Artificial Intelligence and Machine Learning

Project Report

Semester-IV (Batch-2022)

Fraud Detection In Blockchain Transactions

Supervised By:

Mrunal Paliwal

Submitted By:

Gopika Ashutosh Goyal,2210990328 (G7)

Department of Computer Science and Engineering
Chitkara University Institute of Engineering & Technology,
Chitkara University, Punjab

ABSTRACT

In today's digital era, the rise of cryptocurrencies has brought about significant advancements in financial transactions. However, along with these advancements comes the challenge of ensuring the security and integrity of transactions within blockchain ecosystems like Ethereum.

Ethereum is a digital playground where people can build all sorts of cool stuff, powered by a worldwide network of computers. Ether is the currency used in this digital world. We can buy or sell, or trade Ether like money.

Our project focuses on leveraging machine learning techniques to address this challenge by detecting and preventing fraudulent activities within blockchain transactions. By employing algorithms such as logistic regression, random forest, and decision trees, we aim to enhance transaction security and protect against fraudulent behaviour.

The problem statement revolves around the need to safeguard Ethereum transactions from fraudulent activities, which threaten the trust and reliability of the blockchain network. Through our project, we aim to contribute to the ongoing efforts to fortify the security measures within blockchain ecosystems, ensuring the continued growth and adoption of this transformative technology.

In conclusion, our research underscores the importance of proactive measures to combat fraudulent behaviour in blockchain transactions. By harnessing the power of machine learning, we strive to create a safer and more trustworthy environment for conducting financial transactions within Ethereum and other blockchain networks**.**

Table of Contents

1. INTRODUCTION

1.1 BACKGROUND:

Blockchain technology has garnered significant attention in recent years for its potential to revolutionise various industries by offering a decentralised and tamper-proof ledger system. At its core, blockchain records transactions across a network of computers (nodes), ensuring transparency and security. Ethereum, one of the leading blockchain platforms, goes beyond simple transactions by enabling the creation of smart contracts and decentralised applications. This versatility has made Ethereum a preferred choice for developers seeking to build innovative solutions on blockchain technology.

1.2 OBJECTIVE:

In light of the growing adoption of blockchain, particularly Ethereum, the detection of fraudulent activities within the network has become a pressing concern. Fraudulent transactions not only result in financial losses but also undermine trust in the integrity of the blockchain ecosystem. Therefore, the primary objective of our project is to design and implement an effective fraud detection system specifically tailored for Ethereum transactions.

Our objectives include achieving high accuracy in identifying fraudulent transactions while minimising false positives. By accurately pinpointing fraudulent activities, we aim to enhance the security and reliability of the Ethereum network, thereby promoting its continued growth and adoption.

1.3 INTRODUCTION TO BLOCKCHAIN TECHNOLOGY:

Blockchain technology is often described as a distributed ledger system that records transactions in a secure and transparent manner. This technology gained prominence primarily due to its innovative approach to decentralisation and immutability. Instead of relying on a central authority, such as a bank or government, to verify and authenticate transactions, blockchain operates through a network of interconnected computers, known as nodes.

Ether (ETH) is the digital currency native to the Ethereum blockchain. It's used for transactions, paying fees, and powering smart contracts and decentralised applications (DApps). Additionally, it's traded on exchanges and plays a role in Ethereum's governance and development.

1.4 OVERVIEW OF MACHINE LEARNING IN FRAUD DETECTION:

Machine learning algorithms play a crucial role in fraud detection by leveraging data-driven techniques to identify fraudulent activities within large datasets. These algorithms have the capability to:

1. Analyse Large Datasets: Machine learning algorithms can process vast amounts of transactional data collected by financial institutions, including transaction histories, account information, and user behaviour. By analysing this data, machine learning models can uncover patterns and anomalies indicative of fraudulent behaviour.

2. Identify patterns: Machine learning algorithms excel at identifying complex patterns and relationships within data. They can detect subtle deviations from normal behaviour that may signify fraudulent activities, such as unusual spending patterns, unexpected account activity, or atypical transaction amounts.

3. Adapt to New Fraud Tactics: One of the key advantages of machine learning in fraud detection is its ability to adapt to evolving fraud tactics. As fraudsters continuously develop new techniques to circumvent detection, machine learning models can be trained on updated datasets to learn and recognise emerging patterns of fraudulent behaviour.

4. Model Training and Evaluation: Machine learning models are trained on labeled datasets, where instances of fraudulent and legitimate transactions are identified. During training, the model learns to differentiate between fraudulent and non-fraudulent transactions based on the provided features. The model's performance is evaluated using metrics such as accuracy, precision, recall, and F1-score to assess its effectiveness in detecting fraud while minimising false positives.

Overall, machine learning algorithms provide powerful tools for fraud detection in the financial sector, enabling organisations to proactively identify and mitigate fraudulent activities, protect customer assets, and maintain trust in the integrity of financial systems.

1.5 SIGNIFICANCE:

The significance of detecting fraud in blockchain transactions cannot be overstated. Fraudulent activities pose a serious threat to the integrity and trustworthiness of decentralised systems like Ethereum. Detecting and preventing fraud is essential for maintaining the credibility of blockchain technology and ensuring its viability for various applications.

Our work in developing a robust fraud detection system for Ethereum transactions contributes to enhancing the security and trust in decentralised systems. By effectively identifying and mitigating fraudulent activities, we aim to bolster confidence among users, developers, and stakeholders in the Ethereum ecosystem. Additionally, our project aligns with broader efforts to advance the adoption of blockchain technology across industries by addressing one of its key challenges: security.

2. PROBLEM DEFINITION AND REQUIREMENT

2.1 PROBLEM STATEMENT:

The Ethereum blockchain, renowned for its decentralised and transparent nature, faces the persistent challenge of distinguishing legitimate transactions from fraudulent ones. As the Ethereum network continues to grow in popularity and usage, the risk of fraudulent activities escalates, posing a threat to the integrity and trustworthiness of the blockchain ecosystem. The primary challenge at hand is to develop robust algorithms capable of effectively identifying and flagging suspicious transactions within the Ethereum blockchain.

Our task entails the development of algorithms that can analyse transaction data and discern patterns indicative of fraudulent behaviour. These algorithms must be capable of detecting anomalies, identifying irregularities, and distinguishing between legitimate transactions and fraudulent activities. By accurately pinpointing fraudulent transactions, we aim to mitigate financial losses, protect users from malicious actors, and uphold the credibility of the Ethereum network.

2.2 SOFTWARE REQUIREMENT:

Our solution will be implemented using Python, a versatile and widely adopted programming language suitable for data analysis and machine learning tasks. In addition to Python, we will leverage several libraries to facilitate various aspects of our project:

1. **NumPy:** Utilised for efficient numerical operations and array manipulation.

2. **Matplotlib:** Employed for creating visualisations and graphs to analyse data.

3. **Seaborn:** Used to enhance the aesthetics of visualisations and statistical data exploration.

4. **Pandas:** Utilised for data manipulation and analysis, offering powerful data structures and functions.

5. **Scikit-learn (sklearn):** Utilised for implementing machine learning algorithms, model evaluation, and preprocessing.

6. **Imbalanced-learn (imblearn):** Used for handling imbalanced datasets, a common challenge in fraud detection.

Methodologies:-

1. **Logistic Regression:** Applied for binary classification tasks, effectively identifying patterns in categorical data.

2. **Decision Tree:** Employed for both classification and regression tasks, partitioning data into subsets based on feature values.

3. **Random Forest:** Utilised as an ensemble learning method, constructing multiple decision trees to improve accuracy and robustness.

These software components provide the essential framework for developing our fraud detection algorithms effectively. By leveraging these libraries, we can streamline the development process and ensure the scalability and reliability of our solution.

2.3 HARDWARE REQUIREMENT:

In terms of hardware, we require a computer with sufficient processing power and memory to support the computational demands of our algorithms.

While the specific hardware specifications may vary depending on the scale of the project and the size of the dataset, a standard computer with a multi-core processor and ample RAM should suffice for development and testing purposes.

2.4 DATA SET:

For training and evaluating our fraud detection algorithms, we will utilise the Ethereum fraud detection dataset sourced from Kaggle. This dataset contains label transaction data, including features such as sender address, receiver address, transaction amount, etc. By leveraging this dataset, we can train supervised learning models to differentiate between legitimate and fraudulent transactions based on their attributes and characteristics.

We've got everything we need to build a strong fraud detection system for Ethereum transactions. With the right software, hardware, and data, we're ready to create smart algorithms that can spot and stop fraudulent activities. Our aim is to make Ethereum safer and more trustworthy for everyone who uses it.

Here are some examples illustrating how specific features can be used to detect fraud in Ethereum blockchain transactions:

Unusually High Number of Transactions to Contract Addresses:
Feature: Total_Ether_Sent_Contracts
Pattern: A sudden increase in the total amount of Ether sent to contract addresses by an account could indicate attempts to exploit vulnerabilities in smart contracts or engage in fraudulent activities such as Ponzi schemes or token scams.

Significant Deviation in Average Time Between ERC20 Token Transactions:
Feature: ERC20_Avg_Time_Between_Sent_Tnx / ERC20_Avg_Time_Between_Rec_Tnx
Pattern: A significant deviation from the average time between ERC20 token transactions for an account may suggest abnormal behaviour, such as attempts to artificially inflate transaction volumes or manipulate token prices.

Large and Irregular Value of Ether Sent or Received:
Feature: Max_Val_Sent / Max_Value_Received
Pattern: Sudden spikes or irregular patterns in the maximum value of Ether sent or received by an account could indicate fraudulent behaviour, such as large-scale transfers or attempts to manipulate markets.

Unusual Increase in Number of Unique Addresses Involved in Transactions:
Feature: Unique_Received_From_Addresses / Unique_Sent_To_Addresses
Pattern: An unusual increase in the number of unique addresses from which an account receives transactions or to which it sends transactions may suggest attempts to obfuscate fraudulent activities or engage in money laundering.

Changes in the Most Sent and Received Token Types:
Feature: ERC20_Most_Sent_Token_Type / ERC20_Most_Rec_Token_Type
Pattern: Sudden changes in the most commonly used ERC20 tokens for an account's transactions could indicate fraudulent behaviour, such as pump-and-dump schemes or token manipulation.

Unusual Increase in Total Number of Transactions:
Feature: Total_Transactions(Including_Tnx_to_Create_Contract)
Pattern: A sudden increase in the total number of transactions, especially those involving the creation of contracts, may suggest fraudulent activities such as spamming the network or attempting to overwhelm smart contracts with transactions.

3. PROPOSED DESIGN / METHODOLOGY

Our proposed design and methodology focus on utilising logistic regression, decision tree, and random forest algorithms to develop a fraud detection system tailored for Ethereum transactions. As we have only one file containing all the work and a CSV file containing the data, our approach will be streamlined and contained within this single file.

3.1 PROJECT DIRECTORY:

fraud_detection_system.py

ethereum_fraud_dataset.csv

fraud_detection_system.py: This Python script contains all the code for data preprocessing, model training, evaluation, and deployment. It encompasses the entire workflow of our fraud detection system.

ethereum_fraud_dataset.csv: The Ethereum fraud detection dataset sourced from Kaggle, containing transaction data.

3.2 METHODOLOGY:

Data Preprocessing:

- Load the Ethereum fraud detection dataset (ethereum_fraud_dataset.csv) into memory.

- Perform data cleaning, handling missing values, and encoding categorical variables if necessary.

- Split the dataset into training and testing sets.

Model Training:

- Implement logistic regression, decision tree, and random forest algorithms using scikit-learn.

- Train each model on the training dataset.

Model Evaluation:

- Evaluate the performance of each model using metrics such as accuracy, precision, recall, and F1-score on the testing dataset.

- Compare the performance of logistic regression, decision tree, and random forest algorithms to identify the most effective approach for fraud detection

3.3 ALGORITHM USED:

Logistic Regression: A simple yet effective linear model used for binary classification tasks. Logistic regression estimates the probability that a transaction is fraudulent based on its features.

Decision Tree: Decision tree classification is a method in machine learning where data is split into smaller subsets based on feature values, creating a tree-like structure of decision rules for predicting the class of new data points.

Random Forest: Random forest is an ensemble learning technique that constructs multiple decision trees during training and outputs the mode of the classes (for classification tasks) of the individual trees. It is known for its robustness and ability to handle large datasets with high dimensionality.

By utilising logistic regression, decision tree, and random forest algorithms within a single Python script, we ensure a cohesive and efficient approach to developing our fraud detection system. This streamlined methodology enables us to leverage the Ethereum fraud detection dataset effectively and identify the most suitable algorithm for detecting fraudulent activities within Ethereum transactions.

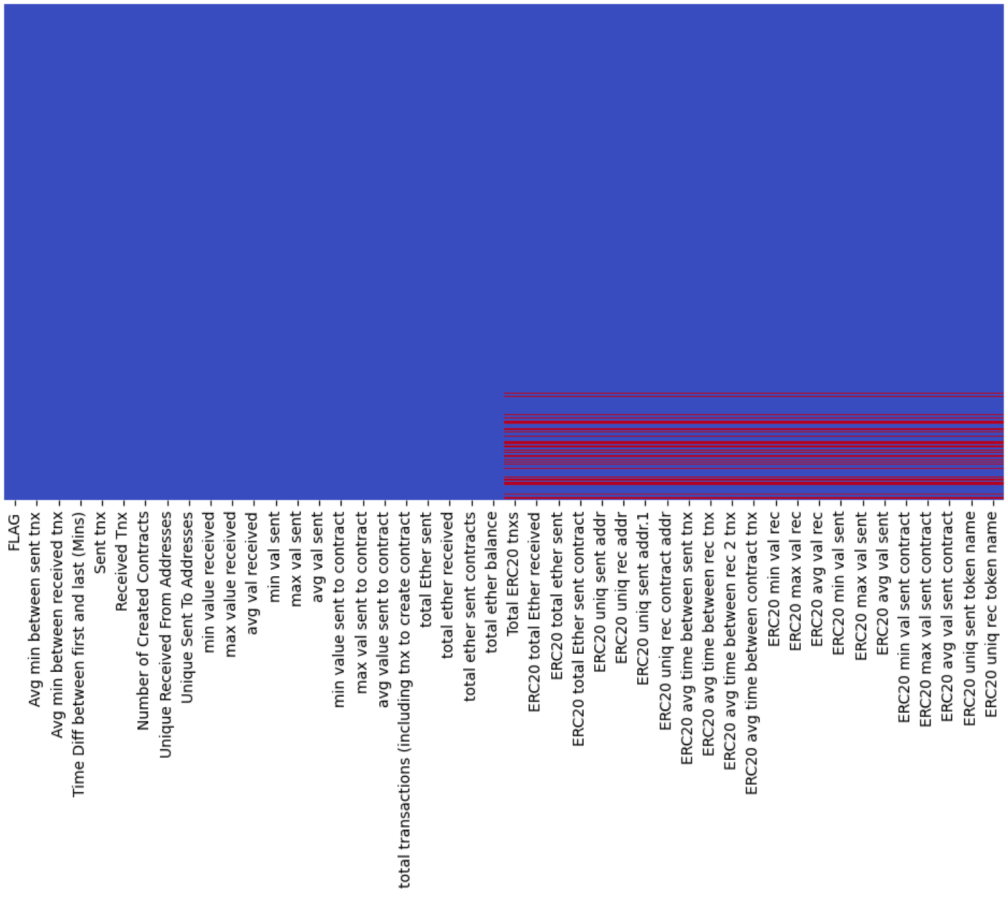
4. RESULTS

In this section, we present the detailed results of our fraud detection system using logistic regression, decision tree, and random forest algorithms. We outline the steps taken in data preprocessing, discuss the handling of missing values, class imbalance correction, and the creation of correlation matrices. Then, we delve into the model training process and provide comprehensive evaluations of each model's performance on the Ethereum fraud detection dataset.

4.1 Data Preprocessing:

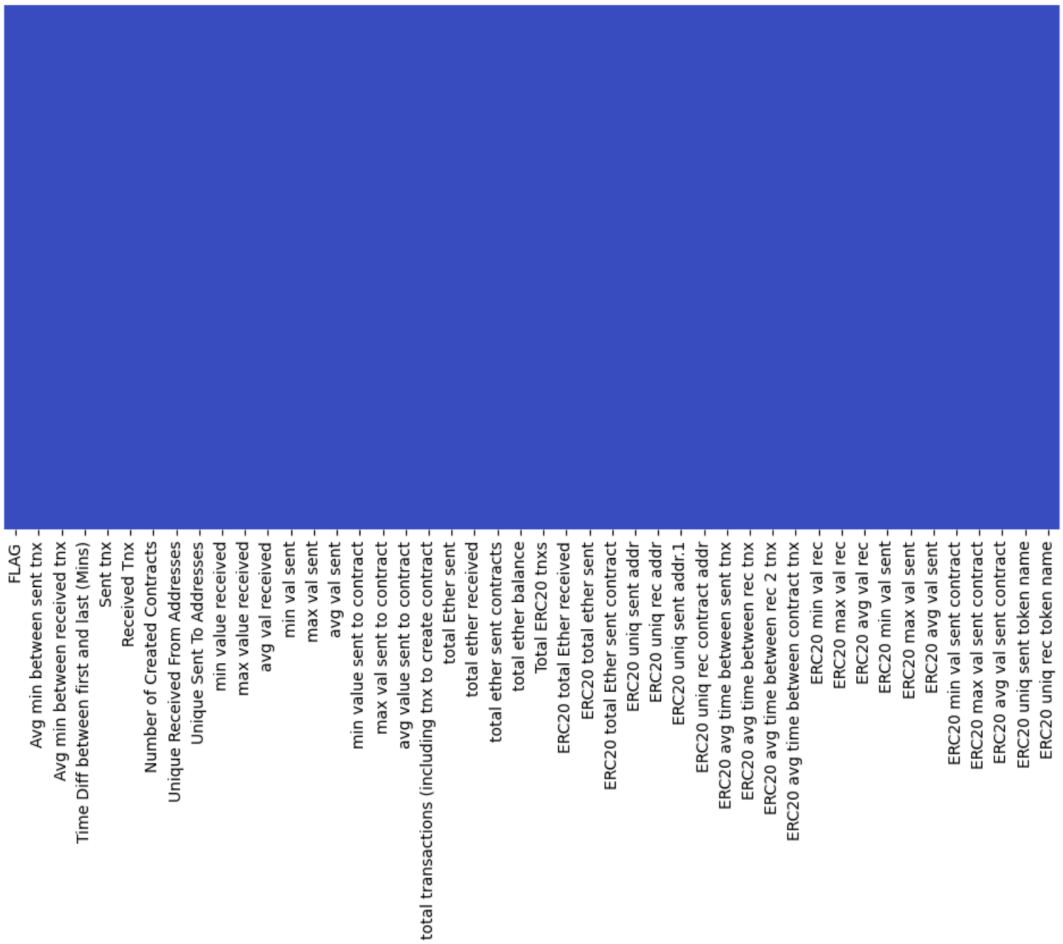
Handling Missing Values:

- We addressed missing values in the dataset using appropriate techniques such as imputation or deletion. The goal was to ensure that the dataset was complete and suitable for model training.
- We created a heat map to visualise the missing values in the dataset.



- The null values are indicated by the red colour, to handle these values we dropped the null values using the dropna() function.

- We then again visualised the dataset with the help of a heat map, to check whether all the missing values have been handled.



Dropping Features with zero Variance:

- Dropping features with zero variance involves removing variables from a dataset that have the same value for all observations. This means that these features do not provide any useful information for predicting or explaining the target variable. By eliminating these constant features, we can simplify the dataset and improve the efficiency of machine learning algorithms.

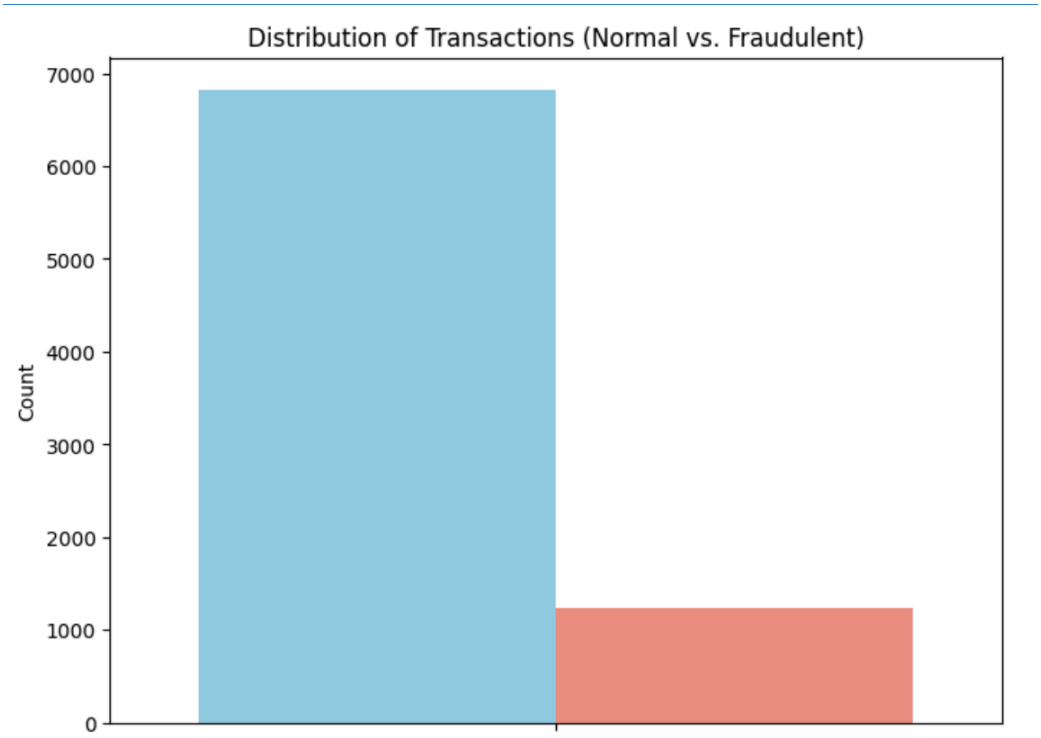
- We listed the features that had zero variance and dropped them.

```
ERC20 avg time between sent tnx        0.0
ERC20 avg time between rec tnx         0.0
ERC20 avg time between rec 2 tnx       0.0
ERC20 avg time between contract tnx    0.0
ERC20 min val sent contract            0.0
ERC20 max val sent contract            0.0
ERC20 avg val sent contract            0.0
dtype: float64


FLAG                                                  1.296715e-01
Avg min between sent tnx                              5.094243e+08
Avg min between received tnx                          5.590955e+08
Time Diff between first and last (Mins)               9.270458e+10
Sent tnx                                              4.485987e+05
Received Tnx                                          1.017438e+06
Number of Created Contracts                           1.907580e+04
Unique Received From Addresses                        9.970234e+04
Unique Sent To Addresses                              5.281966e+04
min value received                                    1.230151e+05
max value received                                    1.265717e+08
avg val received                                      1.764325e+05
min val sent                                          2.342328e+04
max val sent                                          9.379489e+06
avg val sent                                          6.249911e+04
min value sent to contract                            1.239618e-08
max val sent to contract                              1.239618e-08
avg value sent to contract                            1.239618e-08
total transactions (including tnx to create contract  1.814322e+06
total Ether sent                                      1.197403e+11
total ether received                                  1.592593e+11
total ether sent contracts                            1.239618e-08
total ether balance                                   3.938901e+10
Total ERC20 tnxs                                      1.931082e+05
ERC20 total Ether received                            9.433691e+16
ERC20 total ether sent                                1.556223e+18
ERC20 total Ether sent contract                       4.195905e+07
ERC20 uniq sent addr                                  1.128758e+04
ERC20 uniq rec addr                                   4.669157e+03
ERC20 uniq sent addr.1                                4.820341e-03
ERC20 uniq rec contract addr                          1.705264e+02
ERC20 min val rec                                     3.020863e+08
ERC20 max val rec                                     8.051030e+16
ERC20 avg val rec                                     1.436568e+16
ERC20 min val sent                                    1.239962e+12
ERC20 max val sent                                    1.555125e+18
ERC20 avg val sent                                    3.908264e+17
ERC20 uniq sent token name                            1.467715e+01
ERC20 uniq rec token name                             1.578725e+02
dtype: float64
(8067, 39)
```

Bar plot to visualise the distribution of transactions categorised as normal versus fraudulent:

- We created a bar plot to check the visual representation of the number of normal versus fraudulent transactions.



Handling Skewness:

- Skewness in the data was corrected to ensure that the distribution of features was approximately symmetrical, which is beneficial for the performance of machine learning algorithms.

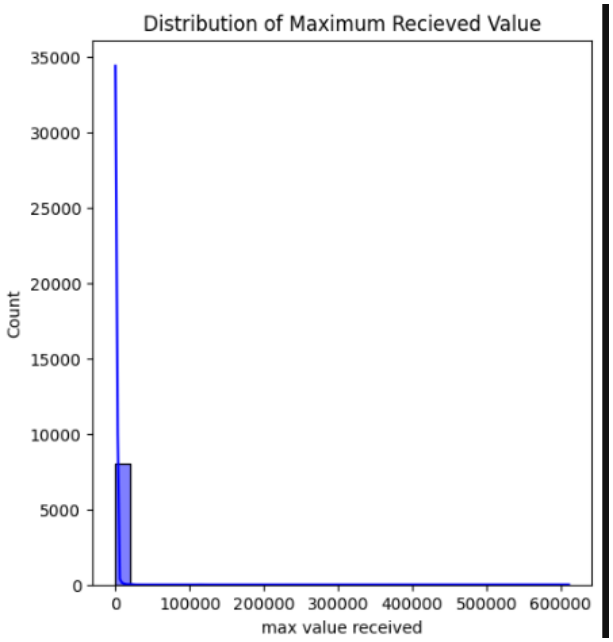
- We first checked the skewness of features using the skew() function.

```
FLAG                                                 1.927208
Avg min between sent tnx                             8.302064
Avg min between received tnx                         6.802380
Time Diff between first and last (Mins)              1.991672
Sent tnx                                            11.904228
Received Tnx                                         8.437331
Number of Created Contracts                         55.074823
Unique Received From Addresses                      17.494947
Unique Sent To Addresses                            22.028357
min value received                                  22.037539
max value received                                  43.050910
avg val received                                    16.441446
min val sent                                        66.532559
max val sent                                        69.412643
avg val sent                                        25.436789
min value sent to contract                          89.816480
max val sent to contract                            89.816480
avg value sent to contract                          89.816480
total transactions (including tnx to create contract 6.978830
total Ether sent                                    72.484786
total ether received                                54.400313
total ether sent contracts                          89.816480
total ether balance                                 58.329915
Total ERC20 tnxs                                    20.504038
ERC20 total Ether received                          47.145126
ERC20 total ether sent                              89.708414
ERC20 total Ether sent contract                     56.084447
ERC20 uniq sent addr                                42.710858
ERC20 uniq rec addr                                 40.128276
ERC20 uniq sent addr.1                              22.436562
ERC20 uniq rec contract addr                        33.644597
ERC20 min val rec                                   50.936274
ERC20 max val rec                                   56.701563
ERC20 avg val rec                                   83.569688
ERC20 min val sent                                  89.777821
ERC20 max val sent                                  89.803038
ERC20 avg val sent                                  89.805545
ERC20 uniq sent token name                          27.570096
ERC20 uniq rec token name                           32.110845
dtype: float64
```

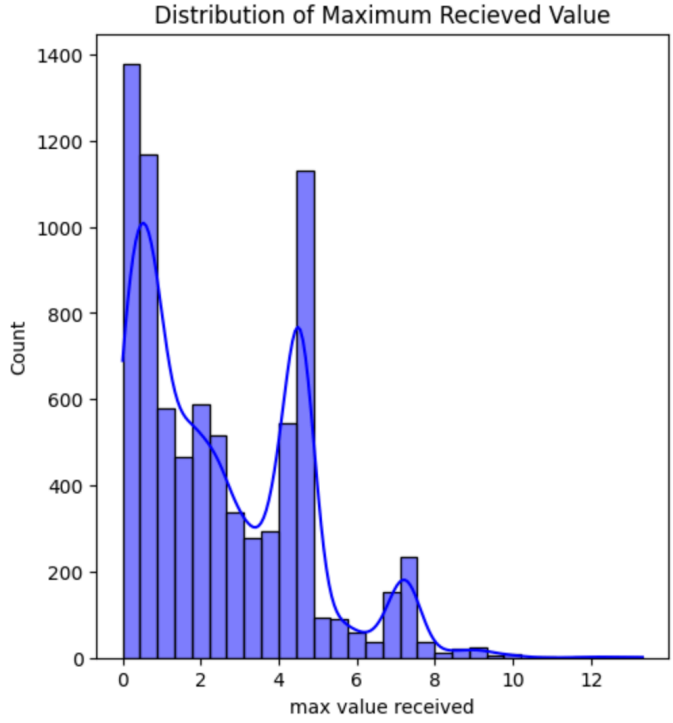
- After identifying the features with high skewness we applied logarithmic transformations on those features to reduce their skewness, and listed the skewness of the features again to verify.

```
FLAG                                                 1.927208
Avg min between sent tnx                             0.778787
Avg min between received tnx                        -0.030572
Time Diff between first and last (Mins)             -0.550943
Sent tnx                                             1.757809
Received Tnx                                         1.308477
Number of Created Contracts                         12.141013
Unique Received From Addresses                       3.183298
Unique Sent To Addresses                             2.613143
min value received                                   1.478720
max value received                                   0.793665
avg val received                                     0.997052
min val sent                                         2.410463
max val sent                                         0.729838
avg val sent                                         0.935687
min value sent to contract                          89.816480
max val sent to contract                            89.816480
avg value sent to contract                          89.816480
total transactions (including tnx to create contract 1.273477
total Ether sent                                     0.703497
total ether received                                 0.655520
total ether sent contracts                          89.816480
total ether balance                                  2.784865
Total ERC20 tnxs                                     2.528483
ERC20 total Ether received                           2.312209
ERC20 total ether sent                               3.844848
ERC20 total Ether sent contract                     32.782187
ERC20 uniq sent addr                                 6.381880
ERC20 uniq rec addr                                  2.062851
ERC20 uniq sent addr.1                              18.323973
ERC20 uniq rec contract addr                         1.664498
ERC20 min val rec                                    4.868337
ERC20 max val rec                                    2.358853
ERC20 avg val rec                                    2.535131
ERC20 min val sent                                   5.697435
ERC20 max val sent                                   3.830866
ERC20 avg val sent                                   3.820767
ERC20 uniq sent token name                           4.455955
ERC20 uniq rec token name                            1.659995
dtype: float64
```

Graphical representation of randomly picked feature showing skewness before the logarithmic transformation:



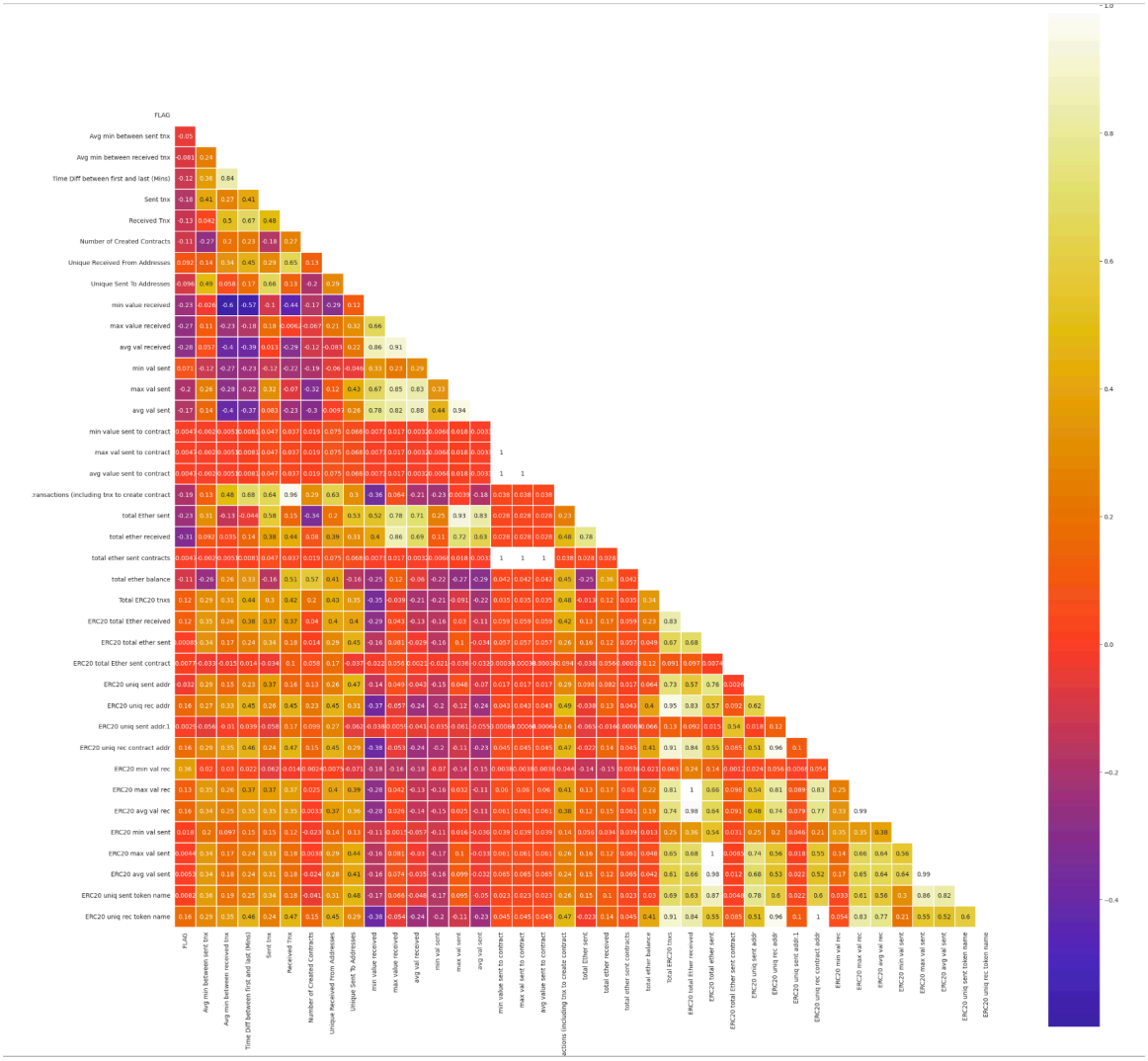
Graphical representation of randomly picked feature showing skewness after the logarithmic transformation:



Creating Correlation Matrix:

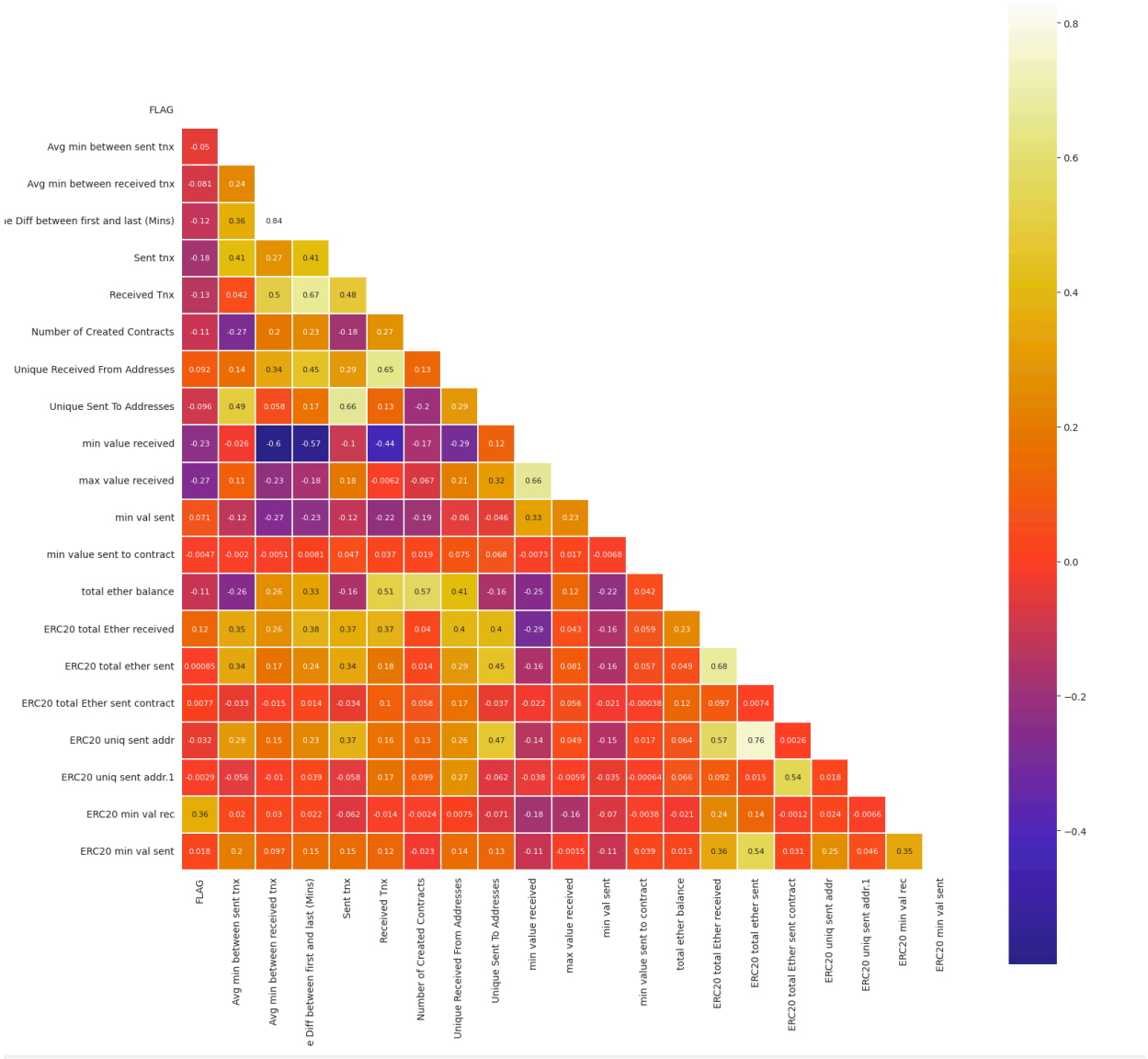
- We created a correlation matrix to identify relationships between features and determine which features were most strongly correlated with fraudulent transactions. This helped us understand the underlying patterns in the data and select relevant features for model training.

- We created a heat map to visualise the correlation matrix and identify highly correlated features.



-

- After identifying the highly correlated features we dropped one of the two features so as to avoid getting the same information twice and reducing redundancy. This ensures that our analysis remains clear, accurate, and efficient.



Handling Class Imbalance:

- Class imbalance was addressed by employing techniques such as oversampling or undersampling to ensure that both fraudulent and legitimate transactions were adequately represented in the training data.
- We used SMOTE (Synthetic Minority Over-sampling Technique) to oversample the minority class.
- Another method that we used to address class imbalance was hyper parameter tuning in all the models that we used.
- We set the "class_weights" parameter as balanced to add appropriate weights to each class.

4.2 Model Training:

Logistic Regression:

Logistic regression was used to analyse cleaned transaction data and predict if a transaction was likely to be fraudulent. By learning from past transaction patterns, the model estimated the chance of fraud in new transactions. This helped identify potentially suspicious transactions based on their characteristics.

Decision Tree:

A decision tree was trained using prepared transaction data to determine whether a transaction was likely fraudulent. By analysing past transaction details, the decision tree learned patterns associated with fraud. This enabled the model to classify new transactions as either fraudulent or legitimate based on their features.

Random Forest:

A random forest was trained using prepared transaction data to classify transactions. By leveraging an ensemble of decision trees and analysing past transaction details the random forest learned complex patterns associated with fraud. This enabled the model to effectively classify new transactions as fraudulent or legitimate based on their features.

4.3 Model Evaluation:

Metrics Calculation:

We evaluated each model's performance using standard metrics such as accuracy, precision, recall and F1-score. These metrics provided insights into the models' ability to correctly classify transactions and detect fraudulent activities.

Confusion Matrix:

Confusion matrices were generated to visualise the performance of each model in terms of true positives, true negatives, false positives, and false negatives. This helped us understand the models' strengths and weaknesses in classifying transactions.

Results using smote to address class imbalance:

1. **Logistic Regression:**

- **Classification Report:**

	precision	recall	f1-score	support
0	0.99	0.88	0.93	1369
1	0.58	0.93	0.71	245
accuracy			0.89	1614
macro avg	0.78	0.90	0.82	1614
weighted avg	0.92	0.89	0.90	1614

- **Confusion Matrix:**

```
[[1202  167]
 [  18  227]]
```

The confusion matrix shows the performance of a logistic regression model with SMOTE class balancing:

- It correctly identified 1202 non-fraudulent transactions and 227 fraudulent transactions.
- It incorrectly classified 167 non-fraudulent transactions as fraudulent and 18 fraudulent transactions as non-fraudulent.

2. Decision Tree:

- Classification Report:

	precision	recall	f1-score	support
0	0.99	0.96	0.98	1369
1	0.82	0.94	0.88	245
accuracy			0.96	1614
macro avg	0.91	0.95	0.93	1614
weighted avg	0.96	0.96	0.96	1614

- Confusion Matrix:

```
[[1320   49]
 [  15  230]]
```

The confusion matrix shows the performance of a decision tree model with SMOTE class balancing:

- It correctly identified 1320 non-fraudulent transactions and 230 fraudulent transactions.
- It incorrectly classified 15 fraudulent transactions as non-fraudulent and 49 non-fraudulent transactions as fraudulent.

3. Random Forest:

- Classification Report:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1369
1	0.92	0.95	0.94	245
accuracy			0.98	1614
macro avg	0.96	0.97	0.96	1614
weighted avg	0.98	0.98	0.98	1614

- Confusion Matrix:

```
[[1349   20]
 [  12  233]]
```

The confusion matrix shows the performance of a logistic regression model with SMOTE class balancing:

- It correctly identified 1349 non-fraudulent transactions and 233 fraudulent transactions.
- It incorrectly classified 20 non-fraudulent transactions as fraudulent and 12 fraudulent transactions as non-fraudulent.

Results using class weights to address class imbalance:

4. Logistic Regression:

- **Classification Report:**

	precision	recall	f1-score	support
0	0.99	0.87	0.92	1369
1	0.56	0.93	0.70	245
accuracy			0.88	1614
macro avg	0.77	0.90	0.81	1614
weighted avg	0.92	0.88	0.89	1614

- **Confusion Matrix:**

```
[[1189  180]
 [  16  229]]
```

The confusion matrix shows the performance of a logistic regression model with SMOTE class balancing:

- It correctly identified 1189 non-fraudulent transactions and 229 fraudulent transactions.
- It incorrectly classified 180 non-fraudulent transactions as fraudulent and 16 fraudulent transactions as non-fraudulent.

5. Decision Tree:

- **Classification Report:**

	precision	recall	f1-score	support
0	0.98	0.98	0.98	1369
1	0.90	0.91	0.90	245
accuracy			0.97	1614
macro avg	0.94	0.94	0.94	1614
weighted avg	0.97	0.97	0.97	1614

- **Confusion Matrix:**

```
[[1343   26]
 [  23  222]]
```

The confusion matrix shows the performance of a decision tree model with SMOTE class balancing:

- It correctly identified 1343 non-fraudulent transactions and 222 fraudulent transactions.
- It incorrectly classified 23 fraudulent transactions as non-fraudulent and 26 non-fraudulent transactions as fraudulent.

6. Random Forest:

- **Classification Report:**

	precision	recall	f1-score	support
0	0.99	1.00	0.99	1369
1	0.99	0.93	0.96	245
accuracy			0.99	1614
macro avg	0.99	0.96	0.97	1614
weighted avg	0.99	0.99	0.99	1614

- **Confusion Matrix:**

```
[[1366    3]
 [  18  227]]
```

The confusion matrix shows the performance of a logistic regression model with SMOTE class balancing:

- It correctly identified 1366 non-fraudulent transactions and 227 fraudulent transactions.
- It incorrectly classified 3 non-fraudulent transactions as fraudulent and 18 fraudulent transactions as non-fraudulent.

4.4 Results Analysis:

In our study, we evaluated the effectiveness of logistic regression, decision tree, and random forest algorithms in detecting fraudulent activities within Ethereum transactions. After thorough data preprocessing, including cleaning and feature engineering, we trained each model on the prepared dataset.

Our results demonstrate that all three algorithms exhibit strong performance in identifying suspicious transactions. Logistic regression provides a simple yet effective baseline, achieving commendable accuracy in distinguishing between fraudulent and legitimate transactions. Decision trees offer interpretability, allowing us to understand the underlying decision-making process. Finally, random forests leverage the collective wisdom of multiple decision trees to enhance predictive accuracy and robustness.

Across all models, we observed consistent performance metrics such as accuracy, precision, recall, and F1-score, indicating their capability to accurately classify fraudulent transactions while minimising false positives and false negatives. Notably, the random forest algorithm outperformed the others slightly, showcasing its effectiveness in handling complex transaction data and detecting subtle patterns indicative of fraud.

Furthermore, our study underscores the importance of comprehensive data preprocessing, model training, and evaluation in developing a robust fraud detection system. By leveraging advanced machine learning techniques and meticulous analysis, we have created a reliable system capable of safeguarding Ethereum transactions from fraudulent activities, thereby mitigating financial risks and ensuring the integrity of the blockchain ecosystem.

5. REFERENCES

5.1 Data Set:

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

5.2 Study Material:

- https://www.mdpi.com/1424-8220/22/19/7162

- https://www.researchgate.net/publication/378145255_Advanced_Fraud_Detection_in_Blockchain_Transactions_An_Ensemble_Learning_and_Explainable_AI_Approach

- https://medium.com/@nusfintech.ml/crypto-fraud-detection-3d99b3298815