**Synthetic Financial Datasets For Fraud Detection**

****

**By :**

* Nedson SIMO MABOU
* Maryame KOURAM
* Hafsa AKEBLI

**Mentor :**

* Maryam GUESSOUSS

**Summary**

[Introduction : 3](#_Toc62486058)

[Headers : 3](#_Toc62486059)

[Import 4](#_Toc62486060)

[Exploratory data analysis : 5](#_Toc62486061)

[a. Correlation between « type » and « isFraud » 5](#_Toc62486062)

[b. « isFlaggedFraud» is it relevant or not? 7](#_Toc62486063)

[c. « Name Orig » and « Name Dest » are they relevant or not 8](#_Toc62486064)

[d. “step” feature analysis 9](#_Toc62486065)

[e. Relationship between amount and isFraud 12](#_Toc62486066)

[f. Other remarks : Transactions with zero balances 13](#_Toc62486067)

[Skewness and kurtosis of numerical features : 14](#_Toc62486068)

[Feature engineering 15](#_Toc62486069)

[Binary encoding 15](#_Toc62486070)

[Machine Learning Models 15](#_Toc62486071)

[Data skewness – data transformation 19](#_Toc62486072)

[Conclusion 22](#_Toc62486073)

## Introduction :

We worked on a synthetic dataset generated using the simulator called PaySim. PaySim uses aggregated data from the private dataset to generate a synthetic dataset that resembles the normal operation of transactions and injects malicious behaviour to later evaluate the performance of fraud detection methods.

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 dataset contains 11 columns and in 6 million rows.

## Headers :

**Step** : maps a unit of time in the real world. In this case 1 step is 1 hour of time. Total steps 744 (30 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

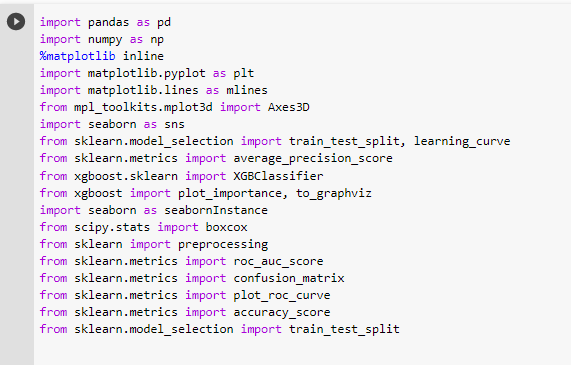
**oldbalanceDest :** initial balance recipient before the transaction. Note that there is not information for customers that start with M (Merchants).

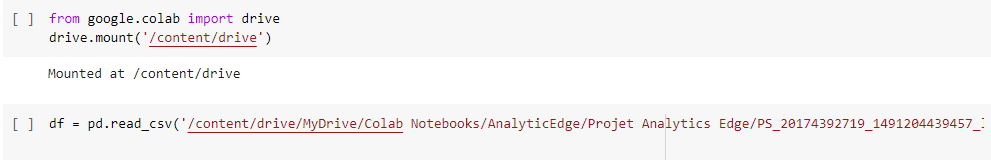
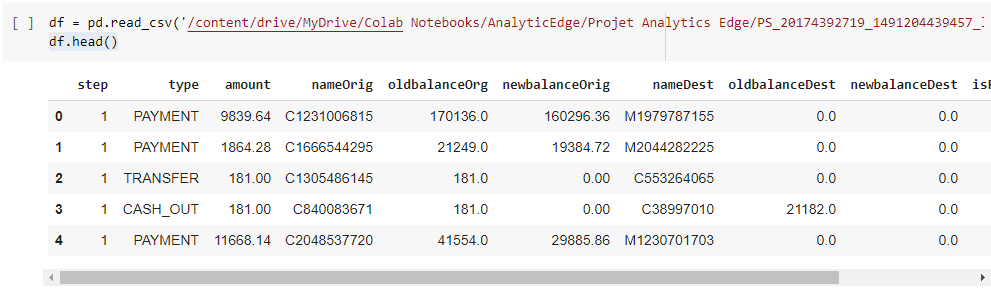
**newbalanceDest :** new balance recipient after the transaction. Note that there is not information for customers that start with M (Merchants).

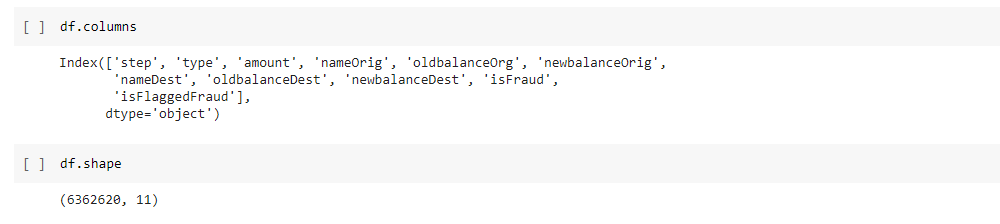
**isFraud :** This is the transactions made by the fraudulent agents inside the simulation. In this specific dataset the fraudulent behavior of the agents aims to profit by taking control or customers accounts and try to empty the funds by transferring to another account and then cashing out of the system.

**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.

## Import

Import libraries :

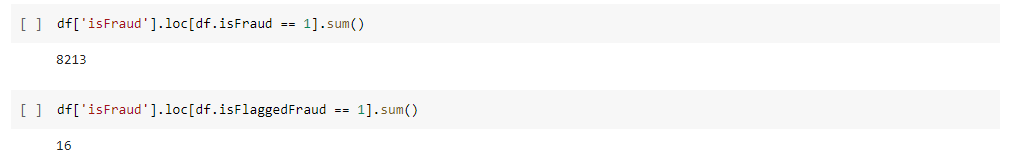
Import data

Let’s discover more our dataframe : it’s columns, it’s shape and the types of all our features



Test if there are any missing values in the DataFrame. It turns out that there are no missing values but.

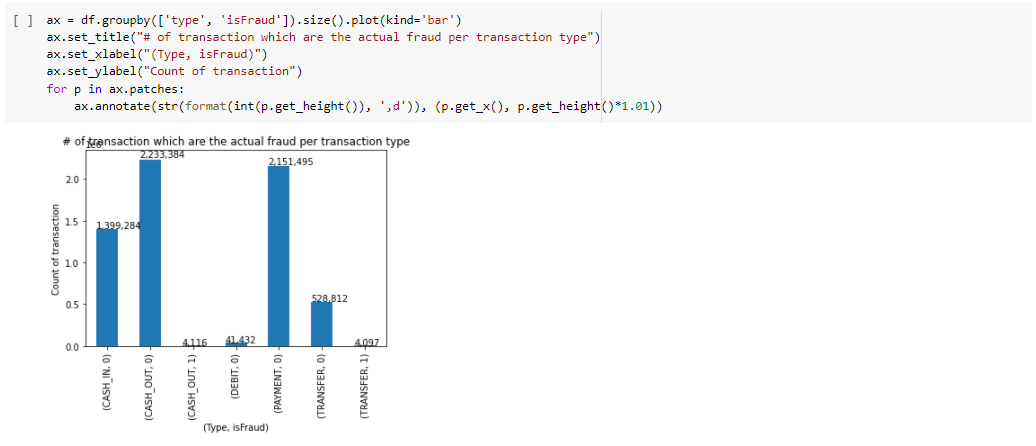


Let’s count the number of lines where our dependant feature **isFraud=1** and where **isFlaggedFraud=1**

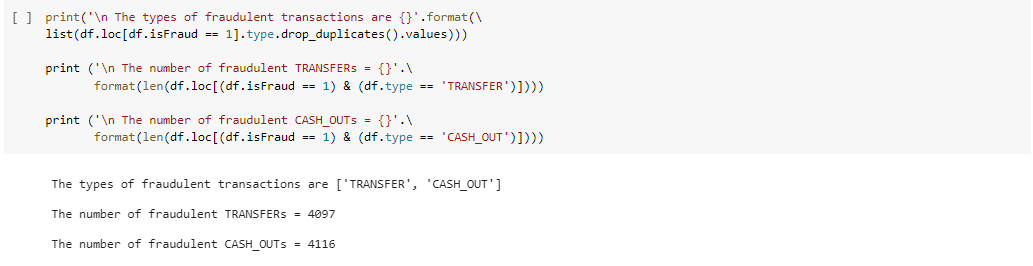
## Exploratory data analysis :

### Correlation between « type » and « isFraud »

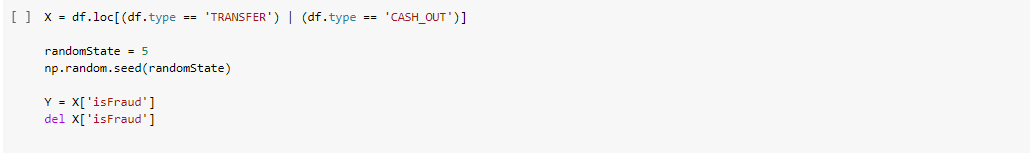
We group the data according to the feature : « type » and « isFraud ».



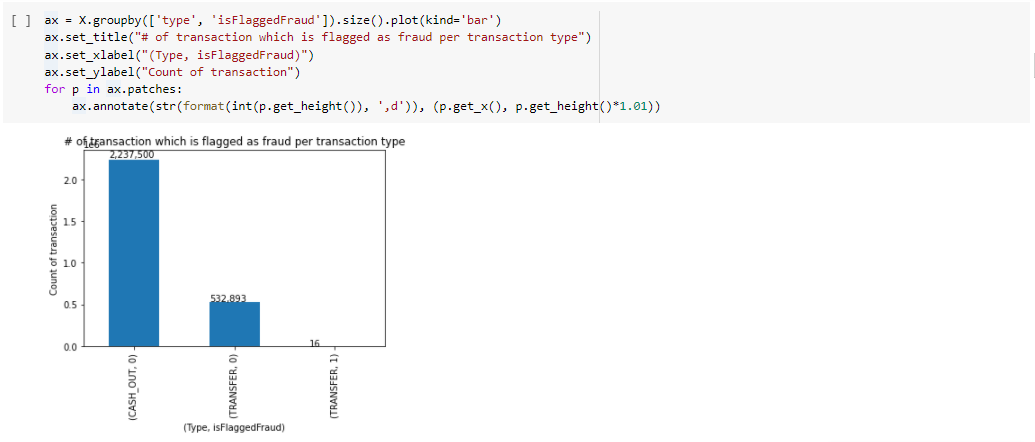
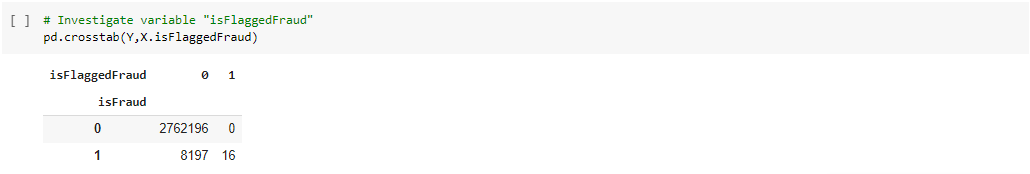
We find that the lines where isFraud=1 exist only when type=Transfer or type=Cashout.

Let’s check that in another way :

So we decide to drop all the lines of the dataframe where « type » is different than « transfer » or « cashout » , because if the type is different than those two types , we already know that the transaction is not fraudulent.

We name X the final dataframe that we’re going to use to train our models at the end.

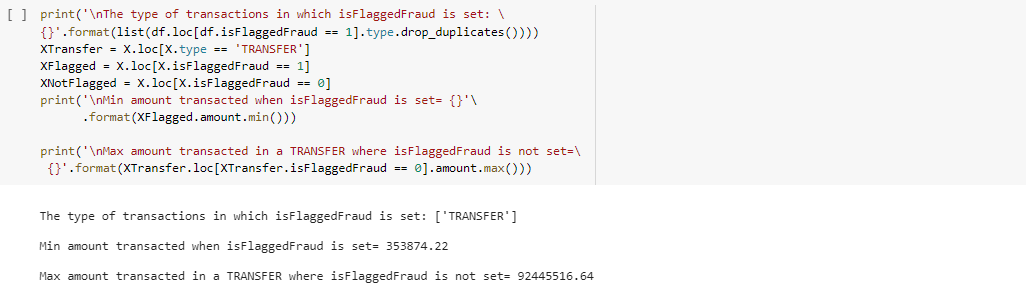
### « isFlaggedFraud» is it relevant or not?

We group the data according to the feature : «isFlaggedFraud » and « type».

Number of lines that have isFlaggedFraud=1 are only 16 and they all belong to the type : Transfer .

In the information given about our dataset we have:

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. Let’s check this information :

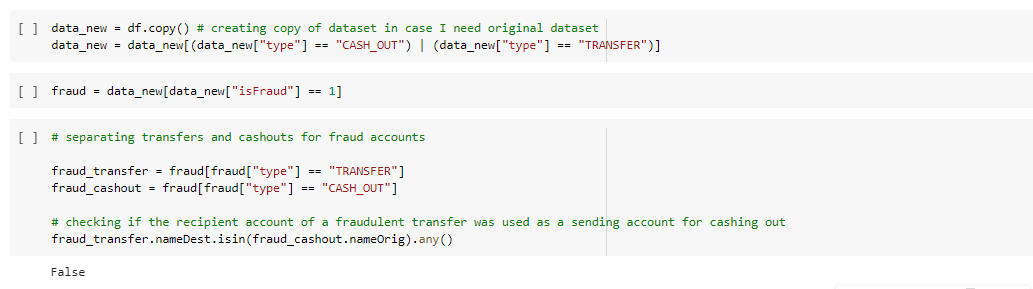


We can see that isFlaggedFraud can remain not set despite this condition being met.

We decide then to drop this feature, because it’s not relevant.

### « Name Orig » and « Name Dest » are they relevant or not

We create a copy of our dataframe df , we drop the lines where type is different than CASHOUT and TRANSFER, we create a dataframe of all the fraudulent transactions called fraud where isFraud=1. We separe this one into two dataframes depending on the feature « type » : the first dataframe is called fraud\_transfer and the second fraud\_cashout. Normally since we have only two kinds of types in the fraudulent transactions (transfer and cashout), a person who hacked an account and transfered it’s money to his account , he must withraw the money stolen through a cashout. Let’s check this information :



Thus in this dataset, for fraudulent transactions, the account that received funds during a transfer was not used at all for cashing out.

If that is the case, there seems to be no use for nameOrig or nameDest since there seems to be no restrictions on which accounts cashout from fraudulent transactions.

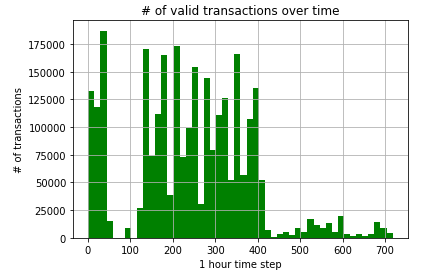
Thus, we're omitting the nameOrig and nameDest columns from analysis.

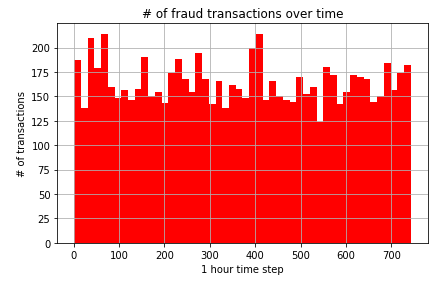
Finally we drop all the features that we have judged not relevant

### “step” feature analysis

We recall that the step feature maps a unit of time in the real world. In this case 1 step is 1 hour of time. The total number of steps in the dataset is 744. In other words: 30 days simulation. The aim of this section is to see whether this feature gives insight on whether a transaction is fraudulent or not.

We start by plotting the number of transactions associated to each hour time step for fraudulent and non-fraudulent transactions.

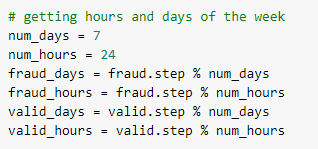




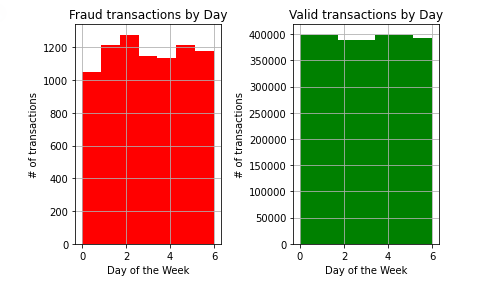
The green plot shows the number of transactions for each hour time step of non-fraudulant transactions. A large proportion of valid transactions occur between around the 0th and 60th timestep as well as the 110th and 410th time-steps.

However, The frequency at which fraudulent transactions occur does not seem to change much over time. Hence, the hour time step doesn’t appear to be an interesting feature.

Let's see what the patterns look like over any particular, day of the week or hour of the day.

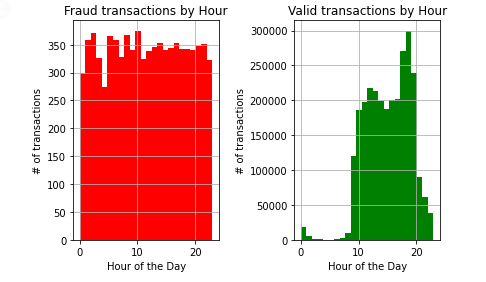


Our aim is to plot number of transactions of each day (a day 0 in the plot for example represents day 0, day 7, day 14, etc. during the entire experiment). Note that day 0 does not necessarily mean the first day of the week. If day 0 is Tuesday, then day 1 is Wednesday, etc.



From the plots above, there is little evidence to suggest that fraudulent transactions occur at particular days of the week. Much like valid transactions, fraudulent transactions seem to occur uniformly for each day of the week. Thus, a feature showing what day of the week a transaction occurred is not interesting at all.

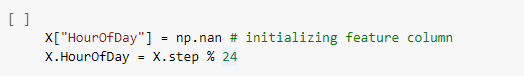
Now, much like the logic we used in the plot above, we do the same thing for hours of the day, we visualise the plotting results below.



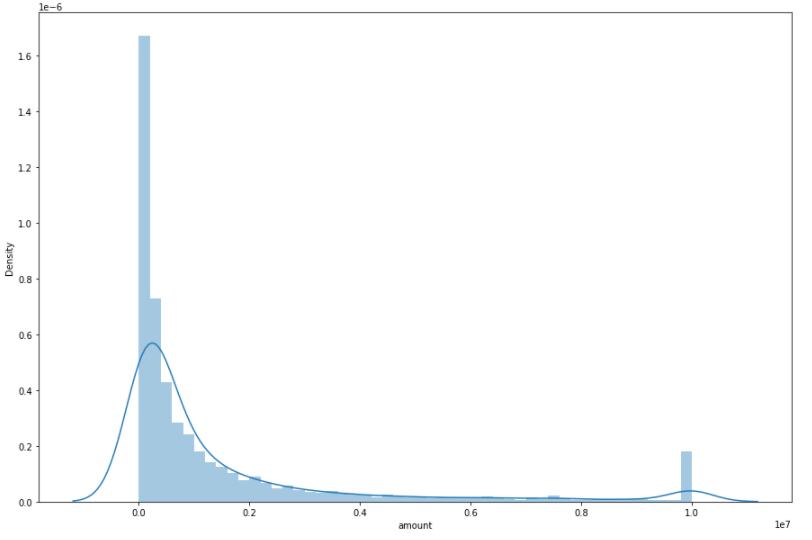
With respect to days, hour 0 does not necessarily mean 1am in the morning.

If hour 0 is 9am, then hour 1 is 10 am, hour 2 is 11am and so on...

From the graphs above, there is strong evidence to suggest that from hour 0 to hour 9 (inclusive) valid transactions very seldom occur. On the other hand, fraudulent transactions still occur at similar rates to any hour of the day outside of hours 0 to 9 (inclusive).

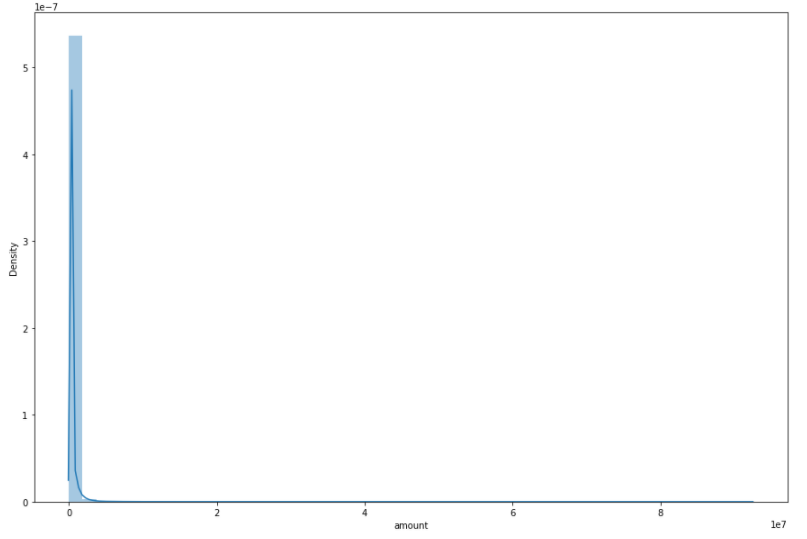
In response to this, we create another feature HourOfDay, which is the step column with each number taken to modulo 24.

### Relationship between amount and isFraud



The distplot above maps the density of amount during fraudulant transactions.

According to distplot above, we notice high density of small amounts and large amounts of money in  fraudulant transaction. We can interpret this as follows: People with low amount transactions have low  balance, therefore they don't give much attention to securing their acounts and therefore more susceptible to fraud. On the other hand, rich people make bigger amount transactions and therefore they attract  cyber thiefs.



The next displot above maps the density of amount during valid transactions. No further conclusions can be noticed except the fact that majority of transactions accure with low amounts of money. The majority of people in the dataset are not wealthy ^^

### Other remarks : Transactions with zero balances

The data has several transactions with zero balances in the destination account both before and after a non-zero amount is transacted. The fraction of such transactions, where zero likely denotes a missing value, is much larger in fraudulent (49.55%) compared to genuine transactions (0.06%). For instance:

* The percentage for fraudulent transactions that has zero balances in the destination account both before and after a non-zero amount is : 0.4955558261293072
* The percentage for valid transactions that has zero balances in the destination account both before and after a non-zero amount is : 0.0006176245277308345

Since the destination account balances being zero is a strong indicator of fraud, we do not impute the account balance (before the transaction is made) with a statistic or from a distribution with a subsequent adjustment for the amount transacted. Doing so would mask this indicator of fraud and make fraudulent transactions appear genuine. Instead, we replace the value of 0 with -1 which will be more useful to a suitable machine-learning (ML) algorithm detecting fraud.

## Skewness and kurtosis of numerical features :

**Skewness** is a measure of symmetry, or more precisely, the lack of symmetry. A distribution, or data set, is symmetric if it looks the same to the left and right of the center point. **Kurtosis** is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution. In general, ML models work better on symmetric and normally distributed data. Hence, we will evaluate current values of skewness and kurtosis and apply The boxcox function to reduce their values and get a better distribution of data

The grid below shows current values of skewness and kurtosis associated with numerical features of the data.

|  |  |  |
| --- | --- | --- |
| feature | Skewness | Kurtosis |
| amount | 21.93 | 868.55 |
| oldbalanceOrg | 70.01 | 9402.87 |
| newbalanceOrig | 127.31 | 27926.00 |
| oldbalanceDest | 17.16 | 668.81 |
| newbalanceDest | 16.43 | 585.02 |

We apply the boxcox function on the previous features :

|  |  |
| --- | --- |
| feature | Skewness after boxcox |
| amount | 0.08 |
| oldbalanceOrg | 0.08 |
| newbalanceOrig | 2.96 |
| oldbalanceDest | -0.32 |
| newbalanceDest | 0.15 |

The skewness is far better after boxcox transformation. However, we end up with the appearance of negative values in all the features above, which makes no sense since we are dealing with positive amounts and balances. Also, the transformed data did poor in the ML models compared to the original data. Hence, we will use original data in the ML models even if it has high kurtosis and skewness.

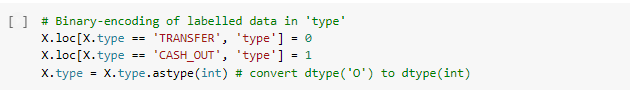
## Feature engineering

**We use Feature engineering** to improve the performance of our machine learning algorithm. Two additional features that seem interesting are the balance errors in the original and destination accounts.

The ML model can compare these balance errors to other features (amount, new and old balance) to identify fraudulent transactions from valid ones.

## Binary encoding

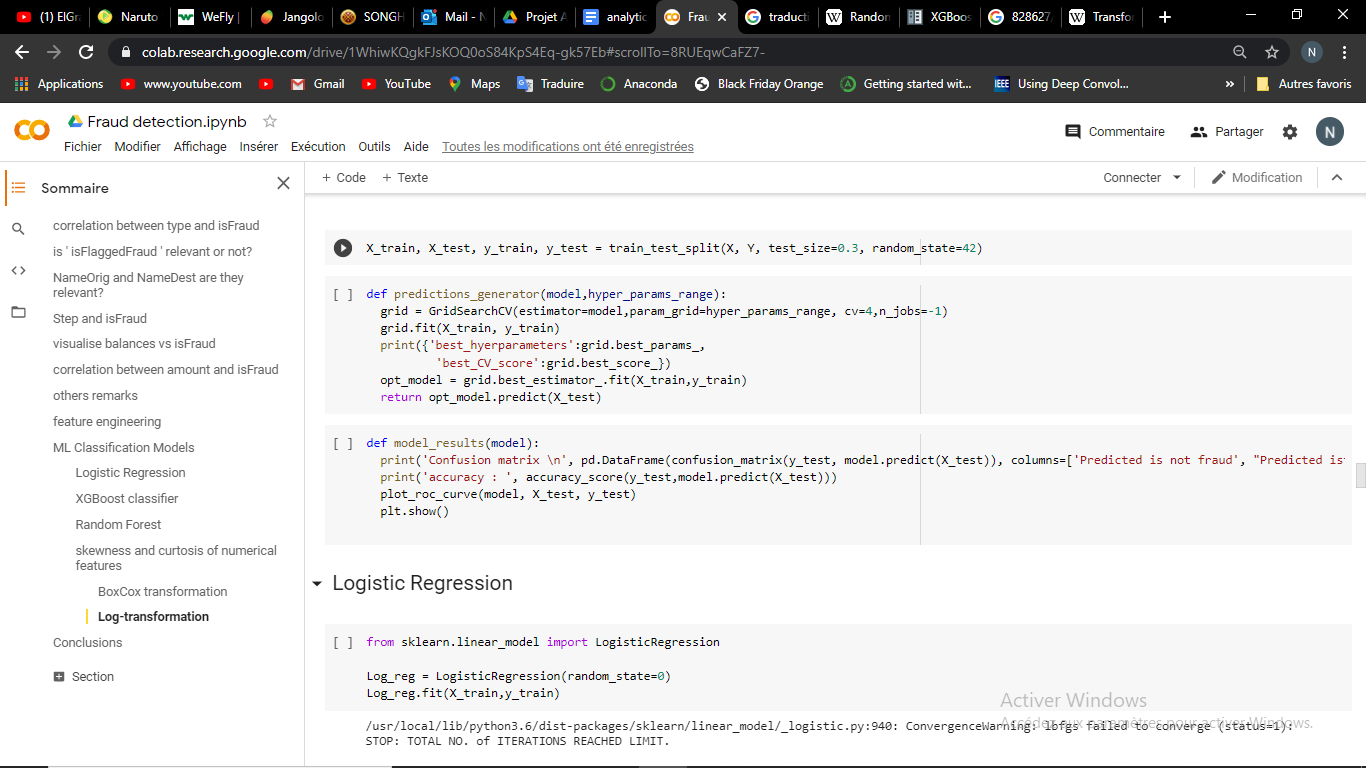
Since we have chosen to work only on two types: transaction and cash-out in our dataset, we apply binary encoding by replacing transfer with 0 and cash-out with 1. This approach is more beneficial because the ML model will operate only on numerical values.



## Machine Learning Models

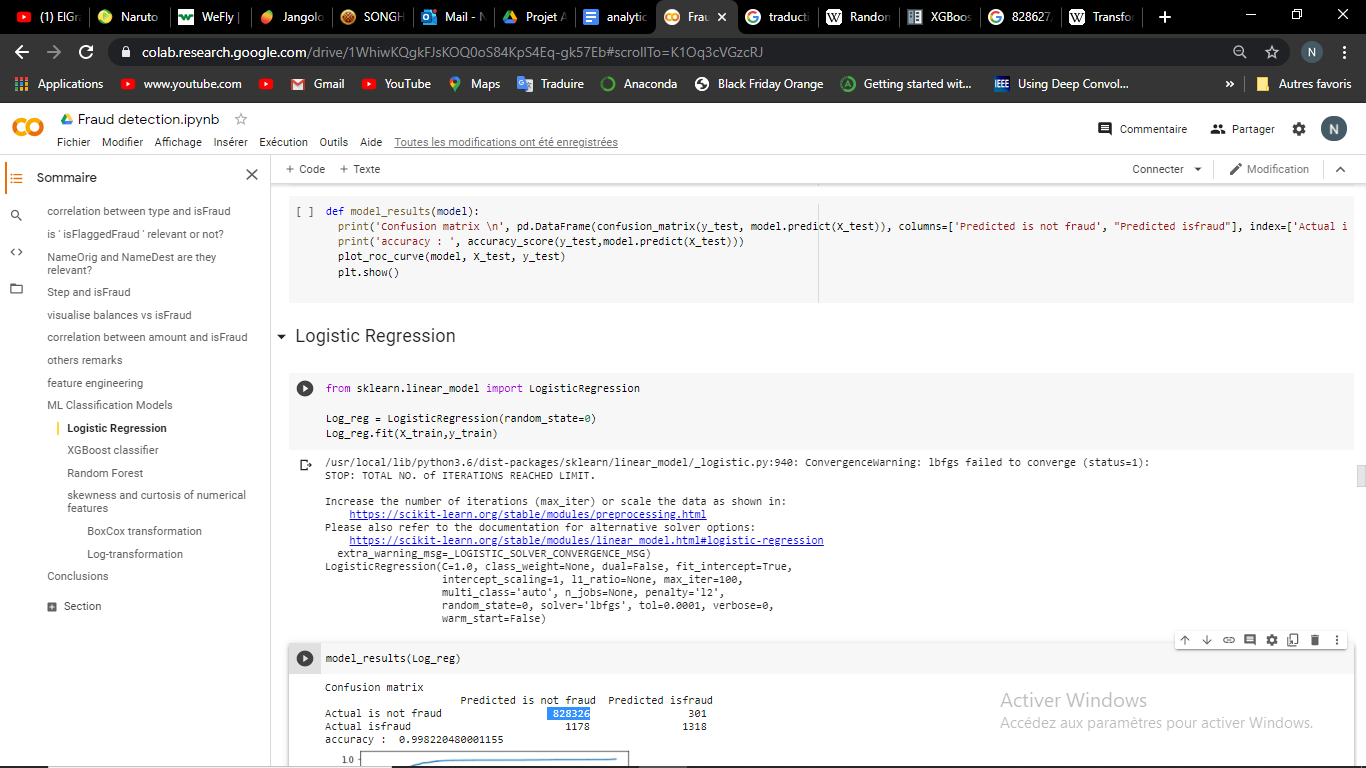
After having prepared our working data frame, we have trained some Classification Algorithms to make our predictions.

We split our data into 70% training set and 30% testing set.



1. **Logistic regression**

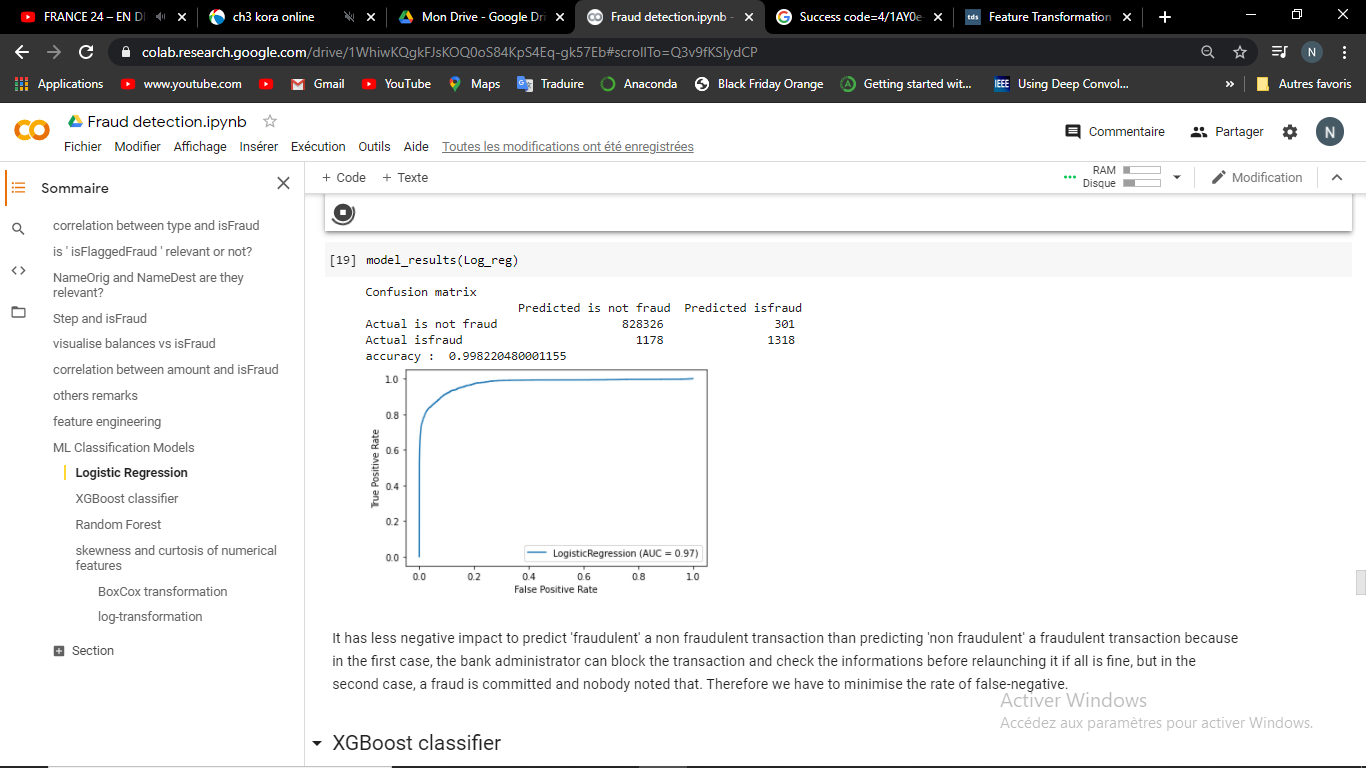
Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In the logistic model, the logit function for the value labelled "1" is a linear combination of one or more independent variables.



* **Results**
* **Accuracy**: 0.998220
* **Confusion matrix**:

|  |  |  |
| --- | --- | --- |
| ***isfraud*** | Predicted negative (0) | Predicted positive (1) |
| Actual negative (0) | 828326 | 301 |
| Actual positive (1) | 1178 | 1318 |

* **ROC curve**



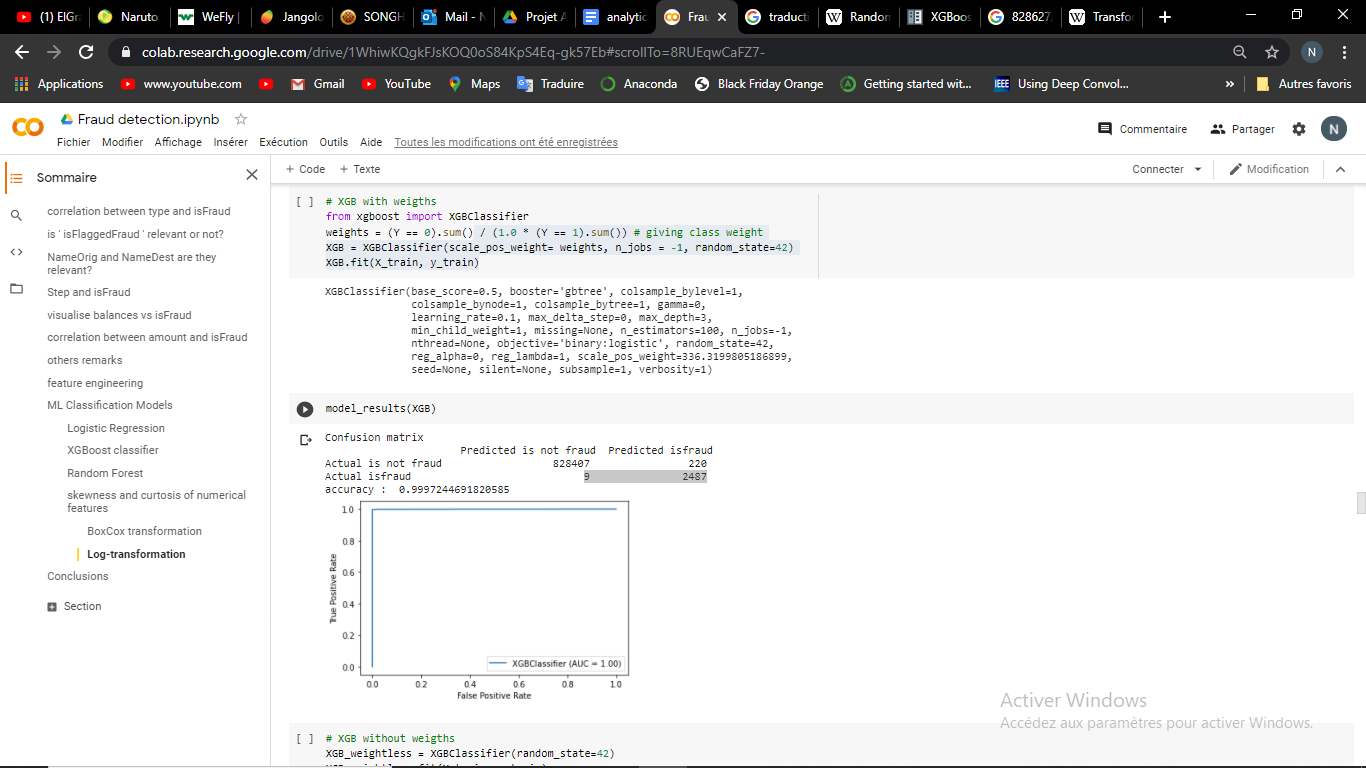
1. **XGBoost Classifier**

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. This ensemble method seeks to create a strong classifier based on previous ‘weaker’

classifiers. By adding models on top of each other iteratively, the errors of the previous model are corrected by the next predictor, until the training data is accurately predicted or reproduced by the model.

In our work, we have defined two forms of XGBoost classifier. For one we set the parameter *scale\_pos\_weight* to the ratio sum of negative instances on sum of positive instances. This is to Control the balance of positive and negative weights, which can be useful for unbalanced classes as ours (99.7% of our observations has ‘isfraud’ equals to 0). For the second, we did not precise that parameter. For others hyperparameters, we used default ones in both cases.

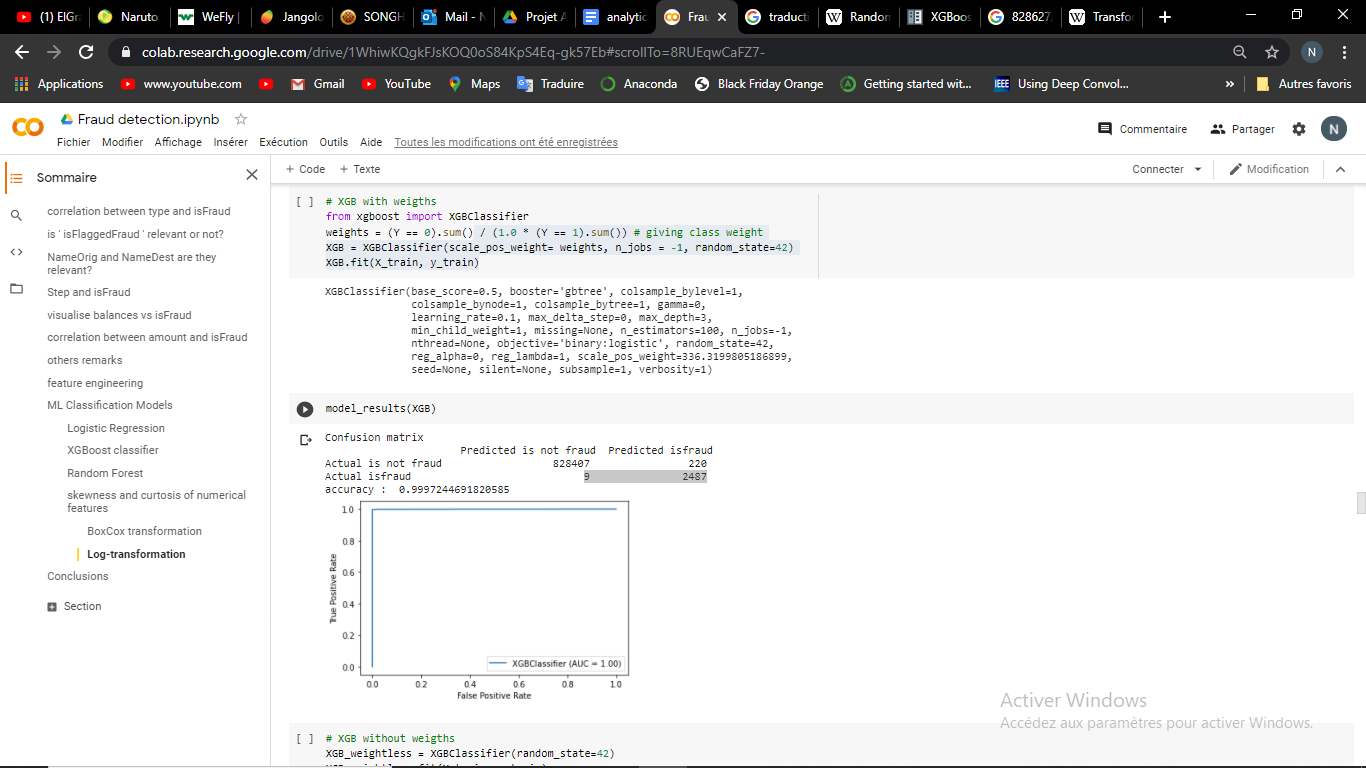
* Results of the weighted-XGBoost



* **Accuracy**: 0.999724
* **Confusion matrix**:

|  |  |  |
| --- | --- | --- |
| ***isfraud*** | Predicted negative (0) | Predicted positive (1) |
| Actual negative (0) | 828407 | 220 |
| Actual positive (1) | 9 | 2487 |

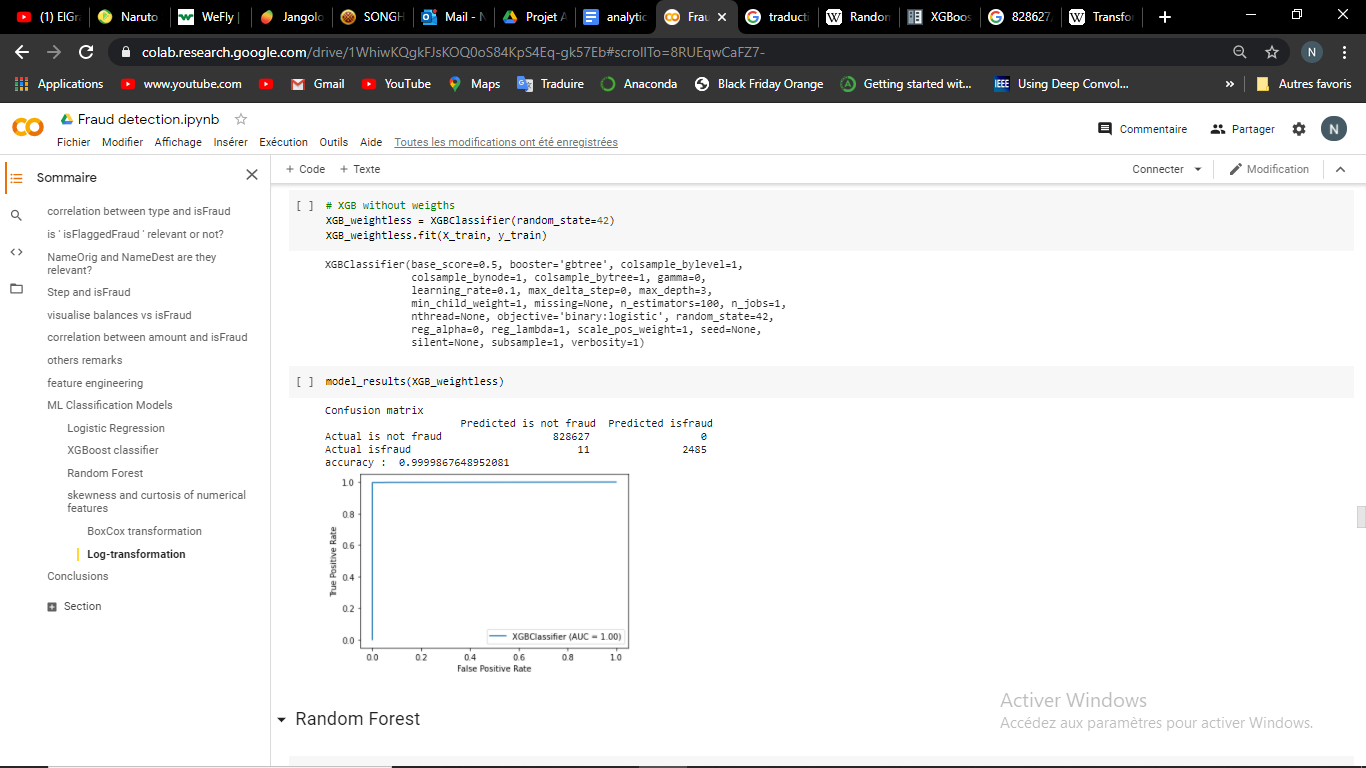
* **ROC curve**

****

* Results of the weightless-XGBoost
* **Accuracy**: 0.999986
* **Confusion matrix**:

|  |  |  |
| --- | --- | --- |
| ***isfraud*** | Predicted negative (0) | Predicted positive (1) |
| Actual negative (0) | 828627 | 0 |
| Actual positive (1) | 11 | 2485 |

* **ROC curve**

****

1. **Random Forest Classifier**

Random forest Classifier is an ensemble learning method for classification that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) of the individual trees.

* **Results**



* **Accuracy**: 0.999983
* **Confusion matrix**:

|  |  |  |
| --- | --- | --- |
| ***isfraud*** | Predicted negative (0) | Predicted positive (1) |
| Actual negative (0) | 828626 | 1 |
| Actual positive (1) | 13 | 2483 |

* **ROC curve**



All the four algorithms performed very well in terms of accuracy. However, the general accuracy here can not be considered as the best performance indicator for two reasons:

* Our data is highly unbalanced between ‘0’ and ‘1’ classes. Therefore, since the all our models are very good at predicting negative values (the rate of false positive is marginal compared to the rate of true negative), added to the fact that more than 99.5% of our data set is negative, the poor result of a model on predicting positive values (‘isfraud’ = 1) are hardly perceived through the accuracy.
* Intuitively we can say that It has less negative impact to predict a non-fraudulent transaction as 'fraudulent' than predicting 'non fraudulent' a fraudulent transaction because in the first case, the bank administrator can block the transaction and have a check of information before relaunching it if all is fine. But in the second case, a fraud is committed, and nobody noted that. Therefore, our model must minimise the rate of false-negative (ideally minimise at the same time the rate of false-positive).

For those reasons, Random forest classifier and XGBoost classifiers can be considered more efficient.

**Note!**

All those algorithms were implemented with their default hyperparameters, except *scale\_pos\_weight* for one of the XGBoost. In view of the very high scores reached, we did not perform further hyperparameters tuning, considering that the default ones are good. However, to ameliorate our results, we focused on the Skewness of our data and the transformation we can applied on it to perform better in predictions.

## Data skewness – data transformation

**Skewnes**s is a measure of symmetry, or more precisely, the lack of symmetry. A distribution, or data set, is symmetric if it looks the same to the left and right of the center point. **Kurtosis** is a measure of whether the data are heavy-tailed or light-tailed relative to a normal distribution. In general, ML models work better on symmetric and normally distributed data. Hence, we will evaluate current values of skewness and kurtosis and apply the boxcox function to reduce their values and get a better distribution of data.

The module scipy.stats has the functions skew and kurtosis which allow to calculate the quantities of which they have the name.

The grid below shows current values of skewness and kurtosis associated with numerical features of the data.

|  |  |  |
| --- | --- | --- |
| feature | Skewness | Kurtosis |
| amount | 21.93 | 868.55 |
| oldbalanceOrg | 70.01 | 9402.87 |
| newbalanceOrig | 127.31 | 27926.00 |
| oldbalanceDest | 17.16 | 668.81 |
| newbalanceDest | 16.43 | 585.02 |

**NB:** skewness study only concern features related to amount of money.

We can see that our data is highly skewed. We will perform some data transformation to reduce that skewness.

1. **Boxcox transformation**

For every x positive value, the boxcox transformation is defined by:

The boxcox function automatically calculate the best parameter

We apply the boxcox function on the previous features:

|  |  |
| --- | --- |
| **feature** | **Skewness after boxcox** |
| **amount** | 0.08 |
| **oldbalanceOrg** | 0.08 |
| **newbalanceOrig** | 2.96 |
| **oldbalanceDest** | -0.32 |
| **newbalanceDest** | 0.15 |

The skewness is far better after boxcox transformation.

* **Models’ results**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **accuracy** | **True Negative** | **False Negative** | **False Positive** | **True positive** |
| **Logistic regression** | 0.998246 | 82 8585 | 1415 | 42 | 1081 |
| **Weighted-XGBoost** | 0.982546 | 814134 | 13 | 14493 | 2483 |
| **Weightless-XGBoost** | 0.999358 | 828608 | 514 | 19 | 1982 |
| **Random Forest** | 0.999465 | 828592 | 409 | 35 | 2087 |

Even though BoxCox considerably reduces the skewness of each column, predictions are not improved compared to the normal data case. This is probably due to the fact boxcox is applied to each feature independently to others. Therefore, the scale between features are no more respected (for example the definition of balances errors is skewed; a formally positive balance can become negative and vice versa).

Now we will try to apply the same transformation to all the features to preserve that relative scale while reducing the skewness of data.

1. **Log transformation**

It consists in replacing a ‘x’ value by log(1+x). This transformation works for our features because they are all right-skewed. It has the advantage of keeping a formal 0 value to 0.

This are the new skewness of our features after the log-transformation.

|  |  |  |
| --- | --- | --- |
| **feature** | **Skewness after boxcox** | **Skewness after log** |
| **amount** | 0.08 | -0.69 |
| **oldbalanceOrg** | 0.08 | 0.09 |
| **newbalanceOrig** | 2.96 | 2.78 |
| **oldbalanceDest** | -0.32 | -1.70 |
| **newbalanceDest** | 0.15 | -2.96 |

we can note that the log-transformation is a little less efficient than the boxcox transformation in terms of reducing the skewness.

* **Model’s results**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **accuracy** | **True Negative** | **False Negative** | **False Positive** | **True positive** |
| **Logistic regression** | 0.998437 | 828464 | 1136 | 163 | 1360 |
| **Weighted-XGBoost** | 0.999720 | 828404 | 9 | 223 | 2487 |
| **Weightless-XGBoost** | 0.999984 | 828627 | 13 | 0 | 2483 |
| **Random Forest** | 0.999986 | 828627 | 11 | 0 | 2485 |

By doing a comparison between the false-negative and the false-positive results on our three datasets (normal, boxcox-ed and log-ed), we can note that the log-transformed data present the best results since it gives the best compromise between those two quantities for the 4 models. However, the difference with the data’s normal format is not that too big.

## Conclusion

The models we have trained with the aim to predict if a bank transaction is fraudulent or not are very precise, but not perfect. The choice to work with one and not the others in real life cases depends on the cost of a false negative VS the cost of a false positive mistake.

In this line of ideas, it can be relevant to make a comparative study between the cost of increasing our false negative rate by 2, and the cost of increasing our false positive rate by 223 (Randomforest VS XGBoost with weights applied on log-transformed data).

These costs can be assessed in terms of the bank brand image, telephone costs and energy for verifying information, working time, or even legal costs.