Fraud Detecting Flora

Kostas Voulgaropoulos

Sept 2019

Contents

1 Introduction

1.1 Problem Set

This kernel aims to create a machine learning algorithm able to detect fraud in financial transactions. The underlying reasoning is that an algorithm trained to predict fraud correctly can efficiently detect and stop fraudulent transactions in real time.

For that, we employ a publicly available dataset generated by the PaySim simulator by Dr. Edgar Lopez. For ease of reproduction and computation, we isolated the first 1 million rows of it.

More specifically, it contains 10 predictors of numeric, integer and character type.

Our target variable is categorical (i.e. label, not quantity) and binary (meaning the number of distinct classes are 2).

For starters we explore our data with an extensive Exploratory Data Analysis. With the insights the EDA equips us, we perform data reduction and engineering whenever possible. Because of the great imbalance in the frequency of the two classes (99% No_Fraud vs 1% Fraud), we undersample the prevalent class to reach a 60/40 ratio. Finally, although our algorithms do not require normalized data to work, we center and scale our numerical features for speed of computations.

The algorithms employed for modelling are classification tree (rpart library) and random forest (rf library). These are chosen because they are good at handling different types of features (since we end up with factor, numerical and logical predictors) and relatively fast to train. On the other hand, their disadvantages include low interpretability (they are actually black boxes), great memory occupation and potential overfitting in case we don't tune their parameters fine. These barriers can be bypassed.

Success of our algorithms is evaluated using the Area under the Receiver Operating Characteristics (ROC) curve. One could argue that financial institutions prefer to be more conservative in their anti-fraud policy (thus prefering an algorithm prone to classifying transactions as frauds) and desire to minimize the algorithm's False Negatives (since we consider No Fraud class as positive), measured by the Sensitivity metric. But on the other hand, institutions could also care about customer satisfaction and do not wish to unnecessarily block non fraudulent transactions classified as frauds. Under this scenario, they would seek to minimize False Positives (cases where the algo predicted Fraud even though it wasn't) and thus maximize specificity. A pretty convenient solution that takes into consideration both strategies is the ROC curve. This curve is constructed by using model's Specificity on the x-axis and Sensitivity on the y-axis.

Because our algorithms are trained on the balanced train set, we also test it in the unbalanced test set, where fraud cases are very rare. Success rates are close to those obtained from the balanced dataset.

1.2 Library and Data Import

To begin, we need to import the necessary libraries.

```
# Data Manipulation
if(!require(tidyverse)) install.packages("tidyverse",
                                repos = "http://cran.us.r-project.org")
# showing multiple plots
if(!require(gridExtra)) install.packages("gridExtra",
                                repos = "http://cran.us.r-project.org")
# Assisting gridExtra
if(!require(grid)) install.packages("grid",
                                repos = "http://cran.us.r-project.org")
# Model training
if(!require(caret)) install.packages("caret",
                                repos = "http://cran.us.r-project.org")
# Pairplot
if(!require(GGally)) install.packages("GGally",
                                repos = "http://cran.us.r-project.org")
# ROC Curve
if(!require(pROC)) install.packages("pROC",
                                repos = "http://cran.us.r-project.org")
# ROC Curve
if(!require(ROCR)) install.packages("ROCR",
                                repos = "http://cran.us.r-project.org")
#Refactoring
if(!require(radiant.data)) install.packages("radiant.data",
                                repos = "http://cran.us.r-project.org")
```

Regarding the data, it exists in a [Github repository] (https://github.com/voulkon/Fraud_Detection) as a zip file.

If you clone the repo and run the .Rmd file contained in it, it will work without any further actions.

In any other case, please modify the variable zipfile so that it points to the path of the data. One convenient way is to use the choose.files() function.

```
# Relative path - assumes data and .rmd file are in the same directory
# Github Repository : https://github.com/voulkon/Fraud_Detection
zipfile <- paste0(getwd(),"/PaySim1M.zip") %>%
  str_replace_all( "/", "\\\\" )

# if data is not the same dir
# enable the line below and point it to the zip file location
#zipfile <- choose.files(caption = "Please choose zipfile location")

#unzip and read the csv contained in it
df <- unzip(zipfile = zipfile) %>%
        read.csv(stringsAsFactors = FALSE)#, nrow = 8000)
```

2 EDA and Data Cleaning

Successfull machine learning requires good knowledge of the underlying data. This knowledge can help us: * eliminate unnecessary observations/features, * transform the remaining and * create new ones.

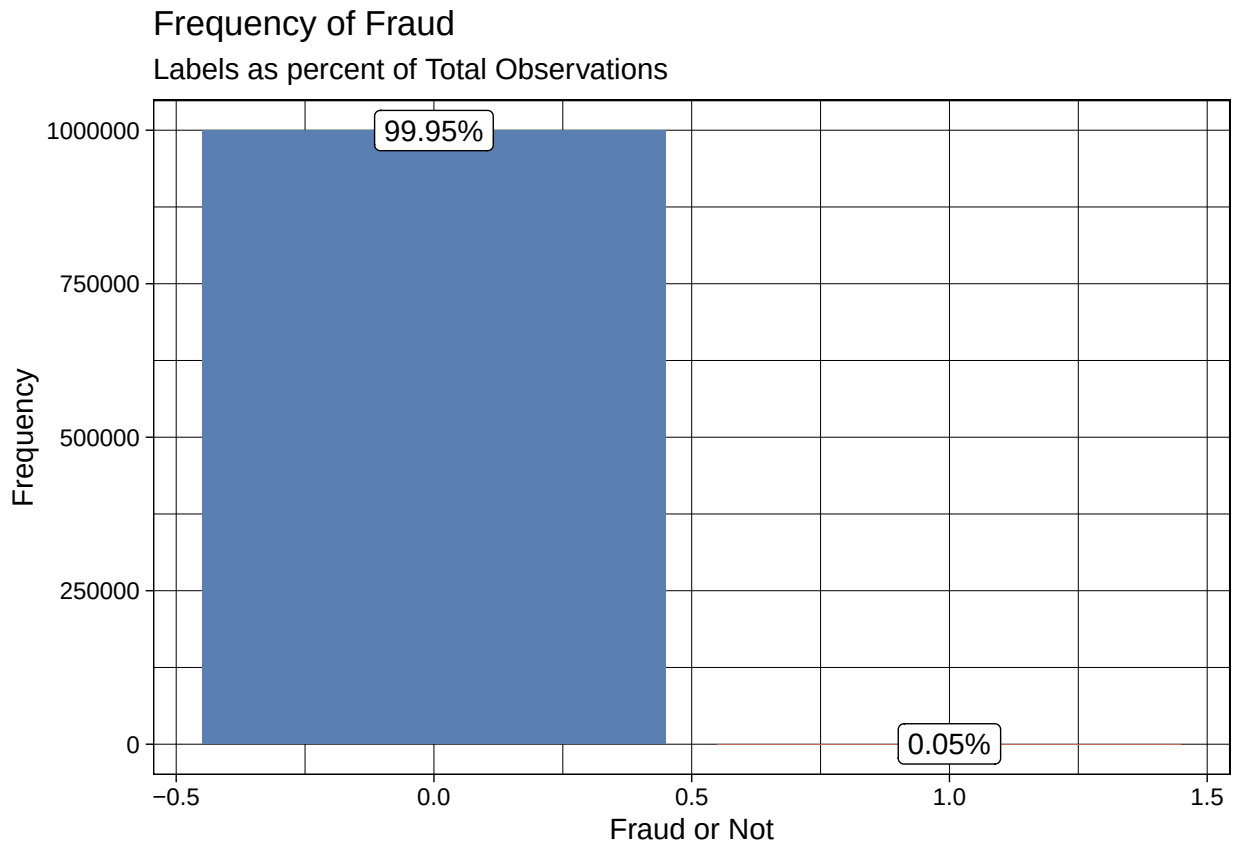
In this section we will explore our data with Exploratory Data Analysis and at the same time reconstruct them. Before we start, let's take a look at the data.

```
## 'data.frame':    1000000 obs. of  11 variables:
##  $ step          : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ type          : chr  "PAYMENT" "PAYMENT" "TRANSFER" "CASH_OUT" ...
##  $ amount        : num  9840 1864 181 181 11668 ...
##  $ nameOrig      : chr  "C1231006815" "C1666544295" "C1305486145" "C840083671" ...
##  $ oldbalanceOrg : num  170136 21249 181 181 41554 ...
##  $ newbalanceOrig: num  160296 19385 0 0 29886 ...
##  $ nameDest      : chr  "M1979787155" "M2044282225" "C553264065" "C38997010" ...
##  $ oldbalanceDest: num  0 0 0 21182 0 ...
##  $ newbalanceDest: num  0 0 0 0 0 ...
##  $ isFraud       : int  0 0 1 1 0 0 0 0 0 0 ...
##  $ isFlaggedFraud: int  0 0 0 0 0 0 0 0 0 0 ...
```

Before we dive into analyzing our features, let's quickly look at our target variable.

2.1 Class Balance of Target Variable

For starters, we will look at the **frequency** and the **percentage** of each type of **target** variable (Fraud vs Not Fraud).



This great imbalance between the two target classes will lead our model to a bias towards the dominating variable. This means that since more than 99% of the cases are not frauds our algorithm will feel comfortable with predicting "Not Fraud". It has a 99% chance to be right.

Later, in the preprocessing section, we will undersample the majority class to create a more balanced dataset. We could also synthesize minority instances with a popular and more sophisticated technique called SMOTE but regarding low-dimensional data, simple undersampling tends to outperform it in most situations. In addition, the size of our dataset is sufficient to exempt us from the need to create artificial data.

2.2 Factors

Looking at the dataset structure, we can observe that, apart from the six numeric features, we have three integer and three character columns. Integer columns get treated as factors when they don't express actual numeric value, but they just mask factor features. Because it seems that that's our case, we include the integer columns under the umbrella of factors.

As for each column separately, from this first impression:

- *isFlaggedFraud* looks like the prediction of the existing algorithm. We don't want our model to depend on someone else's prediction, thus we will get rid of it.
- *isFraud*, the target variable, will be turned into a factor and renamed for ease of plotting (the factor levels will become meaningful labels).
- *nameDest* and *nameOrig* look random and without predictive power, but we will explore them thoroughly in case we can extract any value from them.
- *type* is a potentially powerful feature. We can easily hypothesize that fraud is prone to some types of transactions. For example, cash in and payment (giving money) are quite improbable to co-exist with a

fraud scheme. This will also get explored in detail.

- *step* is just the time variable. It will probably be of no use in our prediction as fraud can happen at any time. Still, we will treat it as predictive until otherwise proved.

For starters, we will take a look at the number of unique values the character columns contain. This will help us shape our cleaning strategy.

```
## [1] "Number of Distinct Values in Character Vectors"
```

```
##     type nameOrig nameDest
##        5   999759   422896
```

type contains only 5 distinct values. We will explore them thoroughtly soon.

nameOrig and *nameDest* (name of origin and name of destination respectively), on the other hand, contain way too many distinct values to be treated as factors. The numbers are obviously random, probably generated by a hash function. Nevertheless, the prefix might symbolize some implicit information (for example the type of account - be it a savings or checking account etc.).

2.2.1 Name of Origin / Destination

To test our assumption, let's look at the number of distinct prefixes and their frequency.

```
## [1] "Distinct Prefixes in nameOrig and nameDest columns : "
```

```
##                   dest_name_prefix
## origin_name_prefix      C      M
##                  C 670247 329753
```

It appears there are only two prefixes, C and M, almost equally present in the *name_dest_pref* column. However, *name_orig_pref* contains only C's, thus it could have no predictive power.

As a result, we can

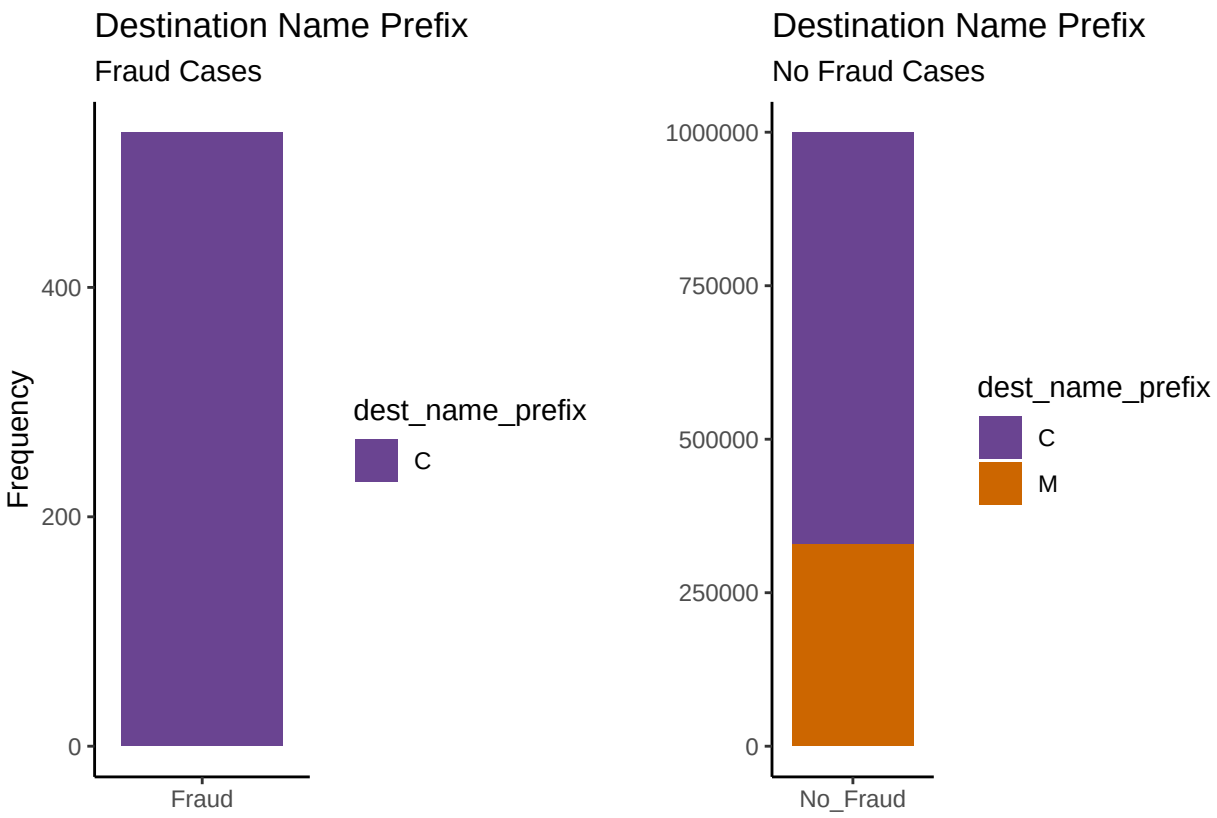
- **add** the **destination prefix** column,
- **omit** creating the respective feature from the **origin prefix** and
- get **rid** of the **initial columns**.
- On top of that, that's a great opportunity to get **rid** of the useless to us *isFlaggedFraud* column.

In addition, we will turn the remaining integer and character columns into **factors**.

Finally, to render our plots more readable, we will **rename** the **balance columns** and the **levels of the target variable**.

2.2.2 Destination Names

Before we eliminate these transactions types, we will approach in a similar manner the *dest_name_prefix column* to account for similar effects.

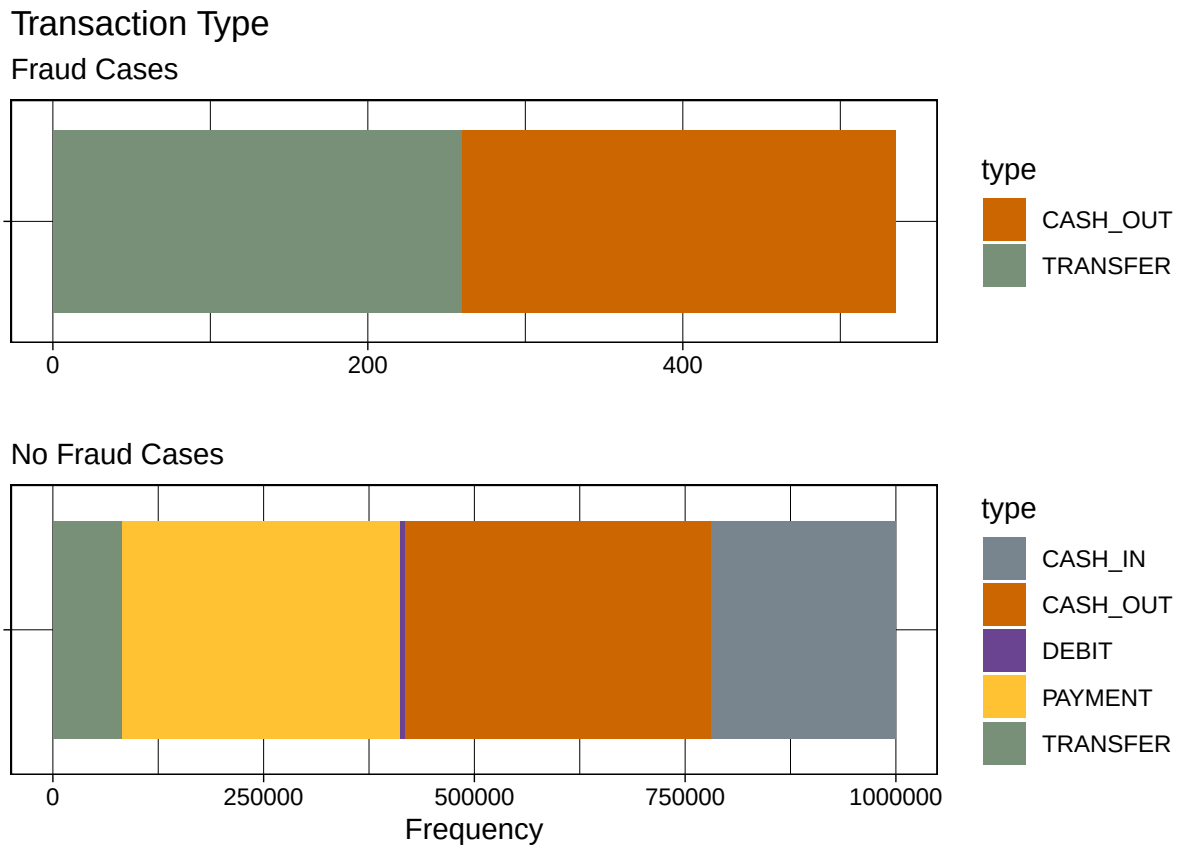


Accordingly, fraud coincides only with the "C" prefix. We could remove observations containing the "M". After the removal, this column contains only "C"s (ie zero variance), meaning we can drop it.

This process became of use to us not as a new column that can predict the result, but as a signal that led us to discard a significant amount of useless rows.

2.2.3 Transaction Type

Next, we will explore the presence of each transaction type in the fraud vs the not fraud cases.



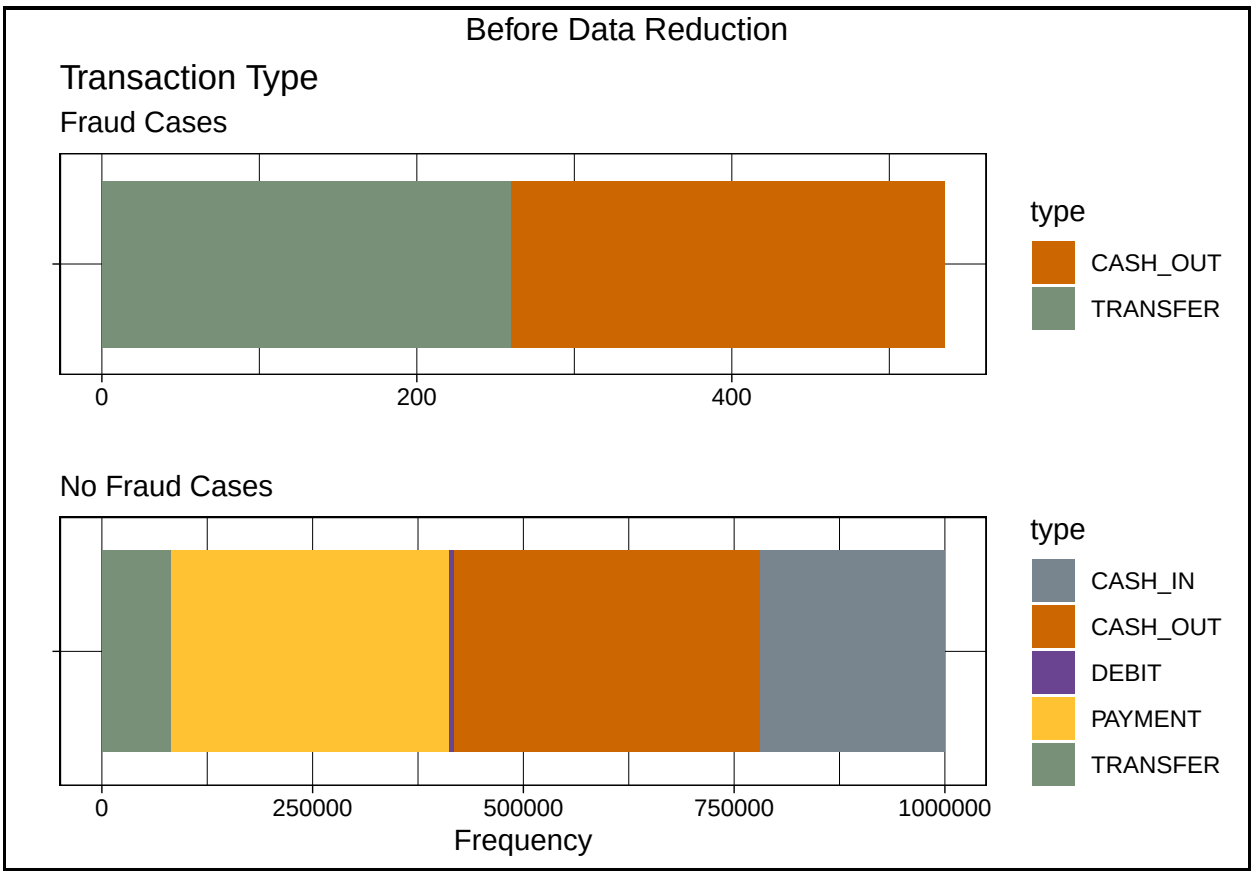
Interestingly enough, frauds coincide only with Cash Out and Transfer transactions. That means the rest of the transaction types (Cash in, Debit, Payment) are never associated with fraud. We will get rid of all observations with these transactions, since they only lead to "Not Fraud".

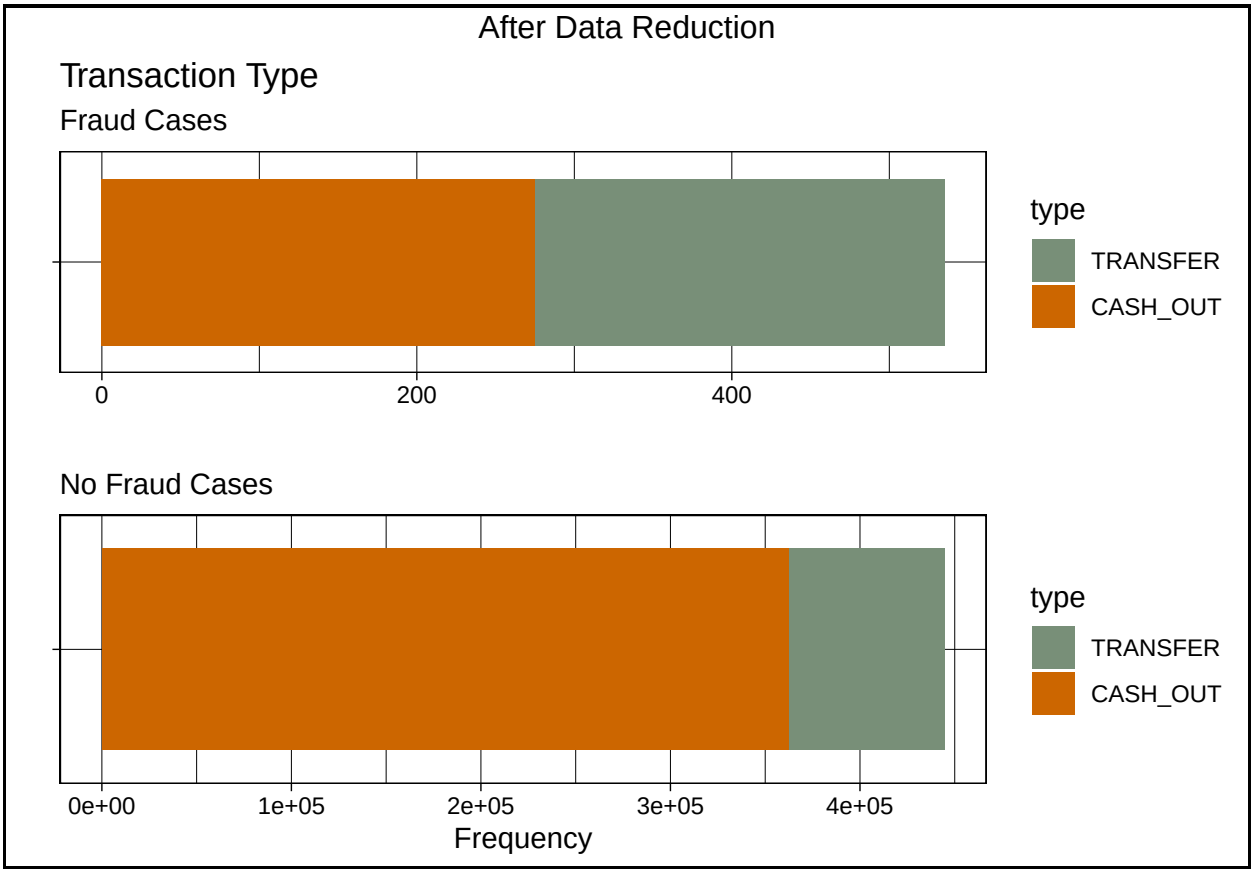
Before we continue, let's look at how many rows we exterminated by cleaning the name and type columns.

```
## [1] "554900 rows removed, equivalent to 55.49%  of the data"
```

More than half of the initial rows can be thrown away. Good news for us!

And now let's visualize the transactions column before and after reduction.


Before Data Reduction
Transaction Type
Fraud Cases
0
200
400
type
CASH_OUT
TRANSFER
No Fraud Cases
0
250000
500000
750000
1000000
Frequency
type
CASH_IN
CASH_OUT
DEBIT
PAYMENT
TRANSFER

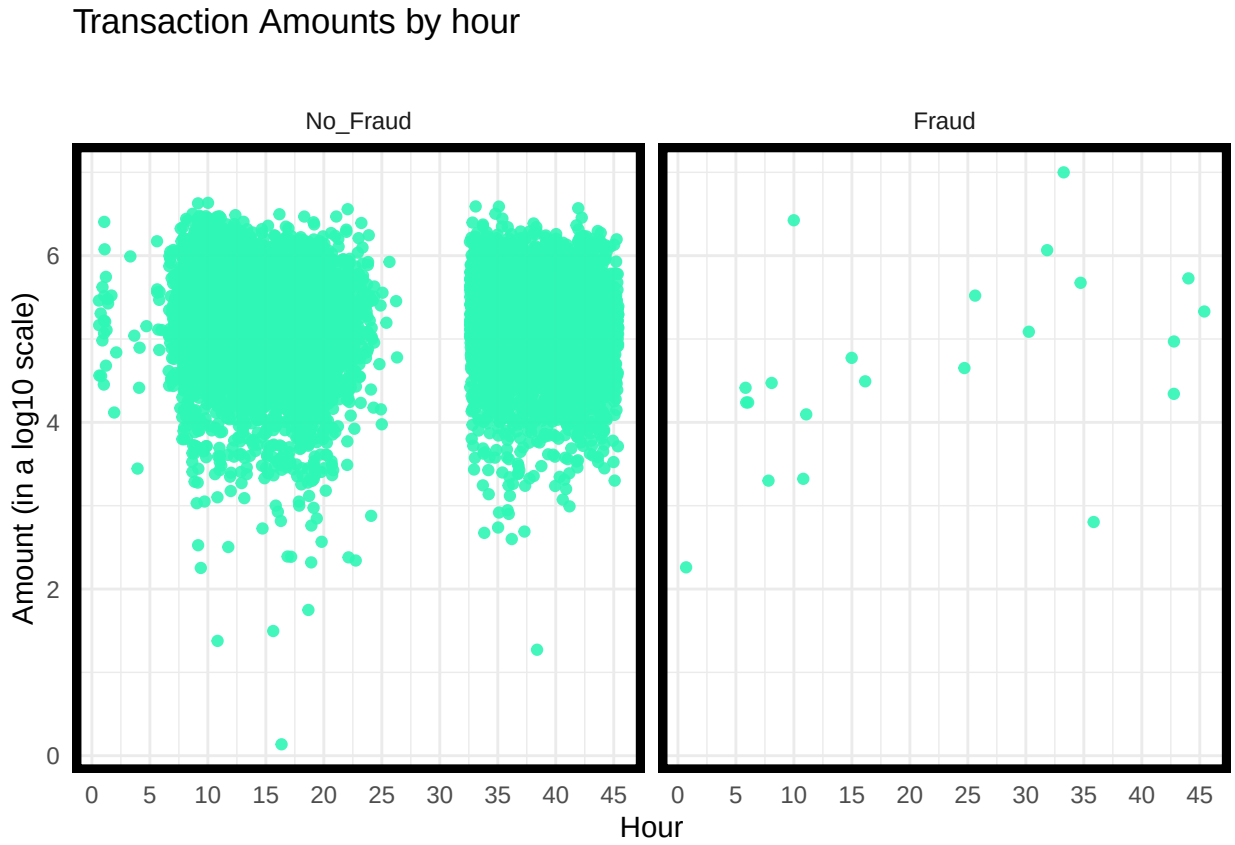


Looks like the no fraud cases contain more cash out transactions, while in frauds transfers are equally present. This might be a hint of the predictive power of this specific feature.

2.2.4 Across Time

Even though it might seem irrational that frauds happen on a certain time instance, we can plot amount vs time coloured by isFraud column in case there is any obvious pattern.

Because observations are so many, they cover one another and our plot becomes uninformative. For that, we will randomly sample 10% of our data. This way we reduce the points plotted while avoiding biases.



Apparently some hours contain little to no data. This could be due to some special circumstance (a strike or a power cut) or just because the simulator didn't happen to produce enough data along these hours.

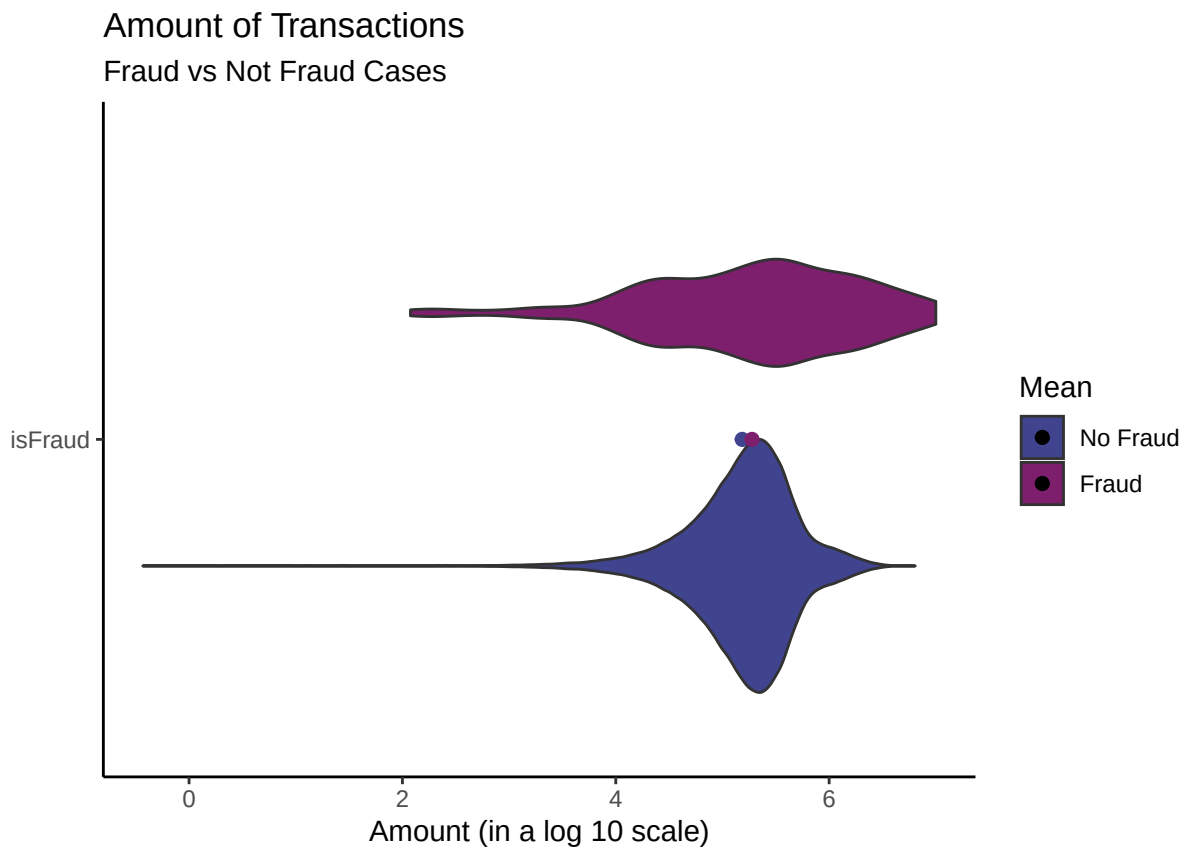
Fraud and Not Fraud cases look randomly scattered throughout time and do not seem to form clusters. However, there are some slight concentrations of frauds in some hours. This could be due to the fact that when fraudsters manage to commit fraud they try to maximize the number of transactions in the tight time window they have.

2.3 Numerics

Now that we explored the character columns, let's look at our numerics.

2.3.1 Amount

Since our numerics range from petty to enormous amounts, we will plot them after we apply the log10 function on them. This way, the distance between low and high amounts gets minimized and it becomes easier for the human eye to detect patterns.



Oddly enough, the amount of transaction looks different in the fraud vs not fraud cases. No_fraud amounts range around 100,000 (10^5) while fraud cases are more evenly distributed. This could be a hint that the amount column can make a contribution in our model.

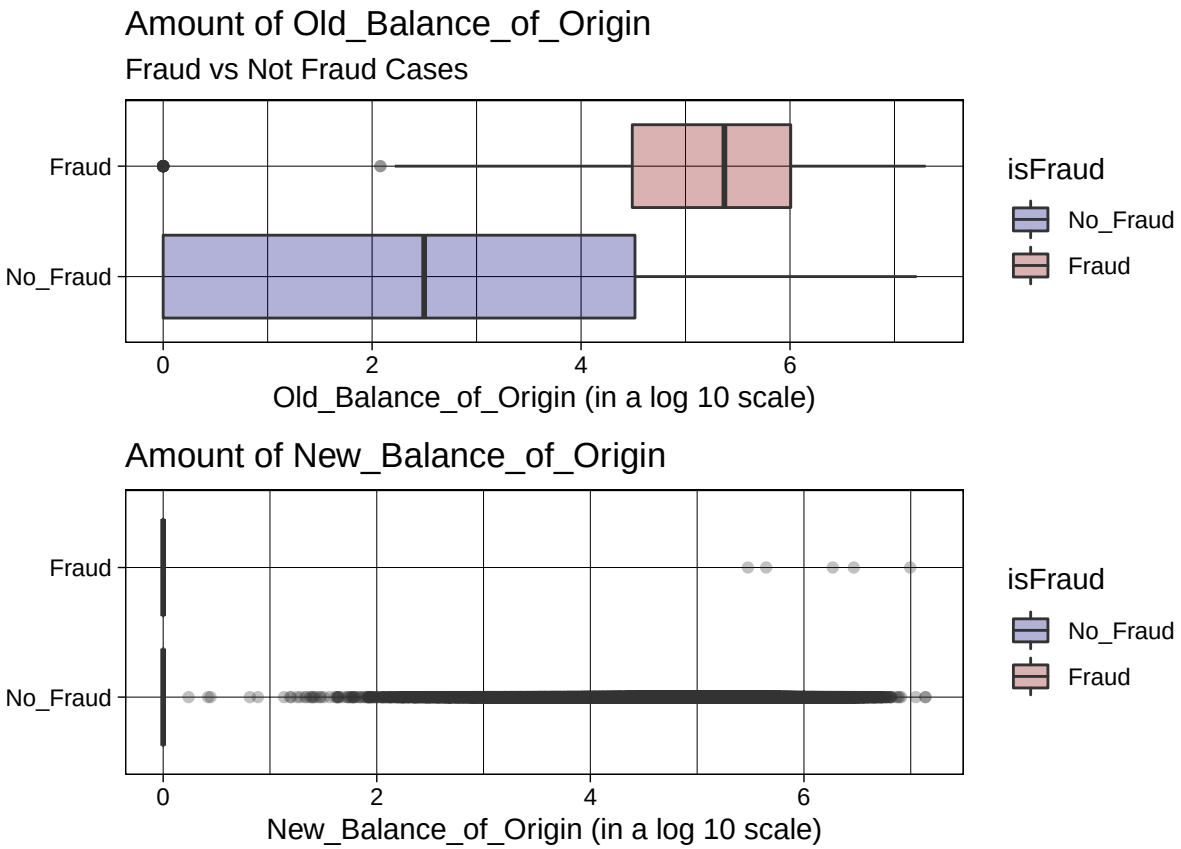
2.3.2 Balance Columns

In contrast with the amount column, the *Balance* columns contain zero values. Zeros return a -Infinity when log is applied on them.

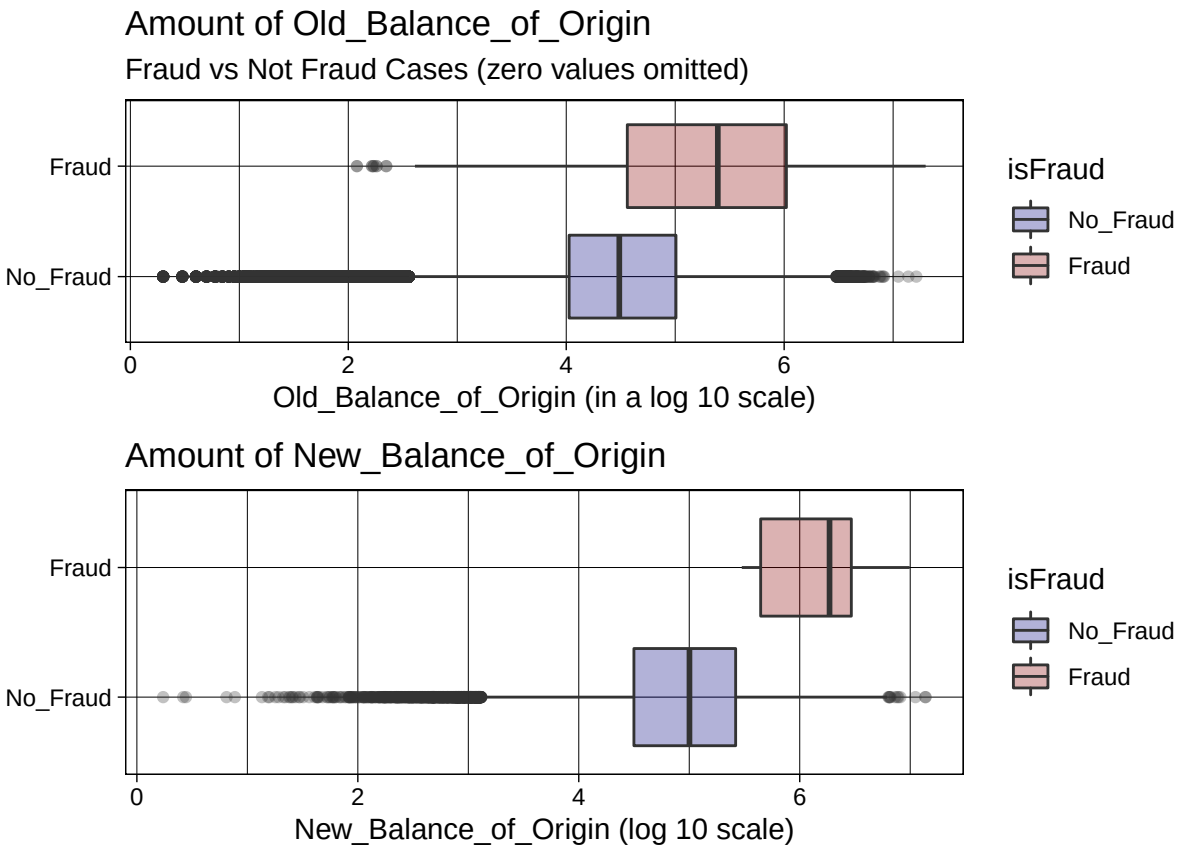
A convenient solution to this problem is to add 1 on our numeric vector before we pass it to the log function. This way 0s become 1s and yield a 0 after the function is applied. This happens because whichever number raised to the power of 0 returns 1.

2.3.2.1 Balance of Origin

We start with the origin balances.



We observe a very high concentration of 0 values that render our plots completely uninformative. Let's try to omit them.

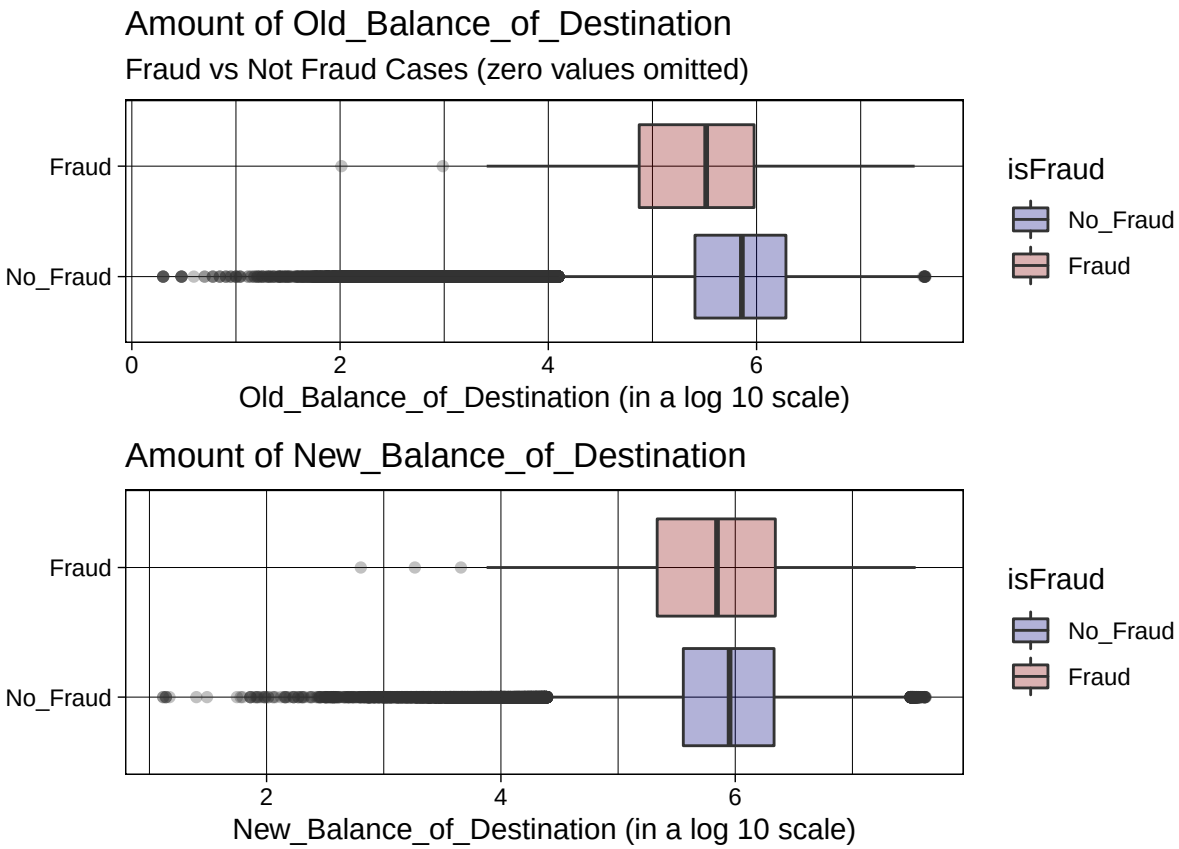


A lot better.

Both in *Old* and in *New Balance of Origin*, frauds tend to happen in greater amounts of balances. This could also be a sign of predictive power.

2.3.2.2 Balance of Destination

We will follow the same conventions for **Balances of Destination**.



In contrast with the origin balances, destination balances do not differ much between fraud and not_fraud cases.

3 Feature Engineering

3.1 The integer effect

Effective fraudsters might think that transfering a not round amount (eg 12,325 instead of 10,000) will seem like a more ordinary transaction and won't raise any suspicions. But on the other hand, they might not get into that trouble. Moreover, the human mind always comes up with rounded numbers and that would be the first thought of every (not so professional) fraudster. That's why we will check whether round amounts of transactions coincide with fraud.

```
#Create a new isRound column
df <- mutate(df, isRound =
               as.factor(
                 ifelse( test = (df$amount - as.integer(df$amount)) == 0 ,
                         yes = 'round', no = 'float' )))
```

Interestingly enough, round numbers seem to be more prevalent in fraud occurences. That's probably a clue of a valuable feature. (Note that in the no fraud cases I used a brighter yellow because otherwise it wouldn't be distinguishable)

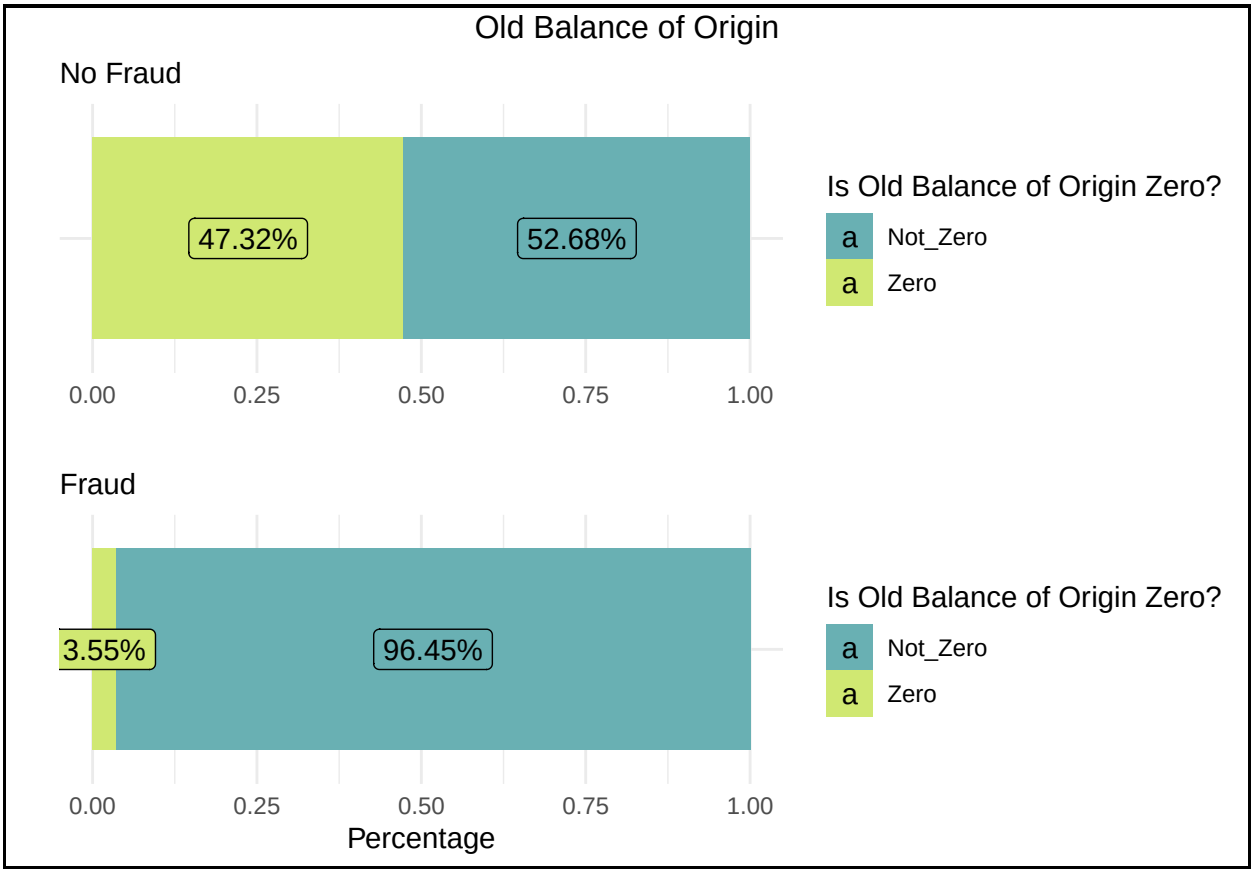
3.2 Is Balance Zero?

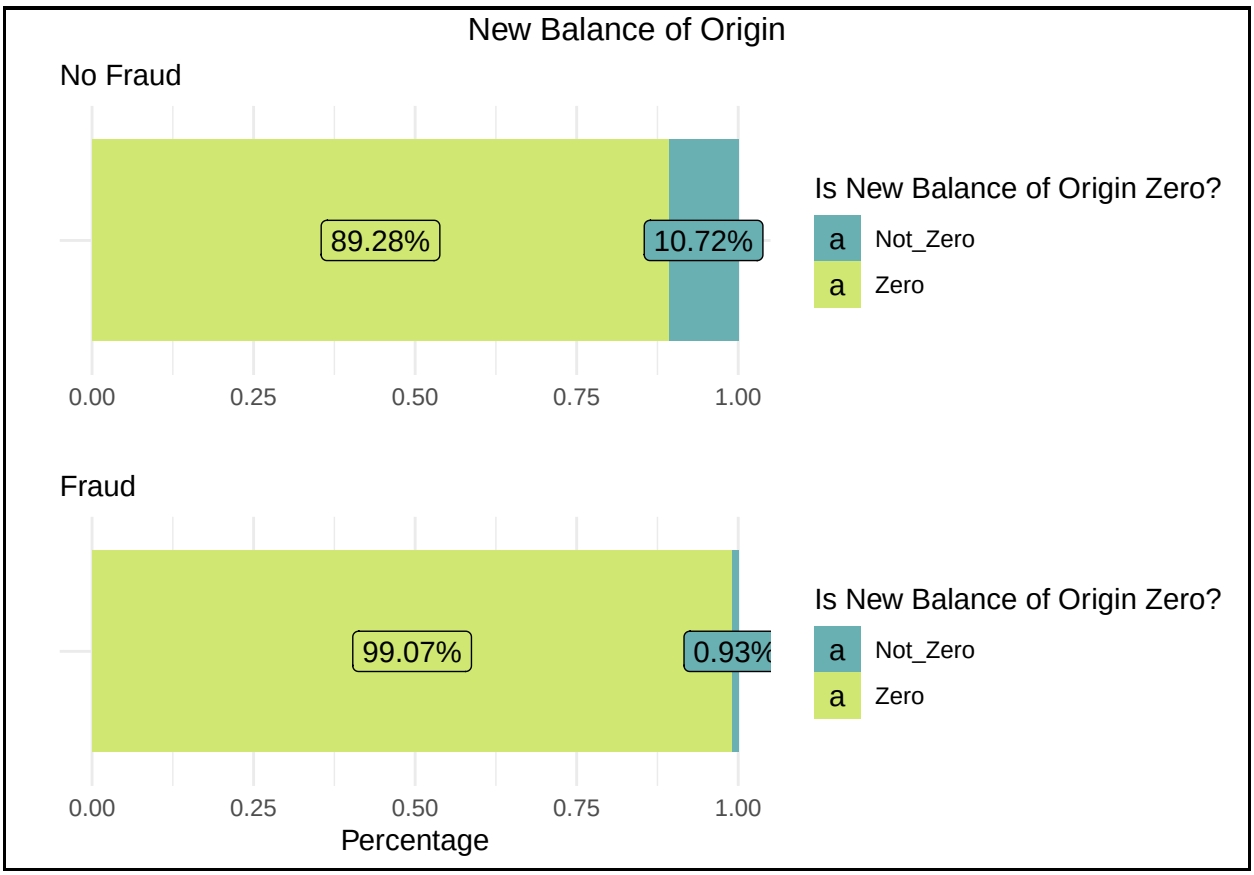
Another assumption one could make is that fraudsters use dummy accounts to send or (mostly) receive money. This is why it would be interesting to explore whether accounts with 0 balances are connected with fraud. For that, we will create a logical vector for each Balance column we already have .

```
iszero <- df %>% select(-amount) %>% #exclude the amount that cannot be 0
    select_if(is.numeric) %>% #select the remaining numerics - ie the balances
    mutate_all(R.utils::isZero) %>% #check whether they are zero
    rename_all(~paste0('isZero_', substr(., 1, nchar(.) - 4))) #add an isZero prefix on the names

df <- df %>% cbind(iszero) #merge with original df
rm(iszero)
```

3.2.1 Origin

First, we will take a look at the origin accounts.


Old Balance of Origin
No Fraud
47.32%
52.68%
0.00
0.25
0.50
0.75
1.00
Is Old Balance of Origin Zero?
a
Not_Zero
a
Zero
Fraud
3.55%
96.45%
0.00
0.25
0.50
0.75
1.00
Percentage
Is Old Balance of Origin Zero?
a
Not_Zero
a
Zero

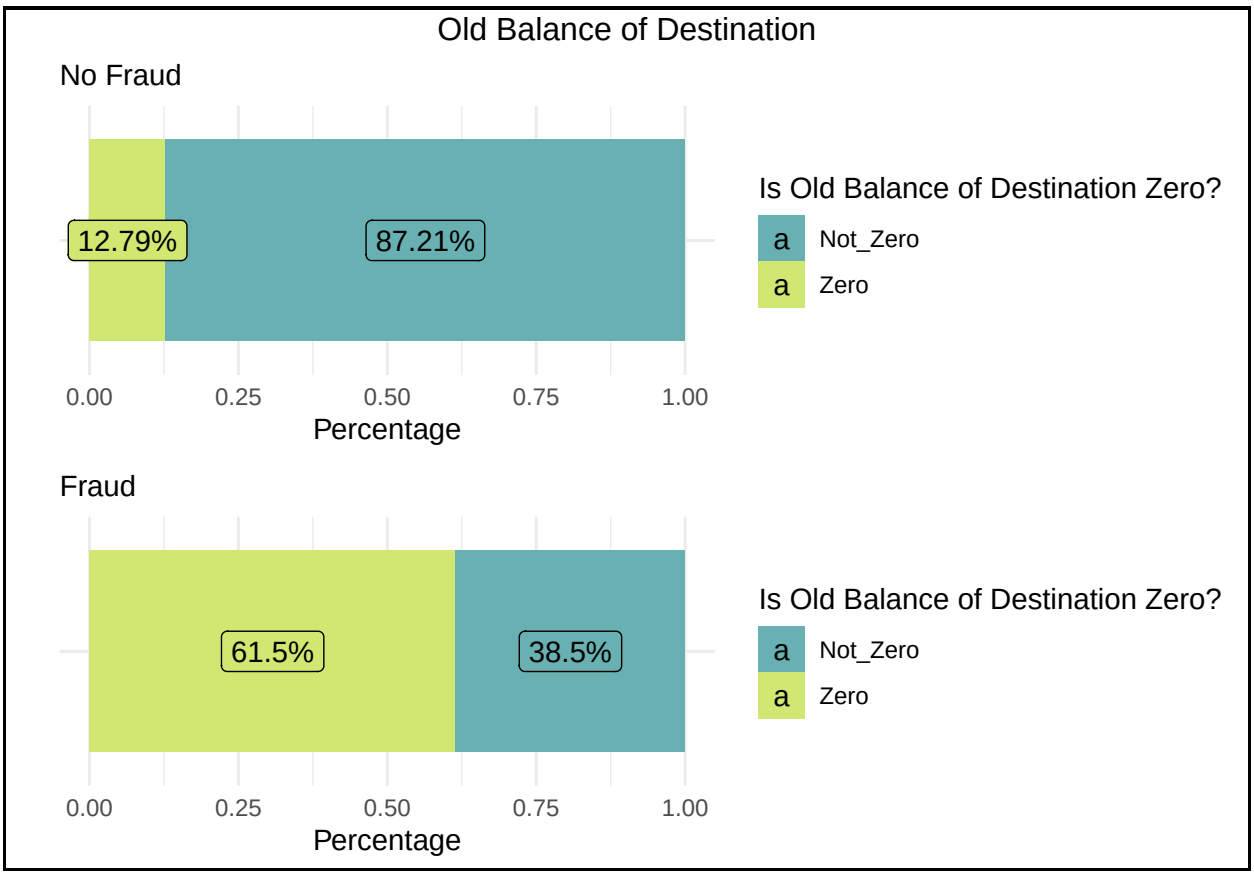


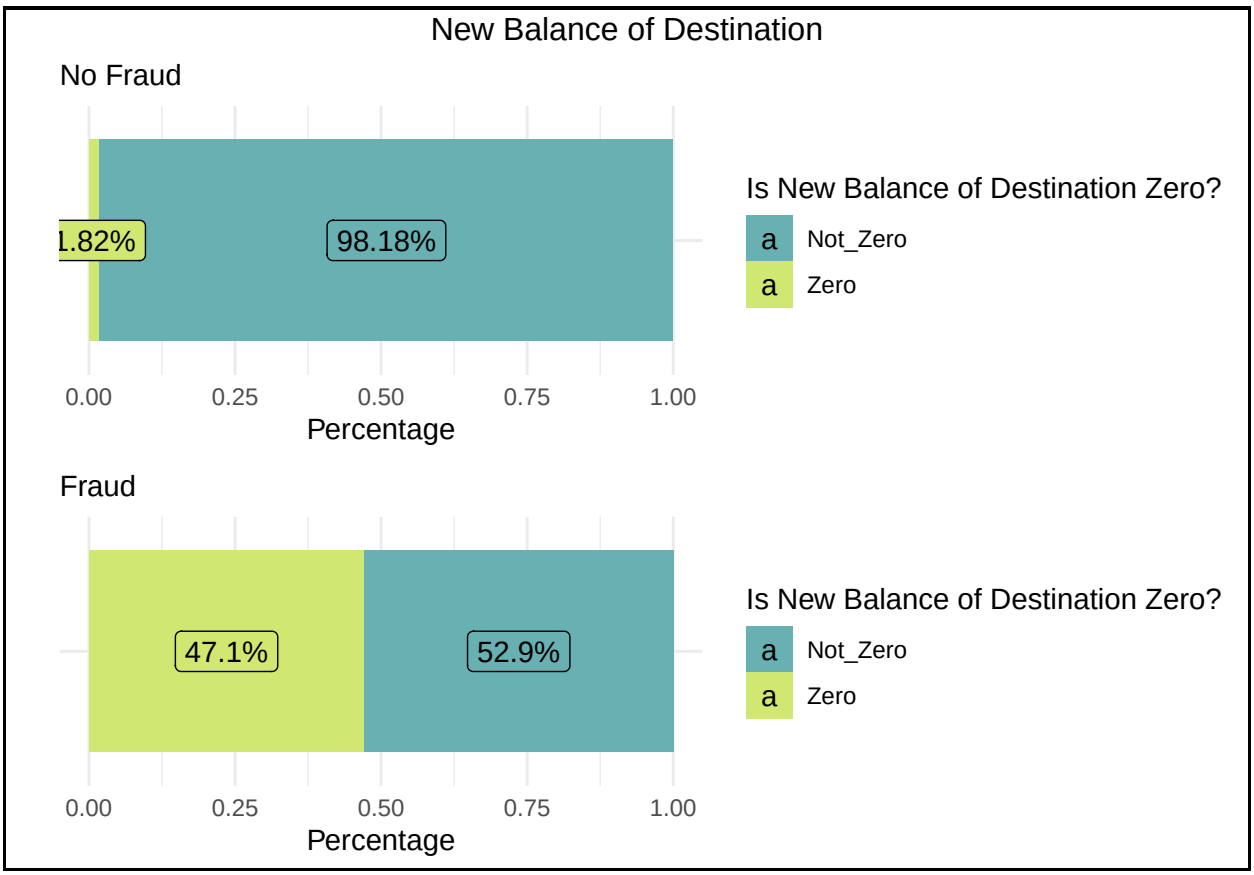
In Old Balance of Origin, fraud cases very seldom contain zero balances. That's expected because there's no point for a fraudster to target a zero balance account.

In New Balance of Origin, on the other hand, 98% of the fraud cases leave it with zero balance (in other words, fraudsters loot the accounts to zero). That could prove zero New Balances of Origin as valuable predictors.

3.2.2 Destination

Now time for the Balances of Destination.

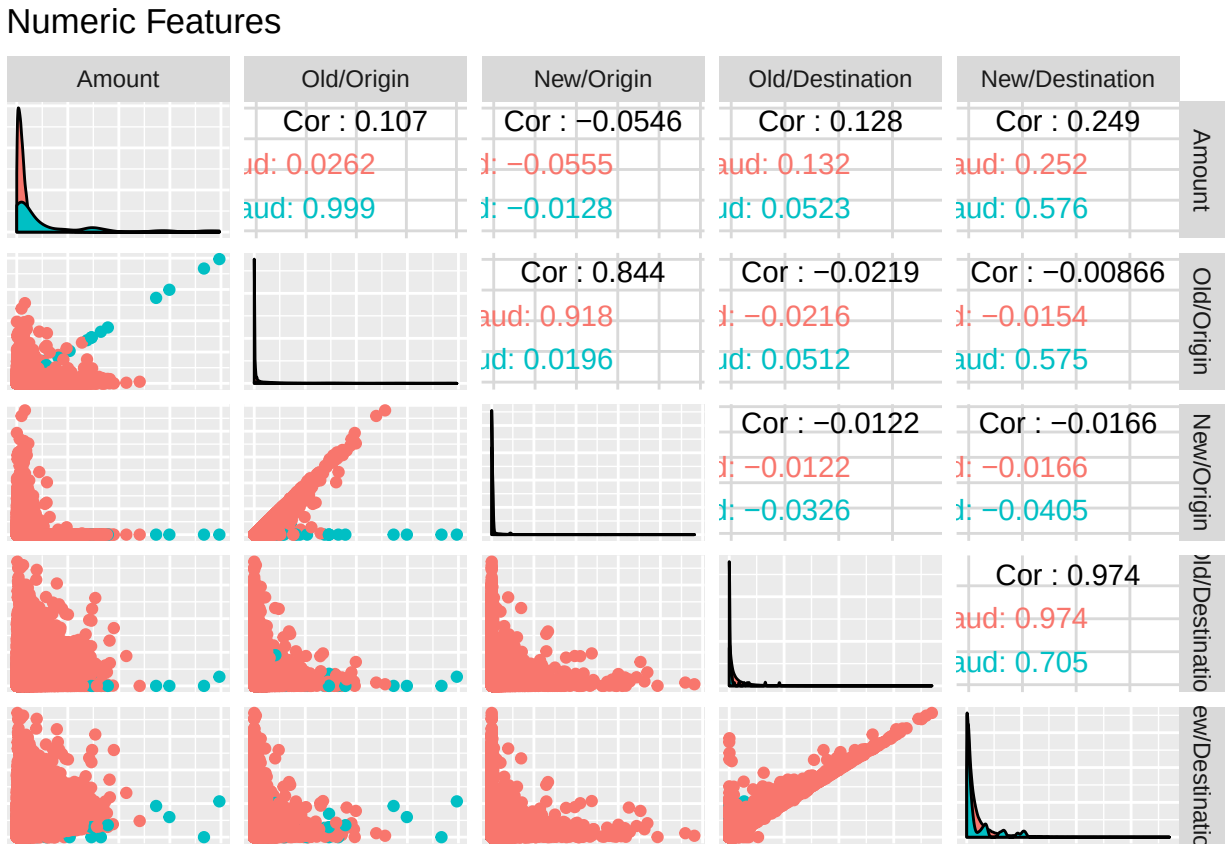

Old Balance of Destination
No Fraud
12.79%
87.21%
0.00
0.25
0.50
0.75
1.00
Percentage
Is Old Balance of Destination Zero?
a
Not_Zero
a
Zero
Fraud
61.5%
38.5%
0.00
0.25
0.50
0.75
1.00
Percentage
Is Old Balance of Destination Zero?
a
Not_Zero
a
Zero



Both in Old and in New Balance of Destination, fraud cases coincide with zero balances way more frequently than no_fraud cases do. This can be easily explained by the fact that fraudsters usually send money to empty dummy accounts.

4 Correlation Among Numeric Features

Before we start preprocessing, we will check for correlation between our numeric features with the help of a pairplot. Because the pairplot is very computationally expensive, we will choose 10% of the dataset for plotting.



Two very high correlations lie inside our numeric data. It appears that new and old balances (regarding both the origin and the destination) correlate. To decide on which columns to keep, we will utilize the findCorrelation function from caret package.

- Its first argument is going to be the **correlation table** of the numeric features of df.
- By choosing a **cutoff** (border of absolute correlation) of 0.8 means that for each correlation below -0.8 or above 0.8, we get a suggestion of a column to eliminate.
- names = TRUE will **assign a character vector** to our newly created variable, making it easy to use with the select() function.

Additionaly, we notice a mild concentration of fraud cases near zero values for some of the Balance Columns. This fortifies our assumption that zero balances might coincide with fraud.

```
## Compare row 5  and column  4 with corr  0.976
##   Means:  0.312 vs 0.231 so flagging column 5
## Compare row 2  and column  3 with corr  0.866
##   Means:  0.329 vs 0.157 so flagging column 2
## All correlations <= 0.8
```

```
## [1] "High Correlation Features to be Removed"
```

```
## [1] "New_Balance_of_Destination" "Old_Balance_of_Origin"
```

5 Preprocessing

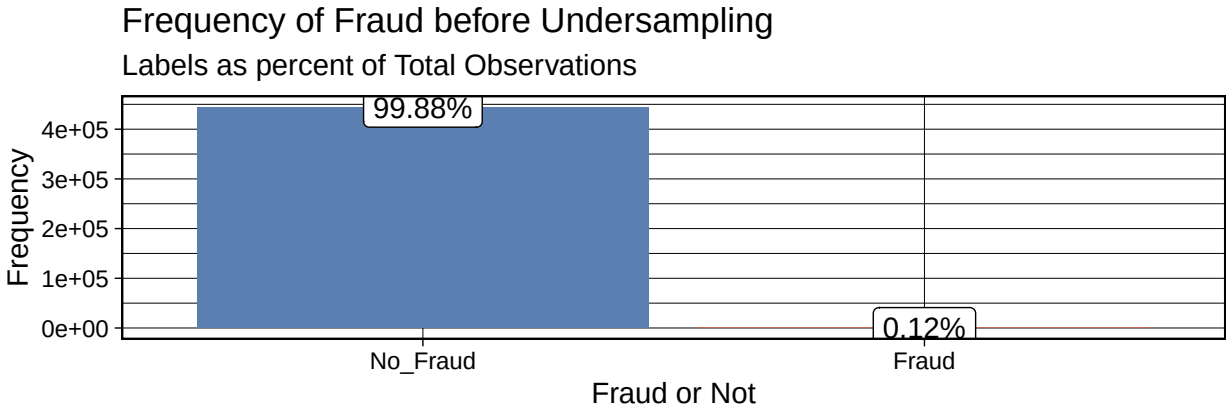
Throughout the preprocessing section, we will create a more balanced dataset and perform train/test split on both the balanced and the unbalanced data.

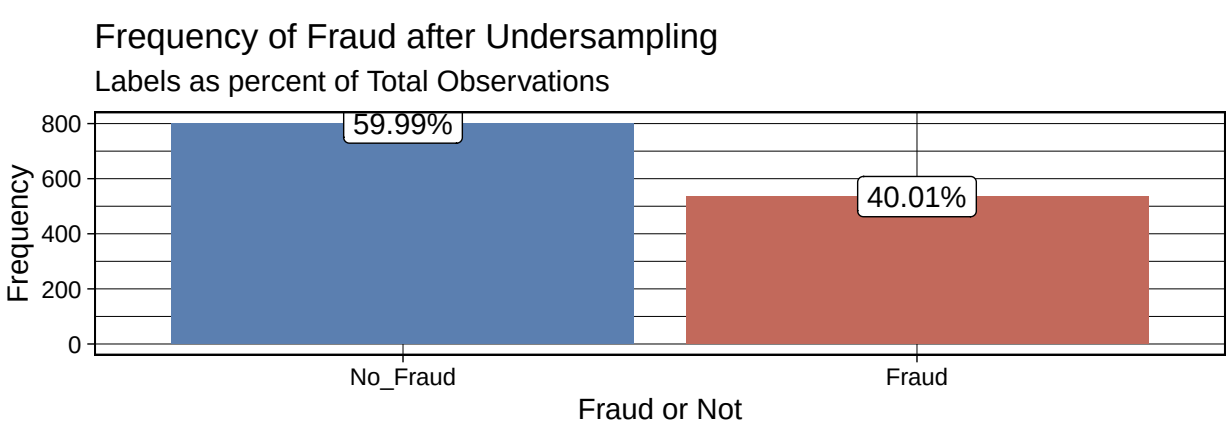
5.1 UnderSampling

Our goal is to create a more balance dataset.

In my opinion, dropping the majority class prevalence to 50% totally eliminates the informational value of this imbalance. This is why in this kernel we will create an 60%-40% target variable.

This can be achieved by sampling from the majority class 6/4ths of the rows of minority instances. In an oversimplified example where the minority instances equal 40, the number of sampled majority instances will be 60 (= 60/40 * 40), and the size of the new dataset totals 100.





5.2 Train/ Test Split

After we split our data into train and test set, we will create a preprocess routine based on the train data (test set is considered unseen). Our preprocess tranformation will include:

- centering , (ie subtracting by each column its own mean)
- scaling , (ie dividing by the column variance)

Then, we will use it to transform both sets.

The same process will be applied both on the undersampled as well as on the original dataset. Because we need to evaluate our algorithm based on real-life and not on synthetic, balanced data we will create train and test set of both categories.

```
## [1] "Original (Unbalanced) Train Set Structure : "
```

```
## 'data.frame':    333826 obs. of  11 variables:
##  $ step                          : Factor w/ 45 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ type                          : Factor w/ 2 levels "TRANSFER","CASH_OUT": 2 1 1 2 2 1 2 2 2 1 ...
```

```
##  $ amount                        : num  -0.789 -0.154 0.13 -0.622 -0.774 ...
##  $ New_Balance_of_Origin         : num  -0.159 -0.159 -0.159 -0.159 -0.159 ...
##  $ Old_Balance_of_Destination    : num  -0.543 -0.543 -0.549 -0.525 -0.31 ...
##  $ isFraud                       : Factor w/ 2 levels "No_Fraud","Fraud": 2 1 1 1 1 1 1 1 1 1 ...
##  $ isRound                       : Factor w/ 2 levels "float","round": 2 1 1 1 1 1 1 1 1 1 ...
##  $ isZero_Old_Balance_of_Or      : logi  FALSE FALSE FALSE FALSE TRUE FALSE ...
##  $ isZero_New_Balance_of_Or      : logi  TRUE TRUE TRUE TRUE TRUE FALSE ...
##  $ isZero_Old_Balance_of_Destina: logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  $ isZero_New_Balance_of_Destina: logi  TRUE TRUE FALSE FALSE FALSE FALSE ...
## NULL
```

```
## [1] "Balanced Train Set Structure : "
```

```
## 'data.frame':    1004 obs. of  11 variables:
##  $ step                          : Factor w/ 45 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ type                          : Factor w/ 2 levels "TRANSFER","CASH_OUT": 1 2 1 2 2 1 2 2 2 2 ...
##  $ amount                        : num  -0.428 -0.428 -0.426 -0.412 -0.106 ...
##  $ New_Balance_of_Origin         : num  -0.0963 -0.0963 -0.0963 -0.0963 -0.0963 ...
##  $ Old_Balance_of_Destination    : num  -0.42 -0.412 -0.42 -0.418 -0.42 ...
##  $ isFraud                       : Factor w/ 2 levels "No_Fraud","Fraud": 2 2 2 2 2 2 2 2 2 2 ...
##  $ isRound                       : Factor w/ 2 levels "float","round": 2 2 2 2 1 1 1 1 1 1 ...
##  $ isZero_Old_Balance_of_Or      : logi  FALSE FALSE FALSE FALSE TRUE FALSE ...
##  $ isZero_New_Balance_of_Or      : logi  TRUE TRUE TRUE TRUE TRUE TRUE ...
##  $ isZero_Old_Balance_of_Destina: logi  TRUE FALSE TRUE FALSE FALSE TRUE ...
##  $ isZero_New_Balance_of_Destina: logi  TRUE TRUE TRUE FALSE FALSE TRUE ...
## NULL
```

6 Modelling

The time has come to start modelling. Throughtout this section, we will compare performances for:

- train vs test set (both for tree and forest)
- balanced vs unbalanced set (both for tree and forest)
- and finally, Tree vs Forest (on unbalanced test set)

But first a quick glance on the cross-validation routine we are about to use.

6.1 Cross Validation

To find the optimal model parameters (both for the tree and the forest), we will use cross-validation to train it. That means each time (total times are the repeats parameter) we train our algorithm, we split our train data into partitions (the number parameter), use one of them, and then evaluate its success on predicting the rest of the partitions. In our case, we measure predictions by summarizing each class probabilities with the twoClassSummary. This function fits well as the choices of prediction (the target variable's unique values) are two, Fraud or No_Fraud.

```
#Setting up the training parameters
tr <- trainControl(method = "repeatedcv", #repeated cross validation
                           number = 9, #9 folds
                           repeats = 3, #3 times
                           classProbs = TRUE, #calculate probabilities for each class
                           summaryFunction = twoClassSummary) #use twoClassSummary to summarize probs
```

6.2 Classification Tree

Before we train our tree, we need to decide on which parameters we want to optimize. Parameters are as follows:

metric - How we measure success of our algorithm. In every classification problem, we want as few False predictions as possible. But false predictions can be either False Positives (cases where we falsely predicted True) or False Negatives (where we falsely predicted False). These two metrics are inversely proportional. To make this statement more straightforward, imagine an algorithm that always predicts True (False). It will yield 0 False Negatives (Positives) and many False Positives (Negatives). Depending on the target, one can aim to minimize one of these errors. A useful tool to describe the balance between these errors is the ROC (Receiver Operating Characteristics) curve. To construct it, we use the False Positive Rate (FPR for short) for the X axis and the True Positive Rate (TPR for short) for the Y axis. TPR, also known as Recall or Sensitivity is calculated by dividing the True Positives by (True Positives + False Negatives). FPR, or (1 - Specificity), on the other hand, is False Positives divided by (False Positives + True Negatives). In case we predict randomly, our ROC curve will be a straight 45 degrees line starting from (0,0) because TPR equals FPR. Any improvement from that state will force the straight line to become a curve facing to the upper left corner (where TPR > FPR). As a result, the more area we have under the curve, the better is our algorithm in predicting the classes correctly. An ideal curve would occupy 100% of the available area. That's why we will use the Area Under the Curve to measure success of classification.

tuneLength - The number of different default values used to optimize the main parameter of the model. In our case, that's the Complexity Parameter (CP). This parameter specifies how the cost of a tree C(T) is penalized by the number of terminal nodes. Because it penalizes, small (large) values of CP lead to large (small) trees with many (few) nodes that overfit (underfit). tuneLength defines how many different CPs we will use. For each CP we calculate our metric and finally pick the CP that yields the optimal metric value (area under the curve in our case). An alternative to

split - Criterion that shows which feature to split on at each step in building the tree. An alternative to it is Gini Impurity. Information Split it the most commonly used of the two.

```
#Seed for reproducibility
set.seed(123)

#Model Training - Regression Tree
start_time <- Sys.time()

rpart_model_b <- train(isFraud ~ ., #training formula --
                                    #we want to predict isFraud by using all predictors
                       data = train_set_b, #our balanced dataset
                       method = "rpart", # library containing the tree algorithm
                       tuneLength = 10, # number of different default values used to
                                        #optimize complexity parameter
                       metric = "ROC", # estimate success by the area under the ROC curve
                       trControl = tr, # use cross validation as defined above
                       parms=list(split='information'))

end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 5.021447 secs
```

6.2.1 Results on Balanced Data

Now that we trained our tree, let's see how it performs.

6.2.1.1 Train Set Results

First, the confusion matrix of the train set results

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No_Fraud Fraud
##   No_Fraud      588    14
##   Fraud         102   300
##
##                Accuracy : 0.8845
##                  95% CI : (0.8631, 0.9036)
##     No Information Rate : 0.6873
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7503
##
##  Mcnemar's Test P-Value : 6.597e-16
##
##             Sensitivity : 0.8522
##             Specificity : 0.9554
##          Pos Pred Value : 0.9767
##          Neg Pred Value : 0.7463
##              Prevalence : 0.6873
##          Detection Rate : 0.5857
##    Detection Prevalence : 0.5996
##       Balanced Accuracy : 0.9038
##
##        'Positive' Class : No_Fraud
##
```

6.2.1.2 Test Set Results

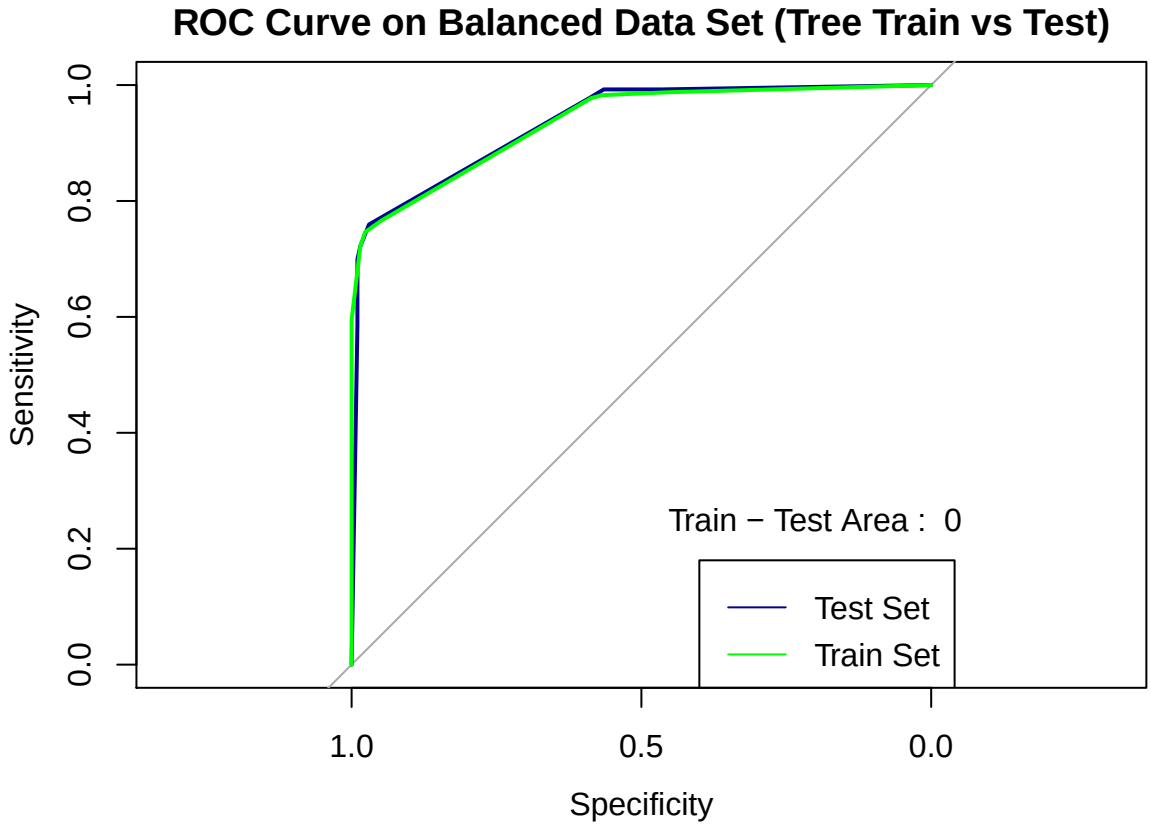
Then, the confusion matrix of the test set results.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No_Fraud Fraud
##   No_Fraud      197     3
##   Fraud          37    96
##
##                Accuracy : 0.8799
##                  95% CI : (0.84, 0.9128)
##     No Information Rate : 0.7027
##     P-Value [Acc > NIR] : 1.329e-14
##
##                   Kappa : 0.7384
##
##  Mcnemar's Test P-Value : 1.811e-07
##
##             Sensitivity : 0.8419
##             Specificity : 0.9697
##          Pos Pred Value : 0.9850
##          Neg Pred Value : 0.7218
##              Prevalence : 0.7027
```

```
##          Detection Rate : 0.5916
##    Detection Prevalence : 0.6006
##       Balanced Accuracy : 0.9058
##
##        'Positive' Class : No_Fraud
##
```

6.2.1.3 ROC Curve - Comparing Train and Test Set Results

To visualize performance on balanced data, we will plot two ROC curves, one for the train and one for the test set. This plot will informa us not only about the raw performance, but also about the degree of overfitting. If the two lines are way apart from each other, our model overfits, meaning that it's good only in the specific data that has seen and cannot generalize.



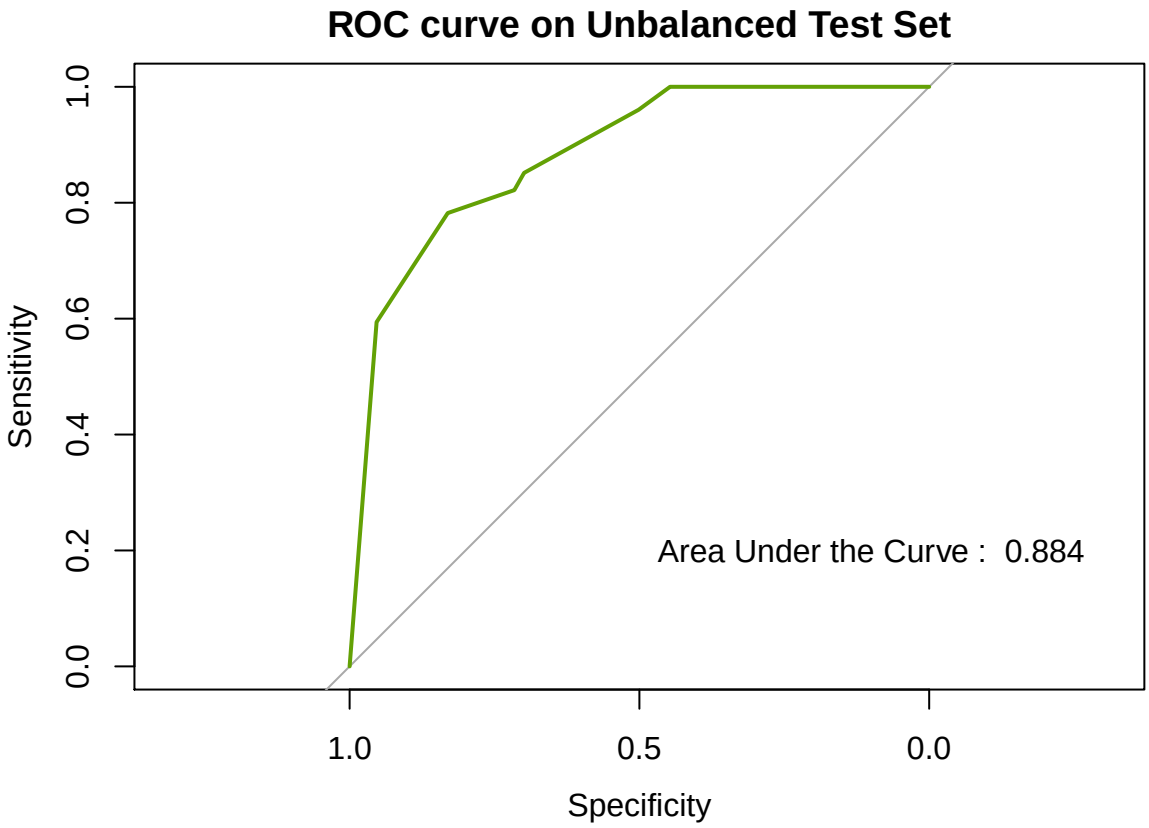
Almost identical! It seems that our tree did well in avoiding overfitting.

6.2.2 Results on Unbalanced Data

To evaluate our algorithm in real life circumstances, we will pick a random sample (slightly smaller for ease of computations) from the original test set and try to predict on it.

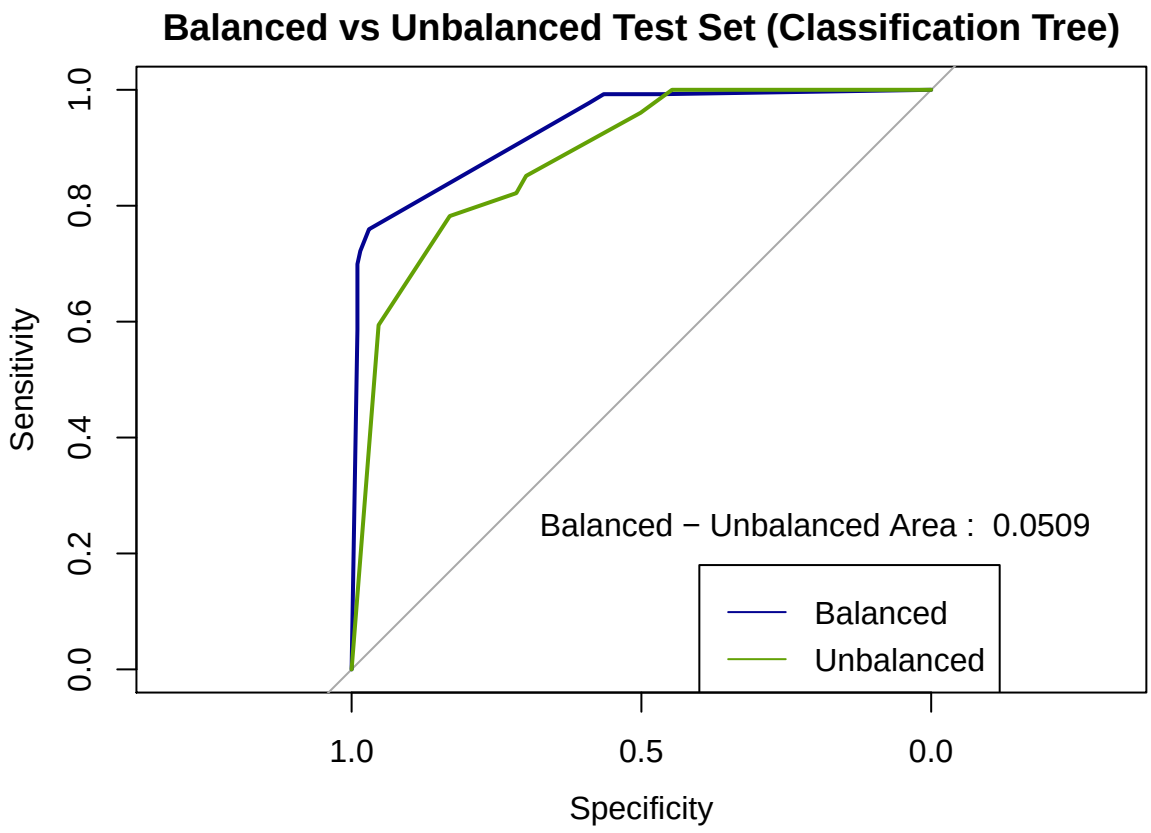
```
#Pick a random sample from the unbalanced test set
random_sample <- test_set %>%
                 sample_n(nrow(test_set)/1.3) #size = 75% of the test set rows
```

And the respective ROC curve:



6.2.3 Comparing Performance on Balanced vs Unbalanced Data

Now time for the ROC curves that represent test set results on Balanced vs the Unbalanced Dataset.



A slight, yet existing, difference of `0.0509` for the tree.

6.3 Random Forest

Random forests are collections of trees. Instead of relying on a tree to predict, we build several and return the mode (most frequent class) of all predictions. The greatest advantage over the tree is that we avoid overfitting. On the other hand, it is much more computationally expensive.

As with the tree we will use the same cross-validation parameters and measure success with ROC curve.

This time, instead of tuneLength that plugs in default values for the model hyperparameters, we will use a TuneGrid where we have these params preset. These hyperparameters include:

- *mtry* - Number of predictors included in search for a node split. Everytime When forming each split, instead of considering all predictors, a different random set of variables is selected within which the best split point is chosen. This feature mostly contributes to speed of algorithm. We will try 40% and 60% of the predictors.
- *ntree* - Number of trees to include in our forest.

```
## [1] "Tune Grid:"
```

```
##   .mtry .ntree
## 1     6     70
## 2     4     70
## 3     6    100
## 4     4    100
## 5     6    130
## 6     4    130
```

6.3.1 Training

As we did with the Tree, we will train our forest using the balanced dataset. Then, we will evaluate its success on the same random sample we generated above.

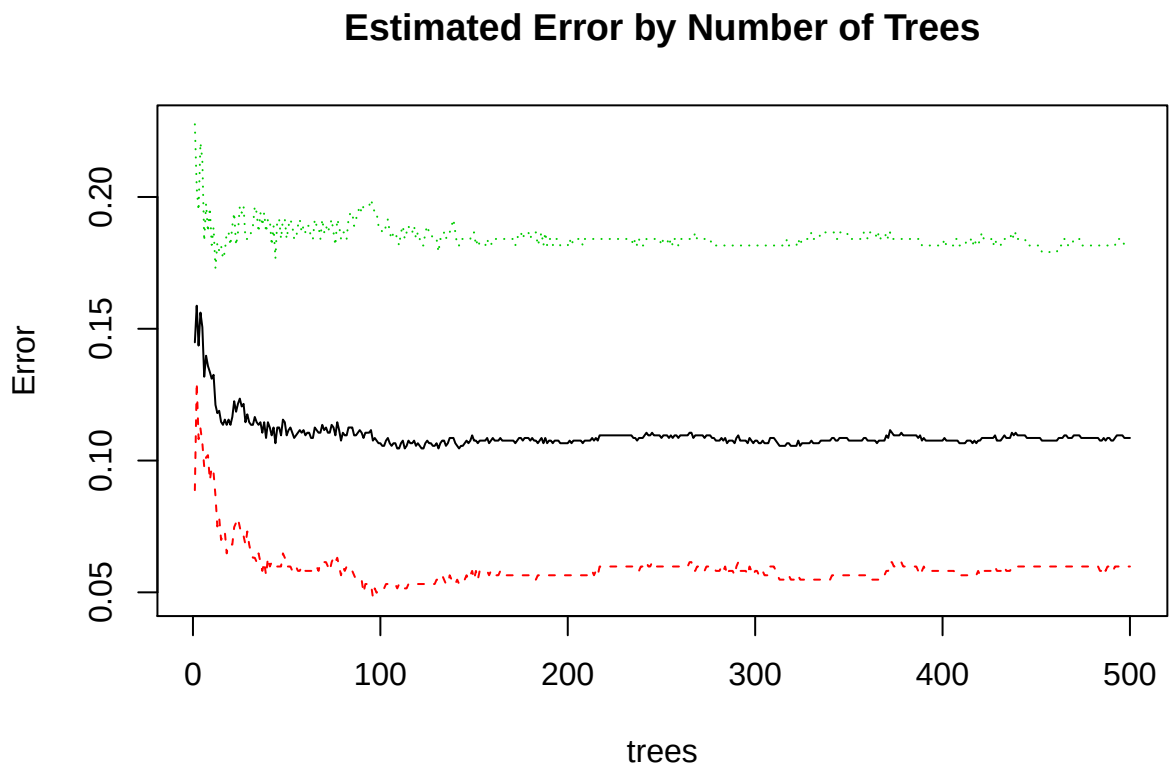
```
#Seed for reproducibility
set.seed(123)
#Model Training - Random Forest
start_time <- Sys.time()
rand_for_b <- train(isFraud ~ .,
                    data = train_set_b, #train on the balanced set
                    method="rf", # use the random forest package
                    metric = "ROC", # elect best model by ROC curve
                    TuneGrid = tune_grid, # use custom grid for hyperparameters
                    trControl=tr) # #use cross validation

end_time <- Sys.time()
end_time - start_time
```

```
## Time difference of 2.145156 mins
```

6.3.1.1 Error by Number of Trees

Now that we trained our random forest, let's look at the estimated error per number of trees.

The black line shows the Out-Of-Bag Error, while the coloured lines show misclassification errors (Fraud identified as No_Frauds in green, No_Frauds identified as Frauds in red).

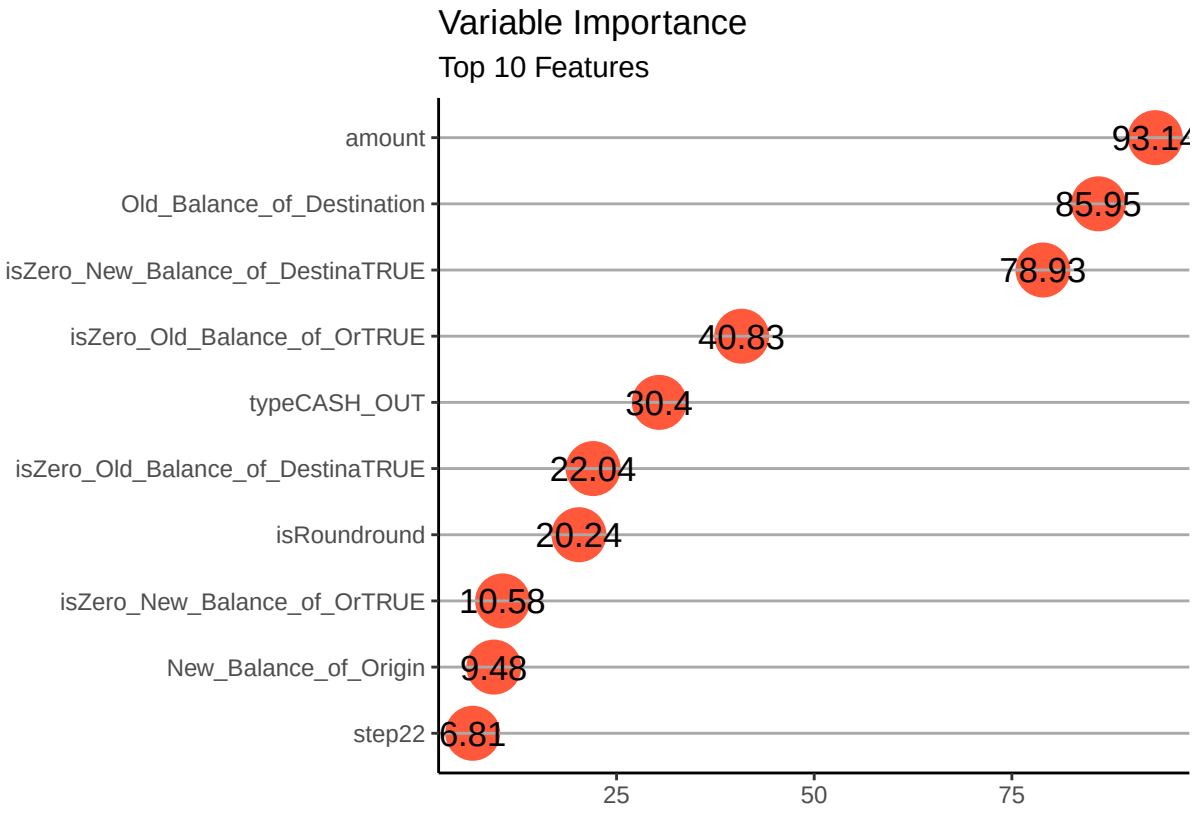


Seems like all lines reach a plateau at around 70 - 100 trees. Luckily for us, we included such values in the

our tune grid.

6.3.1.2 Variable Importance

Now that we've trained our random forest we can check out the variable importance to confirm whether the features we created contain any value.



As we assumed above, engineered features (isInteger, isZero) seem to play an important role in predicting the outcome.

6.3.2 Train vs Test Set (Balanced Dataset)

As we did with the tree, we will take a look at the train vs test set performance:

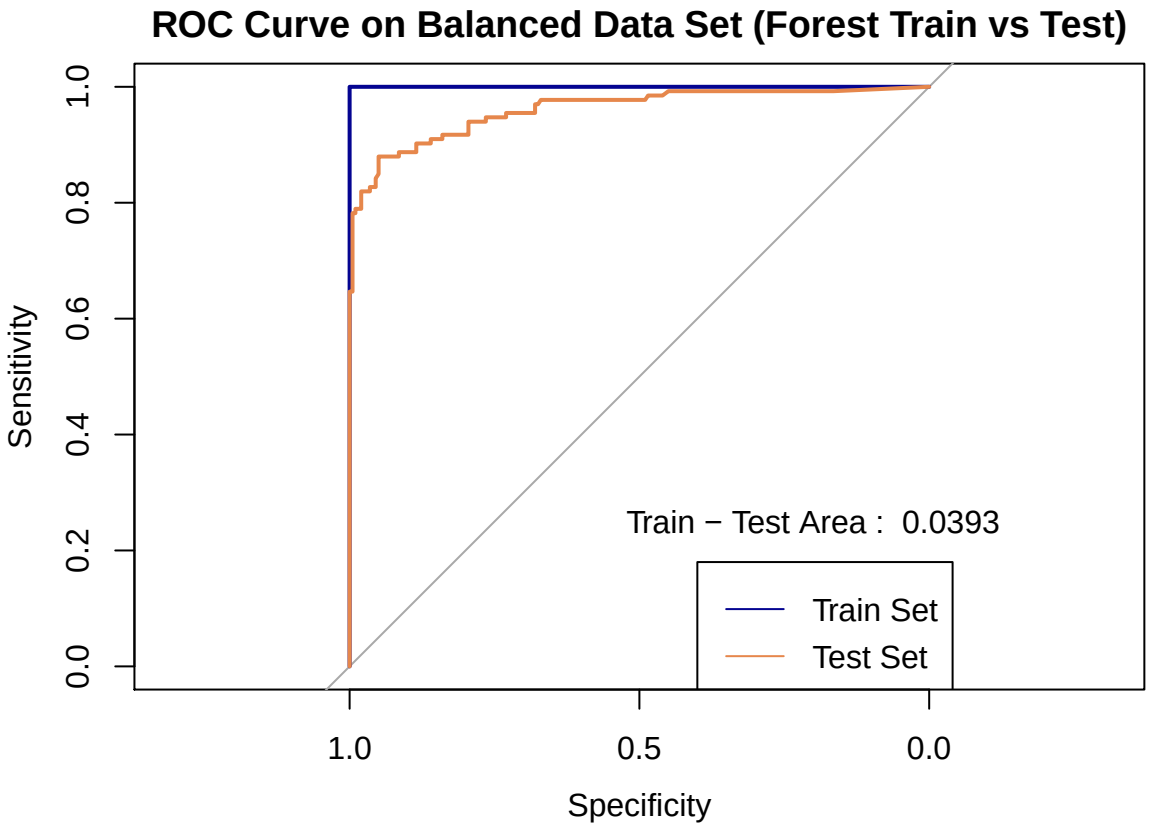
```
## [1] "Train Set Confusion Matrix (Balanced Data) : "
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No_Fraud Fraud
##   No_Fraud      602     0
##   Fraud           0   402
##
##                Accuracy : 1
##                  95% CI : (0.9963, 1)
##     No Information Rate : 0.5996
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                  Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5996
##          Detection Rate : 0.5996
##    Detection Prevalence : 0.5996
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : No_Fraud
##
```

```
## [1] "Test Set Confusion Matrix (Balanced Data) : "
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No_Fraud Fraud
##   No_Fraud      190    10
##   Fraud          20   113
##
##                Accuracy : 0.9099
##                  95% CI : (0.8739, 0.9384)
##     No Information Rate : 0.6306
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8098
##
##  Mcnemar's Test P-Value : 0.1003
##
##             Sensitivity : 0.9048
##             Specificity : 0.9187
##          Pos Pred Value : 0.9500
##          Neg Pred Value : 0.8496
##              Prevalence : 0.6306
##          Detection Rate : 0.5706
##    Detection Prevalence : 0.6006
##       Balanced Accuracy : 0.9117
##
##        'Positive' Class : No_Fraud
##
```



Looks like the forest, unlike the tree, did not avoid overfitting. Especially the line representing train set results looks suspiciously perfect.

6.3.3 Balanced vs Unbalanced

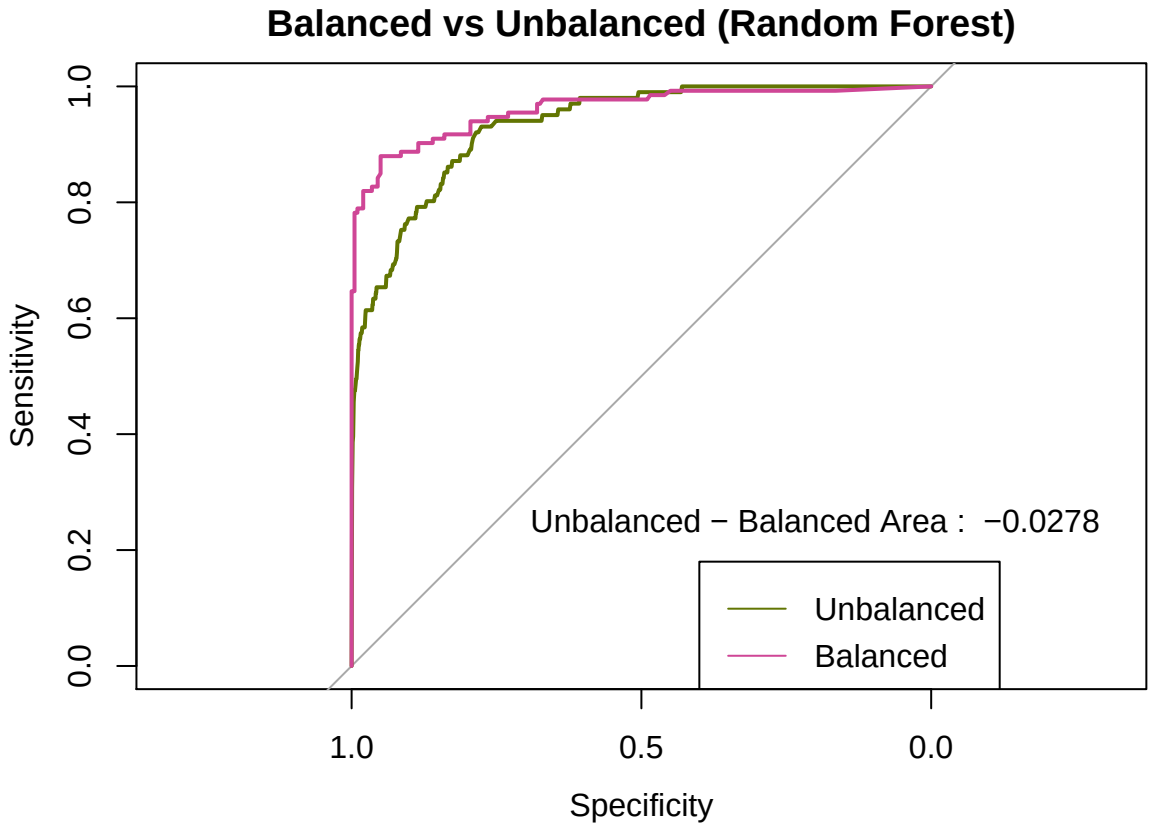
Before we compare ROC curves, let's throw a glance at the confusion matrix we obtain by predicting on the random sample.

```
## [1] "Random Sample Confusion Matrix (Unbalanced Data) : "
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No_Fraud Fraud
##   No_Fraud    56359 29135
##   Fraud           5    96
##
##                Accuracy : 0.6596
##                  95% CI : (0.6564, 0.6627)
##     No Information Rate : 0.6585
##     P-Value [Acc > NIR] : 0.2572
##
##                   Kappa : 0.0042
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.999911
```

```
##             Specificity : 0.003284
##          Pos Pred Value : 0.659216
##          Neg Pred Value : 0.950495
##              Prevalence : 0.658496
##          Detection Rate : 0.658438
##    Detection Prevalence : 0.998820
##       Balanced Accuracy : 0.501598
##
##        'Positive' Class : No_Fraud
##
```

We observe an extremely high Sensitivity and extremely low Specificity. Seems like the forest almost always predicts a no fraud. Though it will almost never misclassify an innocent transaction as fraud, it is very weak in detecting actual frauds as such. The opposite result (seeing frauds everywhere) could be partly explained by the undersampling. Ultimately, the model does not appear very useful for a financial institution looking to minimize chargebacks.



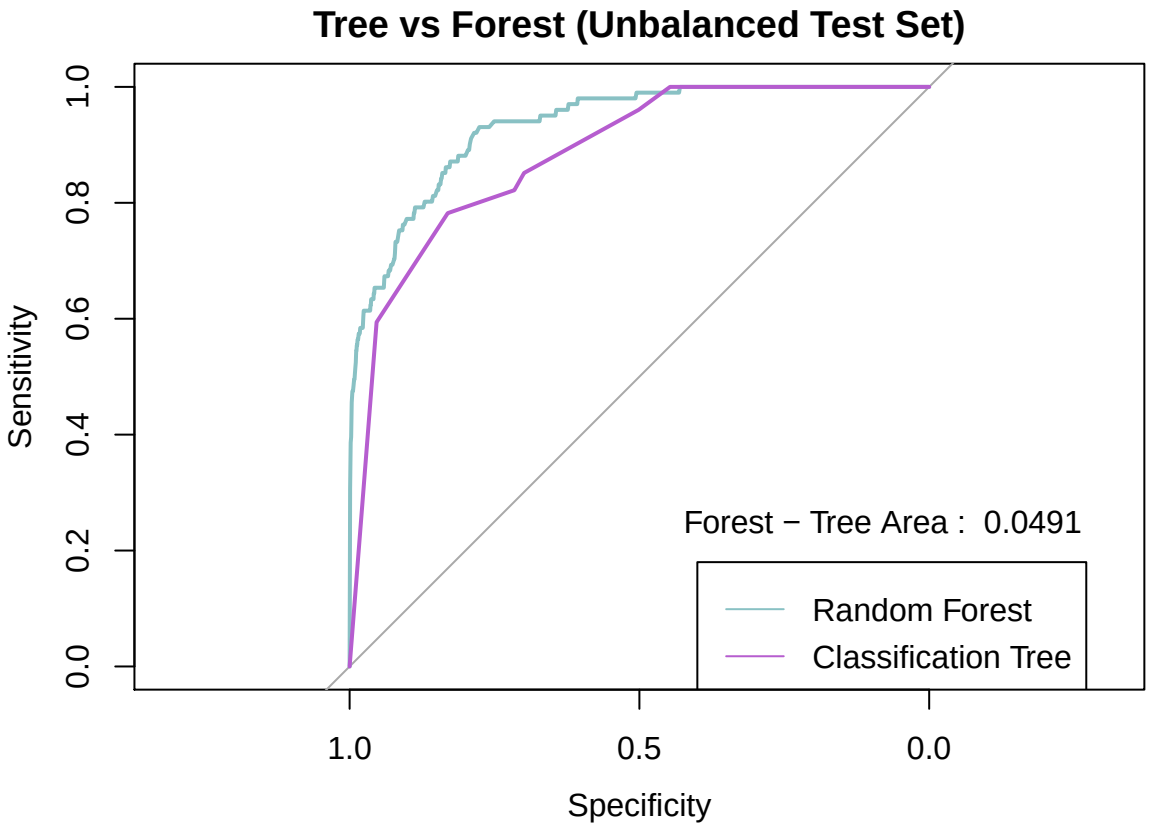
Oddly enough, the ROC curves are very close to each other, showing that the forest did not deal with great problems when it faced the random sample.

7 Tree vs Forest

The time has come to compare performance between the two algorithms. For that, we will employ their ROC curves obtained by the unbalanced test sets.

7.1 Unbalanced Test Set ROC Curve

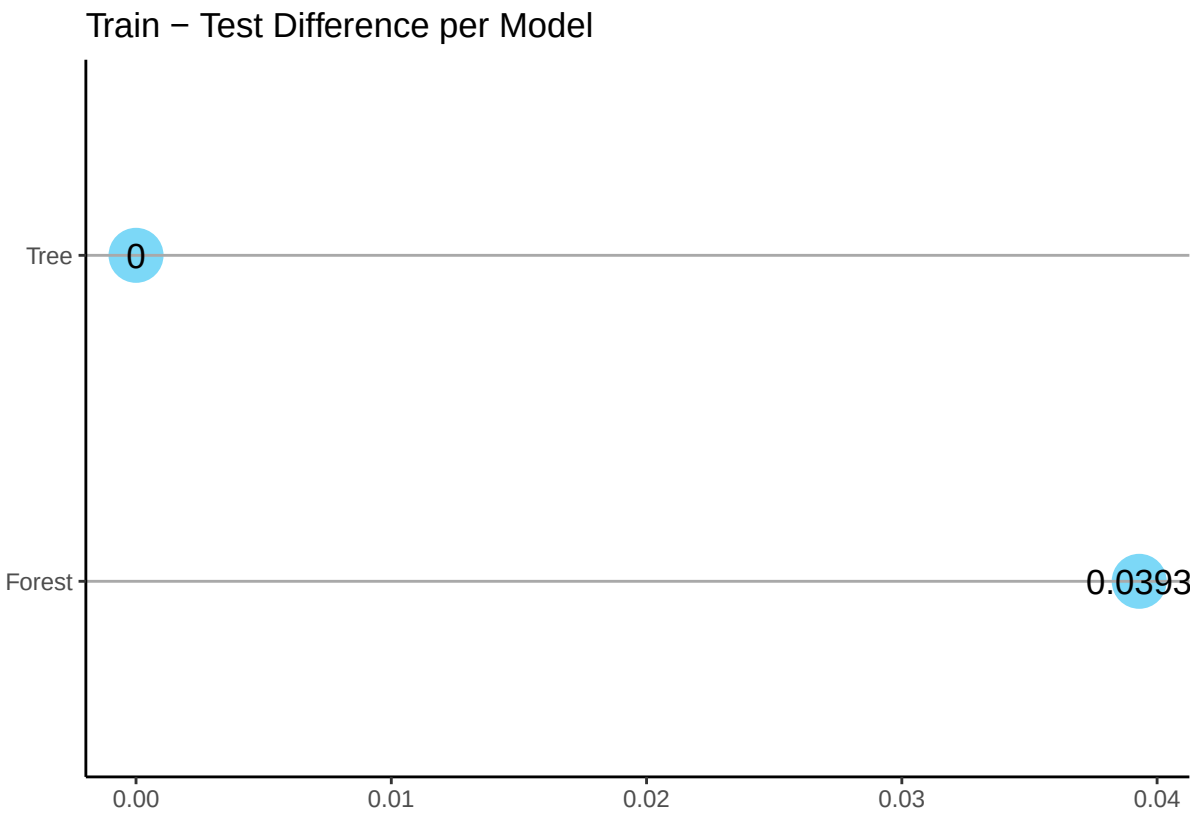
Of all the different ROC curves we obtained, the one closest to real life is the unbalanced (raw data) unseen test set.



It becomes obvious that, at least for the AUC metric, **the Forest outperforms the Tree**. However, that's not the only criterion we would consider.

7.2 Train - Test Overfit

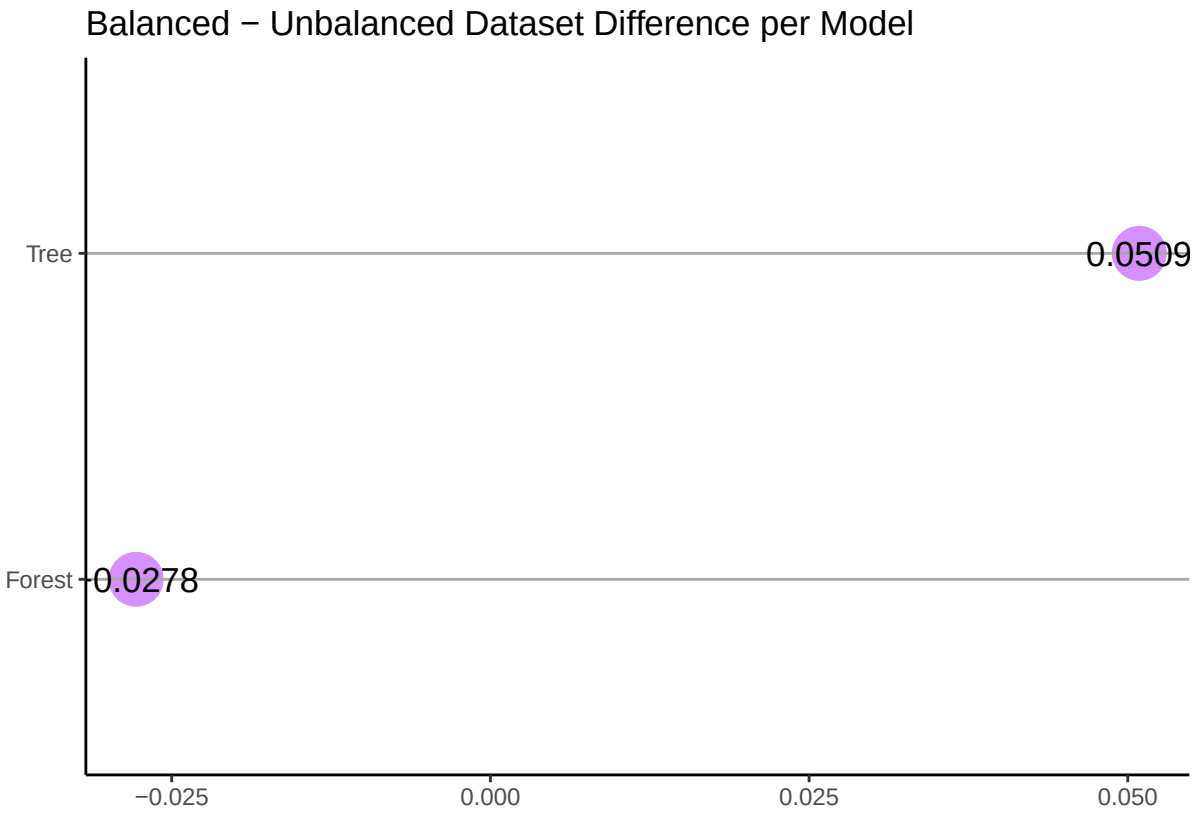
Another interesting aspect of model performance is its power to learn and generalize. The forest did not do well on that. Let's compare it with the tree.



Apparently, the tree is by far better in overfitting between train and test sets.

7.3 Balanced - Unbalanced Difference

We will proceed accordingly with the Balanced - Unbalanced diff.




```
r ifelse(tree_unb_perf>for_unb_perf , yes = "This time won the forest", no = Another
victory for the tree. )
```

8 Conclusion

Both classification trees and random forests can become efficient predictors of fraud, provided that they get their hyperparameters tuned and the underlying data gets engineered.

By this kernel, ROC curves seem to favor the random forest against the classification tree. Nevertheless, that alone is not a safe criterion. The tree showed strengths in transitioning between train-test and balanced-unbalanced data. The parameters to consider in order to elect a winner are almost infinite. After all, it also depends on preferences (in our example, whether minimize chargeback or false alarms).

A crucial limitation of this kernel is the restricted memory and computing power an average personal computer provides. That alone has forced us to reduce available data (1 Million rows out of 10) and, more importantly, kept us away from training many (and complex) algorithms to have a wider perspective on potential solutions.

Interesting extensions of this kernel include:

- comparing models trained with datasets of different **rates of undersampling** (mainly undersampled vs non-undersampled training),
- experiment with **more metrics**,
- test **more sophisticated techniques** like C5.0, Gradient Boosting or Support Vector Machines
- create **ensemble models** (ie combinations of different machine learning algorithm)

9 Appendix (Code)

Because most of the code chunks are hidden in the report, here is the raw code.

```
# Data Manipulation
if(!require(tidyverse)) install.packages("tidyverse",
                                repos = "http://cran.us.r-project.org")
# showing multiple plots
if(!require(gridExtra)) install.packages("gridExtra",
                                repos = "http://cran.us.r-project.org")
# Assisting gridExtra
if(!require(grid)) install.packages("grid",
                                repos = "http://cran.us.r-project.org")
# Model training
if(!require(caret)) install.packages("caret",
                                repos = "http://cran.us.r-project.org")
# Pairplot
if(!require(GGally)) install.packages("GGally",
                                repos = "http://cran.us.r-project.org")
# ROC Curve
if(!require(pROC)) install.packages("pROC",
                                repos = "http://cran.us.r-project.org")
# ROC Curve
if(!require(ROCR)) install.packages("ROCR",
                                repos = "http://cran.us.r-project.org")
#Refactoring
if(!require(radiant.data)) install.packages("radiant.data",
                                repos = "http://cran.us.r-project.org")



# Relative path - assumes data and .rmd file are in the same directory
# Github Repository : https://github.com/voulkon/Fraud_Detection
zipfile <- paste0(getwd(),"/PaySim1M.zip") %>%
  str_replace_all( "/", "\\\\" )

# if data is not the same dir
# enable the line below and point it to the zip file location
#zipfile <- choose.files(caption = "Please choose zipfile location")

#unzip and read the csv contained in it
df <- unzip(zipfile = zipfile) %>%
        read.csv(stringsAsFactors = FALSE)#, nrow = 8000)


#A glance at the structure
df %>% str()

#Bar Plot with count of fraud vs no_fraud cases
df %>% ggplot(aes(x = isFraud, y = (..count..))) +
    geom_bar(fill =  c( "#5b7fb0", "#c2695b" ) , stat = "count")+
    geom_label(stat='count',aes(   label=  paste0(  round(   ((..count..)/sum(..count..)) ,4)*100 ,  "%
    labs(x = "Fraud or Not", y = "Frequency", title = "Frequency of Fraud",
         subtitle = "Labels as percent of Total Observations")+
```

```
    theme_linedraw()


#A quick glance to the number of dinstinct values of each character column
print("Number of Distinct Values in Character Vectors")
df[ , sapply(df,is.character)] %>% sapply(n_distinct) %>% print()



print("Distinct Prefixes in nameOrig and nameDest columns : ")

df %>% mutate(origin_name_prefix = str_sub(nameOrig,1,1), #create columns containing prefix
              dest_name_prefix = str_sub(nameDest, 1, 1)) %>%
        select( c( origin_name_prefix,dest_name_prefix ) ) %>% #isolate them
        table() %>% #count of distinct occurences
        print() #show

df <- df %>% mutate(dest_name_prefix = str_sub(nameDest, 1, 1)) %>% #create columns containing prefix
                    select(-c('nameOrig','nameDest','isFlaggedFraud')) %>% #discard columns with old na
                    mutate_if(is.integer, as.factor) %>% #turn integer features into factors
                    mutate_if(is.character,as.factor) %>% #turn character features into factors
                    rename("Old_Balance_of_Origin" = "oldbalanceOrg" , #rename into a readable format
                           "New_Balance_of_Origin" = "newbalanceOrig",
                           "Old_Balance_of_Destination" = "oldbalanceDest",
                           "New_Balance_of_Destination" = "newbalanceDest")

levels(df$isFraud) <- c("No_Fraud", "Fraud" ) #rename into a readable format

#Define colors associated with each group
dest_name_colors <- c("C" = "#6A4491", "M" = "#CC6600")

#Count of prefix in fraud cases

p3 <- df %>% filter(isFraud == "Fraud") %>%
    ggplot( aes( x = isFraud, fill = dest_name_prefix ) ) +
    geom_bar()+
    scale_fill_manual(values=dest_name_colors) +
    labs(title = "Destination Name Prefix",subtitle = "Fraud Cases", x = "", y = "Frequency" )+
    theme_classic()

p4 <-df %>% filter(isFraud == "No_Fraud") %>%
    ggplot( aes( x = isFraud, fill = dest_name_prefix ) ) +
    geom_bar()+
    scale_fill_manual(values=dest_name_colors) +
    labs(title = "Destination Name Prefix", subtitle =  "No Fraud Cases", x = "", y = "" )+
    theme_classic()

grid.arrange(p3,p4,ncol = 2)
rm(p3,p4)

#Create a color palette
type_colors <- c(CASH_IN = "#78858f", CASH_OUT = "#CC6600",
                 DEBIT ="#6A4491", PAYMENT = "#FFC233", TRANSFER = "#788f78")
```

```r
#Count of type in fraud cases
p1 <- df %>% filter(isFraud == "Fraud") %>%
    ggplot( aes( x = isFraud, fill = type ) ) +
    geom_bar()+
    theme(axis.text.x = element_blank())+
    scale_fill_manual(values=type_colors) +
    labs(title = "Transaction Type", subtitle =  "Fraud Cases", x = "", y = "" ) +
    coord_flip() + theme_linedraw()+
    theme(axis.text.y = element_blank())


p2 <-df %>% filter(isFraud == "No_Fraud") %>%
    ggplot( aes( x = isFraud, fill = type ) ) +
    geom_bar()+
    scale_fill_manual(values=type_colors) +
    labs(subtitle =  "No Fraud Cases", x = "", y = "Frequency" ) +
    coord_flip() + theme_linedraw() +
    theme(axis.text.y = element_blank())

grid.arrange(p1,p2,nrow = 2)

#Store to number of rows before reduction
rows_before <- dim(df)[1]

df <- df %>% filter( (!(type %in% c("CASH_IN", "DEBIT", "PAYMENT") )) , #remove transaction types
                     dest_name_prefix == "C") %>% #remove rows with M prefixes
             select(-dest_name_prefix) #drop the prefixes column

#Refactor the type column
df$type <- (refactor( df$type, unique(df$type) ))

rows_after <- dim(df)[1]
#Difference in rows
diff <- rows_before - rows_after
#As percentage
diff_perc <- round(diff/rows_before * 100, 2)

sprintf( "%i rows removed, equivalent to %s%%  of the data", diff, diff_perc ) %>% print()


p11 <-  df %>% filter(isFraud == "Fraud") %>%
    ggplot( aes( x = isFraud, fill = type ) ) +
    geom_bar()+
    scale_fill_manual(values=type_colors) +
    labs(title = "Transaction Type", subtitle =  "Fraud Cases", x = "", y = "" ) +
    coord_flip() + theme_linedraw()+
    theme(axis.text.y = element_blank())

p12 <-df %>% filter(isFraud == "No_Fraud") %>%
    ggplot( aes( x = isFraud, fill = type ) ) +
    geom_bar()+
    scale_fill_manual(values=type_colors) +
    labs(subtitle =  "No Fraud Cases", x = "", y = "Frequency" ) +
```

```
    coord_flip() + theme_linedraw()+
    theme(axis.text.y = element_blank())


grid.arrange(p1,p2,top = "Before Data Reduction")
grid.rect(gp = gpar(lwd = 3, col = "black", fill = NA))
grid.arrange(p11,p12,top = "After Data Reduction")
grid.rect(gp = gpar(lwd = 3, col = "black", fill = NA))
rm(p1,p2,p11,p12)

temp <- seq(0, max(as.numeric(df$step)) , 5 )

df %>% sample_n(floor(nrow(df)*.03)) %>%
    ggplot()+
    geom_jitter(aes( x = as.numeric(step), y = log(amount+1,10)), alpha = .9, col = "#2ff7b5")+
    facet_grid(~isFraud)+
    scale_x_continuous(breaks = temp )+
    labs(x = "Hour", y = "Amount (in a log10 scale)",
         title = "Transaction Amounts by hour", subtitle = "")+
    theme_minimal()+
    theme(panel.border = element_rect(colour = "black", fill=NA, size=3) )

df %>%
    ggplot(aes(x = "isFraud", y = log(amount,10) , fill = isFraud)) +
    geom_violin() + coord_flip() +
    scale_fill_manual(values=c("#40448f", "#7d1f6d") , name="Mean", labels=c( "No Fraud","Fraud"))+
    labs(x = "", y = "Amount (in a log 10 scale)",
         title = "Amount of Transactions", subtitle = "Fraud vs Not Fraud Cases" ) +
    stat_summary(fun.y = "mean",geom = "point", size = 2, col =c("#40448f", "#7d1f6d"))+
    theme_classic()

which(df %>% sapply(is.numeric))[-1] %>% names() -> labs # store names for use in plot labels

p5 <- df %>%
    ggplot(aes(x = isFraud, y = log(Old_Balance_of_Origin+1,10) , fill = isFraud)) +
    geom_boxplot(alpha = .3) + coord_flip() +
    scale_fill_manual(values=c("navy", "darkred"))+
    labs(x = "", y = paste(labs[1], "(in a log 10 scale)"),
         title = paste("Amount of", labs[1] ), subtitle = "Fraud vs Not Fraud Cases"  ) +
    theme_linedraw()


p6 <- df %>%
    ggplot(aes(x = isFraud, y = log(New_Balance_of_Origin+1,10) , fill = isFraud)) +
    geom_boxplot(alpha = .3) + coord_flip() +
    scale_fill_manual(values=c("navy", "darkred"))+
    labs(x = "", y = paste(labs[2], "(in a log 10 scale)"),
         title = paste("Amount of", labs[2] ) ) +
    theme_linedraw()

grid.arrange(p5,p6)
rm(p5,p6)
```

```r
p7 <- df %>%  filter(Old_Balance_of_Origin != 0) %>%
    ggplot(aes(x = isFraud, y = log(Old_Balance_of_Origin+1,10) , fill = isFraud)) +
    geom_boxplot(alpha = .3) + coord_flip() +
    scale_fill_manual(values=c("navy", "darkred"))+
    labs(x = "", y = paste(labs[1], "(in a log 10 scale)"),
         title = paste("Amount of", labs[1] ), subtitle = "Fraud vs Not Fraud Cases (zero values omitte
    theme_linedraw()


p8 <- df %>% filter(New_Balance_of_Origin != 0) %>%
    ggplot(aes(x = isFraud, y = log(New_Balance_of_Origin+1,10) , fill = isFraud)) +
    geom_boxplot(alpha = .3) + coord_flip() +
    scale_fill_manual(values=c("navy", "darkred"))+
    labs(x = "", y = paste(labs[2], "(log 10 scale)"),
         title = paste("Amount of", labs[2] )  )+
    theme_linedraw()

grid.arrange(p7,p8)
rm(p7,p8)

p9 <- df %>% filter(Old_Balance_of_Destination != 0) %>%
    ggplot(aes(x = isFraud, y = log(Old_Balance_of_Destination+1,10) , fill = isFraud)) +
    geom_boxplot(alpha = .3) + coord_flip() +
    scale_fill_manual(values=c("navy", "darkred"))+
    labs(x = "", y = paste(labs[3], "(in a log 10 scale)"),
         title = paste("Amount of", labs[3] ), subtitle = "Fraud vs Not Fraud Cases (zero values omitte
    theme_linedraw()

p10 <- df %>% filter(New_Balance_of_Destination != 0) %>%
    ggplot(aes(x = isFraud, y = log(New_Balance_of_Destination+1,10) , fill = isFraud)) +
    geom_boxplot(alpha = .3) + coord_flip() +
    scale_fill_manual(values=c("navy", "darkred"))+
    labs(x = "", y = paste(labs[4], "(in a log 10 scale)"),
         title = paste("Amount of", labs[4] ) )+
    theme_linedraw()

grid.arrange(p9,p10)
rm(p9,p10)

#Create a new isRound column
df <- mutate(df, isRound =
               as.factor(
                 ifelse( test = (df$amount - as.integer(df$amount)) == 0 ,
                         yes = 'round', no = 'float' )))

p11 <- df %>% filter( isFraud == "Fraud" ) %>% ggplot() +
    geom_bar(aes( x = isFraud , fill = isRound ) ) +
    scale_fill_manual(values = c( "#849468", "#ded718" )) +
    labs( title = "Round amounts of transactions" , x= "", y = "Count", subtitle = "Fraud Cases" )+
    theme_linedraw()

p12 <- df %>% filter( isFraud == "No_Fraud" ) %>% ggplot() +
    geom_bar(aes( x = isFraud , fill = isRound ) ) +
```

```
    scale_fill_manual(values = c( "#849468", "#f7f300" ))+
    labs( x= "", y = "Count", subtitle = "No_Fraud Cases" ) +
    theme_linedraw()

grid.arrange(p11,p12)
rm(p11,p12)

iszero <- df %>% select(-amount) %>% #exclude the amount that cannot be 0
    select_if(is.numeric) %>% #select the remaining numerics - ie the balances
    mutate_all(R.utils::isZero) %>% #check whether they are zero
    rename_all(~paste0('isZero_', substr(., 1, nchar(.) - 4))) #add an isZero prefix on the names

df <- df %>% cbind(iszero) #merge with original df
rm(iszero)

p13 <- df %>% filter(isFraud == "No_Fraud") %>%
    ggplot( aes( x = isFraud, fill = isZero_Old_Balance_of_Or ) ) +
    geom_bar(position = "fill") +
    labs(x = "", y = "", subtitle = "No Fraud") +
    geom_label(stat="count",
               aes(label= paste0(round(((..count..)/sum(..count..)*100),2),"%") ),
               position = position_fill(vjust=0.5)) +
    scale_fill_manual(values =  c("#69b0b3", "#d0e872"), #colors
                      labels = c("Not_Zero", "Zero"), #legend labels
                      name = "Is Old Balance of Origin Zero?") + #legend title
    theme_minimal()+
    coord_flip()+
    theme(axis.text.y = element_blank())


p14 <- df %>% filter(isFraud == "Fraud") %>%
    ggplot( aes( x = isFraud, fill = isZero_Old_Balance_of_Or ) ) +
    geom_bar(position = "fill") +
    labs(x = "", y = "Percentage",subtitle = "Fraud") +
    geom_label(stat="count",
               aes(label= paste0(round(((..count..)/sum(..count..)*100),2),"%") ),
               position = position_fill(vjust=0.5)) +
    scale_fill_manual(values =  c("#69b0b3", "#d0e872"), #colors
                      labels = c("Not_Zero", "Zero"), #legend labels
                      name = "Is Old Balance of Origin Zero?")+ #legend title
    theme_minimal()+
    coord_flip()+
    theme(axis.text.y = element_blank())

p15 <- df %>% filter(isFraud == "No_Fraud") %>%
    ggplot( aes( x = isFraud, fill = isZero_New_Balance_of_Or ) ) +
    geom_bar(position = "fill") +
    labs(x = "", y = "",subtitle="No Fraud") +
    geom_label(stat="count",
               aes(label= paste0(round(((..count..)/sum(..count..)*100),2),"%") ),
               position = position_fill(vjust=0.5)) +
    scale_fill_manual(values =  c("#69b0b3", "#d0e872"),
                      labels = c("Not_Zero", "Zero"),
```

```
                      name = "Is New Balance of Origin Zero?") +
    theme_minimal() +
    coord_flip()+
    theme(axis.text.y = element_blank())

p16 <- df %>% filter(isFraud == "Fraud") %>%
    ggplot( aes( x = isFraud, fill = isZero_New_Balance_of_Or ) ) +
    geom_bar(position = "fill") +
    labs(x = "", y = "Percentage",subtitle = "Fraud") +
    geom_label(stat="count",
               aes(label= paste0(round(((..count..)/sum(..count..)*100),2),"%") ),
               position = position_fill(vjust=0.5)) +
    scale_fill_manual(values =  c("#69b0b3", "#d0e872"),
                      labels = c("Not_Zero", "Zero"),
                      name = "Is New Balance of Origin Zero?")+
    theme_minimal() + coord_flip()+
    theme(axis.text.y = element_blank())

grid.arrange(p13,p14,nrow = 2, top = "Old Balance of Origin")
grid.rect(gp = gpar(lwd = 3, col = "black", fill = NA))
grid.arrange(p15,p16,nrow = 2, top = "New Balance of Origin")
grid.rect(gp = gpar(lwd = 3, col = "black", fill = NA))

rm(p13,p14,p15,p16)

p17 <- df %>% filter(isFraud == "No_Fraud") %>%
    ggplot( aes( x = isFraud, fill = isZero_Old_Balance_of_Destina ) ) +
    geom_bar(position = "fill") +
    labs(x = "", y = "Percentage",subtitle = "No Fraud") +
    geom_label(stat="count",
               aes(label= paste0(round(((..count..)/sum(..count..)*100),2),"%") ),
               position = position_fill(vjust=0.5)) +
    scale_fill_manual(values =  c("#69b0b3", "#d0e872"), #colors
                      labels = c("Not_Zero", "Zero"), #legend labels
                      name = "Is Old Balance of Destination Zero?") + #legend title
    theme_minimal()+
    coord_flip()+
    theme(axis.text.y = element_blank())

p18 <- df %>% filter(isFraud == "Fraud") %>%
    ggplot( aes( x = isFraud, fill = isZero_Old_Balance_of_Destina ) ) +
    geom_bar(position = "fill") +
    labs(x = "", y = "Percentage", subtitle = "Fraud") +
    geom_label(stat="count",
               aes(label= paste0(round(((..count..)/sum(..count..)*100),2),"%") ),
               position = position_fill(vjust=0.5)) +
    scale_fill_manual(values =  c("#69b0b3", "#d0e872"), #colors
                      labels = c("Not_Zero", "Zero"), #legend labels
                      name = "Is Old Balance of Destination Zero?")+ #legend title
    theme_minimal()+
    coord_flip()+
    theme(axis.text.y = element_blank())
```

```r
p19 <- df %>% filter(isFraud == "No_Fraud") %>%
    ggplot( aes( x = isFraud, fill = isZero_New_Balance_of_Destina ) ) +
    geom_bar(position = "fill") +
    labs(x = "", y = "Percentage",subtitle = "No Fraud") +
    geom_label(stat="count",
               aes(label= paste0(round(((..count..)/sum(..count..)*100),2),"%") ),
               position = position_fill(vjust=0.5)) +
    scale_fill_manual(values =  c("#69b0b3", "#d0e872"),
                      labels = c("Not_Zero", "Zero"),
                      name = "Is New Balance of Destination Zero?") +
    theme_minimal() +
    coord_flip()+
    theme(axis.text.y = element_blank())

p20 <- df %>% filter(isFraud == "Fraud") %>%
    ggplot( aes( x = isFraud, fill = isZero_New_Balance_of_Destina ) ) +
    geom_bar(position = "fill") +
    labs(x = "", y = "Percentage", subtitle = "Fraud") +
    geom_label(stat="count",
               aes(label= paste0(round(((..count..)/sum(..count..)*100),2),"%") ),
               position = position_fill(vjust=0.5)) +
    scale_fill_manual(values =  c("#69b0b3", "#d0e872"),
                      labels = c("Not_Zero", "Zero"),
                      name = "Is New Balance of Destination Zero?")+
    theme_minimal() + coord_flip()+
    theme(axis.text.y = element_blank())


grid.arrange(p17,p18,nrow = 2, top = "Old Balance of Destination")
grid.rect(gp = gpar(lwd = 3, col = "black", fill = NA))
grid.arrange(p19,p20,nrow = 2, top = "New Balance of Destination")
grid.rect(gp = gpar(lwd = 3, col = "black", fill = NA))
rm(p17,p18,p19,p20)

#labels for plotting
labs <- c('Amount',"Old/Origin", "New/Origin", "Old/Destination","New/Destination")

#Create a smaller dataset
df_small <- df %>% sample_n(floor(nrow(df)*.1))

#Pairplot
df_small[ ,sapply(df,is.numeric)]  %>%
    ggpairs(mapping=ggplot2::aes(colour = df_small$isFraud),
            title = "Numeric Features",columnLabels = labs, axisLabels = "none")
rm(df_small)

high_cor_feats <- findCorrelation(
                  cor(df[ ,sapply(df,is.numeric)]), #compute corellation table
                  cutoff = .8, verbose = TRUE, # anything above .8 as high_cor
                  names = TRUE, exact = TRUE) #return the names of columns

# Show which features did not make it
print("High Correlation Features to be Removed")
```

```
print(high_cor_feats)

# Drop the highly correlated columns
df <- df %>% select( -high_cor_feats )

#Finding the number of fraud instances
n_minority <- df %>% filter(isFraud == "Fraud") %>% nrow()

#Sample the appropriate number of rows
rand_majority <- df %>% filter(isFraud == "No_Fraud") %>% sample_n( n_minority*6/4)

#Merge into a new dataset and arrange by time so that they get shuffled
df_bal <- df %>% filter(isFraud == "Fraud") %>% rbind( rand_majority ) %>% arrange(step)

#Repeat the frequencies bar plots to contrast with the earlier version of our dataset
p5 <- df %>% ggplot(aes(x = isFraud, y = (..count..))) +
    geom_bar(fill =  c( "#5b7fb0", "#c2695b" ), stat = "count")+
    geom_label(stat='count',
               aes(   label=  paste0(  round(   ((..count..)/sum(..count..)) ,4)*100 ,  "%" ) ) )+
    labs(x = "Fraud or Not", y = "Frequency", title = "Frequency of Fraud before Undersampling", subtitl
    theme_linedraw()

p6 <- df_bal %>% ggplot(aes(x = isFraud, y = (..count..))) +
    geom_bar(fill =  c( "#5b7fb0", "#c2695b" ), stat = "count")+
    geom_label(stat='count',
               aes(   label=  paste0(  round(   ((..count..)/sum(..count..)) ,4)*100 ,  "%" ) ) )+
    labs(x = "Fraud or Not", y = "Frequency", title = "Frequency of Fraud after Undersampling ", subtitl
    theme_linedraw()

grid.arrange(p5,p6)

#Create row indices for test/train split
indx <- createDataPartition(y = df$isFraud, p = .75, list = FALSE)
train_set <- df[indx,]
test_set <- df[-indx,]

#Train our preprocess function parameters
pp <- preProcess( train_set[, sapply(train_set,is.numeric) ] ,
                            method = c("center", "scale"))

#Transform the train_set
train_set <- predict(pp, newdata = train_set)

#Subsequently, transform the test_set
test_set <- predict(pp, newdata = test_set)

print( "Original (Unbalanced) Train Set Structure : " )
train_set %>% str() %>% print()


indx <- createDataPartition(y = df_bal$isFraud, p = .75, list = FALSE) #75% of data as train set
train_set_b <- df_bal[indx,]
test_set_b <- df_bal[-indx,]
```

```r
#Train our preprocess function parameters
pp_b <- preProcess( train_set_b[, sapply(train_set_b,is.numeric) ] ,
                            method = c("center", "scale"))

#Transform the train_set
train_set_b <- predict(pp_b, newdata = train_set_b)

#Subsequently, transform the test_set
test_set_b <- predict(pp_b, newdata = test_set_b)

print( "Balanced Train Set Structure : " )

train_set %>% str() %>% print()

#Setting up the training parameters
tr <- trainControl(method = "repeatedcv", #repeated cross validation
                        number = 9, #9 folds
                        repeats = 3, #3 times
                        classProbs = TRUE, #calculate probabilities for each class
                        summaryFunction = twoClassSummary) #use twoClassSummary to summarize probabilit

#Model Training - Regression Tree
start_time <- Sys.time()

rpart_model_b <- train(isFraud ~ ., #training formula --
                                    #we want to predict isFraud by using all available predictors
                    data = train_set_b, #our balanced dataset
                    method = "rpart", # library containing the tree algorithm
                    tuneLength = 10, # number of different default values used
                                     # to optimize complexity parameter
                    metric = "ROC", # estimate success by the area under the ROC curve
                    trControl = tr, # use cross validation as defined above
                    parms=list(split='information'))

end_time <- Sys.time()

end_time - start_time


rpart_train_pred_b <- predict(rpart_model_b, train_set_b) # class predictions on train set

confusionMatrix(train_set_b$isFraud, rpart_train_pred_b) # report results of predictions
rm(rpart_train_pred_b)

rpart_test_pred_b <- predict(rpart_model_b, test_set_b) #class predictions on test set
confusionMatrix(test_set_b$isFraud, rpart_test_pred_b) #results report
rm(rpart_test_pred_b)

#probabilities predictions on balanced test set
rpart_probs_b <- predict(rpart_model_b, test_set_b, type = "prob")

rpart_ROC_b <- roc(response = test_set_b$isFraud,
                   predictor = rpart_probs_b$Fraud,
```

```
                levels = levels(test_set_b$isFraud))

#probabilities predictions on balanced test set
rpart_probs_b_train <- predict(rpart_model_b, train_set_b, type = "prob")

rpart_ROC_b_train <- roc(response = train_set_b$isFraud,
                predictor = rpart_probs_b_train$Fraud,
                levels = levels(train_set_b$isFraud))

plot(rpart_ROC_b, col = "#040491", main = "ROC Curve on Balanced Data Set (Tree Train vs Test)" )
plot(rpart_ROC_b_train, col = "green", add = TRUE)
legend(.4,.18, legend = c("Test Set", "Train Set"), col = c("#040491","green"), lty = 1 )
tree_overfit <- round(rpart_ROC_b_train$auc - rpart_ROC_b$auc, 3)
anot <- paste( "Train - Test Area : ", tree_overfit  )
text(labels = anot, x = 0.2, y = .25 )



#Pick a random sample from the unbalanced test set
random_sample <- test_set %>%
                 sample_n(nrow(test_set)/1.3) #size = 75% of the test set rows

rpart_probs_c <- predict(rpart_model_b, random_sample, type = "prob")

rpart_ROC_c <- roc(response = random_sample$isFraud,
                predictor = rpart_probs_c$Fraud,
                levels = levels(random_sample$isFraud))

plot(rpart_ROC_c, col = "#63a105", main = "ROC curve on Unbalanced Test Set")

anot <-   paste("Area Under the Curve : " , round(rpart_ROC_c$auc,3))

text(labels = anot , x = 0, y= .2)



plot(rpart_ROC_b, col = "#040491", main = "Balanced vs Unbalanced Test Set (Classification Tree)" )
plot(rpart_ROC_c, col = "#63a105",add = TRUE)
legend(.4,.18, legend = c("Balanced", "Unbalanced"), col = c("#040491","#63a105"), lty = 1 )
tree_unb_perf <- round((rpart_ROC_b$auc - rpart_ROC_c$auc), 4)
anot <- paste( "Balanced - Unbalanced Area : ",  tree_unb_perf )
text(labels = anot, x = 0.2, y = .25 )


#Define Hyperparameters
tune_grid <- expand.grid(.mtry = c((floor(dim(random_sample)[2] * .6)),  # 60% and of predictors
                                   (floor(dim(random_sample)[2] * .4))) , # 40% of predictors
            .ntree = seq(70, 130, by = 30)) # number of trees

print("Tune Grid:")
print(tune_grid)

#Seed for reproducibility
```

```r
set.seed(123)

#Model Training - Random Forest
start_time <- Sys.time()
rand_for_b <- train(isFraud ~ .,
                    data = train_set_b, #train on the balanced set
                    method="rf", # use the random forest package
                    metric = "ROC", # elect best model by ROC curve
                    TuneGrid = tune_grid, # use custom grid for hyperparameters
                    trControl=tr) # #use cross validation

end_time <- Sys.time()
end_time - start_time

#plot error versus number of trees
plot(rand_for_b$finalModel, main = "Estimated Error by Number of Trees")

#Variable Importance
imps <- rand_for_b$finalModel$importance

#isolate feature names
nams <- (imps %>% attr("dimnames"))[[1]]

#bind names with MeandecreaseGini values
imps <- imps %>% as.data.frame() %>%
  cbind(nams) %>%
  arrange(desc(MeanDecreaseGini)) %>%
  head(10) #keep top 10

#Plot importances as red dots annotating values in black text
imps %>%
  ggplot( aes(x = reorder(nams,MeanDecreaseGini), y = MeanDecreaseGini ) ) +
  geom_point(size = 9, col = "#ff583b")+
  geom_vline(xintercept = 1: nrow(imps), col = "darkgray") +
  geom_text( aes(label = round(MeanDecreaseGini,2), size = 5), show.legend = FALSE , col = "black" ) +
  coord_flip()+
  labs(title = "Variable Importance", x = "", y = "",subtitle = "Top 10 Features" )+
  theme_classic()

#Predictions on train set
rand_for_pred_b_train <- predict(rand_for_b, train_set_b) #class predictions on test set
print( "Train Set Confusion Matrix (Balanced Data) : " )
confusionMatrix(train_set_b$isFraud, rand_for_pred_b_train) #results report

#Predictions on test set
rand_for_pred_b <- predict(rand_for_b, test_set_b) #class predictions on test set
print( "Test Set Confusion Matrix (Balanced Data) : " )
confusionMatrix(test_set_b$isFraud, rand_for_pred_b) #results report

#probabilities predictions on balanced train set
rand_for_probs_b <- predict(rand_for_b, train_set_b, type = "prob")

#respective ROC curve
```

```r
rand_for_ROC_b <- roc(response = train_set_b$isFraud,
                      predictor = rand_for_probs_b$Fraud,
                      levels = levels(train_set_b$isFraud))

#probabilities predictions on balanced test set
rand_for_probs_b_test <- predict(rand_for_b, test_set_b, type = "prob")

#respective ROC curve
rand_for_ROC_b_test <- roc(response = test_set_b$isFraud,
                      predictor = rand_for_probs_b_test$Fraud,
                      levels = levels(test_set_b$isFraud))

#Plot Train vs Test Performance for forest
plot(rand_for_ROC_b, col = "#040491", main = "ROC Curve on Balanced Data Set (Forest Train vs Test)" )
plot(rand_for_ROC_b_test, col = "#e6874c", add = TRUE)
#Add legend
legend(.4,.18, legend = c("Train Set", "Test Set"), col = c("#040491","#e6874c"), lty = 1 )
#Store for use in the results section
for_overfit <- round((rand_for_ROC_b$auc - rand_for_ROC_b_test$auc), 4)
#Add text
anot <- paste( "Train - Test Area : ", for_overfit  )
text(labels = anot, x = 0.2, y = .25 )

#Confusion Matrix of performance on random sample
rand_for_pred_c <- predict(rand_for_b, random_sample) #class predictions on random sample set
print("Random Sample Confusion Matrix (Unbalanced Data) : ")
confusionMatrix(random_sample$isFraud, rand_for_pred_c) #results report
rm(rand_for_pred_c)

#probabilities predictions on random sample
rand_for_probs_c <- predict(rand_for_b, random_sample, type = "prob")

rand_for_ROC_c <- roc(response = random_sample$isFraud,
                      predictor = rand_for_probs_c$Fraud,
                      levels = levels(random_sample$isFraud))

#Plot Roc curve from balanced data
plot(rand_for_ROC_c, col = "#627502", main = "Balanced vs Unbalanced (Random Forest)" )
#Plot Roc curve from unbalanced data
plot(rand_for_ROC_b_test, col = "#cf4696",add = TRUE)
#Add legend
legend(.4,.18, legend = c("Unbalanced", "Balanced"), col = c("#627502","#cf4696"), lty = 1 )
#Store for use in results section
for_unb_perf <- round((rand_for_ROC_c$auc - rand_for_ROC_b_test$auc), 4)
#Add text
anot <- paste( "Unbalanced - Balanced Area : ", for_unb_perf )
text(labels = anot, x = 0.2, y = .25 )
rm(rand_for_probs_c)

#ROC curve of forest
plot(rand_for_ROC_c, col = "#89c1c4", main = "Tree vs Forest (Unbalanced Test Set)" )
#Roc curve of tree
plot(rpart_ROC_c, col = "#b65dcf",add = TRUE)
```

```r
#Show legend
legend(.4,.18, legend = c("Random Forest", "Classification Tree"),
       col = c( "#89c1c4","#b65dcf"), lty = 1 )
#Display the difference
anot <- paste( "Forest - Tree Area : ", round((rand_for_ROC_c$auc - rpart_ROC_c$auc), 4) )
text(labels = anot, x = 0.08, y = .25 )


#Create a dataframe containing the train-test differences
temp <- data.frame( model = c("Tree","Forest"), performance = c(tree_overfit,for_overfit)  )

#Plot the results for each model
temp %>%ggplot( aes(x = model, y = performance  ) ) +
  geom_point(size = 9, col = "#7cd8f7")+
  geom_vline(xintercept = 1:2, col = "darkgray") +
  geom_text( aes(label =(performance), size = 5), show.legend = FALSE , col = "black" ) +
  coord_flip()+
  labs(title = "Train - Test Difference per Model", x = "", y = "" )+
  theme_classic()

#Create a dataframe containing the Balanced - Unbalanced differences
temp <- data.frame( model = c("Tree","Forest"), performance = c(tree_unb_perf,for_unb_perf)  )

#Plot the Results
temp %>%ggplot( aes(x = model, y = performance  ) ) +
  geom_point(size = 9, col = "#d591ff")+
  geom_vline(xintercept = 1:2, col = "darkgray") +
  geom_text( aes(label =(performance), size = 5), show.legend = FALSE , col = "black" ) +
  coord_flip()+
  labs(title = "Balanced - Unbalanced Dataset Difference per Model", x = "", y = "" )+
  theme_classic()
```