Ethereum Fraud Detection

Final project for Data Mining course

Group number - 48

First name	Family name	ID num
Daniel	Raviv	206812851
Nadav	Yashar	313350498

Email address for correspondence:
Rabayev.daniel@gmail.com

ABSTRACT

As the world of online shopping and the online industry increases, people understood that the hottest place to invest their money in cryptocurrency.

The crypto caught the interest of mainstream media a few years ago, especially because it allows us anonymity while transferring money, and it allows us to make a transaction containing an almost unlimited amount of money, from every place in the world to every place in seconds.

There are some rules in cryptocurrency transactions that make the potential fraudulent more frequent, one of them is that after the transaction is executed, there is no option to cancel or get the money back from the receiver.

In addition, unlike our banks and official places to store our money, the crypto wallet status is available only to us. There is no need of informing the authorities about money transactions, and for that reason, crypto became a perfect place to store stolen money and money that came from a criminal source.

In this paper, we will analyze a dataset with several Ethereum transactions (a well-known cryptocurrency), and we will try to classify each transaction, whether it's fraud or not.

The classification will be due to various models, and we will evaluate each of its predictions, and then will choose the “winning” model.

Introduction

Background about the business problem

The main problem of the fraud transactions is that we don’t have the option to return the money from the people the money was stolen from them after the transaction was completed, because of the crypto rules. So, we have to figure out a way to classify each potential transaction before it is completed and try to help decide if the transaction should be completed or not. The main business that needs this kind of help is kind crypto wallets and crypto transfer sites, that provide a solution of money transfer. Whit this kind of help, those sites could stop the transaction before its completed due to potentially fraudulent activity, and this will provide the world a huge help, with reducing credit card stealing, and will protect those sites from potential chargebacks.

Literature review

The Internet contains a lot of papers analyzing these kinds of transactions, due to the big breakout of crypto in the last few years. Here [1] the writers try to describe and analyze the various ways there are today to take advantage of the unique attributes of crypto, they also present the blockchain technology, and the “art” of laundering money and how it connects to criminal organizations and cryptocurrency. In this [2] paper the writer describes the obstacle e-commerce systems have with protecting themselves from fraudulent activity and present an overview of all of these different difficulties.

In this paper, we can see that the researcher is also trying to find a model to classify each transaction, whether it's fraud or not, they used also the XGB boost algorithm, which we did not use.

Material and Methods

The Data

The dataset was taken from this user, and we will present the main features we decided to use, to make a classification on the label column – FLAG:

"FLAG" – The label – 1 stands for fraud and 0 is a legitimate transaction
Avg min between sent/received tnx
avg value of Ethereum received/sent
total transactions
total Ethereum sent/received
ERC20 average value received
Time Diff between first and last (Mins)
ERC20 most sent token type – 1 of the 2 categorical features in the dataset.

The dataset contains 51 columns, but we used only 35 for classification, and one for presenting the actual labeling.

As almost every fraud/non-fraud dataset, this one is also imbalanced (figure 1), with 77 % of legitimate transactions, so actually, even if we will choose a very stupid model with always choosing 0 for each transaction in the training, we already have at least 77 % accuracy, so we have to consider in few things – we need to look on the recall and precision, and not on the accuracy, because those values give us an actual feeling about what is the actual behavior of our model, and refers to the percentages we were right, out of the total fraud transactions. (Figure 1)

The Data we chose to model, contains 36 rows – 33 numerical values, 2 categorical features, and one label column.

Most of the numerical features have most of the values around 0, with a distribution that is not looking like the normal distribution, with potential outliers (Figure 2 and 3).

The methods

Preprocessing:

Dealing With Missing/Duplicate Data

First, we checked if there are null label values, so we could know how many null labels there are, and if it were too much so maybe we would not use this data set for analysis. The result was 0 nulls. Then we looked at the 2 categorical features, that seem to be important when modeling because they contain some company names. Those features contain a lot of null values and contain also both 'None' and '0' values. After checking about the meaning of the features, we got to the conclusion that there is a chance of not having value on this feature, so we made an imputation of all of the categorical null values to the mode, and the rows that have '0' value on those columns, turned to 'None', so the model will have only one of those values available.

The next step was to check the null values of the numeric columns. After finding all the null values (Figure 4) we also made an imputation, we changed the null values to the mean of each feature, we found that more useful than the median, as the data doesn't have some known distribution (Figure 5).

In addition, we found some duplicate rows that will harm our model, so we had to delete those rows also.

Data Validation

The next step is very important and has a high impact on our model – variance 0 (Figure 6). There are a few columns that we found (by the method described) a variance of 0, and what does it mean? It means that all of the values of this feature are actually a constant, and of course that we don't want a constant to evaluate our model, so we found all those features and some other features with variance highly close

to 0, and we deleted those features.

Correlation Analysis

After checking the correlation between each feature and the correlation of each column to the label, we did not find some strong correlation to the label – the highest correlation to the label was 0.26, and this means that we cannot predict the label due to a specific column, much better than the other. This classification will be depended on all of the features (Figure 7).

Next, we looked for finding the correlation between the features, in order to reduce the number of features we have. We found some features with a high correlation between them, and therefore we decided to remove one of the features in pairs of features with a correlation is higher than 95 % (Figure 8).

the exact number of features we deleted after this part is 6, and we stayed with 30 features

Also, given the correlation of the features to the label in Figure 7, we decided to drop all features with correlation less than 0.02 in its absolute value (we found that this number keeps model's results as good as they were before. We stayed with 18 features, and the label.

Normalization

In order to make the modeling as much as accurate we can, we used different techniques like standardization and min-max normalization and then we checked which of them has the best results in modeling. We found out that Min-Max Normalization had the best results in modeling, so we chose this method (Figure 9).

Data Reduction

Because of the fact that we have a quite big number of features, we tried to make a Data reduction with the PCA algorithm, and the results were not better than the original results (Figure 10).

Dealing With Outliers

After making our analysis, we left with 19 columns and the goal now is to find outliers that will improve our model and will remove noisy data points that can harm our prediction models.

To do so, we tried several approaches – the first approach was to find the IQR of each column, and then we tried to find if there are some features, that have less than outliers. We found only 1 feature that has less than 5 % outliers due to our IQR formula, but the results did not get better after removing the outliers from this feature (Figure 11).

so, we had to find different ways to figure out some outliers – we so what are the features that have the largest correlation to the label, and for that feature, we plotted 2 different box plots, one for positive label and the other for a negative label, where the main goal is to try finding some outliers, that were not 'placed' at the right place. This play was vital because the results of the model were much better after dropping a small number of outliers – the false-positive counter decreased from 1 to 0 and the false negative from 13 to 11 (Figure 12). We also plotted scatter plots between high correlative attributes to find outliers.

Label Encoding

As we wrote before, the data has only 2 categorical features, when 1 of them has a lot of none values (note – it's not a null value), so first, we thought that maybe the model will make a better prediction without the help of those 2 features. Then after evaluating the model, we made some label encoding, so we could use those 2 features. In fact, each of the 2 features has about 400 unique values, so one hot method is not really available here for us, due to its 'unique categories' limitation.

so, the right method was to use label encoding, it replaces each value from the list of values with a number, and so in this way, we would train the model with those 2 categorical attributes (the results were much better with considering the 2 features).

Learning Models and Results Analysis

In this part of the analysis, we used several different models, whereas in each model we tried different techniques to see whether the predictions can be improved or not.
we used 30 % of the data as a test and 70 % for training, we chose a relatively high number as the test set because we wanted the test to have enough positive samples, so we can determine better the results of our model.

Logistic Regression

The first classification model we chose was this model because our intuition was that the results for this model will be the worst (and we were right).
at the beginning of the train, we tried to use the data frame without making a normalization, then we tried to use the data frame which is normalized in 2 different ways – standardization and min-max normalization. In the end, we used the data after reduction to 15 features by the PCA algorithm.
the best results came with the data after min-max normalization.
In Figures 13 and 14 we can see that overall, we have an accuracy of 90%, but in fact, the results are not so good. If we remember that the data is imbalanced and most of the rows are labeled 0, then we understand that 273 False-Negative is not so good, and it brings us to recall of 58 %.

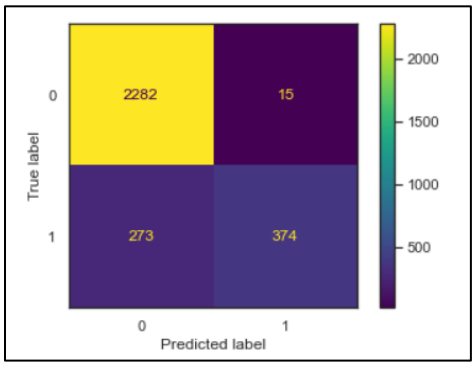

Figure 13 – Confusion Matrix for Logistic Regression Model

	precision	recall	f1-score	support
0	0.89	0.99	0.94	2297
1	0.96	0.58	0.72	647
accuracy			0.90	2944
macro avg	0.93	0.79	0.83	2944
weighted avg	0.91	0.90	0.89	2944

Figure 14 – Results Matrix for Logistic Regression Model

Note- the results for logistic regression model came with max iteration = 10,000

SVM

The next model is support vector machine – in this model we have some parameters that we wanted to find the best results respecting the recall, so we used grid search to the parameters “C” and “kernel” and the grid search made running and found the best parameters for our model. The results are better than the logistic regression model. The result demonstrated in the below figures show AUC of 98% which is great, and the recall made a huge jump from the logistic regression (from 58 % to 96 %).

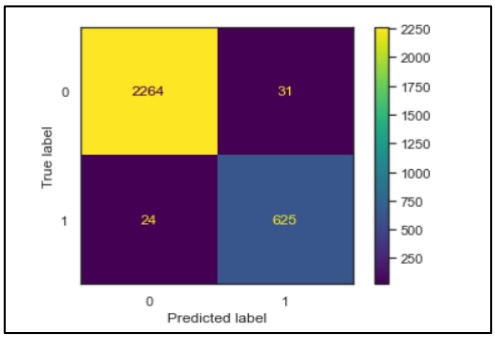

Figure 15 – Confusion Matrix for SVM Model

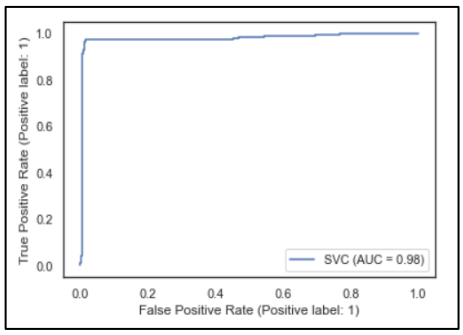

Figure 16 – ROC Curve for SVM Model

	precision	recall	f1-score	support
0	0.99	0.99	0.99	2295
1	0.95	0.96	0.96	649
accuracy			0.98	2944
macro avg	0.97	0.97	0.97	2944
weighted avg	0.98	0.98	0.98	2944

Figure 17 – Results Matrix for SVM Model

Random Forest

The big winner in our project is the Random Forest Model, which makes a lot of decision trees randomly, and this attribute makes this model better then the regular decision tree and less overfitting to the trained model. Using Normalization, and with the help of grid search algorithm, we found that the parameters that predict our model the best are – number of estimators = 25 and max features = 5.
The results we are showing in the figures below are the best results that we got from running the model several times – once with PCA, then with the data after standardization and the last is the winner using Min-Max Normalization. (In fact, the results are the best we found even in Kaggle so we may consider uploading it later ☺). Recall jumped to 98 % and precision to 100%.

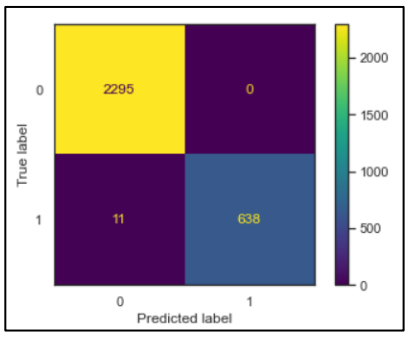

Figure 18 – Confusion Matrix- Random Forest Model

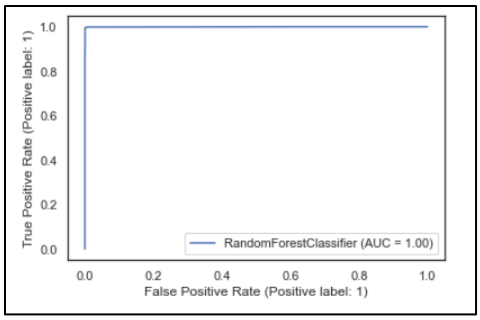

Figure 19 – ROC Curve for Random Forest Model

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2295
1	1.00	0.98	0.99	649
accuracy			1.00	2944
macro avg	1.00	0.99	0.99	2944
weighted avg	1.00	1.00	1.00	2944

Figure 20 – Results Matrix for Random Forest Model

Results Discussion

The Result of the last model seems to be too good to be true. After a long time, we tried to figure out whether the dataset is the real world or not, we believe that such good results probably mean that the data is made by a human, and it is not real data. We still know that there is a way to track the transactions because the transactions are available to the public.

Conclusions

In this project, we wanted to make a classification, and to understand which Ethereum transaction is a fraud, and which is not. We did EDA, we chose which attributes we should use to predict the model, and which features we should avoid. We had to deal with outliers in the data set, and we had to be aware of correlation issues, that can make us decide to avoid some features, that are highly correlated with others.
Then we used 3 different models – Logistic Regression, SVM, and Random Forest
The Random Forest Model did the best work for us. to conclude, we think that we answer the question well, and we could predict fraudulent transactions. For future work, we suggest adding the volatility of Ethereum to the consideration, for example in times there is the high volatility of the crypto, it may be the time that the fraudulent transactions are increasing.

References

- Sanz-Bas, D., del Rosal, C., Náñez Alonso, S. L., & Echarte Fernández, M. Á. (2021). Cryptocurrencies and Fraudulent Transactions: Risks, Practices, and Legislation for Their Prevention in Europe and Spain. Laws, 10(3), 57.
- Abdallah, A., Maarof, M. A., & Zainal, A. (2016). Fraud detection system: A survey. Journal of Network and Computer Applications, 68, 90-113.
- Ostapowicz, M., & Żbikowski, K. (2020, January). Detecting fraudulent accounts on blockchain: a supervised approach. In International Conference on Web Information Systems Engineering (pp. 18-31). Springer, Cham.

Appendix

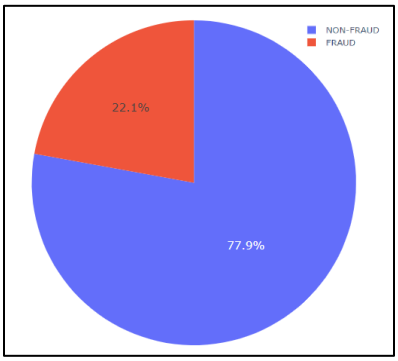

Figure 1 – Data distribution

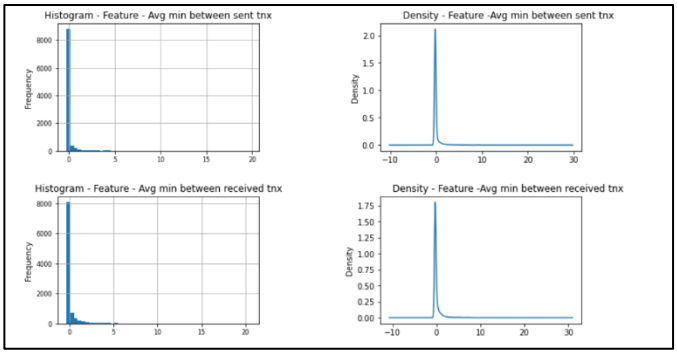

Figure 2 – Example of Distribution and Histogram plot for each feature

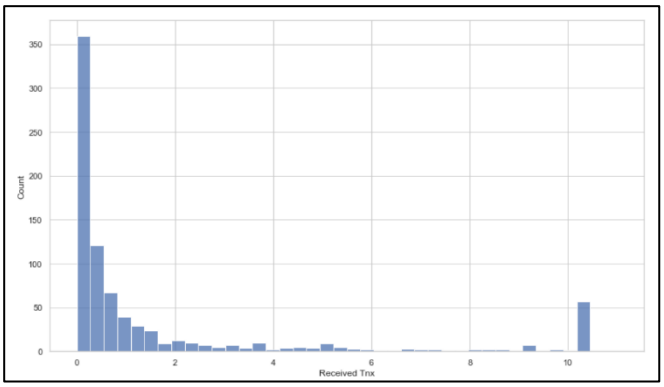

Figure 3 – feature distribution

```
df.isnull().sum()[df.isnull().sum()>0]
```

```
Total ERC20 tnxs                        829
ERC20 total Ether received              829
ERC20 total ether sent                  829
ERC20 total Ether sent contract         829
ERC20 uniq sent addr                    829
ERC20 uniq rec addr                     829
ERC20 uniq sent addr.1                  829
ERC20 uniq rec contract addr            829
ERC20 avg time between sent tnx         829
ERC20 avg time between rec tnx          829
ERC20 avg time between rec 2 tnx        829
ERC20 avg time between contract tnx     829
ERC20 min val rec                       829
ERC20 max val rec                       829
ERC20 avg val rec                       829
ERC20 min val sent                      829
ERC20 max val sent                      829
ERC20 avg val sent                      829
ERC20 min val sent contract             829
ERC20 max val sent contract             829
ERC20 avg val sent contract             829
ERC20 uniq sent token name              829
ERC20 uniq rec token name               829
ERC20 most sent token type              841
ERC20_most_rec_token_type               851
dtype: int64
```

Figure 4 – finding the null values

```
for i in df_copy.columns:
    if df_copy[i].isnull().sum() == 829:
        df_copy[i].replace({np.NaN:df_copy[i].mean()},inplace=True)
    else:
        pass
```

Figure 5 – mean imputation to the numeric

```
#Important information about each feature
df.describe()
```

	FLAG	Avg min between sent tnx	Avg min between received tnx	Time Diff between first and last (Mins)	Sent tnx	Received Tnx	Number of Created Contracts	Unique Received From Addresses	Unique Sent To Addresses	min value received	max va receiv
count	9841.000000	9841.000000	9841.000000	9.841000e+03	9841.000000	9841.000000	9841.000000	9841.000000	9841.000000	9841.000000	9841.000
mean	0.221421	5086.878721	8004.851184	2.183333e+05	115.931714	163.700945	3.729702	30.360939	25.840159	43.845153	523.152
std	0.415224	21486.549974	23081.714801	3.229379e+05	757.226361	940.836550	141.445583	298.621112	263.820410	325.929139	13008.821
min	0.000000	0.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.000000	0.000000	0.000000	3.169300e+02	1.000000	1.000000	0.000000	1.000000	1.000000	0.001000	1.000
50%	0.000000	17.340000	509.770000	4.663703e+04	3.000000	4.000000	0.000000	2.000000	2.000000	0.095856	6.000
75%	0.000000	565.470000	5480.390000	3.040710e+05	11.000000	27.000000	0.000000	5.000000	3.000000	2.000000	67.067
max	1.000000	430287.670000	482175.490000	1.954861e+06	10000.000000	10000.000000	9995.000000	9999.000000	9287.000000	10000.000000	800000.000

Figure 6 – finding the features with variance close to 0

```
#making a list of the features which is their variance is equal to 0
to_drop = list(df_copy.var()[df_copy.var() == 0].keys())
to_drop
```

```
['ERC20 avg time between sent tnx',
 'ERC20 avg time between rec tnx',
 'ERC20 avg time between rec 2 tnx',
 'ERC20 avg time between contract tnx',
 'ERC20 min val sent contract',
 'ERC20 max val sent contract',
 'ERC20 avg val sent contract']
```

Figure 6 – finding the features with variance close to 0

```
df_copy.drop(['min value sent to contract','max val sent to contract'
                    ,'avg value sent to contract'
                    ,'total ether sent contracts','ERC20 uniq sent addr.1', 'Address'],axis =1 , inplace = True)
```

Figure 6 – finding the features with variance close to 0

```
df_copy.corr()['FLAG'].sort_values(ascending= False)
```

FLAG	1.000000
ERC20 min val sent	0.020860
ERC20 avg val sent	0.020597
ERC20 max val sent	0.020593
ERC20 total ether sent	0.020365
ERC20 total Ether sent contract	0.011106
ERC20 min val rec	0.009166
ERC20 uniq sent token name	0.007806
min val sent	0.006603
ERC20 avg val rec	0.006470
total ether balance	-0.003236
ERC20 max val rec	-0.003557
ERC20 total Ether received	-0.003690
avg val received	-0.011871
Number of Created Contracts	-0.013741
ERC20 uniq rec addr	-0.014456
ERC20 uniq rec token name	-0.014830
total Ether sent	-0.015022
ERC20 uniq rec contract addr	-0.015225
total ether received	-0.016934
ERC20 uniq sent addr	-0.016939
max value received	-0.019286
Total ERC20 tnxs	-0.021171
min value received	-0.021555
max val sent	-0.022436
Avg min between sent tnx	-0.029905
Unique Received From Addresses	-0.032031
Unique Sent To Addresses	-0.045654
avg val sent	-0.063358
Sent tnx	-0.078130
Received Tnx	-0.079493
total transactions (including tnx to create contract	-0.100485
Avg min between received tnx	-0.118750
Time Diff between first and last (Mins)	-0.269853

Figure 7 – Correlation between the features to the label

```
corr = df_copy.corr()
corr_df = corr[corr>0.6].dropna(axis = 1 ,thresh = 2)
corr_df = corr_df[corr_df != 1]
to_drop = corr_df[corr_df>0.9]
to_drop.values[to_drop.values>0]
to_drop.unstack().sort_values(kind="quicksort")[to_drop.unstack()>0]
```

ERC20 total ether sent	ERC20 min val sent	0.999311
ERC20 min val sent	ERC20 total ether sent	0.999311
ERC20 total ether sent	ERC20 avg val sent	0.999566
ERC20 avg val sent	ERC20 total ether sent	0.999566
ERC20 uniq rec contract addr	ERC20 uniq rec token name	0.999641
ERC20 uniq rec token name	ERC20 uniq rec contract addr	0.999641
ERC20 total ether sent	ERC20 max val sent	0.999649
ERC20 max val sent	ERC20 total ether sent	0.999649
ERC20 min val sent	ERC20 max val sent	0.999729
ERC20 max val sent	ERC20 min val sent	0.999729
ERC20 min val sent	ERC20 avg val sent	0.999785
ERC20 avg val sent	ERC20 min val sent	0.999785
ERC20 max val sent	ERC20 avg val sent	0.999952
ERC20 avg val sent	ERC20 max val sent	0.999952
ERC20 total Ether received	ERC20 max val rec	0.999967
ERC20 max val rec	ERC20 total Ether received	0.999967

dtype: float64

Figure 8 – finding the features with correlation > 0.9

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range = (0,1))
scaler.fit(X_train)
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Figure 9 – Min-Max Normalization

```
from sklearn.decomposition import PCA
pca_model = PCA(n_components= 15)
X = pca_model.fit_transform(X)
```

Figure 10 – PCA Data Reduction

```
# finding more outliers and trying to improve the model
for i in df_copy.drop('FLAG',axis =1).columns:
    IQR = np.percentile(df_copy[i],75) -  np.percentile(df_copy[i],25)
    lower_limit = np.percentile(df_copy[i],25) - 1.5*IQR
    upper_limit = np.percentile(df_copy[i],75) + 1.5*IQR
    outliers_a = df_copy[i][df_copy[i] > upper_limit].shape \
    + df_copy[i][df_copy[i] < lower_limit].shape
    if outliers_a[0]/df_copy['Unique Sent To Addresses'].shape[0] <= 0.07:
        print(i)


outliers_a = df_copy['avg val sent'][df_copy['avg val sent'] > upper_limit].shape \
    + df_copy['avg val sent'][df_copy['avg val sent'] < lower_limit].shape
outliers_a



outliers_a = df_copy['avg val sent'][df_copy['avg val sent'] > upper_limit].index
df_droped_corr.drop(outliers_a,axis = 0)
```

```
avg val sent
```

Figure 11 – Finding outliers with IQR and less than 5% outliers.

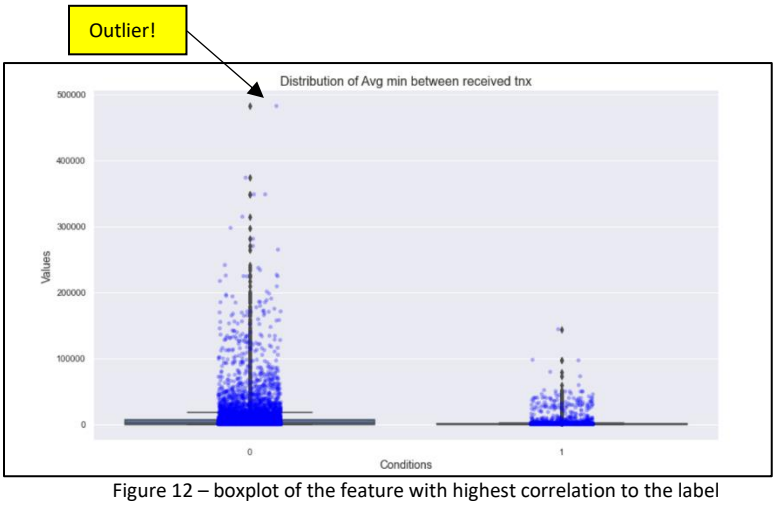

Figure 12 – boxplot of the feature with highest correlation to the label

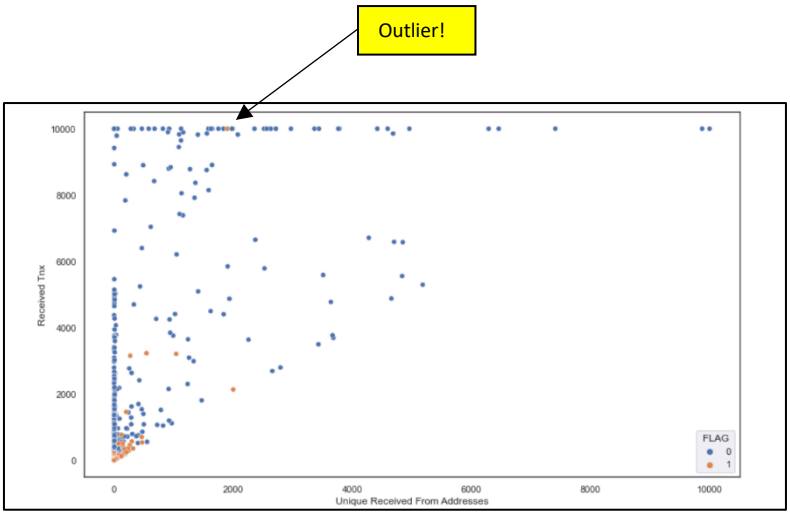

Figure 12 – scatter plot between 2 high correlated features (between 0.6 and 0.9).