# Credit Card Fraud Detection

Rishabh Singh Dodeja

July 22, 2020

# Contents

# 1. Introduction

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

According to creditcards.com, there was over £300m in fraudulent credit card transactions in the UK in the first half of 2016, with banks preventing over £470m of fraud in the same period. The data shows that credit card fraud is rising, so there is an urgent need to continue to develop new, and improve current, fraud detection methods. Using this dataset, we will use machine learning to develop a model that attempts to predict whether or not a transaction is fraudlent. To preserve anonymity, these data have been transformed using principal components analysis.

To begin this analysis, we will first train a random forest model to establish a benchmark, we will also analyze and identify impotatn variables in predicting model. Then we will move one to developint a XG-Boost Classifier a more complex and robust approach.

## 1.1 Dataset

The datasets contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-senstive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

## 1.2 Process and Workflow

The main steps in this project include:

**1. Data Ingestion:** download, parse, import and prepare data for further processing and analysis

**2. Data Exploration:** explore data to understand, analyze and visualize different features and their relationships with movie ratings

**3. Data Cleaning:** deal with or eventually remove data with missing or incorrect values from dataset

**4. Modelling and Analysis:** create models with two different approach and compare their perfomance based of evaluation metric as well as computation time. Analayze and identify important features in predicting the classes

**5. Communicate:** create report and publish results

## 2. Data Ingestion

### 2.1 Loading Data & Packages

This section will automatically download required packages and dataset. The dataset can be found at Kaggle: The .csv file data is read as creditcard dataframe and this data is further used for data exploration and visualization

```
################################
# Create creditcard dataset
################################

# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-pro
ject.org")

if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org
")

if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-p
roject.org")

if(!require(corrplot)) install.packages("corrplot", repos = "http://cran.us.r-proje
ct.org")

#Credit Card Fraud Data
  # Kaggle: https://www.kaggle.com/mlg-ulb/creditcardfraud/

#dl <- tempfile()
#download.file("https://www.kaggle.com/mlg-ulb/creditcardfraud/download", dl)

#creditcard <- fread(text = gsub("::", "\t", readLines(unzip(dl, "creditcard.csv"))
))

#Reading downloaded data from local directory
creditcard <- fread(text = gsub("::", "\t", readLines("creditcard.csv")))
```

```r
# check the data and structure
head(creditcard)
```

```
##    Time         V1          V2        V3         V4          V5
## 1:    0 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077
## 2:    0  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765
## 3:    1 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813
## 4:    1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888
## 5:    2 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338
## 6:    2 -0.4259659  0.96052304 1.1411093 -0.1682521  0.42098688
##             V6          V7          V8         V9         V10        V11
## 1:  0.46238778  0.23959855  0.09869790  0.3637870  0.09079417 -0.5515995
## 2: -0.08236081 -0.07880298  0.08510165 -0.2554251 -0.16697441  1.6127267
## 3:  1.80049938  0.79146096  0.24767579 -1.5146543  0.20764287  0.6245015
## 4:  1.24720317  0.23760894  0.37743587 -1.3870241 -0.05495192 -0.2264873
## 5:  0.09592146  0.59294075 -0.27053268  0.8177393  0.75307443 -0.8228429
## 6: -0.02972755  0.47620095  0.26031433 -0.5686714 -0.37140720  1.3412620
##            V12        V13        V14        V15        V16         V17
## 1: -0.61780086 -0.9913898 -0.3111694  1.4681770 -0.4704005  0.20797124
## 2:  1.06523531  0.4890950 -0.1437723  0.6355581  0.4639170 -0.11480466
## 3:  0.06608369  0.7172927 -0.1659459  2.3458649 -2.8900832  1.10996938
## 4:  0.17822823  0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279
## 5:  0.53819555  1.3458516 -1.1196698  0.1751211 -0.4514492 -0.23703324
## 6:  0.35989384 -0.3580907 -0.1371337  0.5176168  0.4017259 -0.05813282
##            V18         V19         V20          V21          V22
## 1:  0.02579058  0.40399296  0.25141210 -0.018306778  0.277837576
## 2: -0.18336127 -0.14578304 -0.06908314 -0.225775248 -0.638671953
## 3: -0.12135931 -2.26185710  0.52497973  0.247998153  0.771679402
## 4:  1.96577500 -1.23262197 -0.20803778 -0.108300452  0.005273597
## 5: -0.03819479  0.80348692  0.40854236 -0.009430697  0.798278495
## 6:  0.06865315 -0.03319379  0.08496767 -0.208253515 -0.559824796
##            V23         V24        V25        V26          V27         V28
## 1: -0.11047391  0.06692807  0.1285394 -0.1891148  0.133558377 -0.02105305
## 2:  0.10128802 -0.33984648  0.1671704  0.1258945 -0.008983099  0.01472417
## 3:  0.90941226 -0.68928096 -0.3276418 -0.1390966 -0.055352794 -0.05975184
## 4: -0.19032052 -1.17557533  0.6473760 -0.2219288  0.062722849  0.06145763
## 5: -0.13745808  0.14126698 -0.2060096  0.5022922  0.219422230  0.21515315
## 6: -0.02639767 -0.37142658 -0.2327938  0.1059148  0.253844225  0.08108026
##    Amount Class
## 1: 149.62     0
## 2:   2.69     0
## 3: 378.66     0
## 4: 123.50     0
## 5:  69.99     0
## 6:   3.67     0
```

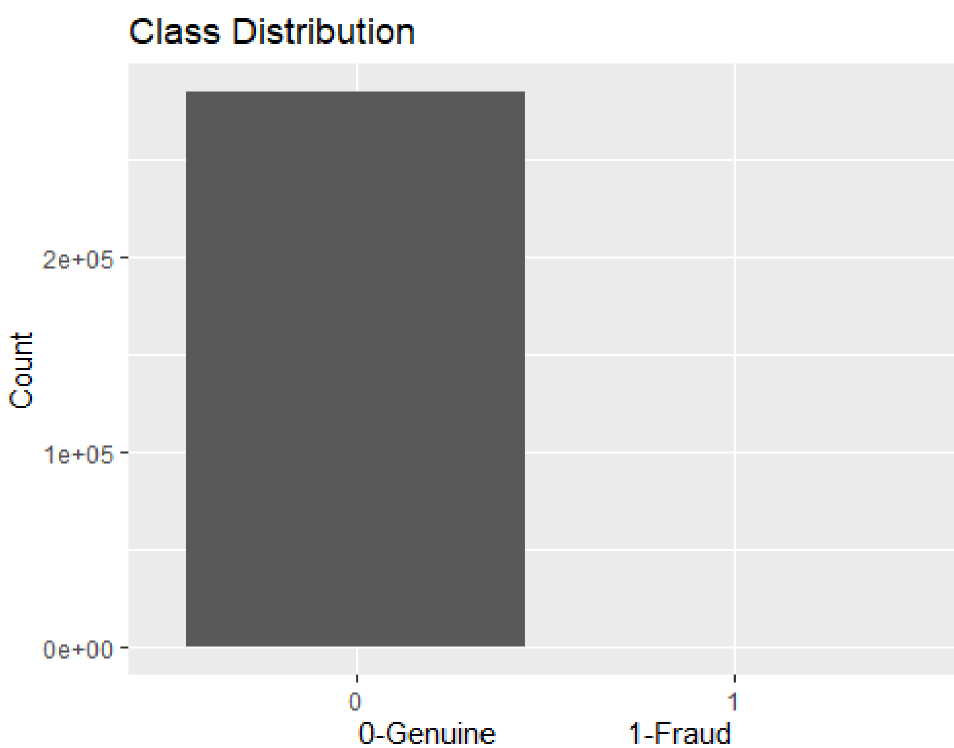
## 2.2 Data Exploration and Visualization

In this section we will explore the data and try to visualize as many aspects as possible, to get insights on relationships between differnet features and Classes. These insight are essential to develop a efficient prediction model

### 2.2.1 Classes

Here we try to visualize how is the distribution between the two classes, i.e., Fraud v/s Genuine defined by 1 and 0 numeircs respective

```
# summary table calculating counts for each class
ClassSummary = creditcard %>% group_by(Class) %>% summarise(Count=n()) %>% mutate(C
lass=as.character((Class)))

## `summarise()` ungrouping output (override with `.groups` argument)

# Bar Plot
ClassSummary %>% ggplot(aes(x = Class, y=Count)) + geom_col() + ggtitle("Class Dist
ribution") + labs(x="0-Genuine                1-Fraud")
```




Clearly, the dataset is extremely unbalanced. Even a "null" classifier which always predicts class=0 would obtain over 99% accuracy on this task. This demonstrates that a simple measure of mean accuracy should not be used due to insensitivity to false negatives.

To overcome this imbalance we can use some transformation techniques to make our dataset better for training. Some commonly used techniques for this kind of problems are listed below: 1.

Oversampling 2. Undersampling 3. SMOTE (Synthetic Minority Over-sampling Technique) In this project we use SMOTE discussed in later sections

The most appropriate measures to use on this task would be:

1. Precision
2. Recall
3. F-1 score (harmonic mean of precision and recall)
4. AUC (area under precision-recall curve)

### 2.2.2 Features

Here we try to find and visualize relationship between different features and classes. Below is summary of all features/columns statistics. We see that all the features V1 to V28 are normalized about zero. This s a great thing and helps building a better trained model. Thus we will also apply this normalization to "Amount" in a later section ahead.

This normalization is important to see how informative a feature actually is while predicting results/classes.

```
summary(creditcard)
```

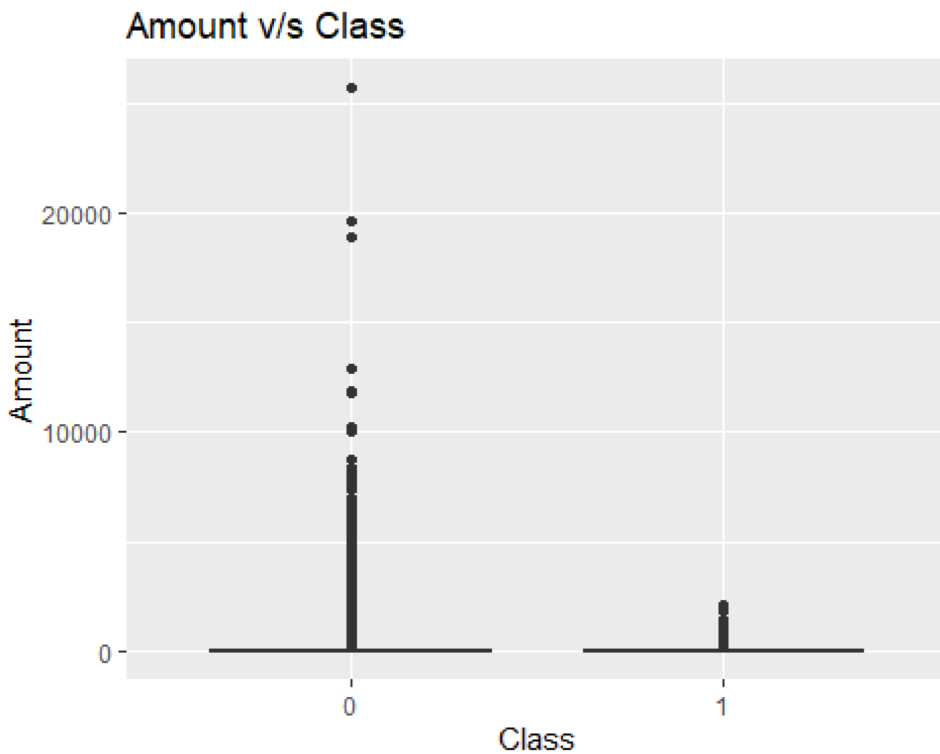
```
##       Time              V1                  V2
##  Min.   :     0   Min.   :-56.40751   Min.   :-72.71573
##  1st Qu.: 54202   1st Qu.: -0.92037   1st Qu.: -0.59855
##  Median : 84692   Median :  0.01811   Median :  0.06549
##  Mean   : 94814   Mean   :  0.00000   Mean   :  0.00000
##  3rd Qu.:139321   3rd Qu.:  1.31564   3rd Qu.:  0.80372
##  Max.   :172792   Max.   :  2.45493   Max.   : 22.05773
##        V3                V4                  V5
##  Min.   :-48.3256   Min.   :-5.68317   Min.   :-113.74331
##  1st Qu.: -0.8904   1st Qu.:-0.84864   1st Qu.:  -0.69160
##  Median :  0.1799   Median :-0.01985   Median :  -0.05434
##  Mean   :  0.0000   Mean   : 0.00000   Mean   :   0.00000
##  3rd Qu.:  1.0272   3rd Qu.: 0.74334   3rd Qu.:   0.61193
##  Max.   :  9.3826   Max.   :16.87534   Max.   :  34.80167
##        V6                V7                  V8
##  Min.   :-26.1605   Min.   :-43.5572   Min.   :-73.21672
##  1st Qu.: -0.7683   1st Qu.: -0.5541   1st Qu.: -0.20863
##  Median : -0.2742   Median :  0.0401   Median :  0.02236
##  Mean   :  0.0000   Mean   :  0.0000   Mean   :  0.00000
##  3rd Qu.:  0.3986   3rd Qu.:  0.5704   3rd Qu.:  0.32735
##  Max.   : 73.3016   Max.   :120.5895   Max.   : 20.00721
##        V9                 V10                  V11
##  Min.   :-13.43407   Min.   :-24.58826   Min.   :-4.79747
##  1st Qu.: -0.64310   1st Qu.: -0.53543   1st Qu.:-0.76249
##  Median : -0.05143   Median : -0.09292   Median :-0.03276
##  Mean   :  0.00000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.:  0.59714   3rd Qu.:  0.45392   3rd Qu.: 0.73959
##  Max.   : 15.59500   Max.   : 23.74514   Max.   :12.01891
```

```
##       V12                V13                V14         
##  Min.   :-18.6837   Min.   :-5.79188   Min.   :-19.2143  
##  1st Qu.: -0.4056   1st Qu.:-0.64854   1st Qu.: -0.4256  
##  Median :  0.1400   Median :-0.01357   Median :  0.0506  
##  Mean   :  0.0000   Mean   : 0.00000   Mean   :  0.0000  
##  3rd Qu.:  0.6182   3rd Qu.: 0.66251   3rd Qu.:  0.4931  
##  Max.   :  7.8484   Max.   : 7.12688   Max.   : 10.5268  
##       V15                V16                 V17          
##  Min.   :-4.49894   Min.   :-14.12985   Min.   :-25.16280  
##  1st Qu.:-0.58288   1st Qu.: -0.46804   1st Qu.: -0.48375  
##  Median : 0.04807   Median :  0.06641   Median : -0.06568  
##  Mean   : 0.00000   Mean   :  0.00000   Mean   :  0.00000  
##  3rd Qu.: 0.64882   3rd Qu.:  0.52330   3rd Qu.:  0.39968  
##  Max.   : 8.87774   Max.   : 17.31511   Max.   :  9.25353  
##       V18                 V19                 V20           
##  Min.   :-9.498746   Min.   :-7.213527   Min.   :-54.49772  
##  1st Qu.:-0.498850   1st Qu.:-0.456299   1st Qu.: -0.21172  
##  Median :-0.003636   Median : 0.003735   Median : -0.06248  
##  Mean   : 0.000000   Mean   : 0.000000   Mean   :  0.00000  
##  3rd Qu.: 0.500807   3rd Qu.: 0.458949   3rd Qu.:  0.13304  
##  Max.   : 5.041069   Max.   : 5.591971   Max.   : 39.42090  
##       V21                 V22                  V23           
##  Min.   :-34.83038   Min.   :-10.933144   Min.   :-44.80774  
##  1st Qu.: -0.22839   1st Qu.: -0.542350   1st Qu.: -0.16185  
##  Median : -0.02945   Median :  0.006782   Median : -0.01119  
##  Mean   :  0.00000   Mean   :  0.000000   Mean   :  0.00000  
##  3rd Qu.:  0.18638   3rd Qu.:  0.528554   3rd Qu.:  0.14764  
##  Max.   : 27.20284   Max.   : 10.503090   Max.   : 22.52841  
##       V24                V25                 V26          
##  Min.   :-2.83663   Min.   :-10.29540   Min.   :-2.60455  
##  1st Qu.:-0.35459   1st Qu.: -0.31715   1st Qu.:-0.32698  
##  Median : 0.04098   Median :  0.01659   Median :-0.05214  
##  Mean   : 0.00000   Mean   :  0.00000   Mean   : 0.00000  
##  3rd Qu.: 0.43953   3rd Qu.:  0.35072   3rd Qu.: 0.24095  
##  Max.   : 4.58455   Max.   :  7.51959   Max.   : 3.51735  
##       V27                  V28                Amount        
##  Min.   :-22.565679   Min.   :-15.43008   Min.   :    0.00  
##  1st Qu.: -0.070840   1st Qu.: -0.05296   1st Qu.:    5.60  
##  Median :  0.001342   Median :  0.01124   Median :   22.00  
##  Mean   :  0.000000   Mean   :  0.00000   Mean   :   88.35  
##  3rd Qu.:  0.091045   3rd Qu.:  0.07828   3rd Qu.:   77.17  
##  Max.   : 31.612198   Max.   : 33.84781   Max.   :25691.16  
##      Class         
##  Min.   :0.000000  
##  1st Qu.:0.000000  
##  Median :0.000000  
##  Mean   :0.001728  
##  3rd Qu.:0.000000  
##  Max.   :1.000000  
```

### *Amount v/s Classes*

To check relation ship betwwen amount and classes we will plot out a box blot

```
# Boxplot for Amount vs. Classes Distribution
ggplot(creditcard, aes(x = as.character(Class), y = Amount)) + geom_boxplot() + ggt
itle("Amount v/s Class") + labs(x="Class")
```




There's very large variability in genuine transaction amounts than the fradulent ones.

```
#Get Mean and  Median of the Amount-Class distribution
creditcard %>% group_by(Class) %>% summarise(mean(Amount), median(Amount))

## `summarise()` ungrouping output (override with `.groups` argument)

## # A tibble: 2 x 3
##   Class `mean(Amount)` `median(Amount)`
##   <int>          <dbl>            <dbl>
## 1     0           88.3            22
## 2     1          122.              9.25
```

Fradulent transactions seem to higher mean than Genuine ones, while on the other hand Fradulent have lower mean than the genuine, this suggests that the amount distribution for genuine transaction is right skewed. However this suggest that amount can be a significant predictor and it will be useful to keep it in our model.

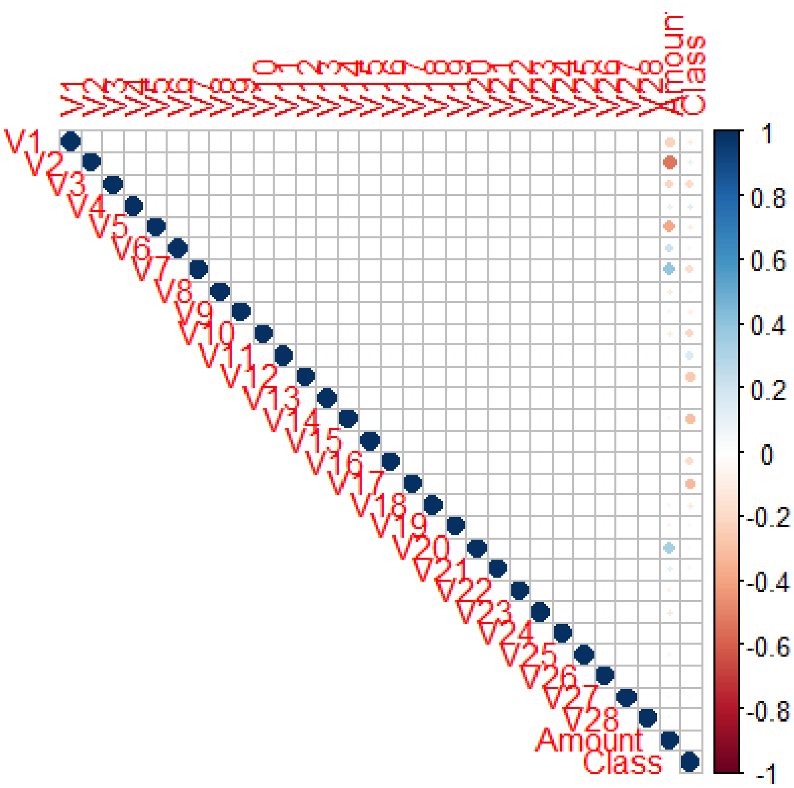
Now, as we discussed let's normalize the amount column as well

```
# function to normalize columns/arrays
normalize <- function(x){
      return((x - mean(x, na.rm = TRUE))/sd(x, na.rm = TRUE))
}
creditcard$Amount <- normalize(creditcard$Amount)
```

### 2.2.3 Correlation

Now that all our features/columns are normalized lets plotcorrelation chart to visualize correlation betwwen different all the variable and factor

```
# correlation plot
corr_plot <- corrplot(cor(creditcard[,-c("Time")]), method = "circle", type = "uppe
r")
```




## 2.3 Data Preparation

In this section we will clean our datset, transform our dataset to overcome bias and prepare test/train datasets

### 2.3.1 Data Cleaning

Here we will check for missing or inappropriate values in the dataset and will discuss how to deal with them.

```r
apply(creditcard, 2, function(x) sum(is.na(x)))
```

```
##   Time     V1     V2     V3     V4     V5     V6     V7     V8     V9 
##      0      0      0      0      0      0      0      0      0      0 
##    V10    V11    V12    V13    V14    V15    V16    V17    V18    V19 
##      0      0      0      0      0      0      0      0      0      0 
##    V20    V21    V22    V23    V24    V25    V26    V27    V28 Amount 
##      0      0      0      0      0      0      0      0      0      0 
##  Class 
##      0
```

Great News! There are no missing or NA values. No Data cleaning is required for our dataset.

### 2.3.2 Data Transformation

To avoid developing a naive model, we should make sure the classes are roughly balanced. Therefore, we will be using transformation techniques, particularly SMOTE to overcome this issue in our dataset.

#### *SMOTE*

SMOTE is a very famous and reliable ovesampling technique. It works roughly as follows:
1. The algorithm selects 2 or more similar instances of data
2. It then perturbs each instance one feature at a time by a random amount. This amount is within the distance to the neighbouring examples.

SMOTE has been shown to perform better classification performance in the ROC space than either over- or undersampling (From Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall and W. Philip Kegelmeyer's "SMOTE: Synthetic Minority Over-sampling Technique" (Journal of Artificial Intelligence Research, 2002, Vol. 16, pp. 321–357)). Since ROC is the measure we are going to optimize for, we will use SMOTE to resample the data.

### 2.3.3 Data Sampling

Finally we break out dataset in train and test sets. A good practice in general case scenario can be 90%-10% or 80%-20% split for train and test respectively.

#### *Special Bias Case*

As in our case the dataset is extremely biased, there are very low fraud cases and a 10% split will make them negilible and we will end up eveloping a biase predictor. Thus here a 50%-50% split is recommended.

But, if we are using oversampling techniques like SMOTE usual 80%-20% split should work just fine. In this project we will be using SMOTE with 80%-20% split.

#### *K-Fold Cross Validation*

Further, we will be using K-Fold Cross validation to avoid overfitting, we will go with usual K=10 in our first attempt.

```r
set.seed(56)

#Create Data Partition
train_index = createDataPartition(creditcard$Class, times = 1, p = 0.8, list = F)

#Distributing data to test and train sets
train = creditcard[train_index]
test = creditcard[!train_index]
train$Class <- as.factor(train$Class)
test$Class <- as.factor(test$Class)
levels(train$Class)=make.names(c("Genuine","Fraud"))
levels(test$Class)=make.names(c("Genuine","Fraud"))

# Uncomment registerDoMC to activate parallel processing
# Parallel processing for faster training
#registerDoMC(cores = 4)

# Use 10-fold cross-validation
ctrl <- trainControl(method = "cv",
                     number = 10,
                     verboseIter = T,
                     classProbs = T,
                     sampling = "smote",
                     summaryFunction = twoClassSummary,
                     savePredictions = T)
```

# 3. Methods and Analysis

## 3.1 Evaluation Scheme

### 3.1.1 Confusion Matrix

Sometimes, like in our case Accuracy won't tell the whole story, due to our class imbalance ratio, our model would be 99% accurate even if never deects a fraud. Thus we need to understand "True Positive", "True Negative", as well as "False Positive" and "False Negative". A Confusion Matrix is given as:

### 3.1.2 F1 Score

We will calculate F1-score to compare and analyse permformance of a model with different parameters. The formula for F1 is given as:

$$\frac{Precision. Recall}{Precision + Recall}$$

i.e.

$$\frac{2TP}{2TP + FP + FN}$$

where, TP is True Positive, FP is False Positive, and FN is False negative

### 3.1.3 AUC

To compare between different Models and given the class imbalance ratio, we recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC or AUC). Confusion matrix accuracy is not meaningful for unbalanced classification. AUC will be used to analayse performance of different Models and approach used in ths project

All these metric functions are available in library MLmetrics

```
#installing MLMatrix Pckage for F1_Score
if(!require(Mlmetrics)) install.packages("MLmetrics", repos = "http://cran.us.r-pro
ject.org")

## Loading required package: Mlmetrics

## Warning in library(package, lib.loc = lib.loc, character.only = TRUE,
## logical.return = TRUE, : there is no package called 'Mlmetrics'

## package 'MLmetrics' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Rishabh\AppData\Local\Temp\RtmpA1n0uj\downloaded_packages

library(MLmetrics)

## Warning: package 'MLmetrics' was built under R version 3.6.3

##
## Attaching package: 'MLmetrics'

## The following objects are masked from 'package:caret':
##
##     MAE, RMSE

## The following object is masked from 'package:base':
##
##     Recall

#installing e1071 package for confusion matrix and AUC
if(!require(e1071)) install.packages("e1071", repos = "http://cran.us.r-project.org
")

## Loading required package: e1071

## Warning: package 'e1071' was built under R version 3.6.3

library(e1071)
```

```r
#installing pROC package for ROC and AUC calculations
if(!require(pROC)) install.packages("pROC", repos = "http://cran.us.r-project.org")

## Loading required package: pROC

## Warning: package 'pROC' was built under R version 3.6.3

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

library(pROC)
```

## 3.2 Random Forest Model

As a first approach to this project we will built a Random Forest Classifier to set a benchmark and the will try tweaking its parameters and input Variables to enhance its performance.

The code below uses SMOTE to resample the data, performs 10-fold CV and trains a Random Forest classifier using ROC as metric to maximize

```r
#install caret, Classification Regresiion and training package
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org
")
library(caret)

#train RendomForst Model
RF_Model <- train(Class ~ ., data = train, method = "rf", trControl = ctrl, verbose
= T, metric = "ROC")

## + Fold01: mtry= 2

## Warning: package 'DMwR' was built under R version 3.6.3

## Loading required package: grid

## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo


## + Fold10: mtry=30
## - Fold10: mtry=30
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 2 on full training set
```

Let's see the results, how well our model fits on training data

```
RF_Model

## Random Forest
##
## 227846 samples
##     30 predictor
##      2 classes: 'Genuine', 'Fraud'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 205061, 205062, 205061, 205061, 205060, 205061, ...
## Addtional sampling using SMOTE
##
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##    2    0.9802746  0.9951639  0.8723077
##   16    0.9796598  0.9889033  0.8901923
##   30    0.9796859  0.9843265  0.8978846
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

Note: SMOTE resampling was done only on the training data. The reason for that is if we performed it on the whole dataset and then made the split, SMOTE would bleed some information into the testing set, thereby biasing the results in an optimistic way.

### 3.2.1 Correlation Matrix

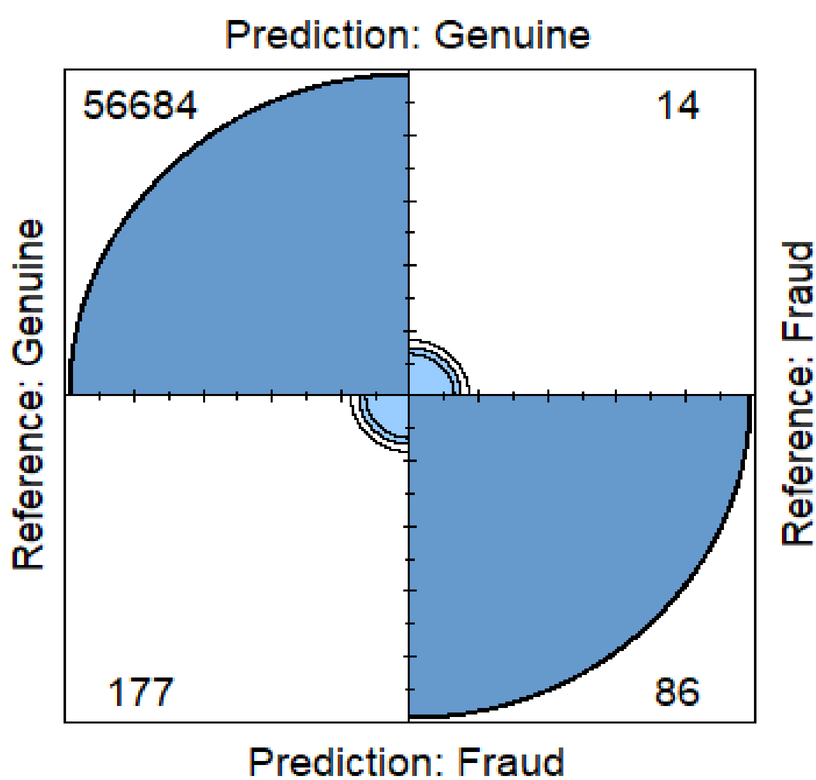
Now, Let's see our model performance on test dataset!

```
#get prediction for test
preds = predict(RF_Model, test, type = "prob")

# threshold is initially selected as 0.5
pred = as.factor(preds$Fraud>0.5)
levels(pred)=make.names(c("Genuine","Fraud"))
y_test = test$Class
#Calcuate Confusion Matrix

conf_mat_RF <- confusionMatrix(pred,y_test)
conf_mat_RF

## Confusion Matrix and Statistics
##
##             Reference
## Prediction Genuine Fraud
##    Genuine   56684    14
##    Fraud       177    86
```

```
##
##                Accuracy : 0.9966
##                  95% CI : (0.9961, 0.9971)
##     No Information Rate : 0.9982
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.4725
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9969
##             Specificity : 0.8600
##          Pos Pred Value : 0.9998
##          Neg Pred Value : 0.3270
##              Prevalence : 0.9982
##          Detection Rate : 0.9951
##    Detection Prevalence : 0.9954
##       Balanced Accuracy : 0.9284
##
##        'Positive' Class : Genuine
##
```

So we have got accuracy of 0.9966 with specificty 0.86 which is pretty good for the first model.

```
fourfoldplot(conf_mat_RF$table)
```

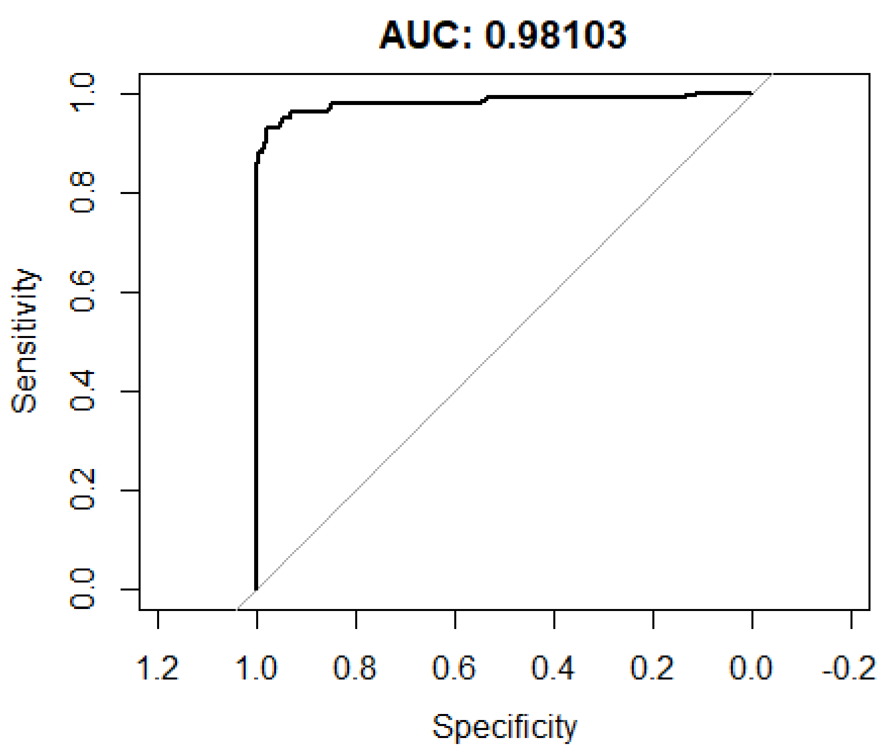


### 3.2.2 AUC

Now it's time to calculate AUC for our model. AUC is basically the area under the curve for Sensitivty v/s Specificity at different Threshold values. It is very useful metric to campare between diffeent classifiers.

AUC = 1 is ideal while 0.5 is worse

```
roc_data <- roc(y_test, predict(RF_Model, test, type = "prob")$Fraud)

plot(roc_data, main = paste0("AUC: ", round(pROC::auc(roc_data), 5)))
```




We got an AUC of 0.981 which is pretty good at this level.

Let's creat a evualation table with the final evaulation for our first Random Forest Model

```
#Specificty
Sp_RF = as.numeric(conf_mat_RF$byClass["Specificity"])

#RF F1_Score
F1_RF = round(F1_Score(y_test,pred),5)

#AUC
roc_data = roc(y_test, predict(RF_Model, test, type = "prob")$Fraud)

## Setting levels: control = Genuine, case = Fraud

## Setting direction: controls < cases
```

```
AUC_RF = round(pROC::auc(roc_data), 5)

# Create Results Table
result = tibble(Method = "Random Forest", Specificty = Sp_RF , F1Score = F1_RF, AUC
= AUC_RF)
result

## # A tibble: 1 x 4
##   Method        Specificty F1Score   AUC
##   <chr>              <dbl>   <dbl> <dbl>
## 1 Random Forest       0.86   0.998 0.981
```

## 3.3 Threshold Tuning

We see there there are only 14 cases that were actually fraudulent and missed out by our classifier. But one the same hand there are 177 genuine cases that were detect as Fraud but were genuine.

We can further change the threshold from 0.5 to False positives ,i.e, fraud transactions detected as Genuine. bUt this comes at a cost of more genuine transactions being identified as Fraud. So again, this a judgmental call to be made with concern of stake holders according to what are the exact needs and purpose.

Let’s try and simulate our model with different thresholds varying from 0.4 to 0.9 and calculate F1 Score for each

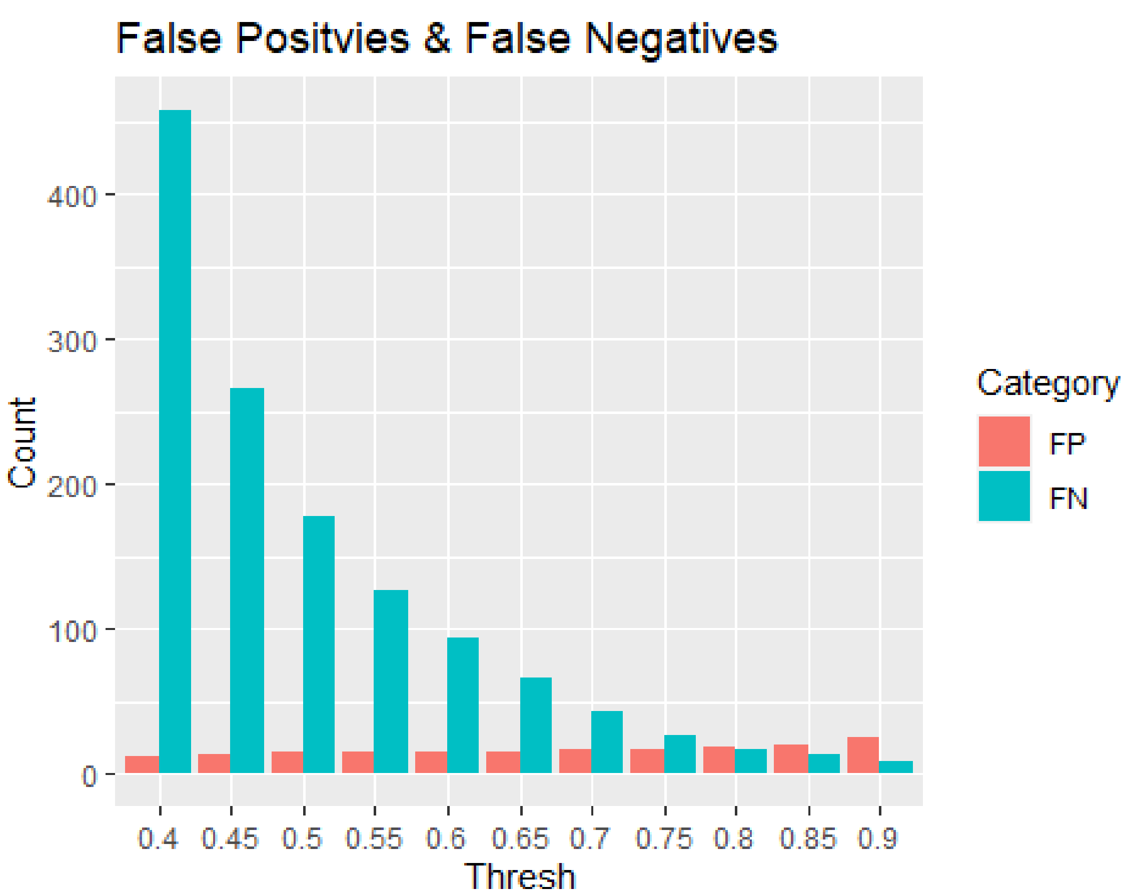
```
#define function to predict classes for variable threshold values
get_FPFN<- function(thresh,preds,y_test){

pred = as.factor(preds$Fraud>thresh)
levels(pred)=make.names(c("Genuine","Fraud"))
y_test = test$Class

#Calcuate Confusion Matrix
conf_mat_RF <- confusionMatrix(pred,y_test)
FP = conf_mat_RF$table[1,2]
FN =conf_mat_RF$table[2,1]
F1 = F1_Score(y_test, pred)
FPN = c(FP,FN,F1)
return (FPN)
}

# data frame to FPs and FNs for different values of threshold
FP_FN = data.frame(Thresh=character(), Count=character(), Category=character(), F1_
Score =numeric())

# run simulation for different threshold values b/w 0.1 and 0.9
for(i in seq(0.4,0.9,0.05)){
  FPN = get_FPFN(i,preds,y_test)
  FP = as.numeric(FPN[1])
```

```
  FN = as.numeric(FPN[2])
  F1 = as.numeric(FPN[3])
  rowFP = data.frame(Thresh=i,Count= FP,Category= "FP", F1_Score = F1)
  rowFN = data.frame(Thresh=i,Count= FN,Category= "FN", F1_Score = F1)
  FP_FN = FP_FN %>% rbind(rowFP)
  FP_FN = FP_FN %>% rbind(rowFN)
}
# mutate threshhold as character for bar plots
FP_FN = FP_FN %>% mutate(Thresh=as.character(Thresh))
```

### 3.3.1 FP & FN

Let's Plot and analyze Results of our simulation

```
FP_FN %>% ggplot(aes(x=Thresh,y=Count,group=Category, fill=Category)) + geom_col(st
at="identity", position="dodge")+ ggtitle("False Positvies & False Negatives")

## Warning: Ignoring unknown parameters: stat
```

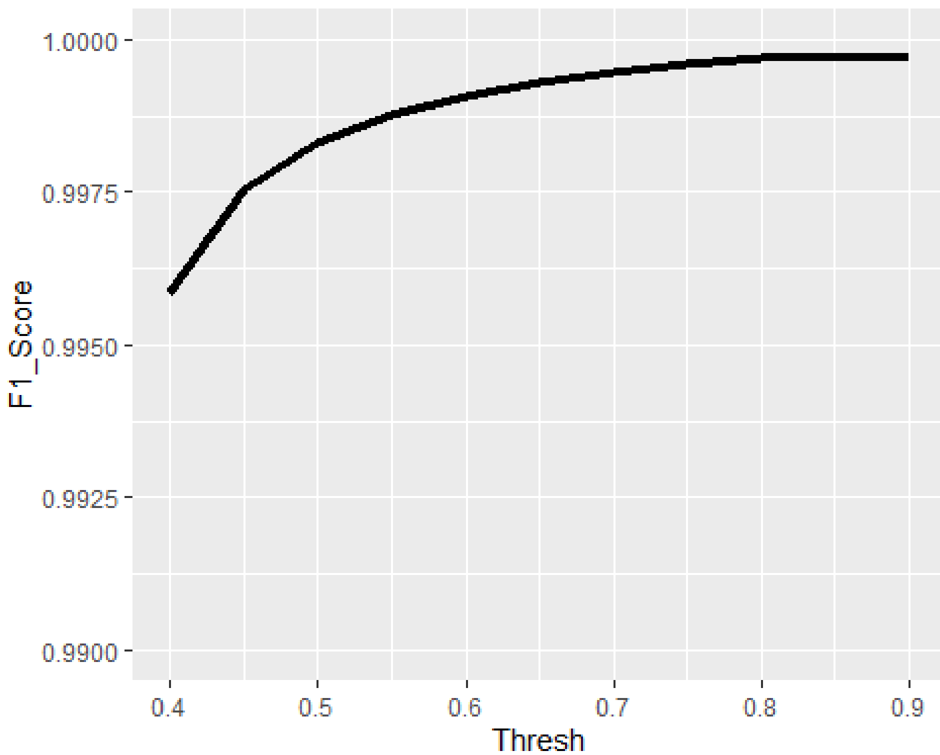



### 3.3.2 F1 Score

Let's plot the F-1 Score. F-1 Score is not really a good metric in this case as it evaluates the model while accounting for bth FPs and FNs.

F1 Score will be maximum when both are equal, in that case the numbers FPs becomes though equal to FN are to high. we can't let so many frauds to slip through our system for sake of decreasing FNs

```
F1Scores = FP_FN %>% group_by(Thresh) %>% summarise(F1_Score=mean(F1_Score)) %>% mutate(Thresh=as.numeric(Thresh))

## `summarise()` ungrouping output (override with `.groups` argument)

F1Scores %>% ggplot(aes(x=Thresh,y=F1_Score)) + geom_line(size=1.5) + ylim(0.99,1)
```




So we see, as wetighten the threshold the False detection of Fraud decrease exponentialy but the actual fraud cases slipping through our system increase.

Look, at 0.4 there are only 4 Fraud cases that slipped through our model! But the False negatives jumped to 465! Which is a lot and a bank would never want to charge their genuine customers for fraud.

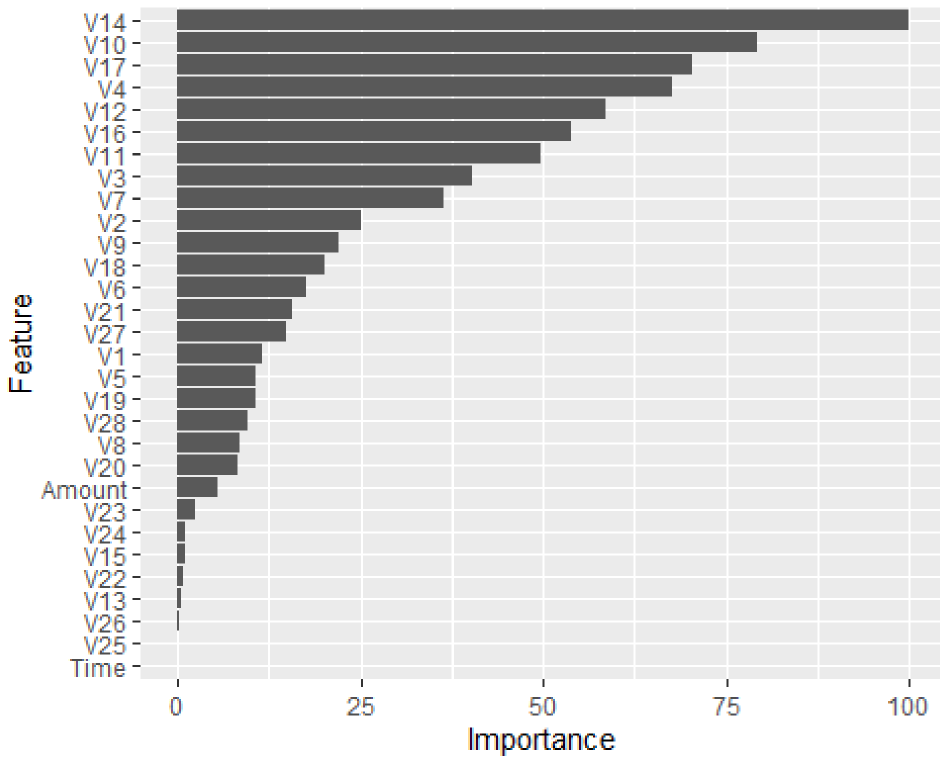
So, considering voth FP-NP distribution and the F1 Scores, our first choice 0.5 seems to be quite optimal and we will proceed with it.

## 3.4 Feature Engineering

Now that we have a achieved a benchmark very quickly, we should notice that there are 30 variables in total used by our model, but we saw in correlation plot that only very few variables have significant correaltion with the classification. Let's see if we can make our model simpler without losing accuracy, also we might end up increasing it.

Let's plot the the most imortant variables/Features and their significance

```
ggplot(varImp(RF_Model))
```




Now Let's see how our results vary if we us different RF models with different no. of variables, starting with single most important variable model.

Note: We will keep common 0.5 threshold for all models

We will use F1 score as the parameter to compare between different variable RF models.

***1 Variable Model***

```
F1Scores = data.frame(Variables = numeric(), F1= numeric())

#train 1-variable model
RF_Model = train(Class ~ V14, data = train, method = "rf", trControl = ctrl, verbos
e = T, metric = "ROC")

preds <- predict(RF_Model, test, type = "prob")

# threshold is selected as 0.5
pred = as.factor(preds$Fraud>0.5)
levels(pred)=make.names(c("Genuine","Fraud"))

F1Scores = F1Scores %>% rbind(data.frame(Variables =1, F1 = F1_Score(y_test,pred)))
```

***2 Variable Model***

```r
#train 2-variable model
RF_Model = train(Class ~ V14+V10, data = train, method = "rf", trControl = ctrl, ve
rbose = T, metric = "ROC")

#predict test dataset
preds <- predict(RF_Model, test, type = "prob")

# threshold is selected as 0.5
pred = as.factor(preds$Fraud>0.5)
levels(pred)=make.names(c("Genuine","Fraud"))

#add F1_Score
F1Scores = F1Scores %>% rbind(data.frame(Variables =2, F1 = F1_Score(y_test,pred)))
```

***3 Variable Model***

```r
#train 3-variable model
RF_Model = train(Class ~ V14+V10+V17, data = train, method = "rf", trControl = ctrl
, verbose = T, metric = "ROC")

#predict test dataset
preds <- predict(RF_Model, test, type = "prob")

# threshold is selected as 0.5
pred = as.factor(preds$Fraud>0.5)
levels(pred)=make.names(c("Genuine","Fraud"))

#add F1_Score
F1Scores = F1Scores %>% rbind(data.frame(Variables =3, F1 = F1_Score(y_test,pred)))
```

***4 Variable Model***

```r
#train 4-variable model
RF_Model = train(Class ~ V14+V10+V17+V4, data = train, method = "rf", trControl = c
trl, verbose = T, metric = "ROC")

#predict test dataset
preds <- predict(RF_Model, test, type = "prob")

# threshold is selected as 0.5
pred = as.factor(preds$Fraud>0.5)
levels(pred)=make.names(c("Genuine","Fraud"))

#add F1_Score
F1Scores = F1Scores %>% rbind(data.frame(Variables =4, F1 = F1_Score(y_test,pred)))
```

***6 Variable Model***

```r
#train 6-variable model
RF_Model = train(Class ~ V14+V10+V17+V4+V12+V16, data = train, method = "rf", trCon
trol = ctrl, verbose = T, metric = "ROC")
```

```
#predict test dataset
preds <- predict(RF_Model, test, type = "prob")

# threshold is selected as 0.5
pred = as.factor(preds$Fraud>0.5)
levels(pred)=make.names(c("Genuine","Fraud"))

#add F1_Score
F1Scores = F1Scores %>% rbind(data.frame(Variables =6, F1 = F1_Score(y_test,pred)))
```

***8 Variable Model***

```
#train 8-variable model
RF_Model = train(Class ~ V14+V10+V17+V4+V12+V16+V11+V3, data = train, method = "rf"
, trControl = ctrl, verbose = T, metric = "ROC")

#predict test dataset
preds <- predict(RF_Model, test, type = "prob")

# threshold is selected as 0.5
pred = as.factor(preds$Fraud>0.5)
levels(pred)=make.names(c("Genuine","Fraud"))

#add F1_Score
F1Scores = F1Scores %>% rbind(data.frame(Variables =8, F1 = F1_Score(y_test,pred)))
```

***10 Variable Model***

```
#train 10-variable model
RF_Model = train(Class ~ V14+V10+V17+V4+V12+V16+V11+V3+V7+V2, data = train, method
= "rf", trControl = ctrl, verbose = T, metric = "ROC")

#predict test dataset
preds <- predict(RF_Model, test, type = "prob")

# threshold is selected as 0.5
pred = as.factor(preds$Fraud>0.5)
levels(pred)=make.names(c("Genuine","Fraud"))

#add F1_Score
F1Scores = F1Scores %>% rbind(data.frame(Variables =10, F1 = F1_Score(y_test,pred))
)
```

***13 Variable Model***

```
#train 13-variable model
RF_Model = train(Class ~ V14+V10+V17+V4+V12+V16+V11+V3+V7+V2+V9+V18+V6, data = trai
n, method = "rf", trControl = ctrl, verbose = T, metric = "ROC")

#predict test dataset
preds <- predict(RF_Model, test, type = "prob")
```

```
# threshold is selected as 0.5
pred = as.factor(preds$Fraud>0.5)
levels(pred)=make.names(c("Genuine","Fraud"))

#add F1_Score
F1Scores = F1Scores %>% rbind(data.frame(Variables =13, F1 = F1_Score(y_test,pred))
)
```

### *18 Variable Model*

```
#train 18-variable model
RF_Model = train(Class ~ V14+V10+V17+V4+V12+V16+V11+V3+V7+V2+V9+V18+V6+V21+V27+V1+V
5+V19, data = train, method = "rf", trControl = ctrl, verbose = T, metric = "ROC")

#predict test dataset
preds <- predict(RF_Model, test, type = "prob")

# threshold is selected as 0.5
pred = as.factor(preds$Fraud>0.5)
levels(pred)=make.names(c("Genuine","Fraud"))

#add F1_Score
F1Scores = F1Scores %>% rbind(data.frame(Variables =18, F1 = F1_Score(y_test,pred))
)
```

### *All Variable Model*

```
#train all-variable model
RF_Model = train(Class ~ ., data = train, method = "rf", trControl = ctrl, verbose
= T, metric = "ROC")

#predict test dataset
preds <- predict(RF_Model, test, type = "prob")

# threshold is selected as 0.5
pred = as.factor(preds$Fraud>0.5)
levels(pred)=make.names(c("Genuine","Fraud"))

#add F1_Score
F1Scores = F1Scores %>% rbind(data.frame(Variables =30, F1 = F1_Score(y_test,pred))
)
```

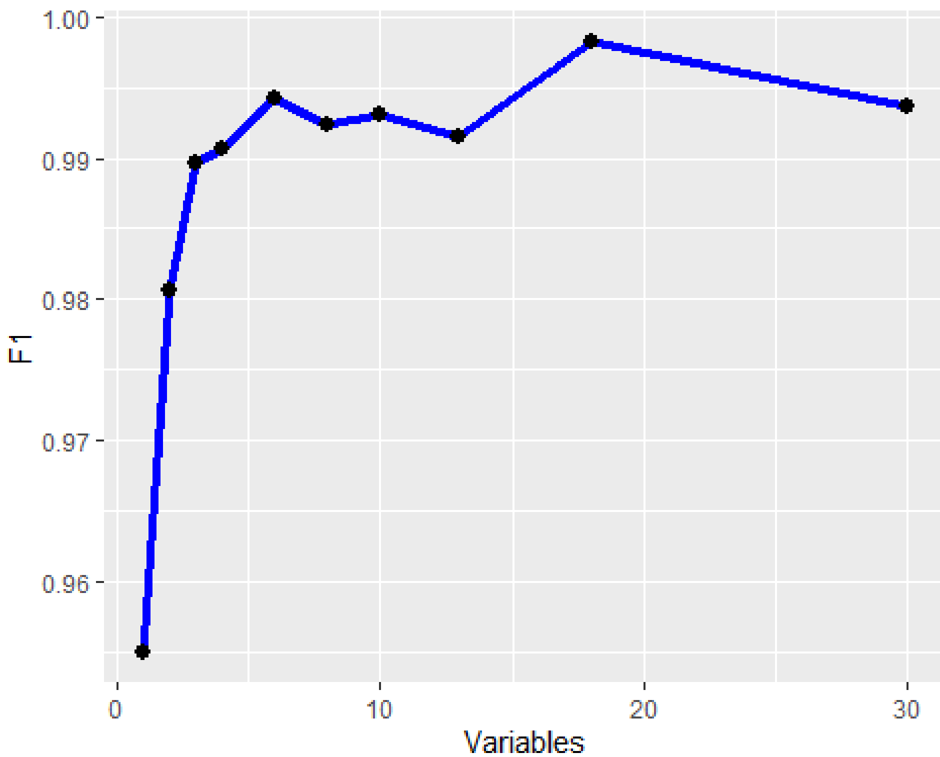
Now that we have calculated F1 scores for different variables models, let's see what we have got.

```
F1Scores

##   Variables        F1
## 1         1 0.9549823
## 2         2 0.9805514
## 3         3 0.9896160
```

```
## 4           4 0.9906290
## 5           6 0.9942882
## 6           8 0.9924262
## 7          10 0.9931575
## 8          13 0.9915244
## 9          18 0.9982651
## 10         30 0.9936920
```

It’s better to plot and visualize how F1-Score varies with including more of less important variables in our model.

```
F1Scores %>% ggplot(aes(x=Variables,y=F1)) + geom_line(size=1.5,colour="blue") + ge
om_point(size =2.5)
```




And there it is! Using Just Top 10 most important variables gives us best F1-Score. Now we can build our final Random forest model on this.

## 3.5 Optimized Random Forest Model

Now, we can build our final Random Forest Model with top 10 most inmportant variables Variables = V14+V10+V17+V4+V12+V16+V11+V3+V7+V2 Threshhold = 0.5

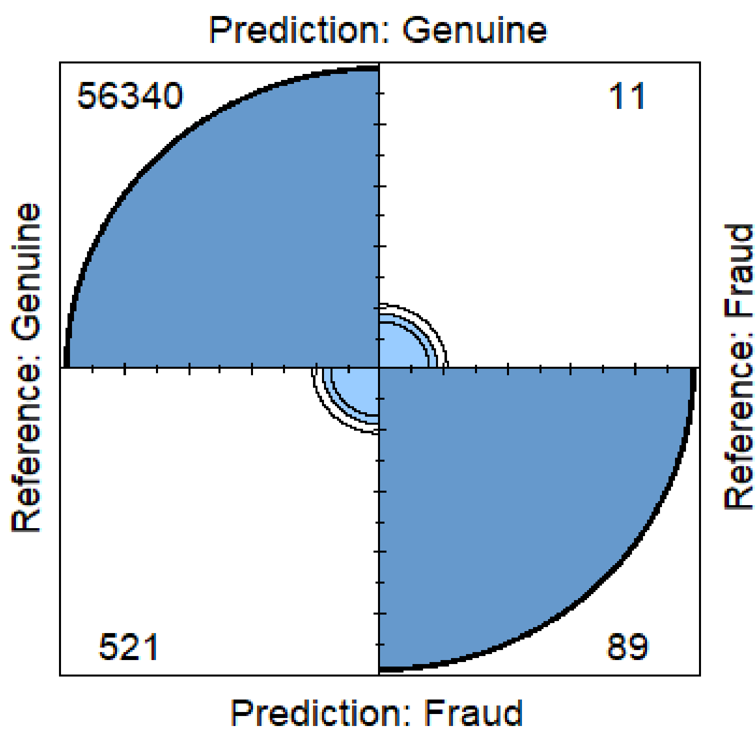
```
#train 10-variable model
RF_Model = train(Class ~ V14+V10+V17+V4+V12+V16+V11+V3+V7+V2, data = train, method
= "rf", trControl = ctrl, verbose = T, metric = "ROC")
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 6 on full training set
```

```
#predict test dataset
preds <- predict(RF_Model, test, type = "prob")

# threshold is selected as 0.5
pred = as.factor(preds$Fraud>0.5)
levels(pred)=make.names(c("Genuine","Fraud"))
```

### 3.5.1 Confusion Matrix

```
#Confusion Matrix
conf_mat_RF <- confusionMatrix(pred,y_test)

#Confusion Matrix Plot
fourfoldplot(conf_mat_RF$table)
```

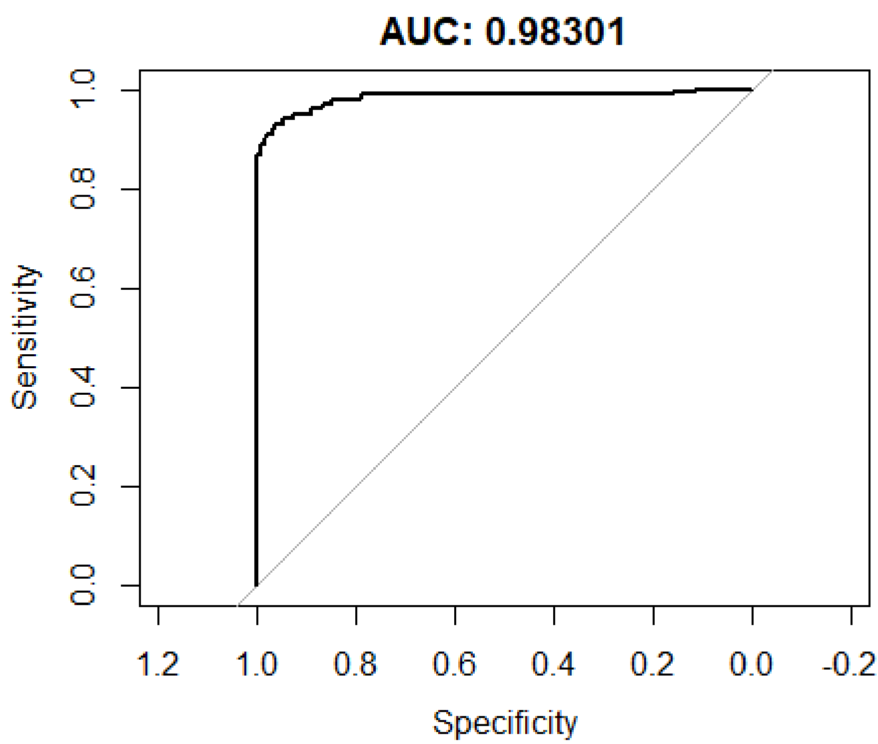



Here we see 13 False Positives, this is just one less than our original RF model, but this is just shows that fairly less complex models can achive better results sometimes

### 3.5.2 AUC

```
#plot ROC
roc_data = roc(y_test, predict(RF_Model, test, type = "prob")$Fraud)

## Setting levels: control = Genuine, case = Fraud
```

```
## Setting direction: controls < cases
plot(roc_data, main = paste0("AUC: ", round(pROC::auc(roc_data), 5)))
```




Add to final Evaluation Table

```
#Specificty
Sp_RF = as.numeric(conf_mat_RF$byClass["Specificity"])

#RF F1_Score
F1_RF = round(F1_Score(y_test,pred),5)

#AUC
roc_data = roc(y_test, predict(RF_Model, test, type = "prob")$Fraud)

## Setting levels: control = Genuine, case = Fraud

## Setting direction: controls < cases

AUC_RF = round(pROC::auc(roc_data), 5)

# Create Results Table
result = bind_rows(result, tibble(Method = "Random Forest (Optimized)", Specificty
= Sp_RF , F1Score = F1_RF, AUC = AUC_RF))
result

## # A tibble: 2 x 4
##   Method                          Specificty F1Score   AUC
```

```
##   <chr>                            <dbl>    <dbl> <dbl>
## 1 Random Forest                     0.86    0.998 0.981
## 2 Random Forest (Optimized)         0.89    0.995 0.983
```

## 3.6 XG-Boost Classifier

Lastly, we will implement XGBoost, which is based on Gradient Boosted Trees and is a more powerful model compared to both Random Forest

Installing and loading Xgboost Package

```
if(!require(xgboost)) install.packages("xgboost", repos = "http://cran.us.r-project
.org")

library(xgboost)
```

First we need to create matrix dataframes accepted by XG-Boost classifier.

```
#recreating test/train dataset for xgb based on previos train_index value
#as we tranformed Class coloum to factor in previous, it creats problem with XGB
train = creditcard[train_index]
test = creditcard[!train_index]

#create Data Matrix form for XGB
dtrain <- xgb.DMatrix(data = as.matrix(train[,-c("Class")]), label = train$Class)
dtest <- xgb.DMatrix(data = as.matrix(test[,-c("Class")]), label = test$Class)
```

Now that we are done let's put on training. I'm using most usual hyperparameters settings for XG-Boost. Although you are encouraged to change some numbers and see wher it takes you.

```
xgb <- xgboost(data = dtrain, nrounds = 100, gamma = 0.1, max_depth = 10, objective
= "binary:logistic", nthread = 7)

## [1]  train-error:0.000386
## [2]  train-error:0.000334
…
## [100]    train-error:0.000000
```

Let's run our model on test dataset and check out the results

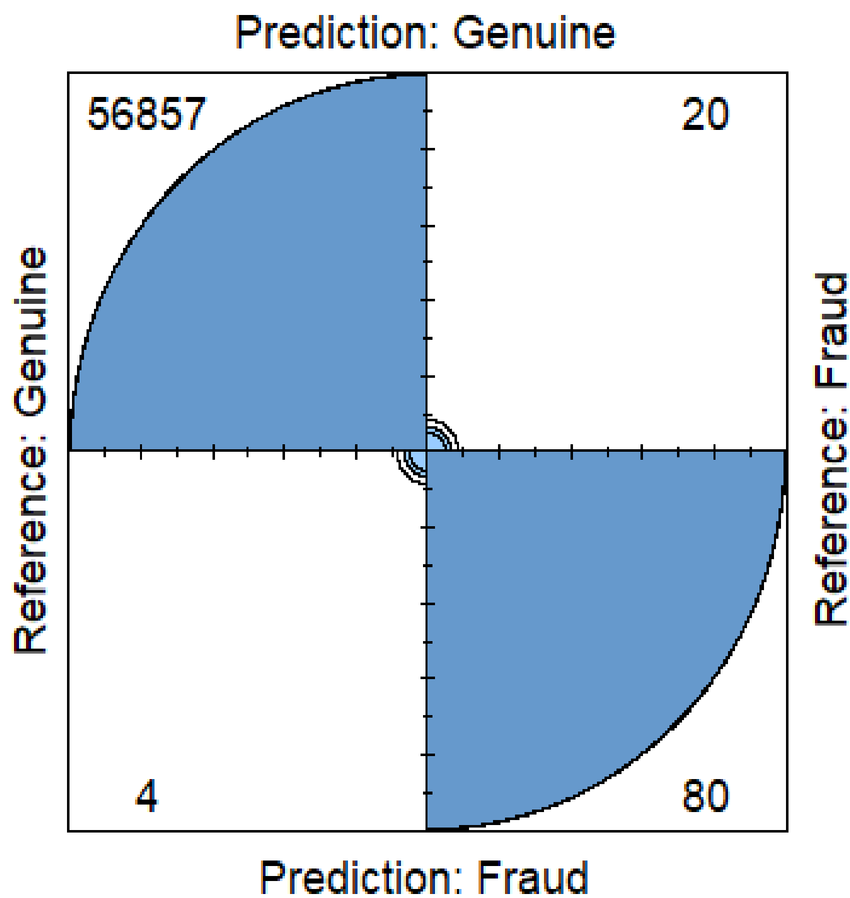
```
#Run Predictions
preds_xgb <- predict(xgb, dtest)

#Convert Prediction to Factors for Confusion Matrix
pred_fac = as.factor(preds_xgb>0.5)
levels(pred_fac)= make.names(c("Genuine","Fraud"))
```

```r
y_test = as.factor(test$Class)
levels(y_test)=make.names(c("Genuine","Fraud"))
```

### 3.6.1 Confusion Matix

The results of XGBoos can be visulaized in confusion matrix to know about how well our model performs with False positives and False negatives

```r
conf_mat_XGB = confusionMatrix(pred_fac, y_test)
conf_mat_XGB

## Confusion Matrix and Statistics
## 
##            Reference
## Prediction Genuine Fraud
##    Genuine   56857    20
##    Fraud         4    80
## 
##                Accuracy : 0.9996
##                  95% CI : (0.9994, 0.9997)
##     No Information Rate : 0.9982
##     P-Value [Acc > NIR] : <2e-16
## 
##                   Kappa : 0.8694
## 
##  Mcnemar's Test P-Value : 0.0022
## 
##             Sensitivity : 0.9999
##             Specificity : 0.8000
##          Pos Pred Value : 0.9996
##          Neg Pred Value : 0.9524
##              Prevalence : 0.9982
##          Detection Rate : 0.9982
##    Detection Prevalence : 0.9985
##       Balanced Accuracy : 0.9000
## 
##        'Positive' Class : Genuine
## 

#Confusion Matrix Plot
fourfoldplot(conf_mat_XGB$table)
```

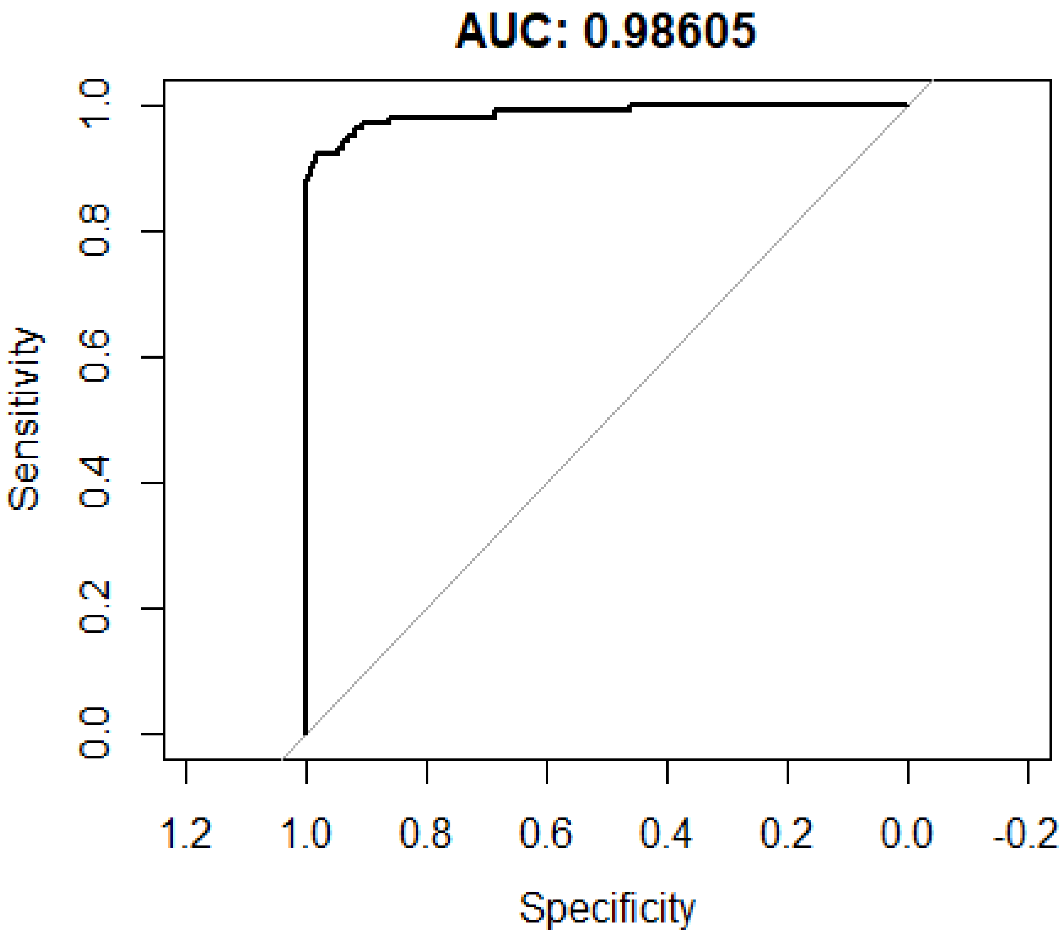



There are only 4 False Negatives!! while the no. of Flase positives has just increased by 3!

### 3.6.2 AUC

To compare the perfomance with other Random Forest Models AUC can be a good metric and is recommended. As we saw in previous case where F1-Score can be misguiding with biased datasets

```
#plot ROC
roc_data <- roc(y_test, preds_xgb)

## Setting levels: control = Genuine, case = Fraud

## Setting direction: controls < cases

plot(roc_data, main = paste0("AUC: ", round(pROC::auc(roc_data), 5)))
```




We have AUC of 0.986!! Clearly XGB achieved considerably higher AUC compared to Random Forest Models

## 4. Results

Let's first make the final evaluation table

```
Sp_XGB = as.numeric(conf_mat_XGB$byClass["Specificity"])

#RF F1_Score
F1_XGB = round(F1_Score(y_test,pred_fac),5)

#AUC
roc_data = roc(y_test, preds_xgb)

## Setting levels: control = Genuine, case = Fraud

## Setting direction: controls < cases

AUC_XGB = round(pROC::auc(roc_data), 5)

# Create Results Table
result = bind_rows(result, tibble(Method = "XG-Boost", Specificty = Sp_XGB , F1Scor
e = F1_XGB, AUC = AUC_XGB))
result

## # A tibble: 3 x 4
##   Method                      Specificty F1Score   AUC
##   <chr>                            <dbl>   <dbl> <dbl>
## 1 Random Forest                     0.86   0.998 0.981
## 2 Random Forest (Optimized)         0.89   0.995 0.983
## 3 XG-Boost                          0.8    1.00  0.986
```

Clearly, XG-Boost outperforms both Random Forest anf Optimized Random Forest Classifier models. Though a we loose on the specificty of the model, but that is considerably low compared to reduction in false negatives.

In Random Forest models we could best achieve 14 False positives with 177 false negatives, but XGboost drops False negative to 4! i.e., 173 units compared to increase in False positives to 20, i.e, only 7 units.

Though both the models have their own limitations and can be made more better, we can declare XG-boost as the winner under the scope of this project.

| **METHOD**<br><CHR> | **SPECIFICITY**<br><DBL> | **F1SCORE**<br><DBL> | **AUC**<br><DBL> |
|---|---:|---:|---:|
| **RANDOM FOREST** | 0.86 | 0.99832 | 0.98103 |
| **RANDOM FOREST (OPTIMIZED)** | 0.91 | 0.99003 | 0.98215 |
| **XG-BOOST** | 0.80 | 0.99979 | 0.98605 |

# 5. Conclusion

Throughout the project we focused on building a classification model to dete credit card frauds. We explored, analyzed and visualized data to understand relationships between variables/features and used the insights to develop a suitable classification model. We set a benchmark with Random Forest Classifier plugging in complete dataset with all variables to use. Then we performed threshold tuning and feature engineering to get a optimal threshold value, and also filtered out less inmportant variables which helpen making the model simple yet enhancing the performance.

We saw how using less features and simple model can be better that using directly everything and end building a complex model that is computationaly expensive and neither has signifiacantly better performance.

Lastely, we used XG-Boost classifier, gardient boosted trees that outperfomed both the models.

## 5.1 Limitations

All models have their own limitations, we saw that even after optimizing and usind advaced classifiers like XG-Boost, there is still a trade off between False Positives and False negatives and thus at the end it is the users Judgement Model on what is he/she more interested in and what the business goals are.

## 5.2 Future Work

To build a more robust model which could solve the False positives and negatives, a Deep Learning approach can outperform many other models. Especially in our case that includes too many normalized variables, Deep Learning nueral nerworks perform extremely well.

Other than deep learning we should also try to to explore capalities of XG-Boost more. We achived considerably well results in our first attempt. We can try tweaking different paramenter and see how it effects our results.

You can find my other related work on my github repository here:
https://github.com/rishabhdodeja/

***End of the Document***