Capstone Project

Prediction of Fraud Transaction on Ethereum

Chaw Hnin Nandar (301163035)

Centennial College

Business Analytics and Insights (Canadian Context)

Professor Mustafa Ahmed,

Professor David Parent

August 13, 2022

Executive Summary

0.1. Executive Introduction

This report includes a step-by-step capstone project for the last semester of Business Analytics and Insights (Canadian Context) program from Centennial College. In this project, the prediction model will be built to predict an upcoming fraud transaction among users in Ethereum. Ethereum is a digital platform with block chain technology which acts like a virtual bank for people with its own value called “Ether”. Although Ethereum was first introduced to the public in 2015, it becomes the second largest cryptocurrency platform at the moment following the bitcoin and it is also assumed to grow increasingly in the future.

With the increasing number of users in Ethereum, the security is the first thing to be considered protecting the users’ property and become the most reliable software in cryptocurrency market. Real users might lose their Ether by scammers, and it is important to make an alert to users from the Ethereum software which predicts suspicious incoming transactions to users’ account or blocks outgoing fraud transactions from hackers or scammers.

0.2. Executive Objective

The objective here is to be able to predict fraud transactions with the highest accuracy and so that Ethereum technicians can take actions on those users and raise the security level. In this way, the company will get more profits by having new users and transactions.

0.3. Executive Model Description

In this project, three predicted models such as logistic regression, random forest and xgboost models are built on the dataset found in Kaggle which includes information about transactions on Ethereum from past two years. After comparing these three models, xgboost is selected as the

best model to use for the business to predict fraud transactions as it has better accuracy, better performance and less errors.

The model also decided the important features for the company to be aware such as number of ERC20 token transactions received from unique addresses, total ether received for account address, time difference between the first and last transaction, average time between received transactions for account in minutes and maximum value in Ether ever received.

0.4. Executive Recommendation

It is recommended to collect recent transaction information and use in this model which will give better accuracy and performance. The model also needs to be trained with a lot more records so that it will be more familiar with fraud and non-fraud features. Last but not the least, the model should be modified frequently depending on how fluctuate the cryptocurrency market does and how many new features and facilities are added to the Ethereum platform.

Introduction

0.5. Background

Ethereum is a blockchain technology that serves transactions and trading of cryptocurrency globally with its own value called ether. Unlike other cryptocurrency like bitcoin, Ethereum is more than just sending and receiving of ether on its platform. The function of Ethereum includes sending transactions, trading ethers, interacting with its related applications, building smart contracts, creating fungible and non-fungible tokens, etc. Its system has blocks which create different nodes to store all data from users such as transactions, information, balances, etc. As the Ethereum's digital system circulates on the blocks, any user can create and find the Ethereum block under defined requirements to expand the network of the system. If someone can create a block on Ethereum and a transaction on that block is completed, that person will get transaction fees or block rewards. Therefore, Ethereum becomes popular with its unique facilities among cryptocurrency industry. Although bitcoin occupies the first place in digital currency market at present, Ethereum is considered to take the place of bitcoin in the future as the usage of Ethereum and its value grow increasingly time to time. Here are some parts of Ethereum which run the blockchain system. (CoinDesk, 2022)

- Ether is the main token of Ethereum which can be used to run functions on Ethereum platform.
- Gas is the amount of ether that can run the Ethereum services in which the unit of gas is called "gwei". The operation cost to complete one process of Ethereum is defined as multiplication of the gas price in gwei which is set by the user and the gas cost which is the quantity of gwei.

- A smart contract is such an Ethereum account which runs and works on the path of Ethereum called Ethereum Virtual Machine (EVM). Users can write code and rules by programming languages for smart contracts they want to apply and execute on EVM for the purpose of the use in decentralized autonomous organizations and financial system.
- Ethereum token includes both fungible with similar properties to bitcoin and other cryptocurrencies which can be traded with one another and non-fungible such as NFT which cannot be traded or exchanged.

Ethereum was built by the Canadian Russian programmer, and it was launched to public in 2015 with eight co-founders. At the moment, Ethereum is the second largest platform in cryptocurrency market after bitcoin. One ether equals to $1772.50 USD by 8th August 2022.

2.0. Problem Statement

As the number of Ethereum users are increasing rapidly these days, more than one million transactions are made on the platform every day. However, there are many fraud transactions which scam money from real users and from the Ethereum organization. For example, in 2016, $50 millions of ether were hacked by a scammer. Therefore, it is crucial for the Ethereum system to notice the fraudulent and avoid losing money from real users.

3.0. Objectives & Measurement

This project will predict the best model to detect fraud transaction on Ethereum and analyze the features which are the most important to take care of in order to avoid fraud transaction. So that, the system can detect and hold or block suspicious transactions, block that users' accounts and their activities on Ethereum which will have similar nature in analyzed features from the chosen model of this project.

In this way, the Ethereum platform will become more reliable and have more users around the world. Moreover, with the less amount of fraud transaction, the block chain platform can have less traffic and the organization can improve infrastructure of Ethereum offering more facilities to the users. The more the users Ethereum has, the higher the price of one Ether compared to other cryptocurrencies. Then, the users will gain more profit from it.

4.0. Assumptions & Limitation

The dataset that will be using in this project was updated two years ago which was in 2020. Therefore, the information and nature of transactions in this dataset might be different in some parts from nowadays' transactions as new blocks and policy of Ethereum are built and changed time to time.

Similarly, the value of Ether was different from today's value, and it might continue growing up to more than 300% at the end of 2022 as the growth of Ether value peaked to 300% last year. These changes may also have an impact on the prediction of fraud transactions in this project. However, generally, the features and characteristics of fraud transactions can be predicted and still can be used to aware of upcoming fraud transactions.

Data Sources

5.0. Data Set Introduction

In this project of predicting the best model for detection of fraud transaction in Ethereum, the real data set of transactions on Ethereum platform which was uploaded in Kaggle website is going to be used.

The direct link to access this dataset is provided below.

https://www.kaggle.com/datasets/vagifa/ethereum-frauddetection-dataset

It includes 51 variables with 9841 records which has both fraud and non-fraud transactions, however, the data is imbalanced as only 22% of this data is considered as fraud and the rests are real transactions. The target variable here is named as “FRAUD” in which it is classified into 1 as a fraud and 0 as a non-fraud transaction.

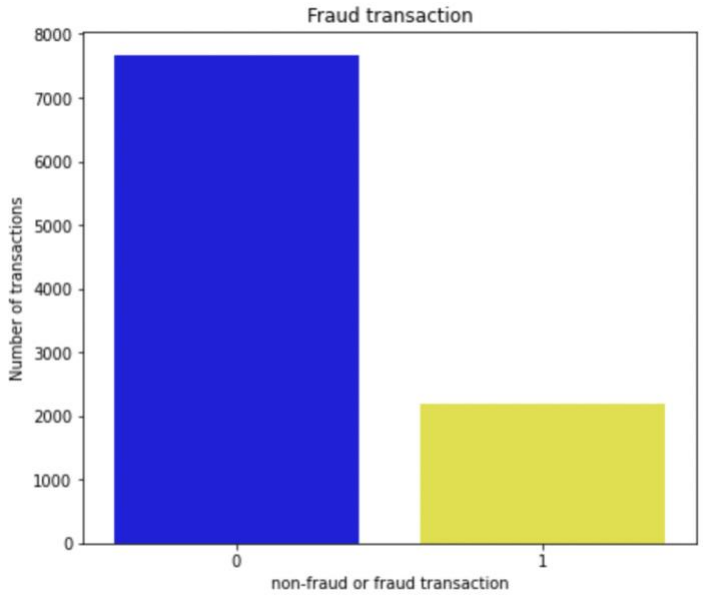


Figure 1:The proportion of fraud and non-fraud transaction in original data set

6.0. Exclusions

Not all the records and variables are necessary in prediction of fraud transaction. It is obvious that the fraud transaction does not depend on some variables such as index and address of users’ account. There are also some records which do not have active transactions on their accounts. Such of these records and variables can be excluded from original data set.

6.1. Initial Data Preparation

Among different programming languages, python will be used to predict the fraud transaction in this capstone project. Python can be accessed from different applications, software and open

sources such as jupyter notebook, anaconda, visual studio, google colab, and so on. In this project, models will be built using python language in google colab.

6.2. Importing data and installing libraries in python

First, all necessary libraries and packages that are going to be used mostly for further process are installed. Pandas, numpy, sklearn, matplotlib are crucial libraries in choosing a right model, preprocessing and cleaning data, plotting figures, prediction of a fraud transaction and so on.

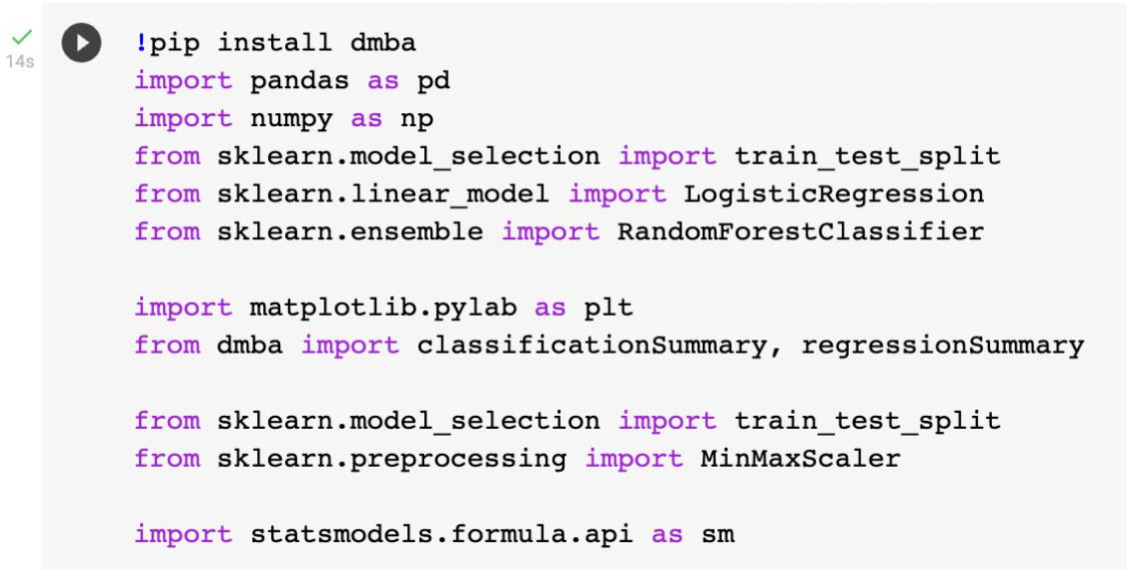


Figure 2: Installing libraries and packages

As a second step, the data set is imported to the python path on google colab to look through the nature and flow of the original data by using “pandas” library. The first five records of the data are shown in the code.

```
df = pd.read_csv('transaction_dataset.csv')
df.head()
```

	Unnamed: 0	Index	Address	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
0	0	1	0x00009277775ac7d0d59eaad8fee3d10ac6c805e8	0	844.26	1093.71	704785.63	721	89	0
1	1	2	0x0002b44ddb1476db43c868bd494422ee4c136fed	0	12709.07	2958.44	1218216.73	94	8	0
2	2	3	0x0002bda54cb772d040f779e88eb453cac0daa244	0	246194.54	2434.02	516729.30	2	10	0
3	3	4	0x00038e6ba2fd5c09aedb96697c8d7b8fa6632e5e	0	10219.60	15785.09	397555.90	25	9	0
4	4	5	0x00062d1dd1afb6fb02540ddad9cdebfe568e0d89	0	36.61	10707.77	382472.42	4598	20	1

5 rows × 51 columns

Figure 3: Importing data into python

After that, the shape of the whole data set and the distribution of each transaction to fraud or non-fraud are determined.

```
[9] df.shape

(9841, 51)

print(df['FLAG'].value_counts())

0    7662
1    2179
Name: FLAG, dtype: int64
```

Figure 4: Showing numbers of rows and columns, fraud and non-fraud transactions

6.3. Filling the spaces in column names with hyphen using replace function

The spaces in column names are replaced with hyphen by using a replace function in python which will help the author to use the columns in further processes.

```
# Making the names of variables to be used easily and clear in future work.
df1.columns = [c.replace(' ', '_') for c in df1.columns]
df1.columns
```

```
Index(['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_value_received_',
       'avg_val_received', 'min_val_sent', 'max_val_sent', 'avg_val_sent',
       'min_value_sent_to_contract', 'max_val_sent_to_contract',
       'avg_value_sent_to_contract',
       'total_transactions_(including_tnx_to_create_contract',
       'total_Ether_sent', 'total_ether_received',
       'total_ether_sent_contracts', 'total_ether_balance',
       '_Total_ERC20_tnxs', '_ERC20_total_Ether_received',
       '_ERC20_total_ether_sent', '_ERC20_total_Ether_sent_contract',
       '_ERC20_uniq_sent_addr', '_ERC20_uniq_rec_addr',
       '_ERC20_uniq_sent_addr.1', '_ERC20_uniq_rec_contract_addr',
       '_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_rec',
       '_ERC20_max_val_rec', '_ERC20_avg_val_rec', '_ERC20_min_val_sent',
       '_ERC20_max_val_sent', '_ERC20_avg_val_sent',
       '_ERC20_min_val_sent_contract', '_ERC20_max_val_sent_contract',
       '_ERC20_avg_val_sent_contract', '_ERC20_uniq_sent_token_name',
       '_ERC20_uniq_rec_token_name', '_ERC20_most_sent_token_type',
       '_ERC20_most_rec_token_type'],
      dtype='object')
```

Figure 5: Column names with hyphens

7.0. Data Dictionary

The data dictionary for the data set to be used in this project has already been described on the Kaggle website by the writer. (ALIYEV, 2020)

Variables	Data Type	Description of the variables
Index	Int64	The index number of a row
Address	Object	The address of the Ethereum account
FLAG	Int64	Whether the transaction is fraud or not
Avg min between sent tnx	Float64	Average time between sent transactions for account in minutes

Avg min between received tnx	Float64	Average time between received transactions for account in minutes
Time Diff between first and last (Mins)	Float64	Time difference between the first and last transaction
Sent tnx	Int64	Total number of sent normal transactions
Received tnx	Int64	Total number of received normal transactions
Number of Created Contracts	Int64	Total Number of created contract transactions
Unique Received from Addresses	Int64	Total Unique addresses from which account received transactions
Unique Sent to Addresses	Int64	Total Unique addresses from which account sent transactions
Min Value Received	Float64	Minimum value in Ether ever received
Max Value Received	Float64	Maximum value in Ether ever received
Avg Value Received	Float64	Average value in Ether ever received
Min Val Sent	Float64	Minimum value of Ether ever sent
Max Val Sent	Float64	Maximum value of Ether ever sent
Avg Val Sent	Float64	Average value of Ether ever sent
Min Value Sent to Contract	Float64	Minimum value of Ether sent to a contract
Max Value Sent to Contract	Float64	Maximum value of Ether sent to a contract
Avg Value Sent to Contract	Float64	Average value of Ether sent to contracts
Total Transactions (Including tnx to Create Contract)	Int64	Total number of transactions
total Ether sent	Float64	Total Ether sent for account address

total ether received	Float64	Total Ether received for account address
total ether sent contracts	Float64	Total Ether sent to Contract addresses
total ether balance	Float64	Total Ether Balance following enacted transactions
Total ERC20 tnxs	Float64	Total number of ERC20 token transfer transactions
ERC20 total Ether received	Float64	Total ERC20 token received transactions in Ether
ERC20 total ether sent	Float64	Total ERC20token sent transactions in Ether
ERC20 total Ether sent contract	Float64	Total ERC20 token transfer to other contracts in Ether
ERC20 uniq sent addr	Float64	Number of ERC20 token transactions sent to Unique account addresses
ERC20 uniq rec addr	Float64	Number of ERC20 token transactions received from Unique addresses
ERC20 uniq sent addr.1	Float64	Number of ERC20 token transactions sent to Unique account addresses
ERC20 uniq rec contract addr	Float64	Number of ERC20token transactions received from Unique contract addresses
ERC20 avg time between sent tnx	Float64	Average time between ERC20 token sent transactions in minutes
ERC20 avg time between rec tnx	Float64	Average time between ERC20 token received transactions in minutes
ERC20 avg time between rec 2 tnx	Float64	Average time between ERC20 token received transactions in minutes

ERC20 avg time between contract tnx	Float64	Average time ERC20 token between sent token transactions
ERC20 min val rec	Float64	Minimum value in Ether received from ERC20 token transactions for account
ERC20 max val rec	Float64	Maximum value in Ether received from ERC20 token transactions for account
ERC20 avg val rec	Float64	Average value in Ether received from ERC20 token transactions for account
ERC20 min val sent	Float64	Minimum value in Ether sent from ERC20 token transactions for account
ERC20 max val sent	Float64	Maximum value in Ether sent from ERC20 token transactions for account
ERC20 avg val sent	Float64	Average value in Ether sent from ERC20 token transactions for account
ERC20 min val sent contract	Float64	Minimum value in Ether sent from ERC20 token transactions for contract
ERC20 max val sent contract	Float64	Maximum value in Ether sent from ERC20 token transactions for contract
ERC20 avg val sent contract	Float64	Average value in Ether sent from ERC20 token transactions for contract
ERC20 uniq sent token name	Float64	Number of Unique ERC20 tokens transferred
ERC20 uniq rec token name	Float64	Number of Unique ERC20 tokens received
ERC20 most sent token type	object	Most sent token for account via ERC20 transaction

ERC20_most_rec_token_type	object	Most received token for account via ERC20 transactions

Data Exploration

8.0. Data Exploration Techniques

8.1. Removing unnecessary columns

It is obvious that the three columns which are unnamed, index and address of user account do not help anywhere in predicting the target variable “FLAG” as the rows show no meaningful records. Therefore, these columns are removed by using drop function.

```
df1 = df.drop(columns=['Unnamed: 0','Index','Address'])
df1.head(5)
```

	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	...	ERC20 min val sent	ERC20 max val sent	ERC20 val s
0	0	844.26	1093.71	704785.63	721	89	0	40	118	0.000000	...	0.000000	1.683100e+07	271779.920
1	0	12709.07	2958.44	1218216.73	94	8	0	5	14	0.000000	...	2.260809	2.260809e+00	2.260
2	0	246194.54	2434.02	516729.30	2	10	0	10	2	0.113119	...	0.000000	0.000000e+00	0.000
3	0	10219.60	15785.09	397555.90	25	9	0	7	13	0.000000	...	100.000000	9.029231e+03	3804.076
4	0	36.61	10707.77	382472.42	4598	20	1	7	19	0.000000	...	0.000000	4.500000e+04	13726.659

5 rows × 48 columns

Figure 6: Dropping unnecessary columns

8.2. Descriptive statistics using describe function

The result shows the statistics of the data including mean, standard deviation, minimum, maximum, count, and percentile of each variable. Here, outliers can be seen in variables.

```
df1.describe(include='all')
```

	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	..
count	9841.000000	9841.000000	9841.000000	9.841000e+03	9841.000000	9841.000000	9841.000000	9841.000000	9841.000000	9841.000000	..
unique	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	..
top	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	..
freq	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	..
mean	0.221421	5086.878721	8004.851184	2.183333e+05	115.931714	163.700945	3.729702	30.360939	25.840159	43.845153	..
std	0.415224	21486.549974	23081.714801	3.229379e+05	757.226361	940.836550	141.445583	298.621112	263.820410	325.929139	..
min	0.000000	0.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	..
25%	0.000000	0.000000	0.000000	3.169300e+02	1.000000	1.000000	0.000000	1.000000	1.000000	0.001000	..
50%	0.000000	17.340000	509.770000	4.663703e+04	3.000000	4.000000	0.000000	2.000000	2.000000	0.095856	..
75%	0.000000	565.470000	5480.390000	3.040710e+05	11.000000	27.000000	0.000000	5.000000	3.000000	2.000000	..
max	1.000000	430287.670000	482175.490000	1.954861e+06	10000.000000	10000.000000	9995.000000	9999.000000	9287.000000	10000.000000	..

11 rows × 48 columns

Figure 7: Descriptive statistics of the data set

8.3. Changing the data type and dropping duplicate rows

```
[9] df1._ERC20_most_sent_token_type = df1._ERC20_most_sent_token_type.astype('category')
    df1._ERC20_most_rec_token_type = df1._ERC20_most_rec_token_type.astype('category')
```

```
df1.duplicated()
```

```
0       False
1       False
2       False
3       False
4       False
        ...
9836    False
9837     True
9838    False
9839    False
9840    False
Length: 9841, dtype: bool
```

```
df1.drop_duplicates()
```

Figure 8: Data type changing and dropping duplicates

8.4. Finding correlation between variables

Correlation plays an important role in building a model which can make a model overfit or underfit.

Python code:

```
fig, ax = plt.subplots(figsize=(16,12))

corr=df1.corr()

import seaborn as sns

sns.heatmap(corr, annot=True, fmt='.1f', cmap='RdYlGn')
```

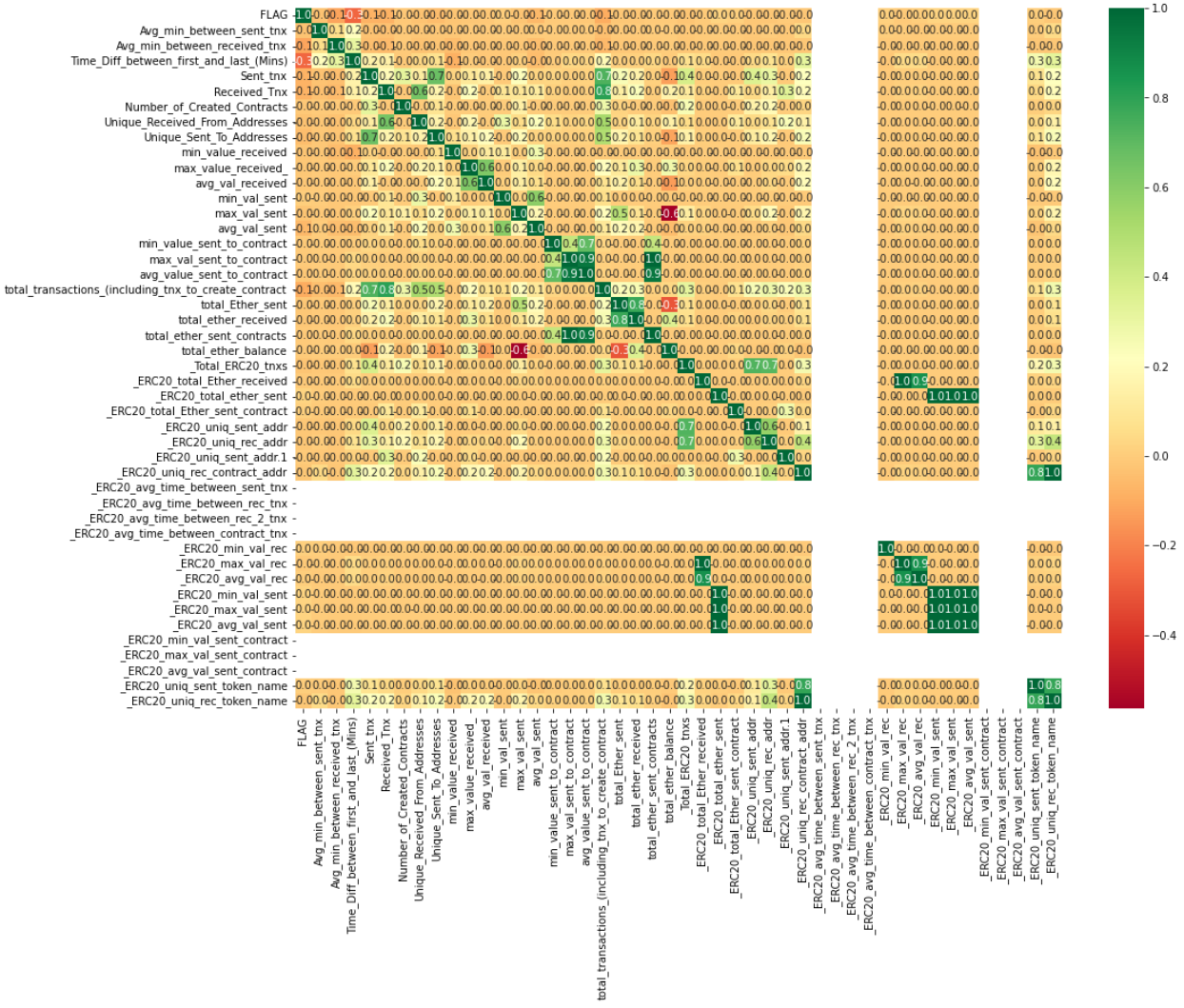


Figure 9: Heatmap showing the correlation between each variable

9.0. Data Cleansing

9.1. Dropping unnecessary variables

By looking at the result of the correlation between variables in heatmap, some variables that are not effective in building a model and analysis of the fraud transaction can be found and these variables will be dropped. The two categorical variables are not shown in the heatmap and so they are considered to be useless for predicting the fraud. Moreover, in order to avoid multicollinearity between variables, those having correlation value higher than 0.7 are dropped.

Python Code: dropping columns without any action or with zero for all records

```
df1=df1.drop(columns=['_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','_ERC20_most_sent_token_type','_ERC20_most_rec_token_type'])
```

Python Code: dropping columns with highly correlated values (>0.7)

```
df1=df1.drop(columns=['total_transactions_(including_tnx_to_create_contract','avg_value_sent_to_contract','_Total_ERC20_tnxs','_ERC20_avg_val_sent','Unique_Sent_To_Addresses','_ERC20_avg_val_rec','_ERC20_uniq_sent_token_name','_ERC20_uniq_rec_contract_addr','_ERC20_uniq_sent_addr','min_value_sent_to_contract','max_val_sent_to_contract','_ERC20_max_val_sent','_ERC20_total_Ether_received','_ERC20_total_ether_sent'])
```

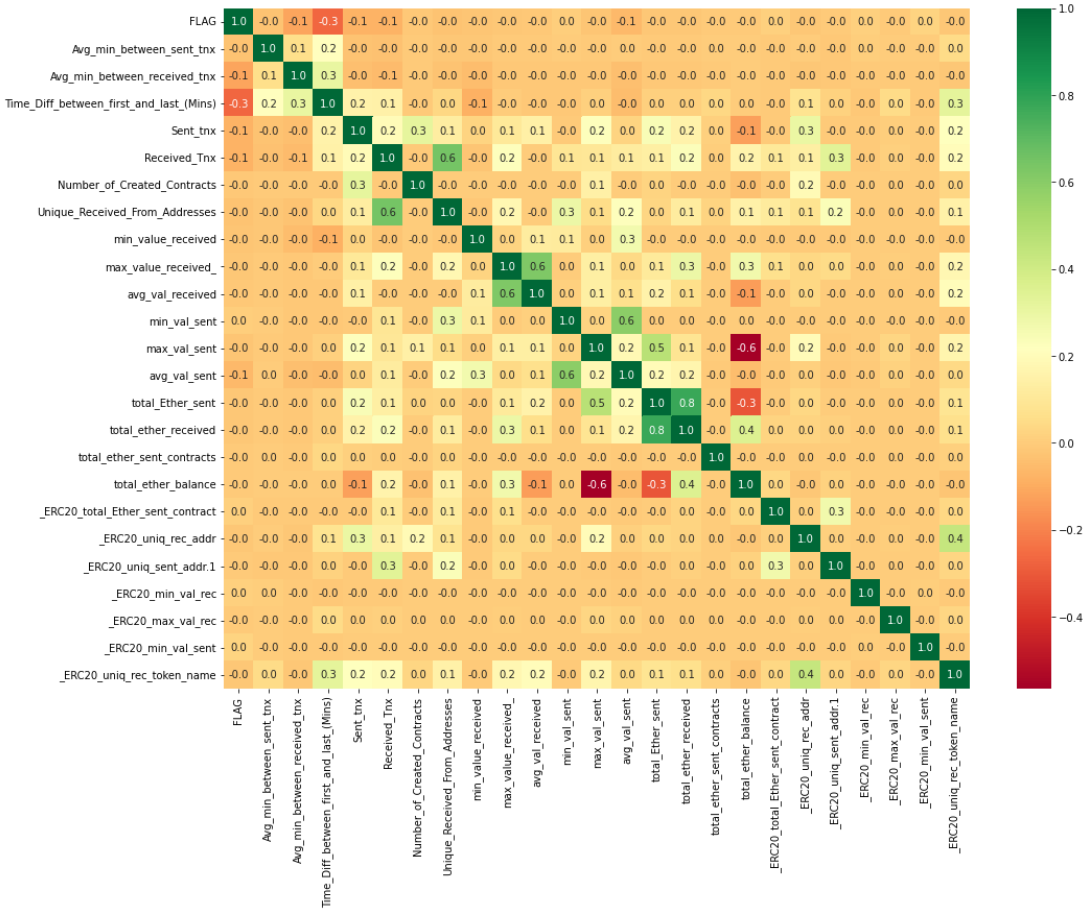


Figure 10: Heatmap after dropping highly correlated and unnecessary variables

9.2. Outliers

The selected variables have outliers, and they are set into ranges by removing outliers for each variable as shown below.

Python code:

```
Upper Limit = df1['column name'].mean() + 3*df1['column name'].std()

Lower limit = df1['column name'].mean() - 3*df1['column name'].std()

df1['column name'] = np.where(df1['column name'] > UL1,UL1,
                              np.where(df1['column name'] <LL1,LL1,df1['column name']))
```

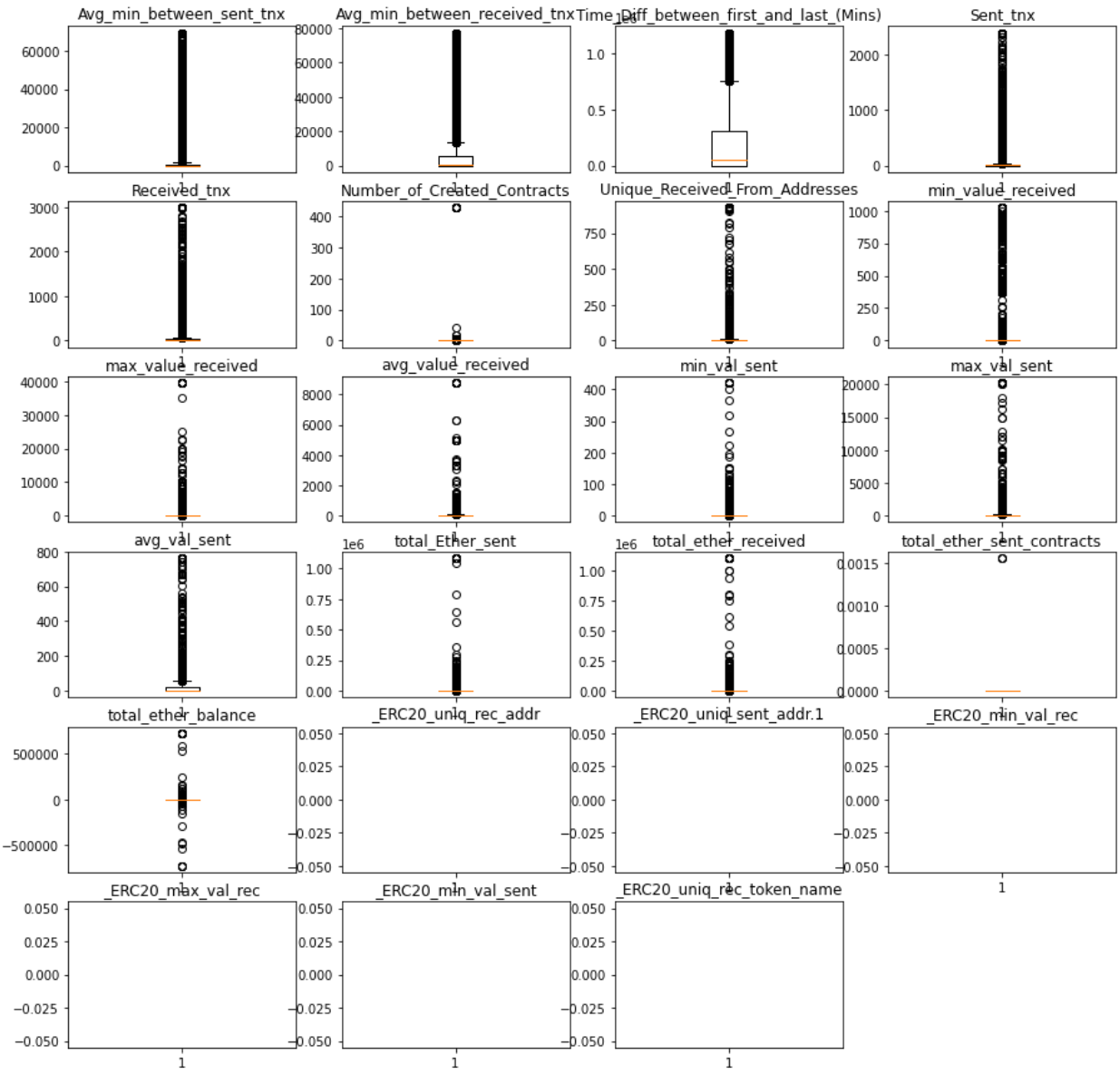


Figure 11: Distribution of selected variables with outliers

9.3. Missing Values

In real data sets, there might be missing values in some variables and here are the total number of missing values in each variable.

```
df1.isnull().sum()
```

```
FLAG                                         0
Avg_min_between_sent_tnx                     0
Avg_min_between_received_tnx                 0
Time_Diff_between_first_and_last_(Mins)      0
Sent_tnx                                     0
Received_Tnx                                 0
Number_of_Created_Contracts                  0
Unique_Received_From_Addresses               0
min_value_received                           0
max_value_received_                          0
avg_val_received                             0
min_val_sent                                 0
max_val_sent                                 0
avg_val_sent                                 0
total_Ether_sent                             0
total_ether_received                         0
total_ether_sent_contracts                   0
total_ether_balance                          0
_ERC20_total_Ether_sent_contract           829
_ERC20_uniq_rec_addr                       829
_ERC20_uniq_sent_addr.1                    829
_ERC20_min_val_rec                         829
_ERC20_max_val_rec                         829
_ERC20_min_val_sent                        829
_ERC20_uniq_rec_token_name                 829
dtype: int64
```

Figure 12: Counting Missing values in each variable

10.0. Summary

In conclusion for data exploration, related data information is found as follow:

- The data set contains 51 variables and 9841 records which includes both fraud and non-fraud transactions.
- The data is imbalanced that only 22% of the whole data is fraud which means this data has to be rebalanced in further processes to avoid overfitting or underfitting the model.
- Some variables which do not have impact on predicting the target variables and which are highly correlated are dropped.

- Missing values and outliers are found in the chosen variables, and these are going to be imputed and removed in data preparation section.
- Duplicated rows are found, and they are dropped.

Data Preparation and Feature Engineering

11.0. Data Preparation Needs

11.1 Imputation

By looking at the result using the “isnull” function, the data set has missing values in some columns which need to be imputed, otherwise, a model cannot be built with missing values. Missing value problem can be solved by replacing the missing value with zero or filling the missing value with median or mean of the variable. Before deciding which method to be used, the distributions of those variables with missing values to the target variable are figured out by using the code shown below for each variable.

```
x1 = list(df1[df1['_ERC20_total_Ether_sent_contract'] == 0]['FLAG'])
x2 = list(df1[df1['_ERC20_total_Ether_sent_contract'] > 0]['FLAG'])

colors = ['#E69F00', '#56B4E9']
names = ['Zero', 'not zero']

plt.figure(figsize=(10,8))
plt.subplot(2,1,1)

plt.hist([x1, x2],bins = int(180/15), color = colors, label=names)

plt.legend()
plt.xlabel('Fraud or Real')
plt.ylabel('_ERC20_total_Ether_sent_contract')
plt.title('Histogram')

x3 = list(df1[df1['_ERC20_total_Ether_sent_contract'].isnull()]['FLAG'])
x4 = list(df1[df1['_ERC20_total_Ether_sent_contract']> 0]['FLAG'])

colors = ['#E69F00', '#56B4E9']
names = ['missing', 'not missing']

plt.figure(figsize=(10,8))
plt.subplot(2,1,2)

plt.hist([x3, x4],color = colors, label=names)

plt.legend()
plt.xlabel('Fraud or Real')
plt.ylabel('_ERC20_total_Ether_sent_contract')
plt.title('Histogram')
```

Figure 13: Python code for distribution of each variable for the target variable "FLAG"

Histograms

_ERC20_total_Ether_sent_contract

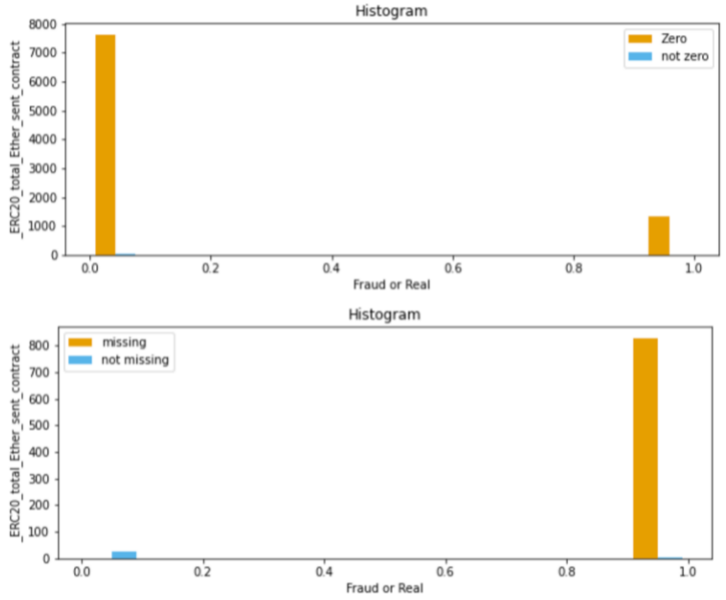


Figure 14: Nature of ERC20 total ether sent contract

_ERC20_uniq_sent_rec_addr

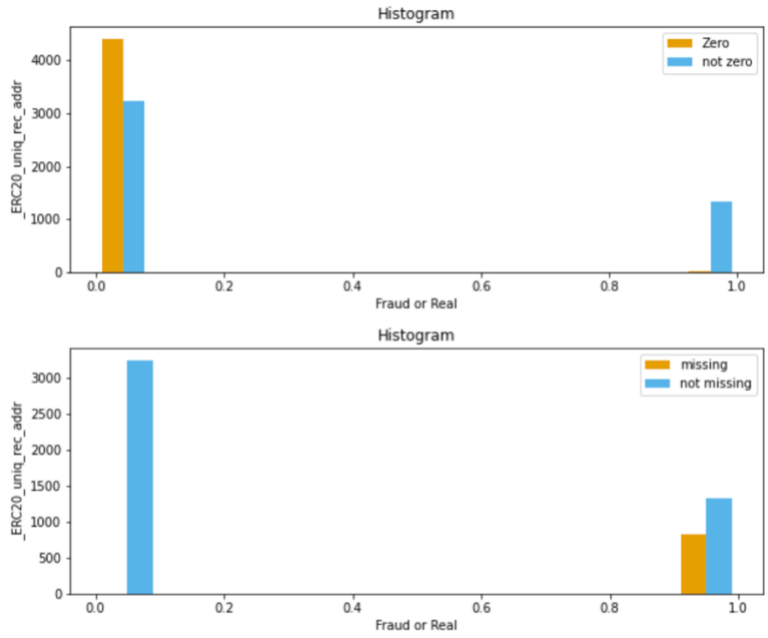


Figure 15: Nature of ERC20 uniq sent rec addr

_ERC20_uniq_sent_addr.1

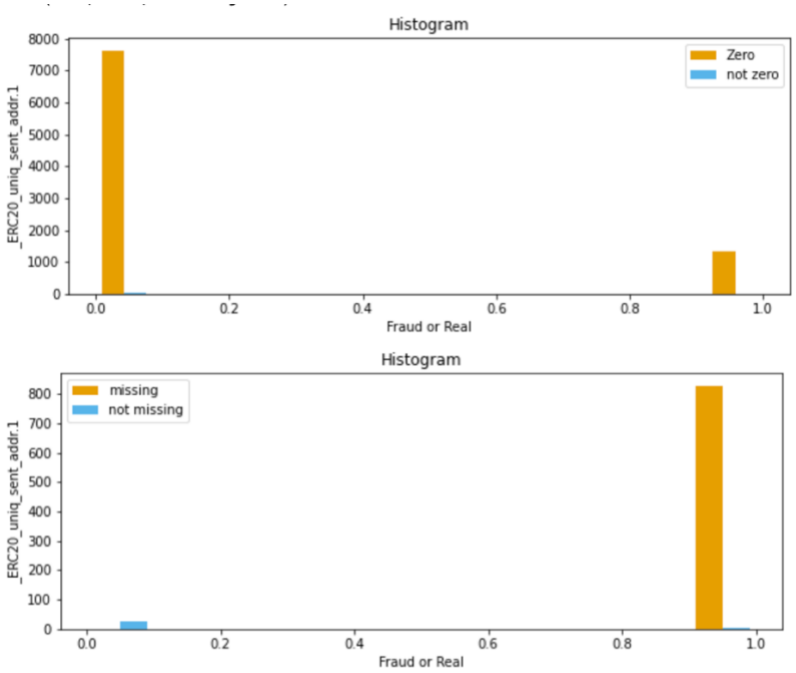


Figure 16: Nature of ERC20 uniq sent addr.1

_ERC20_min_val_rec

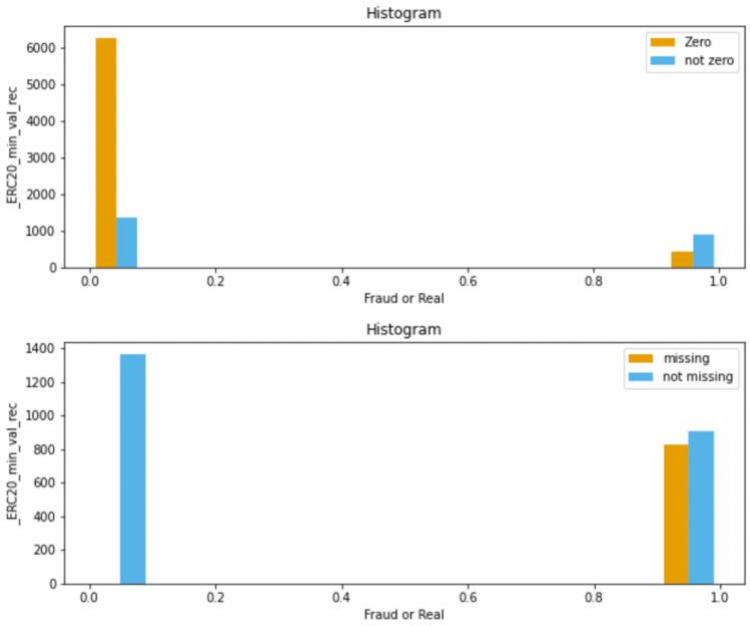


Figure 17: ERC20 min val rec

_ERC20_max_val_rec

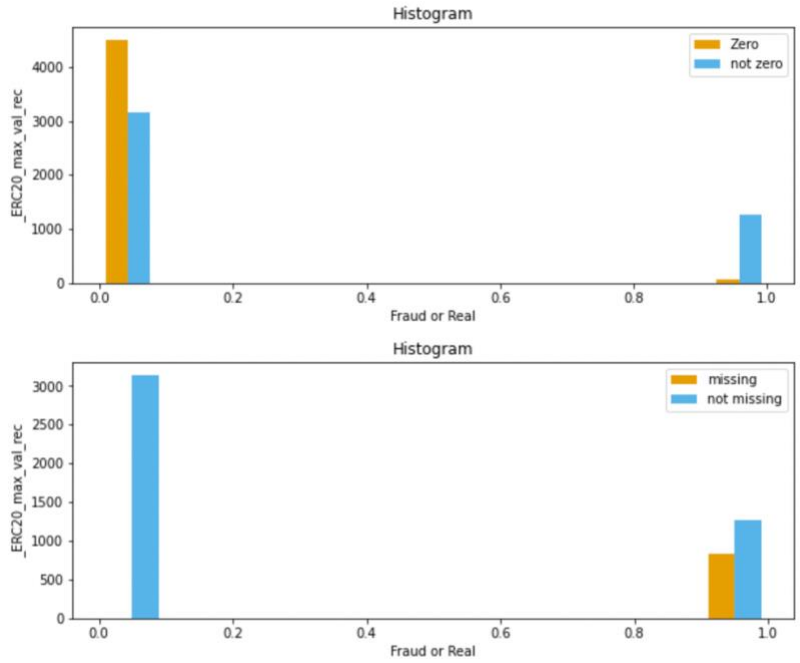


Figure 18: Nature of ERC20 max val rec

_ERC20_min_val_sent

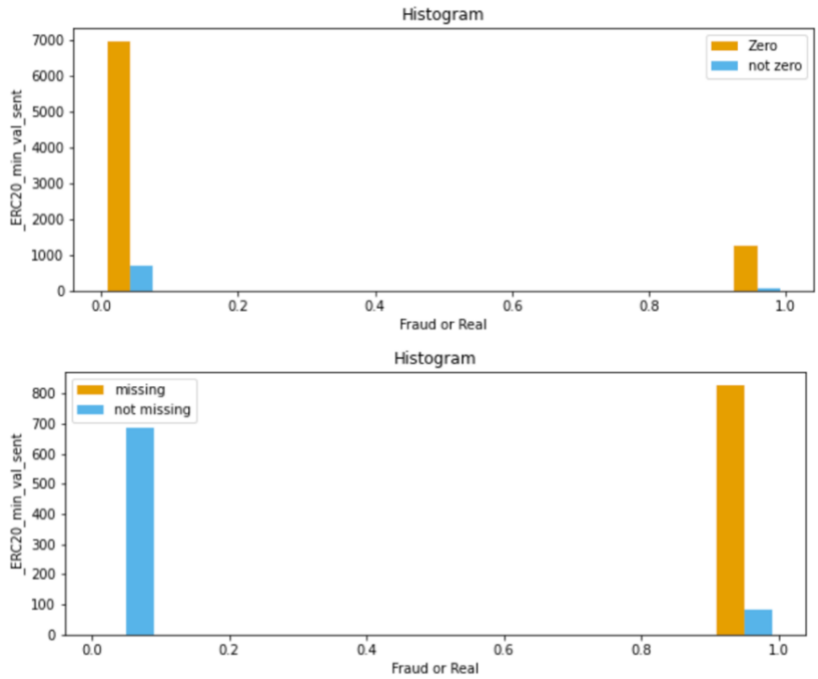


Figure 19: Nature of ERC20 min val sent

_ERC20_uniq_rec_token_name

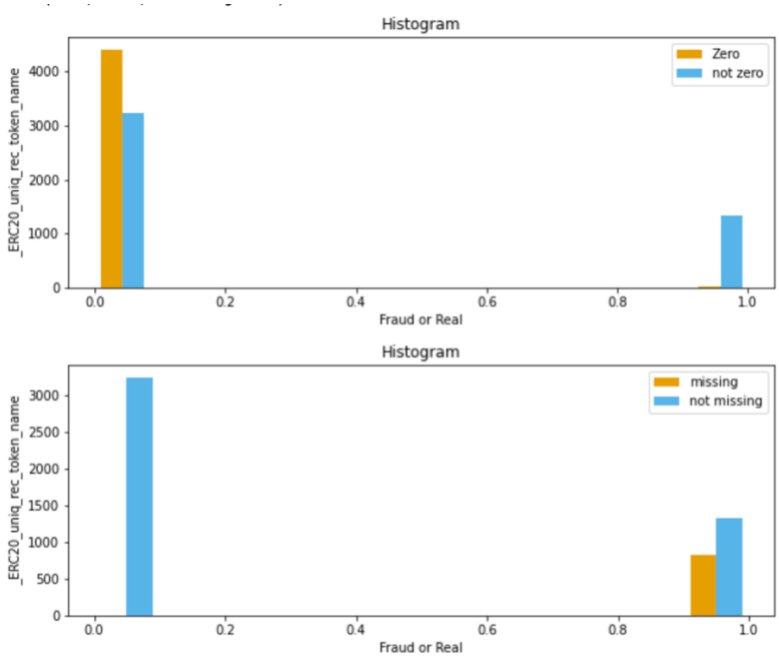


Figure 20: ERC20 uniq rec token name

After comparing the histograms of distribution of zero, non-zero, and null, the sent and received conditions have different nature.

All missing values and most of non-zero variables are shown as fraud in histogram. By looking at the histograms of sent and receive variables, we decided to impute the missing values with mean or median value of all data. As the variables have outliers, median value is chosen to impute missing values to fill with certain numbers and to handle outliers in the variables.

```
[42] #df.fillna(0)
     df1.fillna(df1.median(), inplace=True)
```

```
df1.median()

FLAG                                          0.000000
Avg_min_between_sent_tnx                     17.340000
Avg_min_between_received_tnx                509.770000
Time_Diff_between_first_and_last_(Mins)   46637.030000
Sent_tnx                                      3.000000
Received_Tnx                                  4.000000
Number_of_Created_Contracts                   0.000000
Unique_Received_From_Addresses                2.000000
min_value_received                            0.095856
max_value_received_                           6.000000
avg_val_received                              1.729730
min_val_sent                                  0.049126
max_val_sent                                  4.999380
avg_val_sent                                  1.606000
total_Ether_sent                             12.486800
total_ether_received                         30.529634
total_ether_sent_contracts                    0.000000
total_ether_balance                           0.001722
_ERC20_total_Ether_sent_contract              0.000000
_ERC20_uniq_rec_addr                          1.000000
_ERC20_uniq_sent_addr.1                       0.000000
_ERC20_min_val_rec                            0.000000
_ERC20_max_val_rec                            0.000000
_ERC20_min_val_sent                           0.000000
_ERC20_uniq_rec_token_name                    1.000000
dtype: float64
```

Figure 21: Imputation missing values with Median

11.2. Defining target and independent variables

The target variable is “FLAG” which contains 0 as non-fraud and 1 as fraud transaction. X is the list of input variables which affects the decision of outcome y.

```
[ ] X = df1.drop(columns='FLAG')
    y = df1['FLAG']
```

Figure 22: Setting input and outcome variables

11.2. Train and test data

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 5)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(7380, 24) (7380,)
(2461, 24) (2461,)
```

Figure 23: Training the dataset

12.0. Standardization and scaling the train and test data

Normalizing and balancing the scale of train and test data before building a model are necessary in this project.

```
[ ]  from sklearn import preprocessing

     std_scale = preprocessing.MinMaxScaler().fit(X_train)
     X_train_std = std_scale.fit_transform(X_train)
     X_test_std  = std_scale.transform(X_test)
```

Figure 24: Standardization and scaling the data

12.1. Handling Imbalanced data using SMOTE function

As the original data is imbalanced, SMOTE built-in function is used for upsampling the train data.

```
from collections import Counter
from imblearn.over_sampling import SMOTE
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

SMOTE = SMOTE()

# fit and apply the transform
X_train_SMOTE, y_train_SMOTE = SMOTE.fit_resample(X_train_std, y_train)

# summarize class distribution
print("After oversampling: ",Counter(y_train_SMOTE))
```

```
After oversampling:  Counter({0: 5746, 1: 5746})
```

Figure 25: Balancing the train data set

The shape of the training data is increased from 7380 to 11492 as upsampling is applied using SMOTE. Therefore, the number of records for non-fraud and fraud transactions are balanced after upsampling.

```
[ ] print(f'Shape of the training before SMOTE: {X_train_std.shape, y_train.shape}')

    print(f'Shape of the training after SMOTE: {X_train_SMOTE.shape, y_train_SMOTE.shape}')

    Shape of the training before SMOTE: ((7380, 24), (7380,))
    Shape of the training after SMOTE: ((11492, 24), (11492,))
```

```
    v = sum(y_train == 0)
    j = sum(y_train == 1)
    print(f'non_fraud before oversampling: {v} \n Fauds before oversampling: {j}')

    R = sum(y_train_SMOTE == 0)
    M = sum(y_train_SMOTE == 1)
    print(f'non_fraud after oversampling: {R} \n Fauds after oversampling: {M}')

    non_fraud before oversampling: 5746
     Fauds before oversampling: 1634
    non_fraud after oversampling: 5746
     Fauds after oversampling: 5746
```

Figure 26: Result after using SMOTE for balancing the train data

Model Exploration

13.0. Model Approach Introduction

In this project, three models such as logistic regression, random forest, and XG boost are going to be built and results are going to be compared to each other.

14.0. Logistic Regression

Logistic Regression is used to predict the probability of the outcome in binary such as 0 or 1 by using the input independent variables. In this project, 0 is non-fraud and 1 is fraud transaction of an Ethereum user.

14.1. Building and fitting the logistic regression

```
[49] logit_reg = LogisticRegression(solver = "lbfgs", C=1e42, random_state=100)
     logit_reg.fit(X_train_SMOTE,y_train_SMOTE)

     y_pred=logit_reg.predict(X_test_std)
```

Figure 27: Applying logistic regression model

14.2. Checking the accuracy of test data and f1 score of the model

```
from sklearn.metrics import  confusion_matrix
cm_new = confusion_matrix(y_test, y_pred)

print("Confusion matrix:\n")
print(cm_new)
tn = cm_new[0][0]
fp = cm_new[0][1]
fn = cm_new[1][0]
tp = cm_new[1][1]
print("\nTotal number of true positives", tp)
print("Total number of false negatives",fn)
print("Total number of false positives",fp)
print("Total number of true negatives",tn)

acc=float(tp+tn)/(tp+tn+fp+fn)
print('\nClassifier Accuracy: %.2f%%' % (acc * 100))

tpr = float(tp)/(tp+fn)
print('True Positive Rate (TPR/Recall/Sensitivity): %.2f%%' % (tpr * 100))

specificity = float (tn)/(tn+fp)
print ("True Negative Rate (TNR/Specificity/selectivity):%.2f%%" % (specificity*100))

fpr = float(fp)/(fp+tn)
print("False Positive Rate (FPR): %.2f%%" % (fpr * 100))

fnr = fn/ (fn+ tp)
print("False Negative Rate (FNR): %.2f%%" % (fnr*100))

precision=float(tp)/(tp+fp)
print("Precision/Positive Predictive value:%.2f%%" %(precision*100))

fScore = 2*((precision*tpr)/(precision+tpr))
print("F1-Score: %.2f%%" %(fScore*100))
```

Figure 28: Calculating accuracy and F1 score of Logistic Regression

```
Confusion matrix:

[[1227  689]
 [  83  462]]

Total number of true positives 462
Total number of false negatives 83
Total number of false positives 689
Total number of true negatives 1227

Classifier Accuracy: 68.63%
True Positive Rate (TPR/Recall/Sensitivity): 84.77%
True Negative Rate (TNR/Specificity/selectivity):64.04%
False Positive Rate (FPR): 35.96%
False Negative Rate (FNR): 15.23%
Precision/Positive Predictive value:40.14%
F1-Score: 54.48%
```

Figure 29: Result of accuracy and F1 score

14.3. Accuracy of training data

```
#accuracy for training data by logistic regression
classificationSummary(y_train_SMOTE, logit_reg.predict(X_train_SMOTE))
```

```
Confusion Matrix (Accuracy 0.7543)

       Prediction
Actual    0    1
     0 3687 2059
     1  765 4981
```

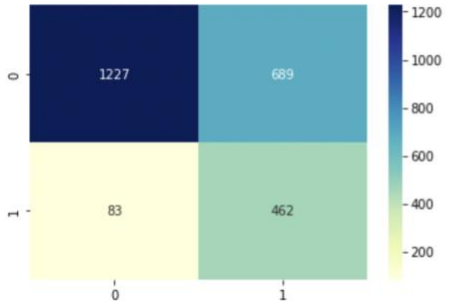
Figure 30: Accuracy of train data

14.4. Heatmap of logistic regression model

```
cls_names=[0,1] # name  of classes
fig, ax = plt.subplots()
ticks = np.arange(len(cls_names))
plt.xticks(ticks, cls_names)
plt.yticks(ticks, cls_names)
# create heatmap
sns.heatmap(pd.DataFrame(cm_new), annot=True, cmap="YlGnBu" ,fmt='g').figure
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix\n', fontsize = 16)
plt.ylabel('Actual label',fontsize = 16)
plt.xlabel('Predicted label',fontsize = 16)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: RuntimeWarning: More than 20 figures hav

([<matplotlib.axis.XTick at 0x7f31fcd78950>,
  <matplotlib.axis.XTick at 0x7f31fcd783d0>],
 [Text(0, 0, '0'), Text(0, 0, '1')])([<matplotlib.axis.YTick at 0x7f31fcd9ab90>,
  <matplotlib.axis.YTick at 0x7f31fcd9a3d0>],
 [Text(0, 0, '0'), Text(0, 0, '1')])
```




```
Text(0.5, 1.0, 'Confusion matrix\n')Text(33.0, 0.5, 'Actual label')Text(0.5, 257.44, 'Predicted label')
```

Figure 31: Heatmap showing performance of Logistic Regression

14.5. Root mean square error of logistic model

```
[57] from sklearn.metrics import mean_squared_error
     import math
     MSE = mean_squared_error(y_test, logit_reg_pred)
     RMSE = math.sqrt(MSE)
     print("Root Mean Square Error of logistic regression model:",RMSE)

     Root Mean Square Error of logistic regression model: 0.5600835834761451
```

Figure 32: RMSE by Logistic Regression

14.6. Accuracy and predicted result with probability

```
[ ] logit_reg_prob = logit_reg.predict_proba(X_test_std)

    logit_reg_pred = logit_reg.predict(X_test_std)
```

```
logit_result = pd.DataFrame({'actual' : y_test,
                             'p_0' : [p[0] for p in logit_reg_prob],
                             'p_1' : [p[1] for p in logit_reg_prob],
                             'predicted': logit_reg_pred})
logit_result
```

	actual	p_0	p_1	predicted
5469	0	0.497235	5.027647e-01	1
4979	0	0.634134	3.658659e-01	0
3716	0	0.621129	3.788707e-01	0
7290	0	0.489031	5.109689e-01	1
3697	0	0.642213	3.577874e-01	0
...	...	...	...	...
3140	0	1.000000	2.969745e-75	0
1861	0	0.291876	7.081236e-01	1
1793	0	0.999541	4.585521e-04	0
6609	0	0.486067	5.139334e-01	1
7398	0	1.000000	1.402645e-10	0

2461 rows × 4 columns

Figure 33: Comparing actual and predicted data

14.7. Gain and Lift charts by logistic regression

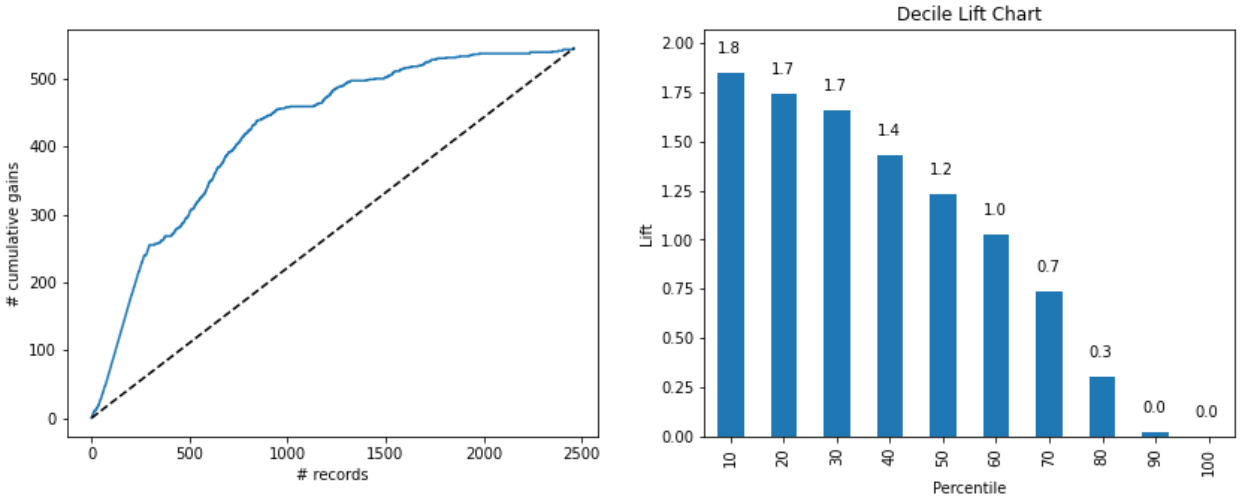


Figure 34: Gain and Lift Charts by Logistic Regression model

14.8. ROC-AUC score

The best ROC score is defined as 1 and the score by logistic regression model inclined from 0.74 to 0.85 after using regression model.

```
y_pred_proba = logit_reg.predict_proba(X_test_std)[::,1]
from sklearn.metrics import roc_auc_score
print("ROC AUC score before training data: ", roc_auc_score(y_test, y_pred))
print("ROC AUC score after training data: ", roc_auc_score(y_test, y_pred_proba))
```

```
ROC AUC score before training data:  0.7440515408630366
ROC AUC score after training data:  0.8553810499703127
```

Figure 35: AUC score by Logistic Regression

15.0. Random Forest Model

Random forest is a common type of machine learning model that is used for classification and prediction. It is easy to use, and it also shows the important features of the data related to the target variable.

15.1. Find the best parameter for the best fit of Random Forest model

To obtain the best outcome from random forest model, the function called “GridSearchCV” is imported from sklearn which defines the best parameter to use in building and fitting random forest model. (Shin, 2021)

```
[669] #Finding the best parameter for random forest

      from sklearn.model_selection import GridSearchCV
      rfc=RandomForestClassifier(random_state=42)

      param_grid = {
          'n_estimators': [100,200, 500],
          'max_features': ['auto', 'sqrt', 'log2'],
          'max_depth' : [4,5,6,7,8],
          'criterion' :['gini', 'entropy']
      }
```

```
[670] #CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
      #CV_rfc.fit(X_train_SMOTE, y_train_SMOTE)
```

```
[671] CV_rfc.best_params_

      {'criterion': 'entropy',
       'max_depth': 8,
       'max_features': 'auto',
       'n_estimators': 100}
```

Figure 36: Setting best parameter to use in Randon Forest

15.2. Apply the Random Forest Model

```
[672] #Fitting the model
      rf = RandomForestClassifier(criterion='entropy', max_depth=8, n_estimators=100,max_features='auto',random_state=42)
      rf.fit(X_train_SMOTE, y_train_SMOTE)

      RandomForestClassifier(criterion='entropy', max_depth=8, random_state=42)
```

Figure 37: Applying Random Forest model to the train data

15.3. Defining Accuracy of train and test data, F1 score of Random Forest

```
from sklearn.metrics import  confusion_matrix
cm_new = confusion_matrix(y_test, rf.predict(X_test_std))
print("Confusion matrix:\n")
print(cm_new)
tn = cm_new[0][0]
fp = cm_new[0][1]
fn = cm_new[1][0]
tp = cm_new[1][1]
print("\nTotal number of true positives", tp)
print("Total number of false negatives",fn)
print("Total number of false positives",fp)
print("Total number of true negatives",tn)

acc=float(tp+tn)/(tp+tn+fp+fn)
print('\nClassifier Accuracy: %.2f%%' % (acc * 100))

tpr = float(tp)/(tp+fn)
print('True Positive Rate (TPR/Recall/Sensitivity): %.2f%%' % (tpr * 100))

specificity = float (tn)/(tn+fp)
print ("True Negative Rate (TNR/Specificity/selectivity):%.2f%%" % (specificity*100))

fpr = float(fp)/(fp+tn)
print("False Positive Rate (FPR): %.2f%%" % (fpr * 100))
fnr = fn/ (fn+ tp)
print("False Negative Rate (FNR): %.2f%%" % (fnr*100))

precision=float(tp)/(tp+fp)
print("Precision/Positive Predictive value:%.2f%%" %(precision*100))

fScore = 2*((precision*tpr)/(precision+tpr))
print("F1-Score: %.2f%%" %(fScore*100))
```

Figure 38: Calculating accuracy and F1 score

```
Confusion matrix:

[[1851   65]
 [  18  527]]

Total number of true positives 527
Total number of false negatives 18
Total number of false positives 65
Total number of true negatives 1851

Classifier Accuracy: 96.63%
True Positive Rate (TPR/Recall/Sensitivity): 96.70%
True Negative Rate (TNR/Specificity/selectivity):96.61%
False Positive Rate (FPR): 3.39%
False Negative Rate (FNR): 3.30%
Precision/Positive Predictive value:89.02%
F1-Score: 92.70%
```

Figure 39: Result of accuracy and F1 score by Random Forest

```
#accuracy for trainig data by random forest
classificationSummary(y_train_SMOTE, rf.predict(X_train_SMOTE))
```

```
Confusion Matrix (Accuracy 0.9842)

       Prediction
Actual    0    1
     0 5608  138
     1   44 5702
```

Figure 40: Accuracy for train data

15.4. Root Mean Squared Error

```
[677] #RMSE using random forest
      from sklearn.metrics import mean_squared_error
      preds_rf = rf.predict(X_test_std)
      rmse_rf = np.sqrt(mean_squared_error(y_test, preds_rf))
      print("RMSE after random forest regression: %f" % (rmse_rf))

      RMSE after random forest regression: 0.183647
```

Figure 41: RMSE after Random Forest model

15.3. Sorting Important features of the data set

The most important features can be defined from Random Forest model, and this will help the business know the characteristic of the fraud transactions. The top four variables that have largest impact on the model are "_ERC20_uniq_rec_addr", "_ERC20_uniq_rec_token_name", "Time_Diff_between_first_and_last_(Mins)" and "avg_val_received".

```
plt.figure(figsize=(10,8))
df2 = pd.DataFrame(X_train_std,columns=X_train.columns)
sort = rf.feature_importances_.argsort()
plt.barh(df2.columns[sort], rf.feature_importances_[sort])
plt.xlabel("Feature Importance")
```

```
Text(0.5, 0, 'Feature Importance')
```

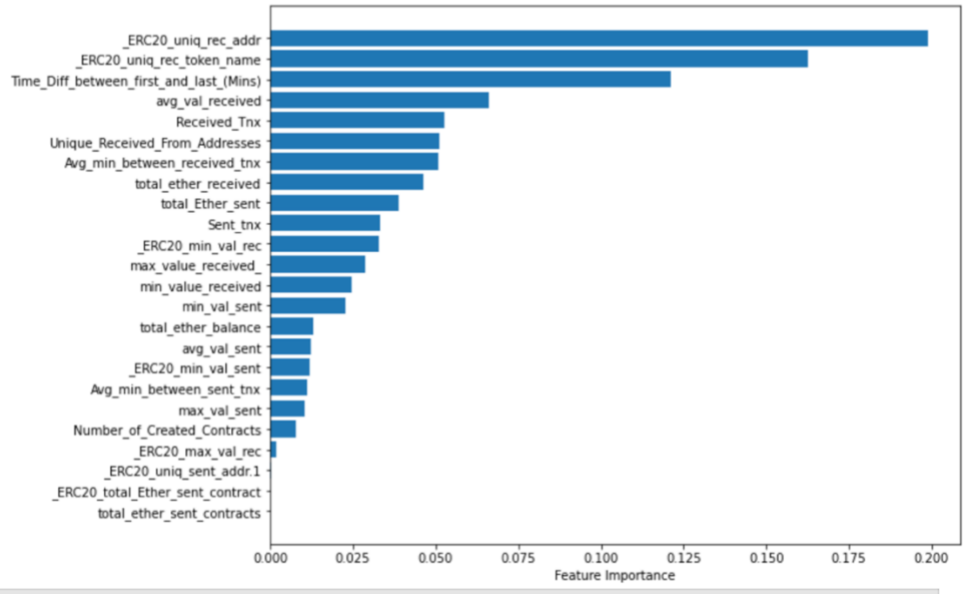


Figure 42: Defining important features by Random Forest

15.4. Defining AUC score

```
import sklearn.metrics as metrics
y_pred_proba11 = rf.predict_proba(X_test_std)[::,1] #calculate probabilities for flag YES (label=

fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba11) # calculate tpr and fpr values

auc = metrics.roc_auc_score(y_test, y_pred_proba11) #calculate auc value
print(auc)

plt.plot(fpr,tpr,label="test_data(AUC= %0.2f)" % auc, linewidth = 4) #plot ROC curve
plt.legend(prop={'size':12},loc='best') #set the legend properties
plt.title('\nROC plot by Random Forest',fontsize = 16) #title
plt.xlabel('False positive rate', fontsize = 16) #x and y labels
plt.ylabel('True positive rate', fontsize = 16)
plt.show() #display the plot
```

```
0.994556702610561
```

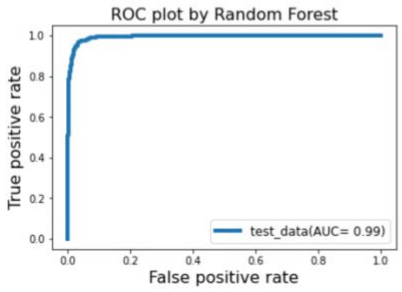


Figure 43: AUC score by Random Forest

16.0. XGboost

XGboost which is known as Extreme Gradient Boosting is also type of efficient and flexible software which can be used in different programming languages.

16.1. Applying the XGboost model with the appropriate parameters

```
import xgboost as xgb

xg_reg = xgb.XGBClassifier(colsample_bytree = 0.5,n_estimators = 84, random_state=55)

xg_reg.fit(X_train_SMOTE,y_train_SMOTE)
preds1 = xg_reg.predict(X_test_std)
```

Figure 44: Building XGBoost model

16.2. Calculating Accuracy of Train and Test Data, F1 score

```
#for train data set
classificationSummary(y_train_SMOTE, xg_reg.predict(X_train_SMOTE))

Confusion Matrix (Accuracy 0.9809)

       Prediction
Actual    0    1
     0 5590  156
     1   64 5682
```

Figure 45: Accuracy for train data by XGBoost

```
from sklearn.metrics import  confusion_matrix
cm_new = confusion_matrix(y_test, xg_reg.predict(X_test_std))
print("Confusion matrix:\n")
print(cm_new)
tn = cm_new[0][0]
fp = cm_new[0][1]
fn = cm_new[1][0]
tp = cm_new[1][1]
print("\nTotal number of true positives", tp)
print("Total number of false negatives",fn)
print("Total number of false positives",fp)
print("Total number of true negatives",tn)

acc=float(tp+tn)/(tp+tn+fp+fn)
print('\nClassifier Accuracy: %.2f%%' % (acc * 100))

tpr = float(tp)/(tp+fn)
print('True Positive Rate (TPR/Recall/Sensitivity): %.2f%%' % (tpr * 100))
specificity = float (tn)/(tn+fp)
print ("True Negative Rate (TNR/Specificity/selectivity):%.2f%%" % (specificity*100))

fpr = float(fp)/(fp+tn)
print("False Positive Rate (FPR): %.2f%%" % (fpr * 100))
fnr = fn/ (fn+ tp)
print("False Negative Rate (FNR): %.2f%%" % (fnr*100))

precision=float(tp)/(tp+fp)
print("Precision/Positive Predictive value:%.2f%%" %(precision*100))

fScore = 2*((precision*tpr)/(precision+tpr))
print("F1-Score: %.2f%%" %(fScore*100))
```

Figure 46: Calculating Accuracy and F1 score of test data by XGBoost

```
Confusion matrix:

[[1857   59]
 [  14  531]]

Total number of true positives 531
Total number of false negatives 14
Total number of false positives 59
Total number of true negatives 1857

Classifier Accuracy: 97.03%
True Positive Rate (TPR/Recall/Sensitivity): 97.43%
True Negative Rate (TNR/Specificity/selectivity):96.92%
False Positive Rate (FPR): 3.08%
False Negative Rate (FNR): 2.57%
Precision/Positive Predictive value:90.00%
F1-Score: 93.57%
```

Figure 47: Accuracy and F1 score of test data by XGBoost

16.2. Calculating RMSE of XGboost

```
rmse2 = np.sqrt(mean_squared_error(y_test, preds1))
print("RMSE after XG Boost: %f" % (rmse2))
```

Figure 48: RMSE by XGBoost

```
↱  RMSE after XG Boost: 0.172229
```

Figure 49: Result of RMSE

16.3. Sorting Important features by XGboost

The top five variables that have largest impact on the model are "_ERC20_uniq_rec_addr", "total_ether_received", "Time_Diff_between_first_and_last_(Mins)", "Avg_min_between_received_tnx", and "max_value_received".

```
[686] plt.figure(figsize=(10,8))
      sort_xgb = xg_reg.feature_importances_.argsort()
      plt.barh(df2.columns[sort], xg_reg.feature_importances_[sort])
      plt.xlabel("Feature Importance of Xg boost")

      Text(0.5, 0, 'Feature Importance of Xg boost')
```

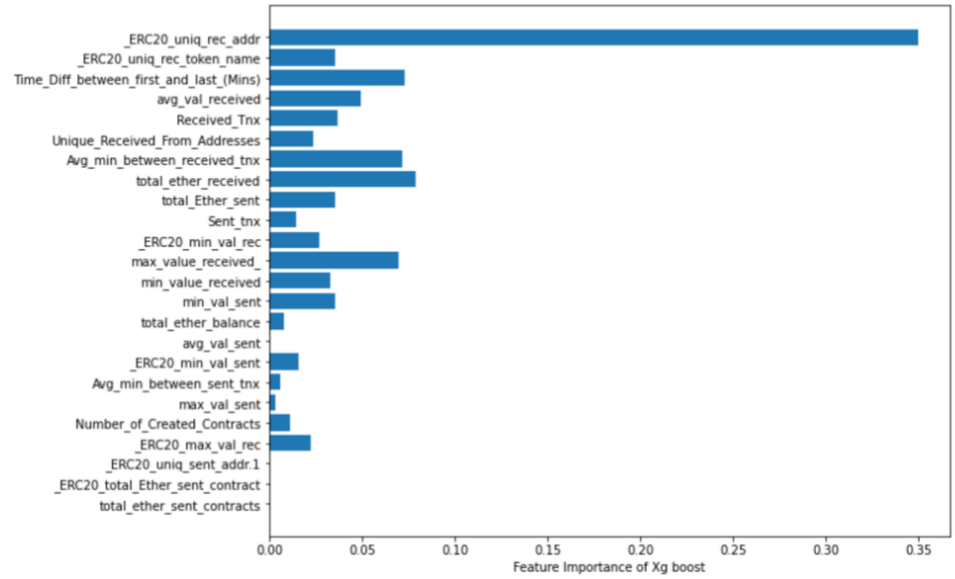


Figure 50: Defining important features by XGBoost

16.4. Defining AUC score

```
import sklearn.metrics as metrics
y_pred_proba14 = xg_reg.predict_proba(X_test_std)[::,1] #calculate probabilities for flag YES (la

fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba14) # calculate tpr and fpr values

auc = metrics.roc_auc_score(y_test, y_pred_proba14) #calculate auc value
print(auc)

plt.plot(fpr,tpr,label="test_data(AUC= %0.2f)" % auc, linewidth = 4) #plot ROC curve
plt.legend(prop={'size':12},loc='best') #set the legend properties
plt.title('\nROC plot by XG boost',fontsize = 16) #title
plt.xlabel('False positive rate', fontsize = 16) #x and y labels
plt.ylabel('True positive rate', fontsize = 16)
plt.show() #display the plot
```

```
0.9954741338032216
```

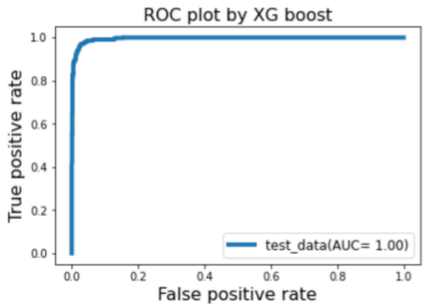


Figure 51: AUC score by XGBoost

17.0. Model Comparison

Models ignoring outliers in variables			
	Logistic Regression	Random Forest	XGboost
Accuracy for Test data	61.28%	96.22%	96.99%
Accuracy for Train data	71.01%	97.87%	98.22%
F1 score	49.34%	91.82%	93.44%
AUC score	0.83	0.9945	0.9933
RMSE	0.6222	0.1943	0.1734

Models after removing outliers from variables and setting the ranges			
	Logistic Regression	Random Forest	XGboost
Accuracy for Test data	78.10%	96.63%	97.03%
Accuracy for Train data	83.40%	98.42%	98.09%
F1 score	64.70%	92.70%	93.57%
AUC score	0.93	0.9945	0.9954
RMSE	0.4679	0.1836	0.1722

Model Recommendation

18.0. Model Selection

After handling the outliers and setting variables between ranges according to outliers, the accuracy, F1 score and AUC score for all models increased, and root mean square errors decreased in all three models. Among three models of logistic regression, random forest and xgboost, the accuracy, F1score, and AUC score of xgboost model are better than the other two

models. Moreover, the difference of the accuracy between test data and train data is only 1.06% which is acceptable and be able to handle variable drifting in the future.

19.0. Model Theory

In the selected model, XGboost, the best parameter to be used in this model is defined first and apply to the model and build the model using the clean training data which does not include outliers, missing values and balanced records for both fraud and non-fraud transaction. Important Features which directly relates to fraud transaction are also classified from XGboost model.

Variables such as "_ERC20_uniq_rec_addr", "total_ether_received", "Time_Diff_between_first_and_last_(Mins)", "Avg_min_between_received_tnx", and "max_value_received" are the most important input variables to predict fraud transactions that the organization should be aware of these features in detecting the scams.

Variables such as "total_ether_sent_contracts", "_ERC20_uniq_sent_addr.1" and "_ERC20_total_Ether_sent_contract" have no distribution in predicting the fraud transaction.

19.1. Model Assumption and Limitations

This model is based on small amount of transaction records from Ethereum software which are from past two years. As the cryptocurrency market has been changing these years, there might be some limitations to use new added input variables or the trend changes on Ethereum. Moreover, as the data is imbalanced with only 22% of fraud transactions, it is assumed to have similar characteristics of fraud transactions after making the data balanced and it might have some impact on the accuracy of the model.

20.0. Model Sensitivity to Key Drivers

It is obvious that this XGboost model is sensitive to those top variables from feature importance. Its performance can change depending on these variables. By looking at those predictive variables, most of the important features are related to received transactions which makes sense for the users to be aware of receiving transactions with advertisements offering incentives, etc.

Validation and Governance

21.0. Variable Level Monitoring

In this project of predicting fraud transaction, input variables level can be monitored by analyzing their statistics such as mean, standard deviation, minimum, maximum, handling the outliers and setting the acceptable ranges, imputing missing values with zero, mean or median.

21.1. Build Statistics

By looking at the descriptive statistics of selected variables, there are outliers in each variable which needs to be capped and floored to improve the quality of the predicting model.

```
df1.describe()
```

	FLAG	Avg_min_between_sent_tnx	Avg_min_between_received_tnx	Time_Diff_between_first_and_last_(Mins)
count	9841.000000	9841.000000	9841.000000	9.841000e+03
mean	0.221421	5086.878721	8004.851184	2.183333e+05
std	0.415224	21486.549974	23081.714801	3.229379e+05
min	0.000000	0.000000	0.000000	0.000000e+00
25%	0.000000	0.000000	0.000000	3.169300e+02
50%	0.000000	17.340000	509.770000	4.663703e+04
75%	0.000000	565.470000	5480.390000	3.040710e+05
max	1.000000	430287.670000	482175.490000	1.954861e+06

8 rows × 25 columns

Figure 52: Descriptive statistics for selected variables 1

```
df1.describe()
```

Sent_tnx	Received_Tnx	Number_of_Created_Contracts	Unique_Received_From_Addresses	min_value_received
9841.000000	9841.000000	9841.000000	9841.000000	9841.000000
115.931714	163.700945	3.729702	30.360939	43.845153
757.226361	940.836550	141.445583	298.621112	325.929139
0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	1.000000	0.000000	1.000000	0.001000
3.000000	4.000000	0.000000	2.000000	0.095856
11.000000	27.000000	0.000000	5.000000	2.000000
10000.000000	10000.000000	9995.000000	9999.000000	10000.000000

Figure 53 Descriptive statistics for selected variables 2

```
df1.describe()
```

_ERC20_total_Ether_sent_contract	_ERC20_uniq_rec_addr	_ERC20_uniq_sent_addr.1	_ERC20_min_val_rec	_ERC20_max_val_rec
9012.000000	9012.000000	9012.000000	9012.000000	9.012000e+03
110.939207	7.598535	0.003440	485.614688	1.252524e+08
6128.634953	81.818470	0.065698	16883.278712	1.053741e+10
0.000000	0.000000	0.000000	0.000000	0.000000e+00
0.000000	0.000000	0.000000	0.000000	0.000000e+00
0.000000	1.000000	0.000000	0.000000	0.000000e+00
0.000000	2.000000	0.000000	0.001523	9.900000e+01
416000.000000	4293.000000	3.000000	990000.000000	1.000000e+12

Figure 54: Descriptive statistics for selected variables 3

```
df1.describe()
```

max_value_received_	...	total_ether_received	total_ether_sent_contracts	total_ether_balance
9841.000000	...	9.841000e+03	9841.000000	9.841000e+03
523.152481	...	1.163832e+04	0.000008	1.477395e+03
13008.821539	...	3.642048e+05	0.000516	2.424254e+05
0.000000	...	0.000000e+00	0.000000	-1.560535e+07
1.000000	...	2.670424e+00	0.000000	6.214900e-04
6.000000	...	3.052963e+01	0.000000	1.722000e-03
67.067040	...	1.010000e+02	0.000000	4.452000e-02
800000.000000	...	2.858159e+07	0.046029	1.428864e+07

Figure 55: Descriptive statistics for selected variables 4

_ERC20_min_val_sent	_ERC20_uniq_rec_token_name
9.012000e+03	9012.000000
1.174126e+04	4.826676
1.053567e+06	16.678607
0.000000e+00	0.000000
0.000000e+00	0.000000
0.000000e+00	1.000000
0.000000e+00	2.000000
1.000000e+08	737.000000

Figure 56: Descriptive statistics for selected variables 5

21.1.1 Distribution of these variables

These boxplots show the outliers of each variable and six variables that have missing values are not shown before imputing the missing values. Therefore, the data set needs to remove outliers and handle missing values for better model.

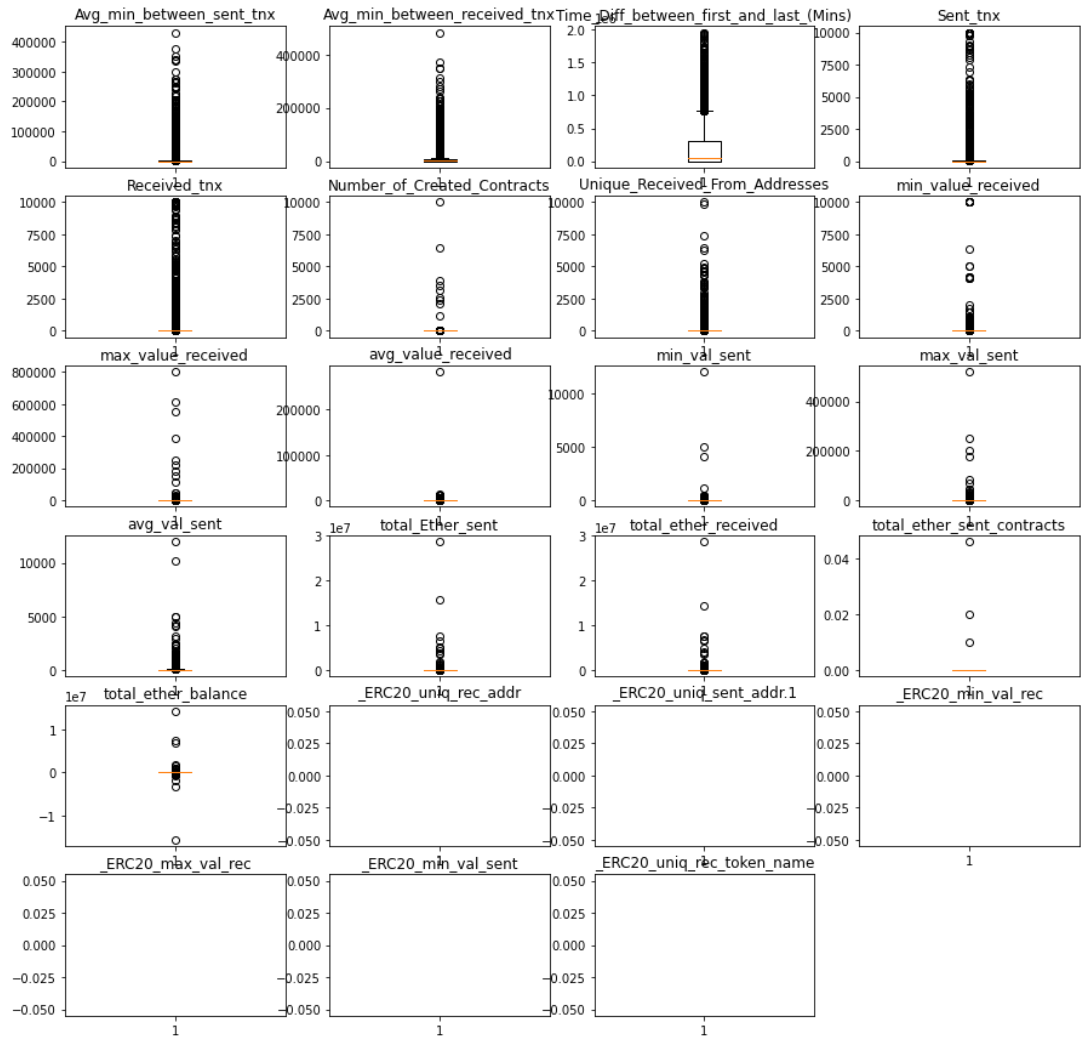


Figure 57: Distribution of varibales before imputing missing values

21.2. Acceptable Ranges

After calculating the outliers in data preparation section, these upper limits and lower limits for each variable can be set.

Variables	Lower Limit	Upper Limit
Avg_min_between_sent_tnx	-59372.771202	69546.52864309154
Avg_min_between_received_tnx	-61240.293219096246	77249.99558674182
Time_Diff_between_first_and_last_(Mins)	-750480.519567207	1187147.0348705298

Sent_tnx	-2155.7473675487086	2387.6107960620707
Received_Tnx	-2658.8087054492635	2986.2105955010875
Number_of_Created_Contracts	-420.6070467102196	428.0664512422794
Unique_Received_From_Addresses	-865.5023978821241	926.2242757400654
min_value_received	-933.94226494196	1021.6325718989767
max_value_received_	-38503.31213514954	39549.617096191716
avg_val_received	-8554.294987722684	8755.718430444156
min_val_sent	-411.02895516469727	420.62913420849367
max_val_sent	-19573.02063078748	20202.255224823453
avg_val_sent	-672.4849144001221	761.996376870806
total_Ether_sent	-1064807.2113808494	1085129.0580569583
total_ether_received	-1080976.000550022	1104252.6378102077
total_ether_sent_contracts	-0.0015397117008669001	0.001555163121047776
total_ether_balance	-725798.8733613549	728753.6639296662
_ERC20_total_Ether_sent_contract	-18274.965652206363	18496.844065581154
_ERC20_uniq_rec_addr	-237.8568753648363	253.05394593740618
_ERC20_uniq_sent_addr.1	-0.19365373933571933	0.20053345527002914
_ERC20_min_val_rec	-50164.22144788942	51135.45082307073
_ERC20_max_val_rec	-31486970011.54963	31737474731.8544
_ERC20_min_val_sent	-3148960.1117158257	3172442.6263105045
_ERC20_uniq_rec_token_name	-45.20914494536328	54.862496032802255

21.3. Caps and Floors

All the 24 variables have outliers and so that they are going to be capped and floored between the acceptable ranges shown above.

```
UL1 = df1['Avg_min_between_sent_tnx'].mean() + 3*df1['Avg_min_between_sent_tnx'].std()
UL2 = df1['Avg_min_between_received_tnx'].mean() + 3*df1['Avg_min_between_received_tnx'].std()
UL3 = df1['Time_Diff_between_first_and_last_(Mins)'].mean() + 3*df1['Time_Diff_between_first_and_last_(Mins)'].std()
UL4 = df1['Sent_tnx'].mean() + 3*df1['Sent_tnx'].std()
UL5 = df1['Received_Tnx'].mean() + 3*df1['Received_Tnx'].std()
UL6 = df1['Number_of_Created_Contracts'].mean() + 3*df1['Number_of_Created_Contracts'].std()
UL7 = df1['Unique_Received_From_Addresses'].mean() + 3*df1['Unique_Received_From_Addresses'].std()
UL8 = df1['min_value_received'].mean() + 3*df1['min_value_received'].std()
UL9 = df1['max_value_received_'].mean() + 3*df1['max_value_received_'].std()
UL10 = df1['avg_val_received'].mean() + 3*df1['avg_val_received'].std()
UL11 = df1['min_val_sent'].mean() + 3*df1['min_val_sent'].std()
UL12 = df1['max_val_sent'].mean() + 3*df1['max_val_sent'].std()
UL13 = df1['avg_val_sent'].mean() + 3*df1['avg_val_sent'].std()
UL14 = df1['total_Ether_sent'].mean() + 3*df1['total_Ether_sent'].std()
UL15 = df1['total_ether_received'].mean() + 3*df1['total_ether_received'].std()
UL16 = df1['total_ether_sent_contracts'].mean() + 3*df1['total_ether_sent_contracts'].std()
UL17 = df1['total_ether_balance'].mean() + 3*df1['total_ether_balance'].std()
UL18 = df1['_ERC20_total_Ether_sent_contract'].mean() + 3*df1['_ERC20_total_Ether_sent_contract'].std()
UL19 = df1['_ERC20_uniq_rec_addr'].mean() + 3*df1['_ERC20_uniq_rec_addr'].std()
UL20 = df1['_ERC20_uniq_sent_addr.1'].mean() + 3*df1['_ERC20_uniq_sent_addr.1'].std()
UL21 = df1['_ERC20_min_val_rec'].mean() + 3*df1['_ERC20_min_val_rec'].std()
UL22 = df1['_ERC20_max_val_rec'].mean() + 3*df1['_ERC20_max_val_rec'].std()
UL23 = df1['_ERC20_min_val_sent'].mean() + 3*df1['_ERC20_min_val_sent'].std()
UL24 = df1['_ERC20_uniq_rec_token_name'].mean() + 3*df1['_ERC20_uniq_rec_token_name'].std()
```

Figure 58: Calculating Upper Limit

```
LL1 = df1['Avg_min_between_sent_tnx'].mean() - 3*df1['Avg_min_between_sent_tnx'].std()
LL2 = df1['Avg_min_between_received_tnx'].mean() - 3*df1['Avg_min_between_received_tnx'].std()
LL3 = df1['Time_Diff_between_first_and_last_(Mins)'].mean() - 3*df1['Time_Diff_between_first_and_last_(Mins)'].std()
LL4 = df1['Sent_tnx'].mean() - 3*df1['Sent_tnx'].std()
LL5 = df1['Received_Tnx'].mean() - 3*df1['Received_Tnx'].std()
LL6 = df1['Number_of_Created_Contracts'].mean() - 3*df1['Number_of_Created_Contracts'].std()
LL7 = df1['Unique_Received_From_Addresses'].mean() - 3*df1['Unique_Received_From_Addresses'].std()
LL8 = df1['min_value_received'].mean() - 3*df1['min_value_received'].std()
LL9 = df1['max_value_received_'].mean() - 3*df1['max_value_received_'].std()
LL10 = df1['avg_val_received'].mean() - 3*df1['avg_val_received'].std()
LL11 = df1['min_val_sent'].mean() - 3*df1['min_val_sent'].std()
LL12 = df1['max_val_sent'].mean() - 3*df1['max_val_sent'].std()
LL13 = df1['avg_val_sent'].mean() - 3*df1['avg_val_sent'].std()
LL14 = df1['total_Ether_sent'].mean() - 3*df1['total_Ether_sent'].std()
LL15 = df1['total_ether_received'].mean() - 3*df1['total_ether_received'].std()
LL16 = df1['total_ether_sent_contracts'].mean() - 3*df1['total_ether_sent_contracts'].std()
LL17 = df1['total_ether_balance'].mean() - 3*df1['total_ether_balance'].std()
LL18 = df1['_ERC20_total_Ether_sent_contract'].mean() - 3*df1['_ERC20_total_Ether_sent_contract'].std()
LL19 = df1['_ERC20_uniq_rec_addr'].mean() - 3*df1['_ERC20_uniq_rec_addr'].std()
LL20 = df1['_ERC20_uniq_sent_addr.1'].mean() - 3*df1['_ERC20_uniq_sent_addr.1'].std()
LL21 = df1['_ERC20_min_val_rec'].mean() - 3*df1['_ERC20_min_val_rec'].std()
LL22 = df1['_ERC20_max_val_rec'].mean() - 3*df1['_ERC20_max_val_rec'].std()
LL23 = df1['_ERC20_min_val_sent'].mean() - 3*df1['_ERC20_min_val_sent'].std()
LL24 = df1['_ERC20_uniq_rec_token_name'].mean() - 3*df1['_ERC20_uniq_rec_token_name'].std()
```

Figure 59: Calculating Lower Limit

```
df1['Avg_min_between_sent_tnx'] = np.where(df1['Avg_min_between_sent_tnx'] > UL1,UL1,
                                  np.where(df1['Avg_min_between_sent_tnx'] < LL1,LL1,df1['Avg_min_between_sent_tnx']))

df1['Avg_min_between_received_tnx'] = np.where(df1['Avg_min_between_received_tnx'] > UL2,UL2,
                                    np.where(df1['Avg_min_between_received_tnx'] < LL2,LL2,df1['Avg_min_between_receive

df1['Time_Diff_between_first_and_last_(Mins)'] = np.where(df1['Time_Diff_between_first_and_last_(Mins)'] > UL3,UL3,
                                                 np.where(df1['Time_Diff_between_first_and_last_(Mins)'] < LL3,LL3,df1['

df1['Sent_tnx'] = np.where(df1['Sent_tnx'] > UL4,UL4,
                  np.where(df1['Sent_tnx'] < LL4,LL4,df1['Sent_tnx']))

df1['Received_Tnx'] = np.where(df1['Received_Tnx'] > UL5,UL5,
                      np.where(df1['Received_Tnx'] < LL5,LL5,df1['Received_Tnx']))

df1['Number_of_Created_Contracts'] = np.where(df1['Number_of_Created_Contracts'] > UL6,UL6,
                      np.where(df1['Number_of_Created_Contracts'] < LL6,LL6,df1['Number_of_Created_Contracts']))

df1['Unique_Received_From_Addresses'] = np.where(df1['Unique_Received_From_Addresses'] > UL7,UL7,
                      np.where(df1['Unique_Received_From_Addresses'] < LL7,LL7,df1['Unique_Received_From_Addresses']))

df1['min_value_received'] = np.where(df1['min_value_received'] > UL8,UL8,
                      np.where(df1['min_value_received'] < LL8,LL8,df1['min_value_received']))

df1['max_value_received_'] = np.where(df1['max_value_received_'] > UL9,UL9,
                      np.where(df1['max_value_received_'] < LL9,LL9,df1['max_value_received_']))

df1['avg_val_received'] = np.where(df1['avg_val_received'] > UL10,UL10,
                      np.where(df1['avg_val_received'] < LL10,LL10,df1['avg_val_received']))

df1['min_val_sent'] = np.where(df1['min_val_sent'] > UL11,UL11,
                      np.where(df1['min_val_sent'] < LL11,LL11,df1['min_val_sent']))
```

Figure 60: Setting Ranges for all variables 1

```
df1['max_val_sent'] = np.where(df1['max_val_sent'] > UL12,UL12,
                      np.where(df1['max_val_sent'] < LL12,LL12,df1['max_val_sent']))

df1['avg_val_sent'] = np.where(df1['avg_val_sent'] > UL13,UL13,
                      np.where(df1['avg_val_sent'] < LL13,LL13,df1['avg_val_sent']))

df1['total_Ether_sent'] = np.where(df1['total_Ether_sent'] > UL14,UL14,
                      np.where(df1['total_Ether_sent'] < LL14,LL14,df1['total_Ether_sent']))

df1['total_ether_received'] = np.where(df1['total_ether_received'] > UL15,UL15,
                      np.where(df1['total_ether_received'] < LL15,LL15,df1['total_ether_received']))

df1['total_ether_sent_contracts'] = np.where(df1['total_ether_sent_contracts'] > UL16,UL16,
                      np.where(df1['total_ether_sent_contracts'] < LL16,LL16,df1['total_ether_sent_contracts']))

df1['total_ether_balance'] = np.where(df1['total_ether_balance'] > UL17,UL17,
                      np.where(df1['total_ether_balance'] < LL17,LL17,df1['total_ether_balance']))

df1['_ERC20_total_Ether_sent_contract'] = np.where(df1['_ERC20_total_Ether_sent_contract'] > UL18,UL18,
                      np.where(df1['_ERC20_total_Ether_sent_contract'] < LL18,LL18,df1['_ERC20_total_Ether_sent_contract

df1['_ERC20_uniq_rec_addr'] = np.where(df1['_ERC20_uniq_rec_addr'] > UL19,UL19,
                      np.where(df1['_ERC20_uniq_rec_addr'] < LL19,LL19,df1['_ERC20_uniq_rec_addr']))

df1['_ERC20_uniq_sent_addr.1'] = np.where(df1['_ERC20_uniq_sent_addr.1'] > UL20,UL20,
                     np.where(df1['_ERC20_uniq_sent_addr.1'] < LL20,LL20,df1['_ERC20_uniq_sent_addr.1']))

df1['_ERC20_min_val_rec'] = np.where(df1['_ERC20_min_val_rec'] > UL21,UL21,
                      np.where(df1['_ERC20_min_val_rec'] < LL21,LL21,df1['_ERC20_min_val_rec']))

df1['_ERC20_max_val_rec'] = np.where(df1['_ERC20_max_val_rec'] > UL22,UL22,
                      np.where(df1['_ERC20_max_val_rec'] < LL22,LL22,df1['_ERC20_max_val_rec']))
```

Figure 61: Setting Ranges for all variables 2

```
df1['_ERC20_min_val_sent'] = np.where(df1['_ERC20_min_val_sent'] > UL23,UL23,
                      np.where(df1['_ERC20_min_val_sent'] < LL23,LL23,df1['_ERC20_min_val_sent']))

df1['_ERC20_uniq_rec_token_name'] = np.where(df1['_ERC20_uniq_rec_token_name'] > UL24,UL24,
                      np.where(df1['_ERC20_uniq_rec_token_name'] < LL24,LL24,df1['_ERC20_uniq_rec_token_name']))
```

Figure 62:Setting Ranges for all variables 3

21.3.1 Changes in distribution after setting acceptable ranges

These boxplots show those variables work after removing outliers and imputing missing values.

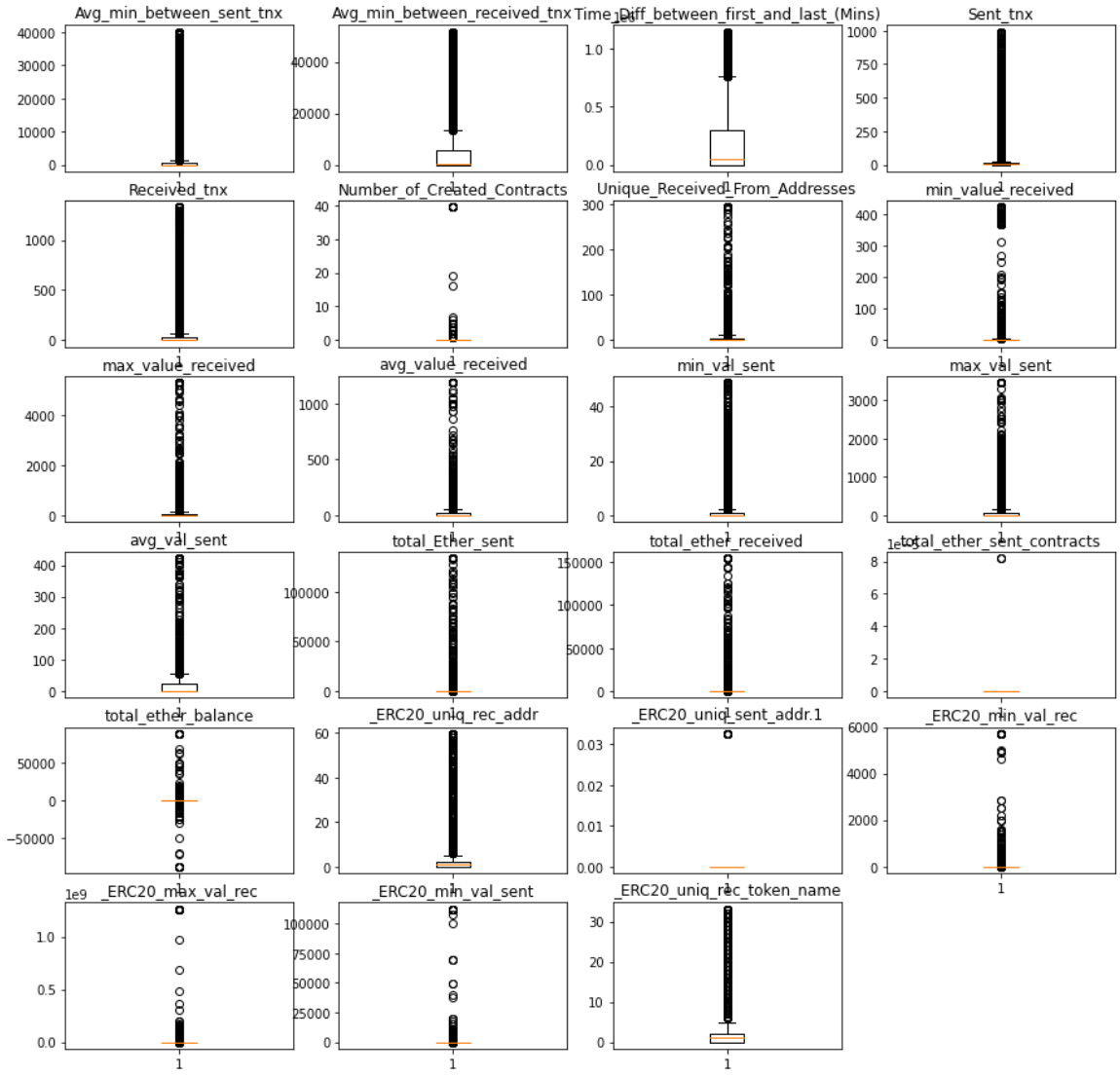


Figure 63: Variable distribution after setting ranges and imputation

21.4. Missing Values

This data set has missing values which have to be imputed. Missing value of a variable is replaced by the median value of that variable after analyzing the nature of those variable with missing values in section 11.1. After deciding which input variables to be include, there are 7 variables which have missing values. Those variables can be classified into two natures which

are related to “sent” and “received” transactions. It is found out that most of non-zero variables and variables that have missing values are fraud transaction. In order to reduce the false negative which, have more impact for both the business and the consumer, we decided to impute the missing values with median values as the data set still have the outliers after cap and floor it. Median values are better fit for those variables with outliers than mean values.

21.4. Variable Drift Monitoring

The variables to be used in the model will not drift frequently as long as there is no inflation or country’s policy changes. However, there might be a drift in some conditions such as the value of cryptocurrency falls immediately in the market and most of the users make lots of activities with their accounts. So that, the variables will have huge impact from users who change their mind. This will also affect the prediction model. So, it is important to keep in touch with how the cryptocurrency market works and the massive users’ activities. To be able to adapt with changeable input variables, the selected model needs to be trained with more recent sample data very often which will probably make the business costly.

21.5. Tolerance for Drift of Each Variable

In order to handle variable drifting, the input variables should not exceed the acceptable ranges shown in section 21.2.

22.0. Model Health and Stability

	Logistic Regression	Random Forest	XG boost
RMSE	0.4679	0.1836	0.1722
F1 score	64.70%	92.70%	93.57%
AUC score	0.93	0.9945	0.9954

23.0. Risk Tiering of the selected model

Minimal Risk	The model runs above 90% and it just shows a red flag if it sees any suspicious transaction in Ethereum.
Limited Risk	The Ethereum account users should be aware of providing their information for the software, similar scammed websites and advertisement. The model can predict a normal transaction as a scam and block the transaction. Solution – The model has to be modified its parameter and assumptions.
High Risk	The model will wrongly predict the fraud transaction as a normal one and users will be scammed and lose their money on Ethereum. Solution – The model has to be rebuilt by collecting more recent real data records.
Unacceptable Risk	The model couldn't predict the worst scam correctly which might threaten the user to get both Ether and money in real life by stealing user's information and tracing them. The accuracy of the model should always be above 88% even with the variable is drifting. Solution - More recent samples have to be collected and different models have to be rebuilt with multiple techniques and chose the best model again.

24.0. Model Risk Manage

- Overview: The purpose of this model is to detect the upcoming fraud transaction in Ethereum software.
- Problem: If the model predicts the normal transaction as suspicious, the information of that users will be recognized as fraud and the model shows the red flag to that user which make the user not be able to use Ethereum itself and in other third-party applications. Moreover, when the model cannot predict the fraud transaction and doesn't make an alert of it, the real user will be scammed and lose its money on Ethereum.
- Solution: The model needs to be trained with more samples and let it remember and experience different characteristics of any fraud transaction. The model should be updated frequently depending on the changes of cryptocurrency market which have the direct impact on the nature of variables input into the model and new facilities added to the Ethereum.

Conclusion and Recommendation

25.0. Impacts on Business Problem

By applying this model to the Ethereum platform and its software, it can reduce the number of fraud transactions entering normal users account that this model will alert the receiver to be aware of the transaction if it is suspicious according to the important features defined from the selected model. The scammers also cannot transfer ethers, make transactions, create smart contracts easily and they will have a warning before they do those scams. When the user is scammed, the technicians can trace the usage of the hackers' account and block its account and transactions.

Therefore, this model will greatly reduce the fraud transactions and Ethereum users being scammed and losing their property. In this way, the Ethereum organization will become more common around the world which helps the value of its Ether increase quicky.

26.0. Recommendation for next steps

This model should be modified with updated input variables in order to increase the accuracy of the model and to be familiar with new features of fraud transactions on the software. As the variables are drifting, the model should be checked and modified frequently to be able to adapt with the changes in the market and policy of the country. Although it might be costly to collect more recent data and modify the model very often, the popularity of Ethereum will surely increase with the higher performance. In conclusion, the company should apply this model to the Ethereum software which will make the “Ether” in demand and boost the Ethereum’s profit.

References

27.0. References

ALIYEV, V. (2020). *Kaggle*. Retrieved from www.kaggle.com: https://www.kaggle.com/datasets/vagifa/ethereum-frauddetection-dataset

CoinDesk. (2022). Retrieved from coindesk.com: https://www.coindesk.com/price/ethereum/

Shin, T. (2021, Feb 26). *Towards Data Science*. Retrieved from https://towardsdatascience.com/: https://towardsdatascience.com/understanding-feature-importance-and-how-to-implement-it-in-python-ff0287b20285#:~:text=Feature%20Importance%20refers%20to%20techniques,to%20predict%20a%20certain%20variable