Fraud Detection

Esteban Emilio Rodriguez

```
library(tidyverse)
library(caret)
library(rpart)
library(rpart.plot)
library(randomForest)
library(stringr)
```

1. Introduction

This project is part of the last course of the **EDX/HarvardX Data Science Specialization**. It consists in the creation of a model to detect **fraudulent transactions** from a set of mostly valid transactions. It uses the **Synthetic Financial Datasets For Fraud Detection** available at https://www.kaggle.com/ntnu-testimon/paysim1 (https://www.kaggle.com/ntnu-testimon/paysim1)

This dataset is too big to be upload to Github. In case you are trying to reproduce the exercise, you should download it and unzip it on your local drive. The other files of this project are available at: https://github.com/esterodr/fraud (https://github.com/esterodr/fraud)

- fraud_report.pdf: the final report with all the analysis
- fraud_report.Rmd: the R markdown file that create the html version of the report
- fraud_final_model.R: the code that trains and run the final model. This code is also included in this report. On the other side, the "R" file doesn't contain the analysis and graphs included here.
- Files that could help to execute faster the code of this report: "Fs.Rda", "DT_1.Rda", "RF_1.Rda", "TD_f.Rda", "RF_f.Rda".

The agenda of the report is the following. In Section 2 I will describe the dataset and perform some exploratory analysis in order to gain some insights in the fraud detection problem. In Section 3 I will explain the metrics I'm going to use to evaluate the model. In Section 4 I will train the model for prediction, tuning the required parameters. The final model will be trained and evaluated in Section 5. Section 6 presents the main conclusions of this project.

2. Database description and analysis

The dataset downloaded from *kaggle* is scaled down 1/4 of the original dataset which is presented in the paper *"PaySim: A financial mobile money simulator for fraud detection"* (https://www.researchgate.net/publication/313138956_PAYSIM_A_FINANCIAL_MOBILE_MONEY_SIMULATOR_FOR_FRAUD_DETECTION (https://www.researchgate.net/publication/313138956_PAYSIM_A_FINANCIAL_MOBILE_MONEY_SIMULATOR_FOR_FRAUD_DETECTION)).

It can be downloaded from the *kaggle* link provided at the first paragraph of the report. As the download and reading of the csv file takes time, I will assume that the database is already saved at the local disk in an R object called "fraud_data.Rda". In case that you are trying to reproduce the analysis, I suggest you download the file, uncompressing it, and make sure that the resulting csv file has the name "PS_20174392719_1491204439457_log.csv". Then you can execute the following code:

```
if(file.exists("fraud_data.Rda")) {
  load("fraud_data.Rda")
} else {
  data <- read_csv("PS_20174392719_1491204439457_log.csv")
  save(data, file="fraud_data.Rda")
}
```

According to the author of the dataset (*Edgar Lopez-Rojas*), there is a lack of public available datasets on financial services due to the intrinsically private nature of financial transactions. The **Synthetic Financial Datasets For Fraud Detection** is a dataset generated by the PaySim mobile money simulator, which uses aggregated data from private datasets to generate a synthetic dataset that resembles the normal operation of transactions and injects malicious behavior to later evaluate the performance of fraud detection methods. In particular, PaySim simulates mobile money transactions based on a sample of real transactions extracted from **one month of financial logs** from a mobile money service implemented in an African country. The original logs were provided by a multinational company, who is the provider of the mobile financial service which is currently running in more than 14 countries all around the world.

The dataset consists in 6362620 observation and 11 variables. As I'm trying to design a model for fraud prediction, I will split the dataset in a **working set (90%) and a validation set (10%)**. I will pretend to not have access to the *validation set* until Section 5, where I will use it to evaluate the final model.

```
set.seed(1)
test_index <- createDataPartition(y=data$isFraud, times=1,p=0.9,list=FALSE)
working_set <- data[test_index,]
validation_set <- data[-test_index,]
rm(test_index, data)
```

Let's take a look at the *working_set*:

```
head(working_set)
```

```
## # A tibble: 6 x 11
##    step type  amount nameOrig oldbalanceOrg newbalanceOrig nameDest
##   <dbl> <chr>  <dbl> <chr>            <dbl>          <dbl> <chr>
## 1     1 PAYM~  9840. C123100~        170136        160296. M197978~
## 2     1 PAYM~  1864. C166654~         21249         19385. M204428~
## 3     1 CASH~   181  C840083~           181             0  C389970~
## 4     1 PAYM~ 11668. C204853~         41554         29886. M123070~
## 5     1 PAYM~  7818. C900456~         53860         46042. M573487~
## 6     1 PAYM~  7108. C154988~        183195        176087. M408069~
## # ... with 4 more variables: oldbalanceDest <dbl>, newbalanceDest <dbl>,
## #   isFraud <dbl>, isFlaggedFraud <dbl>
```

```
str(validation_set)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    636262 obs. of  11 variables:
##  $ step          : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ type          : chr  "TRANSFER" "PAYMENT" "PAYMENT" "PAYMENT" ...
##  $ amount        : num  181 1373 8009 4207 24214 ...
##  $ nameOrig      : chr  "C1305486145" "C20804602" "C1053967012" "C215078753" ...
##  $ oldbalanceOrg : num  181 13854 10968 0 0 ...
##  $ newbalanceOrig: num  0 12481 2959 0 0 ...
##  $ nameDest      : chr  "C553264065" "M1344519051" "M295304806" "M1757317128" ...
##  $ oldbalanceDest: num  0 0 0 0 0 ...
##  $ newbalanceDest: num  0 0 0 0 0 ...
##  $ isFraud       : num  1 0 0 0 0 0 0 0 0 0 ...
##  $ isFlaggedFraud: num  0 0 0 0 0 0 0 0 0 0 ...
```

At the *kaggle* website we can find the following description of the variables:

step: Maps a unit of time in the real world. In this case 1 step is 1 hour of time.

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: customer's balance after the transaction.

nameDest: recipient ID of the transaction.

oldbalanceDest: initial recipient balance before the transaction.

newbalanceDest: recipient's balance after the transaction.

isFraud identifies: a fraudulent transaction (1) and non fraudulent (0).

isFlaggedFraud flags: illegal attempts to transfer more than 200.000 in a single transaction.

The variables *isFraud* and *isFlaggedFraud* are numerical, even though they are categorical. I will convert them to factors, and I'm going to do the same with the character variable *type*:

```
working_set$type <- as.factor(working_set$type)
working_set$isFraud <- as.factor(working_set$isFraud)
working_set$isFlaggedFraud <- as.factor(working_set$isFlaggedFraud)
```

I'm trying to predict the *isFraud* variable, so I'm going to explore in more detail this variable and its possible determinants. Let's see how many fraudulent transactions are in the database:

```
table(working_set$isFraud)
```

```
##
##       0       1
## 5718996    7362
```

```
table(working_set$isFraud)[2]/nrow(working_set)
```

```
##           1
## 0.001285634
```

Only 7,362 (0.13%) of the transactions are fraudulent. Let's explore the types of transactions:

```
table(working_set$type)
```

```
##
##  CASH_IN CASH_OUT    DEBIT  PAYMENT TRANSFER
##  1259383  2013640    37185  1936362   479788
```

```
working_set %>%
  select(type) %>%
  mutate(N=n()) %>%
  group_by(type) %>%
  summarise(p = n()/first(N))
```

```
## # A tibble: 5 x 2
##   type           p
##   <fct>      <dbl>
## 1 CASH_IN  0.220
## 2 CASH_OUT 0.352
## 3 DEBIT    0.00649
## 4 PAYMENT  0.338
## 5 TRANSFER 0.0838
```

Around 90% of the transactions are of one of three types: CASH_OUT, PAYMENT and CASH_IN. Let's see if in some of these types, frauds are more frequent:

```
working_set %>%
  select(type, isFraud) %>%
  group_by(type, isFraud) %>%
  summarise(n=n()) %>%
  spread(isFraud, n)
```

```
## # A tibble: 5 x 3
## # Groups:   type [5]
##   type         `0`    `1`
##   <fct>      <int> <int>
## 1 CASH_IN  1259383    NA
## 2 CASH_OUT 2009938  3702
## 3 DEBIT      37185    NA
## 4 PAYMENT  1936362    NA
## 5 TRANSFER  476128  3660
```

Here is a highly relevant insight: *frauds only seem to occur in TRANSFERS or in CASH_OUT transactions*. As a result, I can formulate a first rule for my model:

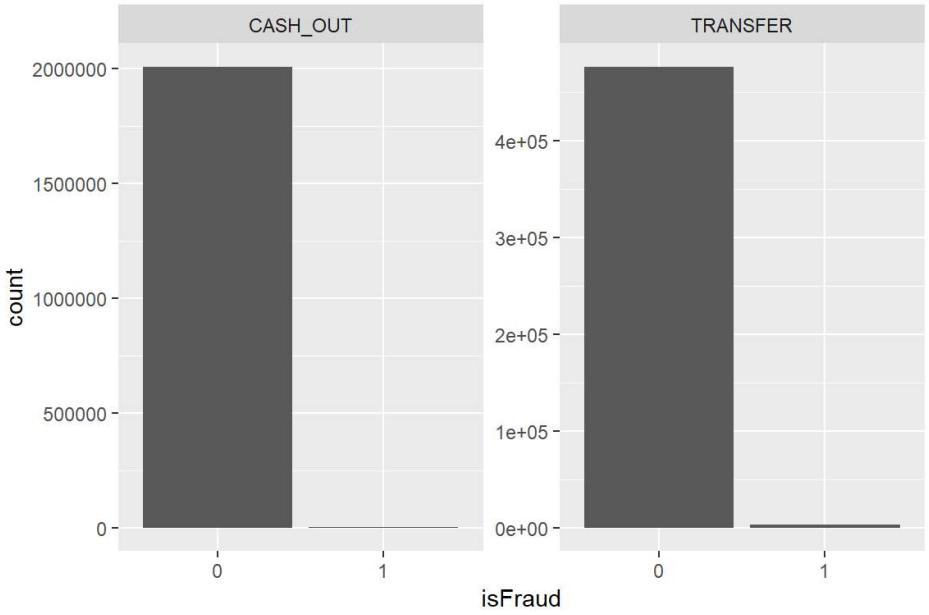
If the type of a transactions is either PAYMENT, CASH_IN or DEBIT, the model should predict that it is a valid transaction (non-fraudulent)

Considering this, I will focus the search for frauds only in TRANSFERS or CASH_OUT transactions. I will filter the working set:

```
working_set <- working_set %>%
  filter(type %in% c("TRANSFER", "CASH_OUT")) %>%
  mutate(type = as.factor(as.character(type)))
```

Let's see how often frauds in this kind of transactions are:

```
working_set %>%
  select(type, isFraud) %>%
  group_by(type) %>%
  ggplot(aes(x=isFraud)) +
  geom_bar() +
  facet_wrap(. ~ type, scales = "free")
```

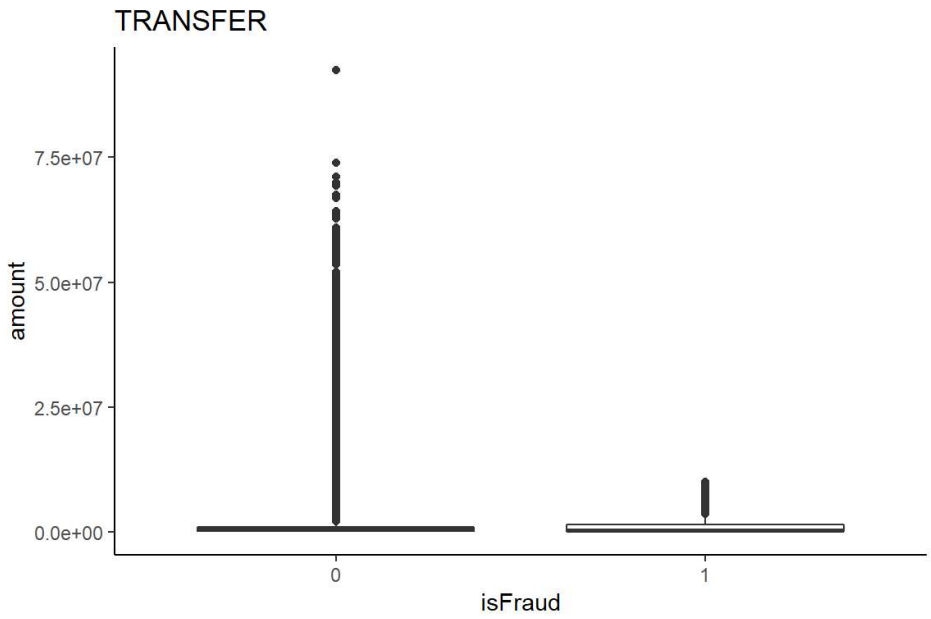



```
working_set %>%
  select(type, isFraud) %>%
  group_by(type) %>%
  mutate(N=n()) %>%
  ungroup() %>%
  group_by(type, isFraud) %>%
  summarise(p = n()/first(N)) %>%
  ungroup() %>%
  spread(isFraud, p)
```

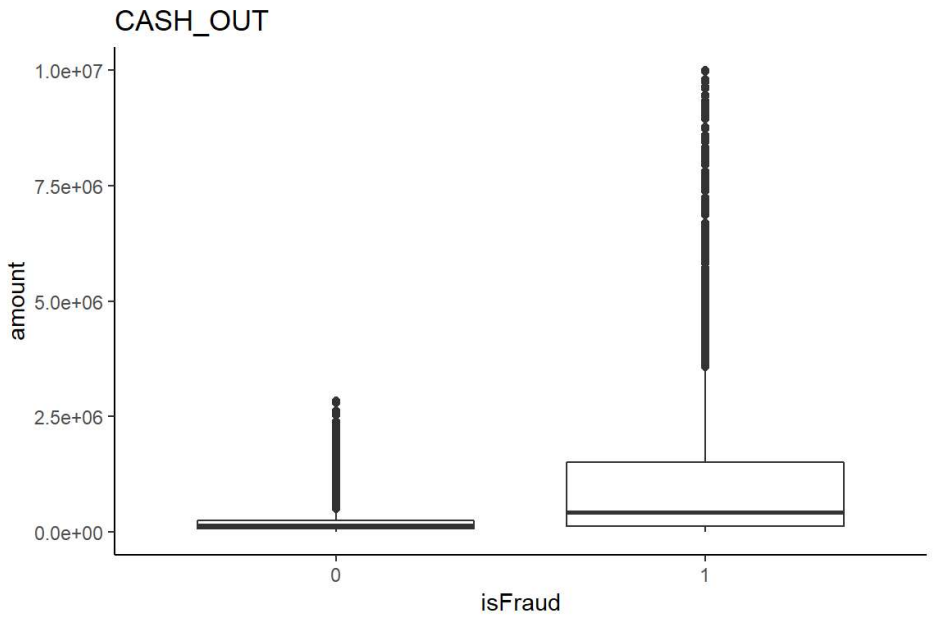
```
## # A tibble: 2 x 3
##   type       `0`     `1`
##   <fct>    <dbl>   <dbl>
## 1 CASH_OUT 0.998 0.00184
## 2 TRANSFER 0.992 0.00763
```

It looks like frauds are slightly more probable in transfers. Regarding *amount*, is there any difference in the amount of fraudulent and no-fraudulent transactions?

```
working_set %>%
  filter(type=="TRANSFER") %>%
  ggplot(aes(x=isFraud, y=amount)) +
  geom_boxplot() +
  ggtitle("TRANSFER") +
  theme_classic()
```




```
working_set %>%
  filter(type=="CASH_OUT") %>%
  ggplot(aes(x=isFraud, y=amount)) +
  geom_boxplot() +
  ggtitle("CASH_OUT") +
  theme_classic()
```




```
working_set %>%
  select(type, isFraud, amount) %>%
  group_by(type, isFraud) %>%
  summarise(mean_amount = mean(amount), median_amount = mean(amount))
```

```
## # A tibble: 4 x 4
## # Groups:   type [?]
##   type     isFraud mean_amount median_amount
##   <fct>    <fct>         <dbl>         <dbl>
## 1 CASH_OUT 0           173894.       173894.
## 2 CASH_OUT 1          1470474.      1470474.
## 3 TRANSFER 0           906622.       906622.
## 4 TRANSFER 1          1509599.      1509599.
```

In both type of transactions, the fraudulent ones have a higher mean/median amount. But this is more evident in *cash_out* transactions, where the mean amount is more than 8 times the mean amount of valid *cash_out* transactions, while in *transfers* the factor is around 1.7. We also must note that, in the case of *transfers*, there are more outliers on the right side of the distributions, that is, valid transactions of very high amount.

Exploring the variables *oldbalanceOrg, newbalanceOrig, oldbalanceDest and newbalanceDest*, it seems that **there are some inconsistencies in the data.** For example, in a regular transaction I would expect the following condition to hold:

$$newbalanceOrig = oldbalanceOrig - amount$$

That is, if the one who started the transaction had $1000 in her account and transferred or cashed out an amount of $300, she should have now $700 in her account. However, this condition not always holds in the dataset, as we can see in the following rows:

```
working_set %>%
  filter(newbalanceOrig != (oldbalanceOrg-amount)) %>%
  head()
```

```
## # A tibble: 6 x 11
##    step type  amount nameOrig oldbalanceOrg newbalanceOrig nameDest
##   <dbl> <fct>  <dbl> <chr>            <dbl>          <dbl> <chr>
## 1     1 CASH~ 2.29e5 C905080~        15325              0 C476402~
## 2     1 TRAN~ 2.15e5 C167099~          705              0 C110043~
## 3     1 TRAN~ 3.12e5 C198409~        10835              0 C932583~
## 4     1 CASH~ 1.10e5 C768216~        26845.             0 C150951~
## 5     1 CASH~ 5.70e4 C157047~         1942.             0 C824009~
## 6     1 CASH~ 5.35e3 C512549~            0              0 C248609~
## # ... with 4 more variables: oldbalanceDest <dbl>, newbalanceDest <dbl>,
## #   isFraud <fct>, isFlaggedFraud <fct>
```

Moreover, sometimes the *amount* of the transactions is higher than the *oldbalance* of the origin. Looking for an explanation for this problem in the *kaggle* forums, I found many open questions regarding it, with no a proper explanation. However, there are two answers of the author of the dataset that could help to understand more of it:

"…As I mentioned earlier the mobile money system is a way to digitally send money. However, like any other debit accounts you also have the possibility to withdraw cash or deposit cash. These operation are only possible using a merchant (or kiosk) the client gives digital money to merchants and they give back cash in return for CASHOUT, similar to this is CASHIN operation. The model we built uses M to denote Merchants and C to denote clients. Merchants balance is not so important for our model so we focus more on Customers balance (old balance and new balance). Each transaction affects the 2 parties involve in the transaction." (https://www.kaggle.com/ntnu-testimon/paysim1/discussion/32786#latest-183354 (https://www.kaggle.com/ntnu-testimon/paysim1/discussion/32786#latest-183354))

"…when the record has the isFlaggedFraud = 1, this means that the transaction was detected and stopped from being processed, perhaps that is the reason why it didn't affect the account destination/origin (previous value). Please notice that there is not record of balance from clients that start with M (Merchants)" (https://www.kaggle.com/ntnu-testimon/paysim1/discussion/33129#latest-187012 (https://www.kaggle.com/ntnu-testimon/paysim1/discussion/33129#latest-187012))

However, in the rows showed before there are no Merchants involved (both "nameOrig" and "nameDest" starts with "C" from "Client"), and there are no *frauds* or *flaggedfrauds*. As I couldn't find a satisfactory explanation for this problem, I will explore in more detail these errors.

One particular kind of error that seems relevant is when both the origin and destination account are 0 (and the amount of the transaction is not 0):

```
working_set %>%
  group_by(isFraud) %>%
  summarise(p_zeros_orig = mean(oldbalanceOrg==0 & newbalanceOrig ==0 & amount!=0),
            p_zeros_dest = mean(oldbalanceDest==0 & newbalanceDest ==0 & amount!=0))
```

```
## # A tibble: 2 x 3
##   isFraud p_zeros_orig p_zeros_dest
##   <fct>          <dbl>        <dbl>
## 1 0            0.474       0.000620
## 2 1            0.00285     0.493
```

We can see that **in more than 49% of fraudulent transactions, both new and old account of the destination are zero, while less than 0.1% of valid transactions has this kind of error. An inverse patter is visible when both new and old account of the originator are zero.** As these kind of error seems relevant in order to predict fraud, I will construct two dummy variables to signal the occurrence of this error. Furthermore, I will compute the error in any transaction as it could also be relevant for prediction.

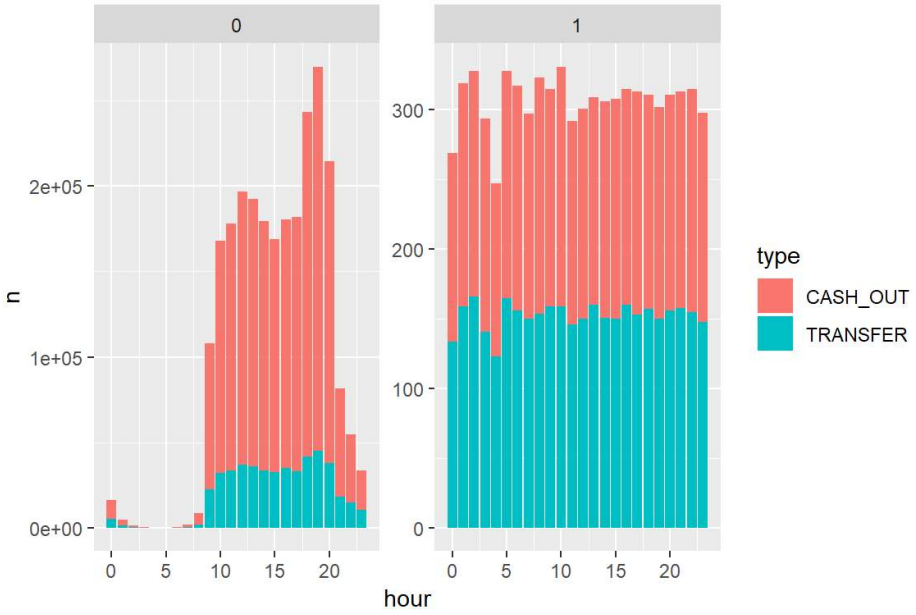
```
working_set <- working_set %>%
  mutate(error_Orig = newbalanceOrig - oldbalanceOrg + amount,
         error_Dest = newbalanceDest - oldbalanceDest - amount,
         zero_Orig = ifelse(oldbalanceOrg==0 & newbalanceOrig ==0 & amount!=0,1,0),
         zero_Dest = ifelse(oldbalanceDest==0 & newbalanceDest ==0 & amount!=0,1,0))
working_set$zero_Orig <- as.factor(working_set$zero_Orig)
working_set$zero_Dest <- as.factor(working_set$zero_Dest)
```

Regarding the *step* variable, 1 step represents 1 hour of time. As the sample covers one month, it could be useful to decompose the variable in *day* and *hour*.

```
working_set$hour <- working_set$step%%24
working_set$day <- findInterval(working_set$step, seq(0,743,24))
```

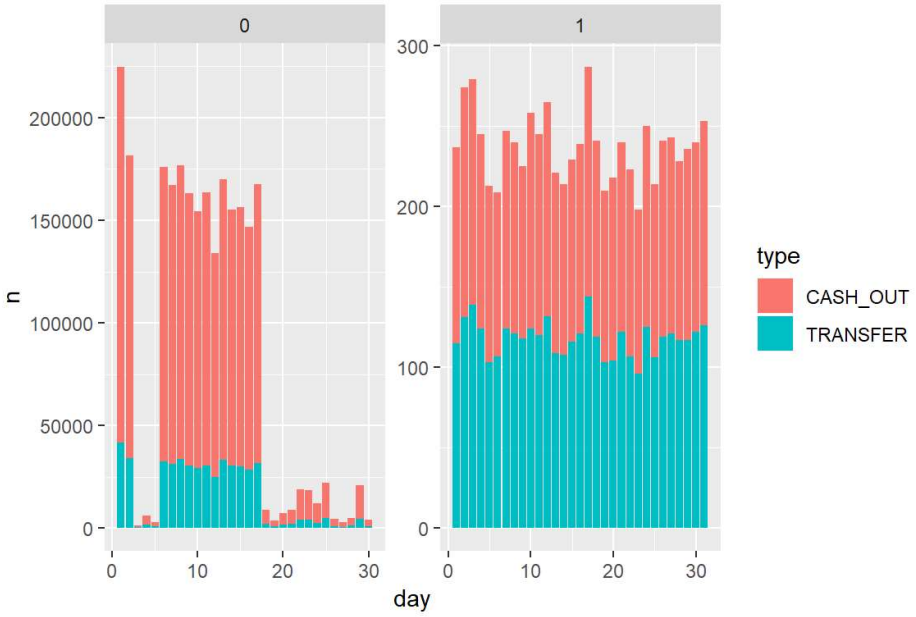
Let's take a look at the time at what the transactions are more frequent, and if there is any difference between frauds and no frauds:

```
working_set %>%
  group_by(hour, type, isFraud) %>%
  summarise(n=n()) %>%
  ungroup() %>%
  ggplot(aes(x=hour, y=n, fill=type)) +
  geom_col() +
  facet_wrap(. ~ isFraud, scales = "free")
```



The time of frauds follows a uniform distribution, while the valid transactions disappear at some moments of the day (presumably at night). As a consequence, a transaction that occurs between the hour 22 of one day and the hour 8 of the next day has higher probability of being fraud. Let's check if there is a similar patron for days:

```
working_set %>%
  group_by(day, type, isFraud) %>%
  summarise(n=n()) %>%
  ungroup() %>%
  ggplot(aes(x=day, y=n, fill=type)) +
  geom_col() +
  facet_wrap(. ~ isFraud, scales = "free")
```



Again, **fraudulent transactions follow a uniform distribution, while valid transactions not.** There appear to be some seasonality of transactions along the month, with peaks on the days 1-2 and 6 to 17. It could correspond to deadlines of some payments, but is something that needs further research. But it is clear that the day that the transaction occurs could help to predict if its fraudulent or not.

The other three variables of the dataset don't seem to provide useful information for fraud detection. There are only 12 cases when the variable *isFlaggedFraud* takes a value of one, and it always corresponds to a fraudulent transaction (obviously) and to a zero amount in both account of destination. I don't think that this variable can tell me something new about frauds.

```
working_set %>%
  filter(isFlaggedFraud==1) %>%
  select(type, nameOrig, nameDest, isFraud, isFlaggedFraud, error_Orig, error_Dest, zero_Orig, zero_Dest)
```

```
## # A tibble: 12 x 9
##    type  nameOrig nameDest isFraud isFlaggedFraud error_Orig error_Dest
##    <fct> <chr>    <chr>    <fct>   <fct>               <dbl>      <dbl>
##  1 TRAN~ C110058~ C114751~ 1       1                1343002.  -1343002.
##  2 TRAN~ C103554~ C110069~ 1       1                 536624.   -536624.
##  3 TRAN~ C908544~ C891140~ 1       1                4892193.  -4892193.
##  4 TRAN~ C689608~ C139280~ 1       1               10000000  -10000000
##  5 TRAN~ C193696~ C484597~ 1       1                3576297.  -3576297.
##  6 TRAN~ C168458~ C177041~ 1       1                 353874.   -353874.
##  7 TRAN~ C786455~ C661958~ 1       1                2542664.  -2542664.
##  8 TRAN~ C190047~ C180619~ 1       1               10000000  -10000000
##  9 TRAN~ C917414~ C108213~ 1       1                3441041.  -3441041.
## 10 TRAN~ C189221~ C130806~ 1       1                3171086.  -3171086.
## 11 TRAN~ C186956~ C186120~ 1       1                7316255.  -7316255.
## 12 TRAN~ C992223~ C136680~ 1       1                5674548.  -5674548.
## # ... with 2 more variables: zero_Orig <fct>, zero_Dest <fct>
```

The other two variables that I consider irrelevant are *nameOrig* and *nameDest*, but this requires further explanation. As it was mentioned earlier, in the *working_set* there are more or less the same amount of frauds in CASH_OUT transactions (3702) than in TRANSFER transactions (3660). This could be the result of the same fraudulent operation: the *thief* transfers money to her account and then cash it out to her pocket. In this case, the destination of the fraudulent transfer (nameDest) should be the origin of the fraudulent cash out (nameOrig). Let's see if there is evidence of this pattern:

```
fraud_transfer_dest <- working_set %>%
  filter(isFraud==1&type=="TRANSFER") %>%
  .$nameDest
working_set %>%
  filter(isFraud==1&type=="CASH_OUT") %>%
  filter(nameOrig %in% fraud_transfer_dest)
```

```
## # A tibble: 0 x 17
## # ... with 17 variables: step <dbl>, type <fct>, amount <dbl>,
## #   nameOrig <chr>, oldbalanceOrg <dbl>, newbalanceOrig <dbl>,
## #   nameDest <chr>, oldbalanceDest <dbl>, newbalanceDest <dbl>,
## #   isFraud <fct>, isFlaggedFraud <fct>, error_Orig <dbl>,
## #   error_Dest <dbl>, zero_Orig <fct>, zero_Dest <fct>, hour <dbl>,
## #   day <int>
```

There are 0 matches, so there is no evidence of the former pattern. Let's explore a little more these variables to see if we can find something interesting.

```
working_set %>%
  group_by(nameOrig) %>%
  summarise(n=n()) %>%
  .$n %>%
  table()
```

```
## .
##       1       2       3
## 2490573    1423       3
```

```
working_set %>%
  select(nameOrig) %>%
  n_distinct()
```

```
## [1] 2491999
```

There are 2,491,999 unique originants in the 2,493,428 observations, which implies that most of them made just 1 transaction. Moreover, no one made more than 3 transactions as an originant.

```
working_set %>%
  group_by(nameDest) %>%
  summarise(n=n()) %>%
  .$n %>%
  summary()
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   3.000   5.029   7.000  69.000
```

```
working_set %>%
  select(nameDest) %>%
  n_distinct()
```

```
## [1] 495775
```

On the other side, the are 495,775 unique accounts at the receiver side, the 50% of them received 3 transactions or more. Let's see if this is the same if we considered only the fraudulent transactions:

```
working_set %>%
  filter(isFraud==1) %>%
  group_by(nameOrig) %>%
  summarise(n=n()) %>%
  .$n %>%
  table()
```

```
## .
##    1
## 7362
```

```
working_set %>%
  filter(isFraud==1) %>%
  group_by(nameDest) %>%
  summarise(n=n()) %>%
  .$n %>%
  table()
```

```
## .
##    1    2
## 7304   29
```

All the originants of the fraudulent transactions has made only one transaction, while there are 29 accounts that in two times have received the fraudulent money. Let's explore a little bit more these 29 accounts:

```
id_29 <- working_set %>%
  filter(isFraud==1) %>%
  group_by(nameDest) %>%
  summarise(n=n()) %>%
  ungroup() %>%
  filter(n==2) %>%
  .$nameDest
```

Did they originate some transaction?

```
working_set %>%
  filter(nameOrig %in% id_29)
```

```
## # A tibble: 0 x 17
## # ... with 17 variables: step <dbl>, type <fct>, amount <dbl>,
## #   nameOrig <chr>, oldbalanceOrg <dbl>, newbalanceOrig <dbl>,
## #   nameDest <chr>, oldbalanceDest <dbl>, newbalanceDest <dbl>,
## #   isFraud <fct>, isFlaggedFraud <fct>, error_Orig <dbl>,
## #   error_Dest <dbl>, zero_Orig <fct>, zero_Dest <fct>, hour <dbl>,
## #   day <int>
```

No. Did they receive some valid transaction?

```
working_set %>%
  filter(nameDest %in% id_29,
         isFraud==0) %>%
  nrow()
```

```
## [1] 291
```

Yes, 291 valid transactions. As I result, **I can't say does an account that was involved in fraudulent transactions will be always involved in fraudulent transactions**.

Is there any difference between "merchants" and "clients"? I will use the *stringr* to detect the names that begins with "C" or "M".

```
which(str_detect(working_set$nameOrig, "^M"))
```

```
## integer(0)
```

```
which(str_detect(working_set$nameDest, "^M"))
```

```
## integer(0)
```

```
length(which(str_detect(working_set$nameOrig, "^C")))==nrow(working_set)
```

```
## [1] TRUE
```

```
length(which(str_detect(working_set$nameDest, "^C")))==nrow(working_set)
```

```
## [1] TRUE
```

Actually, we can see that there are no "merchants" in the dataset. All the values of *nameOrig* and *nameDest* begins with a "C".

After this exploration of the *nameOrig* and *nameDest*, I don't see any relevant information on them.

Conclusions of the data exploratory analysis

After this brief exploration of the dataset, I gained valuable insights regarding the fraudulent transactions:

- Fraud occurs only in TRANSFERS or in CASH_OUT transactions.
- The mean and median amount of fraudulent transaction is higher than in valid transactions. However, there are some very high-amount valid transfers.
- Fraud occurs at any time any day with more or less the same probability. Valid transactions are, by far, more frequent during some hours of the day and some days of the month.
- The apparent inconsistencies in the relation between *amount-oldbalance-newbalance* variables could be useful to predict frauds.
- There is no relevant information in the *isFlaggedFraud*, *nameOrig* or *nameDest* variables.

I will use this information in the Section 4 to construct a model that could predict fraudulent transactions.

```
rm(fraud_transfer_dest, id_29)
```

3. Datasets and metrics for evaluation.

In this Section I will prepare the datasets for training and testing the models, and I will explain how I'm going to evaluate the predictive power of the models.

3.1. Training and test sets

I will remove from the working_set the variables that I'm not going to use in the prediction of frauds.

```
working_set <- working_set %>%
  select(-c(step, nameOrig, nameDest, isFlaggedFraud))
```

I will split the working_set in a *train_set* (90%) and a *test_set* (10%). This will be done with the *createDataPartition* function of the *caret package*.

```
set.seed(1)
test_index <- createDataPartition(y=working_set$isFraud, times=1,p=0.9,list=FALSE)
train_set <- working_set[test_index,]
test_set <- working_set[-test_index,]
rm(test_index)
```

Now we have the datasets ready to start training the models. However, first I'm going to explain how I'm going to evaluate the model performances.

3.2. How to evaluate the models

The transactions could be either fraudulent ($Y = 1$) or not ($Y = 0$). In the same way, a model can predict that any transaction is either fraudulent ($\hat{Y} = 1$) or not ($\hat{Y} = 0$). This gives the possibility of 4 results:

True Positives (TP): a fraudulent transaction predicted as fraudulent.

True Negatives (TN): a valid transaction predicted as valid.

False Positives (FP): a valid transaction predicted as fraudulent.

False Negatives (FN): a fraudulent transaction predicted as valid.

The simplest way to evaluate the algorithm when the outcome is categorical is by simply reporting the proportion of cases that were correctly predicted in the test set. This metric is usually referred to as **overall accuracy**:

$$OverallAccuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

As it was shown in Section 2, the fraudulent transactions are less than 1% of the dataset. This means that a model that predicts that there is never fraud, will achieve an Overall Accuracy of more than 99%, just because of the **prevalence** of valid transactions over fraudulent ones. However, is hard to say that a model for fraud detection is good when it fails to identify 100% of frauds. As a result, other indicators are needed to evaluate the performance for the models in this context.

We can define **Sensitivity** as the ability of an algorithm to predict a positive outcome when the actual outcome is positive. In this case, this means the proportion of fraudulent transactions that are actually predicted as fraudulent:

$$Sensitivity = P(\hat{Y} = 1|Y = 1) = \frac{TP}{TP + FN}$$

This quantity is also referred to as the **true positive rate (TPR) or recall**.

In the same way, we can define **Specificity** as the ability of an algorithm to predict a negative outcome when the actual outcome is negative. In this case, this means to predict a valid transaction when it is actually valid:

$$Specificity = P(\hat{Y} = 0|Y = 0) = \frac{TN}{TN + FP}$$

In the case of a model that predicts that there is never a fraud, *Specificity* would be of 100% but *Sensitivity* would be 0%. That's way we need to check both indicators, and not only *Overall Accuracy*. In fact, as frauds are very costly, in this case **Sensitivity is more relevant than Specificity**. We should try to avoid any fraud, even at the cost of stopping some valid transactions.

One widely used indicator that consider both *Specificity* and *Sensitivity*, allowing to assign more weight to one of them, is the $F_1 - score$:

$$F_1 - score = \frac{1}{\frac{\beta^2}{1+\beta^2}\frac{1}{recall} + \frac{1}{1+\beta^2}\frac{1}{precision}}$$

Where *Precision* is another way to measure *Specificity*:

$$Precision = P(Y = 1|\hat{Y} = 1) = \frac{TP}{TP + FP}$$

and β is a parameter that allows to weight between *Specificity* and *Sensitivity*. A value of $\beta = 1$ mean that both measure weights the same.

Even though I will be checking all of these indicators, the *rule of thumb* for picking one model over another will be the $F_1 - score$ with a $\beta = 2$, which gives a weight of 80% for *Recall or Sensitivity* and a 20% for *Precision or Specificity*.

4. Training a model for fraud prediction

In this section I will train a model in the *train_set* and evaluate it's performance on the *test_set*. There are different models that I could try. However, given the size and characteristics of the dataset, I think that **Decision Trees and Random Forest** should perform well. Decision Trees models faces more risks of overfitting, so a Random Forest could perform better in prediction. However, in a Decision Tree you can actually *see the tree*, allowing a better analysis of the decisions made by the model in the classification process. That's why the analysis will be centered in Decision Trees, although I will also estimate a Random Forest in order to compare the results.

4.1 Decision Trees

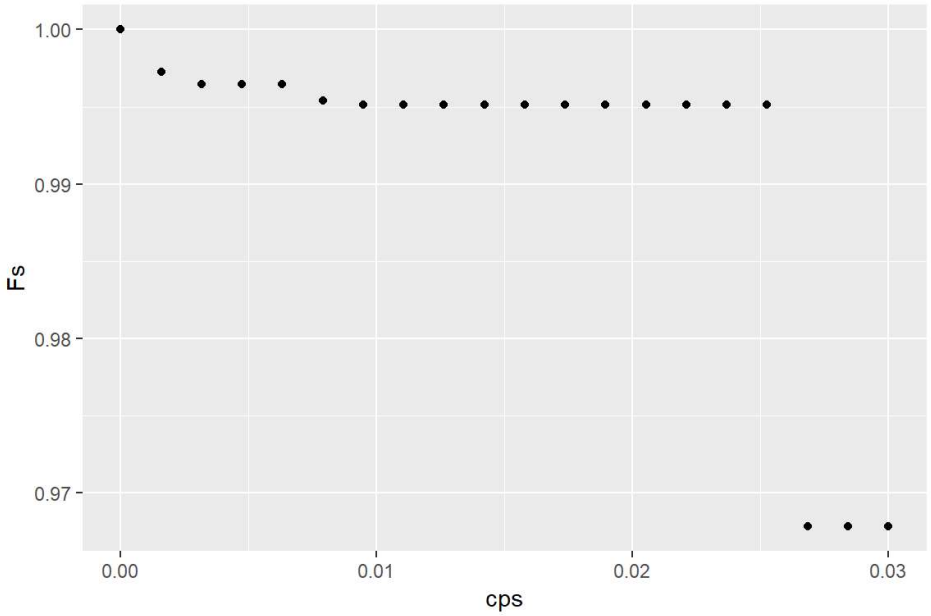
I will train 20 decision trees with *complexities parameters* ranging from 0 to 0.03. I will prune each of this model in order to reduce overfitting. As it was explained in Section 3, I will choose the model that gives a better F-score with a parameter $\beta = 2$.

Note: The execution of the following code will take time the first time. The results are saved in the file "Fs.Rda" (available at github) to make the next execution faster.

```
cps <- seq(0, 0.03, len = 20)

if (file.exists("Fs.Rda")) {
  load("Fs.Rda")
} else {
  Fs <- sapply(cps, function(cp){
  fit <- rpart(isFraud ~ ., data = train_set, control = rpart.control(cp = cp))
  pfit <- prune(fit, cp=cp)
  y_hat <- predict(pfit, test_set, type="class")
  return(F_meas(data=y_hat,
       reference = test_set$isFraud, beta = 2, relevant = "1"))
})
  save(Fs, file="Fs.Rda")
}

qplot(cps,Fs)
```




```
cps[which.max(Fs)]
```

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

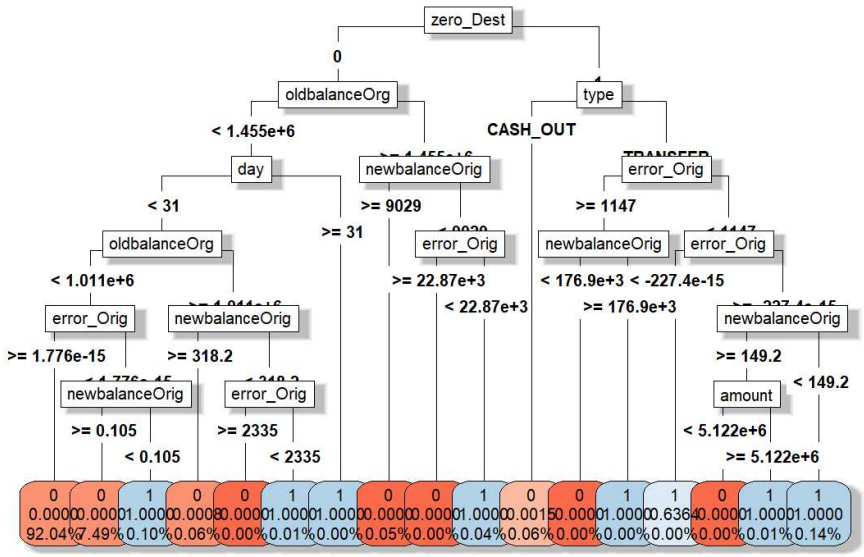
We can see that a complexity parameter of 0 gives the maximum value for the F-Score. Let's train the model and analyze its results.

Note: The execution of the following code will take time the first time. The results are saved in the file "DT_1.Rda" (available at Github) to make the next execution faster.

```
if (file.exists("DT_1.Rda")) {
  load("DT_1.Rda")
} else {
  fit <- rpart(isFraud ~ ., data = train_set, control = rpart.control(cp = 0))
  save(fit, file="DT_1.Rda")
}

pfit <- prune(fit, cp=0)

rpart.plot(pfit, box.palette = "RdBu", shadow.col = "gray", nn=FALSE,
           type = 5, tweak = 1.5, digits = 4)
```



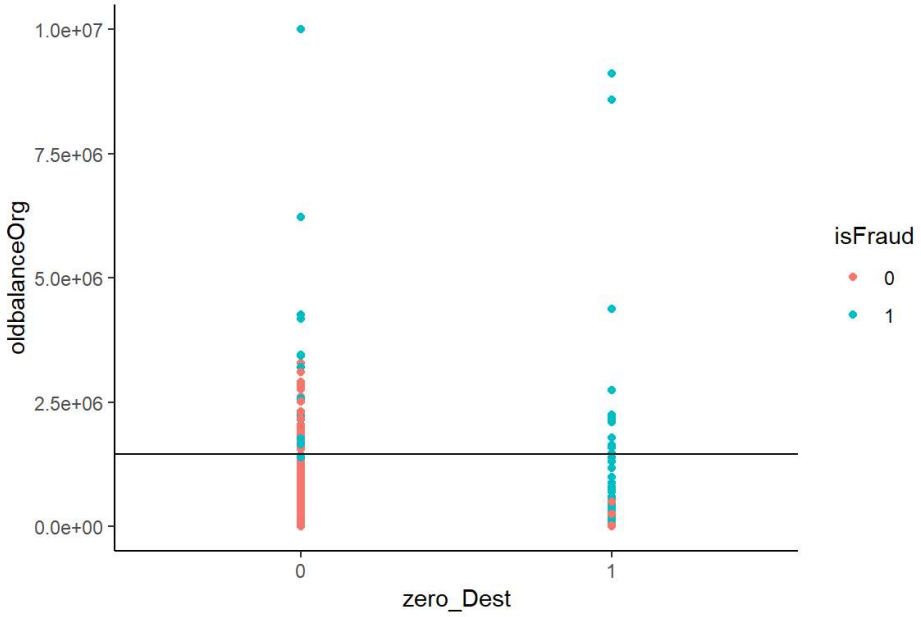
```
y_hat <- predict(pfit, test_set, type="class")
cm <- confusionMatrix(y_hat, test_set$isFraud, positive = "1")
cm$table
```

```
##           Reference
## Prediction      0      1
##          0 248606      0
##          1      0    736
```

We have perfect prediction!! The 736 frauds were detected, with no false alarms. Could this be a case of overfitting? Will see it when comparing with the Random Forest model and using the model on the *validation_set*. But first, let's analyze the results.

The first split of the tree is according the *zero_Dest* variable, that is when the destination account is 0 before and after a non zero transaction. Then, on the left leave, we have a split according to the balance in the account of origin before the transaction. Let's see graphically how these two variables relates with the fraudulent transactions (in a small sample).

```
set.seed(1)
index <- sample(1:nrow(working_set), 50000, replace=FALSE)
working_set[index,] %>%
  select(isFraud, oldbalanceOrg, zero_Dest) %>%
  ggplot(aes(x=zero_Dest, y=oldbalanceOrg, color=isFraud)) +
  geom_point() +
  geom_hline(yintercept = 1455000) +
  theme_classic()
```



As we can see, most of the transaction with a value of 1 for *zero_Dest* are frauds, and when this variable is 0 frauds only occurs at high values of *oldbalanceOrg*. We can conclude that these two variables together are good predictors of frauds. Now let's take a look to the right side of the tree. After splitting by *zero_Dest*, the tree splits by *type*. As both variables are categorical, let's make a table.

```
working_set%>%
  select(isFraud, type, zero_Dest) %>%
  group_by(type, zero_Dest) %>%
  summarise(p_frauds=mean(isFraud=="1"))
```

```
## # A tibble: 4 x 3
## # Groups:   type [?]
##   type     zero_Dest  p_frauds
##   <fct>    <fct>         <dbl>
## 1 CASH_OUT 0         0.00184
## 2 CASH_OUT 1         0.00138
## 3 TRANSFER 0         0.0000609
## 4 TRANSFER 1         0.975
```

We can see that in the case of TRANSFERS with a value of 1 for *zero_Dest*, 97.5% transactions are frauds. I'm not going to analyze every branch of the tree, but I'll conclude the analysis mentioning the two biggest groups of fraudulent transactions.

Main group of fraudulent transactions

According to the tree, the main group of fraudulent transactions can be found filtering, in this order, as follows:

- zero_Dest == 1
- type == TRANSFER
- error_Orig < 1147
- error_Orig >= -227.4e-15
- newbalanceOrig < 149.2

Let's check it:

```
working_set %>%
  filter(zero_Dest == "1") %>%
  filter(type == "TRANSFER") %>%
  filter(error_Orig < 1147) %>%
  filter(error_Orig >= -227.4e-15) %>%
  filter(newbalanceOrig < 149.2) %>%
  group_by(isFraud) %>%
  summarise(n=n())
```

```
## # A tibble: 1 x 2
##   isFraud     n
##   <fct>   <int>
## 1 1        3480
```

We can see that in this group there are 3480 fraudulent transaction and 0 valid ones (only values of 1 for *isFraud* were founded).

Second group of fraudulent transactions

According to the tree, the second group of fraudulent transactions can be found filtering, in this order, as follows:

- zero_Dest == 0
- oldbalanceOrg < 1.455e+6
- day < 31
- oldbalanceOrg < 1.011e+6
- error_Orig < 1.776e-15
- newbalanceOrig < 0.105

Let's check it:

```
working_set %>%
  filter(zero_Dest == "0") %>%
  filter(oldbalanceOrg < 1.455e+6) %>%
  filter(day < 31) %>%
  filter(oldbalanceOrg < 1.011e+6) %>%
  filter(error_Orig < 1.776e-15) %>%
  filter(newbalanceOrig < 0.105) %>%
  group_by(isFraud) %>%
  summarise(n=n())
```

```
## # A tibble: 1 x 2
##   isFraud     n
##   <fct>   <int>
## 1 1        2398
```

We can see that in this group there are 2398 fraudulent transaction and 0 valid ones (only values of 1 for *isFraud* were founded).

As a conclusion, I can say the Decision Tree is very effective identifying frauds, but the rules are kind of complicated. It's probably that the model is overfited.

Let's see how different the results with a Random Forest are.

Note: The execution of the following code will take time the first time. The results are saved in the file "RF_1.Rda" (available at Github) to make the next execution faster.

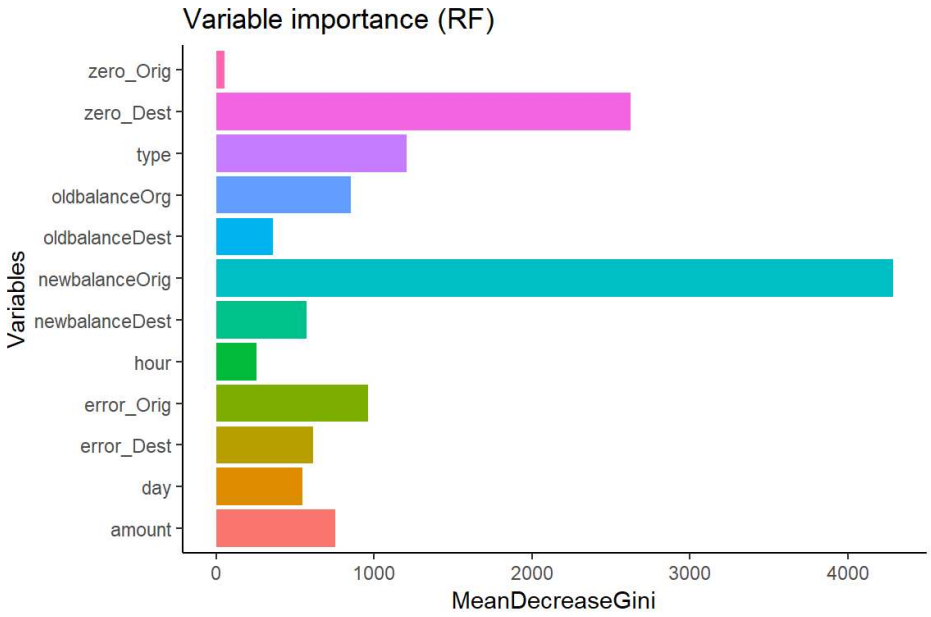
```
if (file.exists("RF_1.Rda")) {
  load("RF_1.Rda")
} else {
  set.seed(1)
  fit_rf <- randomForest(isFraud ~ ., data = train_set, ntree = 20, mtry = 3)
  save(fit_rf, file="RF_1.Rda")
}

y_hat_rf <- predict(fit_rf, newdata = test_set[,-7])
cm_rf <- confusionMatrix(y_hat_rf, test_set$isFraud, positive = "1")
cm_rf$table
```

```
##           Reference
## Prediction      0      1
##          0 248606      0
##          1      0    736
```

The Random Forest performs almost the same than the Decision Tree. It also identifies the 736 fraudulent transactions and only gives 1 false alarm. Let's see which the more relevant variables are according to this model:

```
data.frame(Variables = rownames(fit_rf$importance),
                                fit_rf$importance, row.names = NULL) %>%
  ggplot(aes(y = MeanDecreaseGini, x = Variables, fill = Variables)) +
  geom_col() +
  coord_flip() +
  labs(title= "Variable importance (RF)") +
  theme_classic() +
  theme(legend.position = "none")
```



According to the Random Forest Model, the most important variable is *newbalanceOrig*, followed by *zer_Dest*, *type* and *error_Orig*. This is consistent with the Decision Tree model, as all of these variables where relevant to identify frauds. I must confess that I feel kind of disappointed with the low relevance showed by the *day* and *hour* variables, as they pattern looks pretty clear to identify frauds.

In the next Section, I will train the Decision Tree with a complexity parameter of 0 in all the *working_set*, and test it in the *validation_set*.

5. Results

After identifying the relevant variables and testing the Decision Tree model in the *working_set*, it's time to try it in the *validation_set*. The first thing I need to do is to repeat the conversion in the variables on this dataset.

```
validation_set$type <- as.factor(validation_set$type)
validation_set$isFraud <- as.factor(validation_set$isFraud)
validation_set$isFlaggedFraud <- as.factor(validation_set$isFlaggedFraud)
```

Next, I'm going to test the prediction that all the transactions that were not TRANSFER or CASH_OUT are valid:

```
validation_set %>%
  filter(!(type %in% c("TRANSFER", "CASH_OUT"))) %>%
  filter(isFraud=="1")
```

```
## # A tibble: 0 x 11
## # ... with 11 variables: step <dbl>, type <fct>, amount <dbl>,
## #   nameOrig <chr>, oldbalanceOrg <dbl>, newbalanceOrig <dbl>,
## #   nameDest <chr>, oldbalanceDest <dbl>, newbalanceDest <dbl>,
## #   isFraud <fct>, isFlaggedFraud <fct>
```

Great, there are no frauds in other transaction types. Let's filter these types of transaction and continue with the required variable transformations.

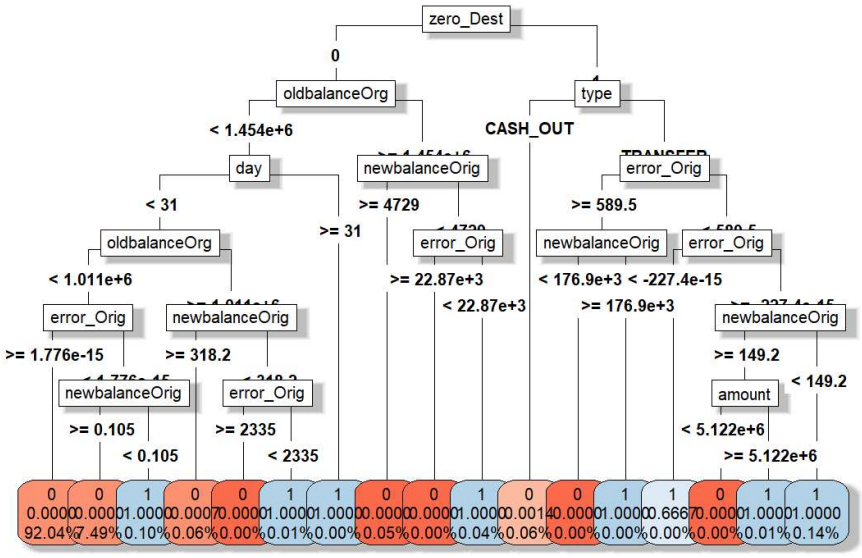
```
validation_set <- validation_set %>%
  filter(type %in% c("TRANSFER", "CASH_OUT")) %>%
  mutate(type = as.factor(as.character(type)))
validation_set <- validation_set %>%
  mutate(error_Orig = newbalanceOrig - oldbalanceOrg + amount,
         error_Dest = newbalanceDest - oldbalanceDest - amount,
         zero_Orig = ifelse(oldbalanceOrg==0 & newbalanceOrig ==0 & amount!=0,1,0),
         zero_Dest = ifelse(oldbalanceDest==0 & newbalanceDest ==0 & amount!=0,1,0))
validation_set$zero_Orig <- as.factor(validation_set$zero_Orig)
validation_set$zero_Dest <- as.factor(validation_set$zero_Dest)
validation_set$hour <- validation_set$step%%24
validation_set$day <- findInterval(validation_set$step, seq(0,743,24))
validation_set <- validation_set %>%
  select(-c(step, nameOrig, nameDest, isFlaggedFraud))
```

Now we have the datasets ready to do the final training. Let's train it and test it.

Note: The execution of the following code will take time the first time. The results are saved in the file "TD_f.Rda" (available at Github) to make the next execution faster.

```
if (file.exists("TD_f.Rda")) {
  load("TD_f.Rda")
} else {
  fit <- rpart(isFraud ~ ., data = working_set, control = rpart.control(cp = 0))
  save(fit, file="TD_f.Rda")
}

pfit <- prune(fit, cp=0)
rpart.plot(pfit, box.palette = "RdBu", shadow.col = "gray", nn=FALSE,
           type = 5, tweak = 1.5, digits = 4)
```




```
y_hat <- predict(pfit, validation_set, type="class")
confusionMatrix(y_hat, validation_set$isFraud, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##          0 276130      4
##          1      0    847
##
##                Accuracy : 1
##                  95% CI : (1, 1)
##     No Information Rate : 0.9969
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9976
##  Mcnemar's Test P-Value : 0.1336
##
##             Sensitivity : 0.995300
##             Specificity : 1.000000
##          Pos Pred Value : 1.000000
##          Neg Pred Value : 0.999986
##              Prevalence : 0.003072
##          Detection Rate : 0.003058
##    Detection Prevalence : 0.003058
##       Balanced Accuracy : 0.997650
##
##        'Positive' Class : 1
##
```

```
F_meas(data=y_hat,reference = validation_set$isFraud, beta = 2, relevant = "1")
```

```
## [1] 0.9962362
```

We can see that the model identified 847 of the 851 frauds, with no false alarms. This results in a F-Score value of 99.6%, given the sensitivity of 99.5% and the specificity of 100%. We can see that the model performs pretty well, even though doesn't give a perfect prediction as in the *working_set*.

Let's take a look at the four frauds that were not identified.

```
f_non_detected <- validation_set %>%
  mutate(pred=y_hat) %>%
  filter(y_hat!=isFraud)
f_non_detected
```

```
## # A tibble: 4 x 14
##   type  amount oldbalanceOrg newbalanceOrig oldbalanceDest newbalanceDest
##   <fct>  <dbl>         <dbl>          <dbl>          <dbl>          <dbl>
## 1 CASH~ 5.09e5             0              0       1082008.       1590790.
## 2 CASH~ 1.22e5             0              0             0         639940.
## 3 CASH~ 5.77e5             0              0             0         577419.
## 4 CASH~ 4.06e4             0              0             0          40611.
## # ... with 8 more variables: isFraud <fct>, error_Orig <dbl>,
## #   error_Dest <dbl>, zero_Orig <fct>, zero_Dest <fct>, hour <dbl>,
## #   day <int>, pred <fct>
```

We can see that the four transactions are located at the left side of the tree:

```
validation_set %>%
  filter(zero_Dest==0) %>%
  filter(oldbalanceOrg<1.454e+6) %>%
  filter(day<31) %>%
  filter(oldbalanceOrg<1.011e+6) %>%
  filter(error_Orig>=1.776e-15) %>%
  group_by(isFraud) %>%
  summarise(n=n())
```

```
## # A tibble: 2 x 2
##   isFraud      n
##   <fct>    <int>
## 1 0       254905
## 2 1            4
```

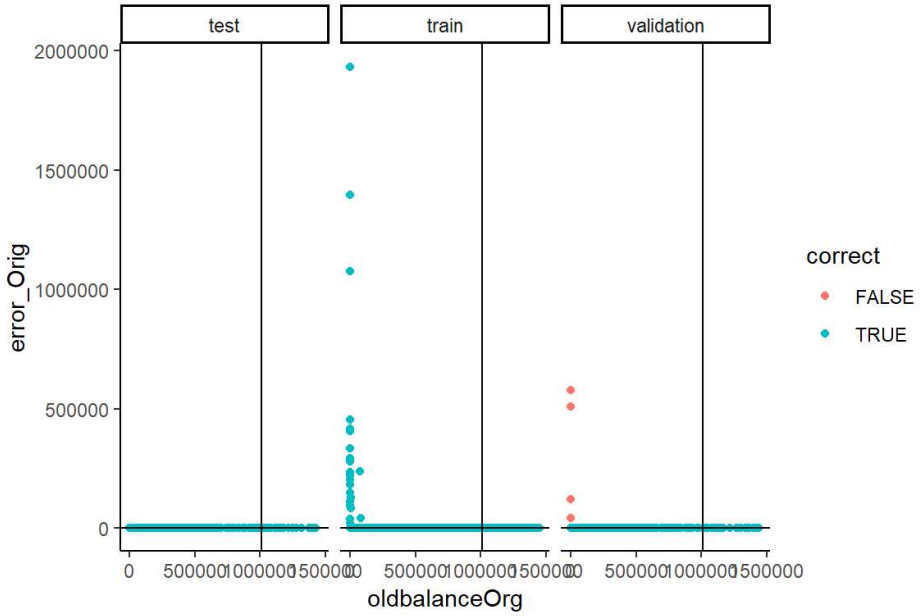
In that part of the tree there are 254,905 valid transactions, and only these 4 frauds. Could a better model have identified these frauds?. Let's make some exploration.

Let's follow the left leave of the tree up to the node *day<31*. Let's compare how are distributed the fraudulent transactions in the three datasets: *train_set, test_set and validation_set*. First, we need to construct a *dataframe* with all the observations:

```
non_detected <- validation_set %>%
  mutate(pred=y_hat) %>%
  mutate(correct = pred==isFraud) %>%
  mutate(set = "validation") %>%
  filter(isFraud == "1")
train_set2 <- train_set %>%
  mutate(pred=isFraud) %>%
  mutate(correct = pred==isFraud) %>%
  mutate(set = "train") %>%
  filter(isFraud == "1")
test_set2 <- test_set %>%
  mutate(pred=isFraud) %>%
  mutate(correct = pred==isFraud) %>%
  mutate(set = "test") %>%
  filter(isFraud == "1")

non_detected <- rbind(non_detected, train_set2)
non_detected <- rbind(non_detected, test_set2)

non_detected %>%
  filter(zero_Dest=="0") %>%
  filter(oldbalanceOrg < 1.454e+6) %>%
  filter(day<31) %>%
  ggplot(aes(x=oldbalanceOrg, y=error_Orig)) +
  geom_point(aes(color=correct)) +
  geom_hline(yintercept = 1.776e-15) +
  geom_vline(xintercept = 1.011e+6) +
  facet_wrap(. ~ set) +
  theme_classic()
```



The next 2 nodes of the tree (*oldbalanceOrg<1.011e+6* and *error_Orig>=1.776e-15*) clasifies the North West quadrant of each of the three graphs as *non fraud*. As in the *test_set* there are no frauds in that region, we have obtained a perfect prediction. However, we wouldn't have had perfect prediction on the *train_test* for the same reason that we didn't have perfect prediction on the *validation* set: there are frauds on that quadrant.

What happens if we change the decision node *error_Orig>=1.776e-15* to something like *error_Orig>=600000*? This would have move the 4 undetected frauds of the *validation_set* to the South Weast quadrant. Could we have a better prediction? Let's see:

```
validation_set %>%
  filter(zero_Dest=="0") %>%
  filter(oldbalanceOrg < 1.454e+6) %>%
  filter(day<31) %>%
  # I will filter error_Orig<600000 to keep the undetected frauds
  filter(error_Orig<600000) %>%
  group_by(isFraud) %>%
  summarise(n=n())
```

```
## # A tibble: 2 x 2
##   isFraud      n
##   <fct>    <int>
## 1 0       253319
## 2 1          302
```

We now have 253,319 non frauds and 302 frauds there. Surely, that region will be clasified as *non fraud*, and the prediction will be even worse. Let's see if with the following rule of the tree there is some improvement.

```
validation_set %>%
  filter(zero_Dest=="0") %>%
  filter(oldbalanceOrg < 1.454e+6) %>%
  filter(day<31) %>%
  # I will filter error_Orig<600000 to keep the undetected frauds
  filter(error_Orig<600000) %>%
  filter(newbalanceOrig<0.105) %>%
  group_by(isFraud) %>%
  summarise(n=n())
```

```
## # A tibble: 2 x 2
##   isFraud      n
##   <fct>    <int>
## 1 0       226164
## 2 1          302
```

There is still a clear majority of non frauds. Again, the model will perform worse. **Instead of clasifying correctly the 4 frauds, we would clasify incorrectly 302 frauds. As we can see, it's not so easy to imporove the model**. After all, it performs really well.

Just to know, could the Random Forest model have performed better?

Note: The execution of the following code will take time the first time. The results are saved in the file "RF_f.Rda" (available at Github) to make the nexts execution faster.

```
if(file.exists("RF_f.Rda")) {
  load("RF_f.Rda")
} else {
  set.seed(1)
  fit_rf <- randomForest(isFraud ~ ., data = working_set, ntree = 20, mtry = 3)
  save(fit_rf, file="RF_f.Rda")
}

y_hat_rf <- predict(fit_rf, newdata = validation_set[,-7])
cm_rf <- confusionMatrix(y_hat_rf, validation_set$isFraud, positive = "1")
cm_rf$table
```

```
##           Reference
## Prediction      0      1
##          0 276130      5
##          1      0    846
```

Actually, the Random Forest performs worse. It fails to identify 5 frauds instead of 4. This confirms that there wasn't a problem of overfitting in the Decision Tree model.

6. Conclusions

This exercise confirms the relevance of the Exploratory Data Analysis. In fact, after exploring the dataset it was possible to construct variables that were highly relevant to identify frauds, like *zero_Dest* and *error_Orig*. Even though the variables *day* and *hour* didn't result so relevant as I thought, I guess that the model performs better with those variables than with the original *step*. In any case, if I would run the models in the original datasets, the results would have been very disappointing.

After identifying and constructing the relevant variables, both Decision Trees and Random Forest performed pretty well. There was some initial suspicion that the Decision Tree was overfitting, but the analysis of the results shows that it wasn't.

Even though the model couldn't achieve perfect prediction on the *validation_set* (as it had achieved on the *test_set), it's performance was quite impressive: the model identified 847 of the 851 frauds, with no false alarms. I'm sure there is room for improvement, but I'm satisfied with the results obtained so far.