Capstone Project - Fraud Detection

Stanislav Taov

28/12/2020

Introduction

This project is a part of the edX - HarvardX Data Science course. According many available reports credit card frauds costed financial institutions a wooping 28 billion dollars just in 2018. Recent years we see a lot of development in cashless/digital transaction technologies for example ApplePay and GoogleWallet that will bring us to chaseless future very soon. Fraud transaction cost can put a lot of pressure on the financial institutions since it is not only loss due to the fraud itself, there is also cost associated with managing losses, cost of managing frustrated customers, cost of potential losing the business and many more. That is why many financial organizations are facing the challenge of building a successful fraud detection model which can flag fraudulent transactions with very small false-negative and false-positive results.

Goal

The main purpose of this project was to create a machine learning model that would not only flag the suspicious/fraud transactions but also would identify changes in the patterns and adapt automatically to new fraud patterns.

Data

The dataset for this project was obtain from Kaggle website (https://www.kaggle.com/ntnu-testimon/paysim1) and was listed under recommended list of dataset for this project (https://www.kaggle.com/annavictoria/ml-friendly-public-datasets). Financial data is very important for many researches that try to build sufficient models for fraud detection and prevent any illegal financial activities. However this data is not publicly available for data scientists to work on since it can raise certain privacy concerns. The group of researchers generated a dataset called PaySim to tackle such a problem. PaySim simulates mobile money transactions based on a sample of private dataset extracted from one month of financial logs from a mobile money service implemented in an African country. Therefore this data is perfect for building and evaluating the performance of fraud detection models

Data dictionary

step - maps a unit of time in the real world. In this case 1 step is 1 hour of time. Total steps 744 (31 days simulation).
type - CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.
amount - amount of the transaction in local currency.
nameOrig - customer who started the transaction.

oldbalanceOrg - initial balance before the transaction.
newbalanceOrig - new balance after the transaction.
nameDest - customer who is the recipient of the transaction.
oldbalanceDest - initial balance recipient before the transaction. Note that there is not information for customers that start with M (Merchants).
newbalanceDest - new balance recipient after the transaction. Note that there is not information for customers that start with M (Merchants).
isFraud - This is the transactions made by the fraudulent agents inside the simulation. In this specific dataset the fraudulent behavior of the agents aims to profit by taking control or customers accounts and try to empty the funds by transferring to another account and then cashing out of the system.
isFlaggedFraud - The business model aims to control massive transfers from one account to another and flags illegal attempts. An illegal attempt in this dataset is an attempt to transfer more than 200.000 in a single transaction.

Data exploration

First, let's install and load all necessary libraries and load the dataset.

```
if(!require(tidyverse))
  install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(ggplot2))
  install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(dplyr))
  install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(readr))
  install.packages("readr", repos = "http://cran.us.r-project.org")
if(!require(outliers))
  install.packages("outliers", repos = "http://cran.us.r-project.org")
if(!require(stringr))
  install.packages("stringr", repos = "http://cran.us.r-project.org")
if(!require(corrplot))
  install.packages("corrplot", repos = "http://cran.us.r-project.org")
if(!require(ROCR))
  install.packages("ROCR", repos = "http://cran.us.r-project.org")
if(!require(pROC))
  install.packages("pROC", repos = "http://cran.us.r-project.org")
if(!require(gbm))
  install.packages("gbm", repos = "http://cran.us.r-project.org")
if(!require(googledrive))
  install.packages("googledrive", repos = "https://cloud.r-project.org/")

library(googledrive)
library(tidyverse)
library(ggplot2)
library(dplyr)
library(readr)
library(outliers)
library(stringr)
library(corrplot)
```

```r
library(caret)
library(ROCR)
library(pROC)
library(gbm)
```

```r
# loading the dataset, it will ask for google authorization and will provide a code
temp <- tempfile(fileext = ".zip")
dl <- drive_download(
  as_id("1PWKrd28N99dIM7CCHPwmlcYHrE9s2soY"), path = temp, overwrite = TRUE)
out <- unzip(temp, exdir = tempdir())
df <- read_csv(out[1])
```

```r
#setting seed for the process replication
set.seed(2020)
```

The dataset is over 400MB and can take a few minutes to load. We can plot the first 5 rows to observe the data.

Displaying data structure can help us to understand the data a little bit further. There are 8 numerical features and 3 categorical.

```r
# displaying data structure
str(df)
```

```
## tibble [6,362,620 x 11] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ step          : num [1:6362620] 1 1 1 1 1 1 1 1 1 1 ...
##  $ type          : chr [1:6362620] "PAYMENT" "PAYMENT" "TRANSFER" "CASH_OUT" ...
##  $ amount        : num [1:6362620] 9840 1864 181 181 11668 ...
##  $ nameOrig      : chr [1:6362620] "C1231006815" "C1666544295" "C1305486145" "C840083671" ...
##  $ oldbalanceOrg : num [1:6362620] 170136 21249 181 181 41554 ...
##  $ newbalanceOrig: num [1:6362620] 160296 19385 0 0 29886 ...
##  $ nameDest      : chr [1:6362620] "M1979787155" "M2044282225" "C553264065" "C38997010" ...
##  $ oldbalanceDest: num [1:6362620] 0 0 0 21182 0 ...
##  $ newbalanceDest: num [1:6362620] 0 0 0 0 0 ...
##  $ isFraud       : num [1:6362620] 0 0 1 1 0 0 0 0 0 0 ...
##  $ isFlaggedFraud: num [1:6362620] 0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   step = col_double(),
##   ..   type = col_character(),
##   ..   amount = col_double(),
##   ..   nameOrig = col_character(),
##   ..   oldbalanceOrg = col_double(),
##   ..   newbalanceOrig = col_double(),
##   ..   nameDest = col_character(),
##   ..   oldbalanceDest = col_double(),
##   ..   newbalanceDest = col_double(),
##   ..   isFraud = col_double(),
##   ..   isFlaggedFraud = col_double()
##   .. )
```

```r
# checking missing values
sum(is.na(df))
```

```
## [1] 0
```

There is no missing value in the dataset. The table below displays the number and percentage of fraudulent transactions by the transaction type.

```r
df %>%
  group_by(type) %>%
  summarise(transactions = n(), fraud = sum(isFraud), percentage=(fraud/transactions)*100) %>%
  knitr::kable()
```

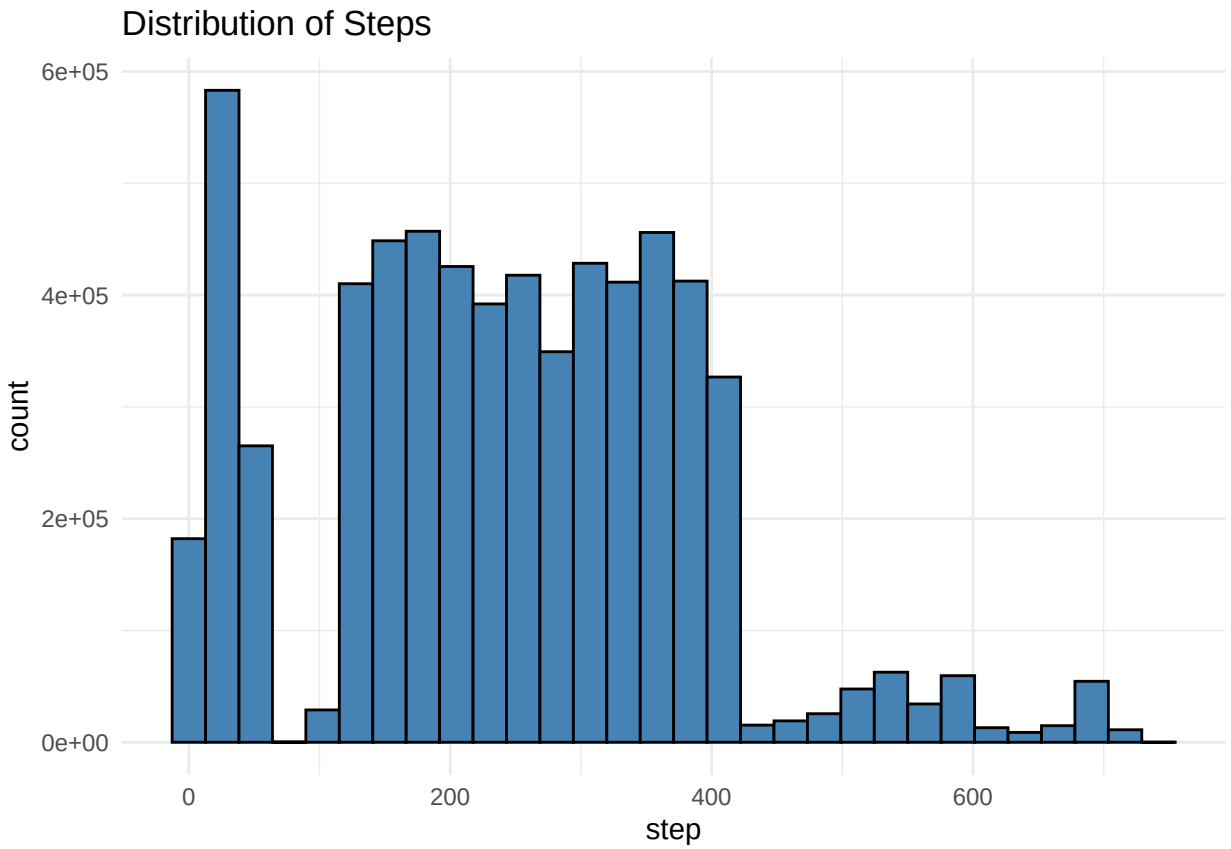
type	transactions	fraud	percentage
CASH_IN	1399284	0	0.0000000
CASH_OUT	2237500	4116	0.1839553
DEBIT	41432	0	0.0000000
PAYMENT	2151495	0	0.0000000
TRANSFER	532909	4097	0.7687992

As expected, the dataset's labels are highly imbalanced which can lead to a bias towards the dominant label, so potentially some models can predict all labels as not fraud and still produce very high accuracy close to 99%. We have roughly 1% of transaction labelled as a fraud. There are a few techniques that can solve the problem with unbalanced data. There are undersample, oversample and SMOTE. Due to the compute power limitation we are going to use the undersample technique to balance the data.

We will be evaluating our models using the AUC - ROC curve. For every financial institution is very important to have both false-positive and false-negative to a minimum. False-positives can cause a lot of loss in long run since people whose accounts were flagged with fraudulent transactions can think about switching to a different bank. Therefore, banks can neglect some false-negatives since they might cause less loss. Thus our main goal is to create a model that maximizes both Sensitivity and Specificity metrics and the most convenient solution that encompasses both metrics is to use the AUC - ROC curve.

Now let's plot the distribution of the steps to see whether it looks normal or whether there are any anomalies.

```r
ggplot(df, aes(x=step)) +
  geom_histogram(fill="steelblue", color = "black") +
  theme_minimal() +
  ggtitle("Distribution of Steps")
```



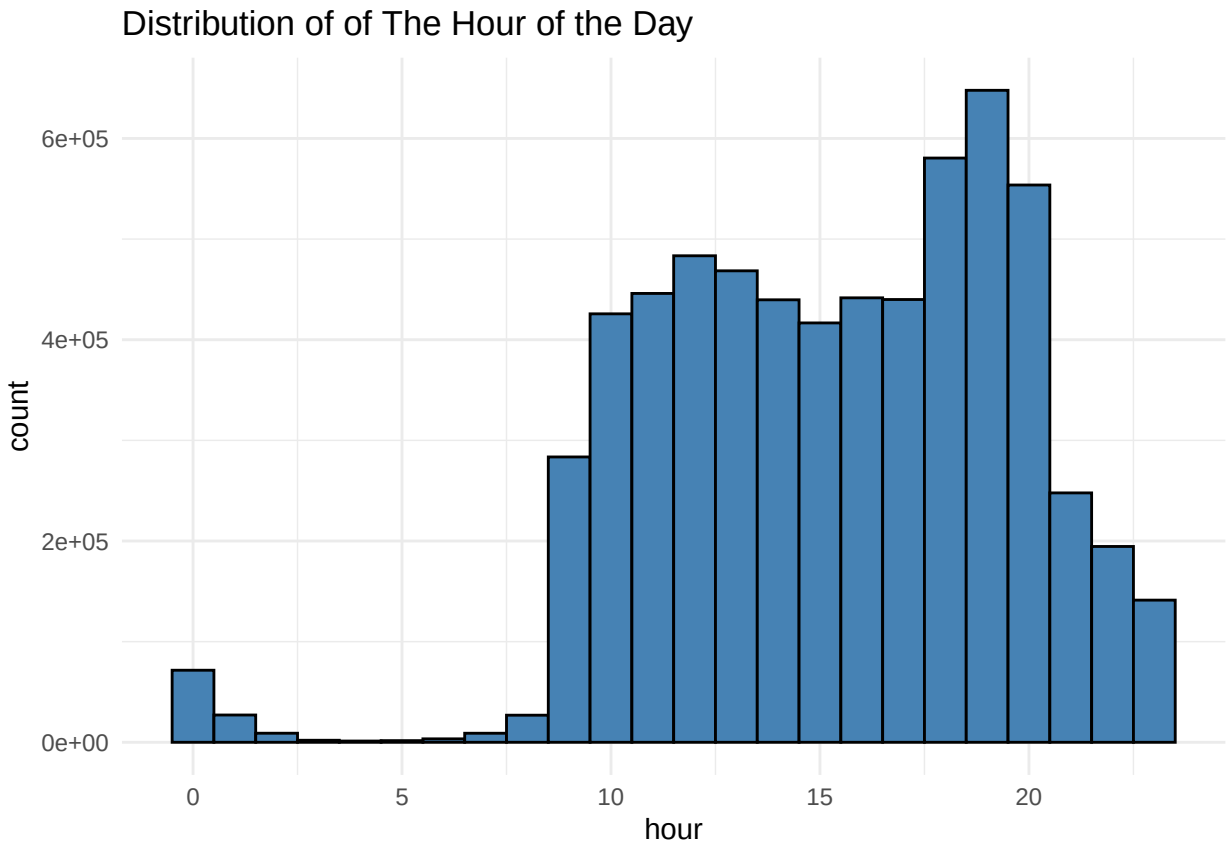
As we know, steps represent the hour of our date range as a step. There is a total of 744 steps. It seems like there were more transactions at the beginning of the given dataset, and then the amount of transactions has dropped significantly. It could be an indication that the data is generated not perfectly. It is highly unlikely that any financial institution can see these drastic changes just within a few months.

We can manipulate this feature to make it more usable for our model, if we apply a modal value of 24 we can convert steps into the hour of the day which we can use to see if there are any correlations between the hour of the day and our predicting labels (fraudulent transactions).

```
df$hour=df$step %%24
```

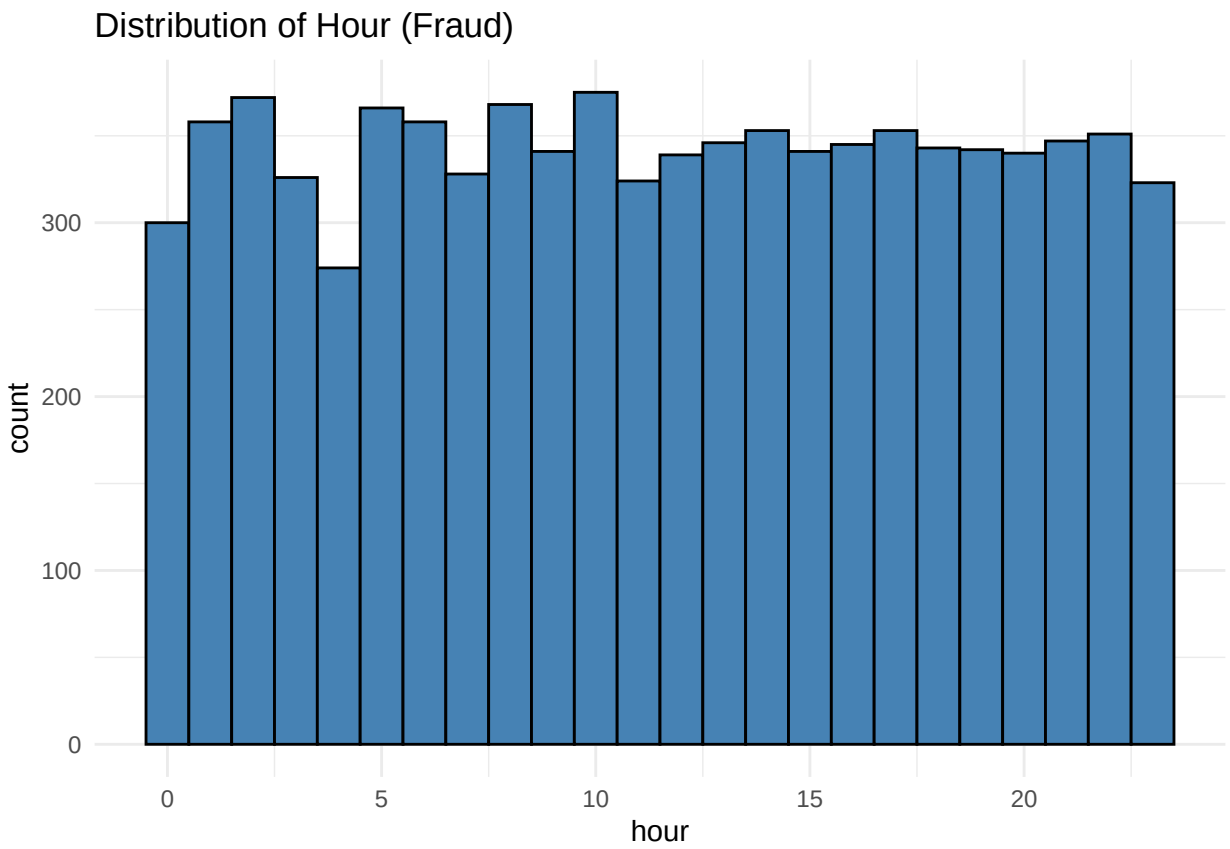
Now, we can plot the hour feature to see the distribution of the data by the hour.

```
ggplot(df, aes(x=hour)) +
  geom_histogram(bins = 24, fill="steelblue", color = "black") +
  theme_minimal() +
  ggtitle("Distribution of of The Hour of the Day")
```



Observing this graph, we can conclude that most of the transactions happened between 10 am and 8 pm. Now we can filter the data and look at data with only fraudulent transactions.

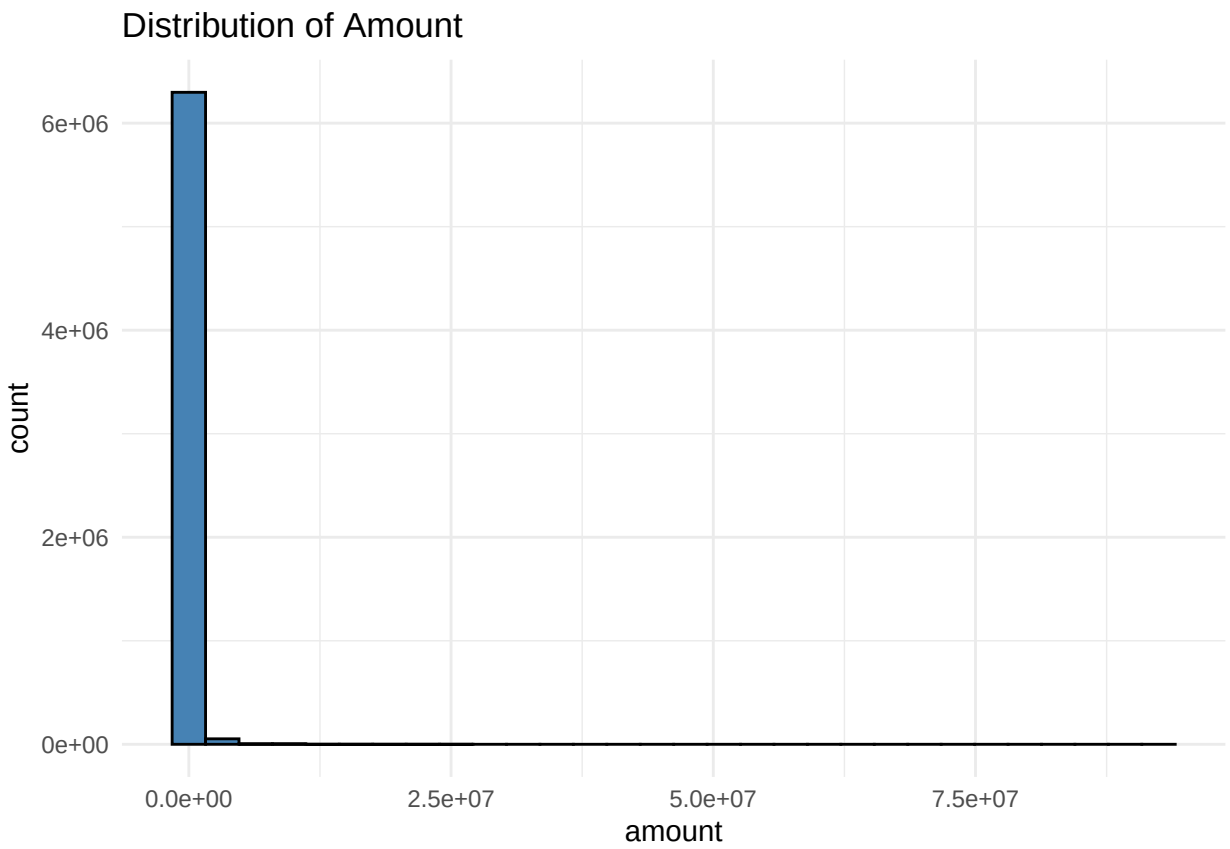
```
df %>%
  filter(isFraud == 1) %>%
  ggplot(aes(x=hour)) +
    geom_histogram(bins = 24, fill="steelblue", color = "black") +
    theme_minimal() +
    ggtitle("Distribution of Hour (Fraud)")
```



Interestingly it doesn't have a normal distribution and looks mostly like a uniform distribution. So, it is equally likely to get a fraudulent transaction at any hour of the day, which looks very unrealistic and can indicate the synthetic nature of the data.

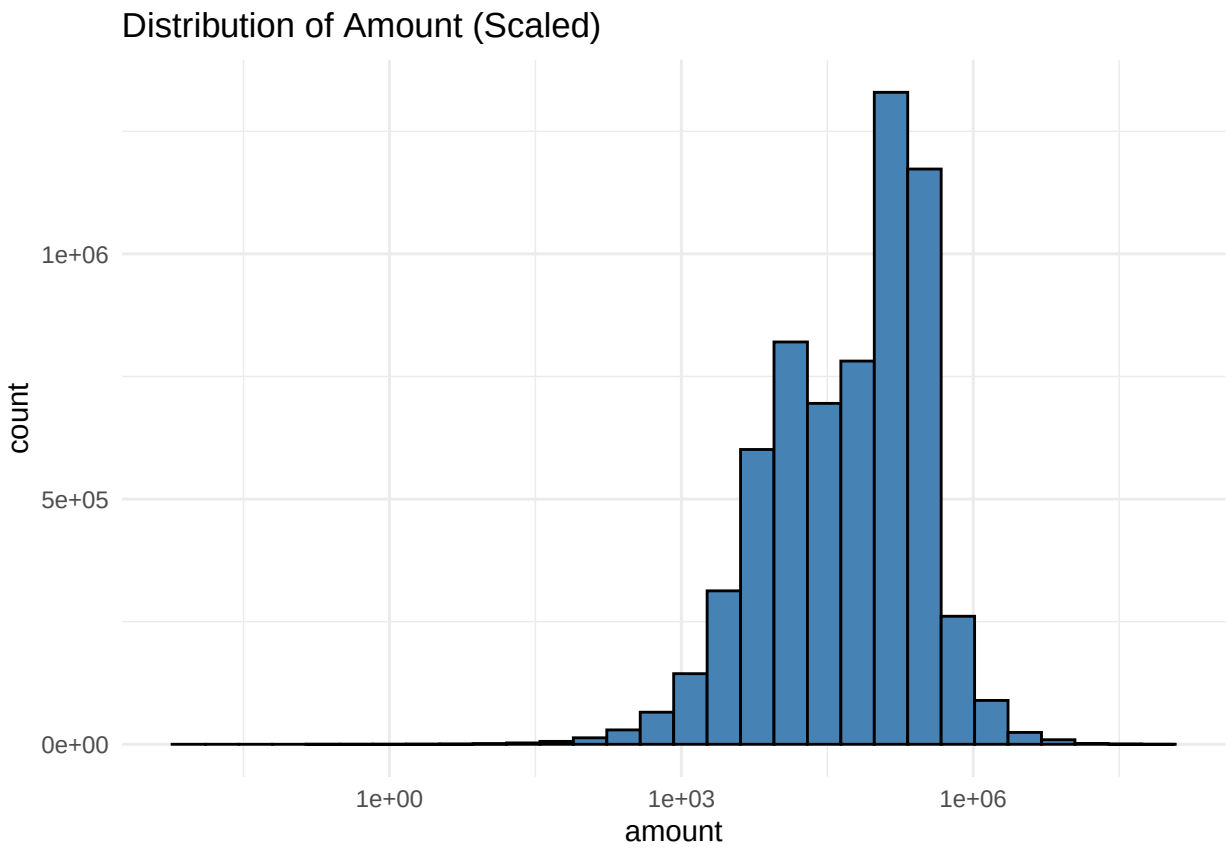
Let's look at the distribution of transfer amounts. As we can see the data has a lot of outliers. But we still can check the distribution by scaling the data.

```
ggplot(df, aes(x=amount)) +
  geom_histogram(fill="steelblue", color = "black") +
  theme_minimal() +
  ggtitle("Distribution of Amount")
```



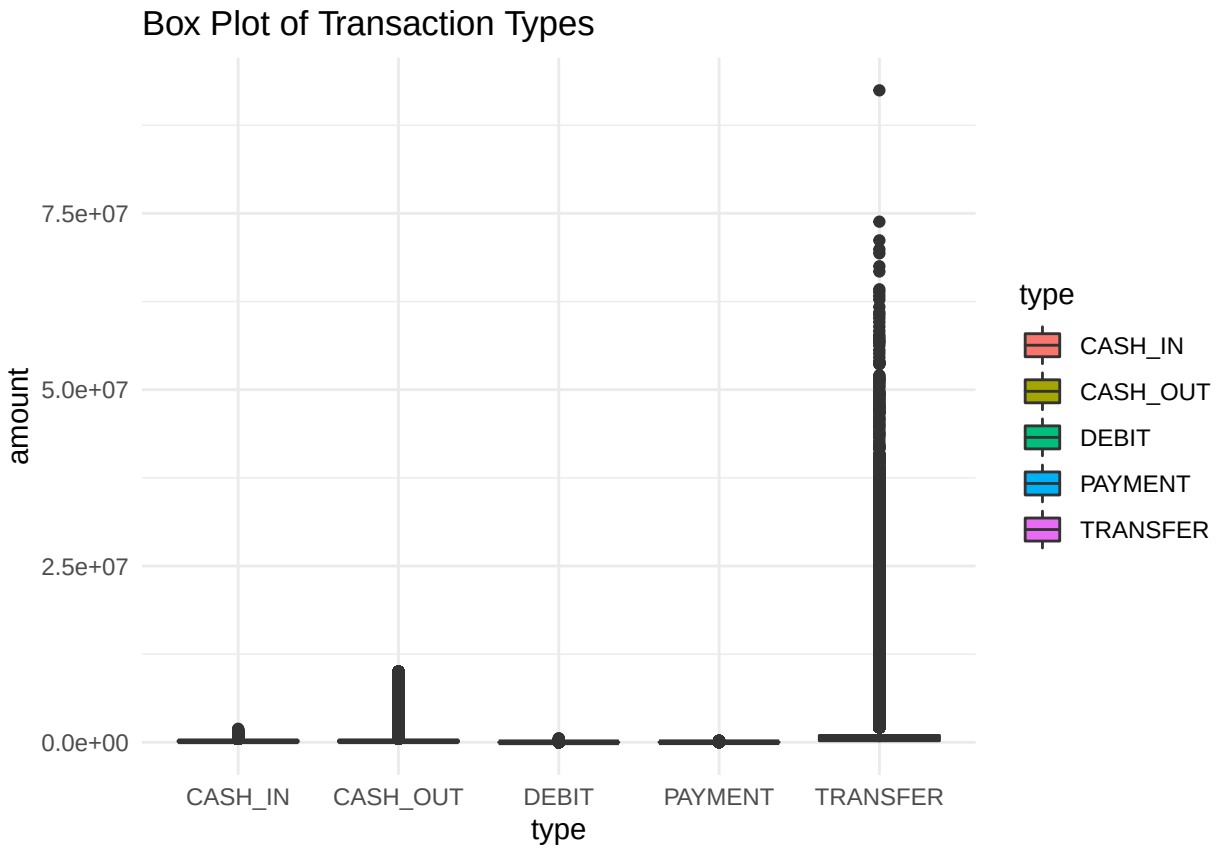
After scaling X-axis with log10 we can observe that the distribution of the amount feature looks quite normal.

```
ggplot(df, aes(x=amount)) +
  geom_histogram(fill="steelblue", color = "black") +
  theme_minimal() +
  scale_x_log10() +
  ggtitle("Distribution of Amount (Scaled)")
```



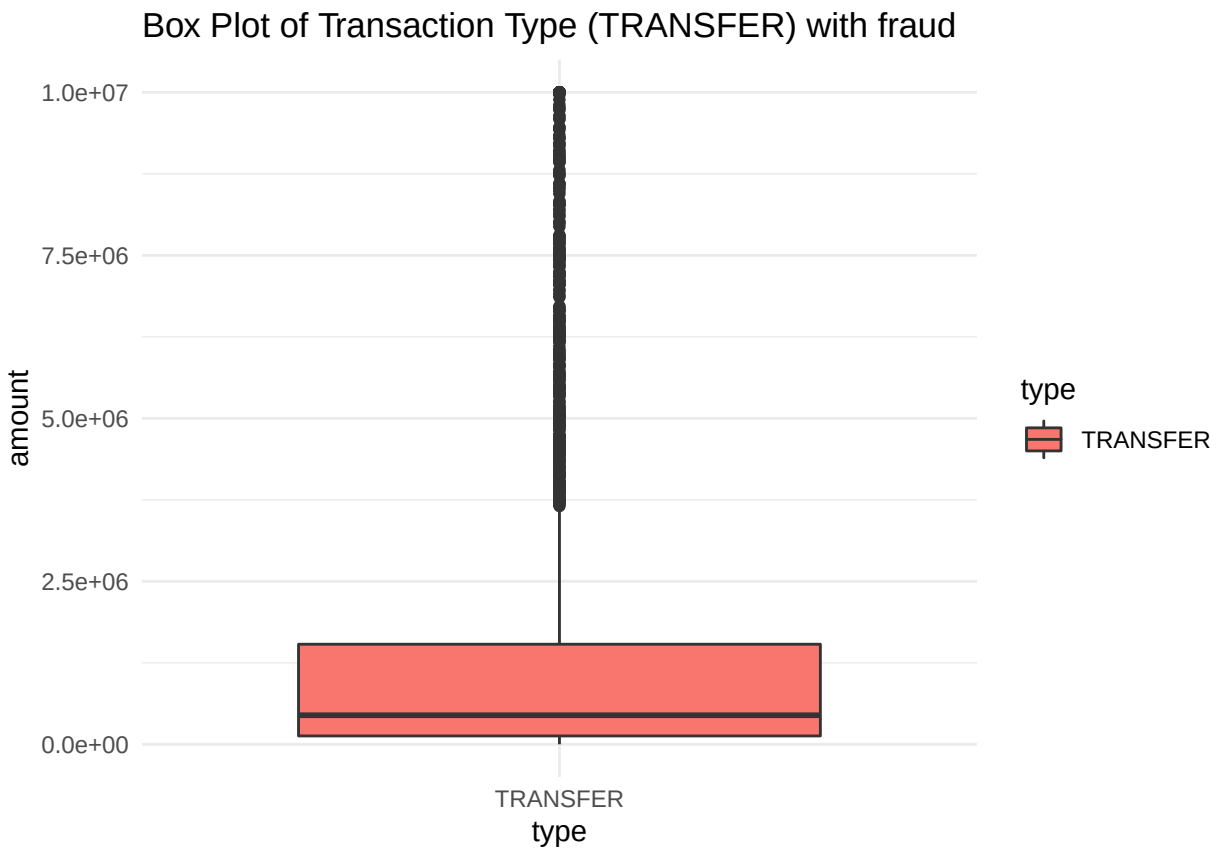
However, it would be very interring to see what type of transaction has the most outliers.

```
ggplot(df, aes(x=type, y=amount, fill=type)) +
  geom_boxplot() +
  theme_minimal() +
  ggtitle("Box Plot of Transaction Types")
```



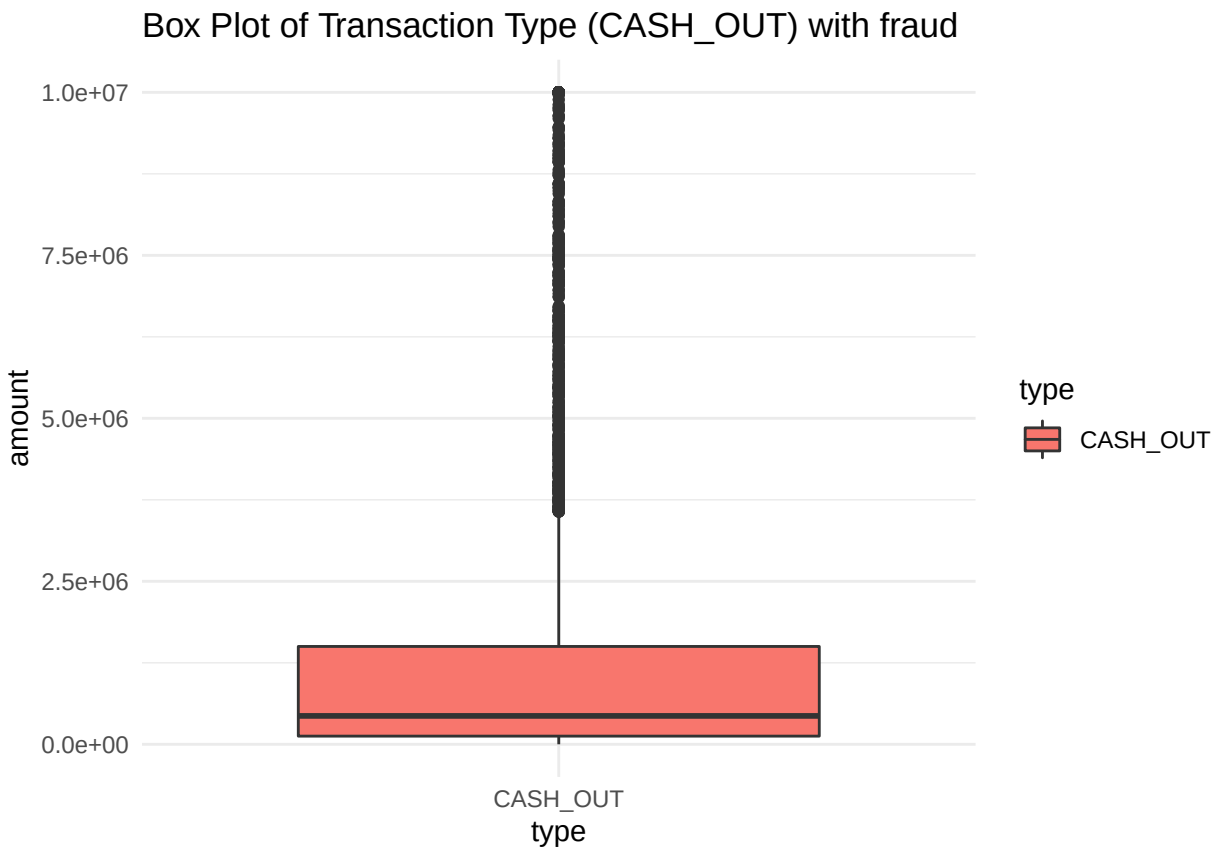
From the box-plot we can see that **TRANSFER** and **CASH_OUT** have the most outliers. We should point out that only these two transaction types have fraudulent transactions. Let's further investigate these two transaction types and plot box-plots only for **TRANSFER** and **CASH_OUT** types with fraud transactions.

```
df %>%
  filter(type == "TRANSFER", isFraud == 1) %>%
  select(type, amount, isFraud) %>%
  ggplot(aes(x=type, y=amount, fill=type)) +
   geom_boxplot() +
  theme_minimal() +
  ggtitle("Box Plot of Transaction Type (TRANSFER) with fraud")
```



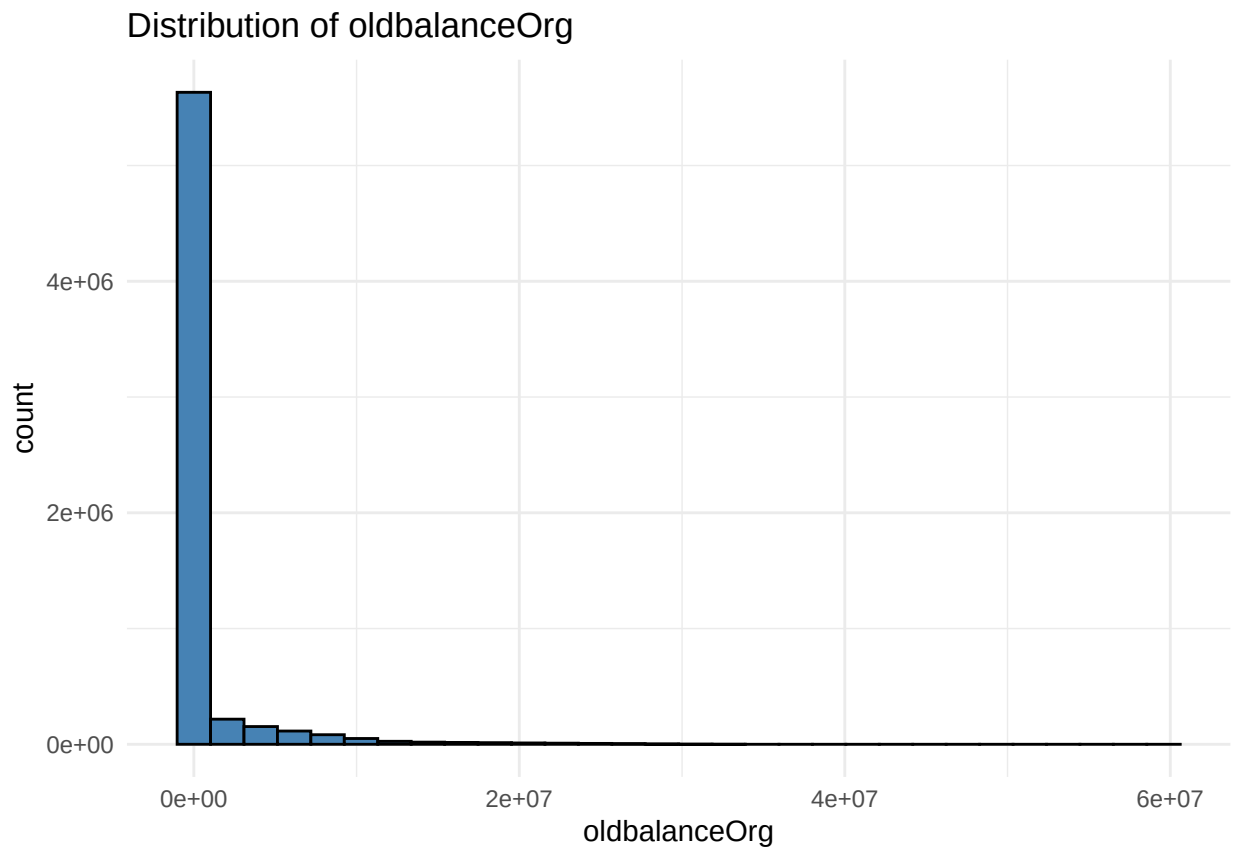
As we can observe there are a lot of fraudulent transactions with **TRANSFER** type that has astronomical transfer amounts. Again this can show the syntactical nature of the data as it is highly unlikely that these amounts were linked with real bank accounts.

```
df %>%
  filter(type == "CASH_OUT", isFraud == 1) %>%
  select(type, amount, isFraud) %>%
  ggplot(aes(x=type, y=amount, fill=type)) +
   geom_boxplot() +
  theme_minimal() +
  ggtitle("Box Plot of Transaction Type (CASH_OUT) with fraud")
```



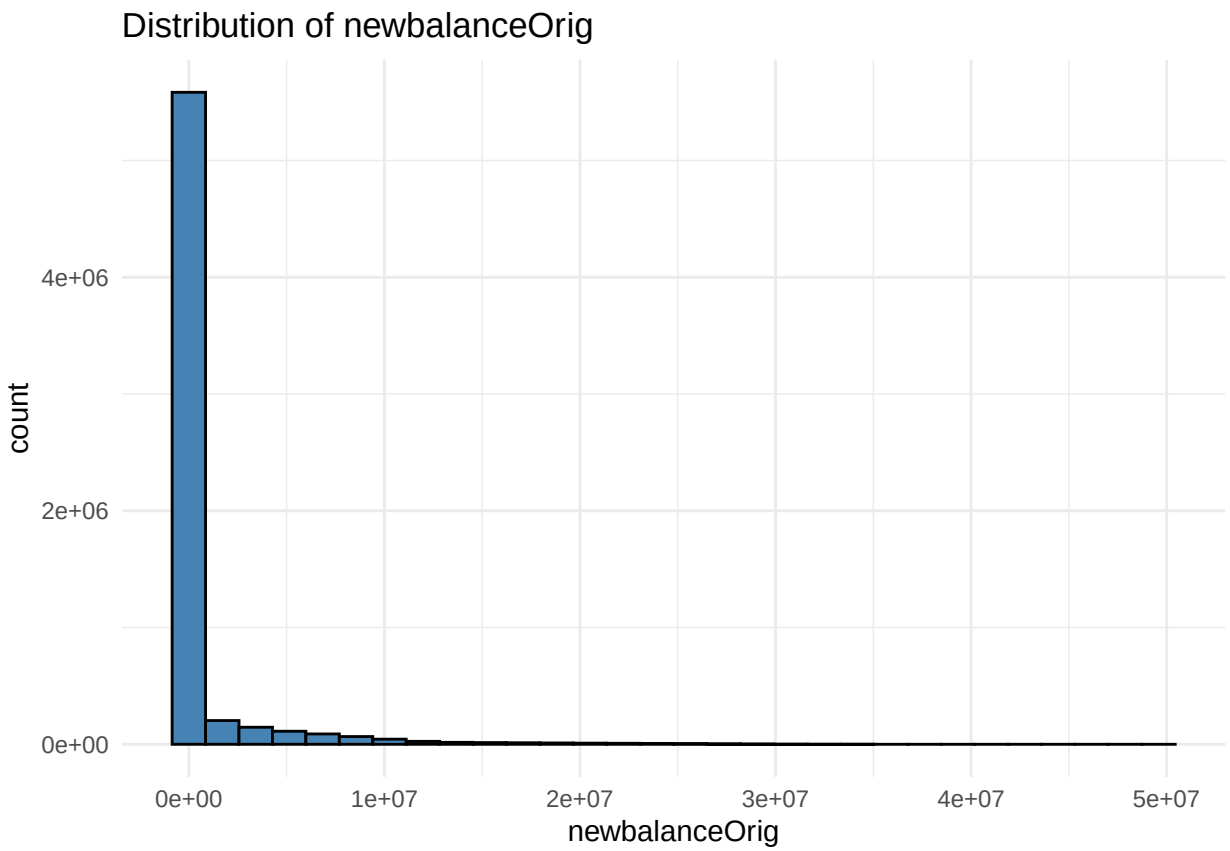
A similar picture we can see when we create a box-plot for **CASH_OUT** type. Let's plot the distribution of (oldbalanceOrg) initial balance before the transaction.

```
ggplot(df, aes(x=oldbalanceOrg)) +
  geom_histogram(fill="steelblue", color = "black") +
  theme_minimal() +
  ggtitle("Distribution of oldbalanceOrg")
```



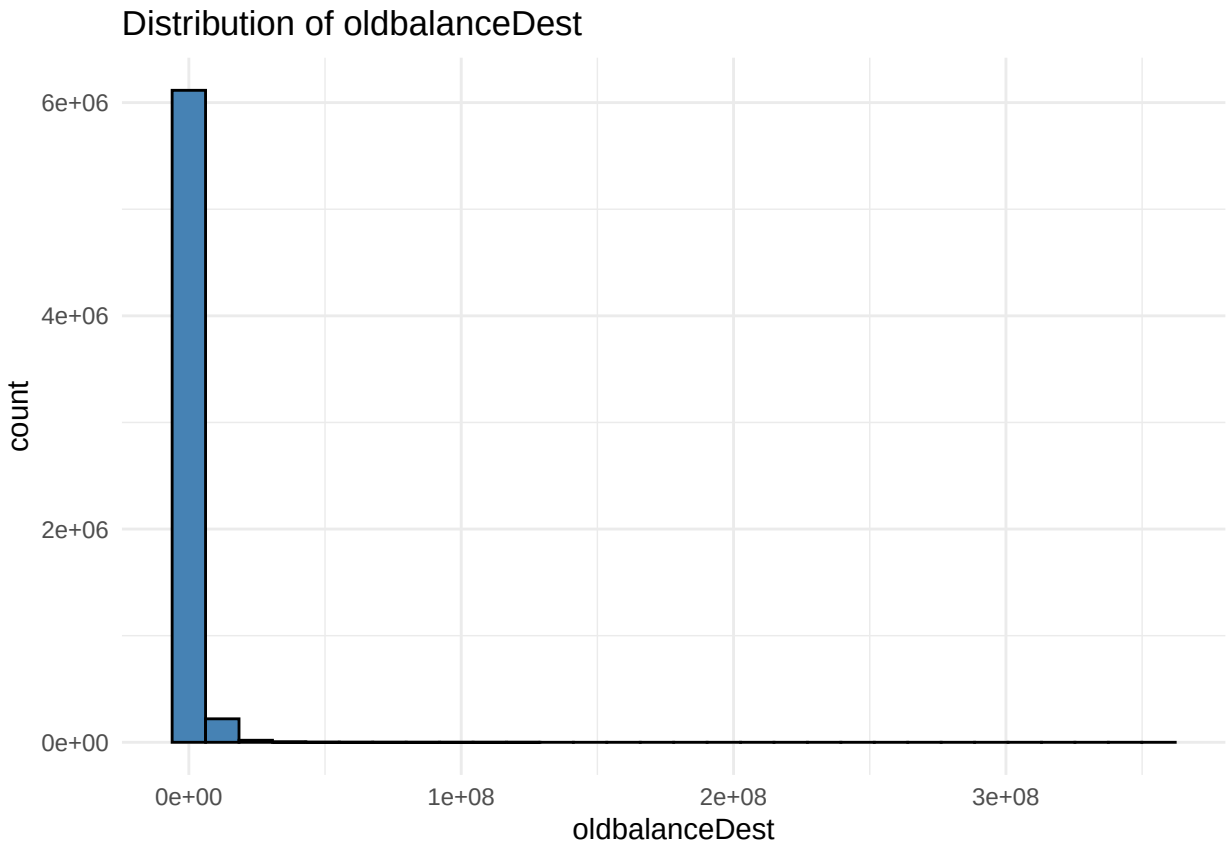
The initial balance before the transaction distribution doesn't look normal and skewed right that can indicate that there are some bank accounts with an extremely high initial balance. Let's display the next newbalanceOrig (new balance after the transaction) distribution.

```
ggplot(df, aes(x=newbalanceOrig)) +
  geom_histogram(fill="steelblue", color = "black") +
  theme_minimal() +
  ggtitle("Distribution of newbalanceOrig")
```



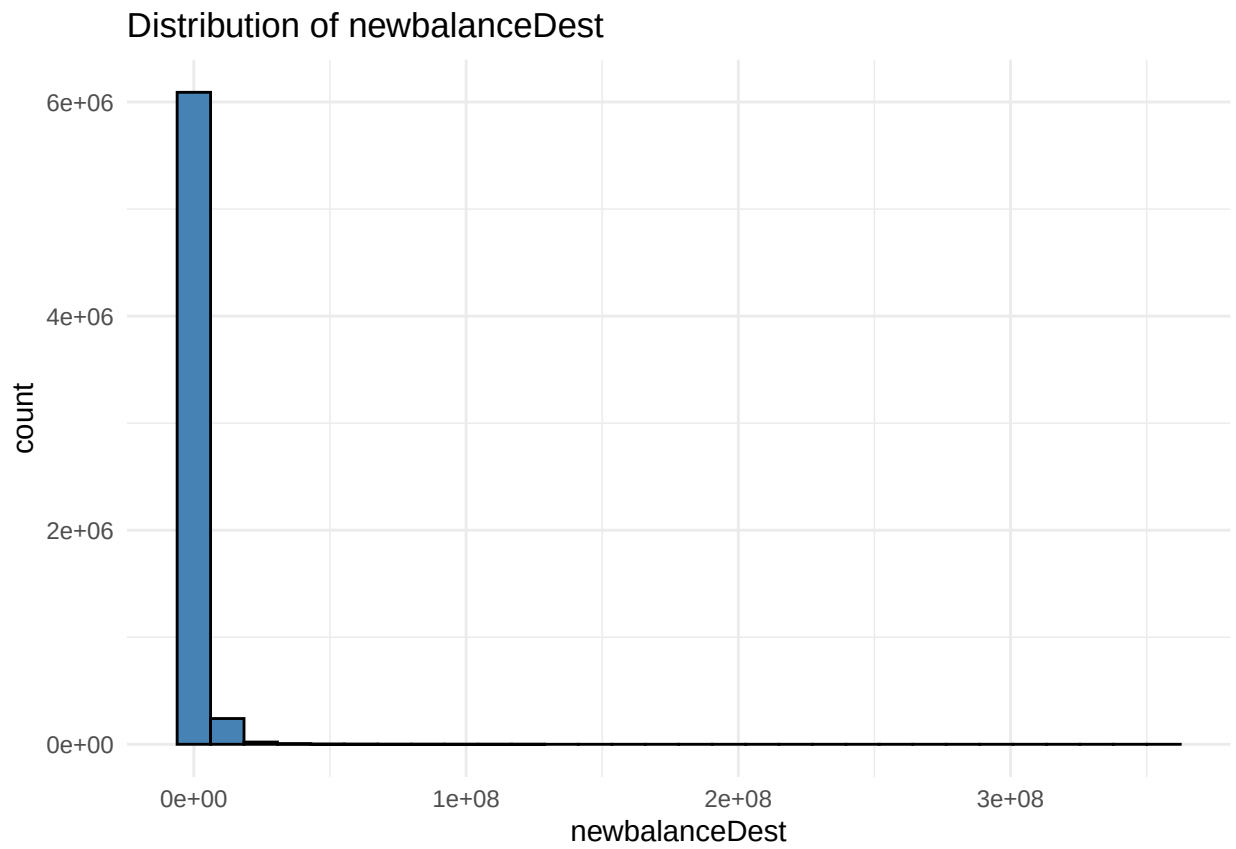
As expected we see a similar right-skewed not a normal distribution of newbalanceOrig. Lastly, we are going to check the distribution of oldbalanceDest (initial balance recipient before the transaction) and newbalanceDest (new balance recipient after the transaction). It is important to note here that in reality, this information can be very limited

```
ggplot(df, aes(x=oldbalanceDest)) +
  geom_histogram(fill="steelblue", color = "black") +
  theme_minimal() +
  ggtitle("Distribution of oldbalanceDest")
```



The oldbalanceDest distribution looks very similar to oldbalanceDest.

```
ggplot(df, aes(x=newbalanceDest)) +
  geom_histogram(fill="steelblue", color = "black") +
  theme_minimal() +
  ggtitle("Distribution of newbalanceDest")
```



As we observed previously with the "Amount" feature all numerical features have a lot of outliers that skew the data a lot. Let's check outliers for each numerical feature.

```
grubbs.test(df$amount)
```

```
##
##  Grubbs test for one outlier
##
## data:  df$amount
## G = 152.79357, U = 0.99633, p-value < 2.2e-16
## alternative hypothesis: highest value 92445516.64 is an outlier
```

```
grubbs.test(df$oldbalanceOrg)
```

```
##
##  Grubbs test for one outlier
##
## data:  df$oldbalanceOrg
## G = 20.34149, U = 0.99993, p-value < 2.2e-16
## alternative hypothesis: highest value 59585040.37 is an outlier
```

```
grubbs.test(df$newbalanceOrig)
```

```
##
##  Grubbs test for one outlier
```

```
##
## data:  df$newbalanceOrig
## G = 16.66523, U = 0.99996, p-value < 2.2e-16
## alternative hypothesis: highest value 49585040.37 is an outlier
```

```
grubbs.test(df$oldbalanceDest)
```

```
##
##  Grubbs test for one outlier
##
## data:  df$oldbalanceDest
## G = 104.41200, U = 0.99829, p-value < 2.2e-16
## alternative hypothesis: highest value 356015889.35 is an outlier
```

```
grubbs.test(df$newbalanceDest)
```

```
##
##  Grubbs test for one outlier
##
## data:  df$newbalanceDest
## G = 96.60910, U = 0.99853, p-value < 2.2e-16
## alternative hypothesis: highest value 356179278.92 is an outlier
```

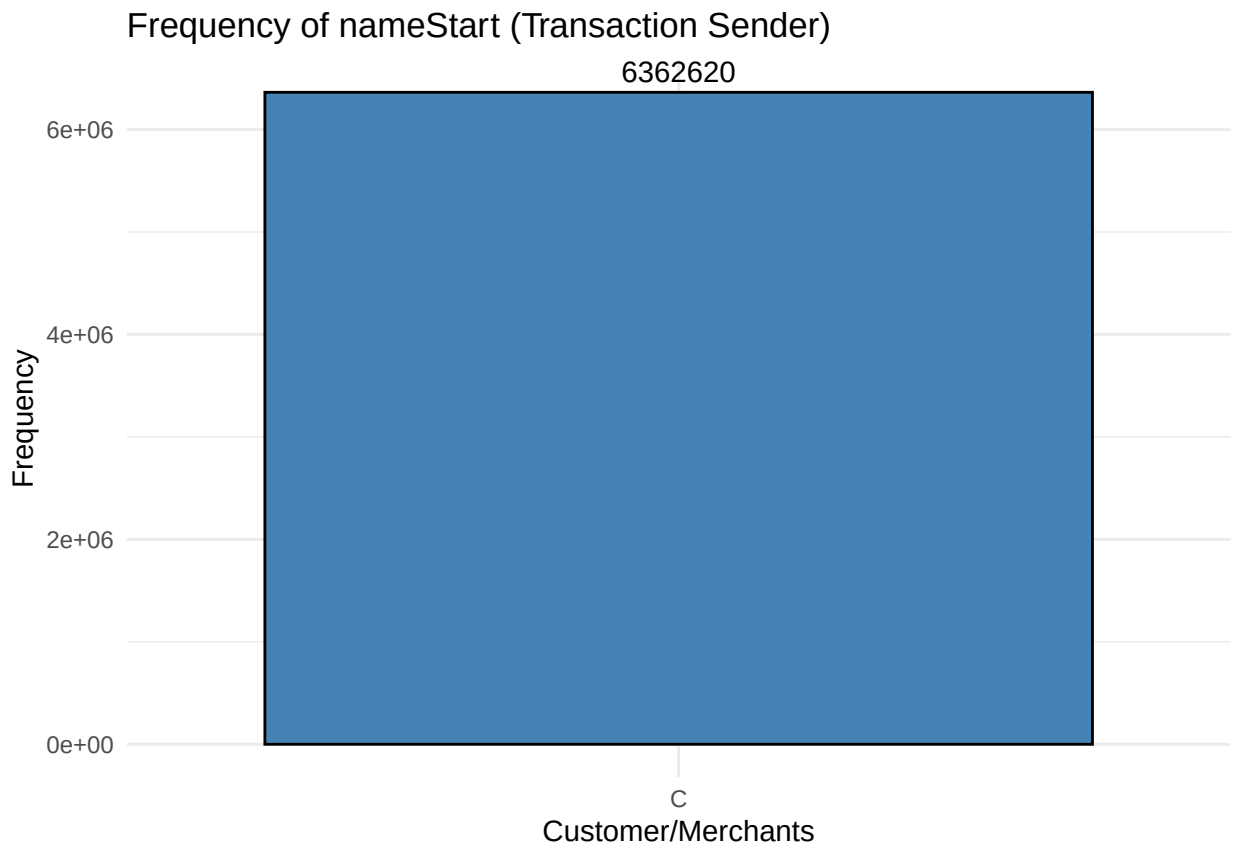
As we discussed previously the data has many outliers with extremely high numbers and we should not use the logistic regression algorithm to create a model since it is highly sensitive to outliers.

nameOrig (customer who started the transaction) and nameDest (customer who is the recipient of the transaction) start with C (Customer) or M (Merchants) letter. We can use this knowledge to create new features and use them for our model building.

```
df <- df %>%
  mutate(nameStart = str_sub(nameOrig, 1, 1), nameRecives = str_sub(nameDest, 1, 1))
```

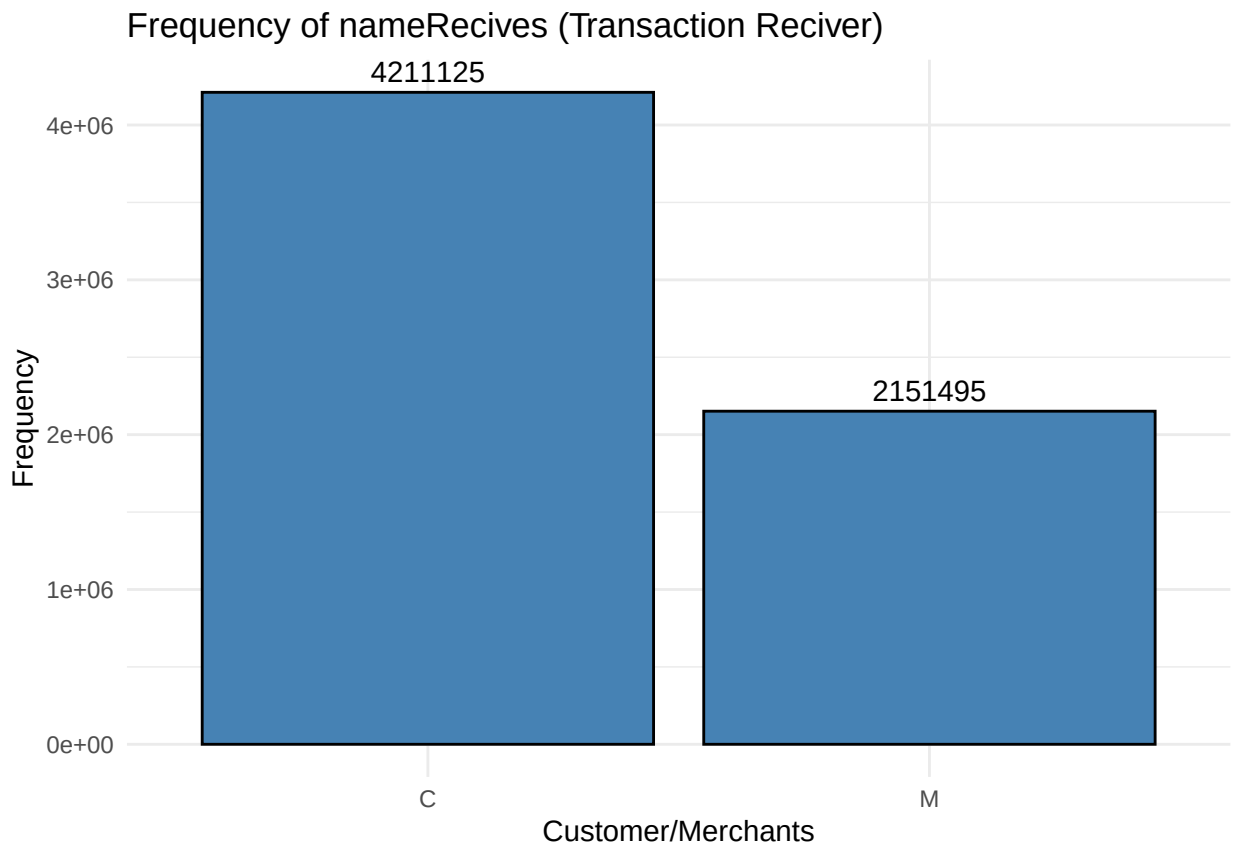
Now, let's plot both nameStart and nameRecives features.

```
ggplot(df, aes(x=nameStart, y =(..count..))) +
  geom_bar(fill="steelblue", color = "black") +
  theme_minimal() +
  geom_text(stat='count', aes(label=..count..), vjust=-0.5) +
  labs(x = "Customer/Merchants", y = "Frequency",
       title = "Frequency of nameStart (Transaction Sender)")
```



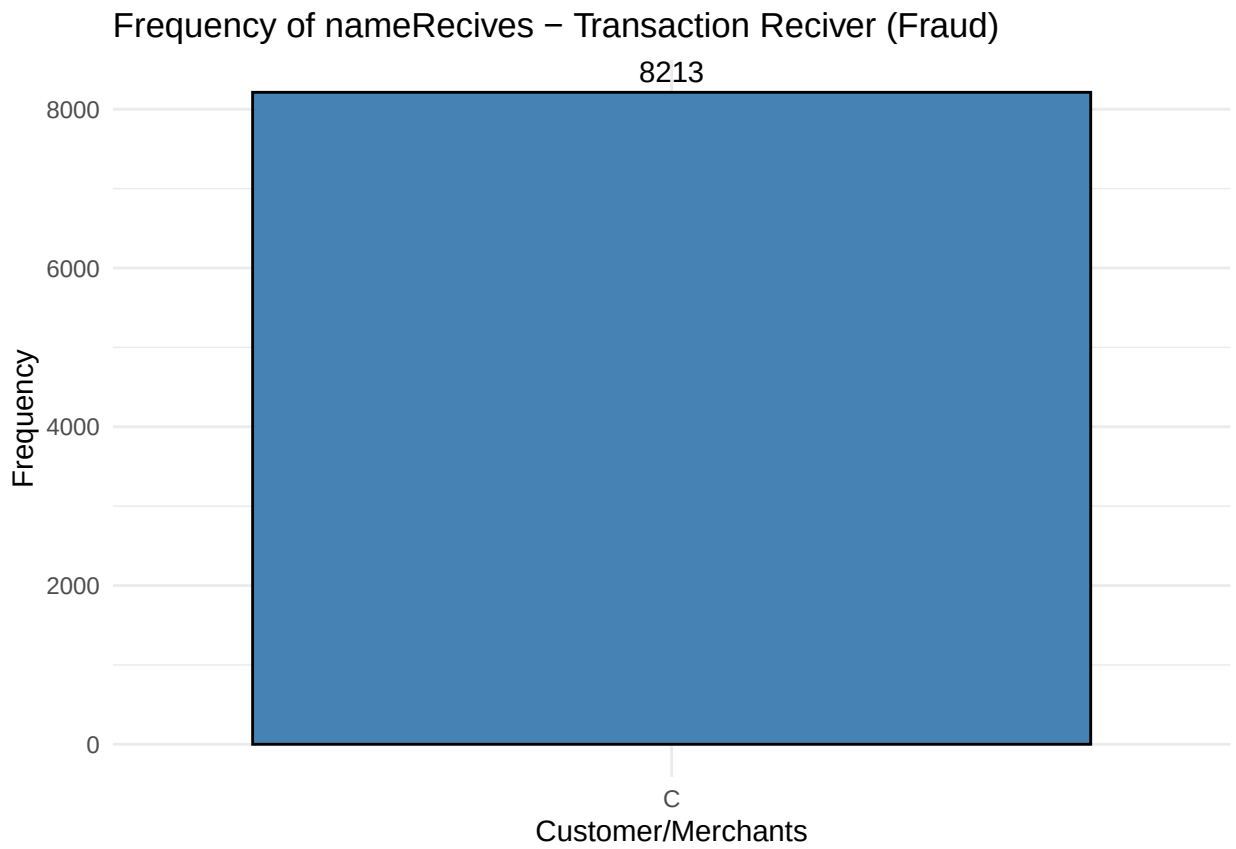
Interestingly all transactions were initiated by customers with zero merchant transactions and since there is no variance we can disregard this feature.

```
ggplot(df, aes(x=nameRecives, y =(..count..))) +
  geom_bar(fill="steelblue", color = "black") +
  theme_minimal() +
  geom_text(stat='count', aes(label=..count..), vjust=-0.5) +
  labs(x = "Customer/Merchants", y = "Frequency",
       title = "Frequency of nameRecives (Transaction Reciver)")
```



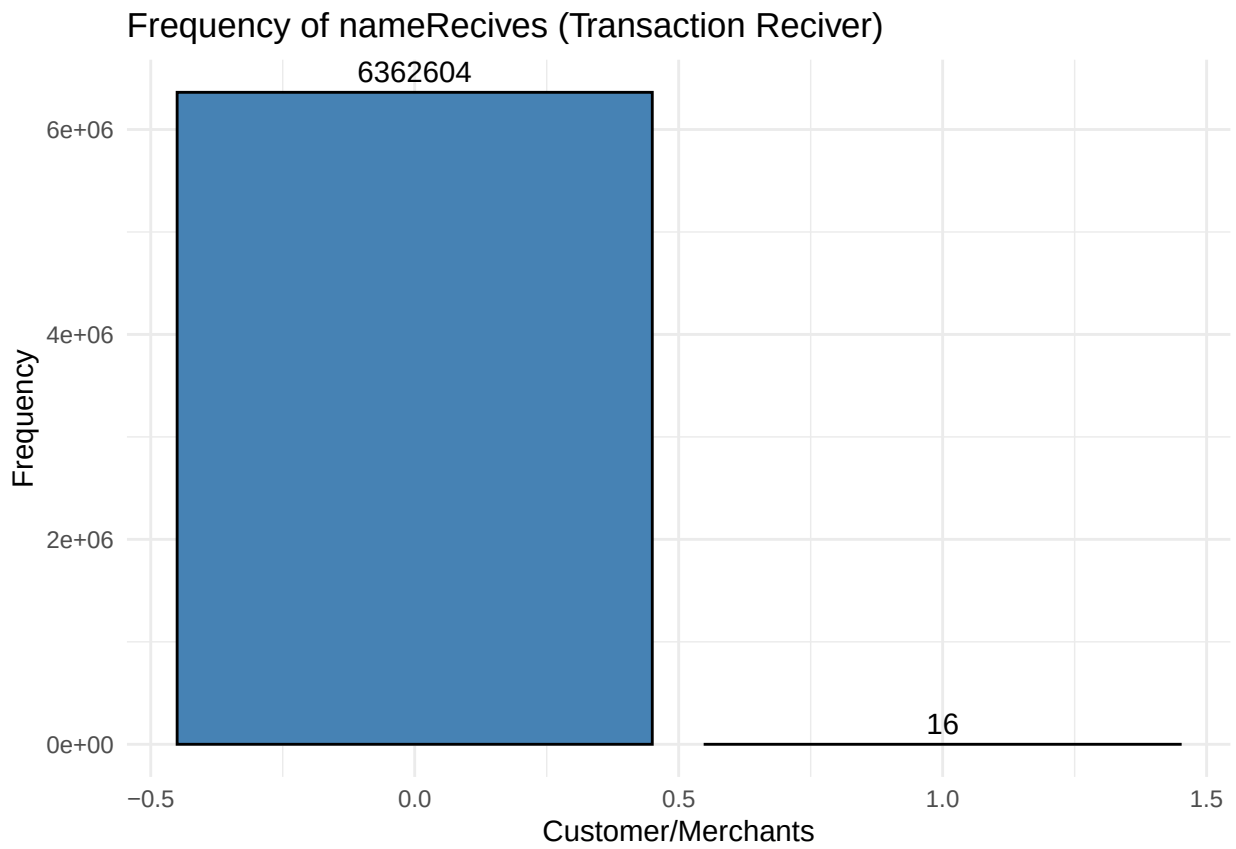
There is about a third of all transactions address to merchants. So the biggest porting of transactions were made between customers. It would be interesting to see what portion of received transactions were fraud so, we can do it by filtering only fraud transactions and plotting them.

```
df %>%
  filter(isFraud == 1) %>%
  ggplot(aes(x=nameRecives, y =(..count..))) +
    geom_bar(fill="steelblue", color = "black") +
    theme_minimal() +
    geom_text(stat='count', aes(label=..count..), vjust=-0.5) +
    labs(x = "Customer/Merchants", y = "Frequency",
         title = "Frequency of nameRecives - Transaction Reciver (Fraud)")
```



Again, very interesting observation, only transactions between customers were fraudulent and there is none fraudulent transaction between customers and merchants which looks quite unrealistic. There is another very interesting feature in the dataset **isFlaggedFraud**. This feature represents any transfers from one account to another with more than 200,000 in a single transaction. All these transactions were flags as illegal attempts. Let's now plot it and see its frequency.

```
ggplot(df, aes(x=isFlaggedFraud, y =(..count..))) +
  geom_bar(fill="steelblue", color = "black") +
  theme_minimal() +
  geom_text(stat='count', aes(label=..count..), vjust=-0.5) +
  labs(x = "Customer/Merchants", y = "Frequency",
       title = "Frequency of nameRecives (Transaction Reciver)")
```



Only 16 transactions were flagged as potentially fraudulent transactions, which is very strange as we know that there are a lot of transactions with over 200,000 per transaction. We can check this by plotting these transactions.

```
df %>%
  filter(amount > 200000) %>%
  summarise(transactions_over_200000 = n()) %>%
  knitr::kable()
```

transactions_over_200000
1673570

There are whooping 1,673,570 transactions with over 200,000 per transaction, so it is not very clear that this feature will be beneficial for the model building. Therefore we can drop this column along with nameOrig, nameDest and nameStart.

```
df <- df %>%
select(-c('nameDest', 'nameOrig', 'isFlaggedFraud', 'nameStart'))
```

In the next step, we can create extra features to display account balance differences by subtracting newbalanceOrig from oldbalanceOrg and newbalanceDest from oldbalanceDest. These features will provide information on the current status of the account. We are going to encode these new features with 1 for positive account balances, -1 for negative account balances and 0 from neutral account balances.

```
df <- df %>%
  mutate(balanceDifSender = if_else(newbalanceOrig - oldbalanceOrg > 0, 1, if_else(newbalanceOrig - oldb
```

```
df <- df %>%
  mutate(balanceDifReciver = if_else(newbalanceDest - oldbalanceDest > 0, 1, if_else(newbalanceDest - ol
```

Converting type and nameRecives to factor.

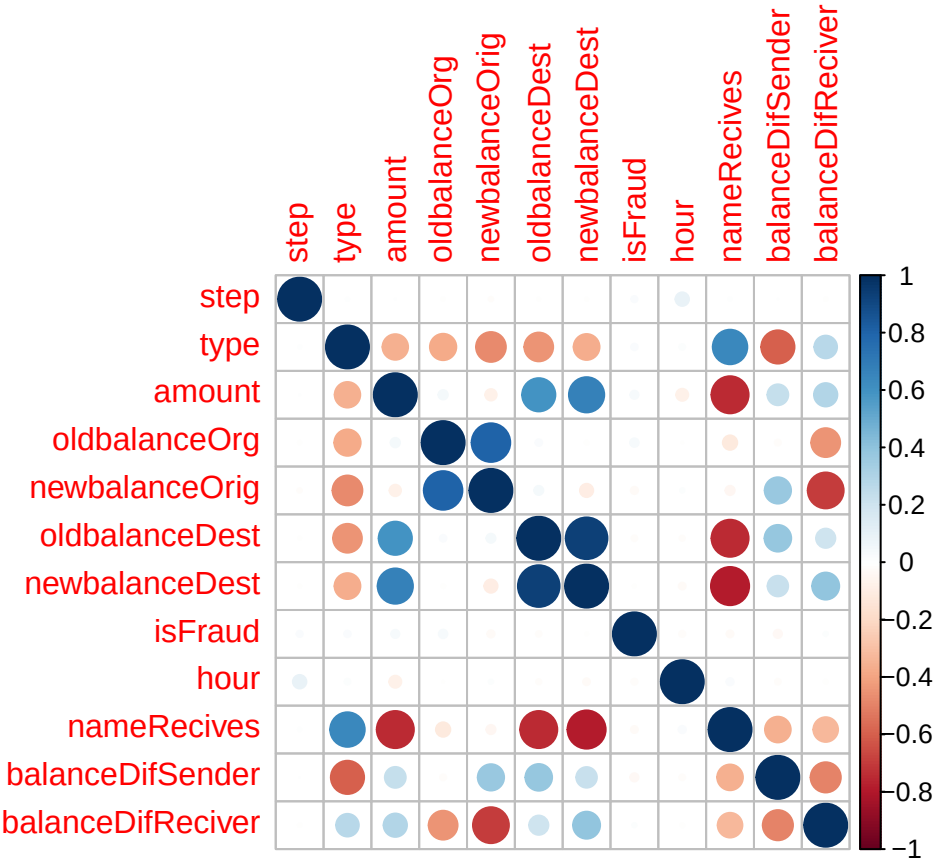
```
df$type<-as.factor(df$type)
df$nameRecives<-as.factor(df$nameRecives)
```

Now we are going to convert all categorical features into numerical since need to use only numerical values when building the models.

```
df <- df %>%
  mutate_if(is.factor, as.numeric)
```

Now we can create a correction matrix to display correlations between features.

```
df.cor <- cor(df, method = c("spearman"))
corrplot(df.cor)
```



It is worth noting that the isFraud dependent variable is not correlated with any other features therefore, we will not have an issue with autocorrelation. In addition, we can observe that there are a lot of variables with strong negative correlations.

Before building any models we are going to undersample the data. For the majority label, we will use a slightly higher ratio (extra 30%) just to make our training dataset a bit bigger.

```
fraud <- df %>% filter(isFraud == 1) %>% nrow()
not_fraud <- df %>% filter(isFraud == 0) %>% sample_n( fraud*1.3)
```

Creating a new undersampled dataset and average it by step to keep the original data sequence.

```
under_df <- df %>% filter(isFraud == 1) %>% rbind(not_fraud) %>% arrange(step)
```

The new dataset has around 19,000 of observations we believe that this amount will be enough to create sufficient models. Splitting the data into train, test and validation sets. We will be using the following ratio: train - 60%, test - 30% and validation - 10% of the original dataset. Train/test split is very close to 80/20 Pareto Principle rule that stays 80% of effects come from 20% of causes. THe validation set will be used to validate the final model.

```
train_part <- 0.60
test_part <- 0.30
validation_part <- 0.10
```

```
train_size <- floor(train_part * nrow(under_df))
test_size <- floor(test_part * nrow(under_df))
validation_size <- floor(validation_part * nrow(under_df))
```

```
train_size <- floor(train_part * nrow(under_df))
test_size <- floor(test_part * nrow(under_df))
validation_size <- floor(validation_part * nrow(under_df))
```

```
train_indices    <- sort(sample(seq_len(nrow(under_df)), size=train_size))
not_train_indices <- setdiff(seq_len(nrow(under_df)), train_indices)
validation_indices  <- sort(sample(not_train_indices, size=validation_size))
test_indices <- setdiff(not_train_indices, validation_indices)
```

```
train   <- under_df[train_indices, ]
validation <- under_df[validation_indices, ]
test       <- under_df[test_indices, ]
```

Model building

Decision Tree, Random Forest and XGBoost algorithms used for model building. Decision Tree was picked because it has the power to predict not linear data, requires less computational power and quite easy to explain the results and rationale of the model decisions. Random Forest was picked because it's a very powerful algorithm and can handle outliers with low overfitting risk. As the Decision Tree, it is quite efficient. XGBoost is a very famous and powerful boosting algorithm with a low probability to overfit, however, it can be sensitive to outliers and can act as a black box.

Decision Tree

For the decision tree model, 5-fold cross-validation was used to avoid overfilling since decision tree models are prone to overfilling. tuneLength parameter defines how many complexity parameters were used, we picked 10 and we are splitting the data on the information gain.

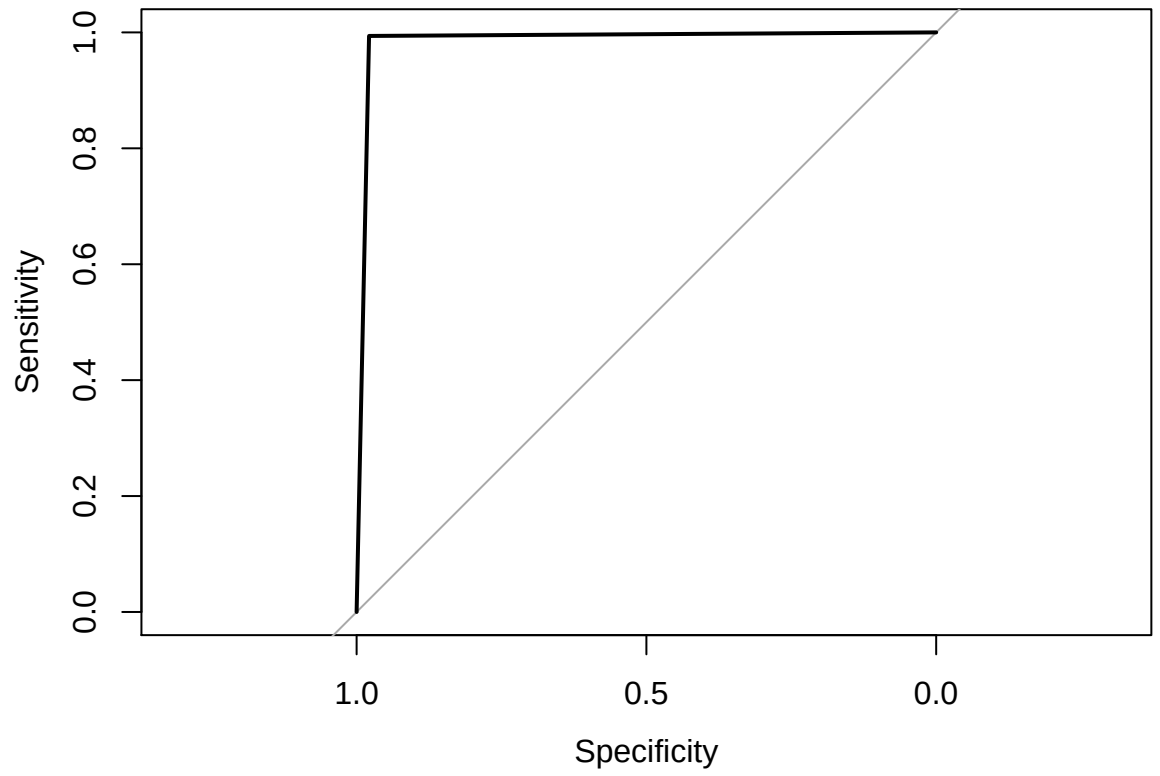
```
model_dt <- train(as.factor(isFraud) ~ .,
                  data = train, method = "rpart",
                  trControl = trainControl(method = "cv", number =5),
                      tuneLength = 10,
                      parms=list(split='information'))
```

Testing the decision tree model.

```
results_dt <- predict(model_dt , test)
roc_curve_dt <- roc(test$isFraud ~ as.numeric(results_dt))
roc_curve_dt
```

```
##
## Call:
## roc.formula(formula = test$isFraud ~ as.numeric(results_dt))
##
## Data: as.numeric(results_dt) in 3211 controls (test$isFraud 0) < 2457 cases (test$isFraud 1).
## Area under the curve: 0.9862
```

```
plot(roc_curve_dt)
```

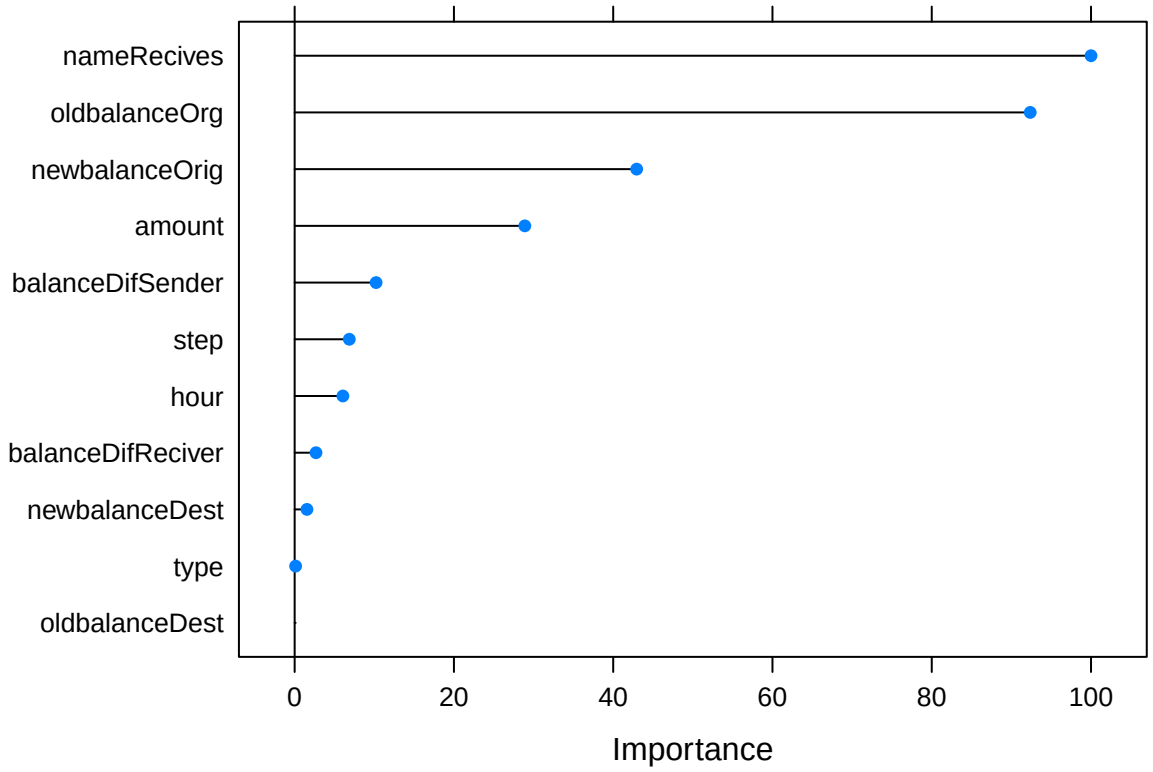


The decision tree model achieved a great area under the curve score of 0.9837. Next let's create a random forest model.

Random Forest

The same 5-fold cross-validation was used with the number of trees equal to 5. We also check the importance of variables for this model.

```
model_rf <- train(as.factor(isFraud) ~ .,
                      data = train,
                      trControl = trainControl(method = "cv", number =5),
                      importance=TRUE,
                      method="rf",ntree=5)
#Variable importance chart
plot(varImp(model_rf))
```

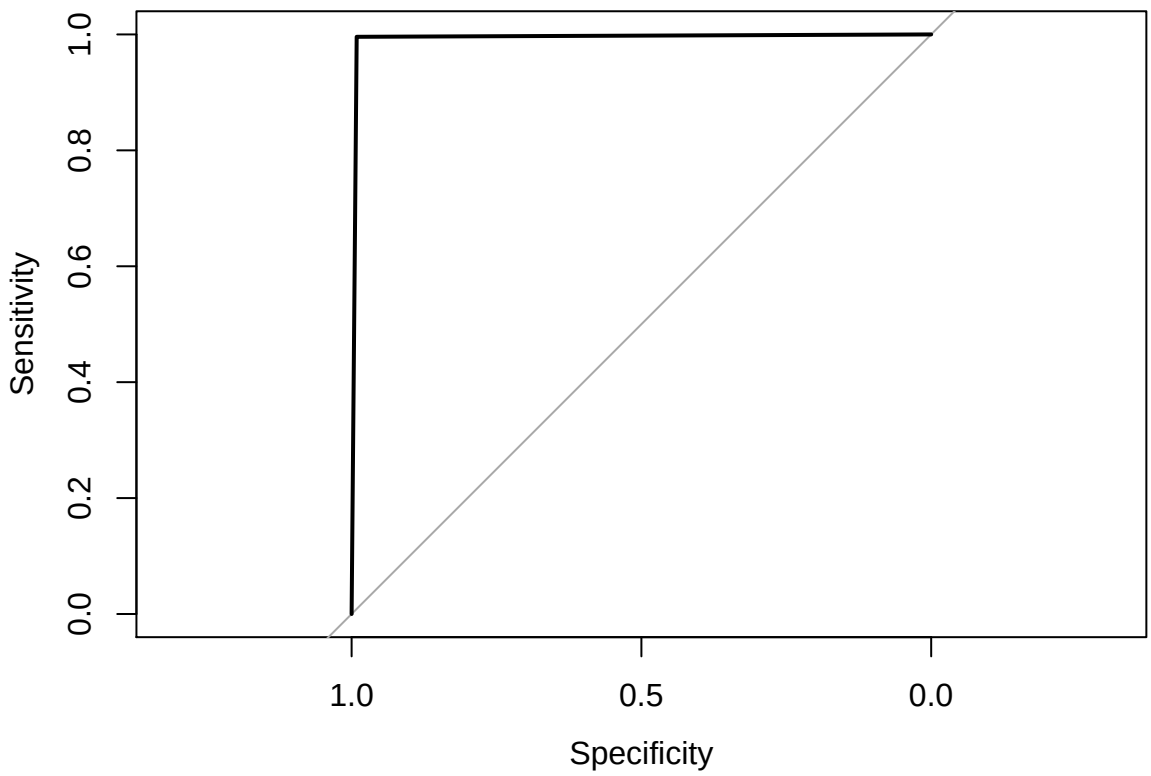


nameRecivers, oldbalanceOrg, amount and newbalanceOrig features played an important role in detecting fraudulent transactions. We can simplify the model using just these four features and still achieve great model accuracy.

```
results_rf <- predict(model_rf , test)
roc_curve_rf <- roc(test$isFraud ~ as.numeric(results_rf))
roc_curve_rf
```

```
##
## Call:
## roc.formula(formula = test$isFraud ~ as.numeric(results_rf))
##
## Data: as.numeric(results_rf) in 3211 controls (test$isFraud 0) < 2457 cases (test$isFraud 1).
## Area under the curve: 0.9936
```

```
plot(roc_curve_rf)
```



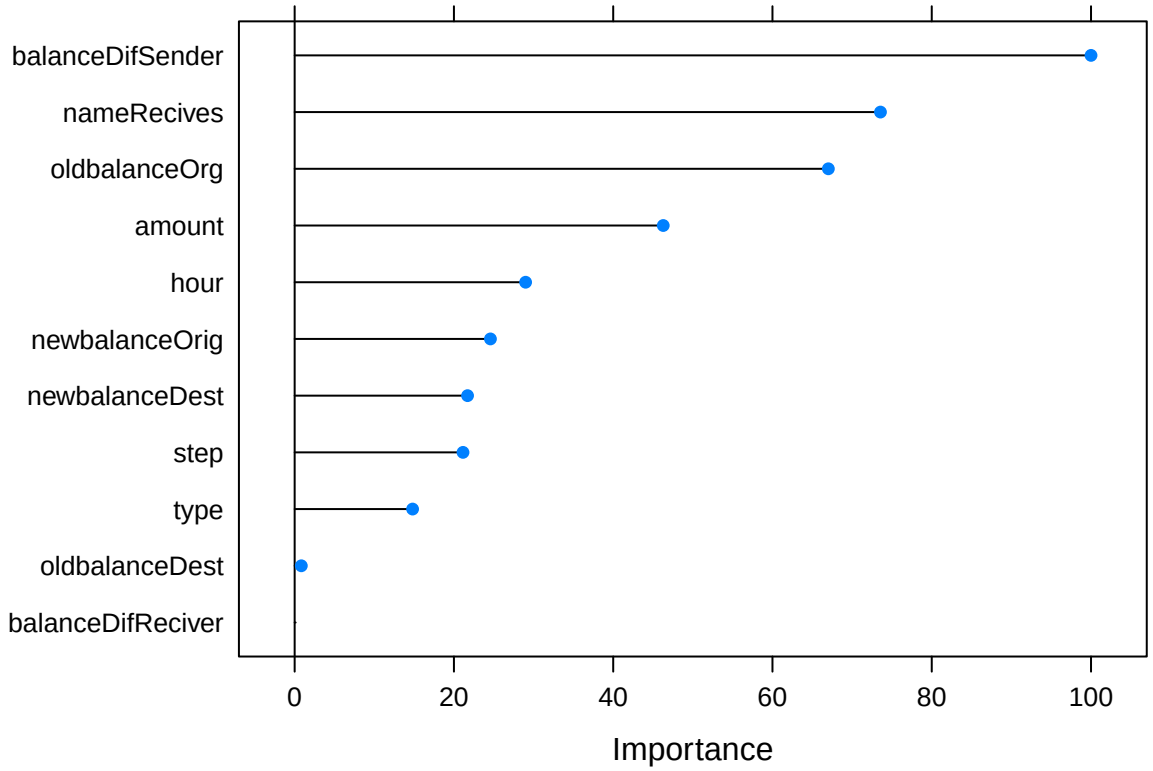
The Random Forest model achieved a better area under the curve score of 0.9937. Next, let's create and test the next model XGBoost.

XGBoost

```
model_xgb <- train(
  as.factor(isFraud) ~ .,  data = train, method = "xgbTree",
  trControl = trainControl("cv", number = 5)
  )
```

After we built the model we can check the feature importance of each variable.

```
#Variable importance chart
plot(varImp(model_xgb))
```

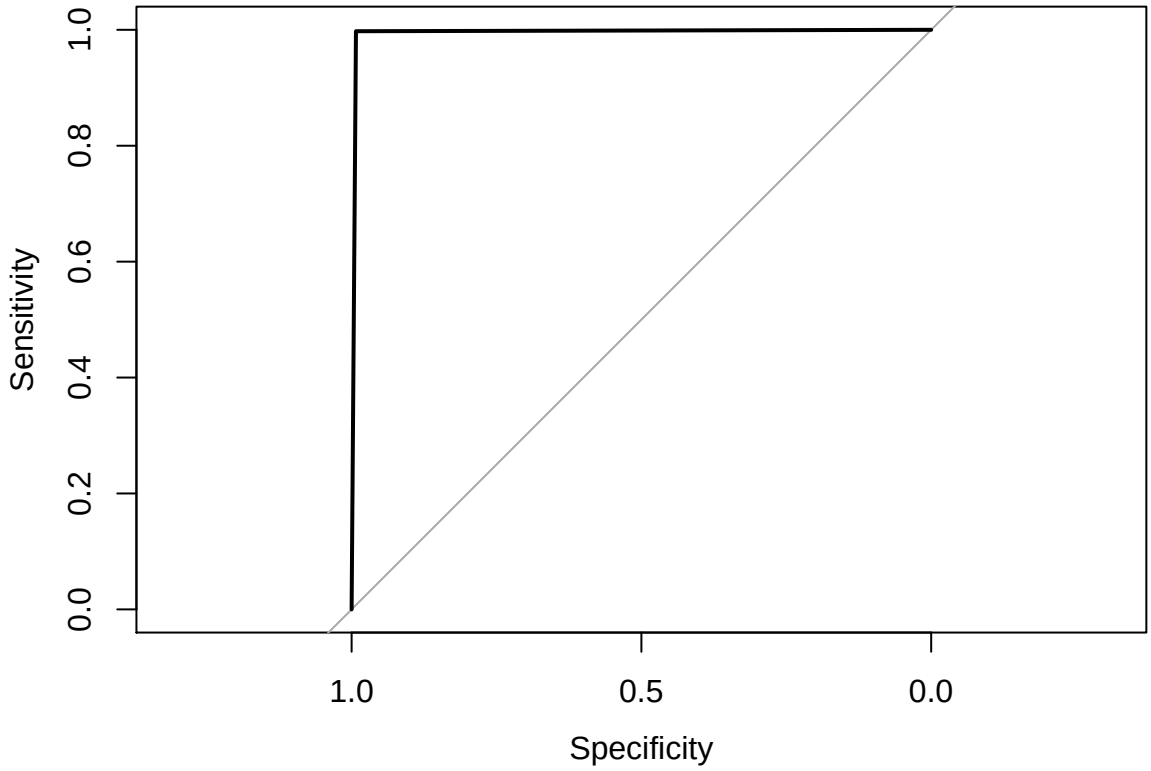


Interestingly xgboost model used different features and the most important are balanceDifSender, oldbalanceOrg and amount. Again we can simplify the model using just a few features and still achieve exceptional results.

```
results_xgb <- predict(model_xgb , test)
roc_curve_xgb <- roc(test$isFraud ~ as.numeric(results_xgb))
roc_curve_xgb
```

```
##
## Call:
## roc.formula(formula = test$isFraud ~ as.numeric(results_xgb))
##
## Data: as.numeric(results_xgb) in 3211 controls (test$isFraud 0) < 2457 cases (test$isFraud 1).
## Area under the curve: 0.995
```

```
plot(roc_curve_xgb)
```

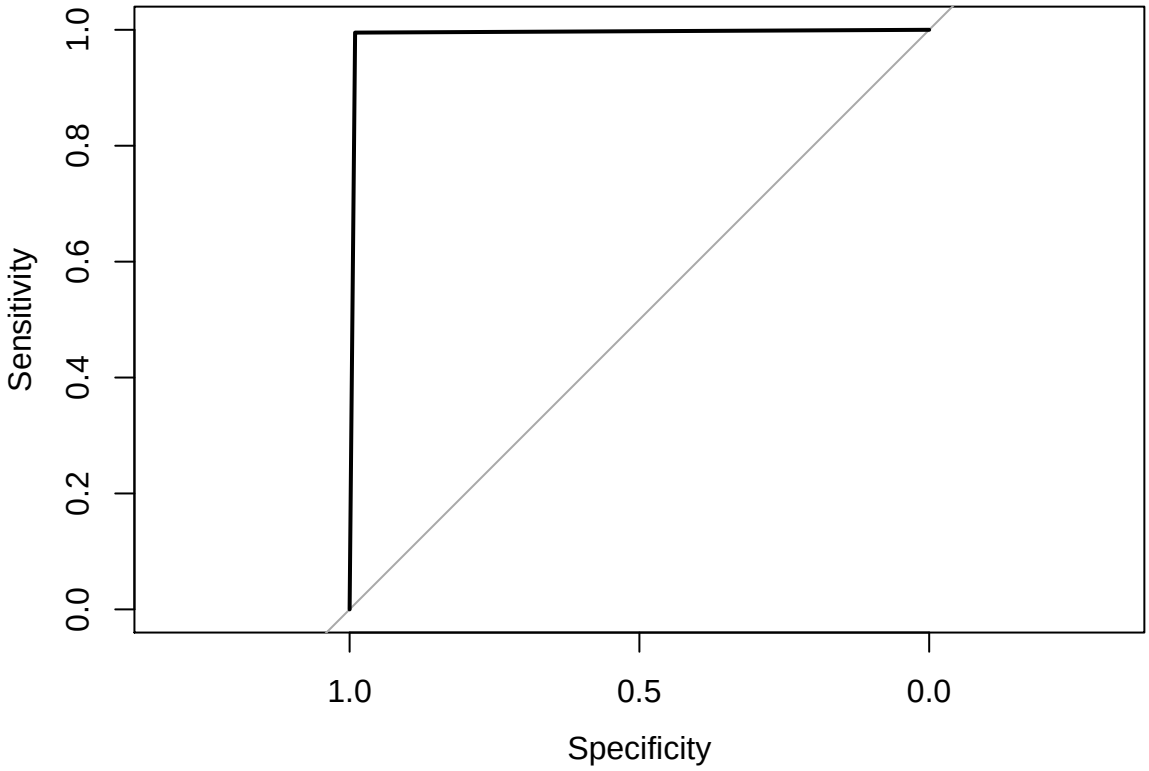


The XGBoost model achieved comparable results (Area under the curve: 0.9939) very close to the Random Forest model. However, the XGBoost model is significantly slower(more computationally intensive) and more complex compare to the Random Forest model. Thus, we will pick the Random Forest model as our preferable model and evaluate it with the validation data to see if it achieves comparable results.

```
results <- predict(model_rf , validation)
roc_curve<- roc(validation$isFraud ~ as.numeric(results))
roc_curve
```

```
##
## Call:
## roc.formula(formula = validation$isFraud ~ as.numeric(results))
##
## Data: as.numeric(results) in 1059 controls (validation$isFraud 0) < 829 cases (validation$isFraud 1)
## Area under the curve: 0.9929
```

```
plot(roc_curve)
```



After the validation, the model achieved exceptional results with the area under the curve of 0.991. Next, we create a confusion matrix to see some statistics of the model. The model produces great Sensitivity : 0.9906 and Specificity 0.9915 results, meaning only a small portion of transactions were misclassified.

```
xtab <- table(results, validation$isFraud)
confusionMatrix(xtab)
```

```
## Confusion Matrix and Statistics
##
##
## results    0    1
##       0 1049    4
##       1   10  825
##
##                Accuracy : 0.9926
##                  95% CI : (0.9876, 0.9959)
##     No Information Rate : 0.5609
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.985
##
##  Mcnemar's Test P-Value : 0.1814
##
##             Sensitivity : 0.9906
##             Specificity : 0.9952
##          Pos Pred Value : 0.9962
##          Neg Pred Value : 0.9880
```

```
##              Prevalence : 0.5609
##          Detection Rate : 0.5556
##    Detection Prevalence : 0.5577
##       Balanced Accuracy : 0.9929
##
##        'Positive' Class : 0
##
```

Resutls

All three models provided great results and can be used by businesses to create a functional model to predict fraudulent transactions. The Random Forest model was selected as the best option since it provides results comparable to the XGBoost model, but works faster and offers better explainability. The model achieved Accuracy of 0.991, Sensitivity of 0.9906 and Specificity of 0.9915 on the validation dataset.

Conclusion

The importance of detecting fraud is very crucial for financial institutions because it can significantly reduce loss and improve the customer experience for clients. However, we need to remember the scalability and explainability of models. The model should predict fast enough and not cause any service interruptions. In this project, we explore the transnational data, feature engineered numerous variables and built competitive models.

Future work

Even though the achieved results were great they can be improved further using hyperparameter tunning with the current models or creating completely new more advanced models using other bagging and stacking algorithms.

Limitations

I was not able to use the whole dataset with SMOTE and Oversampling data balancing methods since my hardware has very limited processing power.

References

Annavictoria. (2018, May 18). ML-friendly Public Datasets. Retrieved December 28, 2020, from https://www.kaggle.com/annavictoria/ml-friendly-public-datasets

Kassambara. (2018, March 10). Gradient Boosting Essentials in R Using XGBOOST. Retrieved December 28, 2020, from http://www.sthda.com/english/articles/35-statistical-machine-learning-essentials/139-gradient-boosting-essentials-in-r-using-xgboost/

Max Kuhn [aut, C. (2020, March 20). ConfusionMatrix: Create a confusion matrix in caret: Classification and Regression Training. Retrieved December 28, 2020, from https://rdrr.io/cran/caret/man/confusionMatrix.html

Narkhede, S. (2019, May 26). Understanding AUC - ROC Curve. Retrieved December 28, 2020, from https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Detective, T. (2020, January 31). Finally: Why We Use an 80/20 Split for Training and Test Data Plus an Alternative Method (Oh Yes...). Retrieved December 29, 2020, from https://towardsdatascience.com/finally-why-we-use-an-80-20-split-for-training-and-test-data-plus-an-alternative-method-oh-yes-edc77e96295d