Bitcoin Fraud Detection System

1.Introduccion

a) Problem statement

Pretty much Bitcoin transactions are technically irreversible and this makes them very attractive for merchants because the received funds are immediately spendable.

Our Goal:

It is extremely important to have a fraud detection system in BTC transactions and, if possible, to work online to detect any abnormality in time, that is the objective of this work.

b)Background

Losses from cryptocurrency theft, hacks, and fraud fell 57% last year to $ 1.9 billion, as market participants boosted security systems, but crime in the 'decentralized finance' space continued to grow, a report from crypto intelligence company CipherTrace showed.
Criminals got away with a record $4.5 billion in 2019 in the crypto market.

Fraud was the dominant cryptocurrency crime in 2020, followed by theft, and ransomware. Half of all thefts, or about $129 million, were hacks tied to decentralized finance (DeFi), which are transactions on platforms that facilitate lending outside of banks.

Cryptocurrencies have attracted renewed scrutiny and interest as institutional investors have piled into digital assets, particularly bitcoin, propelling the latter to a record high of $42,000 in January 2021.

Goal:

Therefore, a fraud detection system in the BTC is necessary to alert us and prevent the fact from being consumed.

I began to investigate the state of the art in this field and at least I was able to extrapolate some ideas of ML models used in the detection of Fraud for Credit cards and in general for the semi-supervised processing of unbalanced classes.

First of all I needed to analyze the data on BTC that I had on hand for that I use Kaggle https://www.kaggle.com/ellipticco/elliptic-data-set
I analyzed 3 files that I describe below

2. Datasets

Elliptic_txs_classes.csv : File with the Classes: Legal equal 2 , ILlegal equal 1 and unknown
Elliptic_txs_edgelist.csv : File for edges, I use to see how the frauders work
Elliptic_txs_features.csv : File that has all the features already scaling, I guess with PCA to protect the clients and sensitive information

All the characteristics are taken in a period of 2 years and due to the cost of the labeling we have some transactions labeled as Legal and Illegal, we would say as a (1/5) and the rest most of the transactions are unknown (4/5) of the total data. Which provides an imbalance in the data and shows us the possible use of semi-supervised methods.

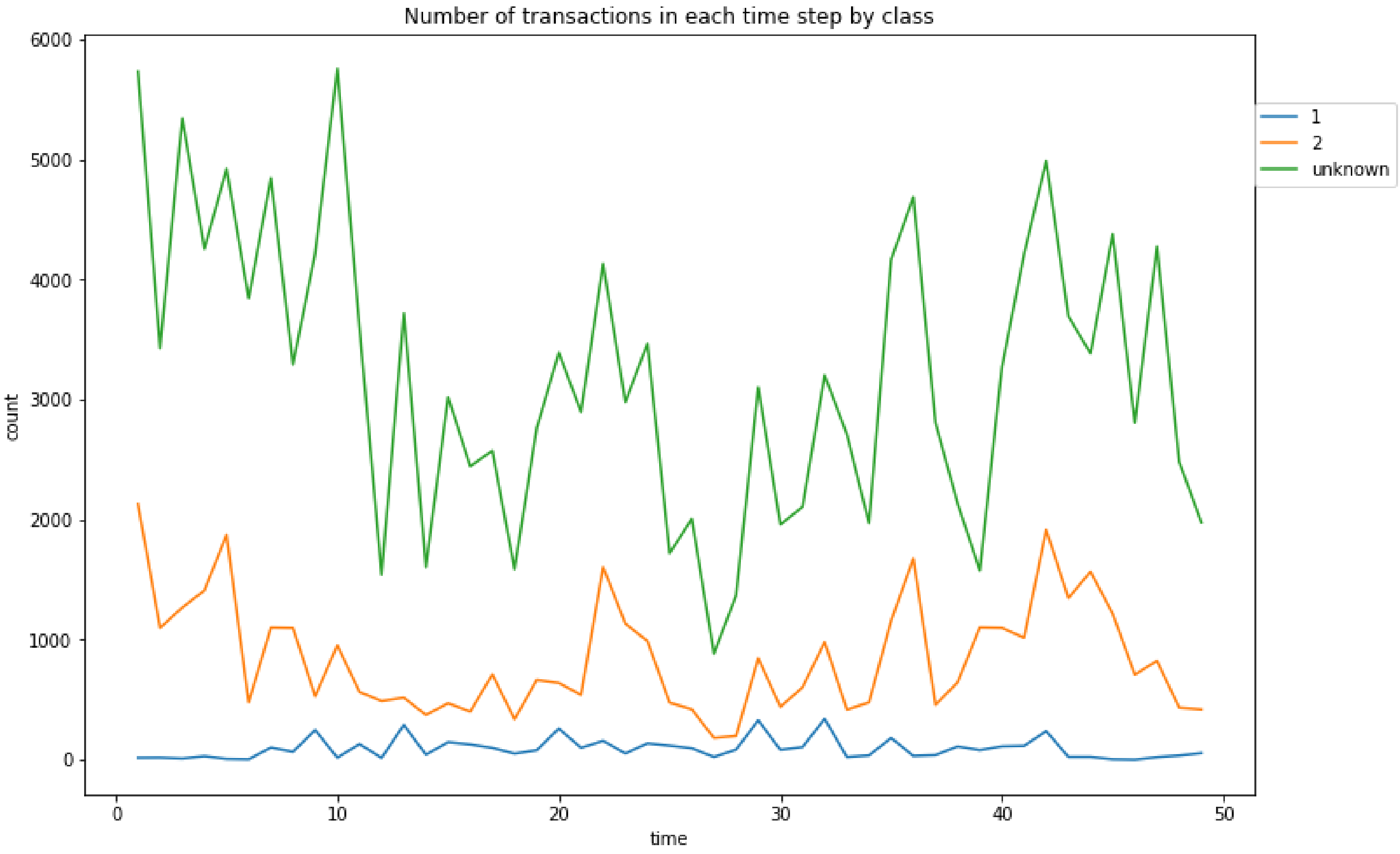
Therefore the interesting column here is Class in the elliptic_txs_classes.csv file and the features in the elliptic_txs_features.csv file.

3. Data Cleaning and Data Wrangling

After reading the csv files in different dataframes, we can make a merge between the features file and the classes file and thus obtain a single dataframe with both properties. I eliminated unnecessary columns as a result of the same merge and the most important thing is that The class unknown obtains real values of Legal and Illegal based on a random distribution that will maintain the relationship of 1/10 between both classifications. This will allow us in the end to be able to evaluate the results of the model in a more objective way, since the unknown class itself does not give us any information and it would not be helpful without a specific binary value of Legal or Illegal.

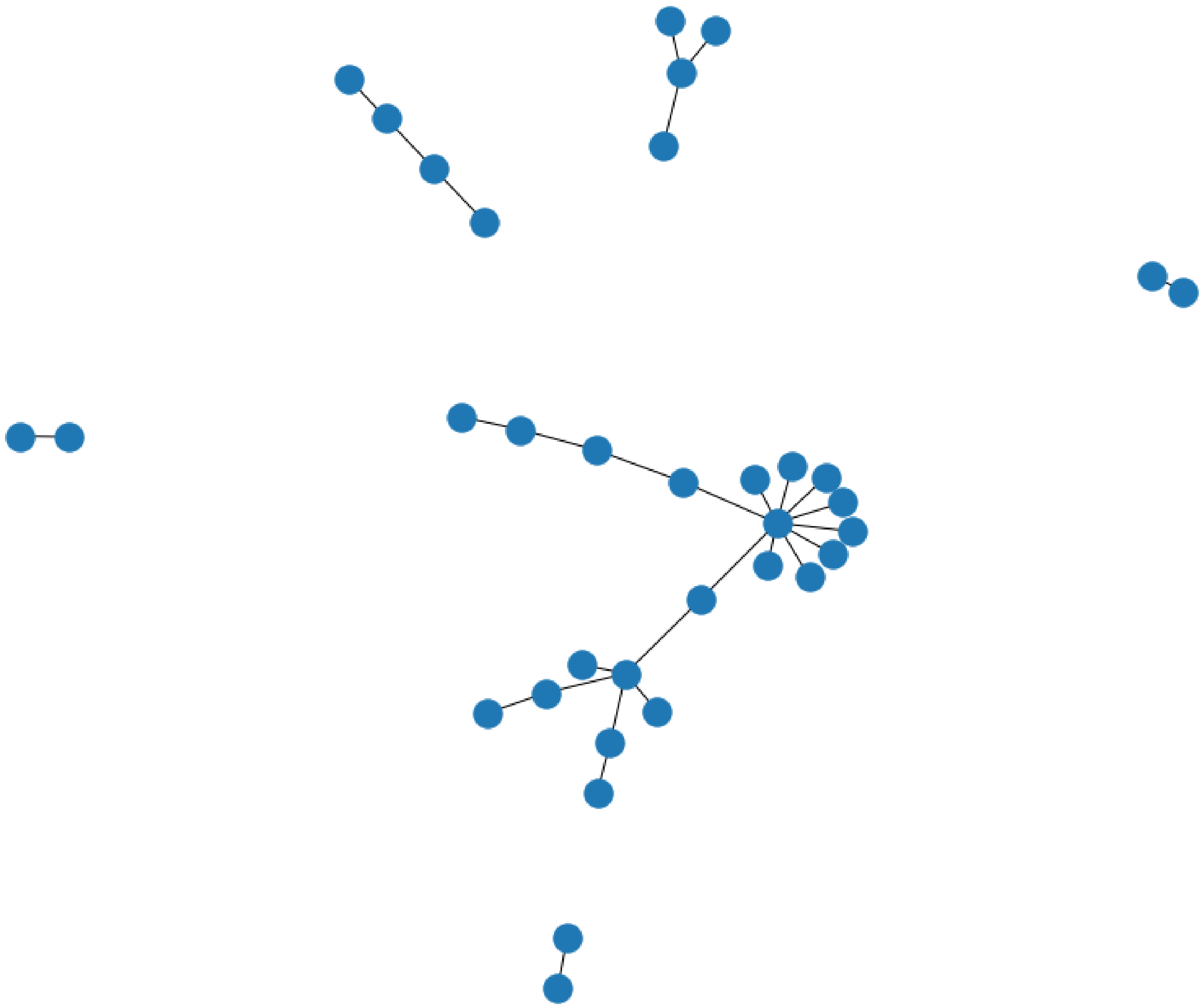
4. Exploratory Data Analysis

The exploratory analysis began trying to identify some feature that could identify Legal, Illegal and unknown transactions, the same first (trans_feat_0) shows us all the classes

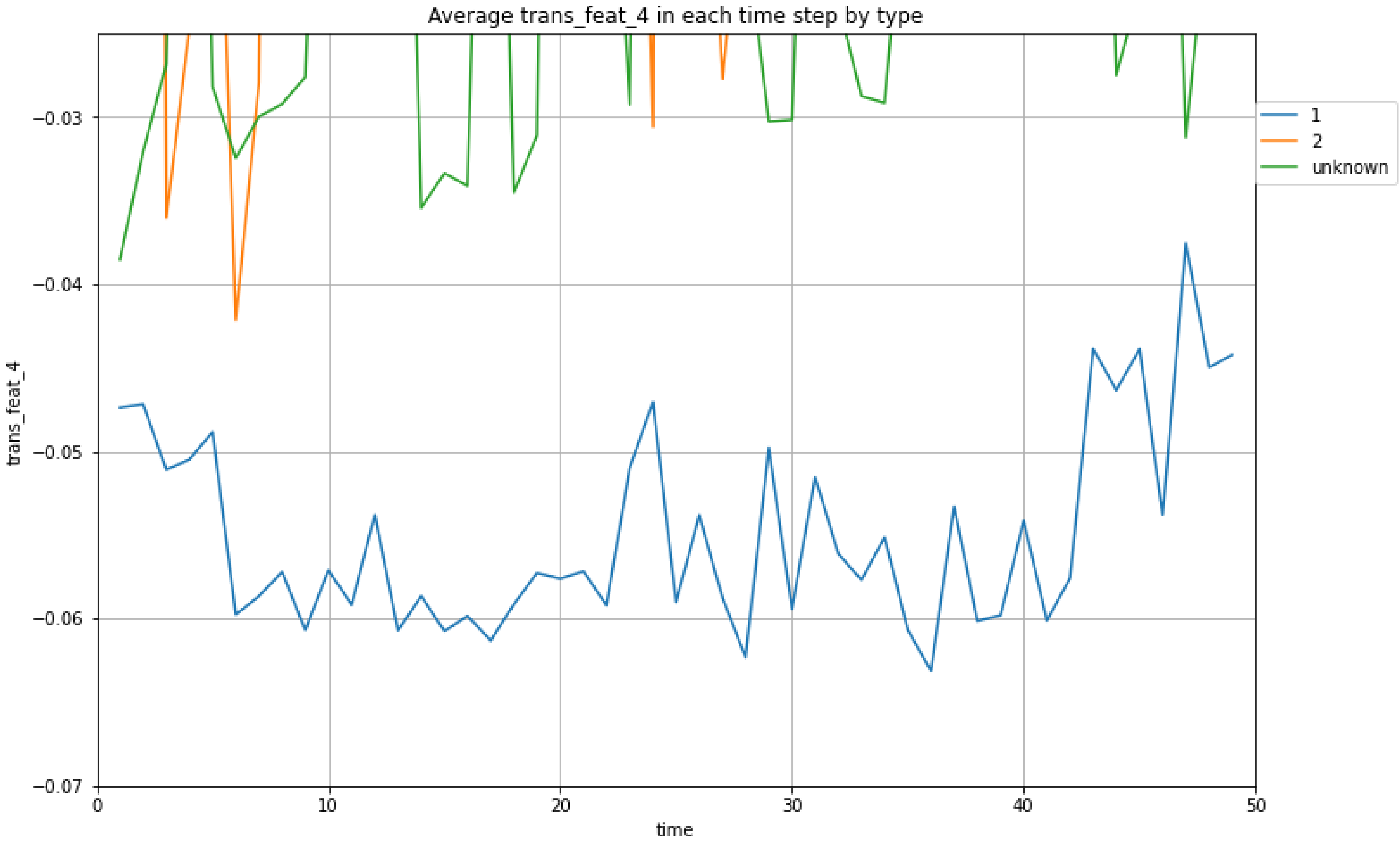


Class with value 1: ILegal
Class with value 2: Legal
Class with value Unknown: Is unknown

It is also interesting to know, based on another graph, that some of those who commit fraud do so in isolation but others do so in groups, which validates the importance of being able to detect these criminals in time.



I think the (trans_feat_4) function also shows us another interesting way to see how illegal transactions, for example, outnumber legal transactions only once in the entire time frame. Illegal ones between time step 46 and 48 exceed time step 6 of the legal ones



Anyway, all these graphs give me an idea but they cannot predict what may be Legal or Illegal with any precision or accuracy, for that I must go to the ML models

5. Algorithms & Machine Learning

I wanted to start my ML Analysis with the simplest models applicable to a semi supervised process and go looking for variants that will present better performance.
I started with the supervised method applied to our semi supervised. you can get the results here

https://github.com/GuyenSoto/BTC/blob/master/BTC_EDA_w_Metrics.ipynb

As can be seen, the results were not the best, the detection of illegals was practically null.

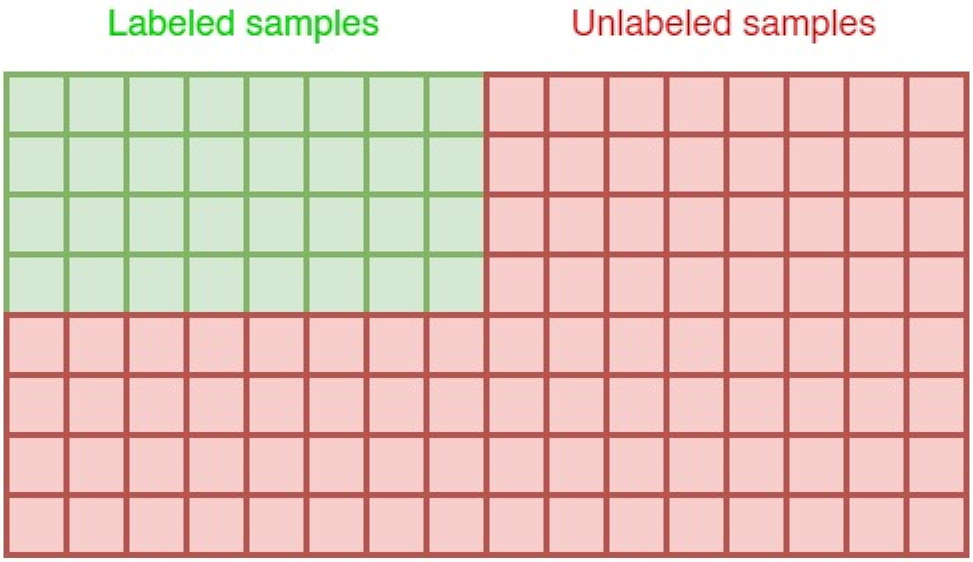
I also wanted to try the Label propagation and Label Spreading procedures that scikit Learn brings, and I don't think the results were good either. Without taking into account the large consumption of computational resources, where only 1/2 part of the DatFrame could be used for Label propagation and only 1/4 in Label Spreading, since above those values the consumption exceeded the 50 Gb of RAM allocated by Colab + . You can get the results here:

https://github.com/GuyenSoto/BTC/blob/master/BTC_labelPropagation.ipynb https://github.com/GuyenSoto/BTC/blob/master/BTC_labelSpreading.ipynb

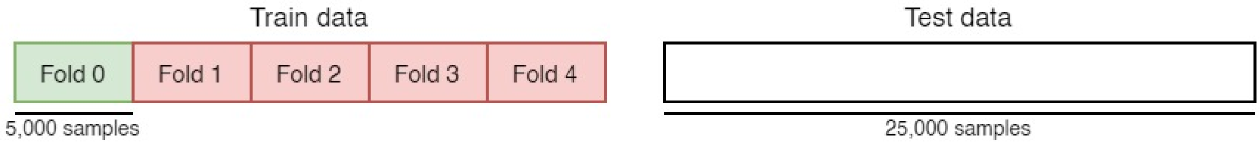
Finally I tried the combination of Autoencoders with Logistic Regression and the results improved incredibly. Which encouraged me to try to find different models of neural networks, even tried the encoder without the decoders to see if it could improve the performance of the model.

As I will expand this topic further, I want to explain first that I also tried the use of KFold, a solution used frequently in Cross Validation where the average is used to validate a model, but in this case it would be used with a different purpose, to create a structure using a partitioning of the data in different Folds and only one of them has known values, the rest is found from this first fold, since the output of the first fold would be the training data of the second fold and so on until the last fold is reached.

You can get the results here:
https://github.com/GuyenSoto/BTC/blob/master/BTC_KFolds.ipynb



This second figure shows the idea of using the data from the first fold



The second figure shows the idea of using the data from the first fold In my case I have a total sample of 203,769 and a quantity of 46564 data that can be labeled, as follows: 42019 as Legal transactions and 4545 as Illegal, a ratio of approximately 1/10.

The idea seemed exciting to me but after testing the model I did not obtain the desired results either, I tried different combinations of Neural Networks and their parameters, maybe it can be worked on and improved in the future for now I prefer to put it aside and work with the model that gave me better results, the Autoencoder with Logistic Regression

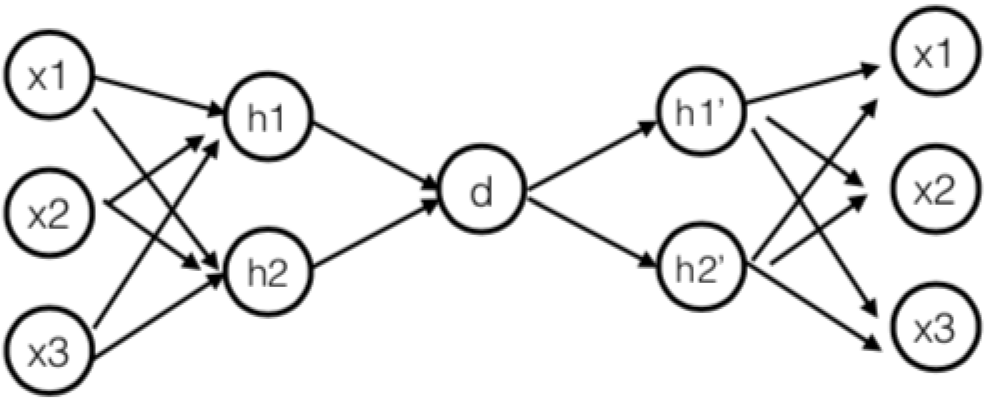
ML model using Autoencoders and Logistic Regression
You can get the results here:
https://github.com/GuyenSoto/BTC/blob/master/BTC_Autoencoders_with_1_Like_Fraud.ipynb

Autoencoder is a kind of Deep Learning architecture. Autoencoder architecture encompasses two subsystems as encoder and decoder. Both these subsystems are made up of independent Neural Networks with a defined set of layers and activation functions. The fundamental characteristic feature of Autoencoder architecture is extracting the latent(hidden) data points from the given dataset.

Autoencoders (AE) are a family of neural networks for which the input is the same as the output*. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation.

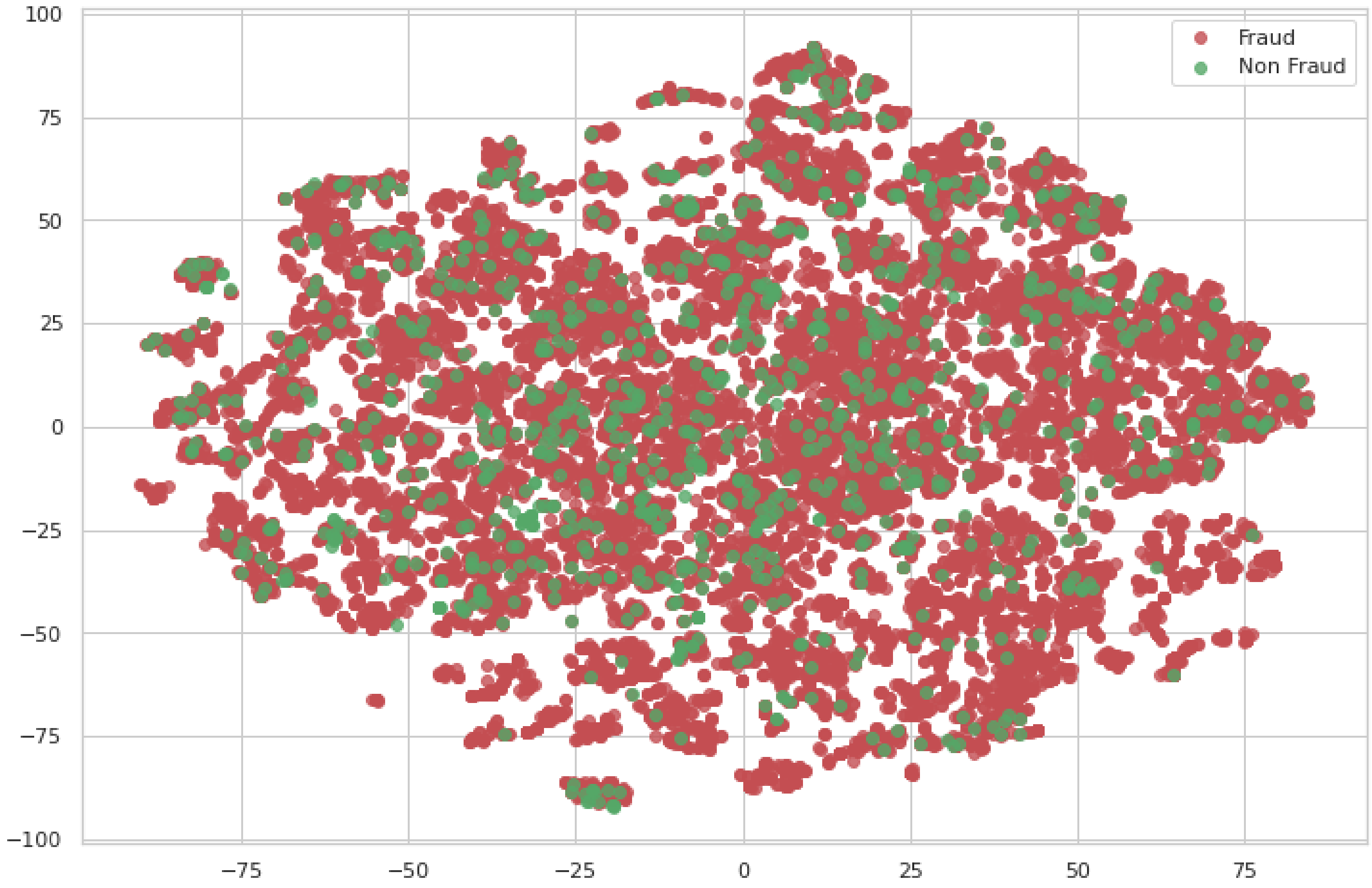
In a rough way an autoencoder serves to replicate the sample that I have at the entrance to the exit and due to the funnel in the center we can avoid noises at the exit and a more successful replication

They are an unsupervised learning method, although technically, they are trained using supervised learning methods, referred to as self-supervised. Usually they are restricted in ways that allow them to copy only approximately, and to copy only input that resembles the training data. Because the model is forced to prioritize which aspects of the input should be copied, it often learns useful properties of the data.

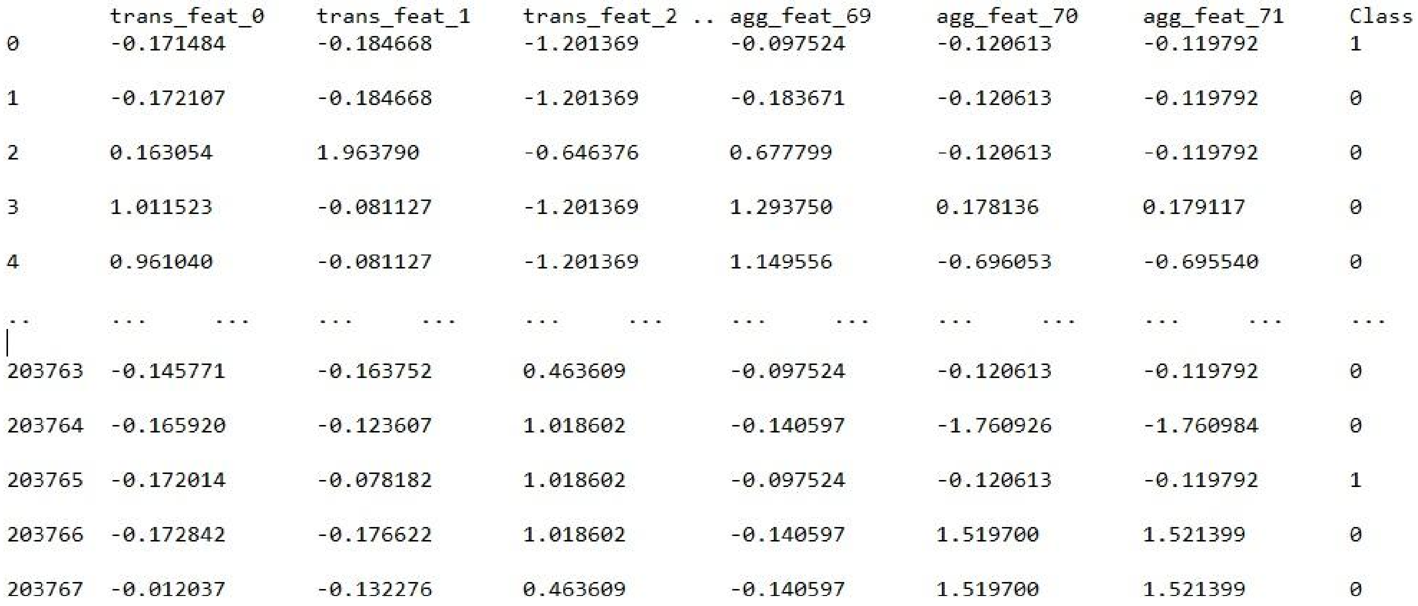


In my case I used a TSE function to sample the initial data and after being processed by the Autoencoder to get an idea of how it works.In my case, I used a TSE function to sample the initial data and after being processed by the Autoencoder to get an idea of how it works. I also used a Logistic Regression after the Autoencoder to finally predict what kind of Class my data set had, especially those considered unknown.

Before processing the data on the Autoencoder

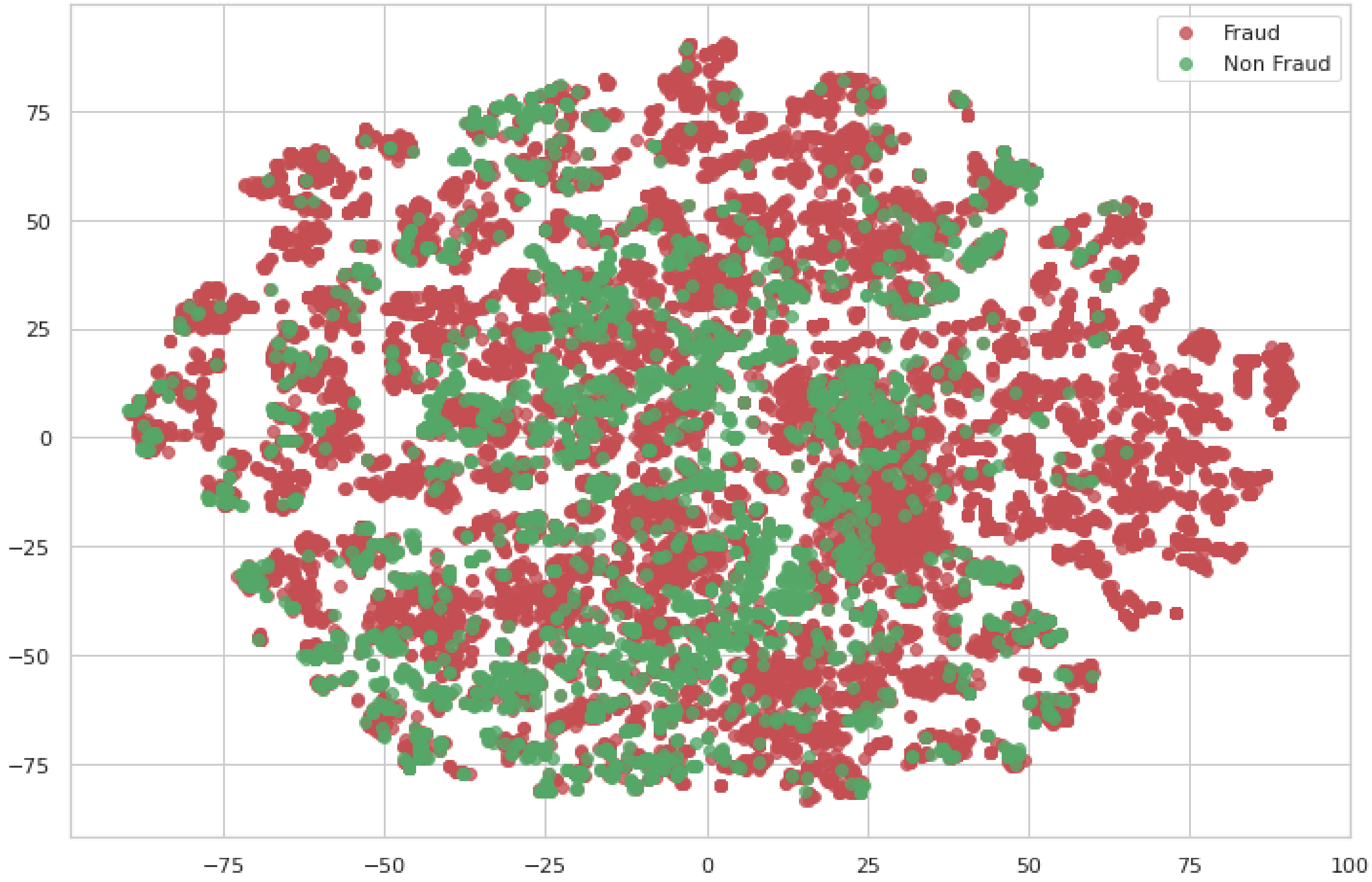


6. Dataset



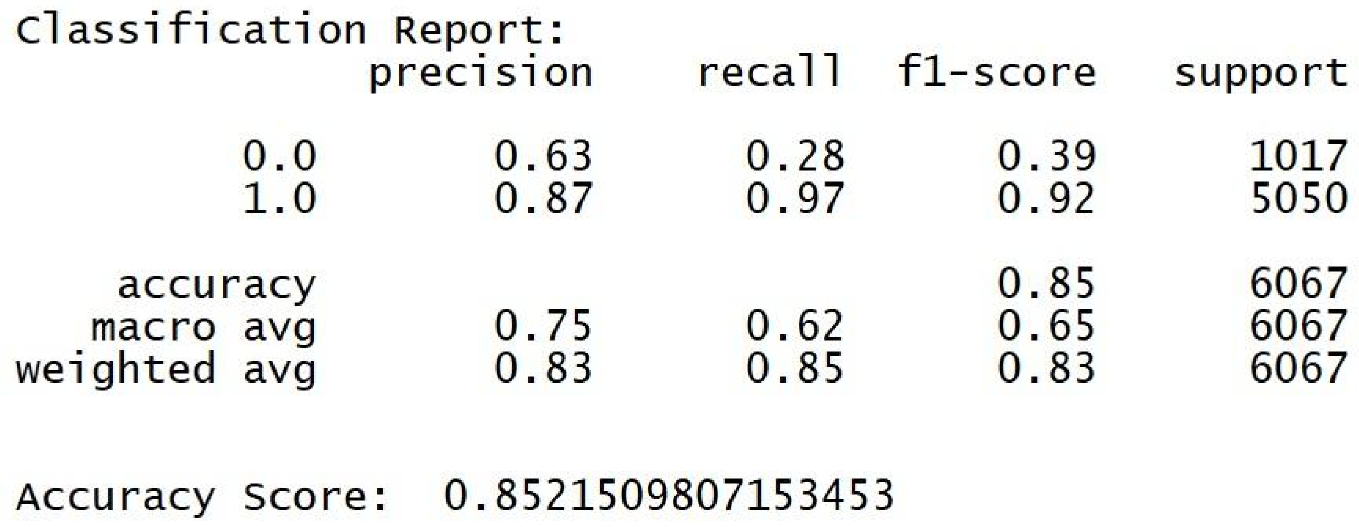
```
        trans_feat_0   trans_feat_1   trans_feat_2 .. agg_feat_69    agg_feat_70    agg_feat_71    Class
0       -0.171484      -0.184668      -1.201369      -0.097524      -0.120613      -0.119792      1

1       -0.172107      -0.184668      -1.201369      -0.183671      -0.120613      -0.119792      0

2       0.163054       1.963790       -0.646376      0.677799       -0.120613      -0.119792      0

3       1.011523       -0.081127      -1.201369      1.293750       0.178136       0.179117       0

4       0.961040       -0.081127      -1.201369      1.149556       -0.696053      -0.695540      0

..      ...     ...    ...     ...    ...     ...    ...     ...    ...     ...    ...     ...    ...

203763  -0.145771      -0.163752      0.463609       -0.097524      -0.120613      -0.119792      0

203764  -0.165920      -0.123607      1.018602       -0.140597      -1.760926      -1.760984      0

203765  -0.172014      -0.078182      1.018602       -0.097524      -0.120613      -0.119792      1

203766  -0.172842      -0.176622      1.018602       -0.140597      1.519700       1.521399       0

203767  -0.012037      -0.132276      0.463609       -0.140597      1.519700       1.521399       0
```

After processing in the Autoencoder



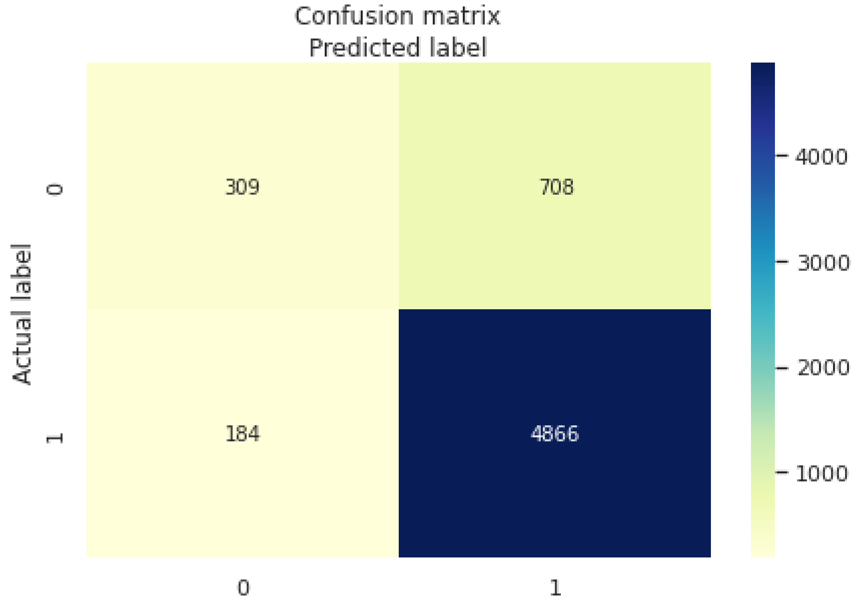
7. Predictions

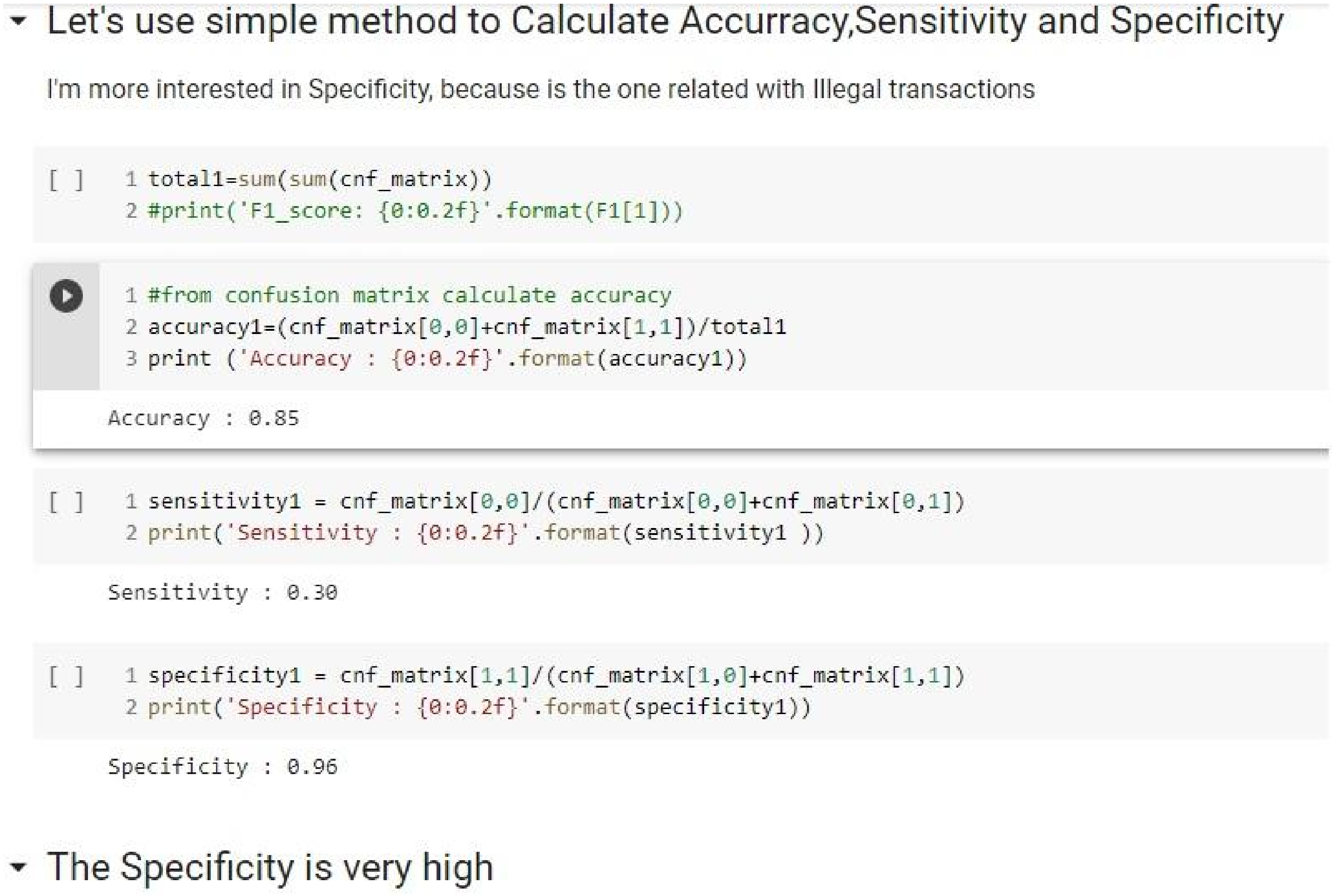
After applying Logistic Regression, I obtained the following results:



```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.63      0.28      0.39      1017
         1.0       0.87      0.97      0.92      5050

    accuracy                           0.85      6067
   macro avg       0.75      0.62      0.65      6067
weighted avg       0.83      0.85      0.83      6067


Accuracy Score:  0.8521509807153453
```

My Confusion Matrix





Let's use simple method to Calculate Accurracy,Sensitivity and Specificity

I'm more interested in Specificity, because is the one related with Illegal transactions

```
total1=sum(sum(cnf_matrix))
#print('F1_score: {0:0.2f}'.format(F1[1]))
```

```
#from confusion matrix calculate accuracy
accuracy1=(cnf_matrix[0,0]+cnf_matrix[1,1])/total1
print ('Accuracy : {0:0.2f}'.format(accuracy1))
```

```
Accuracy : 0.85
```

```
sensitivity1 = cnf_matrix[0,0]/(cnf_matrix[0,0]+cnf_matrix[0,1])
print('Sensitivity : {0:0.2f}'.format(sensitivity1 ))
```

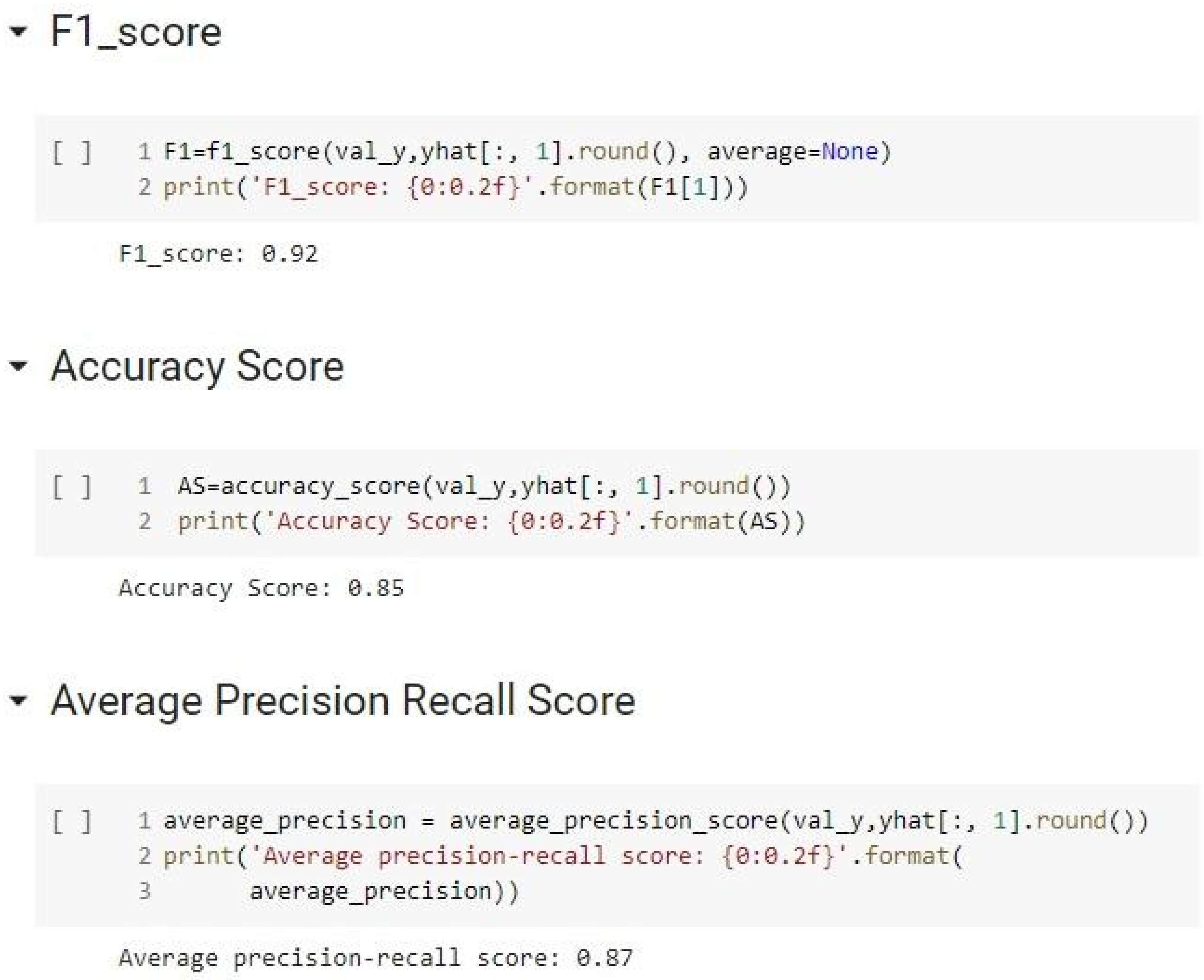
```
Sensitivity : 0.30
```

```
specificity1 = cnf_matrix[1,1]/(cnf_matrix[1,0]+cnf_matrix[1,1])
print('Specificity : {0:0.2f}'.format(specificity1))
```

```
Specificity : 0.96
```

The Specificity is very high

Because in my statement I wanted to detect the illegal transactions that in this case are identified as class 1, the Specificity parameter gives me an idea of how good my model can be at detecting it.



F1_score

```
F1=f1_score(val_y,yhat[:, 1].round(), average=None)
print('F1_score: {0:0.2f}'.format(F1[1]))
```

```
F1_score: 0.92
```

Accuracy Score

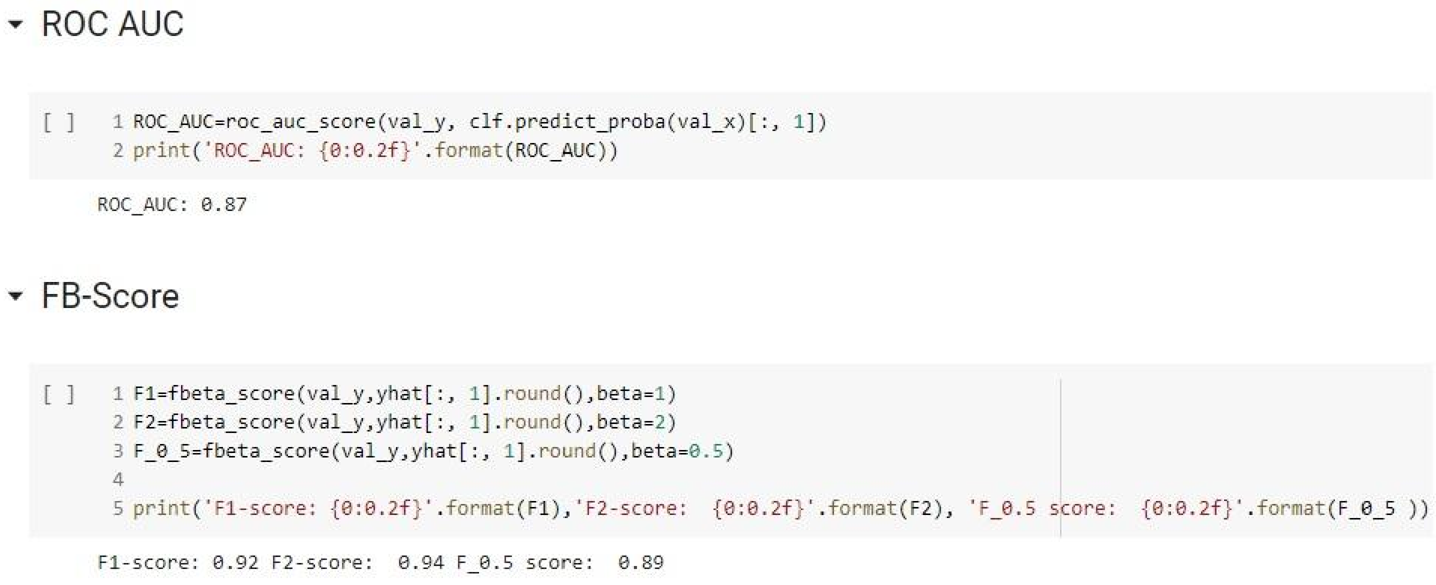
```
AS=accuracy_score(val_y,yhat[:, 1].round())
print('Accuracy Score: {0:0.2f}'.format(AS))
```

```
Accuracy Score: 0.85
```

Average Precision Recall Score

```
average_precision = average_precision_score(val_y,yhat[:, 1].round())
print('Average precision-recall score: {0:0.2f}'.format(
      average_precision))
```

```
Average precision-recall score: 0.87
```



ROC AUC

```
ROC_AUC=roc_auc_score(val_y, clf.predict_proba(val_x)[:, 1])
print('ROC_AUC: {0:0.2f}'.format(ROC_AUC))
```

```
ROC_AUC: 0.87
```

FB-Score

```
F1=fbeta_score(val_y,yhat[:, 1].round(),beta=1)
F2=fbeta_score(val_y,yhat[:, 1].round(),beta=2)
F_0_5=fbeta_score(val_y,yhat[:, 1].round(),beta=0.5)

print('F1-score: {0:0.2f}'.format(F1),'F2-score:  {0:0.2f}'.format(F2), 'F_0.5 score:  {0:0.2f}'.format(F_0_5 ))
```

```
F1-score: 0.92 F2-score:  0.94 F_0.5 score:  0.89
```

Summary

We can see how using Autoencoders and Logistic Regression can significantly improve the predictions of our Model for Illegal transactions, with the advantage that training it does not require much data and with the possibility of being able to train the model in an environment that begins with low volumes of information. .

I propose that the Model be tested in companies that work directly with Bitcoin, to check the real effectiveness of the system in a production environment.

8. Future Improvements

* In the future, I would like to spend more time on the behavior for the distribution of unknown classes. Maybe we can find a better array to replace the unknown class with a series that optimally reflects the unknown behavior. Although I think the current one definitely meets the requirements

* Due to the RAM limitations in Google Colab, I had to train a small sample of the original data set, although this is precisely the advantage of being able to use this ML model that doesn't really need a large set to get good representations.

* I would like to try the dataframe fast.ai since it serves to process and work with tabular data and maybe analyze more, if we could find any series that identify the data.
Check this link which uses Autoencoders and Tabular data in Fast.ai
https://walkwithfastai.com/tab.ae