart2.rda")
cf_rpart_final$variable.importance
```

```
##         transactionAmount            posEntryMode.05
##               604.4533951                276.1930293
##           transactionType merchantCategoryCode.autor
##                28.9633918                 23.1249216
##                 ageMonths
##                 0.6868789
```

We can see our hypothesis on transaction amount to be best predictor holds. Also age in months is one of the top predictors.

Variable Importance of Best Decision Tree Constructed using balanced data (data 3)

```
benchmarkDT1 <- load(file = "C:/A_Work/UIC/Internship/Capital One/Data_Science_Challenge/rpart3.rda")
cf_rpart_final$variable.importance
```

```
##          transactionAmount              posEntryMode.05
##               1.830227e+04                 9.731353e+03
##  merchantCategoryCode.autor              transactionType
##               1.183434e+03                 7.569114e+02
##                  ageMonths              posEntryMode.09
##               4.827709e+02                 3.801546e+02
## merchantCategoryCode.travel              posEntryMode.02
##               3.486472e+02                 2.359768e+02
##                creditLimit          posConditionCode.01
##               7.458753e+01                 4.059192e+01
##        posConditionCode.08               currentBalance
##               3.856726e+01                 2.683410e+01
##       posEntryMode.missing merchantCountryCode.missing
##               9.590940e+00                 4.839465e+00
##        posConditionCode.99                       cvvInd
##               3.064246e+00                 1.950974e+00
##         acqCountry.missing   merchantCategoryCode.Home
##               1.759806e+00                 2.656061e-01
```

Balanced data (high number of observations have used 18 variables in total with above variable importance!)

Looking at results of Final Tuned Decision Tree model, its clear that our problem requires improving event rate in data from 1.7%. Both undersampled at 50% and balanced give good results. 50% undersample is very fast and computationally less expensive. While, DT with balanced data has slightly better performance at the cost of computational power.

The best results are for undersampling with 50% true class and balanced sample with 50% of each class. There is slight lift seen even with undersampling with 20% of Fraudulent cases and hence for further models I will be focusing on these two datasets (Trn_u5, Trn_b)

Random Forest

Benchmark Model #2 Random Forest Random Forest is computationally more expensive and hence I am building 3 fold cross validation in this instead of 5 in the final iteration. The performance of Decision Tree on 50% undersampling data and balanced data were good and hence using these two for building random forest

```
data <- list(Trn_u5, Trn_b)
```

```
rf_oos <- list()
rf_oot <- list()
rf_confs <- list()
rf_conft <- list()
rf_lifts1 <- list()
rf_liftt1 <- list()
rf_lifts5 <- list()
rf_liftt5 <- list()
rf_aucs <- list()
rf_auct <- list()
rf_trees <- list()
rf_mt <- list()

for (l in 1:length(data)){
  df <- as.data.frame(data[l])
  Train <- df[,varf]
  Train <- Train[sample(1:nrow(Train)),]

  folds <- cut(seq(1,nrow(Train)),breaks=3,labels=FALSE)

  cv_accuracy_rf <- vector()
  # lets find best values of num of trees and mtry
  ntrees <- c(200, 500, 750)
  mt <- c(5, 7, 10) #I have tried different parameters for number of variables in each iteration of tree bui
lding but these are sufficient
  # using caret for cp tuning

  auc <- matrix(nrow = length(ntrees), ncol = length(mt))
  lift_2 <- matrix(nrow = length(ntrees), ncol = length(mt))
  lift_10 <- matrix(nrow = length(ntrees), ncol = length(mt))

  bestntrees_a <- vector()
  bestmtry_a <- vector()
  mauc_cv <- vector()

  bestntrees_l1 <- vector()
  bestmtry_l1 <- vector()

  bestntrees_l2 <- vector()
  bestmtry_l2 <- vector()

  for(k in 1:3){

    testIndexes <- which(folds==k,arr.ind=TRUE)
    Train_int <- Train[-testIndexes, ]
    Validation_int <- Train[testIndexes, ]

    for(i in 1:length(ntrees)){

      # ind <- sample(2, nrow(data_train), replace = T, prob = c(0.8, 0.2))
      # Train_int <- data_train[ind == 1, ]
      # Validation_int <- data_train[ind == 2, ]
      #
      pr.err <- c()
      for(j in 1:length(mt)){

        rf <- ranger(isFraud~., data = Train_int, num.trees = ntrees[i], mtry = mt[j], sample.fraction = 1,
         importance = 'impurity')
        predicted <- predict(rf, data = Validation_int, predict.all = FALSE, type = "response", num.trees =
rf$num.trees)

        score <- ifelse(predicted$predictions > 0.5,1,0)
        table(true = Validation_int$isFraud, pred = score)
        pr.err <- c(pr.err, mean(Validation_int$isFraud != score))

        auc[i,j] <- calc_auc(Validation_int$isFraud, score)
        lift <- lift_model(Validation_int$isFraud, score, 50)
        lift_2[i,j] <- lift[1]
        lift_10[i,j] <- lift[5]
      }
```

```
    }

    index_a <- which(auc == max(auc, na.rm = T), arr.ind = T)
    index_l1 <- which(lift_2 == max(lift_2, na.rm = T), arr.ind = T)
    index_l2 <- which(lift_10 == max(lift_10, na.rm = T), arr.ind = T)
    index1 <- index_a[1,]
    index2 <- index_l1[1,]
    index3 <- index_l2[1,]

    bestntrees_a[k] <- ntrees[index1[1]]
    bestntrees_l1[k] <- ntrees[index2[1]]
    bestntrees_l2[k] <- ntrees[index3[1]]

    bestmtry_a[k] <- mt[index1[2]]
    bestmtry_l1[k] <- mt[index2[2]]
    bestmtry_l2[k] <- mt[index3[2]]

    mauc_cv[k] <- mean(auc)

  }

  # For Data 3 (Balanced), just with 1st CV best ntrees = 1000, best mtry = 15 and mauc_cv = 0.9777307

  index_a <- which(mauc_cv == max(mauc_cv), arr.ind = T)

  best_trees_f<- bestntrees_a[index_a]
  best_mt_f <- bestmtry_a[index_a]

  rf_trees[[l]] <- best_trees_f
  rf_mt[[l]] <- best_mt_f

  # best_msplit <- which.max(cv_accuracy2)
  # # min split should be 10 and corresponding cp
  # best_cp <- cp_final[best_msplit]

  # Lets train the final model
  rf_final <- ranger(isFraud~., data = Train_int, num.trees = best_trees_f, mtry = best_mt_f, sample.fractio
n = 1,
   importance = 'impurity')

  save(rf_final, file = paste("C:/A_Work/UIC/Internship/Capital One/Data_Science_Challenge/rf", as.character
(l),
  ".rda", sep = ""))
  # Saving models for future use

  score_final <- predict(rf, data = oos_bm[,varf], predict.all = FALSE, type = "response", num.trees = rf_fi
nal$num.trees)
  score_final <- ifelse(score_final$predictions > 0.5,1,0)
  score_final_oot <- predict(rf, data = oot_bm[,varf], predict.all = FALSE, type = "response", num.trees = r
f_final$num.trees)
  score_final_oot <- ifelse(score_final_oot$predictions > 0.5,1,0)

  rf_oos[[l]] <- score_final
  rf_oot[[l]] <- score_final_oot

  confusion_dts <- table(pred = score_final, true=oos_bm$isFraud)
  confusion_dtt <- table(pred = score_final_oot, true=oot_bm$isFraud)

  rf_confs[[l]] <- confusion_dts
  rf_conft[[l]] <- confusion_dtt

  lifts1 <- lift_model(oos_bm$isFraud, score_final,50)[1]
  liftt1 <- lift_model(oot_bm$isFraud, score_final_oot,50)[1]

  lifts5 <- lift_model(oos_bm$isFraud, score_final,50)[5]
  liftt5 <- lift_model(oot_bm$isFraud, score_final_oot,50)[5]

  rf_lifts1[[l]] <- lifts1
  rf_liftt1[[l]] <- liftt1

  rf_lifts5[[l]] <- lifts5
  rf_liftt5[[l]] <- liftt5
```

```
  aucs <- calc_auc(oos_bm$isFraud, score_final)
  auct <- calc_auc(oot_bm$isFraud, score_final_oot)

  rf_aucs[[l]] <- aucs
  rf_auct[[l]] <- auct

}
```

```
## Growing trees.. Progress: 85%. Estimated remaining time: 5 seconds.
## Growing trees.. Progress: 47%. Estimated remaining time: 35 seconds.
## Growing trees.. Progress: 95%. Estimated remaining time: 3 seconds.
## Growing trees.. Progress: 37%. Estimated remaining time: 53 seconds.
## Growing trees.. Progress: 79%. Estimated remaining time: 16 seconds.
## Growing trees.. Progress: 34%. Estimated remaining time: 59 seconds.
## Growing trees.. Progress: 67%. Estimated remaining time: 30 seconds.
## Growing trees.. Progress: 23%. Estimated remaining time: 1 minute, 46 seconds.
## Growing trees.. Progress: 43%. Estimated remaining time: 1 minute, 20 seconds.
## Growing trees.. Progress: 64%. Estimated remaining time: 52 seconds.
## Growing trees.. Progress: 86%. Estimated remaining time: 19 seconds.
## Growing trees.. Progress: 15%. Estimated remaining time: 2 minutes, 50 seconds.
## Growing trees.. Progress: 33%. Estimated remaining time: 2 minutes, 8 seconds.
## Growing trees.. Progress: 50%. Estimated remaining time: 1 minute, 33 seconds.
## Growing trees.. Progress: 67%. Estimated remaining time: 1 minute, 1 seconds.
## Growing trees.. Progress: 84%. Estimated remaining time: 30 seconds.
## Growing trees.. Progress: 23%. Estimated remaining time: 1 minute, 44 seconds.
## Growing trees.. Progress: 48%. Estimated remaining time: 1 minute, 8 seconds.
## Growing trees.. Progress: 71%. Estimated remaining time: 37 seconds.
## Growing trees.. Progress: 95%. Estimated remaining time: 6 seconds.
## Growing trees.. Progress: 15%. Estimated remaining time: 3 minutes, 8 seconds.
## Growing trees.. Progress: 30%. Estimated remaining time: 2 minutes, 27 seconds.
## Growing trees.. Progress: 46%. Estimated remaining time: 1 minute, 50 seconds.
## Growing trees.. Progress: 62%. Estimated remaining time: 1 minute, 16 seconds.
## Growing trees.. Progress: 77%. Estimated remaining time: 45 seconds.
## Growing trees.. Progress: 92%. Estimated remaining time: 15 seconds.
## Growing trees.. Progress: 10%. Estimated remaining time: 4 minutes, 27 seconds.
## Growing trees.. Progress: 21%. Estimated remaining time: 3 minutes, 50 seconds.
## Growing trees.. Progress: 33%. Estimated remaining time: 3 minutes, 12 seconds.
## Growing trees.. Progress: 43%. Estimated remaining time: 2 minutes, 42 seconds.
## Growing trees.. Progress: 54%. Estimated remaining time: 2 minutes, 13 seconds.
## Growing trees.. Progress: 65%. Estimated remaining time: 1 minute, 41 seconds.
## Growing trees.. Progress: 75%. Estimated remaining time: 1 minute, 13 seconds.
## Growing trees.. Progress: 85%. Estimated remaining time: 45 seconds.
## Growing trees.. Progress: 99%. Estimated remaining time: 3 seconds.
## Growing trees.. Progress: 98%. Estimated remaining time: 0 seconds.
## Growing trees.. Progress: 67%. Estimated remaining time: 15 seconds.
## Growing trees.. Progress: 44%. Estimated remaining time: 40 seconds.
## Growing trees.. Progress: 95%. Estimated remaining time: 3 seconds.
## Growing trees.. Progress: 42%. Estimated remaining time: 42 seconds.
## Growing trees.. Progress: 83%. Estimated remaining time: 12 seconds.
## Growing trees.. Progress: 26%. Estimated remaining time: 1 minute, 26 seconds.
## Growing trees.. Progress: 58%. Estimated remaining time: 45 seconds.
## Growing trees.. Progress: 90%. Estimated remaining time: 10 seconds.
## Growing trees.. Progress: 22%. Estimated remaining time: 1 minute, 49 seconds.
## Growing trees.. Progress: 45%. Estimated remaining time: 1 minute, 17 seconds.
## Growing trees.. Progress: 66%. Estimated remaining time: 47 seconds.
## Growing trees.. Progress: 87%. Estimated remaining time: 18 seconds.
## Growing trees.. Progress: 29%. Estimated remaining time: 1 minute, 17 seconds.
## Growing trees.. Progress: 58%. Estimated remaining time: 45 seconds.
## Growing trees.. Progress: 85%. Estimated remaining time: 16 seconds.
## Growing trees.. Progress: 18%. Estimated remaining time: 2 minutes, 21 seconds.
## Growing trees.. Progress: 37%. Estimated remaining time: 1 minute, 47 seconds.
## Growing trees.. Progress: 56%. Estimated remaining time: 1 minute, 13 seconds.
## Growing trees.. Progress: 74%. Estimated remaining time: 43 seconds.
## Growing trees.. Progress: 95%. Estimated remaining time: 8 seconds.
## Growing trees.. Progress: 14%. Estimated remaining time: 3 minutes, 16 seconds.
## Growing trees.. Progress: 27%. Estimated remaining time: 2 minutes, 46 seconds.
## Growing trees.. Progress: 41%. Estimated remaining time: 2 minutes, 14 seconds.
## Growing trees.. Progress: 54%. Estimated remaining time: 1 minute, 45 seconds.
## Growing trees.. Progress: 67%. Estimated remaining time: 1 minute, 15 seconds.
## Growing trees.. Progress: 81%. Estimated remaining time: 43 seconds.
## Growing trees.. Progress: 95%. Estimated remaining time: 12 seconds.
```

```
## Growing trees.. Progress: 75%. Estimated remaining time: 10 seconds.
## Growing trees.. Progress: 49%. Estimated remaining time: 32 seconds.
## Growing trees.. Progress: 41%. Estimated remaining time: 44 seconds.
## Growing trees.. Progress: 84%. Estimated remaining time: 11 seconds.
## Growing trees.. Progress: 29%. Estimated remaining time: 1 minute, 15 seconds.
## Growing trees.. Progress: 58%. Estimated remaining time: 45 seconds.
## Growing trees.. Progress: 83%. Estimated remaining time: 18 seconds.
## Growing trees.. Progress: 20%. Estimated remaining time: 2 minutes, 7 seconds.
## Growing trees.. Progress: 42%. Estimated remaining time: 1 minute, 26 seconds.
## Growing trees.. Progress: 62%. Estimated remaining time: 56 seconds.
## Growing trees.. Progress: 83%. Estimated remaining time: 25 seconds.
## Growing trees.. Progress: 30%. Estimated remaining time: 1 minute, 12 seconds.
## Growing trees.. Progress: 60%. Estimated remaining time: 42 seconds.
## Growing trees.. Progress: 88%. Estimated remaining time: 12 seconds.
## Growing trees.. Progress: 17%. Estimated remaining time: 2 minutes, 33 seconds.
## Growing trees.. Progress: 34%. Estimated remaining time: 2 minutes, 0 seconds.
## Growing trees.. Progress: 52%. Estimated remaining time: 1 minute, 24 seconds.
## Growing trees.. Progress: 71%. Estimated remaining time: 51 seconds.
## Growing trees.. Progress: 90%. Estimated remaining time: 17 seconds.
## Growing trees.. Progress: 13%. Estimated remaining time: 3 minutes, 35 seconds.
## Growing trees.. Progress: 27%. Estimated remaining time: 2 minutes, 54 seconds.
## Growing trees.. Progress: 40%. Estimated remaining time: 2 minutes, 18 seconds.
## Growing trees.. Progress: 54%. Estimated remaining time: 1 minute, 47 seconds.
## Growing trees.. Progress: 67%. Estimated remaining time: 1 minute, 16 seconds.
## Growing trees.. Progress: 81%. Estimated remaining time: 45 seconds.
## Growing trees.. Progress: 94%. Estimated remaining time: 13 seconds.
## Growing trees.. Progress: 14%. Estimated remaining time: 3 minutes, 8 seconds.
## Growing trees.. Progress: 28%. Estimated remaining time: 2 minutes, 36 seconds.
## Growing trees.. Progress: 43%. Estimated remaining time: 2 minutes, 2 seconds.
## Growing trees.. Progress: 58%. Estimated remaining time: 1 minute, 31 seconds.
## Growing trees.. Progress: 73%. Estimated remaining time: 59 seconds.
## Growing trees.. Progress: 87%. Estimated remaining time: 27 seconds.
```

```
# Looking at confusion matrix of Decision Tree and Random Forest model here apart from checking AUC or Gini
(Rank Ordering power), it is clear Decision Tree has better performance with 50% undersampling data as compa
red to Random Forest in both Out of Sample and Out of Time data. Also, Random Forest was computationally and
time expensive as compared to Decision Trees here.

# 50% undersamppling has outperformed all other datasets with both the methods


lr <- which(rf_aucs == max(unlist(rf_aucs), na.rm = T), arr.ind = T)
lr1 <- which(rf_auct == max(unlist(rf_auct), na.rm = T), arr.ind = T)

print(paste("According to AUC metric on OOS model", as.character(lr), "is best"))
```

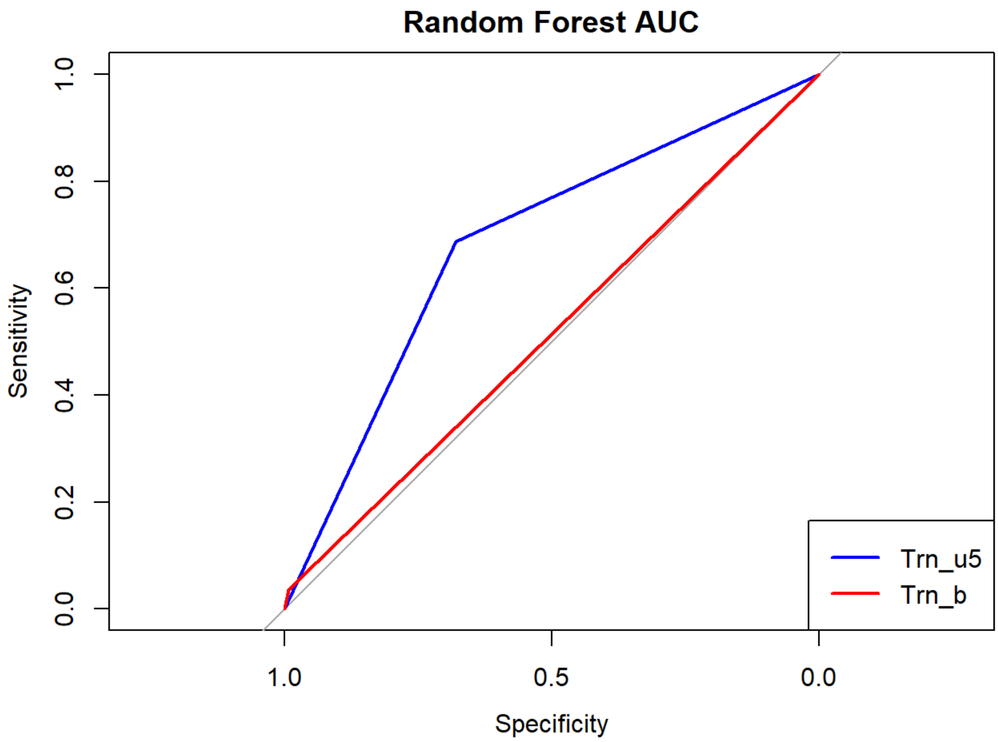
```
## [1] "According to AUC metric on OOS model 1 is best"
```

```
print(paste("According to AUC metric on oot model", as.character(lr1), "is best"))
```

```
## [1] "According to AUC metric on oot model 1 is best"
```

50% undersampled data has given best results according to area under the curve metric.

```
# Let's Plot ROC Curves for RF

# OOS
plot(roc(as.numeric(oot_bm$isFraud), as.numeric(rf_oot[[1]])),print.auc = F, col = "blue", main = "Random Fo
rest AUC" )
plot(roc(as.numeric(oot_bm$isFraud), as.numeric(rf_oot[[2]])),print.auc = F, col = "red", add = TRUE, print.
auc.y = .4)
legend("bottomright", legend=c("Trn_u5", "Trn_b"), col=c("blue", "red"), lwd=2)
```



Again area under the curve for 50% undersampling is highest!

Confusion Matrix of RF constructed for Out of Sample data Also, True Negative cases are only!

```
rf_confs
```

```
## [[1]]
##      true
## pred       0      1
##     0 108812    906
##     1  57418   2118
##
## [[2]]
##      true
## pred       0      1
##     0 164688   2875
##     1   1542    149
```

we can see final tuned model on 50% undersampled data has best predictions. There are 2088 of 3024 frauds (69%) cases predicted correctly in out of sample data!

Lets check the confusion matrix for Out of Time (December Transactions kept separate for OOT Validation)! The true positive are higest for 50% undersampled data at and cases of True negative are also low at

```
rf_conft
```

```
## [[1]]
##      true
## pred      0     1
##     0 36774   269
##     1 17353   590
##
## [[2]]
##      true
## pred      0     1
##     0 53714   828
##     1   413    31
```

we can see final tuned model on 50% undersampled data has best predictions. There are 577 of 859 frauds (67.17%) cases predicted correctly in out of time data!

Area under the curve (Gini = 2*AUC -1)!

```
print(paste("Area under the curve for best Random Forest model with 50% undersampled data on out of sample v
alidation is", rf_aucs[[lr]]))
```

```
## [1] "Area under the curve for best Random Forest model with 50% undersampled data on out of sample valida
tion is 0.677491921691976"
```

```
print(paste("Area under the curve for best Random Forest model with 50% undersampled data on out of time val
idation is", rf_auct[[lr]]))
```

```
## [1] "Area under the curve for best Random Forest model with 50% undersampled data on out of time validati
on is 0.68312365780191"
```

Lift in top 2 percentile population

```
print(paste("lift in top 2 percentile population for best Random Forest model with 50% undersampled data on
out of sample validation is", rf_lifts1[[lr]]))
```

```
## [1] "lift in top 2 percentile population for best Random Forest model with 50% undersampled data on out o
f sample validation is 2.14952169937399"
```

```
print(paste("lift in top 2 percentile population for best Random Forest model with 50% undersampled data on
out of time validation is", rf_liftt1[[lr]]))
```

```
## [1] "lift in top 2 percentile population for best Random Forest model with 50% undersampled data on out o
f time validation is 2.15507801038302"
```

Lift in top 10 percentile population

```
print(paste("lift in top 10 percentile population for best Random Forest model with 50% undersampled data on
out of sample validation is", rf_lifts5[[lr]]))
```

```
## [1] "lift in top 10 percentile population for best Random Forest model with 50% undersampled data on out
of sample validation is 1.95441126820004"
```

```
print(paste("lift in top 10 percentile population for best Random Forest model with 50% undersampled data on
out of time validation is", rf_liftt5[[lr1]]))
```

```
## [1] "lift in top 10 percentile population for best Random Forest model with 50% undersampled data on out
of time validation is 2.10848172907744"
```

Best parameter for this decision tree built on 50% undersampled data and according to AUC are

```
rf_trees[[lr]]
```

```
## [1] 500
```

```
rf_mt[[lr]]
```

```
## [1] 5
```

Variable Importance of Best Decision Tree Constructed!

```
benchmarkrf <- load(file = "C:/A_Work/UIC/Internship/Capital One/Data_Science_Challenge/rf1.rda")
#rf_final$variable.importance
varimp <- as.data.frame(rf_final$variable.importance)
colnames(varimp) <- "imp"
varimp$var <- rownames(varimp)
rownames(varimp) <- c(1:27)

varimp <- varimp[with(varimp,order(-imp)),]
varimp
```

```
##          imp                                  var
## 2  401.738345                    transactionAmount
## 4  204.767155                       currentBalance
## 6  186.392935                            ageMonths
## 7  126.136625                           expireYear
## 1  103.054303                          creditLimit
## 14  82.875595                     posEntryMode.05
## 13  40.954267                     posEntryMode.09
## 5   24.762429                          cardPresent
## 22  23.718787 merchantCategoryCode.leisureActivities
## 27  19.533093            merchantCategoryCode.travel
## 25  18.653673     merchantCategoryCode.online_retail
## 26  18.286572             merchantCategoryCode.autor
## 12  17.840587                     posEntryMode.02
## 19  16.782861                 posConditionCode.08
## 18  16.018912                 posConditionCode.01
## 23  13.829873           merchantCategoryCode.healthr
## 24  13.466833         merchantCategoryCode.rideshare
## 16  10.484537                posEntryMode.missing
## 10   7.632854              merchantCountryCode.US
## 17   7.500622                     posEntryMode.90
## 3    7.227344                     transactionType
## 21   7.131138             merchantCategoryCode.Home
## 20   7.116103                 posConditionCode.99
## 15   6.361504                     posEntryMode.80
## 8    5.589048                              cvvInd
## 9    4.058789                 acqCountry.missing
## 11   2.435092            merchantCountryCode.missing
```

Transaction Amount, current balance and age in month are among top predictors in random forest.

Benchmark Model #3 XGBoost (Hypertuning is done using single 70-30 split and not cross validation for this model!)

```
library(xgboost)
data <- list(Trn_u5, Trn_b)

var_xg <- varf[1:27]

xg_oos <- list()
xg_oot <- list()
xg_confs <- list()
xg_conft <- list()
xg_lifts1 <- list()
xg_liftt1 <- list()
xg_lifts5 <- list()
xg_liftt5 <- list()
xg_aucs <- list()
xg_auct <- list()
xg_depth <- list()
xg_eta <- list()
xauc <- list()

doos <- xgb.DMatrix(data=as.matrix(oos_bm[,var_xg]), label = oos_bm$isFraud)
doot <- xgb.DMatrix(data=as.matrix(oot_bm[,var_xg]), label = oot_bm$isFraud)


for (l in 1:length(data)){

  df <- as.data.frame(data[l])
  Train <- df[,var_xg]

  dtrain <- xgb.DMatrix (data=as.matrix(Train), label = df$isFraud)

  max_depth <- c(3, 4, 5, 6)
  eta <- c(0.001, 0.01, 0.1, 0.2)

  auc_1 <- matrix(nrow = length(max_depth), ncol = length(eta))

  ind <- sample(2, nrow(Train), replace = T, prob = c(0.7, 0.3))
  df_t <- df[ind==1, ]
  df_tv <- df[ind==2, ]
  Train_t <- Train[ind == 1, ]
```

```
  Validation_t <- Train[ind == 2, ]

  for (j in 1:length(max_depth))
    {
    max_depth_1 <- max_depth[j]
    for (k in 1:length(eta)){
      eta_1 <- eta[k]

      input_cv <- sapply(Train_t, as.numeric)
      inputv_cv <- sapply(Validation_t, as.numeric)

      dtrain_cv <- xgb.DMatrix (data=as.matrix(input_cv), label = df_t$isFraud)
      dvalid_cv <- xgb.DMatrix (data=as.matrix(inputv_cv), label = df_tv$isFraud)

      watchlist_cv <- list (eval = dvalid_cv, train= dtrain_cv)

      posweight_cv <- sum(df_t$isFraud == 1)
      negweight_cv <- sum(df_t$isFraud == 0)

      param_cv <- list (max.depth = max_depth_1, eta = eta_1, silent = 0, objective = "multi:softprob",
      maximize = TRUE, eval_metric="mlogloss", num_class=2)

      bigxgboost_cv <- xgb.train (params = param_cv, data = dtrain_cv, nrounds=100, watchlist=watchlist_cv,
       verbose = 0, scale_pos_weight = negweight_cv/posweight_cv)

      prediction_cv <- predict (bigxgboost_cv, dvalid_cv, reshape = T)

      xgb.pred.cv = as.data.frame(prediction_cv)
      colnames(xgb.pred.cv) = c(0:1)

      xgb.pred.cv$prediction_cv = apply(xgb.pred.cv,1,function(x) colnames(xgb.pred.cv)[which.max(x)])

      auc_1[j,k] <- calc_auc(df_tv$isFraud, as.numeric(xgb.pred.cv$prediction_cv))
    }
  }

  index <- which(auc_1 == max(auc_1, na.rm = T), arr.ind = T)
  index1 <- index[1,]

  best_depth <- max_depth[j]
  best_eta <- eta[k]

  xg_depth[[l]] <- best_depth
  xg_eta[[l]] <- best_eta

  xauc[l] <- mean(auc_1, na.rm = T)

  watchlist <- list(eval = doos, train= dtrain)

  posweight <- sum(df$isFraud == 1)
  negweight <- sum(df$isFraud == 0)

  param <- list (max.depth = best_depth, eta = best_eta, silent = 0, objective = "multi:softprob",
  maximize = TRUE, eval_metric="mlogloss", num_class=2)

  bigxgboost <- xgb.train (params = param, data = dtrain, nrounds=100, watchlist=watchlist,
   verbose = 0, scale_pos_weight = negweight/posweight)

  save(bigxgboost, file = paste("C:/A_Work/UIC/Internship/Capital One/Data_Science_Challenge/xg", as.charact
er(l),
  ".rda", sep = ""))

  prediction_oos <- predict(bigxgboost, doos, reshape = T)
  prediction_oot <- predict(bigxgboost, doot, reshape = T)

  xgb.pred.oos = as.data.frame(prediction_oos)
  colnames(xgb.pred.oos) = c(0:1)

  xgb.pred.oot = as.data.frame(prediction_oot)
  colnames(xgb.pred.oot) = c(0:1)

  xgb.pred.oos$prediction = apply(xgb.pred.oos,1,function(x) colnames(xgb.pred.oos)[which.max(x)])
  xgb.pred.oot$prediction = apply(xgb.pred.oot,1,function(x) colnames(xgb.pred.oot)[which.max(x)])
```

```
  confusion_xgs <- table(pred = as.numeric(xgb.pred.oos$prediction), true=oos_bm$isFraud)
  confusion_xgt <- table(pred = as.numeric(xgb.pred.oot$prediction), true=oot_bm$isFraud)

  xg_oos[[l]] <- as.numeric(xgb.pred.oos$prediction)
  xg_oot[[l]] <- as.numeric(xgb.pred.oot$prediction)

  xg_confs[[l]] <- confusion_xgs
  xg_conft[[l]] <- confusion_xgt

  xg_aucs[[l]] <- calc_auc(oos_bm$isFraud, as.numeric(xgb.pred.oos$prediction))
  xg_auct[[l]] <- calc_auc(oot_bm$isFraud, as.numeric(xgb.pred.oot$prediction))

  lifts <- lift_model(oos_bm$isFraud, as.numeric(xgb.pred.oos$prediction),50)

  xg_lifts1[[l]] <- lifts[1]
  xg_lifts5[[l]] <- lifts[5]

  lifft <- lift_model(oot_bm$isFraud, as.numeric(xgb.pred.oot$prediction),50)

  xg_liftt1[[l]] <- lifft[1]
  xg_liftt5[[l]] <- lifft[5]

}

lx <- which(xg_aucs == max(unlist(xg_aucs), na.rm = T), arr.ind = T)
lx1 <- which(xg_auct == max(unlist(xg_auct), na.rm = T), arr.ind = T)

print(paste("According to AUC metric on OOS, model", as.character(lx),"is", "best"))
```

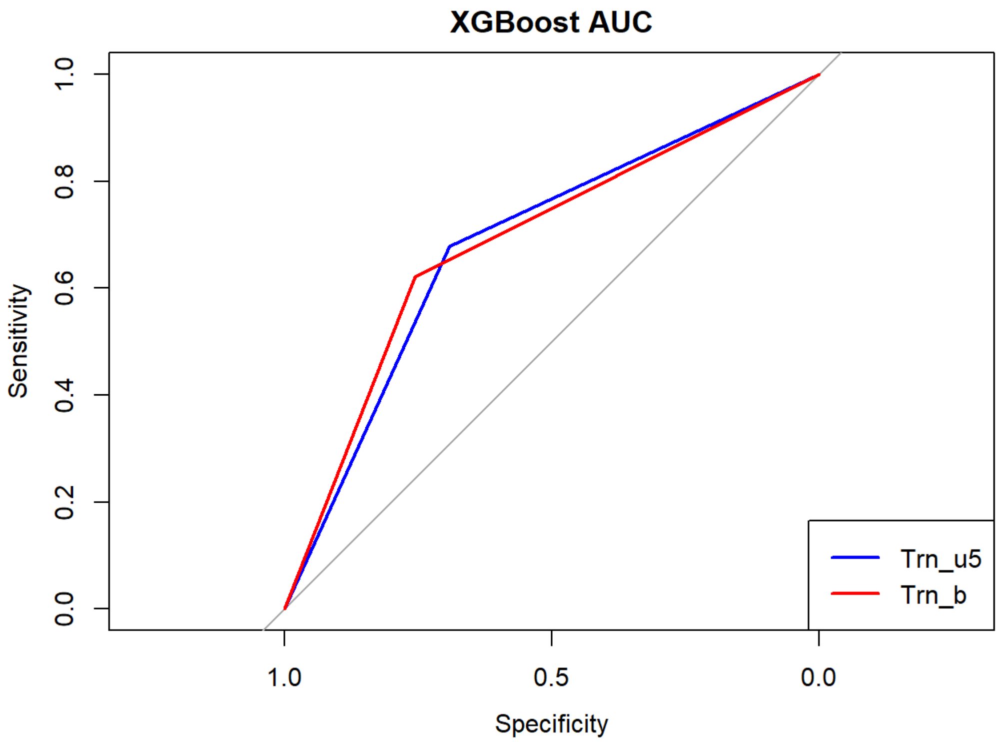
```
## [1] "According to AUC metric on OOS, model 1 is best"
```

```
print(paste("According to AUC metric on oot model", as.character(lx1),"is", "best"))
```

```
## [1] "According to AUC metric on oot model 2 is best"
```

Plotting AUC for XGBoost

```
plot(roc(as.numeric(oot_bm$isFraud), as.numeric(xg_oot[[1]])),print.auc = F, col = "blue", main = "XGBoost A
UC")
plot(roc(as.numeric(oot_bm$isFraud), as.numeric(xg_oot[[2]])),print.auc = F, col = "red", add = TRUE, print.
auc.y = .4)
legend("bottomright", legend=c("Trn_u5", "Trn_b"), col=c("blue", "red"), lwd=2)
```



Confusion Matrix of XGBoost constructed for Out of Sample data. Model built on 50% undersampled data looks best.

```
xg_confs
```

```
## [[1]]
##      true
## pred      0      1
##    0 110962    886
##    1  55268   2138
##
## [[2]]
##      true
## pred      0      1
##    0 122383   1112
##    1  43847   1912
```

we can see final tuned model on 50% undersampled data has best predictions. There are 2138 of 3024 (70.7%) frauds cases predicted correctly!

For out of sample data: Model built on 50% undersampled data predicted 2138 of 3024 cases (70.7%) Model built on balanced data predicted 1912 of 3024 cases (63.22)

Lets check the confusion matrix for Out of Time (December Transactions kept separate for OOT Validation)! The true positive are higest for 50% undersampled data at 583 (67.86%) and cases of True negative are also low at 276.

```
xg_conft
```

```
## [[1]]
##      true
## pred     0     1
##    0 37459   276
##    1 16668   583
##
## [[2]]
##      true
## pred     0     1
##    0 40907   325
##    1 13220   534
```

For out of Time data: Model built on 50% undersampled data predicted 583 of 859 cases (67.86%) Model built on balanced data predicted 534 of 859 cases (62.16%)

Area under the curve (Gini = 2*AUC -1)!

```
print(paste("Area under the curve for best XGBoost model with 50% undersampled data on out of sample validat
ion is", xg_aucs[[lx]]))
```

```
## [1] "Area under the curve for best XGBoost model with 50% undersampled data on out of sample validation i
s 0.68726574339054"
```

```
print(paste("Area under the curve for best XGBoostmodel with 50% undersampled data on out of time validation
is", xg_auct[[lx]]))
```

```
## [1] "Area under the curve for best XGBoostmodel with 50% undersampled data on out of time validation is 0
.685376863317598"
```

Lift in top 2 percentile population

```
print(paste("lift in top 2 percentile population for best XGBoost model with 50% undersampled data on out of
sample validation is", xg_lifts1[[lx]]))
```

```
## [1] "lift in top 2 percentile population for best XGBoost model with 50% undersampled data on out of samp
le validation is 2.36447386931139"
```

```
print(paste("lift in top 2 percentile population for best XGBoost model with 50% undersampled data on out of
time validation is", xg_liftt1[[lx]]))
```

```
## [1] "lift in top 2 percentile population for best XGBoost model with 50% undersampled data on out of time
validation is 2.56279547180684"
```

Lift in top 10 percentile population

```
print(paste("lift in top 10 percentile population for best XGBoost model with 50% undersampled data on out o
f sample validation is", xg_lifts5[[lx]]))
```

```
## [1] "lift in top 10 percentile population for best XGBoost model with 50% undersampled data on out of sam
ple validation is 2.01724344095097"
```

```
print(paste("lift in top 10 percentile population for best XGBoost model with 50% undersampled data on out o
f time validation is", xg_liftt5[[lx]]))
```

```
## [1] "lift in top 10 percentile population for best XGBoost model with 50% undersampled data on out of tim
e validation is 2.13177986973023"
```

Best parameter for this decision tree built on 50% undersampled data and according to AUC are

```
xg_depth[[lx]]
```

```
## [1] 6
```

```
xg_eta[[lx]]
```

```
## [1] 0.2
```

Variable Importance of Best Decision Tree Constructed!

```
benchmarkxg <- load(file = "C:/A_Work/UIC/Internship/Capital One/Data_Science_Challenge/xg1.rda")
#rf_final$variable.importance
xgb.importance(model = bigxgboost)
```

```
##                                      Feature        Gain        Cover
##  1:                       transactionAmount 0.370083843 0.2636561786
##  2:                          currentBalance 0.148997049 0.2630288615
##  3:                               ageMonths 0.121639142 0.1457887610
##  4:                         posEntryMode.05 0.098323249 0.0355927664
##  5:                             creditLimit 0.050533343 0.0376162983
##  6:                              expireYear 0.042946725 0.0336774512
##  7:                             cardPresent 0.021517153 0.0150452729
##  8: merchantCategoryCode.leisureActivities 0.020376974 0.0195024004
##  9:              merchantCategoryCode.autor 0.018664888 0.0106467744
## 10:                         posEntryMode.09 0.015247925 0.0161189452
## 11:     merchantCategoryCode.online_retail 0.011807279 0.0052909895
## 12:         merchantCategoryCode.rideshare 0.011099925 0.0059904634
## 13:            merchantCategoryCode.travel 0.009551842 0.0116722791
## 14:                         posEntryMode.02 0.008034359 0.0020042096
## 15:                    posEntryMode.missing 0.007805496 0.0231271413
## 16:                    posConditionCode.08 0.007239279 0.0150314237
## 17:                    posConditionCode.99 0.005374011 0.0260265308
## 18:              merchantCategoryCode.Home 0.004665852 0.0073900369
## 19:           merchantCategoryCode.healthr 0.004585128 0.0070013015
## 20:                 merchantCountryCode.US 0.003678449 0.0067982453
## 21:                         posEntryMode.80 0.003654547 0.0042188475
## 22:            merchantCountryCode.missing 0.003526995 0.0230383364
## 23:                    posConditionCode.01 0.003504766 0.0026978609
## 24:                         posEntryMode.90 0.002476872 0.0029156107
## 25:                                  cvvInd 0.002446850 0.0143007522
## 26:                      acqCountry.missing 0.001181713 0.0006460662
## 27:                         transactionType 0.001036348 0.0011761950
##                                      Feature        Gain        Cover
##       Frequency
##  1: 0.239256398
##  2: 0.229357798
##  3: 0.174070497
##  4: 0.016417190
##  5: 0.070980203
##  6: 0.085707388
##  7: 0.027764365
##  8: 0.018590053
##  9: 0.009174312
## 10: 0.014244326
## 11: 0.017624336
## 12: 0.013520039
## 13: 0.007484307
## 14: 0.012554322
## 15: 0.006035732
## 16: 0.007725736
## 17: 0.006035732
## 18: 0.004104297
## 19: 0.007242878
## 20: 0.003138580
## 21: 0.005552873
## 22: 0.004345727
## 23: 0.007242878
## 24: 0.005070014
## 25: 0.003621439
## 26: 0.001207146
## 27: 0.001931434
##       Frequency
```

Again with XGBoost also the top variables are in line with other models. Transaction amount being top predictor, age in months is another important variable.

After building various models, I have decided to use XGBoost model with 50% undersampled data as my final benchmark model. I will be using the same tuned parameters of the best model constructed. (depth = 6, eta = 0.2)

```
# Data: Trn_u5
# Parameters: depth = 6, eta = 0.2
```

6) Feature Engineering

Lets start building new features!

Data will be working with is 50% undersampled data Trn_u5, oos_bm, oot_bm

```
Trn_f <- Trn_u5[,varf]
oos_f <- oos_bm[,varf]
oot_f <- oot_bm[,varf]
Trn_bc <- Trn_bm

# Copy of Trn, oos, oot
Trn_c <- Trn
oos_c <- oos
oot_c <- oot

# Time of Day (Morning, Afternoon etc)
Trn_c$tranMorn <- ifelse(6 < as.numeric(Trn_c$Hour) & as.numeric(Trn_c$Hour) <= 12, 1, 0)
oos_c$tranMorn <- ifelse(6 < as.numeric(oos_c$Hour) & as.numeric(oos_c$Hour) <= 12, 1, 0)
oot_c$tranMorn <- ifelse(6 < as.numeric(oot_c$Hour) & as.numeric(oot_c$Hour) <= 12, 1, 0)

Trn_c$tranAftr <- ifelse(12 < as.numeric(Trn_c$Hour) & as.numeric(Trn_c$Hour) <= 18, 1, 0)
oos_c$tranAftr <- ifelse(12 < as.numeric(oos_c$Hour) & as.numeric(oos_c$Hour) <= 18, 1, 0)
oot_c$tranAftr <- ifelse(12 < as.numeric(oot_c$Hour) & as.numeric(oot_c$Hour) <= 18, 1, 0)

Trn_c$tranEven <- ifelse(18 < as.numeric(Trn_c$Hour) & as.numeric(Trn_c$Hour) <= 24, 1, 0)
oos_c$tranEven <- ifelse(18 < as.numeric(oos_c$Hour) & as.numeric(oos_c$Hour) <= 24, 1, 0)
oot_c$tranEven <- ifelse(18 < as.numeric(oot_c$Hour) & as.numeric(oot_c$Hour) <= 24, 1, 0)

Trn_c$tranNigh <- ifelse(24 < as.numeric(Trn_c$Hour) & as.numeric(Trn_c$Hour) <= 6, 1, 0)
oos_c$tranNigh <- ifelse(24 < as.numeric(oos_c$Hour) & as.numeric(oos_c$Hour) <= 6, 1, 0)
oot_c$tranNigh <- ifelse(24 < as.numeric(oot_c$Hour) & as.numeric(oot_c$Hour) <= 6, 1, 0)


# Card Utilization (availableMoney/creditLimit)
Trn_c$cardUtil <- Trn_c$availableMoney/Trn_c$creditLimit
oos_c$cardUtil <- oos_c$availableMoney/oos_c$creditLimit
oot_c$cardUtil <- oot_c$availableMoney/oot_c$creditLimit

# Transaction amount to Limit Ration
Trn_c$transToLimit <- Trn_c$transactionAmount/Trn_c$creditLimit
oos_c$transToLimit <- oos_c$transactionAmount/oos_c$creditLimit
oot_c$transToLimit <- oot_c$transactionAmount/oot_c$creditLimit

# Transaction amount to Available Limit
Trn_c$transToAvail <- Trn_c$transactionAmount/Trn_c$availableMoney
oos_c$transToAvail <- oos_c$transactionAmount/oos_c$availableMoney
oot_c$transToAvail <- oot_c$transactionAmount/oot_c$availableMoney

# acqCountry==merchantCountry indicator
Trn_c$acqToMerInd <- ifelse(Trn_c$acqCountry == Trn_c$merchantCountryCode, 1, 0)
oos_c$acqToMerInd <- ifelse(oos_c$acqCountry == oos_c$merchantCountryCode, 1, 0)
oot_c$acqToMerInd <- ifelse(oot_c$acqCountry == oot_c$merchantCountryCode, 1, 0)

# Neighborhood expenditure
# Avg spend, limit etc on posentry and condition code

# Age of account already created and in top variables (Age in Months)

# Number of transactions on previous day on same card and on same account

# Previous montly average expenditure

# Count of transactions and avg transaction amount on the same day
countTran <- df_model_f %>% group_by(cardLast4Digits, cardCVV) %>% tally()
colnames(countTran)[3] <- "countTran"
avgTran <- df_model_f %>% group_by(cardLast4Digits, cardCVV) %>% summarise(avgTran = mean(transactionAmount)
)
colnames(avgTran)[3] <- "avgTran"

Trn_c <- left_join(Trn_c, countTran , by = c("cardLast4Digits","cardCVV"))
oos_c <- left_join(oos_c, countTran , by = c("cardLast4Digits", "cardCVV"))
oot_c <- left_join(oot_c, countTran , by = c("cardLast4Digits", "cardCVV"))

Trn_c <- left_join(Trn_c, avgTran, by = c("cardLast4Digits", "cardCVV"))
```

```r
oos_c <- left_join(oos_c, avgTran, by = c("cardLast4Digits", "cardCVV"))
oot_c <- left_join(oot_c, avgTran, by = c("cardLast4Digits", "cardCVV"))

# A few neighborhood variables (transaction amount, availableMoney) (based on acqcountry, merchantCategoryCode , posEntryMode, posConditionCode, card present)

df_nbh <- df_model_f
df_nbh$merchantCategoryCode <- recode(df_nbh[, 'merchantCategoryCode'], "c('fastfood', 'food', 'food_delivery') = 'foodr'; c('airline', 'hotels') = 'travel'; c('personal care', 'health') = 'healthr'; c('auto', 'fuel') = 'autor';
c('airline', 'hotels') = 'travel'; c('cable/phone', 'furniture') = 'Home'; c('entertainment', 'online_subscriptions' , 'subscriptions', 'gym', 'mobileapps', 'online_gifts') = 'leisureActivities'")

df_nbh$cardPresent <- ifelse(df_nbh$cardPresent=="TRUE", 1, 0)

nbhAmount <- df_nbh %>% group_by(acqCountry, merchantCategoryCode , posEntryMode, posConditionCode, cardPresent) %>% summarise(nbhAmount = mean(transactionAmount, na.rm = T))

nbhMoney <- df_nbh %>% group_by(acqCountry, merchantCategoryCode , posEntryMode, posConditionCode, cardPresent) %>% summarise(nbhMoney = mean(availableMoney))

Trn_c <- left_join(Trn_c, nbhAmount , by = c("acqCountry", "merchantCategoryCode" , "posEntryMode", "posConditionCode", "cardPresent"))
oos_c <- left_join(oos_c, nbhAmount , by = c("acqCountry", "merchantCategoryCode" , "posEntryMode", "posConditionCode", "cardPresent"))
oot_c <- left_join(oot_c, nbhAmount , by = c("acqCountry", "merchantCategoryCode" , "posEntryMode", "posConditionCode", "cardPresent"))

Trn_c <- left_join(Trn_c, nbhMoney , by = c("acqCountry", "merchantCategoryCode" , "posEntryMode", "posConditionCode", "cardPresent"))
oos_c <- left_join(oos_c, nbhMoney , by = c("acqCountry", "merchantCategoryCode" , "posEntryMode", "posConditionCode", "cardPresent"))
oot_c <- left_join(oot_c, nbhMoney , by = c("acqCountry", "merchantCategoryCode" , "posEntryMode", "posConditionCode", "cardPresent"))
```

Now as new features are ready, I would like to use it on Best and tuned Decision Tree model. I will be using same optmized parameter. We can perform variable selection and remove correlated variable. Which we can try in next step!

one hot encoding for catgeorical variables Trn_fm (Feature model) is created here!

```r
#one hot encoding for catgeorical variables
varEncod <- c("acqCountry", "merchantCountryCode", "posEntryMode", "posConditionCode", "merchantCategoryCode")
varNum <- c("creditLimit", "availableMoney", "transactionAmount", "transactionType", "currentBalance", "cardPresent",
"expirationDateKeyInMatch", "ageMonths", "expireYear", "cvvInd", "cardUtil", "transToLimit", "transToAvail", "acqToMerInd", "countTran", "avgTran", "nbhAmount", "nbhMoney", "tranMorn", "tranEven", "tranAftr", "tranNigh")
label <- "isFraud"


for (var in varEncod){
name_trn <- paste(var, "_cat_trn", sep = "")
name_oos <- paste(var, "_cat_oos", sep = "")
name_oot <- paste(var, "_cat_oot", sep = "")
assign(name_trn, unique(Trn_c[,var]))
assign(name_oos, unique(oos_c[,var]))
assign(name_oot, unique(oot_c[,var]))
}

acqCountry_trn_notoos <- acqCountry_cat_trn[which(!acqCountry_cat_trn %in% acqCountry_cat_oos)]
acqCountry_trn_notoot <- acqCountry_cat_trn[which(!acqCountry_cat_trn %in% acqCountry_cat_oot)]
acqCountry_oos_nottrn <- acqCountry_cat_oos[which(!acqCountry_cat_oos %in% acqCountry_cat_trn)]
acqCountry_oot_nottrn <- acqCountry_cat_oot[which(!acqCountry_cat_oot %in% acqCountry_cat_trn)]

merchantCountryCode_trn_notoos <- merchantCountryCode_cat_trn[which(!merchantCountryCode_cat_trn %in% merchantCountryCode_cat_oos)]
merchantCountryCode_trn_notoot <- merchantCountryCode_cat_trn[which(!merchantCountryCode_cat_trn %in% merchantCountryCode_cat_oot)]
merchantCountryCode_oos_nottrn <- merchantCountryCode_cat_oos[which(!merchantCountryCode_cat_oos %in% merchantCountryCode_cat_trn)]
merchantCountryCode_oot_nottrn <- merchantCountryCode_cat_oot[which(!merchantCountryCode_cat_oot %in% merchantCountryCode_cat_trn)]
```

```
ntCountryCode_cat_trn)]

posEntryMode_trn_notoos <- posEntryMode_cat_trn[which(!posEntryMode_cat_trn %in% posEntryMode_cat_oos)]
posEntryMode_trn_notoot <- posEntryMode_cat_trn[which(!posEntryMode_cat_trn %in% posEntryMode_cat_oot)]
posEntryMode_oos_nottrn <- posEntryMode_cat_oos[which(!posEntryMode_cat_oos %in% posEntryMode_cat_trn)]
posEntryMode_oot_nottrn <- posEntryMode_cat_oot[which(!posEntryMode_cat_oot %in% posEntryMode_cat_trn)]

posConditionCode_trn_notoos <- posConditionCode_cat_trn[which(!posConditionCode_cat_trn %in% posConditionCod
e_cat_oos)]
posConditionCode_trn_notoot <- posConditionCode_cat_trn[which(!posConditionCode_cat_trn %in% posConditionCod
e_cat_oot)]
posConditionCode_oos_nottrn <- posConditionCode_cat_oos[which(!posConditionCode_cat_oos %in% posConditionCod
e_cat_trn)]
posConditionCode_oot_nottrn <- posConditionCode_cat_oot[which(!posConditionCode_cat_oot %in% posConditionCod
e_cat_trn)]

merchantCategoryCode_trn_notoos <- merchantCategoryCode_cat_trn[which(!merchantCategoryCode_cat_trn %in% mer
chantCategoryCode_cat_oos)]
merchantCategoryCode_trn_notoot <- merchantCategoryCode_cat_trn[which(!merchantCategoryCode_cat_trn %in% mer
chantCategoryCode_cat_oot)]
merchantCategoryCode_oos_nottrn <- merchantCategoryCode_cat_oos[which(!merchantCategoryCode_cat_oos %in% mer
chantCategoryCode_cat_trn)]
merchantCategoryCode_oot_nottrn <- merchantCategoryCode_cat_oot[which(!merchantCategoryCode_cat_oot %in% mer
chantCategoryCode_cat_trn)]

# As all unique values for all categorical columns are available exhaustively in Training, Out of Sample and
Out of Time (If not we would require additional treatment) - All above variables created are empty, hence sa
fe for below one hot encoding of categorical variables

#One Hot for Trn
Trn_fm <- Trn_c[,varNum]

for (var in varEncod) {
  for (unique_value in unique(Trn_c[,var])){
  Trn_fm[paste(var, unique_value, sep = ".")] <- ifelse(Trn_c[,var]==unique_value,1,0)
  }
}

# One Hot for oos
oos_fm <- oos_c[,varNum]

for (var in varEncod) {
  for (unique_value in unique(oos_c[,var])){
  oos_fm[paste(var, unique_value, sep = ".")] <- ifelse(oos_c[,var]==unique_value,1,0)
  }
}

# One Hot for oot
oot_fm <- oot_c[,varNum]

for (var in varEncod) {
  for (unique_value in unique(oot_c[,var])){
  oot_fm[paste(var, unique_value, sep = ".")] <- ifelse(oot_c[,var]==unique_value,1,0)
  }
}


# As all unique values for all categorical columns are available exhaustively in Training, Out of Sample and
Out of Time (If not we would require additional treatment)
```

Data for benchmark model is now ready!

```
df_Trn_fm <- Trn_fm
df_oos_fm <- oos_fm
df_oot_fm <- oot_fm
```

Lets check Gini/Area Under the Curve or Rank Order power of our Independend variables with respect to dependent variable.

```r
Trn_fm$isFraud <- Trn_c$isFraud
oos_fm$isFraud <- oos_c$isFraud
oot_fm$isFraud <- oot_c$isFraud

Trn_fm$isFraud <- ifelse(Trn_fm$isFraud=="TRUE",1,0)
oos_fm$isFraud <- ifelse(oos_fm$isFraud=="TRUE",1,0)
oot_fm$isFraud <- ifelse(oot_fm$isFraud=="TRUE",1,0)

calc_auc <- function (actual, predicted)
{
r <- rank(predicted)
n_pos <- as.numeric (sum(actual == 1))
n_neg <- as.numeric (length(actual) - n_pos)
denom <- as.double (as.double (n_pos) * as.double(n_neg))
auc <- (sum(r[actual == 1]) - n_pos * (n_pos + 1)/2)/(denom)
auc
}

setDT(Trn_fm)

allnms <- colnames(Trn_fm)
allnms <- allnms[! allnms %in% c("isFraud")]

monkey = 1
actual <- ifelse (Trn_fm$isFraud==monkey,1,0)
aucDF <- Trn_fm[,allnms, with = F][,lapply(.SD, function (x) calc_auc (actual, x))]
aucDF <- as.data.frame (aucDF)
aucDF <- t (aucDF)
aucDF <- as.data.frame (aucDF)
aucDF$varName <- rownames (aucDF)
names (aucDF)[1] <- "auc"
aucDF$gini <- aucDF$auc
bigaucdf <- aucDF[, c("varName", "gini")]
rownames(bigaucdf) <- c(1:50)

bigaucdf$gini <- abs (bigaucdf$gini - 0.5)

# To understand which have maximum Gini
varSel_auc_0.001 <- bigaucdf$varName [which (bigaucdf$gini > 0.001)]
varSel_auc_0.007 <- bigaucdf$varName [which (bigaucdf$gini > 0.007)]
varSel_auc_0.03 <- bigaucdf$varName [which (bigaucdf$gini > 0.03)]
varSel_auc_0.01 <- bigaucdf$varName [which (bigaucdf$gini > 0.01)]
varSel_auc_0.05 <- bigaucdf$varName [which (bigaucdf$gini > 0.05)]
varSel_auc_0.08 <- bigaucdf$varName [which (bigaucdf$gin > 0.08)]

# Created multiple subset of variables based on area under the curve value (can also perform rank plots whic
h will give exactly same understanding on which predictors can perform best on given target variable)

bigaucdf <- bigaucdf[with(bigaucdf,order(-gini)),]
bigaucdf
```

```
##                                  varName         gini
## 3                      transactionAmount 1.884581e-01
## 12                          transToLimit 1.223826e-01
## 13                          transToAvail 1.121565e-01
## 35                      posEntryMode.05 1.088522e-01
## 34                      posEntryMode.09 8.991142e-02
## 16                              avgTran 3.647051e-02
## 48      merchantCategoryCode.online_retail 3.638185e-02
## 45 merchantCategoryCode.leisureActivities 3.620466e-02
## 18                             nbhMoney 3.264396e-02
## 6                           cardPresent 3.027042e-02
## 5                        currentBalance 2.943012e-02
## 17                            nbhAmount 2.938942e-02
## 15                            countTran 2.616431e-02
## 40                   posConditionCode.08 2.219006e-02
## 1                           creditLimit 1.744614e-02
## 11                             cardUtil 1.665357e-02
## 49            merchantCategoryCode.autor 1.659775e-02
## 39                   posConditionCode.01 1.653579e-02
## 50           merchantCategoryCode.travel 1.337761e-02
## 47        merchantCategoryCode.rideshare 1.015484e-02
## 4                       transactionType 9.388599e-03
## 33                      posEntryMode.02 9.355936e-03
## 37                 posEntryMode.missing 8.852770e-03
## 2                        availableMoney 7.713595e-03
## 8                             ageMonths 6.283249e-03
## 42                   posConditionCode.99 5.309580e-03
## 28                merchantCountryCode.US 4.973047e-03
## 23                         acqCountry.US 4.631774e-03
## 21                             tranAftr 4.275314e-03
## 20                             tranEven 4.118490e-03
## 46          merchantCategoryCode.healthr 3.741526e-03
## 10                               cvvInd 3.464098e-03
## 9                            expireYear 3.212792e-03
## 31           merchantCountryCode.missing 3.102532e-03
## 24                    acqCountry.missing 2.964466e-03
## 43             merchantCategoryCode.Home 2.601937e-03
## 38                      posEntryMode.90 1.871609e-03
## 36                      posEntryMode.80 1.139545e-03
## 32               merchantCountryCode.CAN 9.167888e-04
## 44            merchantCategoryCode.foodr 7.684355e-04
## 27                        acqCountry.CAN 7.071368e-04
## 19                             tranMorn 5.476881e-04
## 25                        acqCountry.MEX 5.148092e-04
## 29               merchantCountryCode.MEX 5.083649e-04
## 26                         acqCountry.PR 4.453616e-04
## 30                merchantCountryCode.PR 4.453616e-04
## 41              posConditionCode.missing 3.446928e-04
## 14                          acqToMerInd 3.163179e-05
## 7              expirationDateKeyInMatch 1.818218e-05
## 22                             tranNigh 0.000000e+00
```

Merging new features created with variables selected for previous modelling exercise

```
new_f <- c("cardUtil", "transToLimit", "transToAvail", "acqToMerInd", "countTran", "avgTran", "nbhAmount", "
nbhMoney", "tranMorn", "tranEven", "tranAftr", "tranNigh")
varf_f <- c(varf[1:27], new_f)
# This doesnot have label
varf_f
```

```
##  [1] "creditLimit"
##  [2] "transactionAmount"
##  [3] "transactionType"
##  [4] "currentBalance"
##  [5] "cardPresent"
##  [6] "ageMonths"
##  [7] "expireYear"
##  [8] "cvvInd"
##  [9] "acqCountry.missing"
## [10] "merchantCountryCode.US"
## [11] "merchantCountryCode.missing"
## [12] "posEntryMode.02"
## [13] "posEntryMode.09"
## [14] "posEntryMode.05"
## [15] "posEntryMode.80"
## [16] "posEntryMode.missing"
## [17] "posEntryMode.90"
## [18] "posConditionCode.01"
## [19] "posConditionCode.08"
## [20] "posConditionCode.99"
## [21] "merchantCategoryCode.Home"
## [22] "merchantCategoryCode.leisureActivities"
## [23] "merchantCategoryCode.healthr"
## [24] "merchantCategoryCode.rideshare"
## [25] "merchantCategoryCode.online_retail"
## [26] "merchantCategoryCode.autor"
## [27] "merchantCategoryCode.travel"
## [28] "cardUtil"
## [29] "transToLimit"
## [30] "transToAvail"
## [31] "acqToMerInd"
## [32] "countTran"
## [33] "avgTran"
## [34] "nbhAmount"
## [35] "nbhMoney"
## [36] "tranMorn"
## [37] "tranEven"
## [38] "tranAftr"
## [39] "tranNigh"
```

Keeping threshold of 70% correlation

```
df_Trn_f <- Trn_fm[, ..varf_f]
df_Trn_f <- as.data.frame(df_Trn_f)

df_cor_f <- data.frame(matrix(NA, nrow = 394925, ncol = 39))

bigaucdf <- bigaucdf[bigaucdf$varName %in% varf_f,]

colnames(df_cor_f) <- bigaucdf$varName


for (i in 1:length(bigaucdf$varName)){
  df_cor_f[,as.character(bigaucdf$varName[i])] <- df_Trn_f[,as.character(bigaucdf$varName[i])]
}

cor_inp_f <- sapply(df_cor_f, as.numeric)
cor_mat_f <- cor(cor_inp_f, use = "complete.obs")
```

```
## Warning in cor(cor_inp_f, use = "complete.obs"): the standard deviation is
## zero
```

```r
rm_cor_f <- vector() #remove correlated
kp_cor_f <- vector() #keep correlated

for (i in 1:(length(bigaucdf$varName)-1)){
temp <- data.frame(cor_mat_f[i, (i+1):39])
colnames(temp) <- "cor"
temp$var <- rownames(cor_mat_f)[(i+1):39]
rownames(temp) <- c(1:(39-i))
var1 <- as.character(temp$var[temp$cor > 0.70])
rm_cor_f <- union(rm_cor_f, var1)
}

# using varSel_auc_0.001 (variables selected using AUC/gini method); removing correlated variables calculate
d above from this list

varf_f
```

```
##  [1] "creditLimit"
##  [2] "transactionAmount"
##  [3] "transactionType"
##  [4] "currentBalance"
##  [5] "cardPresent"
##  [6] "ageMonths"
##  [7] "expireYear"
##  [8] "cvvInd"
##  [9] "acqCountry.missing"
## [10] "merchantCountryCode.US"
## [11] "merchantCountryCode.missing"
## [12] "posEntryMode.02"
## [13] "posEntryMode.09"
## [14] "posEntryMode.05"
## [15] "posEntryMode.80"
## [16] "posEntryMode.missing"
## [17] "posEntryMode.90"
## [18] "posConditionCode.01"
## [19] "posConditionCode.08"
## [20] "posConditionCode.99"
## [21] "merchantCategoryCode.Home"
## [22] "merchantCategoryCode.leisureActivities"
## [23] "merchantCategoryCode.healthr"
## [24] "merchantCategoryCode.rideshare"
## [25] "merchantCategoryCode.online_retail"
## [26] "merchantCategoryCode.autor"
## [27] "merchantCategoryCode.travel"
## [28] "cardUtil"
## [29] "transToLimit"
## [30] "transToAvail"
## [31] "acqToMerInd"
## [32] "countTran"
## [33] "avgTran"
## [34] "nbhAmount"
## [35] "nbhMoney"
## [36] "tranMorn"
## [37] "tranEven"
## [38] "tranAftr"
## [39] "tranNigh"
```

```r
# Finally we have 39 variables to build our benchmark model with!
```

Using finalized data (50% undersampling) as best results for this data in benchmark models!

```r
# library(ROSE)

Trn_fm %>% group_by(isFraud) %>% count()
```

```
## # A tibble: 2 x 2
## # Groups:   isFraud [2]
##   isFraud      n
##     <dbl>  <int>
## 1       0 387942
## 2       1   6983
```

```
#Undersampling majority class with 80% majority and 20% minority (currently minority is 1.7%)
Trn_fu5 <- ovun.sample(isFraud ~., data=Trn_fm, na.action = na.pass, method = "under", p=0.5)$data
Trn_fu5 %>% group_by(isFraud) %>% count()
```

```
## # A tibble: 2 x 2
## # Groups:   isFraud [2]
##   isFraud     n
##     <dbl> <int>
## 1       0  6975
## 2       1  6983
```

Building decision tree with new features

```
Train_f <- Trn_fu5[,varf_f]
Train_f <- Train_f[sample(1:nrow(Train_f)),]


doosf <- xgb.DMatrix(data=as.matrix(oos_fm[,varf_f]), label = oos_fm$isFraud)
dootf <- xgb.DMatrix(data=as.matrix(oot_fm[,varf_f]), label = oot_fm$isFraud)


dtrainf <- xgb.DMatrix (data=as.matrix(Train_f), label = Trn_fu5$isFraud)

watchlistf <- list(eval = doosf, train= dtrainf)

posweight <- sum(Trn_fu5$isFraud == 1)
negweight <- sum(Trn_fu5$isFraud == 0)

paramf <- list (max.depth = 6, eta = 0.2, silent = 0, objective = "multi:softprob", maximize = TRUE, eval_me
tric="mlogloss", num_class=2)

bigxgboostf <- xgb.train (params = paramf, data = dtrainf, nrounds=100, watchlist=watchlistf, verbose = 0, s
cale_pos_weight = negweight/posweight)

save(bigxgboostf, file = paste("C:/A_Work/UIC/Internship/Capital One/Data_Science_Challenge/xgf.rda"))

prediction_oosf <- predict(bigxgboostf, doosf, reshape = T)
prediction_ootf <- predict(bigxgboostf, dootf, reshape = T)

xgb.pred.oosf = as.data.frame(prediction_oosf)
colnames(xgb.pred.oosf) = c(0:1)

xgb.pred.ootf = as.data.frame(prediction_ootf)
colnames(xgb.pred.ootf) = c(0:1)

xgb.pred.oosf$predictionf = apply(xgb.pred.oosf,1,function(x) colnames(xgb.pred.oosf)[which.max(x)])
xgb.pred.ootf$predictionf = apply(xgb.pred.ootf,1,function(x) colnames(xgb.pred.ootf)[which.max(x)])

confusion_xgsf <- table(pred = as.numeric(xgb.pred.oosf$predictionf), true=oos_fm$isFraud)
confusion_xgtf <- table(pred = as.numeric(xgb.pred.ootf$predictionf), true=oot_fm$isFraud)

xg_oosf <- as.numeric(xgb.pred.oosf$predictionf)
xg_ootf <- as.numeric(xgb.pred.ootf$predictionf)

xg_confsf <- confusion_xgsf
xg_conftf <- confusion_xgtf

xg_aucsf <- calc_auc(oos_fm$isFraud, as.numeric(xgb.pred.oosf$predictionf))
xg_auctf <- calc_auc(oot_fm$isFraud, as.numeric(xgb.pred.ootf$predictionf))

liftsf <- lift_model(oos_fm$isFraud, as.numeric(xgb.pred.oosf$predictionf),50)

xg_liftsf1 <- liftsf[1]
xg_liftsf5 <- liftsf[5]

lifttf <- lift_model(oot_fm$isFraud, as.numeric(xgb.pred.ootf$prediction),50)

xg_lifttf1 <- lifttf[1]
xg_lifttf5 <- lifttf[5]
```

```
xg_confsf
```

```
##      true
## pred     0     1
##    0 83957  1550
##    1 82273  1474
```

The performance with new features have come down and we might need to tune our parameters and select variables efficiently!

For out of sample data: Model built on 50% undersampled data predicted 1216 of 3024 cases (Previously 2138)

Lets check the confusion matrix for Out of Time (December Transactions kept separate for OOT Validation)! The true positive are higest for 50% undersampled data at 308 (35.86%) which was earlier 68% and cases of True negative are also high at 551.

```
xg_conftf
```

```
##      true
## pred     0     1
##    0 26780   444
##    1 27347   415
```

For out of Time data: Model built on 50% undersampled data predicted 308 of 859 cases (35.86%)

Area under the curve (Gini = 2*AUC -1)!

```
print(paste("Area under the curve for best XGBoost model with 50% undersampled data on out of sample validat
ion is", xg_aucsf))
```

```
## [1] "Area under the curve for best XGBoost model with 50% undersampled data on out of sample validation i
s 0.496249566721954"
```

```
print(paste("Area under the curve for best XGBoostmodel with 50% undersampled data on out of time validation
is", xg_auctf))
```

```
## [1] "Area under the curve for best XGBoostmodel with 50% undersampled data on out of time validation is 0
.488941112559986"
```

AUC is also poor

Lift in top 2 percentile population

```
print(paste("lift in top 2 percentile population for best XGBoost model with 50% undersampled data on out of
sample validation is", xg_liftsf1))
```

```
## [1] "lift in top 2 percentile population for best XGBoost model with 50% undersampled data on out of samp
le validation is 1.10783041429275"
```

```
print(paste("lift in top 2 percentile population for best XGBoost model with 50% undersampled data on out of
time validation is", xg_lifttf1))
```

```
## [1] "lift in top 2 percentile population for best XGBoost model with 50% undersampled data on out of time
validation is 1.22315238427145"
```

Lift is also poor for this model

Lift in top 10 percentile population

```
print(paste("lift in top 10 percentile population for best XGBoost model with 50% undersampled data on out o
f sample validation is", xg_liftsf5))
```

```
## [1] "lift in top 10 percentile population for best XGBoost model with 50% undersampled data on out of sam
ple validation is 0.932561721882254"
```

```
print(paste("lift in top 10 percentile population for best XGBoost model with 50% undersampled data on out o
f time validation is", xg_lifttf5))
```

```
## [1] "lift in top 10 percentile population for best XGBoost model with 50% undersampled data on out of tim
e validation is 1.0950126106811"
```

Variable Importance of XGBoost!

```
xgb.importance(model = bigxgboostf)
```

```
##                                    Feature         Gain        Cover
##  1:                      transactionAmount 0.1134994176 1.158739e-01
##  2:                                avgTran 0.0988847462 1.623952e-01
##  3:                            transToLimit 0.0912799824 1.261770e-01
##  4:                               cardUtil 0.0911907982 1.208129e-01
##  5:                         currentBalance 0.0899802830 7.478666e-02
```

```
##  5:                         currentBalance 0.0899802830 7.478666e-02
##  6:                           transToAvail 0.0852099107 9.141893e-02
##  7:                              countTran 0.0824909569 5.740492e-02
##  8:                              ageMonths 0.0822751728 6.353096e-02
##  9:                              nbhAmount 0.0579815785 4.555893e-02
## 10:                             expireYear 0.0576174585 3.208969e-02
## 11:                               nbhMoney 0.0564114207 6.284937e-02
## 12:                            creditLimit 0.0139971128 3.656346e-03
## 13:                    posConditionCode.01 0.0094508486 5.523446e-04
## 14:                               tranAftr 0.0080038339 4.462560e-04
## 15:                               tranEven 0.0079688606 5.277303e-03
## 16:                               tranMorn 0.0072267579 7.108471e-04
## 17:                        posEntryMode.05 0.0061005733 2.450637e-03
## 18:                        posEntryMode.02 0.0060441036 6.549317e-04
## 19: merchantCategoryCode.leisureActivities 0.0055159200 3.235223e-03
## 20:         merchantCategoryCode.rideshare 0.0047470660 7.733749e-04
## 21:     merchantCategoryCode.online_retail 0.0047416224 2.988602e-04
## 22:            merchantCategoryCode.travel 0.0038083982 1.156652e-03
## 23:                        posEntryMode.09 0.0035737844 1.399038e-04
## 24:           merchantCategoryCode.healthr 0.0016545527 4.260022e-03
## 25:                            cardPresent 0.0015772276 2.777095e-05
## 26:                    posConditionCode.08 0.0014679753 2.805598e-04
## 27:                   posEntryMode.missing 0.0013061572 5.337919e-03
## 28:                        posEntryMode.90 0.0011973617 7.969985e-04
## 29:                        posEntryMode.80 0.0010575139 3.314693e-03
## 30:                                 cvvInd 0.0008925888 8.487578e-03
## 31:             merchantCategoryCode.autor 0.0008752165 6.746191e-05
## 32:                    posConditionCode.99 0.0008302900 1.127290e-04
## 33:              merchantCategoryCode.Home 0.0004518618 1.136718e-04
## 34:                 merchantCountryCode.US 0.0003513290 3.466996e-06
## 35:            merchantCountryCode.missing 0.0003373183 4.945922e-03
##                                    Feature         Gain        Cover
##        Frequency
##  1: 0.1125171939
##  2: 0.1031636864
##  3: 0.0850068776
##  4: 0.0883081155
##  5: 0.0858321871
##  6: 0.0742778542
##  7: 0.0830811554
##  8: 0.0839064649
##  9: 0.0605226960
## 10: 0.0552957359
## 11: 0.0610729023
## 12: 0.0165061898
## 13: 0.0093535076
## 14: 0.0079779917
## 15: 0.0077028886
## 16: 0.0090784044
## 17: 0.0068775791
## 18: 0.0066024759
## 19: 0.0060522696
## 20: 0.0057771664
## 21: 0.0063273728
## 22: 0.0038514443
## 23: 0.0044016506
## 24: 0.0016506190
## 25: 0.0049518569
## 26: 0.0013755158
## 27: 0.0013755158
## 28: 0.0013755158
## 29: 0.0008253095
## 30: 0.0019257221
## 31: 0.0005502063
## 32: 0.0008253095
## 33: 0.0005502063
## 34: 0.0002751032
## 35: 0.0008253095
##        Frequency
```

Variable Importance of new model is promising, showing confidence towards features created. It might require fine tuning but ideas will work!. XGBoost model was not hypertuned properly and variable correlation analysis is required which could not be

performed due to time constraint

Card Utility, Average Transaction amount even surpassed transaction amount in this model!

Also, Neighborhood variables created have appreared on top.

This importance looks promising. We need to try these variables on different models, fine tune models to see the exact performance of these variables