USE CASE STUDY REPORT

***Group No.*: 5**

***Student Names*:** Siddharth Muthe, Vinit Deshbhratar

Executive Summary

The project aims to build a machine learning model which is able to identify a fraudulent transaction from a pool of transactions. The data has been taken from the website Kaggle and the data has been made public by IEEE Computation Intelligence Society in coordination with Vesta Corporation. The data is in two files, one file contains the transaction details like the location, time, amount of the transaction, etc. and the other file contains the identity information like the devices on which the transaction was carried out. The target variable is Fraud is in the transaction file and is a boolean value represented by '0' showing no fraudulent transaction and '1' showing the record is a fraudulent transaction. The major difficulty we faced in the project was the columns have been named in codes, like C_1, C_2, D_1 etc. This made it difficult to understand what the data is about. The strategy we employed here is we clubbed the columns which were named same, like columns starting with C, D, M and V and applied correlation and PCA to remove the columns which were highly correlated. We were able to reduce the 394-column transaction dataset down to 89 columns. Identity dataset had a column DeviceInfo which had information about the devices which were used for transaction. It had 1786 different values in it. Suitable data cleaning was used to reduce the number of categories down to 23 categories. Appropriate imputation methods were used to impute the missing values. We used left outer join to merge the two datasets because not all TransactionID were present in the identity dataset. The dataset was also imbalanced. The number of fraudulent records were very less as compared to the number of non-fraudulent records. In order to build a model, we used different sampling techniques such as under-sampling, over-sampling and making the dataset balanced. We identified that under-sampling technique worked the best for our model. In order to choose the best machine learning model, we compared 7 different models that we built using Random Forest, CART, Naive Bayes, SVM, Logistic Regression, k-NN, and LDA. If the task was to identify most of the fraudulent transactions, we would recommend using the random forest model. The model gives a sensitivity of 90%. That means, it could identify 90% of the fraudulent cases. Though we understand that the performance of the model is not perfect, we recommend increasing the sample size of the sampled dataset to get more improved results. The memory computation limitations limited the progress of the project due to the high number of row counts.

I. Background Information and Introduction:

The count of fraudulent transactions has been rampantly increasing since the boom of the e-commerce sector. It is common that a person has all the bank details and other relevant information on the mobile phone, ready to be hacked. This has contributed the most to the increase in frauds. Credit cards are responsible for most of the fraudulent transactions in the US. Still credit card is considered one of the safest methods to do shopping, the reason being if a fraud happens the loss must be faced by the banks, making them financially vulnerable.

The project aims at doing the exact same thing, correctly identifying a fraudulent transaction from a set of transactions. The project doesn't target a type of transaction but tries to identify from a pool of transactions if a case is a fraud or not.

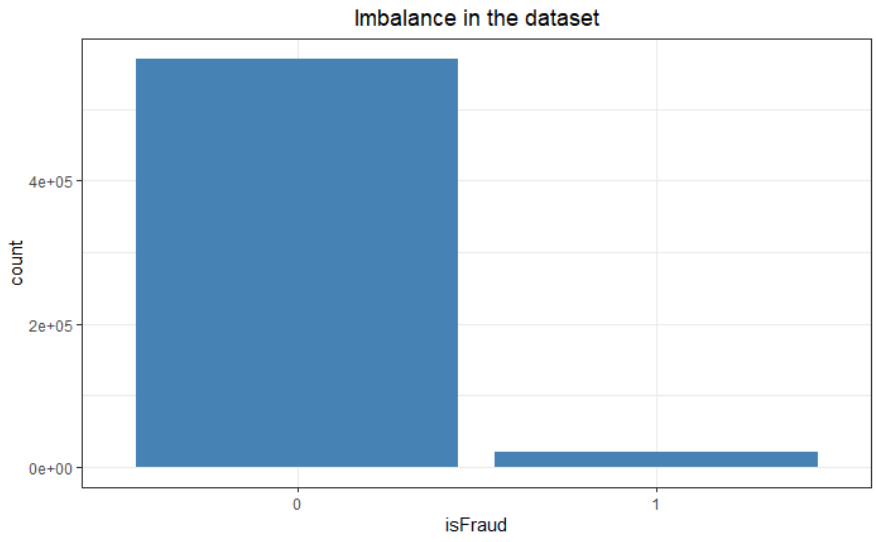
Various strategies have been used to find a possible solution to curb the fraudulent transactions. The possible solution for the problem can be to build a machine learning model to identify the trend of a particular transaction and find the possibility of the transaction being a fraud. This can help us alarm the authority of the possible fraud and help limit the damage caused because of this issue.

II. Data Exploration and Visualization

The dataset has been taken from the website Kaggle and can be found here. The dataset has 394 columns and 590540 rows for the transaction dataset and 144233 rows and 41 columns in the identity dataset. Both the datasets must be joined on the column TransactionID. Not all transactions have TransactionID. Exploratory analysis of the columns in the dataset is given below.

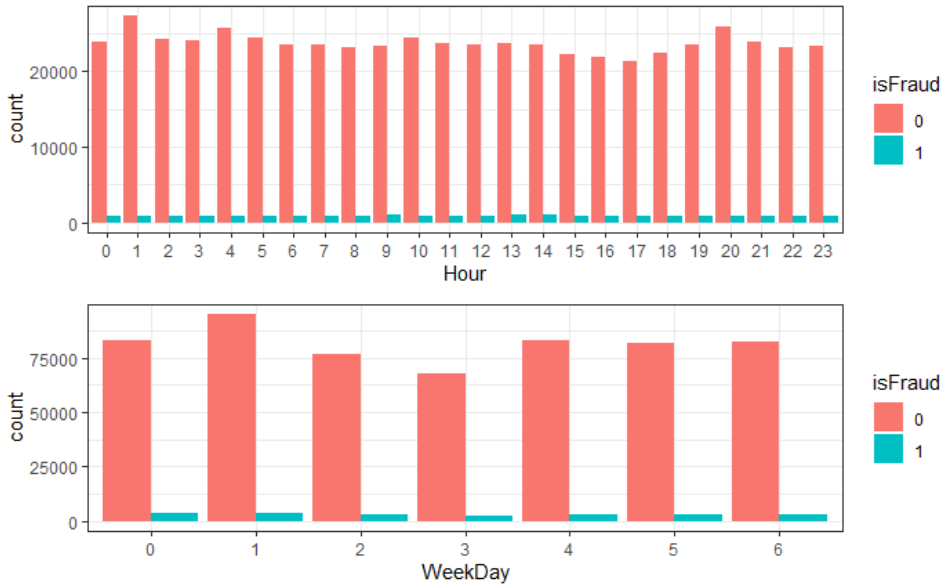
- **isFraud**

We started the exploratory analysis with the target variable to identify if the dataset is balanced or imbalanced. We concluded that the dataset is imbalanced.



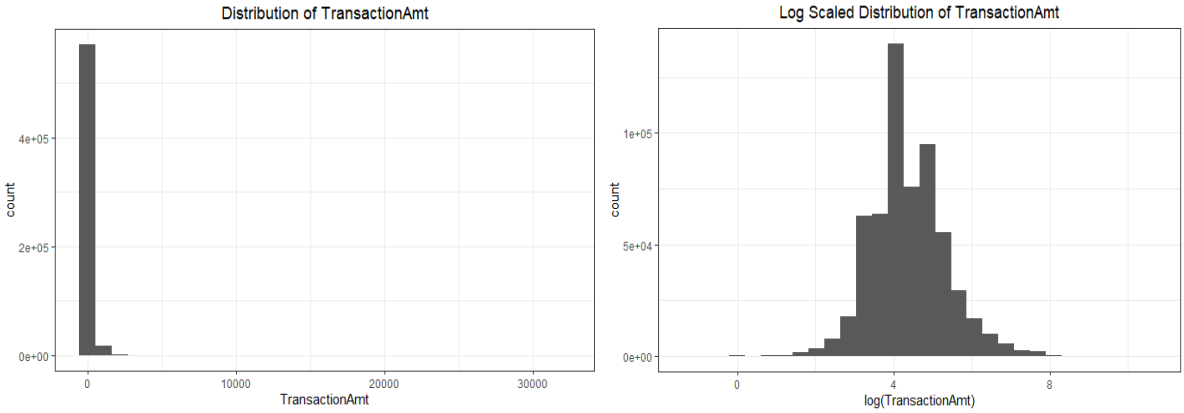
- **Transaction Date and Time**

The transaction time delta that was provided was from a given reference datetime. We employed feature engineering to create two features, hour of the day and week of the day from the feature. We identify that there is irregularity in the time and the day of the transaction. The column had no missing values.



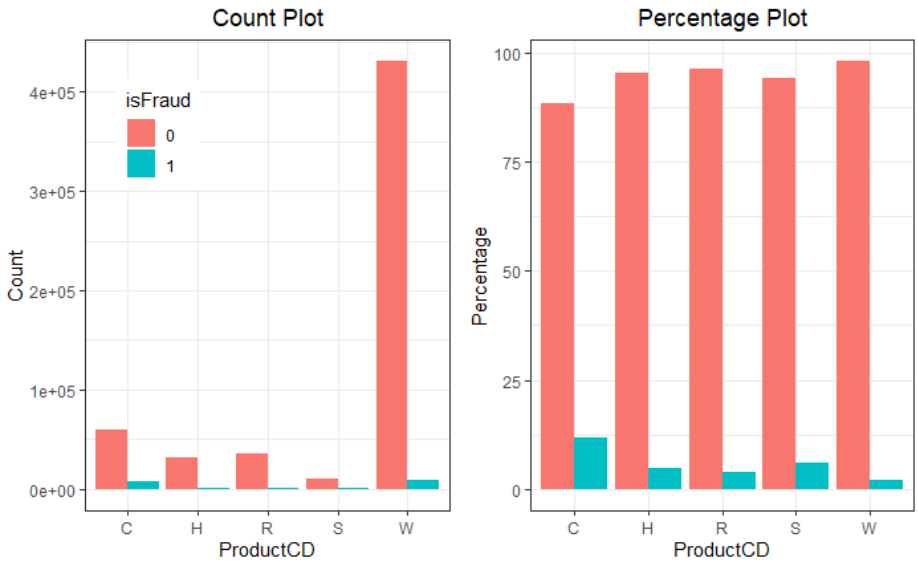
- **Transaction Amount**

We checked the distribution of the column. We observed that the values in the column are very highly skewed. So, we did a log transformation to make the distribution normal. The column had no missing values.



- **ProductCD**

This is a categorical column. For this column we checked the count of each type for determining the imputing strategy for the missing values of the column. '**W**' was the most common category in the column and category '**C**' had the most percentage of fraud amongst the categories. The column had no missing values.

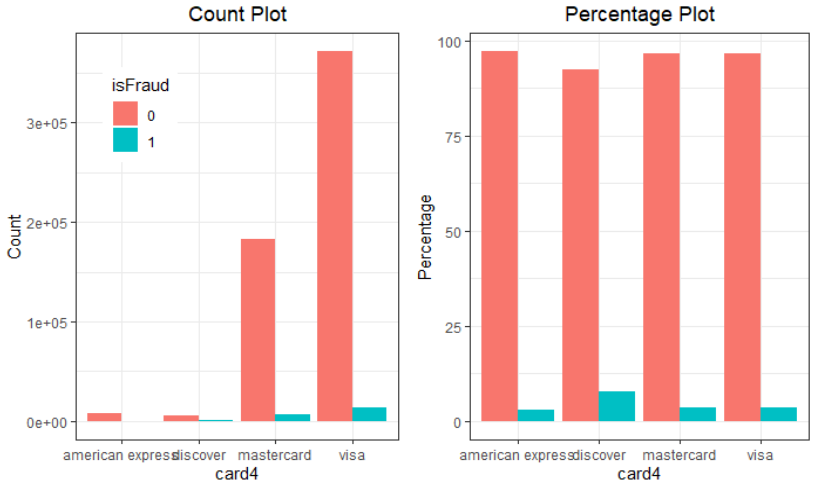


- **Card columns**

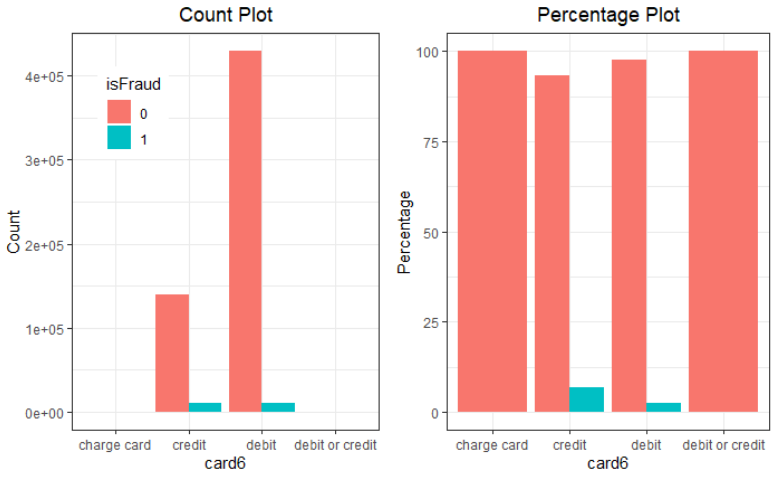
There are 6 columns starting with the name card. Card_4 and card_6 are categorical columns and the remaining columns are numerical. The missing value ratio is given in the image below.

card1 <dbl>	card2 <dbl>	card3 <dbl>	card4 <dbl>	card5 <dbl>	card6 <dbl>
0	0.01512683	0.002650117	0.002670437	0.007212043	0.002660277

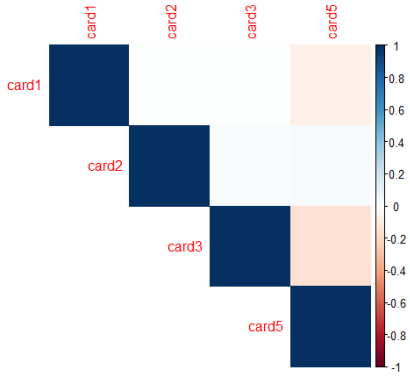
Count and percentage plots for the card_4 and card_6 is given below.



We can see that the visa card is the most famous category in this column. Discover, though ti has the least market share, leads to the greatest number of frauds.



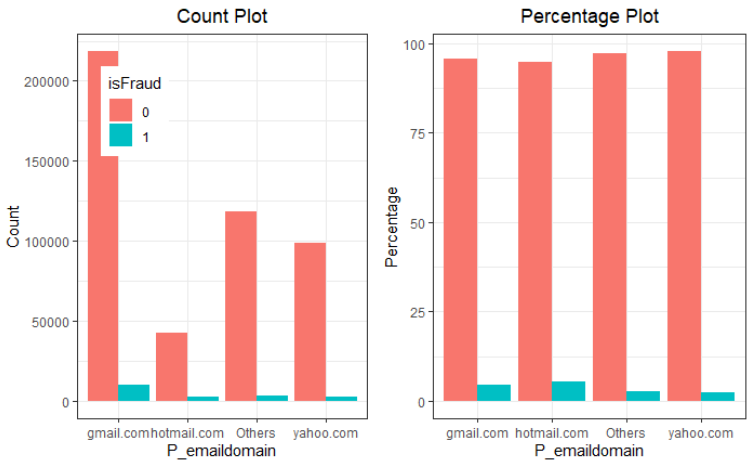
Debit card is the most common category in the column card_6, but credit card leads to the most frauds.



The correlation plot for the numerical columns can be seen above. We concluded that there is no highly correlated columns in the card columns.

- **Email Domain (P_emaildomain)**

This column contains the email addresses of the person doing transaction. There are 59 unique values in the column. We cleaned the values in the column to reduce the number of categories.



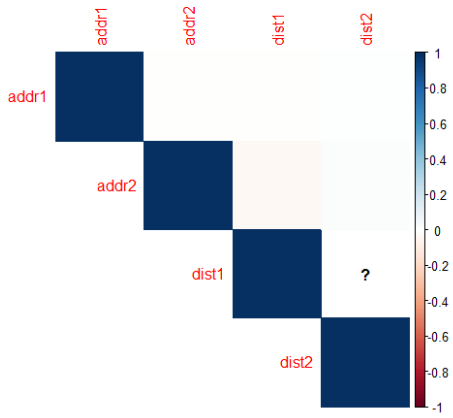
16% of the values in the column were null. **R_emaildomain** column has 76% null values. Since it is a categorical variable, and most of the values are null, we dropped this column from consideration for building a model with.

- **Address and Distance column**

These columns contain information about the address from which the transaction was done. The distance columns contain information about the distance. The missing values ratios for the columns are as defined below.

addr1 <dbl>	addr2 <dbl>	dist1 <dbl>	dist2 <dbl>
0.1112643	0.1112643	0.5965235	0.9362837

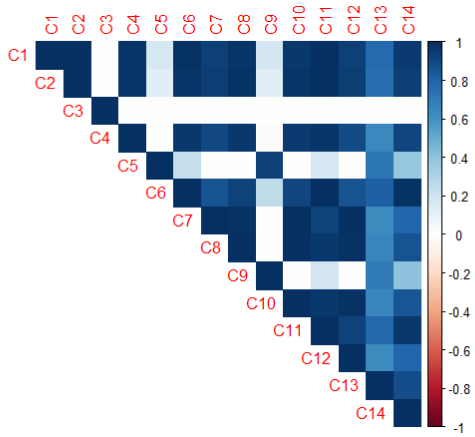
Since the dist2 column has 94% missing values, we dropped that column from consideration for using towards building the model.



The columns are not highly correlated.

- **C – Columns**

There are 14 columns starting from the name C. There are no missing values in these columns. We checked for correlation amongst these columns and obtained the below plot.



We can see that the columns are highly correlated.

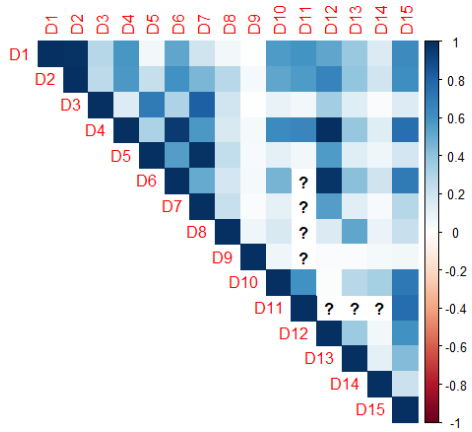
- **D – Columns**

These are the columns starting with the name D. The columns are numerical columns. The ratio of missing values is given below.

D1 <dbl>	D2 <dbl>	D3 <dbl>	D4 <dbl>	D5 <dbl>	D6 <dbl>	D7 <dbl>	D8 <dbl>
0.002148881	0.4754919	0.4451485	0.2860467	0.524674	0.8760677	0.9340993	0.8731229

D8 <dbl>	D9 <dbl>	D10 <dbl>	D11 <dbl>	D12 <dbl>	D13 <dbl>	D14 <dbl>	D15 <dbl>
0.8731229	0.8731229	0.128733	0.4729349	0.8904105	0.8950926	0.8946947	0.1509009

We checked for correlation amongst the columns.



There are columns which are highly correlated and will need to be removed while building a model.

- **V – Columns**

There are 332 columns starting with ‘V’. Since it is not feasible to obtain a correlation plot for these many columns, we decided to go for PCA. The values in the columns are sparsely populated hence we replaced the missing values with 0. The columns also follow missing values pattern where columns have same ratio of missing values. On taking the correlation of those columns, we found that the columns have high correlation amongst them.

- **Id – Columns**

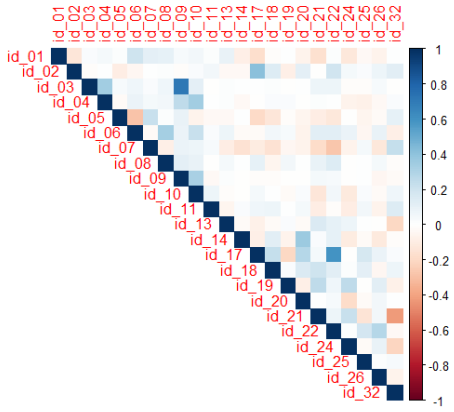
There are 23 columns in the identity dataset starting from the name Id. The ratio of missing values in these columns is given below.

id_01 <dbl>	id_02 <dbl>	id_03 <dbl>	id_04 <dbl>	id_05 <dbl>	id_06 <dbl>	id_07 <dbl>	id_08 <dbl>
0	0.02330257	0.5401607	0.5401607	0.05108401	0.05108401	0.9642592	0.9642592

id_09 <dbl>	id_10 <dbl>	id_11 <dbl>	id_13 <dbl>	id_14 <dbl>	id_17 <dbl>	id_18 <dbl>	id_19 <dbl>
0.4805211	0.4805211	0.02256765	0.1172617	0.4450369	0.03372321	0.6872214	0.03407681

id_19 <dbl>	id_20 <dbl>	id_21 <dbl>	id_22 <dbl>	id_24 <dbl>	id_25 <dbl>	id_26 <dbl>	id_32 <dbl>
0.03407681	0.034472	0.9642315	0.9641622	0.967088	0.9644187	0.9642038	0.4620787

The correlation plot for these columns was obtained as below.



We obtained that none of the columns in the id columns are highly correlated.

III. Data Preparation and Pre-processing

We carried out various variable conversions. The important ones were converting isFraud, the target variable to a categorical variable. We also created new features like Hour and Weekday from the TransactionDT column to represent the hour and the day of the week the transaction was carried out. In order to employ feature reduction, we used correlation plots as well as PCA. We used correlation plots for reducing the number of features from C and D columns. The final features that we selected from them were C1, C3, C5, C13, D1, D3, D4, D5, D8, D9, D10, D11, D13, D14, D15. We used PCA to reduce the number of features from the V-columns. On observing the variability explained by the PCA output, we decided on using the top 25 features for using towards the final machine learning model. The strategy we employed here is we clubbed the columns which were named same, like columns starting with C, D, M, id, card and V and applied correlation and PCA to remove the columns which were highly correlated. We were able to reduce the 394-column transaction dataset down to 93 columns. Identity dataset had a column DeviceInfo which had information about the devices which were used for transaction. It had 1786 different values in it. Suitable data cleaning was used to reduce the number of categories down to 23 categories. Appropriate imputation methods were used to impute the missing values. We used left outer join to merge the two datasets because not all TransactionID were present in the identity dataset. After all the pre-processing we got a dataset with 590540 rows and 93 columns. Then we created dummies for categorical data which increased our columns to 203.

IV. Data Mining Techniques and Implementation

Objective of our project was to identify fraudulent transaction based on previous transaction history. We had a response variable named 'isFraud' which had information whether transaction was fraud or no fraud. It was binary classification supervised machine learning problem. We had imbalanced dataset with 96% non-fraudulent transactions and only 4% fraudulent transactions. In order to build a model, we used different sampling techniques such as under-sampling and over-sampling. We trained model by using oversampled training dataset and used test dataset having original proportion of fraudulent and non-fraudulent transactions for measuring accuracy and then followed same steps for under sampled training data. After comparing results, we found that we were getting better accuracy using under sampled training data. We built models using this under sampled data. We used various data mining techniques like Naïve Bayes classifier, Random Forest, CART, SVM, Logistic Regression, k-NN and LDA.

Proportion of fraudulent and non-fraudulent transactions in training and test data can be seen below.

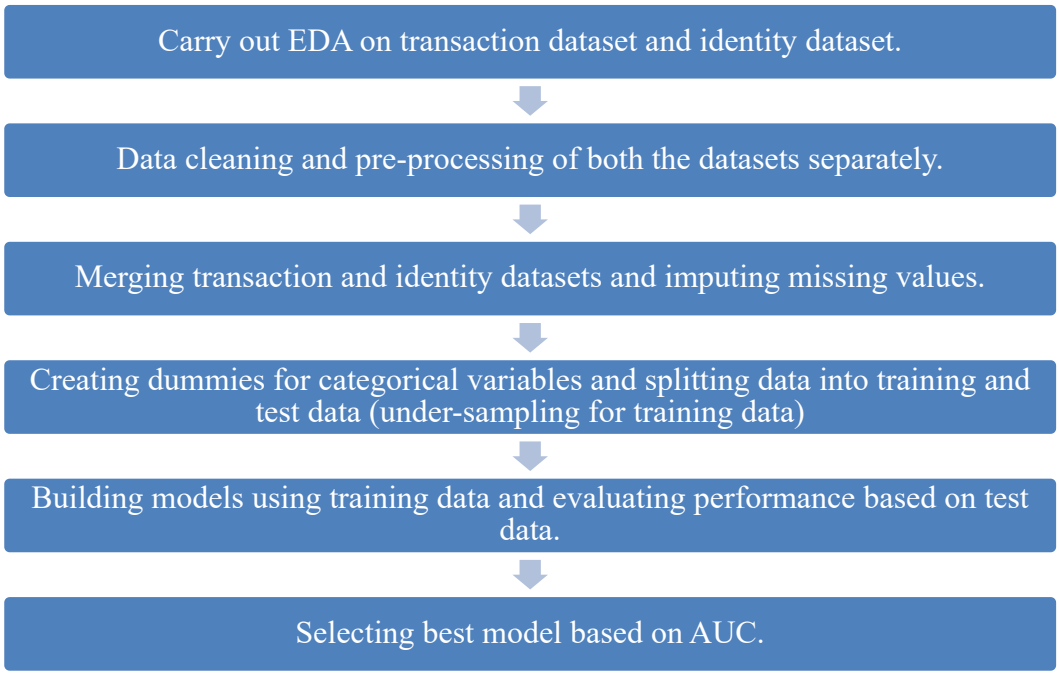
```
> prop.table(table(df_train_ub$isFraud))

        0         1
0.4980218 0.5019782
```

```
> prop.table(table(df_test$isFraud))

         0          1
0.96500339 0.03499661
```

Flow chart of our whole process:



V. Performance Evaluation

As we were having binary classification problem with one important class (fraudulent transactions) we used sensitivity (TPR), 1-specificity (FPR) and AUC for model evaluation. We used threshold of 0.5 for plotting confusion matrix and for calculating sensitivity and specificity we used sensitivity and specificity functions from caret package which determines appropriate threshold by its own. Following are the various models we built and their performances:

- **Naïve Bayes:**

Confusion Matrix:

		Predicted	
		0	1
Actual	0	7686	862
	1	200	110

Sensitivity (TPR): 0.8991577
1-Specificity (FPR): 0.6451613
AUC: 0.627

- **Random Forest:**

After varying number of trees from 100 to 700 we found that we were getting largest AUC at ntree=500. So we used 500 trees for building our model.

Confusion Matrix:

		Predicted	
		0	1
Actual	0	7302	1246
	1	65	245

Sensitivity (TPR): 0.8542349
1-Specificity (FPR): 0.2096774
AUC: 0.8223

- **CART:**

Confusion Matrix:

	Predicted		
Actual		0	1
	0	7677	871
	1	200	110

Sensitivity (TPR): 0.8981048
1-Specificity (FPR): 0.6451613
AUC: 0.6265

- **Logistic Regression:**

Confusion Matrix:

	Predicted		
Actual		0	1
	0	6842	1706
	1	104	206

Sensitivity (TPR): 0.8004212
1-Specificity (FPR): 0.3354839
AUC: 0.7325

- **SVM:**

  To improve performance of SVM improves when we use scaled data, we standardized data before building the model. After that we built model using standardized data and evaluated performance using linear, polynomial and radial kernels. If found that model performed better using radial kernel. Then we varied cost function and got best model with cost function 3.

Confusion Matrix:

	Predicted		
Actual		0	1
	0	7203	1345
	1	86	224

Sensitivity (TPR): 0.8426533
1-Specificity (FPR): 0.2774194
AUC: 0.782

- **LDA:**

Confusion Matrix:

	Predicted		
Actual		0	1
	0	6722	1826
	1	102	208

Sensitivity (TPR): 0.7863828
1-Specificity (FPR): 0.3290323
AUC: 0.7287

- **K-NN:**

We varied k from 1 to 15 and calculated AUC for each k on training dataset. We found largest AUC at k=5. So, we used k=5 for test data.

Confusion Matrix:

	Predicted		
Actual		0	1
	0	7601	947
	1	65	245

Sensitivity (TPR): 0.7903225806
1-Specificity (FPR): 0.3092138512
AUC: 0.7144

- **Neural Net:**

We tried implementing neural net but because of more than 590k records and 203 columns and limited memory of 8 GB, RStudio was shutting down while building the model. Furthermore we needed to compare models by varying various parameters like threshold, number of nodes, hidden layers and learning rate so we had to build model again and again by varying these parameters. To tackle this issue, we reduced number of records using sampling. We only took 5% of our original dataset but because of limited number of records available for training model, we were getting AUC less than 0.56 on test data which is almost as bad as random guessing.

Because of these limitations we were not able to implement neural net in our project.

VI. Discussion and Recommendation

We built a model by under-sampling the majority class. We sampled the data in such a way that the newly generated dataset has the same number of positive and negative class. This method is predominantly targeted at being able to find the fraudulent cases the greatest number of times. This leaves out the room for making errors in recognizing the non-fraudulent cases. Hence describing the lower accuracy for the model. There are two possible solutions for the problem here:

1. To increase data with more fraudulent cases as this will make sure we can incorporate more non-fraudulent cases in the sampled dataset.
2. To over-sample the minority class, i.e., have repeated entries of the fraudulent cases in the sampled dataset and increase the sample size with more non-fraudulent dataset.

Due to memory limitations on the system we performed this project on, we could not fully test the over-sampling method to the limit. The results were similar to the results we obtained were slightly better for sampling the minority class twice as compared to the under-sampling method.

VII. Summary

The project was aimed at creating a machine learning model for detecting the fraudulent transactions. We successfully carried out the exploratory analysis to identify the trends in the data and possible imputation strategy for the missing values of the dataset. We employed feature selection by observing the correlation plots for the columns and used PCA to reduce the number of features. We employed data cleaning to reduce the number of categories in the device information column in identity dataset. We also tested sampling techniques to find the suitable sampling technique to make a dataset to build the model upon. Under-sampling technique was chosen as the most competent strategy. Multiple models were built using different machine learning techniques. The performance metrics provided for the model were the best results we obtained for using the models after hyper-parameter tuning. If the task was to identify most of the fraudulent transactions, we would recommend using the random forest model. The model gives a sensitivity of 90%. That means, it could identify 90% of the fraudulent cases. Though we understand that the performance of the model is not perfect, we recommend increasing the sample size of the sampled dataset to get a lot more improved results. The memory computation limitations limited the progress of the project due to the high number of row counts.

VIII. Appendix R code:

```
---
title: "Project"
author: "Vinit Deshbhratar, Siddharth Muthe"
date: "16/02/2020"
output: html_document
---

Loading the libraries
```{r, message=FALSE}
library(ggplot2)
library(gridExtra)
library(dplyr)
library(corrplot)
library(factoextra)
library(stringr)
library(ROSE)
library(e1071)
library(dummies)
library(class)
library(caTools)
library(randomForest)
library(pROC)
library(MASS)
library(rpart)
library(caret)
```
```

```{r}
train_transaction = read.csv('train_transaction.csv', stringsAsFactors = FALSE, header = TRUE)
```

Taking a look at the train transaction dataset.

```{r}
head(train_transaction)
```

Taking a look at the isFraud column of the dataset. This column contains the information whether the transction is fraud or not.

```{r}
train_transaction$isFraud = as.factor(train_transaction$isFraud)
```

```{r}
ggplot(train_transaction) +
    geom_bar(aes(isFraud), fill='steelblue') +
    ggtitle('Imbalance in the dataset') +
    theme_bw() +
    theme(plot.title = element_text(hjust = 0.5))
```

Analysis of TransactionDT

```{r}
range(train_transaction$TransactionDT/(60*60*12))
```

```

```{r}
range(train_transaction$TransactionDT/(60*60*24))
```

There are 366 days in a leap year and our data has close to 6 months of data. We'll create two new columns in the dataset, which will have the data for day in the week, hour of the day.

```{r}
train_transaction['Hour'] =
as.factor(floor(train_transaction$TransactionDT/(3600*24))%%24)

train_transaction['WeekDay'] =
as.factor(floor(train_transaction$TransactionDT/(3600*24))%%7)
```

```{r}
hour_plot = ggplot(train_transaction) +

        geom_bar(aes(Hour, fill=isFraud), position = 'dodge')
```

```{r}
weekday_plot = ggplot(train_transaction) +

            geom_bar(aes(WeekDay, fill = isFraud), position='dodge')
```

```{r}
gridExtra::grid.arrange(hour_plot, weekday_plot)
```

Trasnsaction Amount
```

Now we'll check the distribution of the column TransactionAmt. The column holds the value for the transaction amount. If the distribution is not normal, we'll have to transform the data so that it has normal distribution.

```{r}
ggplot(train_transaction) +
    geom_histogram(aes(TransactionAmt)) +
    theme_bw() +
    ggtitle('Log Scaled Distribution of TransactionAmt') +
    theme(plot.title = element_text(hjust = 0.5))
```

Taking log

```{r, warning=FALSE}
ggplot(train_transaction) +
    geom_histogram(aes(log(TransactionAmt))) +
    theme_bw() +
    ggtitle('Log Scaled Distribution of TransactionAmt') +
    theme(plot.title = element_text(hjust = 0.5))
```

```{r}
train_transaction$TransactionAmt = log(train_transaction$TransactionAmt)
```

A common function to give a count and percentage plot

```{r}
```

```
percentage_plot = function(df, column){
    new_df = df %>%
        group_by(!!column, isFraud)%>%
        filter(!!column != '') %>%
        summarise(Count = n()) %>%
        mutate(Percentage = Count/sum(Count)*100)
    plot1 = ggplot(new_df) +
        geom_bar(aes(!!column, Count, fill=isFraud), position='dodge', stat = 'identity') +
        theme_bw() +
        theme(legend.position = c(0.2, 0.8)) +
        ggtitle('Count Plot') +
        theme(plot.title = element_text(hjust = 0.5))
    plot2 = ggplot(new_df) +
        geom_bar(aes(!!column, Percentage, fill=isFraud), position='dodge', stat =
'identity')+
        theme_bw() +
        theme(legend.position = 'none') +
        ggtitle('Percentage Plot') +
        theme(plot.title = element_text(hjust = 0.5))
    grid.arrange(plot1, plot2, ncol=2, nrow=1)
}
```

A common function to identify the highly correlated columns in the dataset

```{r}
get_correlated_columns = function(mat){
    cols = c()
    for(i in 1:length(colnames(mat))){
        for(j in 1:i){
```

```
            if(i != j){
                if(!is.na(mat[i, j]) & mat[i, j] > 0.9)
                    cols = c(cols, colnames(mat)[i])
            }
            else{
                break
            }
        }
    }
    return(unique(cols))
}
```

A common function to return the percentage of NA in the columns

```{r}
get_na_percentage = function(df){
    na_df = df %>% summarise_all(list(name = ~sum(is.na(.) | . == '')/length(.)))
    names(na_df) = names(df)
    return(na_df)
}
```

A common function to return the mean and standard deviation of the columns

```{r}
get_stats = function(df){
    mean_df = df %>% summarise_all(list(name = ~mean(., na.rm = TRUE)))
    std_df = df %>% summarise_all(list(name = ~sd(., na.rm = TRUE)))


```

```
    stats_df = rbind(mean_df, std_df)
    names(stats_df) = names(df)
    rownames(stats_df) = c('Mean', 'Std Dev')
    return(stats_df)
}
```

A common function to return the scaled data

```{r}
get_scaled_data = function(stats, df){
    new_df = data.frame(matrix(0, nrow = nrow(df), ncol = ncol(df)))
    names(new_df) = names(df)
    for(col in names(df)){
        new_df[, col] = (df[, col] - stats[1, col]) / stats[2, col]
    }
    return(new_df)
}
```

A common function to select the columns with numerical data types

```{r}
check_type_numerical <- function(col1){
    if(typeof(col1) != 'character'&typeof(col1) != 'logical')
        return (T)
    else
        return(F)
}
```

```
# A common function to replace NA values


```{r}
impute_na = function(df, fun){
    for(col in names(df)){
        if(fun == 'mean')
            df[is.na(df[, col]), col] = mean(df[, col], na.rm = TRUE)
        else
            if(fun == 'median')
                df[is.na(df[, col]), col] = median(df[, col], na.rm = TRUE)
            else
                if(fun == 'mode')
                    df[is.na(df[, col]) | df[, col] == '', col] = names(sort(table(df[, col]),
decreasing = T))[1]
                else
                    df[is.na(df[, col]), col] = 0
    }
    return(df)
}
```


Product


```{r}
get_na_percentage(data.frame(train_transaction$ProductCD))
```


```{r}
percentage_plot(train_transaction, quo(ProductCD))


```
```

```

## Card columns Analysis

```{r}
cols = names(train_transaction)
card_columns = cols[startsWith(cols, 'card')]
```

```{r}
get_na_percentage(train_transaction[ , card_columns])
```

card_4 and card_6 columns are categorical columns whereas card1, card2, card3, card5 are numerical columns. We'll first analyze categorical columns and then perform correlation analysis on the numerical columns to identify if there are columns which are highlt correlated.

```{r}
percentage_plot(train_transaction, quo(card4))
```

```{r}
percentage_plot(train_transaction, quo(card6))
```

```{r}
card_columns_num = card_columns[!card_columns %in% c('card4', 'card6')]
```
```

```
corrMatrix = round(cor(train_transaction[, card_columns_num], use =
'pairwise.complete.obs'), 2)

corrplot(corrMatrix, method = 'color', type = 'upper')
```

We can see from the plot that the numerical columns are not highly correlated.

Imputing the missing values for numerical columns

```{r}
train_transaction[, card_columns_num] = impute_na(train_transaction[, card_columns_num],
'mean')
```

Imputing the missing values for categorical columns

```{r}
train_transaction[, c('card4', 'card6')] = impute_na(train_transaction[, c('card4', 'card6')],
'mode')
```

Email

1. P_emaildomain

```{r}
train_transaction[!(train_transaction$P_emaildomain %in% c('gmail.com', 'yahoo.com',
'hotmail.com', '')), 'P_emaildomain'] = 'Others'
```

```{r}
get_na_percentage(data.frame(train_transaction$P_emaildomain))
```

```{r}

percentage_plot(train_transaction, quo(P_emaildomain))

```

Imputing the NA values

```{r}

train_transaction[, 'P_emaildomain'] = impute_na(data.frame(train_transaction[, 'P_emaildomain']), 'mode')

```

2. R_emaildomain

```{r}

get_na_percentage(data.frame(train_transaction$R_emaildomain))

```

Since the percentage of NA is very high in this categorical column and adding a new field will add a new category, we'll drop this column.

Address and Distance column Analysis

```{r}

addr_dist_columns = c('addr1', 'addr2', 'dist1', 'dist2')

```

```{r}

get_na_percentage(train_transaction[, addr_dist_columns])

```

```{r}
```

```
corrMatrix = cor(train_transaction[, addr_dist_columns], use = 'pairwise.complete.obs')

corrplot(corrMatrix, method = 'color', type = 'upper')
```

Since the dist2 column has 93.6% NA values and it signifies the distance, we cannot ambiguosly impute the NA values. So we'll drop the column and impute the remaining columns using the mean.

```{r}
addr_dist_columns = addr_dist_columns[addr_dist_columns != 'dist2']

train_transaction[, addr_dist_columns] = impute_na(train_transaction[, addr_dist_columns], 'mean')
```

C-Columns

```{r}
C_columns = cols[startsWith(cols, 'C')]

corrMatrix = round(cor(train_transaction[, C_columns]), 2)

corrplot(corrMatrix, method = 'color', type = 'upper')
```

This shows that there are columns which are highy correlated. We'll write a function which will return us the columns such that highly correlated columns are removed.

```{r}
get_na_percentage(train_transaction[, C_columns])
```

```{r}
corr_columns = get_correlated_columns(corrMatrix)
corr_columns
```

Here we have got the highly correlated columns. The next task would be to remove the highly correlated columns form the dataset and select the columns which are missing from the list.

```{r}
selected_columns = C_columns[!C_columns %in% corr_columns]
selected_columns
```

D columns

```{r}
D_columns = cols[startsWith(cols, 'D')]

corrMatrix = cor(train_transaction[, D_columns], use = 'pairwise.complete.obs')

corrplot(corrMatrix, method = 'color', type = 'upper')
```

```{r}
corr_columns = get_correlated_columns(corrMatrix)
corr_columns
```

Imputing the missing values

```{r}
get_na_percentage(train_transaction[, D_columns])
```

```{r}
train_transaction[, D_columns] = impute_na(train_transaction[, D_columns], 'mean')
```

```{r}
selected_columns = c(selected_columns, D_columns[!D_columns %in% corr_columns])
selected_columns
```

V-Columns Analysis

Here we'll take a look at the NA percentage and the mean and standard deviation of the columns so that we can devise a imputing strategy for the columns.

```{r}
V_columns = cols[startsWith(cols, 'V')]
get_stats(train_transaction[, V_columns])
```

```{r}
get_na_percentage(train_transaction[, V_columns])
```

As we can see that the NA percentage is same in a lot of columns. We'll use the correaltion function that we created early to see if there are any columns which are highly correalted.

```{r}
x = cor(train_transaction[, V_columns[138:166]], use = 'pairwise.complete.obs')
get_correlated_columns(x)
```

```{r}
x = cor(train_transaction[, V_columns[12:34]], use = 'pairwise.complete.obs')
get_correlated_columns(x)
```

So we can see that there are columns which are highly correlated and have same NA percentage amongst them. But the descriptive statistics for them vary a lot. Since it is impossible to know the exact information that can be stored in these columns, we will treat them as the transaction had 0 for the values which had NA in them.

Imputing the missing values in V-Columns

```{r}
train_transaction[, V_columns] = impute_na(train_transaction[, V_columns], '0')
```

PCA for V_Columns

There are 339 columns starting with 'V'. The values in them are very distributed. The maximum is very distant from the minimum value. The method that we use in this scenario is that we'll replace the NA values with 0 and then try to do the principal component analysis. Then using the scree plot we'll try to find the number of principal components that explain most of the variability of the columns. In order to do the PCA, we'll use the factextra

```{r, cache=TRUE}
df = train_transaction[, V_columns]
```

```

```{r, cache=TRUE}
pca_output = prcomp(df, scale=TRUE)
```

```{r}
summary(pca_output)
```

As we can see here most of the variability is explained by the first 25 principal component. So we'll use the first 50 components to build the model.

```{r}
df = data.frame(pca_output$x[, 1:25])
```

```{r}
rm(pca_output)
```

Selecting the required columns from the transaction dataset

```{r}
selected_columns = c(selected_columns, 'TransactionID', 'Hour', 'WeekDay', 'TransactionAmt', 'ProductCD', 'P_emaildomain', 'addr1', 'addr2', 'dist1', card_columns, 'isFraud')
```

```{r}
train_transaction = train_transaction[, selected_columns]
train_transaction = cbind(train_transaction, df)
```
```

```

# Identity table exploratory analysis

```{r}
train_identity <- read.csv('train_identity.csv', stringsAsFactors = FALSE, header = TRUE)
```

```{r}
num_cols <- unlist(lapply(train_identity, check_type_numerical))
num_cols = names(num_cols[num_cols])
numerical_ids = num_cols[startsWith(num_cols, 'i')]
```

Checking the percentage of NA values in the numerical columns

```{r}
get_na_percentage(train_identity[, numerical_ids])
```

Checking for correlation between the columns

```{r}
corrMatrix = round(cor(train_identity[, numerical_ids], use = 'pairwise.complete.obs'), 2)
corrplot(corrMatrix, method = 'color', type = 'upper')
```

```{r}
corr_columns = get_correlated_columns(corrMatrix)
corr_columns
```
```

```

Hence there are no numerical columns which have a very high correlation amongst them.

Imputing values in place of NULL in columns containing numerical data:

```{r}
id_cols_impute_mean <- c('id_24', 'id_25','id_07', 'id_08', 'id_21', 'id_26', 'id_22', 'id_18', 'id_03', 'id_32', 'id_14', 'id_13', 'id_20', 'id_19', 'id_17', 'id_02', 'id_11')

id_cols_impute_median <- c('id_04', 'id_09', 'id_10', 'id_05', 'id_06')

train_identity[, id_cols_impute_mean] = impute_na(data.frame(train_identity[, id_cols_impute_mean]), 'mean')

train_identity[, id_cols_impute_median] = impute_na(data.frame(train_identity[, id_cols_impute_median]), 'median')

get_na_percentage(train_identity)
```

Categorical columns

```{r}
cat_cols <- unlist(lapply(train_identity, check_type_numerical))

cat_cols = names(cat_cols[!cat_cols])
```

Taking a look at the NA values in the categorical columns

```{r}
get_na_percentage(train_identity[, cat_cols])
```
```

96% of data in columns 'id_23' and 'id_27' is missing so we drop those columns. Also, id_33 contains information on dimension of device so we drop it.

Imputing 0 in place of NULL in columns containing logical data:

```{r}
train_identity$id_35[is.na(train_identity$id_35)|train_identity$id_35==''] <- FALSE
train_identity$id_35 <- as.logical(train_identity$id_35)
train_identity$id_36[is.na(train_identity$id_36)|train_identity$id_36==''] <- FALSE
train_identity$id_36 <- as.logical(train_identity$id_36)
train_identity$id_37[is.na(train_identity$id_37)|train_identity$id_37==''] <- FALSE
train_identity$id_37 <- as.logical(train_identity$id_37)
train_identity$id_38[is.na(train_identity$id_38)|train_identity$id_38==''] <- FALSE
train_identity$id_38 <- as.logical(train_identity$id_38)
```

Reducing number of categories in Device Info column:

```{r}
train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo, ('(SM)|(SAMSUNG)|(GT)|(SGH)|(Grand)'))] <- 'SAMSUNG'
train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo, ('(LG)|(K10)|(^RS)|(VS)|(VK)'))] <- 'LG'
train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo, ('(HUAWEI)|(HONOR)|(DUK)'))] <- 'HUAWEI'
train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo, ('(lenovo)|(idea)|(yoga)'))] <- 'LENOVO'
train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo, ('(HTC)|(HT)'))] <- 'HTC'
train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo, ('(MI)|(XI)|(REDMI)'))] <- 'XIOMI'
train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo, ('(moto)|(xt)|(edison)'))] <- 'MOTO'
train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo, ('(zte)|(^z)|(blade)|(K88)'))] <- 'ZTE'
```

```
train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo,  ('(windows)|(rv)|(win)'))] <- 'WINDOWS NT'

train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo, ('(^F)|(E2)|(E5)|(E66)|(^(H3))|(^LT)|(^D)'))] <- 'SONY'

train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo,  ('(^80)|(ONE TOUCH)|(^50)|(^60)|(^90)|(^40)|(alcatel)'))] <- 'ALCATEL'

train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo,  ('(^P00)|(ASUS)|(ME)'))] <- 'ASUS'

train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo, ('(PIXEL)|(NEXUS)'))] <- 'PIXEL'

train_identity$DeviceInfo[(train_identity$DeviceInfo == 'TA')] <- 'NOKIA'

train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo, ('^(KF)'))] <- 'AMAZON'

train_identity$DeviceInfo[(train_identity$DeviceInfo == 'ilium')] <- 'ILIUM'

train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo, ('(E68)|(KY)'))] <- 'KYOCERA'

train_identity$DeviceInfo[(train_identity$DeviceInfo == 'Hisense')] <- 'HISENSE'

train_identity$DeviceInfo[(train_identity$DeviceInfo == 'verykool')] <- 'VERYKOOL'

train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo, ('(lava)|(iris)|(A97)'))] <- 'LAVA'

train_identity$DeviceInfo[str_detect(train_identity$DeviceInfo, ('(X78)|(Max)'))] <- 'MICROMAX'

train_identity$DeviceInfo[(train_identity$DeviceInfo == 'aquaris')] <- 'AQUARIS'

companies <- c('SAMSUNG','LG','HUAWEI','LENOVO','HTC','XIOMI', 'ZTE','WINDOWS NT', 'SONY','ALCATEL', 'ASUS','PIXEL','NOKIA','AMAZON', 'ILIUM','KYOCERA', 'HISENSE','VERYKOOL', 'LAVA', 'MICROMAX','AQUARIS','')

train_identity$DeviceInfo[!(train_identity$DeviceInfo %in% companies)] <- 'OTHERS'
```

Processed train identity dataset:

```{r}
keep_cols <- !(colnames(train_identity) %in% c('id_23','id_27', 'id_33'))

train_identity <- train_identity[, keep_cols]
```

Joining the two datasets

```{r}
merged_df<-merge(x=train_transaction, y=train_identity, by="TransactionID", all.x=TRUE)
#str(merged_df)
```

Get percentage of NA in each column:

```{r}
get_na_percentage(merged_df)
```

Impute NA:

```{r}
id_cols_impute_mean <- c('id_24', 'id_25','id_07', 'id_08', 'id_21', 'id_26', 'id_22', 'id_18', 'id_03', 'id_32', 'id_14', 'id_13', 'id_20', 'id_19', 'id_17', 'id_02', 'id_11', 'id_01')

id_cols_impute_median <- c('id_04', 'id_09', 'id_10', 'id_05', 'id_06')

id_cols_impute_mode <- c('id_35', 'id_36', 'id_37','id_38', 'id_12','id_15','id_16', 'id_28', 'id_29','id_30', 'id_31', 'id_34', 'DeviceType', 'DeviceInfo')

merged_df[, id_cols_impute_mean] = impute_na(data.frame(merged_df[, id_cols_impute_mean]), 'mean')

merged_df[, id_cols_impute_median] = impute_na(data.frame(merged_df[, id_cols_impute_median]), 'median')

merged_df[, id_cols_impute_mode] = impute_na(data.frame(merged_df[, id_cols_impute_mode]), 'mode')

rem <- c('id_30', 'id_34')

rem_col <- which(colnames(merged_df) %in% rem)

merged_df <- merged_df[, -rem_col]
```

```
get_na_percentage(merged_df)

#str(merged_df)

```


Get dummies:

```{r}

need_dummies <- c('Hour', 'WeekDay', 'ProductCD', 'P_emaildomain', 'card4', 'card6', 'id_15', 'DeviceInfo', 'id_31')


merged_df[, need_dummies] <- as.data.frame(apply(merged_df[, need_dummies], 2, function(x) gsub('\\s+', ',x)))


merged_df$id_31 <- str_extract(merged_df$id_31, '([a-z]+)')

merged_df$id_31[is.na(merged_df$id_31)] = 'Other'


merged_df[sapply(merged_df, is.character)] <- lapply(merged_df[sapply(merged_df, is.character)], as.factor)


merged_df<-cbind(merged_df, dummy(merged_df$Hour, sep = "_Hour_"))

merged_df<-cbind(merged_df, dummy(merged_df$WeekDay, sep = "_WeekDay_"))

merged_df<-cbind(merged_df, dummy(merged_df$ProductCD, sep = "_ProductCD_"))

merged_df<-cbind(merged_df, dummy(merged_df$P_emaildomain, sep = "_P_emaildomain_"))

merged_df<-cbind(merged_df, dummy(merged_df$card4, sep = "_card4_"))

merged_df<-cbind(merged_df, dummy(merged_df$id_15, sep = "_id_15_"))

merged_df<-cbind(merged_df, dummy(merged_df$card6, sep = "_card6_"))

merged_df<-cbind(merged_df, dummy(merged_df$id_31, sep = "_id_31_"))

merged_df<-cbind(merged_df, dummy(merged_df$DeviceInfo, sep = "_DeviceInfo_"))


levels(merged_df$id_12) <- c(0,1)

levels(merged_df$id_16) <- c(0,1)
```

```
levels(merged_df$id_28) <- c(0,1)
levels(merged_df$id_29) <- c(0,1)
levels(merged_df$id_35) <- c(0,1)
levels(merged_df$id_36) <- c(0,1)
levels(merged_df$id_37) <- c(0,1)
levels(merged_df$id_38) <- c(0,1)
levels(merged_df$DeviceType) <- c(0,1)

remove_cols <- c(which(colnames(merged_df) %in% need_dummies))
merged_df <- merged_df[, -remove_cols]
```

DATA AFTER DUMMIES:

```{r}
#temp <- merged_df
```

```{r}
merged_df <- temp
dim(temp)
```

Sampling Data Set:

```{r}
set.seed(123)
splitted_set <- sample.split(temp$isFraud, SplitRatio = 0.025)
merged_df <- subset(temp, subset = splitted_set)
#df_test <- subset(merged_df, subset = !splitted_set)
dim(merged_df)
prop.table(table(merged_df$isFraud))
```

```

# Train Test Split:

We split data into 70% training and 30% test.

We will use undersampling technique in order to have equal number of fradulent and non-fraudulent cases in training dataset while keeping the proportion equal in test data.

```{r}

set.seed(123)

splitted_set <- sample.split(merged_df$isFraud, SplitRatio = 0.70)

df_train <- subset(merged_df, subset = splitted_set)

df_test <- subset(merged_df, subset = !splitted_set)

dim(df_train)

#df_train_ub <- df_train

df_train_ub <- ovun.sample(isFraud~., data = df_train, method = "under", seed = 1)$data

dim(df_train_ub)

dim(df_test)

set.seed(123)

splitted_set <- sample.split(df_test$isFraud, SplitRatio = 0.05)

df_test <- subset(df_test, subset = splitted_set)

isFraud_col_num <- which(colnames(df_train_ub)=='isFraud')
```

Proportion of fradulent and non-fraudulent cases in training dataset.

```{r}

prop.table(table(df_train_ub$isFraud))

dim(df_train_ub)
```

Proportion of fradulent and non-fraudulent cases in test dataset.

```{r}

```
prop.table(table(df_test$isFraud))
dim(df_test)
```

Comparing Models:

Naive Bayes Model:

Fitting model and plotting Confusion Matrix:

```{r}
n_bayes <- naiveBayes(isFraud~., data = df_train_ub)
predict_nb <- as.data.frame(predict(n_bayes, df_test[,-isFraud_col_num], 'raw'))
predict_nb$predict_class <- ifelse(predict_nb$`1`> predict_nb$`0`, 1, 0)
confusion_matrix_nb <- table(df_test$isFraud, predict_nb$predict_class)
confusion_matrix_nb

#mean(predict_nb$predict_class == df_test_nb$isFraud)*100
#Accuracy = 86.20456
```

AUC:

```{r}
roc(response = df_test$isFraud, predictor = as.numeric(predict_nb$predict_class))
```

Area under the curve = 0.6443

Sensitivity and specificity:

```{r}
sensitivity(factor(predict_nb$predict_class), df_test$isFraud)
1-specificity(factor(predict_nb$predict_class), df_test$isFraud)
```

```

# Random Forest:

Fitting Model and plotting Confusion Matrix.

```{r}
rf_model <- randomForest(x=df_train_ub[,-isFraud_col_num], y=df_train_ub$isFraud, ntree=500)

prediction_rf <- as.data.frame(predict(rf_model, df_test[,-isFraud_col_num], type='prob'))

rf_predicted_class <- ifelse(prediction_rf$`1` > 0.5, 1, 0)

confusion_matrix_rf <- table(df_test$isFraud, rf_predicted_class)

confusion_matrix_rf


#mean(rf_predicted_class == df_test$isFraud)*100

#Accuracy = 85.19982
```


AUC:

```{r}
roc(response = df_test$isFraud, predictor = as.numeric(rf_predicted_class))
```

Area under the curve = 0.8195

Sensitivity and specificity:

```{r}
sensitivity(factor(rf_predicted_class), df_test$isFraud)

1-specificity(factor(rf_predicted_class), df_test$isFraud)
```
```

CART:

Fitting Model and plotting Confusion Matrix.

```{r}
cart_model <- rpart(isFraud~., data=df_train_ub, method='class')
predict_cart <- as.data.frame(predict(cart_model, df_test[,-isFraud_col_num]))
predict_cart$predict_class <- ifelse(predict_cart$`1`> predict_nb$`0`, 1, 0)

confusion_matrix_cart <- table(df_test$isFraud, predict_cart$predict_class)
confusion_matrix_cart

#mean(predict_cart$predict_class == df_test$isFraud)*100
#Accuracy = 87.90923
```

AUC:

```{r}
roc(response = df_test$isFraud, predictor = as.numeric(predict_cart$predict_class))
```

Area under the curve = 0.7187

Sensitivity and specificity:

```{r}
sensitivity(factor(predict_cart$predict_class), df_test$isFraud)
1-specificity(factor(predict_cart$predict_class), df_test$isFraud)
```

Logistic regression:

```{r}
```

```
lr_model <- glm(isFraud~., family=binomial(link="logit"), data=df_train_ub)

predict_lr <- as.data.frame(predict(lr_model, newdata=df_test[,-isFraud_col_num],
type="response"))

predict_lr$predict_class <- ifelse(predict_lr$`predict(lr_model, newdata = df_test[, -
isFraud_col_num], type = "response")` > 0.5, 1, 0)


confusion_matrix_lr <- table(df_test$isFraud, predict_lr$predict_class)

confusion_matrix_lr


#mean(lr_predicted_class == df_test$isFraud)*100

#Accuracy = 79.56649
```

AUC:

```{r}
roc(response = df_test$isFraud, predictor = as.numeric(predict_lr$predict_class))
```

Area under the curve = 0.7325

Sensitivity and specificity:

```{r}
sensitivity(factor(predict_lr$predict_class), df_test$isFraud)

1-specificity(factor(predict_lr$predict_class), df_test$isFraud)
```

Support Vector Machines:

Normalize train and test data:

```{r}
nums <- unlist(lapply(df_train_ub, is.numeric))
```

```
num_cols <- names(nums[nums])

df_train_normal <- df_train_ub
df_test_normal <- df_test

stats = get_stats(df_train_normal[, num_cols])
df_train_normal[, num_cols] = get_scaled_data(stats, df_train_normal[, num_cols])
df_test_normal[, num_cols] = get_scaled_data(stats, df_test_normal[, num_cols])

train_na <- colnames(df_train_normal)[colSums(is.na(df_train_normal)) > 0]
na_cols <- c(which(colnames(df_train_normal) %in% train_na))
df_train_normal <- df_train_normal[, -na_cols]
df_test_normal <- df_test_normal[, -na_cols]
```

Fitting Model and plotting Confusion Matrix:

```{r}
svm_model <- svm(isFraud~., data = df_train_normal, scale = F, kernel = "radial", cost = 3, type='C')
predict_svm <- as.data.frame(predict(svm_model, df_test_normal[,-isFraud_col_num]))
confusion_matrix_svm <- table(df_test$isFraud, predict_svm$`predict(svm_model, df_test_normal[, -isFraud_col_num])`)
confusion_matrix_svm

#mean(predict_svm$`predict(svm_model, df_test_normal[, -isFraud_col_num])` == df_test$isFraud)*100
#Accuracy = 83.84511
```

AUC:

```{r}
```

roc(response = df_test$isFraud, predictor = as.numeric(predict_svm$`predict(svm_model, df_test_normal[, -isFraud_col_num])`))

```

Area under the curve = 0.7826

Sensitivity and specificity:

```{r}

sensitivity(factor(predict_svm$`predict(svm_model, df_test_normal[, -isFraud_col_num])`), df_test$isFraud)

1-specificity(factor(predict_svm$`predict(svm_model, df_test_normal[, -isFraud_col_num])`), df_test$isFraud)

```

Linear Discriminant Analysis:

Fitting Model and plotting Confusion Matrix:

```{r}

lda_model <- lda(isFraud~., data = df_train_normal)

predict_lda <- as.data.frame(predict(lda_model, df_test_normal[,-isFraud_col_num]))

confusion_matrix_lda <- table(df_test_normal$isFraud, predict_lda$class)

confusion_matrix_lda

#mean(predict_lda$class==df_test_normal$isFraud)*100

#Accuracy = 78.23436

```

AUC:

```{r}

roc(response = df_test_normal$isFraud, predictor = as.numeric(predict_lda$class))

```

Area under the curve = 0.7287
```

Sensitivity and specificity:

```{r}
sensitivity(factor(predict_lda$class), df_test$isFraud)

1-specificity(factor(predict_lda$class), df_test$isFraud)
```

k-nn model:

Selecting k using error in training data and then using it for test data:

```{r}
ks = 1:10

accuracy_train = c()

for(k in ks){

   model_knn <- knn(df_train_normal[, -isFraud_col_num], df_train_normal[, -isFraud_col_num], k = k, cl = df_train_normal$isFraud)

   accuracy_train <- c(roc(response = df_test_normal$isFraud, predictor = as.numeric(predict_knn)))

}

which(accuracy_train %in% max(accuracy_train))


predict_knn <- knn(df_train_normal[, -isFraud_col_num], df_test_normal[, -isFraud_col_num], k =5, cl = df_train_normal$isFraud, prob=T)


knn_predicted_prob <- as.data.frame(attr(predict_knn,"prob"))

knn_predicted_prob$class <- ifelse(knn_predicted_prob$`attr(predict_knn, "prob")`>0.5, 0, 1)


confusion_matrix_knn <- table(df_test_normal$isFraud, knn_predicted_prob$class)

confusion_matrix_knn
```

```
#mean(predict_knn == df_test_normal$isFraud)
#Accuracy = 0.7992775
```

AUC:

```{r}
roc(response = df_test_normal$isFraud, predictor = as.numeric(predict_knn))
```

Area under the curve = 0.7435

Sensitivity and specificity:

```{r}
sensitivity(factor(knn_predicted_prob$class), df_test$isFraud)
1-specificity(factor(knn_predicted_prob$class), df_test$isFraud)
```