Ethereum Blockchain Illicit Activity Detection & Mitigation

Prepared by - Ayushi Singhal, Landon Smith,
Anamika Medhi, Manaswini Mishra, Saddiq Jeelani
MS in Business Analytics
Saint Mary's College of California

Mentors - Dr. Navid Sabbaghi & Musa Ogunyemi
Dated - September 20th, 2023

Abstract

The recent exponential adoption of blockchain networks such as Ethereum have prompted regulators to introduce measures such as KYC and sanctions to help fend off malicious activity. This research paper builds upon the framework that blockchain networks must have robust systems in place to mitigate illicit activity if they are to be adopted for mainstream use. Our cohort sought to first replicate the highly effective machine learning models utilized by contemporary researchers in the field to differentiate between illicit and legal addresses, but also harness these methodologies into two main additions: real-time mempool transaction monitoring and multi-categorical classification of illicit addresses. While this research paper doesn't provide a viable solution to our ambitious goal of ushering in a transaction safety layer onto the decentralized Ethereum network, our research contradicts and exposes the lack of efficacy of models created using the aforementioned contemporary methodologies for the purpose of illicit activity detection. Our cohort believes that the segments of the Etherum community seeking to mitigate illicit activity using machine learning models may need to take a step backward and reassess the foundation of our models before additional progress in the field can be made.

Acknowledgements

We extend our sincere gratitude to Dr. Navid Sabbaghi, our dedicated advisor, for his unwavering support and invaluable guidance throughout the course of this project. Our mentor and CEO of AI Chain, Musa Ogunyemi, has also played a pivotal role in shaping the direction of this work, offering insightful feedback, and pushing us to excel beyond our boundaries. Their expertise and mentorship have not only enriched this project but have also enhanced our understanding of blockchain technology. We are deeply appreciative of the time and effort Professor Navid and mentor Musa invested in sharing their knowledge and providing invaluable insights that have significantly contributed to the success of this endeavor. We feel fortunate to have had the opportunity to work under their supervision and are immensely grateful for their belief in our abilities.

Table of Contents

Introduction

Why analyze transactions occurring over the Ethereum blockchain?

The rise in popularity of blockchain technology over recent years has opened the floodgates for new capital to begin flowing into blockchain protocols. While this tidal wave of new interest has included many genuinely interested in the benefits that blockchain technology has the potential to enable, this spike in engagement has also given malicious actors a larger audience to prey on. The heightened transaction volume and malicious activity now present on blockchain networks such as Ethereum has become too prevalent for regulators of traditional financial markets to dismiss any longer. Regulators have responded to the uptick in victims of malicious blockchain activity through the implementation of a wide variety of combative measures, such as the enforcement of both Know Your Customer (KYC) regulations and sanctions on the ecosystem's bad actors. Our cohort shares the regulators view that if blockchain networks are to achieve mainstream adoption, there must be anti-fraud measures in place to counteract rampant malicious activity.

What is a blockchain?

A blockchain can be described as a public database that is governed by the participating computers which host it. The “block” portion of the name refers to a chunk of updates that is appended to the current state of the database, while “chain” refers to the bonded nature of these chunks. The aggregation of all blocks in the chronological chain results in the entire history of the blockchain. Blockchain technology originated and perhaps best known for its role in the conception of the world largest cryptocurrency named Bitcoin in 2009.

What is Ethereum?

Ethereum builds on Bitcoin's innovation of blockchain technology, but layers additional technology on top. The decentralized physical computers (nodes) that host the Ethereum network determine the state of a virtual computer called the EVM (Ethereum Virtual Machine). Anyone who is a participant on the network can request that this computer perform arbitrary computation on their behalf. Ethereum is the world's second largest blockchain network by overall market cap, currently standing at $200 billion at time of writing.

Relevant Ethereum Infrastructure

What is Ether?

The purpose of the Ether cryptocurrency is to usher in a market for computation on the EVM (Ethereum virtual Machine) and provide an economic incentive for participants to provide computational resources to the network. Ether is divisible up to 18 decimal places (10^{-18}), with its smallest units denoted as Wei. The Ethereum community refers to the Ether payments made to nodes to facilitate arbitrary computation and verify the legitimacy of transactions as "gas". These gas payments are usually quoted in Gwei, which is equivalent to one billionth of an Ether token (10^{-9}). We take care to communicate to our readers what denomination of Ether we utilize throughout this research paper.

What are addresses?

The two types of addresses that exist on the Ethereum blockchain are known as externally owned accounts (EOA) and contract accounts. An EOA account costs nothing to create, can

initiate transactions between itself and other addresses, and is controlled via a public and private key pair. Contract accounts differ in the capacity that they cost Ether to create due to the necessity for network storage costs, and that they can only initiate a transaction in response to receiving a transaction. Additionally, contract accounts have a public key that can be interacted with by other accounts but are not controlled by private keys. Instead, a contract account is controlled by the logic that is defined in their smart-contract code. The commonalities between these account types consist of the ability for both types of accounts to receive, hold, and send Ether and other tokens that adhere to token formats supported by Ethereum. Both account types can also interact with smart-contracts hosted on the Ethereum network, meaning smart-contracts have the capability of interacting with each other just as EOA accounts utilized by humans can. All addresses are denoted by a unique 42-character hexadecimal, an example being 0x06012c8cf97bead5deae237070f9587f8e7a266d.

What are transactions?

Transactions change the state of the EVM and require inclusion in a validated block to be confirmed on the blockchain. Transactions require a gas fee to be paid for the computing resources needed for validators to assess the validity of the transactions and append them to their proposed blocks. Transactions are made up of many different fields of data, such as from, to, value, nonce, input data, gas, and others. Transactions can also be uniquely identified using a hexadecimal hash, however, in this case, the associated hash is 66 characters in length.

What is a consensus mechanism?

As we have discussed in prior sections, the Ethereum network consists of a decentralized network of physical computers (nodes) that agree on a single true state of the EVM. The decentralized nature of these nodes necessitates a protocol that allows these separate actors to coordinate and come to agreement on the state of the Ethereum blockchain, called a consensus mechanism. While the Ethereum network utilizes a consensus mechanism called Proof of Stake at time of writing, Ethereum originally shared the same consensus mechanism as used in the Bitcoin blockchain titled Proof of Work until September 15th, 2022. The Proof of Stake consensus mechanism requires nodes who want to take part in network consensus (validators) to stake native Ether currency as collateral. These validators are then responsible for attesting to newly proposed blocks or creating them when selected to be a block proposer. However, every validator isn't responsible for voting on every block. Instead, validators are divided into committees to conduct their attestation duties. If a proposed block is determined to be valid by the nodes controlling 2/3rds of all staked Ether within their assigned blocks committee, the block is finalized onto the blockchain. Validators are rewarded Ether for attesting to a block that is congruent with the majority of other validator nodes and proposing a new block themselves. However, validators can also suffer penalties for inactivity, lateness, and incorrectness on their attestations. Additionally, validators can have their Ether stake slashed and be forced to leave the network if the majority of other nodes deem them to be acting maliciously.

What is the mempool?

Before transactions are able to be included into a block by validators, they are broadcast to a nearby Ethereum node and added to the node's local mempool. The mempool is the place where pending transactions that have not yet been added to a proposed block reside. As block proposers sift through the mempool looking for transactions to add to their proposed blocks, transactions that include higher gas fees are more likely to be chosen first due to validators' economic incentive to seek out transactions that offer them the highest reward.

What is MEV Boost?

MEV Boost is a technology that allows block proposers to outsource their block building duties. The acronym MEV (maximum extractable value) refers to the differences in rewards block proposers receive based on transaction composition and ordering within each block. Validators can choose which relays they would like to accept produced blocks from when they are selected to act as a block proposer. While MEV Boost is not native to the Ethereum protocol, its usage has proliferated amongst validator nodes with usage rates around 90% at time of writing.

Contemporary Research

Exploring the prior research done in the area of illicit activity detection on Ethereum revealed many studies where authors were able to create highly effective machine learning models that could distinguish between illicit and legal account activity. A 2020 study authored

by Steven Farrugia reported achieving a 96.3% accuracy and 96% F1-Score utilizing a XGBoost model, while a similar study by Lavina Pahuja and Ahmad Kamal boasted a 99.2 % accuracy and 99% F1-Score using a LightGBM model. A final referenced study by Ismail Alarab and Simant Prakoonwit showcased a 98.91% accuracy and 97.60% F1-Score.

Study Author	Model Type	Accuracy	F1-Score
Farrugia	XGBoost	96.3%	96%
Pahuja & Kamal	LightGBM	99.2%	99%
Alarab & Prakoonwit	XGBoost	98.91%	97.60%

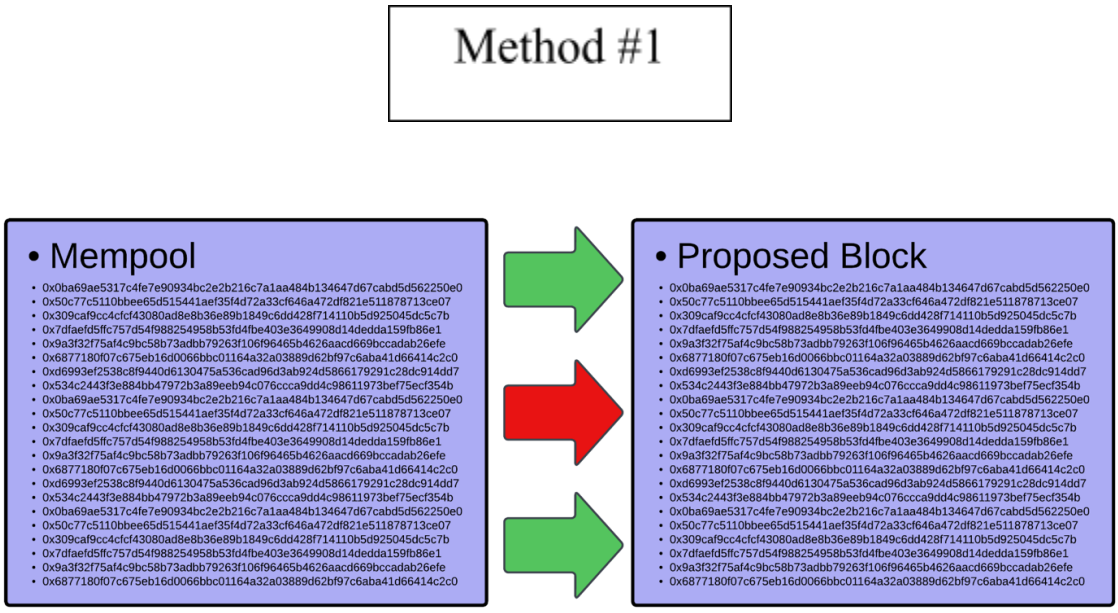
Given that prior research in the area of illicit activity detection on Ethereum advertised such highly successful results, our cohort decided that we wanted to first replicate the results of these aforementioned studies and subsequently further implement their methodologies in two separate exploratory areas. The first direction our cohort explored was real-time mempool transaction classification and mitigation using machine learning, with the second being multi-categorization classification of illicit addresses.

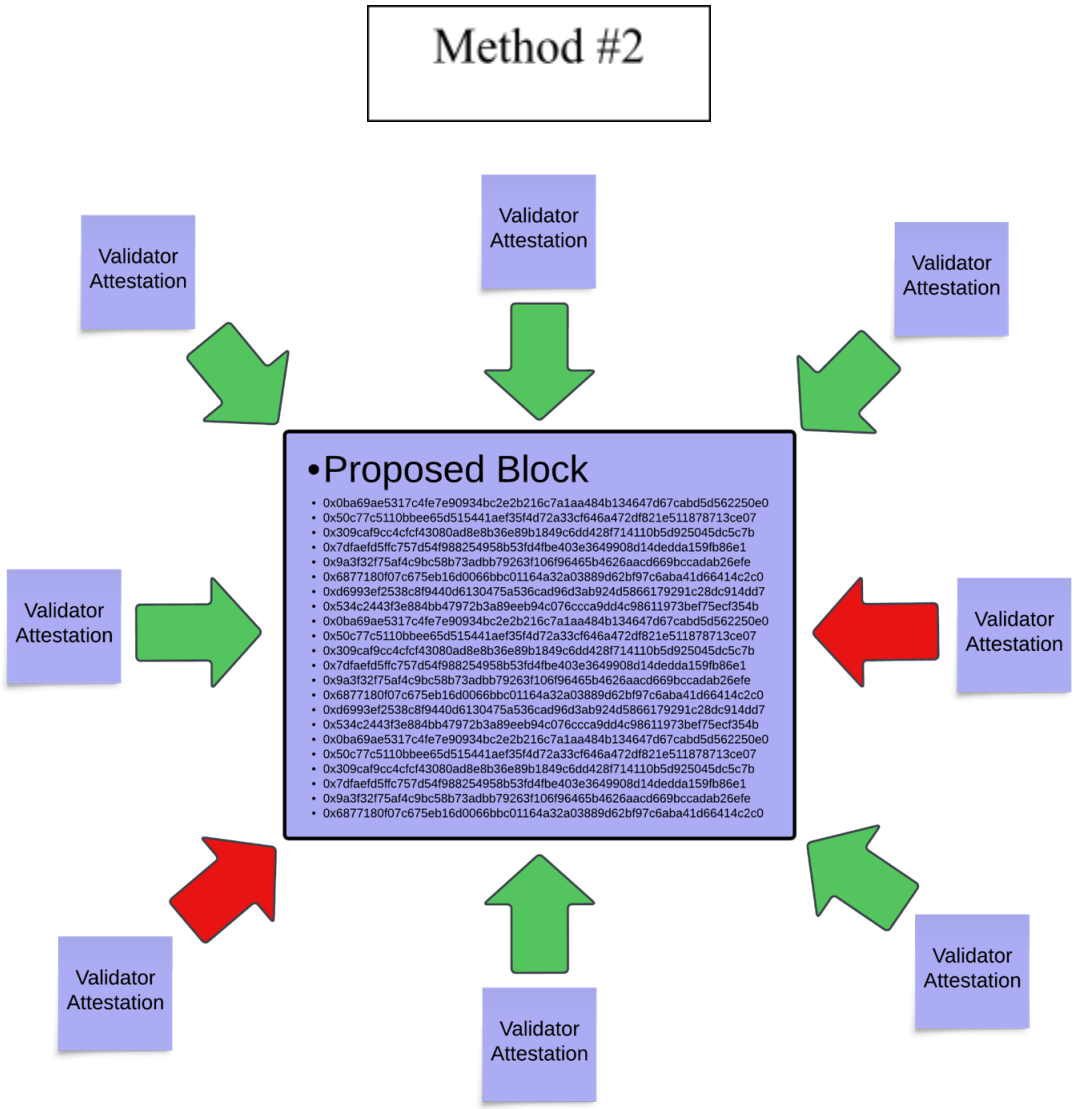
Real-Time Mempool Transaction Monitoring

Deployment Methods

Given that contemporary research already offered us an effective way to conduct illicit activity monitoring on the Ethereum, the logical next step for our cohort was to utilize such a

model by build an infrastructure that could be used to deploy a similar machine learning model on real-time mempool transactions waiting to be processed on the Ethereum blockchain. This capability would give machine learning models the ability to be used in a preventative manner, stopping transactions implicating illicit addresses from being finalized on the blockchain in the first place. Our cohort foresees a future where an anti-fraud system could be integrated into the Ethereum transaction lifecycle, with two main methods of deployment able to be utilized in tandem with one another. The first potential method of deployment would be to enable block proposers to utilize a machine learning model to avoid selecting transactions from the mempool that implicate illicit addresses, while the second deployment method would enable the classification of the entirety of a proposed block at once, allowing validators to avoid attesting to blocks that contain fraudulent transactions.





Adoption Scenarios

We believe that this anti-fraud system could be adopted by a significant portion of Ethereum nodes in two possible progressions. Firstly, we believe that there is a strong chance of adoption of an anti-fraud system amongst nodes within certain jurisdictions through regulatory enforcement. In this scenario, nodes would have no choice but to implement an architecture that would be capable of flagging and mitigating illicit activity before it is finalized in order to remain operational in the jurisdiction that has passed such regulation. However, despite the decentralized nature of nodes, our cohort also sees the ability for an anti-fraud transaction

monitoring system to be adopted emergently through a grassroots movement. The introduction of even a small number of nodes to the network who utilize a system to block address activity deemed illicit would place economic pressure on other nodes to follow the same standard. Due to block proposers receiving Ether rewards for their blocks successfully being confirmed on the blockchain after acquiring two-thirds of the votes within the validator committee assigned to their slot, block proposers are forced to make game theoretical decisions in order to achieve the highest possible probability of them successfully receiving their reward. If block proposers have knowledge that even a small percentage of overall validators are running anti-fraud systems in tandem with their nodes, block proposers will be forced to run the same system to ensure that they are not including illicit mempool transactions within their proposed blocks due to the risk that other validators will decline to attest to proposed blocks comprised of illicit transactions. This risk is amplified by the possibility that a block proposer receives a validator committee with a higher concentration of nodes running this software than the overall percentage.

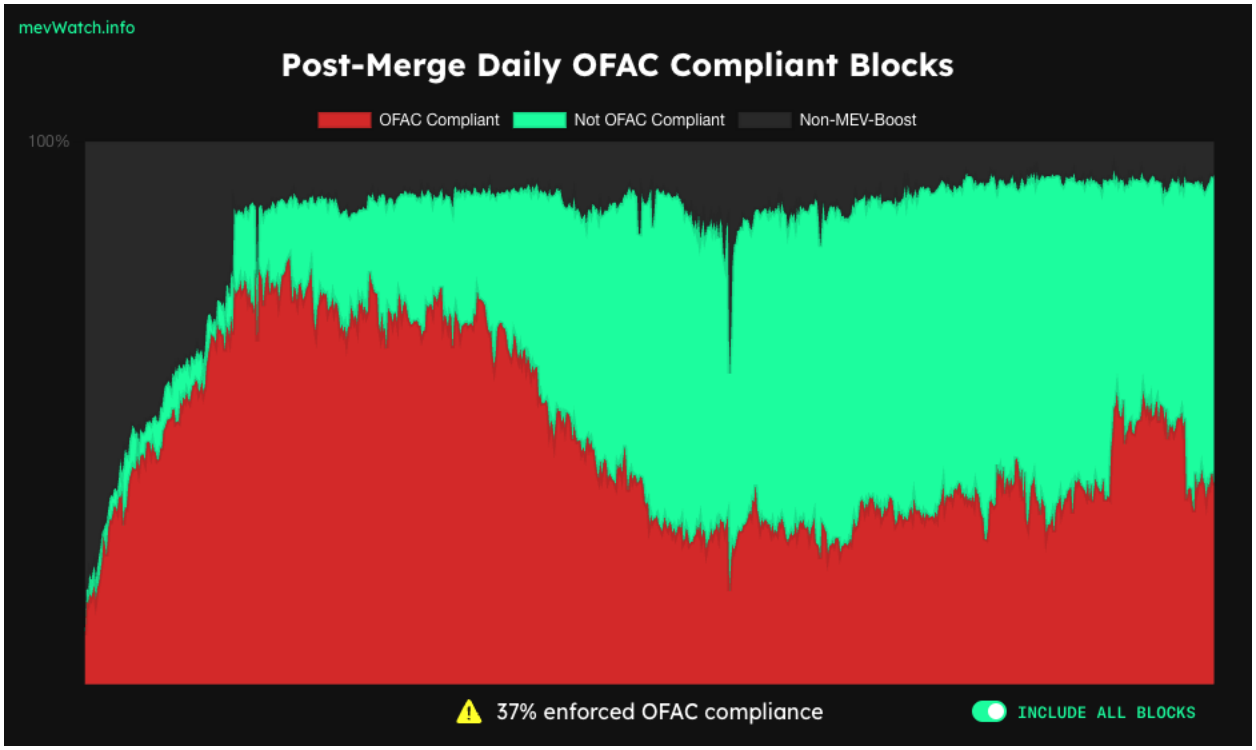
Censorship Stigma

Despite our good intentions and those of the authors of contemporary research papers on the topic of Ethereum illicit activity detection, our cohort foresees potential adoption issues due to the perception that certain ecosystem actors may hold pertaining to the censorship of illicit transactions. The ethos of the cryptocurrency space at large is one that is against censorship in any form, with many proponents in favor of open and permissionless protocols which are apathetic to questions of morality. This potential roadblock to adoption drove our cohort to seek out historical precedent for validator nodes accepting any form of transaction censorship in Ethereum's past.

Illicit Transaction Censorship Precedent: Tornado Cash

Our cohort believes that validators' reaction to the sanctions placed on Ethereum smart-contract Tornado Cash by the Office of Foreign Assets Control (OFAC) are evidence that widespread adoption of an anti-fraud system is possible across a decentralized blockchain like Ethereum. The Tornado Cash smart-contract functioned as an autonomous cryptocurrency mixer service. Due to the nature of blockchain technology, all transactions performed are publicly visible. While this feature of blockchain allows for greater transparency than traditional financial systems, the privacy of users is degraded. The intended goal of a cryptocurrency mixer service is to ensure privacy for users by obfuscating the origin of funds. Unfortunately, this capability makes crypto mixer services attractive conduits for illicit activity such as money laundering as well. Despite 80% of the activity on Tornado Cash being legitimate according to blockchain analytics company Elliptic, the smart-contract was blanket sanctioned for allowing the Lazarus Group, who are a North Korean state-sponsored hacking group, to launder over $455 million worth of stolen crypto at the time of the theft. The addition of Tornado Cash to the SDN list signaled a stark pivot from OFAC's prior enforcement actions. While OFAC typically designates people or organizations as being illegal for US citizens to transact with, the addition of Tornado Cash marked the first time that a decentralized and autonomous entity was added to the list. Due to Tornado Cash being autonomous smart-contract code in its simplest form, OFAC specified that US citizens are no longer allowed to interact or facilitate interaction with Tornado Cash associated Ethereum addresses in any way, not physical people or organizations. Theoretically, this decree is also applicable to validator nodes on the Ethereum network due to their control over the transactions that are finalized as part of the blockchains history.

The response of validator nodes to OFAC's sanctions on Tornado Cash is evidence that transaction censorship can indeed occur on the Ethereum blockchain despite its decentralized governance. As mentioned in the section describing relevant Ethereum infrastructure, MEV Boost relays exist as a way for block proposers to outsource the production of their blocks to 3rd party relays. Once the block production process is outsourced, individual MEV Boost relays have the ability to include or exclude transactions from a proposed block based on their own arbitrary set of rules. In the aftermath of the Tornado Cash sanctions, MEV Boost relays had to decide for themselves whether or not they would enforce the sanctions on blocks that they were given the opportunity to produce by validators. On the following chart taken from the MEV Watch website, we can see the percentage of blocks built by validator nodes that have outsourced their block building responsibilities to OFAC compliant MEV Boost relays since the Merge presented in the color red. This percentage has fluctuated greatly over time, but we can see that the majority of the blocks finalized on the Ethereum blockchain were OFAC compliant over the four-month period of October 2022 to February 2023, making Ethereum a primarily censored blockchain. While the percentage of compliant blocks has since dipped below 50%, there is still a large contingency of validators who continue to run MEV Boost relays which censor Tornado Cash transactions from their blocks. Our cohort believes that this is evidence that despite Ethereum's decentralized structure, there are external centralizing forces such as MEV Boost relays through which the effects of regulation can still be felt. We believe that this analysis of validator behavior bodes well for our intention to usher in an illicit transaction mitigation mechanism to the Ethereum ecosystem.



Once our cohort observed prior precedent for systems enabling illicit transaction mitigation to be adopted on Ethereum's decentralized node structure, we began to build our own solution. This report outlines the process of acquiring addresses, categorizing them into illicit and legal accounts, and extracting features to create two distinct datasets: the Coinbase dataset and the Random Sample dataset. Address acquisition involves sourcing addresses from various repositories and platforms, while feature engineering encompasses extracting relevant features using the Etherscan API.

Address Acquisition

Sourcing Illicit Addresses

Illegal addresses were gathered from multiple sources, including Kaggle, Github, Etherscan, Chainabuse, CryptoScam, and OFAC. These sources provided datasets where particular addresses were already flagged as illicit, either by Ethereum community members or regulatory agencies. Kaggle and Github provided datasets used in other academic studies pertaining to the field (Fraud Detection in Blockchains). The following datasets were considered for initial analysis from these sources:

- Farrugia Dataset
- Aliyev Dataset
- Escobero Dataset

Further, the dataset was enriched with additional illicit accounts reported on various fraud tracking websites which are as follows:

- OFAC
- EtherScamDB
- Chainabuse
- Etherscan

As multiple sources might contain the same illicit accounts reported, these potential duplicates were searched for and removed. The final number of accumulated illicit accounts that were flagged as 1 in the final dataset was 11,378.

Establishing Legal Accounts

Establishing a legal framework within the blockchain and cryptocurrency space presents several unique challenges due to the decentralized and pseudonymous nature of the technology. Here's an elaboration on the difficulties in creating a legal framework for blockchain:

Pseudonymity and Anonymity

- Pseudonymity: Blockchain transactions are associated with cryptographic addresses, not personal identities. While transactions are recorded on a public ledger, the real-world identities behind these addresses are often masked or pseudonymous.
- Anonymity Tools: Various technologies and tools (e.g., mixers, privacy coins) allow users to enhance the privacy of their transactions, making it challenging to link blockchain activities to real-world individuals.

Cross-Border Nature of Blockchain

- Global Reach: Blockchain operates on a global scale, transcending jurisdictional boundaries. Legal frameworks typically vary from one country to another, making it difficult to create a uniform set of rules governing blockchain use.
- Jurisdictional Conflict: Determining which country's laws apply to blockchain transactions involving parties from multiple jurisdictions can lead to legal uncertainty and conflicts.

Lack of Standardization

- Interoperability Challenges: Blockchains and cryptocurrencies are often built using different technologies and standards. The lack of interoperability hampers the development of a unified legal framework applicable to all blockchain platforms.

- Smart Contract Legality: Determining the legal status and enforceability of smart contracts, a fundamental aspect of blockchain, remains a challenge due to their varying complexities and potential legal implications.

Security and Fraud Concerns

- Fraud and Scams: Blockchain networks can be susceptible to various types of fraud, including Ponzi schemes, phishing attacks, and ICO (Initial Coin Offering) scams, necessitating robust legal measures to protect users.
- Security Breaches: Despite being considered secure, blockchain platforms can experience security breaches, leading to significant financial losses and legal disputes.

Technology Advancement and Innovation

- Rapid Technological Evolution: The rapid advancement of blockchain and related technologies like DeFi (Decentralized Finance) and NFTs (Non-Fungible Tokens) often challenges traditional legal paradigms, necessitating continuous legal updates and adaptations.
- Emerging Legal Issues: New legal challenges arise as blockchain is applied to novel use cases, such as DAOs (Decentralized Autonomous Organizations), AI on the blockchain, and more.

Despite all of the aforementioned challenges associated with defining what constitutes legal activity, our cohort settled on the usage of addresses that had interacted with Coinbase, a prominent digital platform for cryptocurrency exchange. This decision was due to their implementation of a vital legal measure known as Know Your Customer (KYC). This process involves verifying the identity of users by collecting personal information before allowing them

to engage in transactions and activities on the platform. By enforcing KYC compliance, Coinbase ensures that users are identifiable and associated with their respective cryptocurrency addresses. When a user interacts with Coinbase and undergoes the KYC process, the resulting tagged addresses are considered legal within the cryptocurrency space. Leveraging Etherscans address tagging, any addresses that were determined to have interacted with an address tagged as being Coinbases own wallets were identified by iterating through blocks ranging from 17,995,000 to 18,000,000 using the Etherscan API. This set of addresses was utilized to represent our legal accounts within our dataset. Additionally, our cohort decided to create another dataset where the legal addresses utilized were instead randomly selected addresses. This was done with the goal of performing comparative analysis to derive greater insights into our data. Within this dataset, our only criteria for legality was cross-checking our illicit addresses to ensure that none of our randomly sampled addresses were present. These addresses were also collected by iterating through the block range of 17,995,000 to 18,000,000.

Dataset Creation

Once we had gathered our legal Coinbase addresses, legal randomly sampled addresses, and illicit addresses, two distinct datasets were formed: the Coinbase dataset containing addresses verified by Coinbase and our illicit addresses, and the Random Sample dataset consisting of randomly selected addresses and our illicit addresses.

Feature Engineering

Utilizing the Etherscan API

To extract features for the accumulated addresses, the Etherscan API was utilized. The Etherscan API is a widely used tool in the blockchain analytics domain. Leveraging the Etherscan API, aggregate transaction information pertaining to the addresses was gathered. The first query made to the API garnered the current Ether balance associated with each address, which provided a snapshot of their financial standing at block 18,000,000. Furthermore, a query to extract normal transactions, which encompass simple transfers of Ether between addresses, shed light on transactional histories and patterns. Additionally, ERC20 transactions, which involve the transfer of fungible tokens built on the Ethereum blockchain, were also queried and recorded. This transaction data is a fundamental way of understanding the utilization and movement of various tokens within the Ethereum network among illicit and legal actors.

Custom Feature Extraction

By systematically collecting and aggregating this data through the Etherscan API, a rich set of features was curated, forming the foundation for subsequent analysis and modeling. Custom functions were developed to extract 22 relevant features from the acquired transaction data. The features extracted were subsequently combined with their respective addresses to create our finalized Coinbase dataset and Random Sample dataset. Below are the 22 extracted features for both the datasets:

Feature	Description	Data Type
Sent_tnx	The # of transactions sent by the address	Integer
Recieved_tnx	The # of transactions received by the address	Integer
Total_Transactions	The total number of transactions the address has either sent or received	Integer
Total_Ether_Balance	The current balance of the address in Ether (Block 18,000,000)	Float
Max_Value_Received	The maximum amount of Ether received in a single transaction	Float
Min_Value_Received	The minimum amount of Ether received in a single transaction	Float
Total_Ether_Received	The total amount of Ether received by the address across all address transactions	Float
Max_Value_Sent	The maximum amount of Ether sent in a single transaction	Float
Min_Value_Sent	The minimum amount of Ether sent in a single transaction	Float
Total_Ether_Sent	The total amount of Ether sent by the address across all address transactions	Float
Avg_Value_Received	The average Ether value received across all address transactions	Float
Avg_Value_Sent	The average Ether value sent across all address transactions	Float
Time_Delta_First_Last	The difference in time between an addresses first and last transaction in minutes	Float
Avg_min_between_received_tnx	The average number of minutes between an addresses received transactions	Float
Avg_min_between_sent_tnx	The average number of minutes between an addresses sent transactions	Float
Unique_Received_From_Addresses	The number of unique addresses the address received Ether from	Integer

Unique_Sent_To_Addresses	The number of unique addresses the address sent Ether to	Integer
Total_ERC20_Tnxs	The total number of ERC20 transactions the address has either sent or received	Integer
ERC20_Avg_Time_Between_Rec_Tnx	The average number of minutes between an addresses received ERC20 transactions	Float
ERC20_Avg_Time_Between_Sent_Tnx	The average number of minutes between an addresses sent ERC20 transactions	Float
ERC20_Uniq_Rec_Addr	The number of unique addresses the address received ERC20 tokens from	Integer
ERC20_Uniq_Sent_Addr	The number of unique addresses the address sent ERC20 tokens to	Integer

Exploratory Data Analysis

In this section, we conduct a comparative analysis of the two fully engineered Ethereum datasets: the Coinbase dataset and a Random Sample dataset. The aim is to understand the distribution and transactional behaviors of legal and illicit accounts within these datasets. Furthermore, we explore the importance of various features in understanding fraudulent activities. The report also introduces an interactive dashboard for comprehensive analysis and evaluation of account activities.

Dataset Overview

The Coinbase dataset and the Random Sample dataset are briefly described. The Coinbase dataset is noted for its balanced distribution, with approximately 49.5% licit accounts and 50.5% illicit accounts. Conversely, the Random Sample dataset exhibits a higher proportion of non-illicit accounts, indicating a relatively balanced distribution.

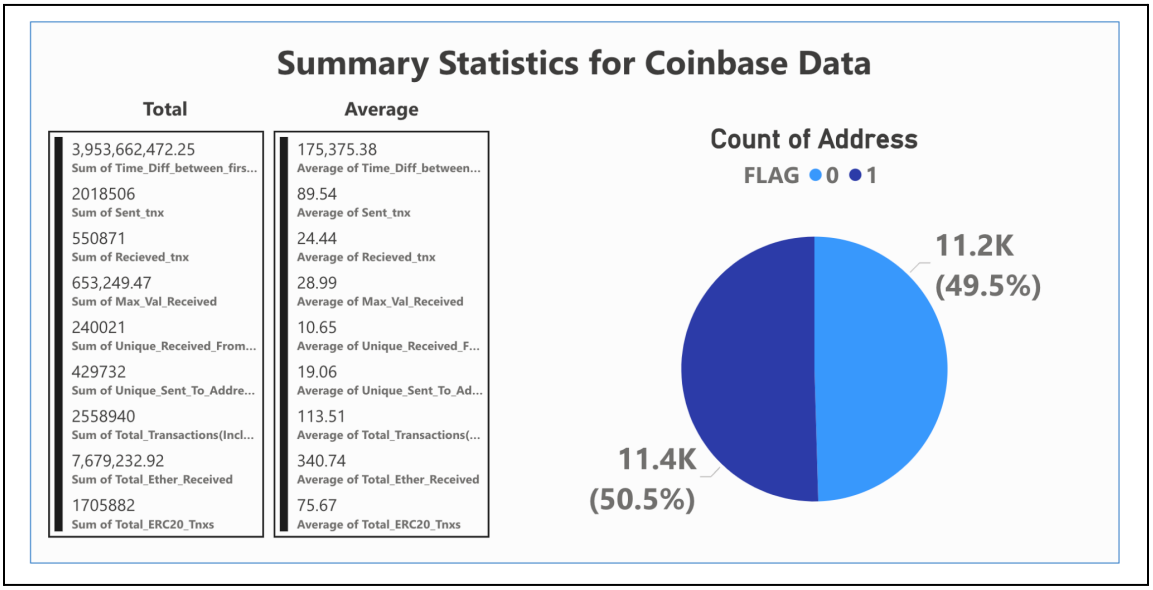


Fig 1. Summary Statistics of the Coinbase Dataset

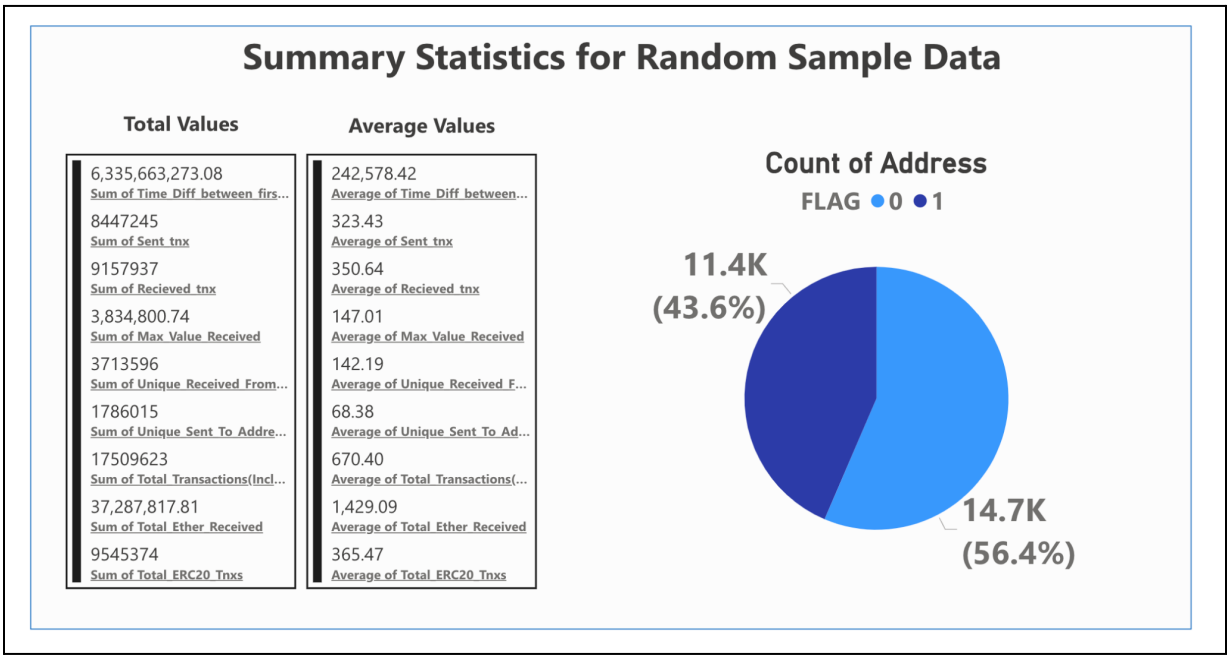


Fig 2. Summary Statistics of the Random Sample Dataset

Distribution Analysis

Pie Chart Analysis

Pie charts visually represent the distribution of illicit and licit accounts in both datasets. In the Coinbase dataset, the distribution is almost evenly split, while the Random Sample dataset shows a higher percentage of non-illicit accounts.

Transaction Analysis

Analysis of received and sent transactions among illicit and licit accounts reveals a significant number of received transactions in illicit accounts, suggesting a prevalence of scam

accounts. A lower count of sent transactions is observed, potentially linked to money laundering or related activities. This trend is consistent across both datasets.

Unique Address Analysis

Comparison of unique received from addresses and sent to addresses shows similar distribution patterns among both categories (licit and illicit) in both datasets. Further analysis is focused on received transactions.

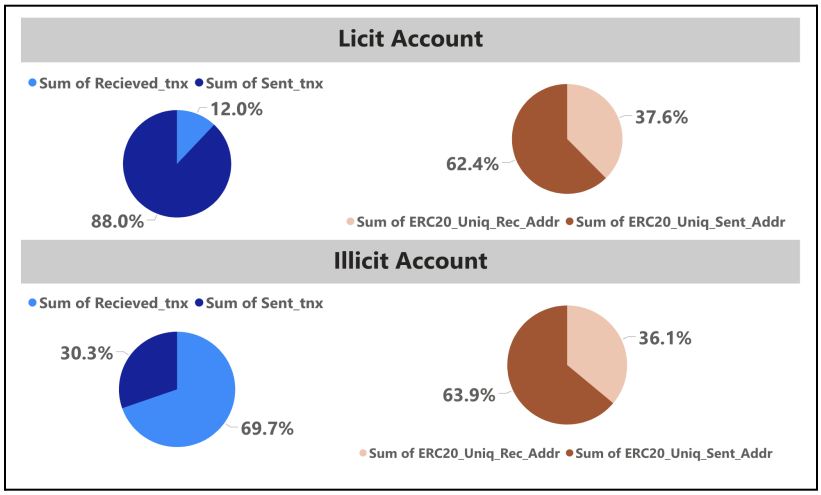


Fig. 3. Pie Chart of Received transactions and sent transaction and unique received from address and sent to addresses categorized by illicit and licit accounts for Coinbase Dataset.

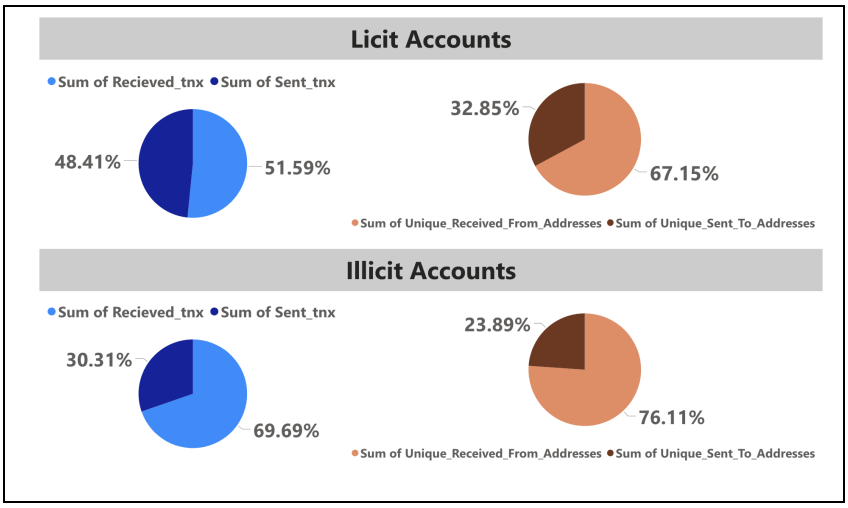
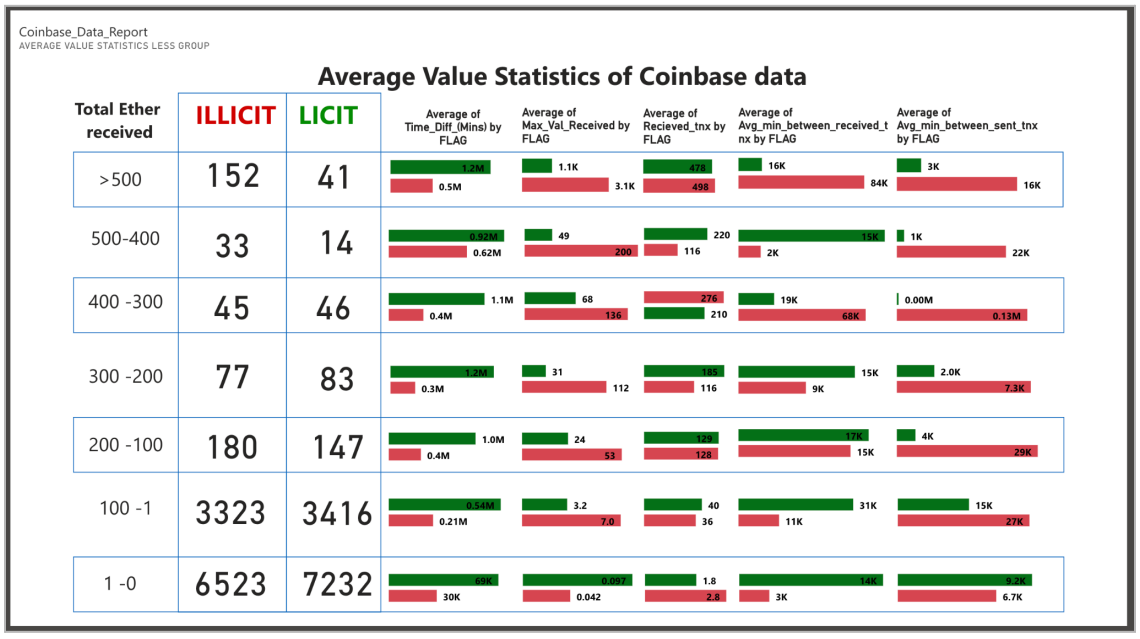


Fig. 4. Pie Chart of Received transactions and sent transaction and unique received from address and sent to addresses categorized by illicit and licit accounts for Randomly sampled Dataset.

Detailed Statistics and Feature Importance Visualization

Detailed statistics involve categorizing the total ether received by accounts into bins and computing average feature values for both illicit and licit accounts. This analysis sheds light on feature importance and their distribution across different transaction bins.



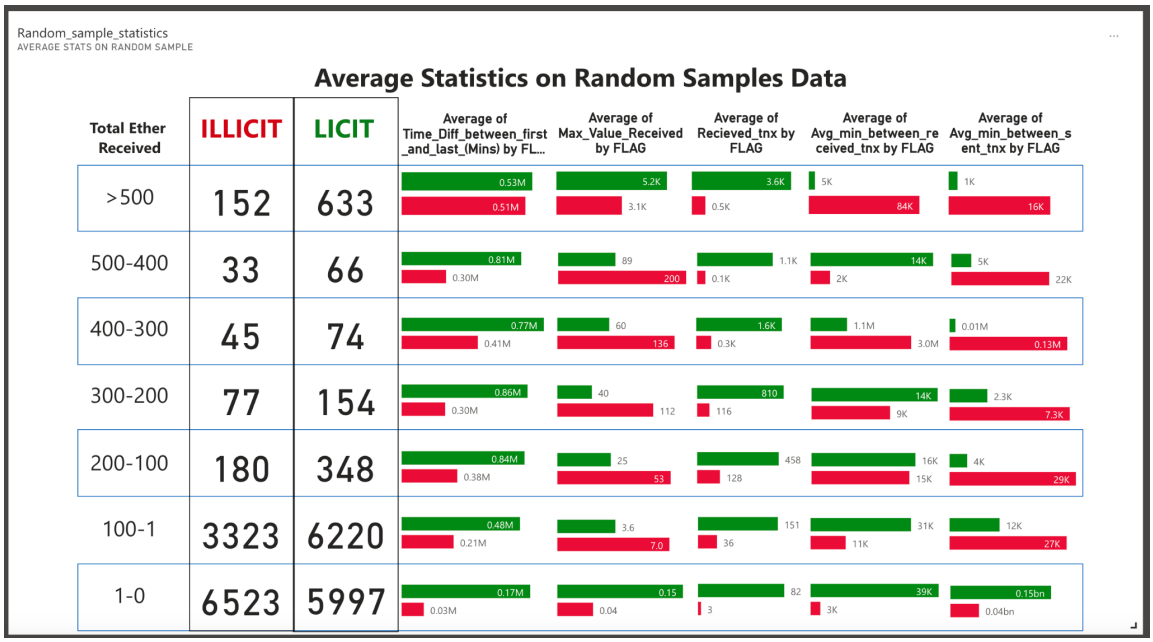


Fig: 5. Average Value Statistics for Coinbase and Randomly sampled data.

In Figure 5, we display the average distribution of 5 features across the bins. It is clear that the distribution of illicit and licit accounts in these bins are easily distinguishable. The average time difference between the first and last transaction value is much higher for licit accounts than that of illicit accounts. These observations are the same for both the datasets. However, the same cannot be seen in the case of the average maximum value received, in which the value for illicit accounts is higher than that of licit accounts. The observation of average value of average minutes between sent and received transactions is similar where the value of illicit accounts are higher than that of licit accounts. This analysis paved the way in identification of important features and evaluated their importance in identifying the illicit and licit accounts. These insights further helped in differentiating the model performance based on the feature importance of each model.

Dashboard Development

An interactive dashboard is introduced, providing a comprehensive overview of account information and transactional activities. The dashboard is designed to interface with a real-time database containing addresses, transaction features, and associated risk factors. This dashboard is a valuable tool for quickly reviewing accounts and assessing potential fraudulent activities.

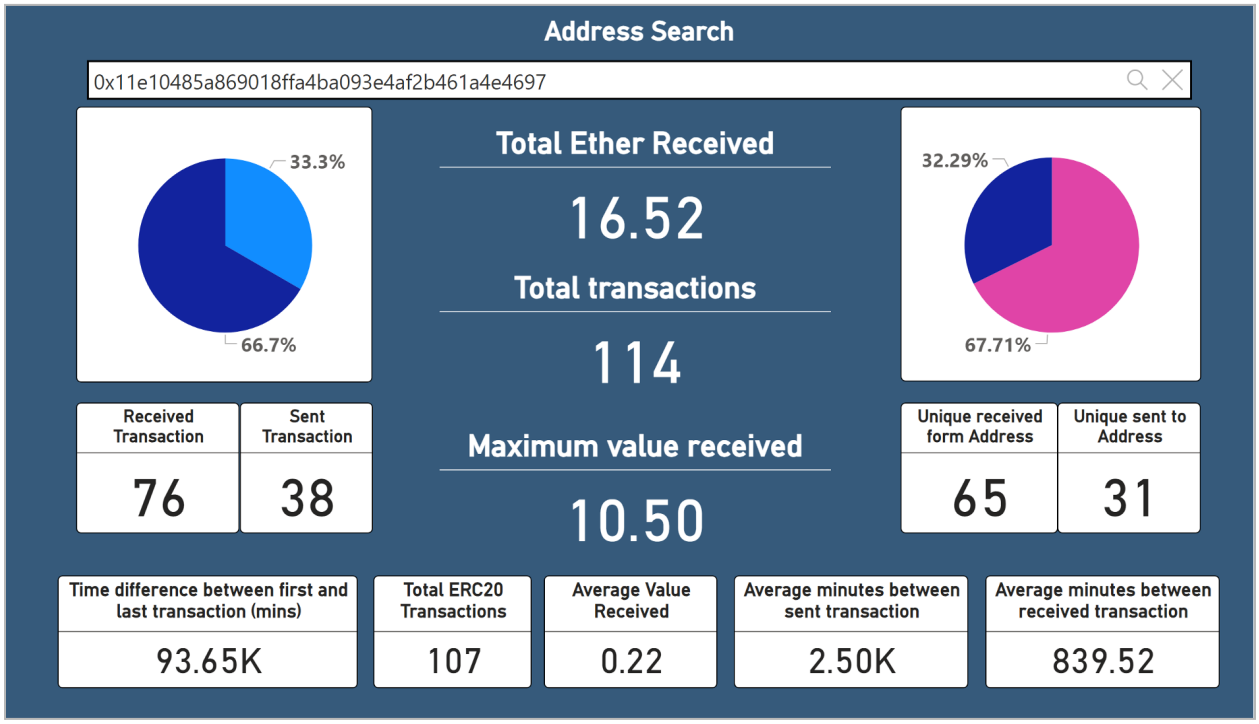


Fig: 6. Interactive Address Search Dashboard.

EDA Conclusion

This section concludes by summarizing the findings from the comparative analysis, feature importance, and the interactive dashboard development. Emphasis is placed on the significance of understanding transactional behaviors, feature importance, and the potential for real-time monitoring to evaluate fraudulent activities associated with Ethereum accounts.

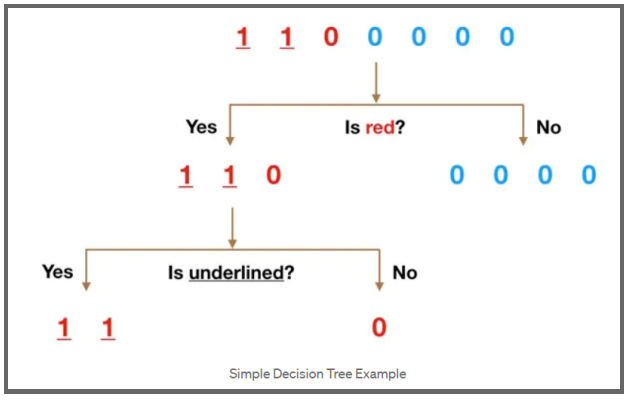
Model Implementation

In the field of fraud detection, the use of machine learning models is critical in discriminating between legal and fraudulent activity. Supervised and unsupervised machine learning models are two essential techniques for efficiently combating fraudulent conduct. Supervised models that have been trained on labeled data learn to detect patterns associated with frauds and non-frauds, allowing them to make accurate predictions. Unsupervised models, on the other hand, probe into the data's intrinsic structure, looking for abnormalities and departures from the norm that may indicate fraudulent activities. We investigate how supervised and unsupervised models contribute to our fraud detection efforts in this research, providing insights into their strengths and uses.

Supervised Learning Models

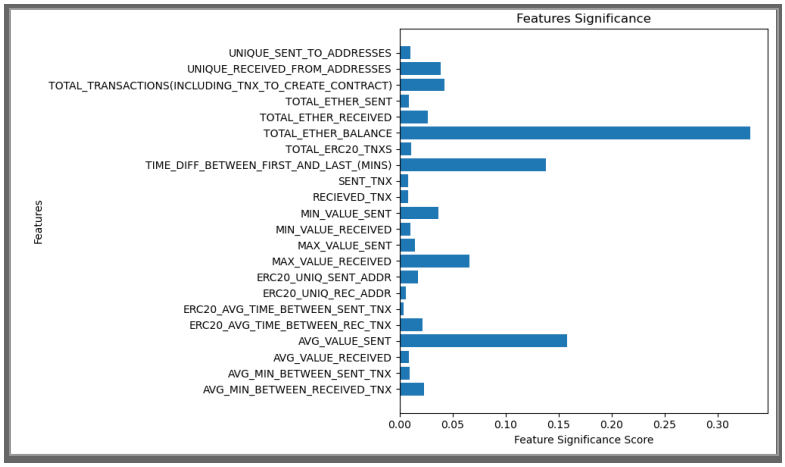
Decision Trees

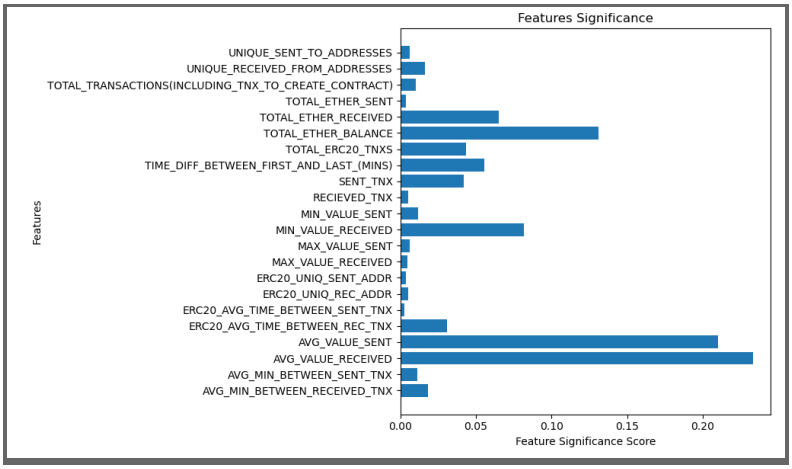
A Decision Tree is a non-parametric supervised learning algorithm used for classification and regression tasks. It is structured as a hierarchical tree with a root node, branches, internal nodes, and leaf nodes. Decision Trees offer interpretable models that facilitate decision-making. In the figure below we can see how the decision tree works.

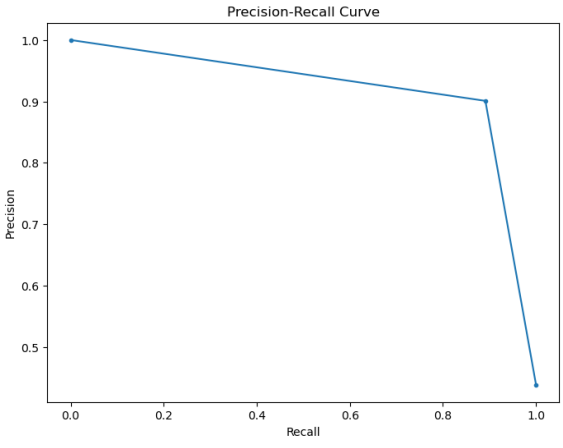


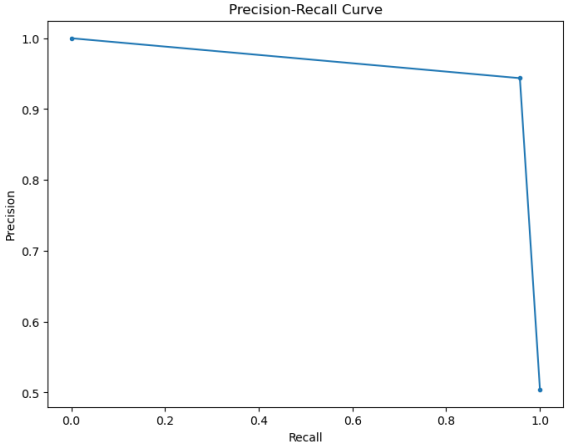
Simple Decision Tree Example

Feature Importance – Random Sampling vs Coinbase









	Random Sampling	Coinbase
Accuracy	90.90%	94.80%
Precision	90.00%	94.30%
Recall	89.20%	95.50%

In our analysis, we utilized a decision tree classifier, achieving a remarkable 91% and 94.8% accuracy In conclusion, the decision tree excels in classifying both classes, demonstrating its suitability for this task.

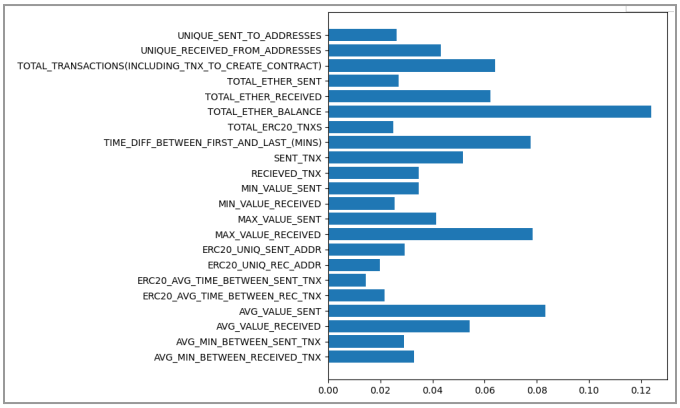
Random Forest

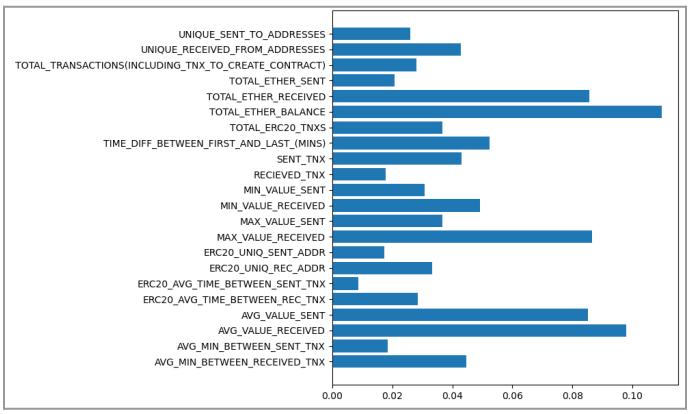
In our analysis, we employed a Random Forest classifier with 100 estimators, achieving outstanding results. The model exhibited an impressive accuracy of 94% and 97.2% in random sampling and coinbase resp, indicating its ability to correctly classify instances. Further examination of the classification performance revealed exceptional results:

	Random Sampling	Coinbase
Accuracy	94.00%	97.20%
Precision	95.60%	97.70%
Recall	90.60%	96.80%

Additionally, we conducted a feature importance analysis, which revealed the significance of different features in making accurate predictions. The important scores indicated that they were the most influential in the classification process.

Feature Engineering of Random Sampling vs Coinbase





ADA Boost

In our analysis, we also employed the AdaBoost Classifier to enhance our model's performance. We conducted a hyperparameter search using GridSearchCV to find the optimal settings, resulting in the following best hyperparameters: n_estimators=100 and learning_rate=0.01.

	Random Sampling	Coinbase
Accuracy	89.50%	93.90%
Precision	90.80%	90.80%
Recall	84.60%	84.60%

The AdaBoost model achieved a test accuracy of approximately 89.51% and 94% in random sampling and coinbase dataset respectively, demonstrating its effectiveness in classification as shown above. Upon checking the confusion matrix it showed that it correctly identified 4106 non-fraudulent cases and 2908 fraudulent cases, with some misclassifications. The classification report provides further insights into the model's performance, with precision and recall values indicating its ability to distinguish between the two classes. Overall, AdaBoost is a valuable addition to our ensemble of classifiers, contributing to our fraud detection task's success.

XGBoost

XGBoost, short for Extreme Gradient Boosting, is a versatile and scalable machine learning algorithm that stands out for its efficient memory usage and parallel and distributed computing capabilities.

In our analysis, we employed the XGBoost (Extreme Gradient Boosting) classifier, a powerful ensemble learning technique known for its outstanding performance in various machine

learning tasks. XGBoost is particularly well-suited for both classification and regression problems, making it a versatile choice for our dataset.

To optimize the XGBoost model's performance, we utilized a grid search with cross-validation (GridSearchCV) to find the best combination of hyperparameters. This approach allowed us to fine-tune parameters such as the number of estimators (trees), maximum tree depth, and learning rate. The hyperparameter tuning process resulted in a set of optimal hyperparameters that enhanced the model's predictive accuracy.

For both datasets, "random sampling" and "coinbase," we observed impressive performance metrics:

	Random Sampling	Coinbase
Accuracy	94.84%	97.55%
Precision	96.00%	98.00%
Recall	92.00%	97.00%
F1 Score	94.00%	98.00%

These results highlight the effectiveness of the XGBoost classifier in accurately classifying instances from both datasets. The model demonstrated strong precision and recall values, indicating its ability to correctly identify positive and negative cases.

The high test accuracy suggests that the XGBoost model is well-suited for handling these particular datasets, showcasing its robustness in capturing complex patterns and relationships within the data. The classifier proved to be a valuable asset in our analysis, providing high

predictive accuracy and reliable classification performance. Its adaptability and ability to handle diverse datasets make it a prominent choice for various machine learning tasks.

Deep Neural Network (DNN)

Deep Neural Networks (DNNs) are complex neural networks with multiple layers, widely used for tasks like image recognition and natural language processing due to their ability to model intricate data relationships. They require careful design and significant computational resources for training.

We applied Deep Neural Network (DNN) models to two distinct datasets: "Random" and "Coinbase." DNNs are a powerful class of machine learning models known for their ability to capture complex patterns and relationships in data.

For both datasets, we developed a DNN model architecture consisting of multiple dense layers with activation functions to facilitate feature transformations. The model was optimized using the Adam optimizer and binary cross-entropy loss function, with accuracy and mean squared error as evaluation metrics.

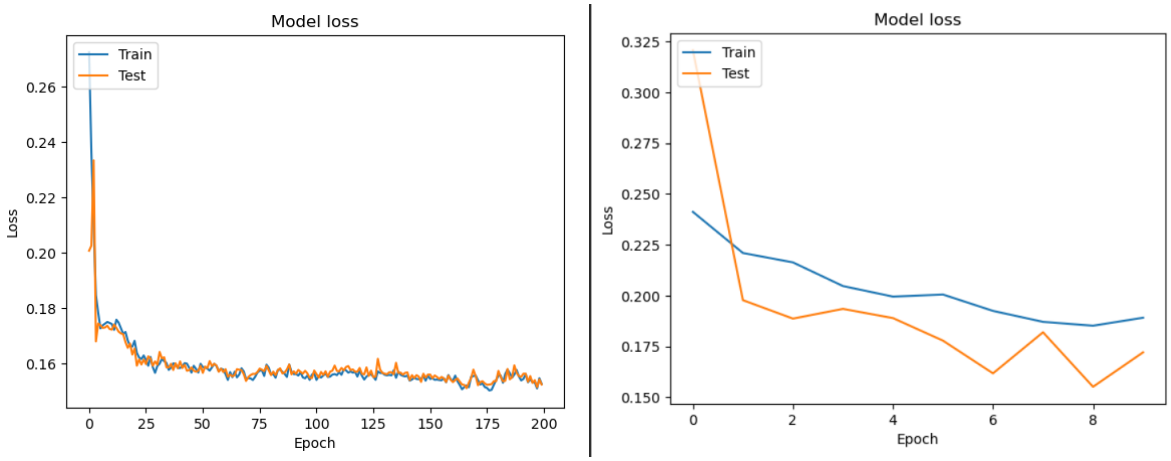
These results provide insights into the performance of the DNN models on the respective datasets. While achieving respectable test accuracy, it's important to consider other performance metrics, such as precision and recall, to gain a deeper understanding of the model's effectiveness in correctly classifying instances.

	Random Sampling	**Coinbase**
Accuracy	81.40%	76.70%

Precision	81.80%	78.19%
Recall	74.10%	74.70%

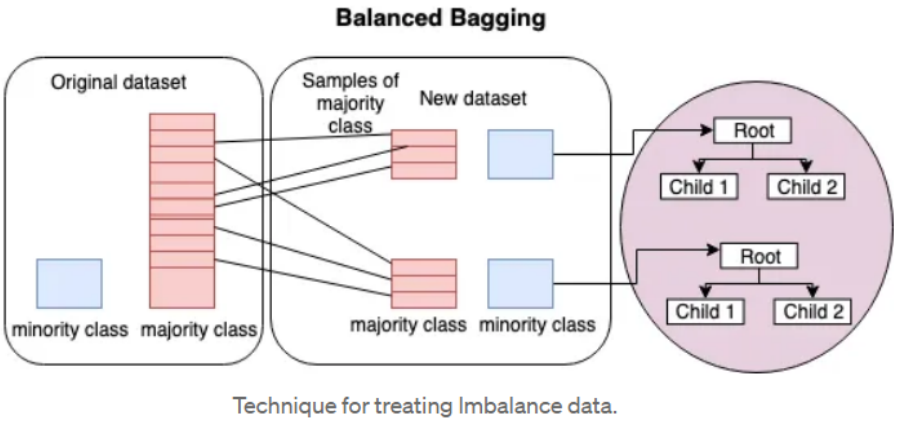
The visualization of model loss over epochs for both datasets indicates the training and validation performance, showing the convergence of the model during training.

Model Loss Random Sampling vs Coinbase



The DNN models showcased their ability to learn intricate patterns from the data and make accurate predictions. The differences in performance between the two datasets may stem from variations in dataset characteristics, highlighting the importance of selecting appropriate models and tuning hyperparameters for specific tasks.

Balanced Bagging Classifier



The Balanced Bagging Classifier is an extension of the Bagging Classifier. It addresses class imbalance by using random undersampling, balancing class distribution in each subset. This mitigates the bias towards the majority class, improving the model's performance on minority classes.

We implemented the Balanced Bagging Classifier on both the datasets Random Sampling and Coinbase, a binary classification problem with severe class imbalance that will help solve the balancing issues when we execute the models on the real datasets that might be imbalanced. The Balanced Bagging Classifier yielded remarkable results:

	Random Sampling	**Coinbase**
Accuracy	94.17%	96.67%
Precision	95.30%	97.74%
Recall	92.90%	95.53%

The Balanced Bagging Classifier showed good computational efficiency, with a computation time of 53.99 seconds.

This technique stands out as an effective solution for imbalanced classification tasks. By combining bagging with resampling techniques, it balances class distribution and enhances classifier performance. However, it requires enough minority class samples to be effective. Its combination of bagging and resampling techniques can significantly improve model performance, making it a valuable addition to the machine learning toolkit.

Unsupervised Learning Models

Anomaly Detection

Anomaly detection is a crucial area in machine learning with diverse applications. It plays a pivotal role in ensuring the integrity and security of data-driven systems.

The Significance of Anomaly Detection

Despite its relatively low profile, anomaly detection is indispensable in various domains. Its importance stems from its ability to identify rare events or outliers within a dataset. Two key properties distinguish anomalies:

- Fewness: Anomalies are scarce; they constitute only a small fraction of the overall data. In any given dataset, there are typically very few anomalous samples.
- Difference: Anomalous samples exhibit values or attributes that deviate significantly from those of normal samples. This distinctiveness makes anomalies easier to isolate when compared to normal data points.

Understanding the Isolation Forest Algorithm

The Isolation Forest algorithm capitalizes on the inherent properties of anomalies to efficiently detect them. It works by creating a series of random partitions, effectively building an isolation tree. Here's a simplified overview of how the algorithm operates:

The algorithm randomly selects a feature and a split value to partition the data. This process continues recursively until each data point is isolated. The depth of the isolation tree required to isolate a specific data point serves as a measure of its anomaly score. Anomalies are typically isolated with shorter path lengths, while normal points require longer paths. Multiple isolation trees are constructed in this manner, forming an ensemble. The anomaly score for each data point is calculated as the average path length across all trees. Shorter average path lengths indicate higher anomaly scores. A threshold is applied to the anomaly scores to classify data points as normal or anomalous. Data points with scores above the threshold are considered anomalies. The goal was to identify anomalies within the data and gain insights into the patterns that distinguish them from regular data points.

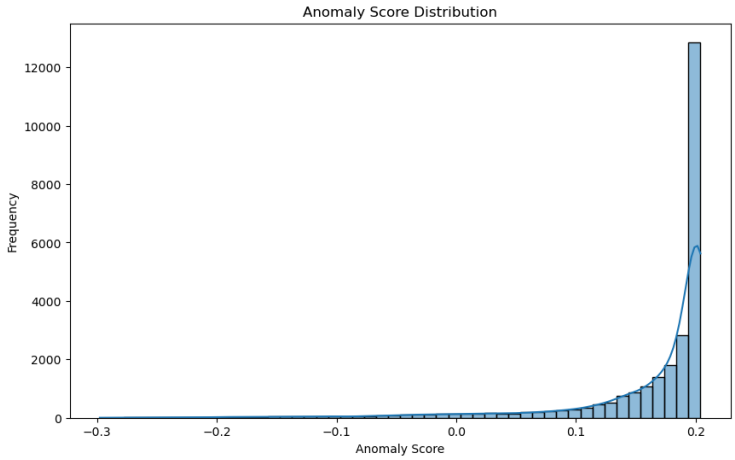
The first step in the anomaly detection process was to define the contamination parameter, which represents the expected proportion of outliers within the dataset. For this analysis, we set contamination to 0.05, reflecting our assumption that anomalies make up approximately 5% of the data.

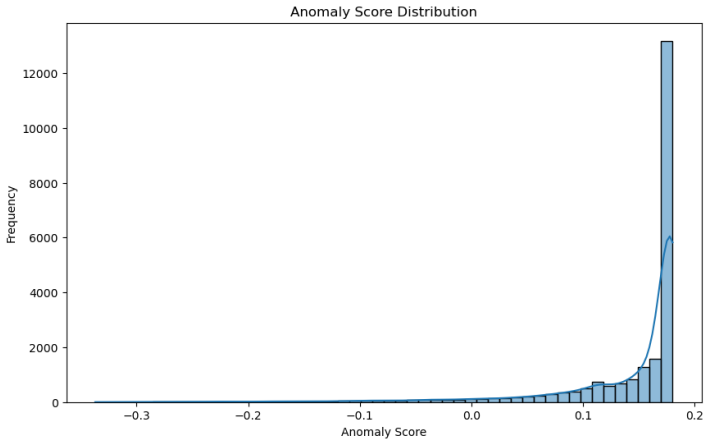
We initialized the Isolation Forest model and fitted it to the dataset. The model assigns an anomaly score to each data point, which quantifies how likely a point is to be an anomaly. Additionally, it assigns binary labels: 1 for inliers (normal data points) and -1 for outliers (anomalies).

Anomaly Score Distribution

To gain an understanding of the distribution of anomaly scores, we created a histogram plot. This plot provides insights into the spread of anomaly scores across the dataset. The presence of anomalies is indicated by lower anomaly scores, while higher scores correspond to normal data points. The histogram's shape and characteristics can reveal information about the separation between anomalies and regular data.

Random Sampling vs Coinbase



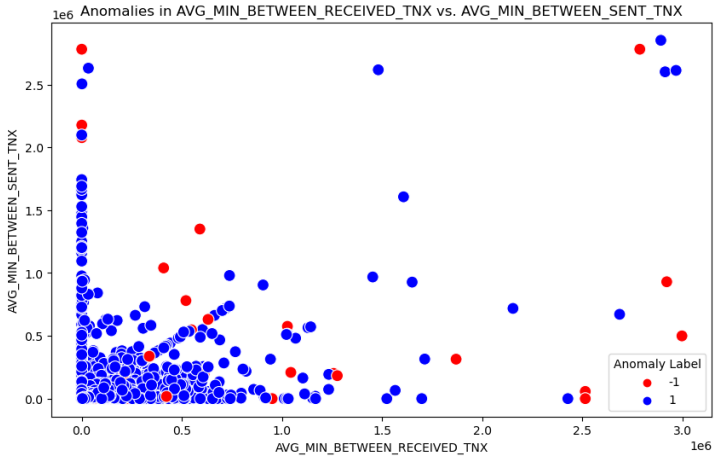


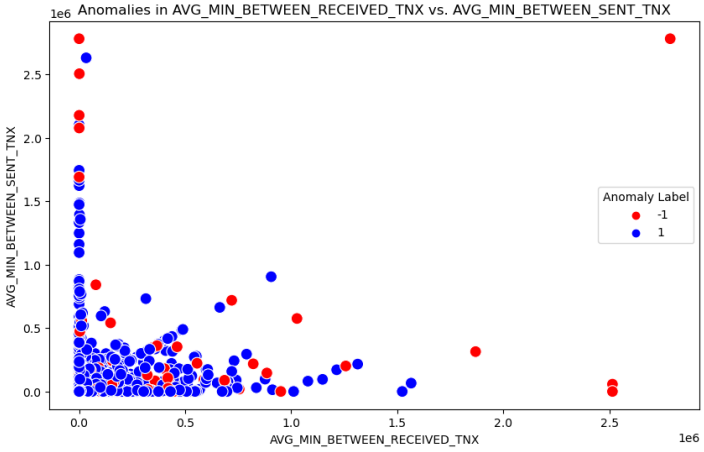
To visualize how anomalies are distributed in the feature space, we selected two features, namely AVG_MIN_BETWEEN_RECEIVED_TNX and AVG_MIN_BETWEEN_SENT_TNX and

created a scatter plot. In this plot, anomalies are highlighted in red, while normal data points are shown in blue.

As evident in the graph we used, the Isolation Forest algorithm effectively identifies anomalies as distinct data points, segregating them from the majority of normal data. This clear separation demonstrates the algorithm's ability to detect anomalies that exhibit significantly different behaviors from the regular data points.

Random Sampling vs Coinbase





Ensembling Model

Voting Classifier

We encountered scenarios where individual models did not perform optimally. The Voting Classifier comes to the rescue by aggregating the predictions of multiple models to make a collective decision. This ensemble learning approach yielded more accurate results compared to individual classifiers.

In our analysis, we employed the Voting Classifier using a variety of base models, including Random Forest, Decision Tree, AdaBoost, Gradient Boosting, XGBoost and Balanced Bagging Classifier. We used the 'soft' voting strategy, which accounts for the confidence of each classifier and attained final scores. It is an ensemble technique in machine learning where each classifier provides probability estimates for classifying data. These probabilities are averaged across all classifiers, resulting in a more nuanced, probabilistic prediction. Soft voting is robust when classifiers have varying confidence levels in their predictions and is suitable for classifiers that provide probability estimates, offering a refined decision boundary and a probabilistic view of the final prediction. It's valuable when understanding prediction confidence is crucial.

	Random Sampling	Coinbase
Accuracy	94.80%	97.60%
Precision	95.80%	98.10%
Recall	93.70%	94.60%
F1 Score	94.70%	97.20%

In conclusion, the combination of supervised and unsupervised machine learning models has proven to be a powerful strategy for enhancing fraud detection. By leveraging the strengths of both approaches, we can more effectively identify fraudulent activities while minimizing false positives. The results presented in this report demonstrate the significant impact of these models in safeguarding against fraud.

As technology and data continue to evolve, there is ample room for further research and innovation in the field of fraud detection. By staying at the forefront of these developments and embracing the capabilities of machine learning, we can continue to adapt and refine our approaches to combat emerging fraud threats.

Feature Importance Analysis

In this section, we present an analysis of feature importances for all the machine learning models that we have used in our project. Feature importance analysis is a critical step in understanding the factors that contribute to the model's predictions. It helps in identifying which features have the most significant impact on the model's performance and can guide feature selection, model optimization, and business decision-making.

- In our project, we employed two powerful tools, SHAP (SHapley Additive exPlanations) and ELI5 (Explain Like I'm 5), to gain a deep understanding of feature importance within our machine learning models.
- SHAP, a cutting-edge technique rooted in cooperative game theory, allowed us to dissect the complex interactions among features and quantify their contributions to model predictions. With SHAP, we were able to visualize the impact of each feature comprehensively, providing valuable insights into the black-box nature of machine learning algorithms.

- Complementing SHAP, ELI5 provided us with a user-friendly interface for interpreting model predictions, making it easier for both technical and non-technical stakeholders to grasp the significance of individual features. Together, these tools equipped us with a robust feature importance analysis framework, enabling us to make data-driven decisions, optimize our models, and enhance our project's overall success.

Shapley Values

In their project “A Unified Approach to Interpreting Model Predictions” authors Lundberg and Lee explain how a model’s interpretability is as central as its underlying accuracy. They adapt the Shapley values from the 1950’s to machine learning scenarios; intuitively, they explain how much a feature, p, contributes to f(x), they count over all possible subsets of features and calculate the model result with and without feature p. This results in a marginal error which is the contribution with or without feature p. Then after averaging it over all possible features, you can get the SHAP value for feature p. Summing over the contributions for all features p, you get a value that depicts how much information contained in the features causes the prediction to shift relative to a model prediction φ0. By summing φ0 and all the individual feature contributions φ1(f, x), i = 1, 2, ..., p, we can get the model prediction :

$f(x) = \varphi 0(f, x) + X \sum \varphi i(f, x)$.

The Shapley value for a certain feature i (out of n total features), given a prediction p is:

$\varphi i(p) = \sum |S|!(n - |S| - 1)!/n!(p(S \cup i) - p(S))$

Behind the scenes, the SHAP package samples a certain amount of features instead of going over all the subset of features (you can set however many subsets you want). They posit that high accuracy in large datasets that are relevant today is often achieved by utilizing complex models (such as ensemble ones). They state that although there are an abundance of methods which seek to drive the explainability of models, it is often unclear which one is superior over the other. In fact, we demonstrate this issue in the next section where we show feature importance plots utilizing different bases. To address this debate on which one to use in which situation, Lundberg and Lee propose the SHAP(Shapley Additive explanation) framework. The entire purpose of SHAP is to get closer to explaining individual predictions for an isolated observation rather than predictions on a global scale. By exploring SHAP, we show that the Machine Learning model is interpretable and explanatory – both of which are vital aspects of any AML model that is designed. Some factors which give rise for such a need are the need to give consumer explanations, anti-discrimination measures, compliance costs and also risk management purposes. SHAP allows us to visualize explanations for a single transaction since it can be both applied for:

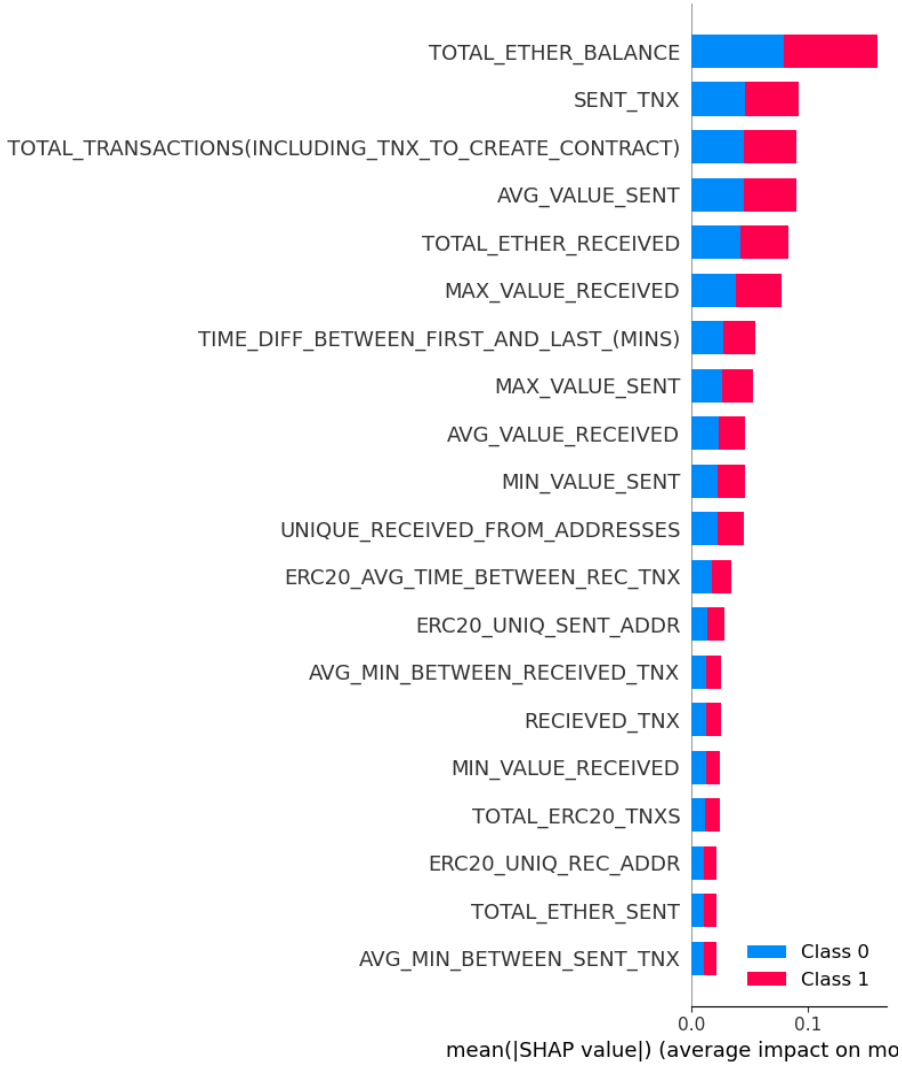
1. Global Interpretability – shows how much (either through a positive or negative relationship) each predictor affects the target variable
2. Local Interpretability – every single observation in our model can have its own SHAP values generated in a map.

Local SHAP values can also be used to calculate overall importance. The global SHAP model has been shown to provide more accurate and useful metrics for model behavior. Shapley

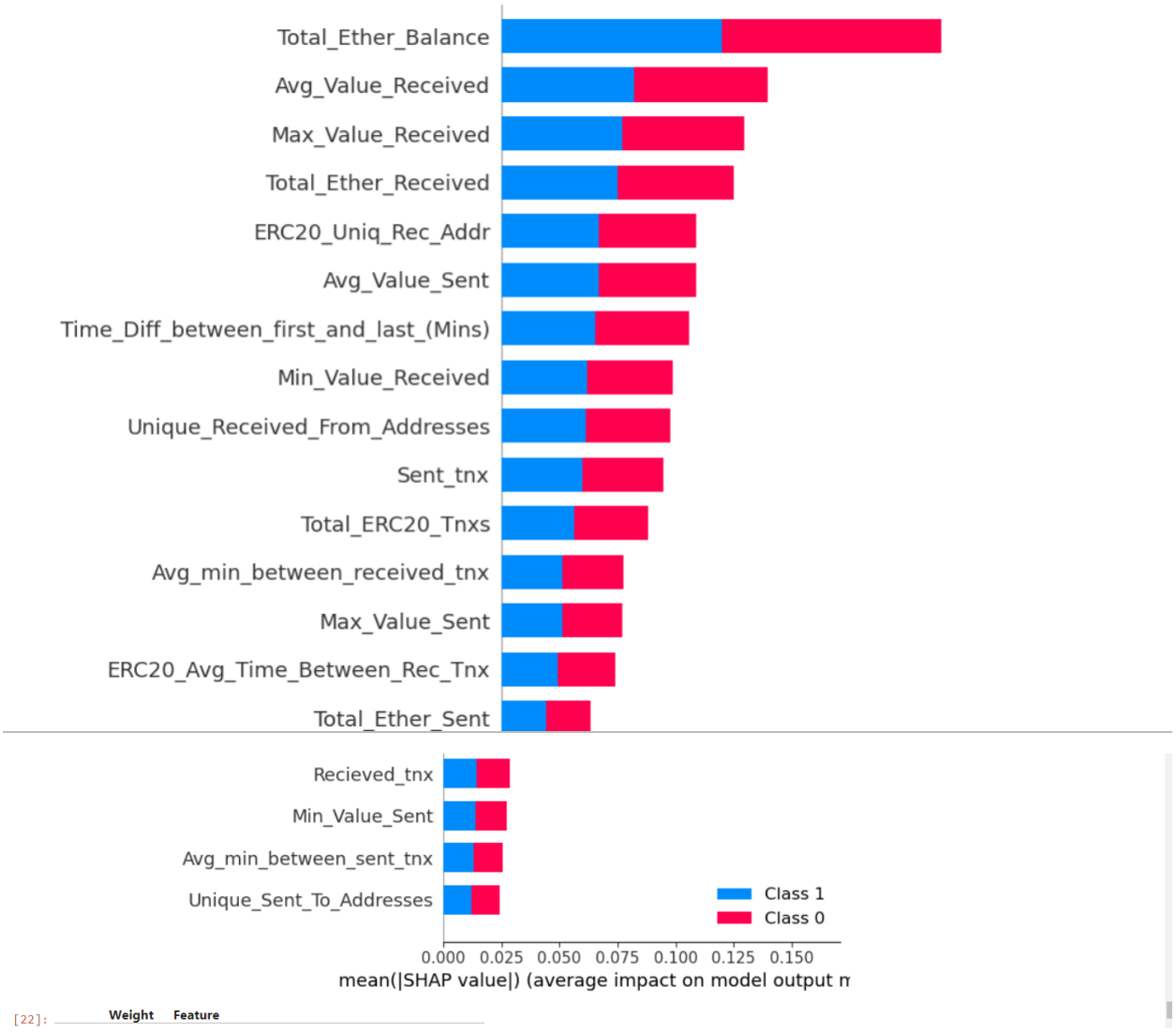
values result from average over all N! possible ordering (which is NP-hard) and require us to find a way to compute these values extremely efficiently. At the end, each Shapley value is a measure of contributions that each predictor (feature) has on a machine learning model.

SHAP value in Random Forest Model

The following Summary plot we have got from our Random Forest model of Random sampling dataset. This plot shows what are the best individual features of this model using SHAP value. We can see the best 5 features here are Total_ether_balance, SENT_TNX, Total_transactions, AVG_Value_sent, Total_ether_recieved.

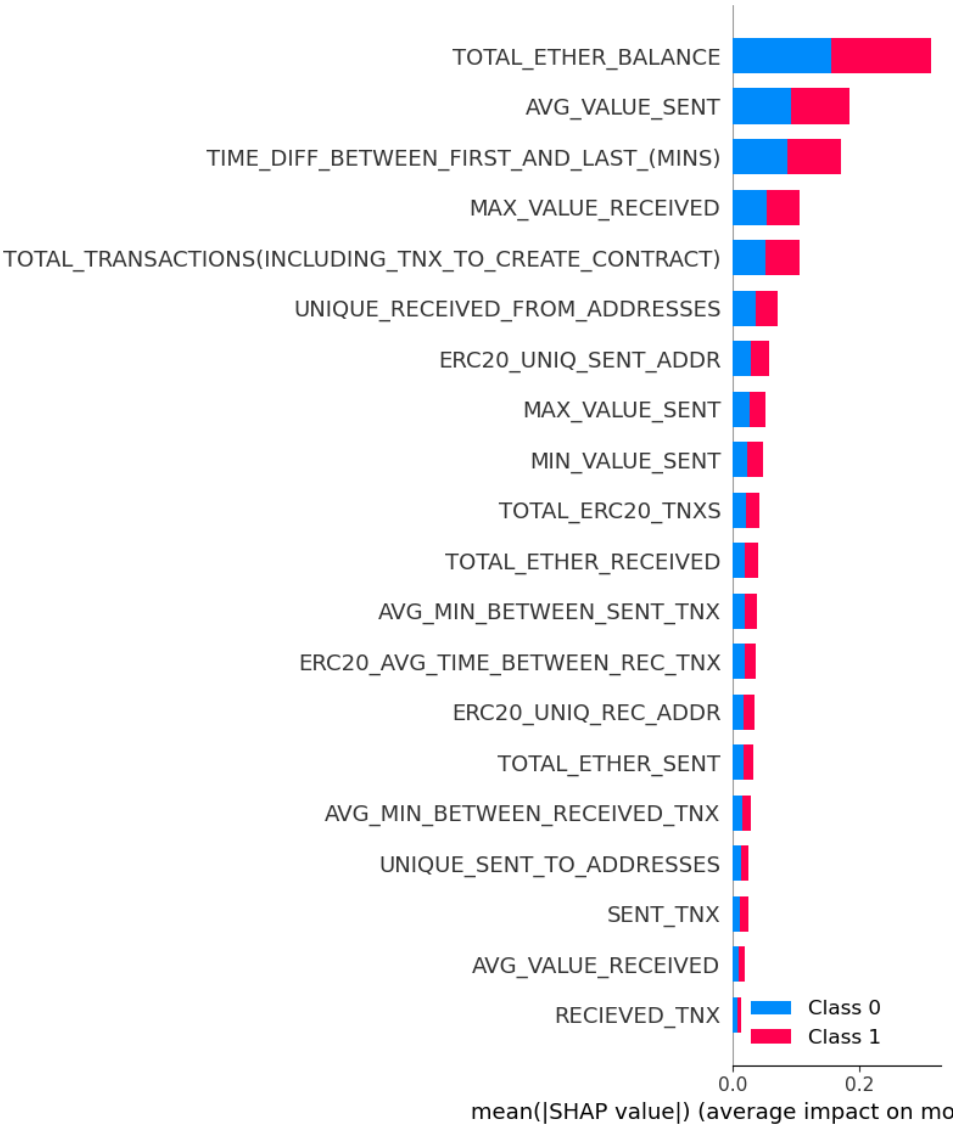


The following summary plot is the plot for Coinbase dataset. This plot shows the best individual features of this model using SHAP value. We can see the best 5 features here are Total_ether_balance, AVG_Value_recieved, Max_value_received, Total_ether_recieved, ERC20_Uniq_Rec_Addr.

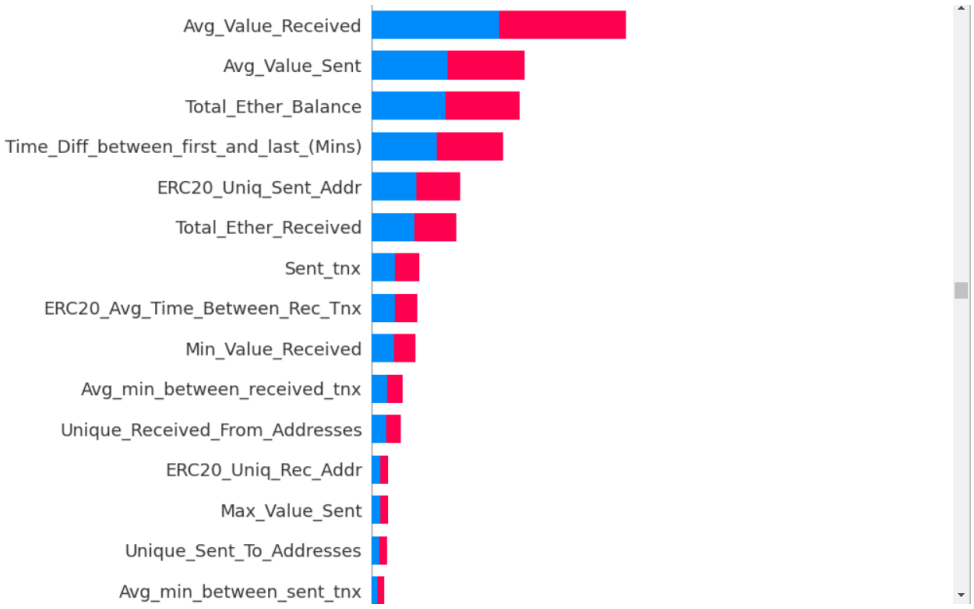


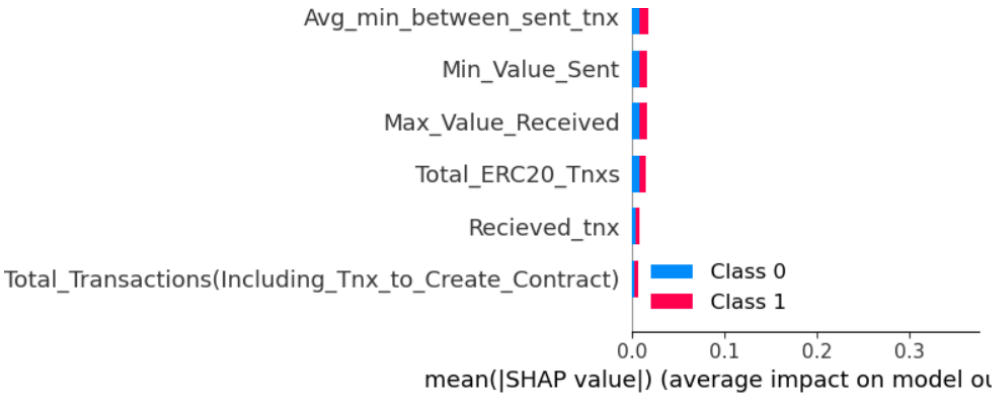
SHAP Value in Decision Tree Model:

The following output is the summary plot for the decision tree model of the Random sampling dataset. This plot shows what are the best individual features of this model using SHAP value. We can see the best 5 features here are Total_ether_balance, AVG_value_sent, Time_Diff_between_first_and_last, Max_value_received, Total_ransactions.



The following output is the summary plot for the decision tree model of the Coinbase dataset. This plot shows what are the best individual features of this model using SHAP value.

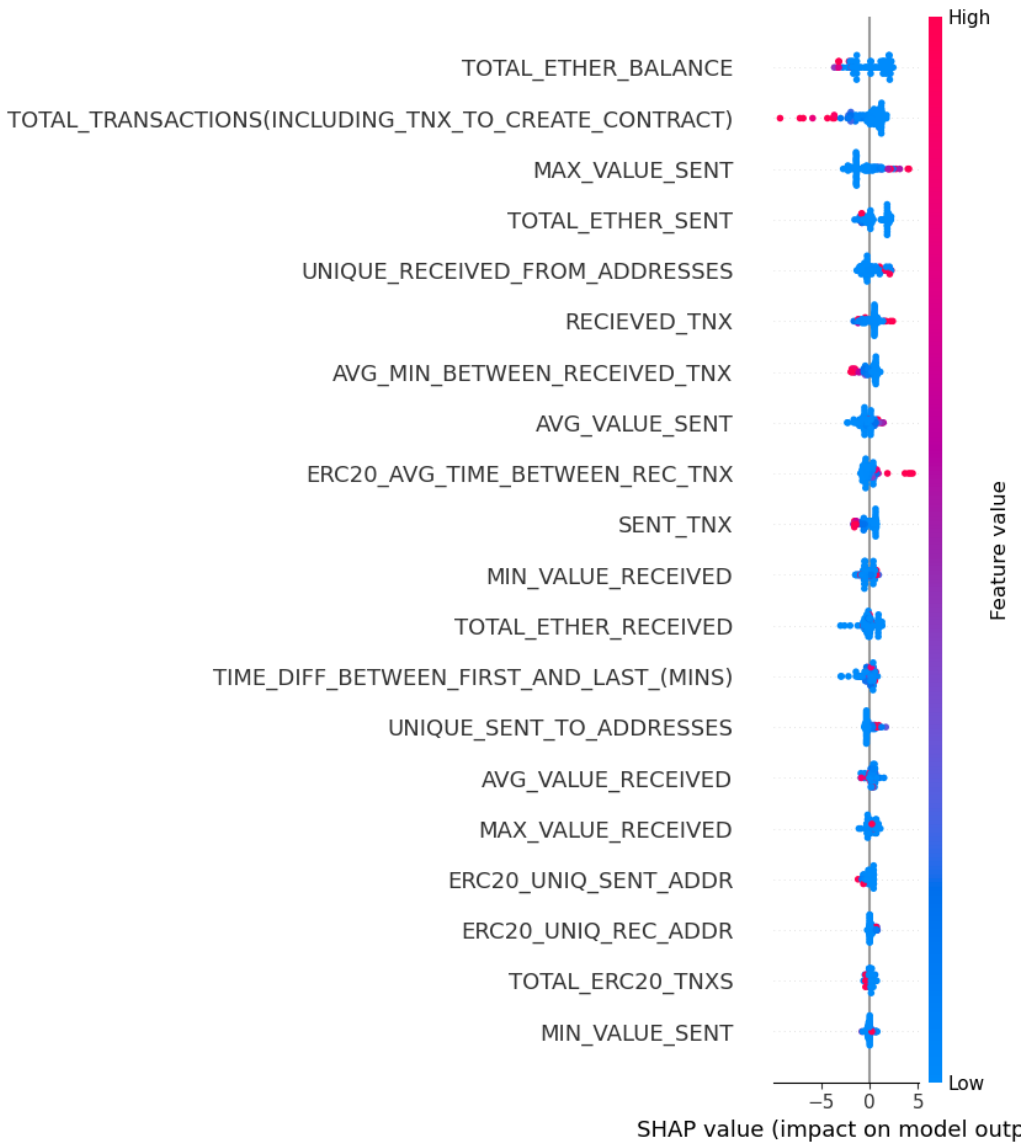




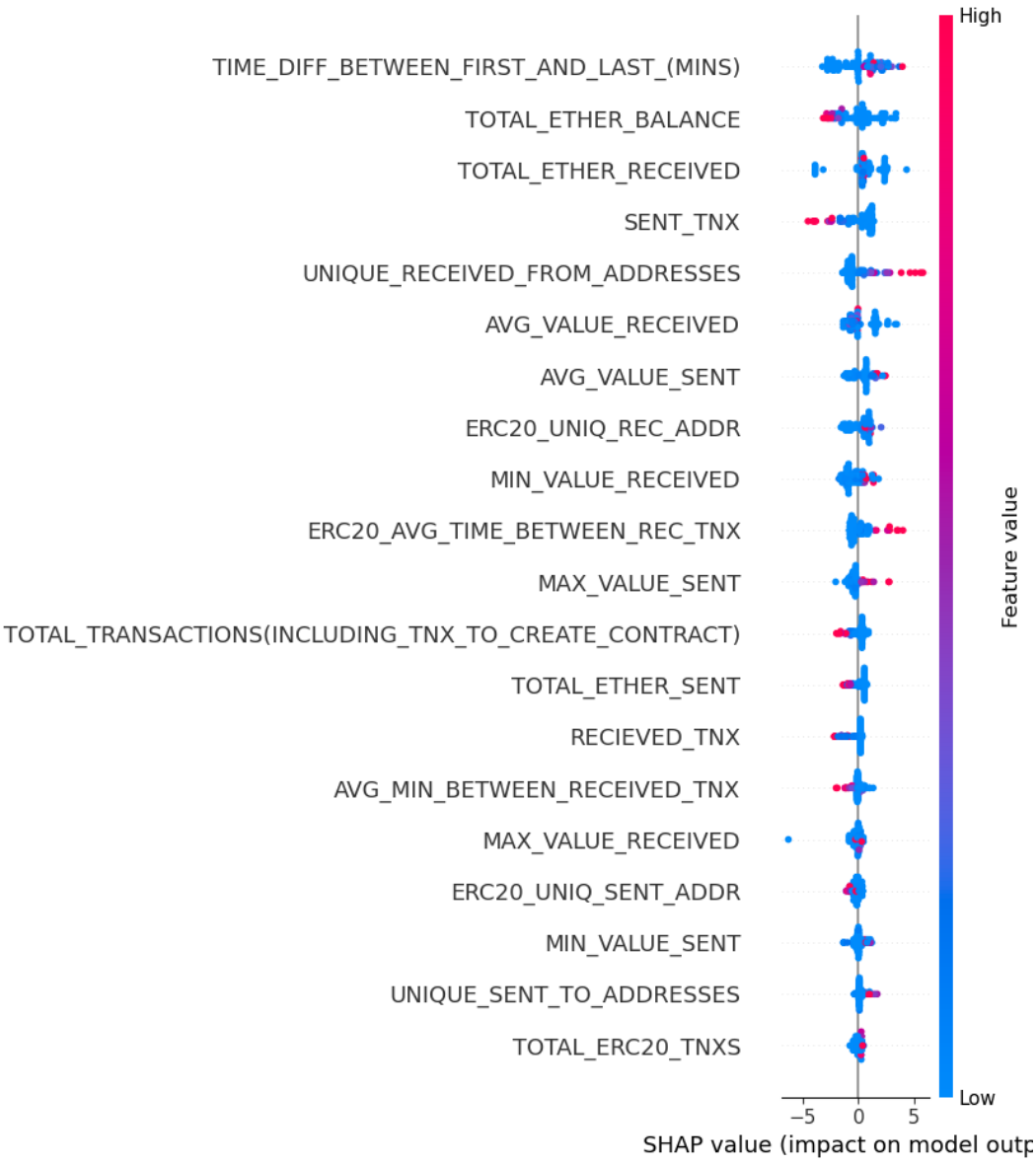
SHAP Value in Gradient Boost Model:

The following summary plot is for Gradient based model features on Randomly sampled dataset.

This plot shows what are the best individual features of this model using SHAP value.

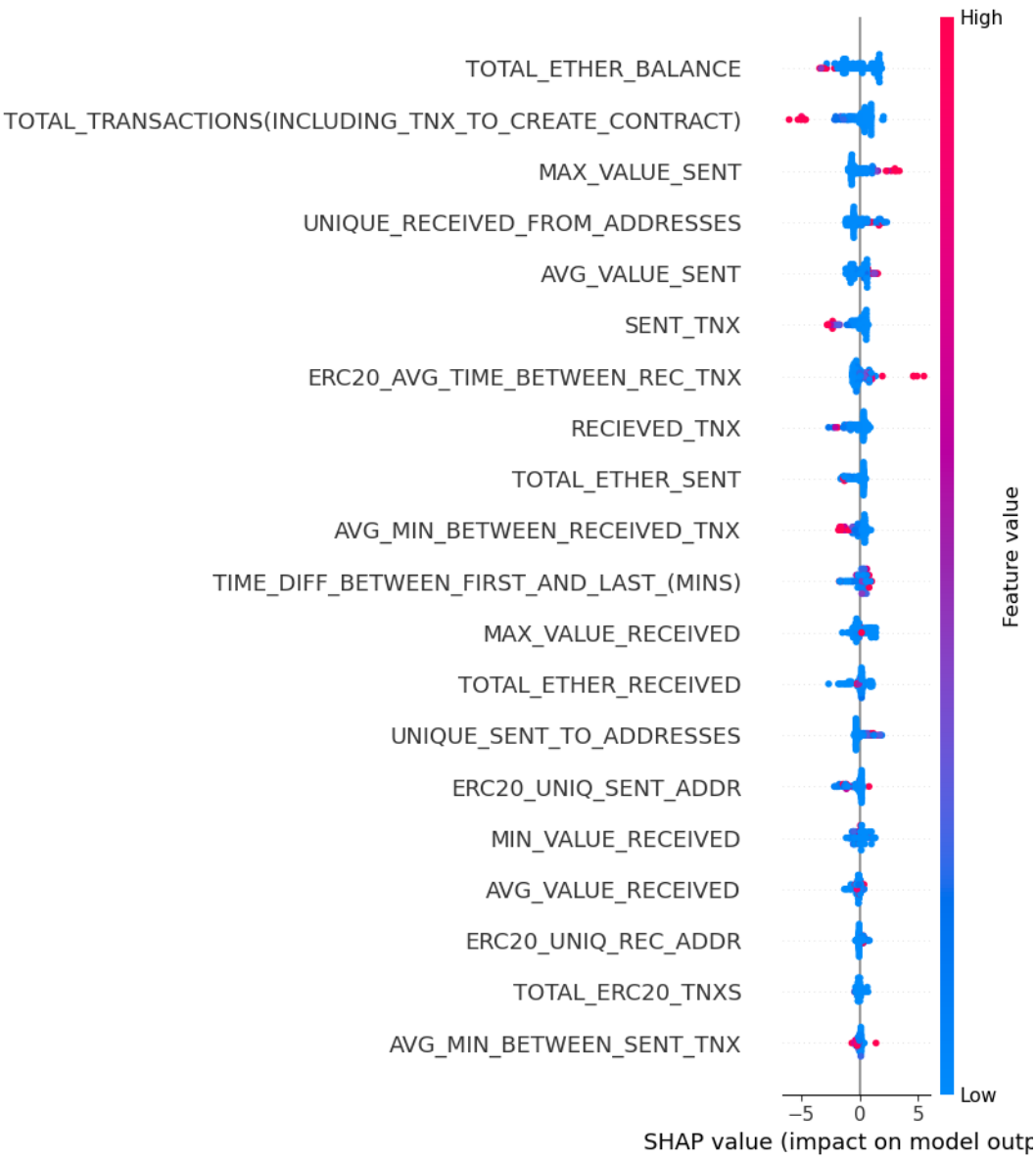


The following summary plot is for Gradient based model features on the Coinbase dataset. This plot shows what are the best individual features of this model using SHAP value.

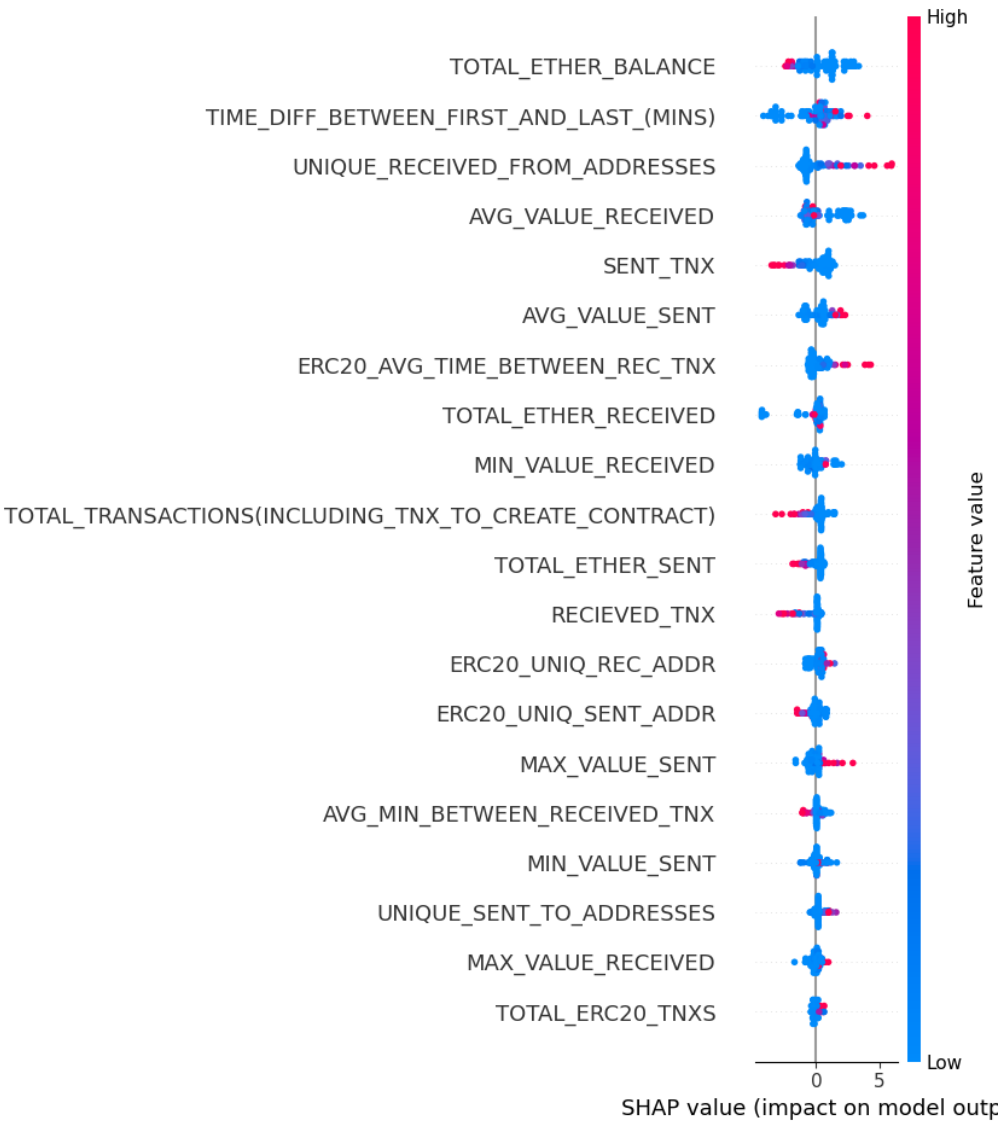


SHAP Value in XGboost Model:

The following summary plot is for XGboost model features on Randomly sampled dataset. This plot shows what are the best individual features of this model using SHAP value.

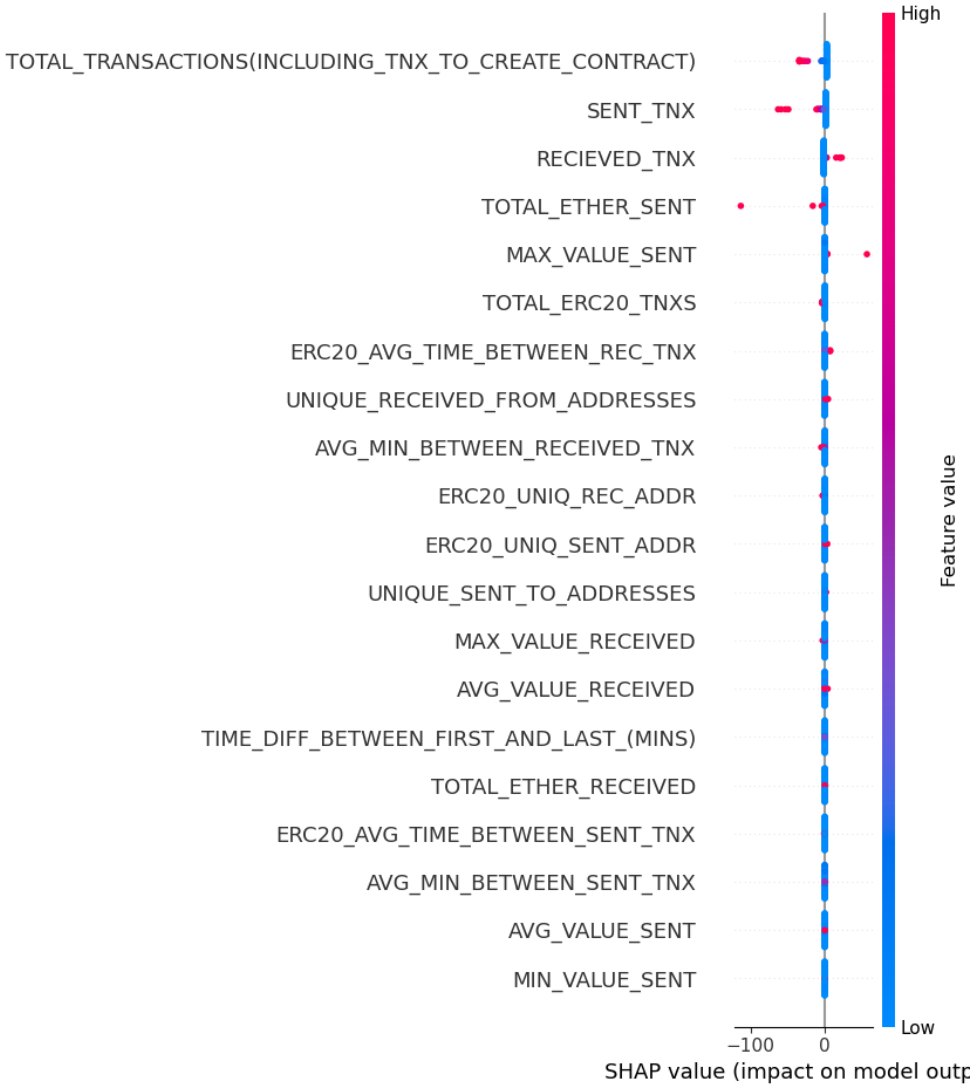


The following summary plot is for XGboost model features on the Coinbase dataset. This plot shows what are the best individual features of this model using SHAP value.

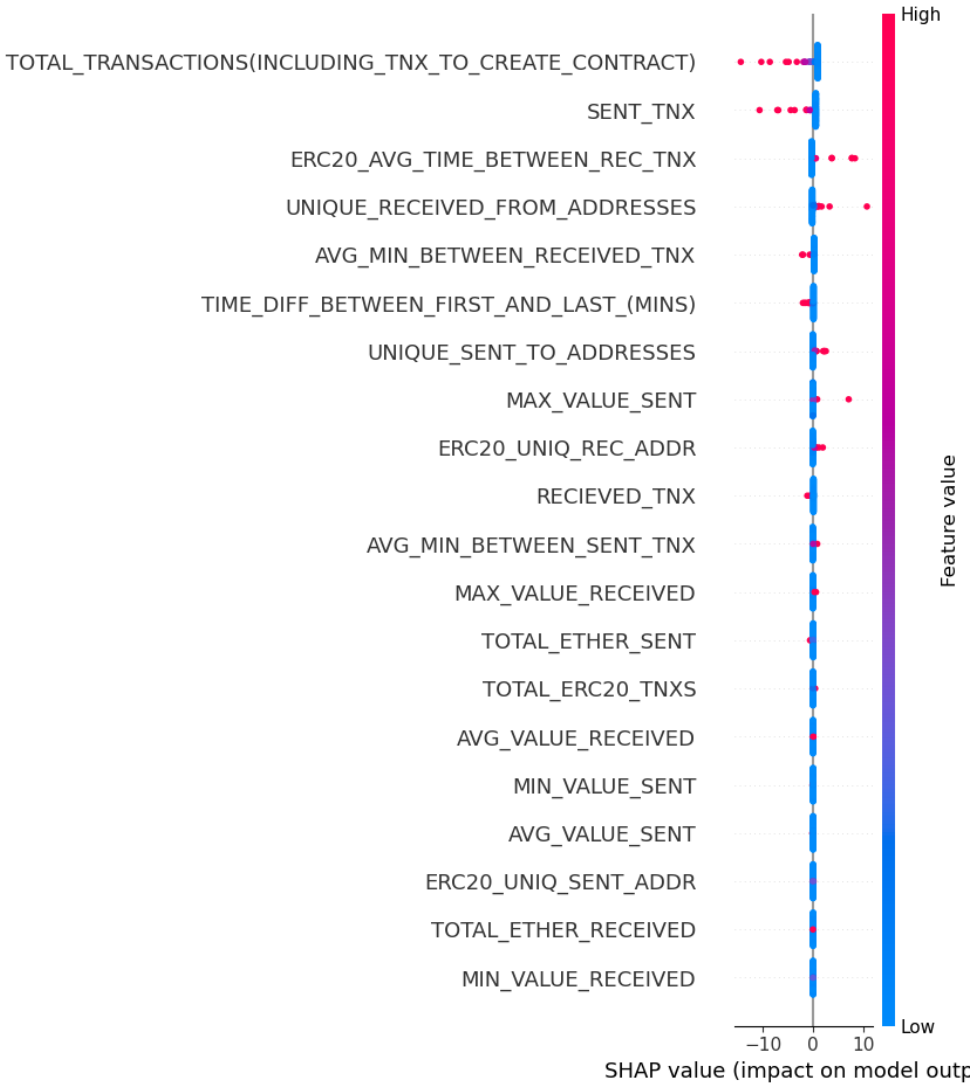


SHAP Value in Logistic Regression Model:

The following summary plot is for logistic regression model feature importance on randomly sampled dataset. This plot shows what are the best individual features of this model using SHAP value.



The following summary plot is for logistic regression model features on Coinbase dataset. This plot shows what are the best individual features of this model using SHAP value.



ELI5 Values

Similarly to SHAP values, a popular metric to explain Machine Learning models is ELI5. An advantage is that it uses actual features for permutation rather than the inputs and outputs of the model itself to determine importance. ELI5 gives us two different ways to understand our model which can be directly compared to SHAP values:

1. Analyze model weights to understand the global performance of the model (Global Interpretability)
2. Analyze individual sample prediction to understand the local performance of the model.

This can help us drill down to the reasons why a particular prediction was made and which parameters played what role in that prediction (Local Interpretability). ELI5 is especially superior than the feature importance available by default with XGBoost as it sets clearly defined permutation definitions. On the other hand, for the regular built-in feature importance on XGBoost, we must interpret whether we care about weight, cover or gain for each specific case. This also allows the model to be explained to a stakeholder who does not need the advanced technical know-how in machine learning model debugging or understanding.

ELI5 Value for Each Model:

Where SHAP values are techniques used to explain the output of complex machine learning models. But in the case of the Eli5 value, the advantage is that it uses actual features for permutation rather than inputs and outputs of the model itself to determine importance.

Feature Importance in XGboost Model:

The following output is the ELI5 score for XGBoost model. Which is showing different weightage for different features. Here we can consider the best score. Higher the score the higher the impact of that feature on the model.

Weight	Feature
0.0923 ± 0.0068	Total_Ether_Balance
0.0850 ± 0.0025	Time_Diff_between_first_and_last_(Mins)
0.0479 ± 0.0060	Unique_Received_From_Addresses
0.0287 ± 0.0018	Sent_tnx
0.0252 ± 0.0022	Total_Ether_Received
0.0124 ± 0.0023	ERC20_Avg_Time_Between_Rec_Tnx
0.0101 ± 0.0008	Max_Value_Received
0.0073 ± 0.0026	Max_Value_Sent
0.0072 ± 0.0024	Avg_min_between_received_tnx
0.0071 ± 0.0022	Min_Value_Sent
0.0064 ± 0.0020	Avg_Value_Sent
0.0057 ± 0.0023	ERC20_Uniq_Rec_Addr
0.0056 ± 0.0013	Avg_Value_Received
0.0048 ± 0.0013	Unique_Sent_To_Addresses
0.0043 ± 0.0012	ERC20_Uniq_Sent_Addr
0.0034 ± 0.0026	Min_Value_Received
0.0032 ± 0.0007	Recieved_tnx
0.0024 ± 0.0021	Total_ERC20_Tnxs
0.0024 ± 0.0011	Total_Ether_Sent
0.0018 ± 0.0011	Total_Transactions(Including_Tnx_to_Create_Contract)
	... 2 more ...

The comparison of both SHAP AND ELI5 values we have got from our Coinbase dataset**.** We have also compared both values and organized them in descending order which can clearly suggest the best features.

Comparison Output:

	Feature	SHAP Value	ELI5 Score
2	Total_Ether_Balance	1.260036	0.092304
6	Time_Diff_between_first_and_last_(Mins)	1.145239	0.085030

18	Unique_Received_From_Addresses	1.041479	0.047904
8	Avg_Value_Received	0.955259	0.005589
0	Sent_tnx	0.801125	0.028698
12	Avg_Value_Sent	0.613343	0.006387
17	ERC20_Avg_Time_Between_Rec_Tnx	0.596542	0.012375
9	Max_Value_Sent	0.521138	0.007274
5	Total_Ether_Received	0.432436	0.025238
21	ERC20_Uniq_Rec_Addr	0.417819	0.005678
20	ERC20_Uniq_Sent_Addr	0.392161	0.004258
4	Min_Value_Received	0.367226	0.003371
1	Recieved_tnx	0.366244	0.003194
15	Total_ERC20_Tnxs	0.346077	0.002440
14	Avg_min_between_received_tnx	0.337779	0.007230
11	Total_Ether_Sent	0.336518	0.002395
3	Max_Value_Received	0.305842	0.010113
10	Min_Value_Sent	0.283321	0.007053
7	Total_Transactions(Including_Tnx_to_Create_Con...	0.232204	0.001819
13	Avg_min_between_sent_tnx	0.159050	0.001109
19	Unique_Sent_To_Addresses	0.150708	0.004790
16	ERC20_Avg_Time_Between_Sent_Tnx	0.031908	-0.000355

The above figure presents the most important variables (in descending order) in our model to help understand what is driving its decisions. In other words, the plot is sorted in order to use SHAP values which show the distribution and subsequent impact that each feature carries on the actual model. This is an improvement to simply using the generic feature importance(s) which varied significantly depending on the importance metric used. The SHAP Global Summary Plot figure gives us a combination of feature importance and also feature effects. Here, we have the most important 5 features and their subsequent effects. Moreover, we can see that the most important features are: Total_Ether_Balance, Time_Diff_between_first_and_last_(Mins) , Unique_Received_From_Addresses , Avg_Value_Received , ERC20_Avg_Time_Between_Rec_Tnx.

Feature Importance in Decision Tree Model:
The Following Output is ELI5 values for each feature as per decision tree model.

Weight	Feature
0.1646 ± 0.0106	Avg_Value_Received
0.1274 ± 0.0061	Time_Diff_between_first_and_last_(Mins)
0.1205 ± 0.0042	ERC20_Uniq_Sent_Addr
0.1111 ± 0.0080	Total_Ether_Balance
0.1013 ± 0.0036	Sent_tnx
0.0846 ± 0.0059	Avg_Value_Sent
0.0743 ± 0.0027	Total_Ether_Received
0.0495 ± 0.0029	ERC20_Avg_Time_Between_Rec_Tnx
0.0442 ± 0.0022	ERC20_Uniq_Rec_Addr
0.0425 ± 0.0049	Min_Value_Received
0.0285 ± 0.0027	Avg_min_between_received_tnx
0.0244 ± 0.0043	Avg_min_between_sent_tnx
0.0206 ± 0.0038	Unique_Received_From_Addresses
0.0160 ± 0.0024	Unique_Sent_To_Addresses
0.0098 ± 0.0011	Min_Value_Sent
0.0094 ± 0.0037	Max_Value_Received
0.0050 ± 0.0015	Max_Value_Sent
0.0046 ± 0.0010	Total_ERC20_Tnxs
0.0038 ± 0.0010	Recieved_tnx
0.0012 ± 0.0010	Total_Ether_Sent
	... 2 more ...

Random Forest Output:

The following Output is ELI5 values for each feature as per Random Forest model.

Weight	Feature
0.1002 ± 0.0072	Total_Ether_Balance

Weight	Feature
0.0578 ± 0.0038	Time_Diff_between_first_and_last_(Mins)
0.0147 ± 0.0044	Unique_Received_From_Addresses
0.0113 ± 0.0022	Sent_tnx
0.0045 ± 0.0018	Avg_min_between_received_tnx
0.0043 ± 0.0019	ERC20_Avg_Time_Between_Rec_Tnx
0.0037 ± 0.0037	Avg_Value_Sent
0.0034 ± 0.0011	ERC20_Uniq_Rec_Addr
0.0032 ± 0.0012	Min_Value_Sent
0.0028 ± 0.0013	Total_Ether_Received
0.0024 ± 0.0010	Min_Value_Received
0.0016 ± 0.0005	Unique_Sent_To_Addresses
0.0016 ± 0.0008	Total_ERC20_Tnxs
0.0015 ± 0.0012	Avg_Value_Received
0.0014 ± 0.0014	Max_Value_Received
0.0014 ± 0.0011	Total_Transactions(Including_Tnx_to_Create_Contract)
0.0014 ± 0.0010	Avg_min_between_sent_tnx
0.0013 ± 0.0012	Recieved_tnx
0.0012 ± 0.0008	Max_Value_Sent
0.0006 ± 0.0007	ERC20_Uniq_Sent_Addr
	... 2 more ...

As shown above, we have calculated ELI5 values for each feature on different models. These figures gave us the most important variables in our model which helped us to understand which features have more impact on our models.

To date, there is a shortage of work with regards to explaining and interpreting anomalies on the blockchain. Shapley values and ELI5 offer an opportunity for regulators and blockchain investigators to be quite confident about the results of a model by showing which variables contribute to a certain prediction the most. Such explanations prevent the model from falling

under regulatory scrutiny of being too opaque. By being able to see and assess the impact that each individual feature (or variable) has on the predicted outcome, it allows investigators to dive much deeper into a potential fraudulent scheme. Moreover, by continuing research in this direction, it can be possible to prevent cryptocurrencies from being overly regulated - which would keep payments accessible for all society.

Hyperparameter Tuning

Hyperparameter tuning is essential to optimize machine learning models. It helps find the best configuration for hyperparameters, improving model performance by maximizing accuracy and reducing overfitting. This process enhances a model's predictive power, making it more effective in real-world applications.

- Sklearn's model_selection package has a tool which assists with hyperparameter tuning of our model. We employ the GridSearchCV tool as a method to ensure that we have the most optimal hyperparameters for our specific dataset and subsequent model.
- GridSearchCV is a brute force way that tries every possible combination of hyperparameters and tells us in the end which one to use. This is one of the most important steps in any machine learning problem as the performance of the model can vary drastically based on the values of each hyperparameter.
- We have got higher precision and recalls after tuning the model. And also, we got better accuracy for each model after hyperparameter Tuning.

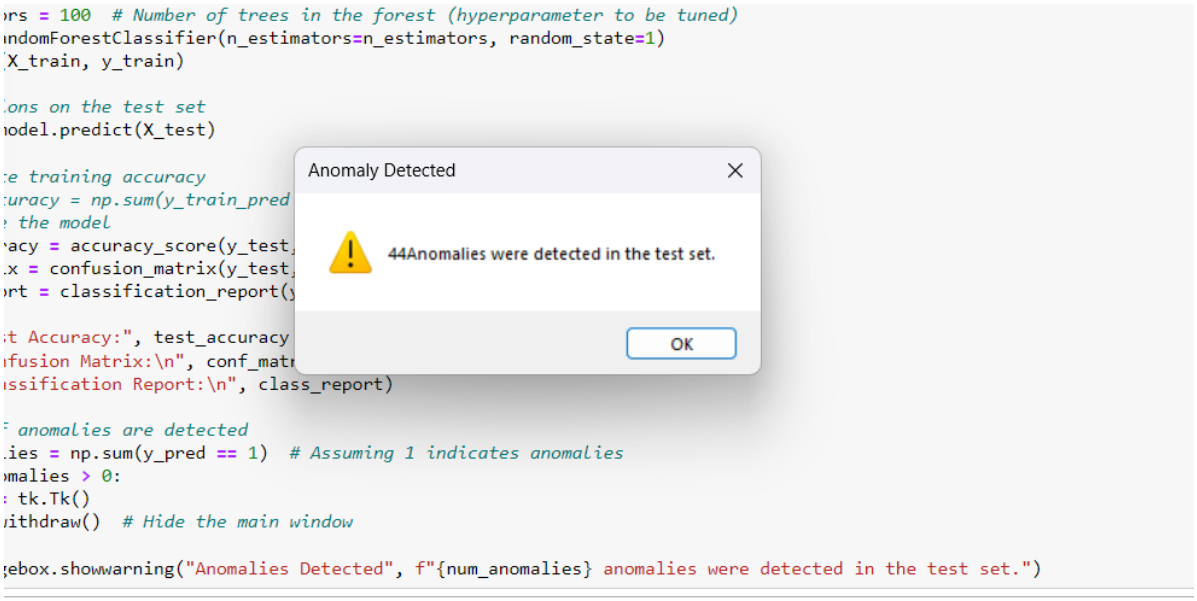
The table below lists the best parameters we have used for hyperparameter tuning of our models.

Models	Best Hyperparameters/Grid Search Parameters
Random Forest Classifier	Number of estimators: [100, 200, 300],max_depth: [None, 5, 10, 20], min_samples_split: [2, 5, 10],min_samples_leaf: [1, 2, 4]
Decision Tree Classifier	max_depth: [None, 5, 10, 20],min_samples_split: [2, 5, 10], min_samples_leaf: [1, 2, 4],
XGBoost	Number of estimators-[100,200,300], max_depth: [3, 4, 5], learning_rate': [0.01, 0.1, 0.2]
DNN Keras Classifier	Number of neurons-128, number of layers-4,activation function-RELU.Output-Sigmoid
Gradient Boost	Number of estimators: [100, 200, 300],learning_rate: [0.01, 0.1, 0.2], max_depth: [3, 4, 5],
Ada Boost	Number of estimators: [50, 100, 200],learning_rate: [0.01, 0.1, 0.2]

Alert System

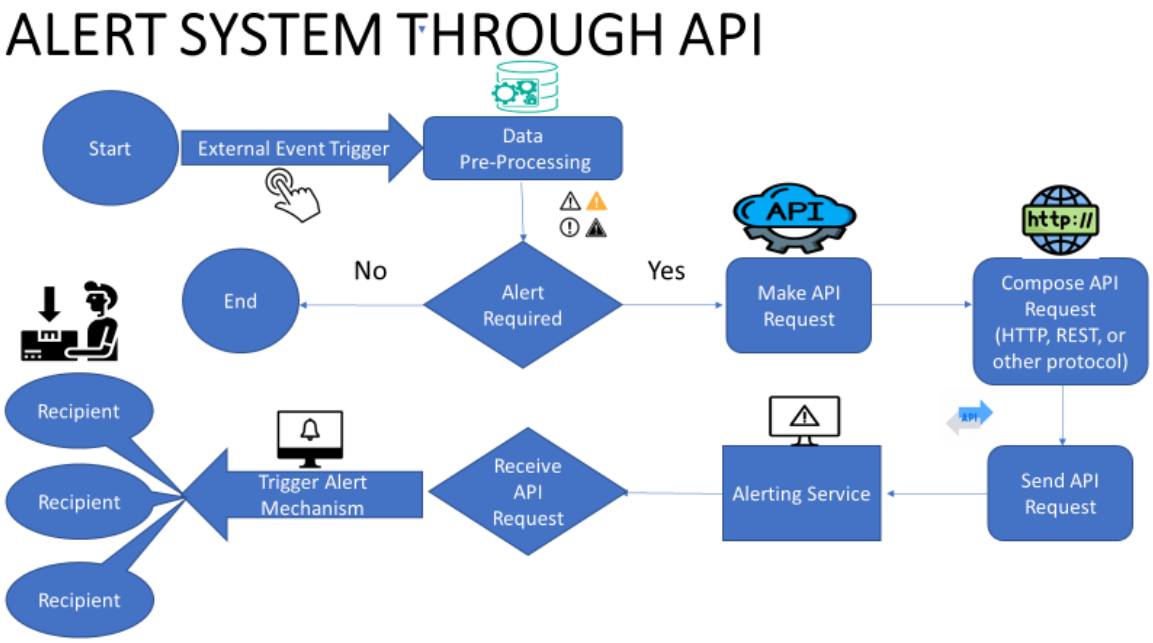
- An alert service for Ethereum fraud detection is crucial for real-time monitoring and early fraud detection in cryptocurrency transactions.
- It enhances security, ensures compliance, and provides customizable alerts, helping users and organizations protect their assets and prevent financial losses.

One of our cohorts goals was to create an alert system using the TKinter python interface to implement window popups for anomalies detected by our models. Additionally, we experimented with code that would allow us to send email notifications to ecosystems actors signed up to receive them.



Unfortunately, the short timeframe we had to conduct this study stifled our attempts to create a fully functioning alert system. Thus, we have instead outlined plans to achieve this using API in our future work section. The below flowchart is a depiciton of how such a system would operate, and can potentially help guide future researchers toward implementation of this alert system.

Flowchart for Alert system



Model Deployment

Structural Classification Speed Impediment

Now that we have trained multiple machine learning models using similar methodologies as the contemporary research cited in earlier sections and combined them into ensemble models, our cohort sought to utilize them on real-time mempool transactions in order to unlock our models' preventative capabilities. However, the addresses implicated in mempool transactions do not have features associated with them, creating a processing hurdle for our cohort. Rather than being able to simply stream mempool transactions and utilize our model to classify each one close to instantaneously, we must first generate features for each address associated with a

particular transaction in order to properly mirror the datasets that our models were trained on. For real-time utilization on Ethereum, without creating an impediment to transaction activity on the blockchain, our classification time must be confined to under 10 seconds. This is due to the fact that this is the average pace at which Ethereum blocks are validated. While the feature engineering process often takes less than 3/10ths of a second per address, when we multiply this figure by the average number of transactions contained within proposed blocks (~100) and consider that each transaction requires two separate addresses to be classified in order to form a recommendation for said transaction, we can see that it is infeasible to utilize our models for real-time transaction monitoring in a meaningful way using the current architecture. Our cohort tried to remedy this feature engineering speed issue using 2 separate approaches.

Transaction-Based Model Approach

Our first approach was to create machine learning models that could classify a transaction as illicit on what we called a "transactional level." This title refers to the fact that the contemporary approaches we had replicated up to this point utilized aggregate address information in order to identify illicit activity and operated on an "address level." Our cohort believed that if we were able to instead successfully flag illicit transactions using the data native to each transaction itself, we would be able to remedy the speed issue caused by our account-based models feature engineering process which requires three separate Etherscan API calls and lengthy subsequent calculations. Our cohort set a goal to create two separate models, one that could classify normal Ether transactions effectively, and another to be able to classify ERC-20 transactions with similar efficacy. In order to do this, we needed to identify criteria for what constitutes an illicit and a legal transaction. Building on top of our previous assumptions, for each illicit address that we had already harvested for utilization in our account-based models,

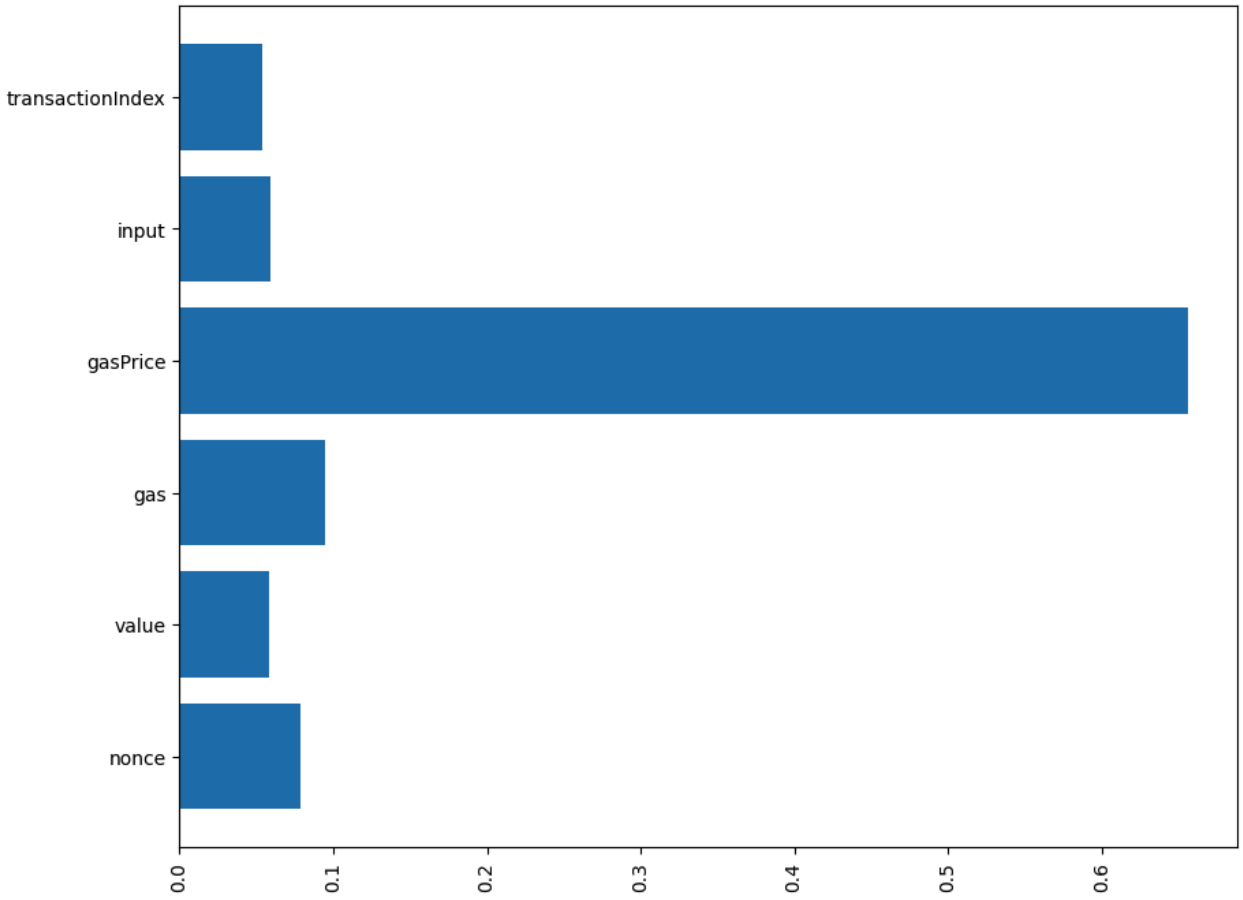
we utilized Etherscan to gather the addresses complete transaction history up to block number 18,000,000. Thus, we made the assumption that every transaction conducted by an illicit address could also be deemed illicit in nature by extension. During the process of obtaining normal Ether transactions conducted by each illicit address, we obtained 420,484 illicit transactions for our new Ether transaction-based dataset. To acquire our legal transactions, we decided to locate the earliest and most recent timestamps associated within our illicit transactions and to sample transactions randomly between these two timestamps. Given that the earliest illicit transaction in our dataset, occurred at block number 1,997,275 on August 8th 2016 and the most recent transaction occurred at block number 17,997,043 on August 25th 2023, we iterated randomly through blocks within this block range and extracted the transactions contained within each randomly sampled block. We iterated through 4,200 random blocks within this block range to garner 549,245 normal Ether transactions. We dropped 105 transactions to ensure that there was no overlap between transactions designated as illegal and legal and began to select the features we would use to build our model. The features selected for our initial transaction-based dataset and preliminary testing of a baseline model were nonce, value, gas, gasPrice, input, and transactionIndex. The following table explains how to interpret the meaning of each selected feature for normal Ether transactions:

Feature Name	Description
nonce	A zero-indexed transaction counter for each address, a transaction with nonce of 1 is the second transaction an address has sent
value	The amount of Ether sent in a transaction, denominated in Wei
gas	The amount of gas units required to complete your computation request, determined by the Ethereum network
gasPrice	The price you are willing to pay for each unit of gas used in your computation request
input	The field where additional data that may be necessary for transaction completion is stored
transactionIndex	The indexed position of the transaction within the block it is within

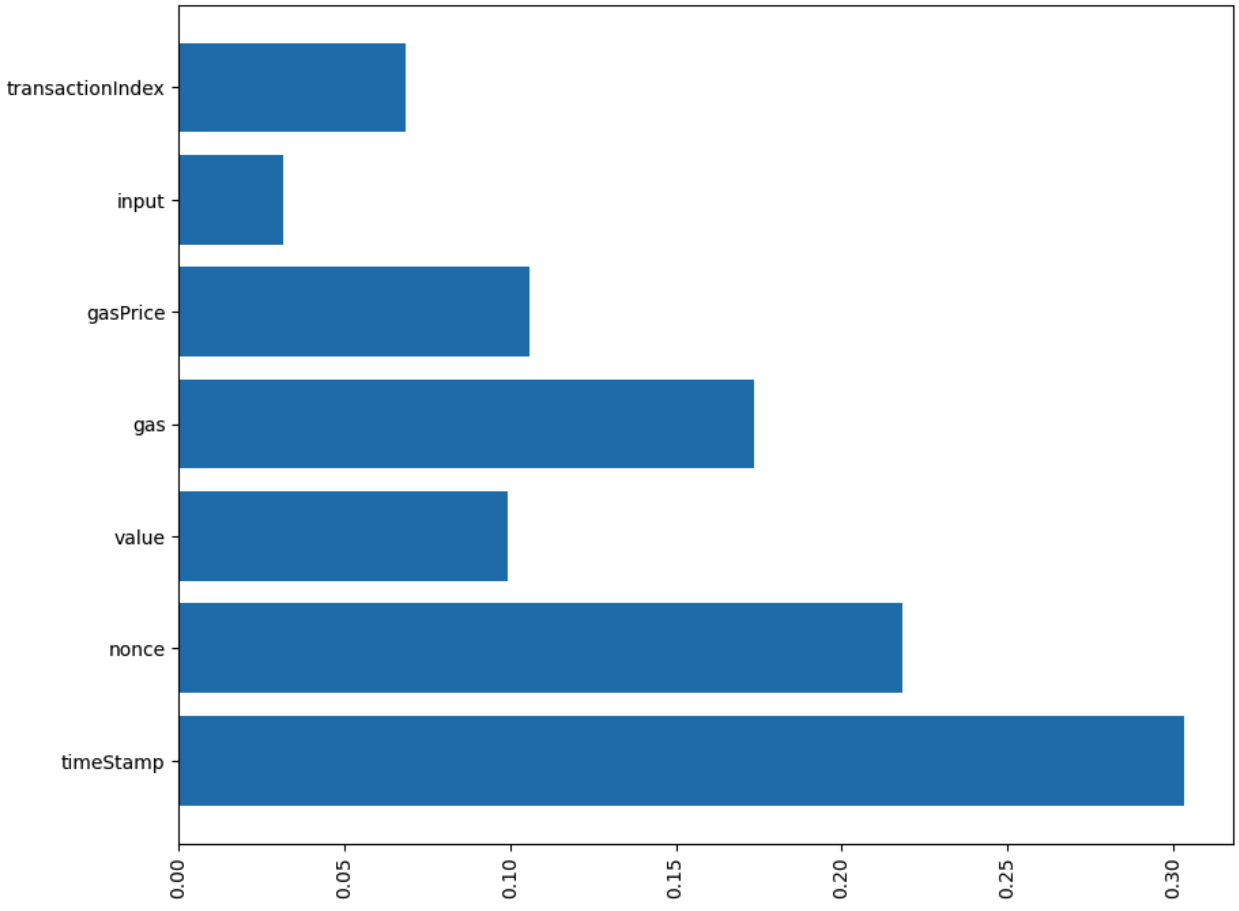
As discussed earlier, the primary advantage of a transaction-based model is that the features required to classify the transaction as illicit can theoretically be derived from the transaction itself rather than having to engage in the time-consuming process of engineering abstract aggregate features. However, there is still a small amount of data cleaning necessary in order to translate the raw transaction data into our desired features. For our illicit transaction data, we decided to convert the input field from our raw transaction data into a binary feature which would denote whether or not input data was present within each transaction. We also converted the value field from Wei to Ether to make this value more interpretable during data analysis. We performed the same transformations on the input and value fields for legal transactions as well but were required to convert all data from hexadecimal format to decimal format prior to transformation due to a difference in the way the transactions were acquired. We

finally combined our illicit and legal transactions together into a single dataset of 834,134 transactions.

Once we had finalized our Ether transaction-based dataset, we split our dataset into training and testing subsets at 70% and 30% of our original dataset respectively. We then opted to utilize a preliminary Random Forest model without any modified hyperparameters to test our initial performance. To our surprise, this model was able to achieve an 87% accuracy on a dataset with an extremely limited set of only six features, with our F1-Score being 87% as well. When evaluating the feature importances of our preliminary Random Forest model, we observed that gasPrice was by far the most important feature in differentiating illicit transactions from legal transactions.



While the performance of our model was much better than expected with only 6 features, our cohort remained skeptical and felt that these results were too good to be true. We felt that the features utilized thus far in our model were not descriptive enough to truly achieve an 87% accuracy, and we began to scrutinize the model's production process to further assess the legitimacy of your results. During our experimentation process we discovered that adding the timestamp feature with all else the same resulted in an increased model accuracy of 93% and F1-Score of 93%. On our feature importance chart, we can also observe that timestamp immediately jumps to the most important feature, with gasPrice receding into a slim 4th place lead.



This revelation suggested to our cohort that our models may not actually be learning the relationships between feature values that can be used to differentiate between illicit and legal transactions. Our cohort became worried that our transaction-based models may somehow inadvertently be learning to classify the transactions by the timestamp in which they occur, utilizing the gasPrice feature as a proxy measure for this. To test our hypothesis, we decided to split our training and test subsets based on their timestamps. The transactions with timestamps in the bottom 70th percentile, interpretable as older transactions, were placed into our training dataset. The remaining transactions, which can be interpreted as more recent transactions, were added to the test dataset. We then trained a new basic Random Forest model on our training data, only to find that when our model was applied to our test data, our model's accuracy crumbled to 58% with an overall F1-Score of 55%. The following depiction showcases the sharp decline in performance after splitting our data based on timestamp:

	precision	recall	f1-score	support
0	0.56	0.83	0.66	125792
1	0.66	0.34	0.45	126145
accuracy			0.58	251937
macro avg	0.61	0.58	0.56	251937
weighted avg	0.61	0.58	0.56	251937

After assessing these results, it is clear that our simple Ether transaction-based model was unable to learn the relationships between the features that indicate a differentiation between illicit and legal transactions when attempting to apply its learnings to transactions from a different time period. Our cohort attempted to build a similar model able to classify ERC-20 transactions as

either illicit or legal as well, but this model also succumbed to the same timestamp related limitations, falling from 97% accuracy to 59% after splitting our data with respect to timestamp. After the lack of efficacy of our preliminary transaction-based models was exposed, our cohort chose not to pursue this methodology for monitoring transactions in real-time. However, we maintain that the applicability of transaction-based models remains an important piece for designing a real-time mempool transaction monitoring system. We encourage future researchers to ensure that their transaction-based models are resilient to timestamp variation before dismissing them from scrutiny.

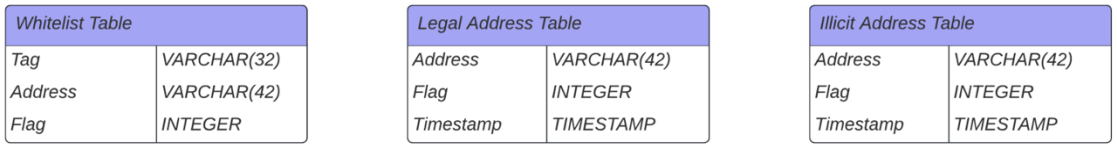
Database Implementation Approach

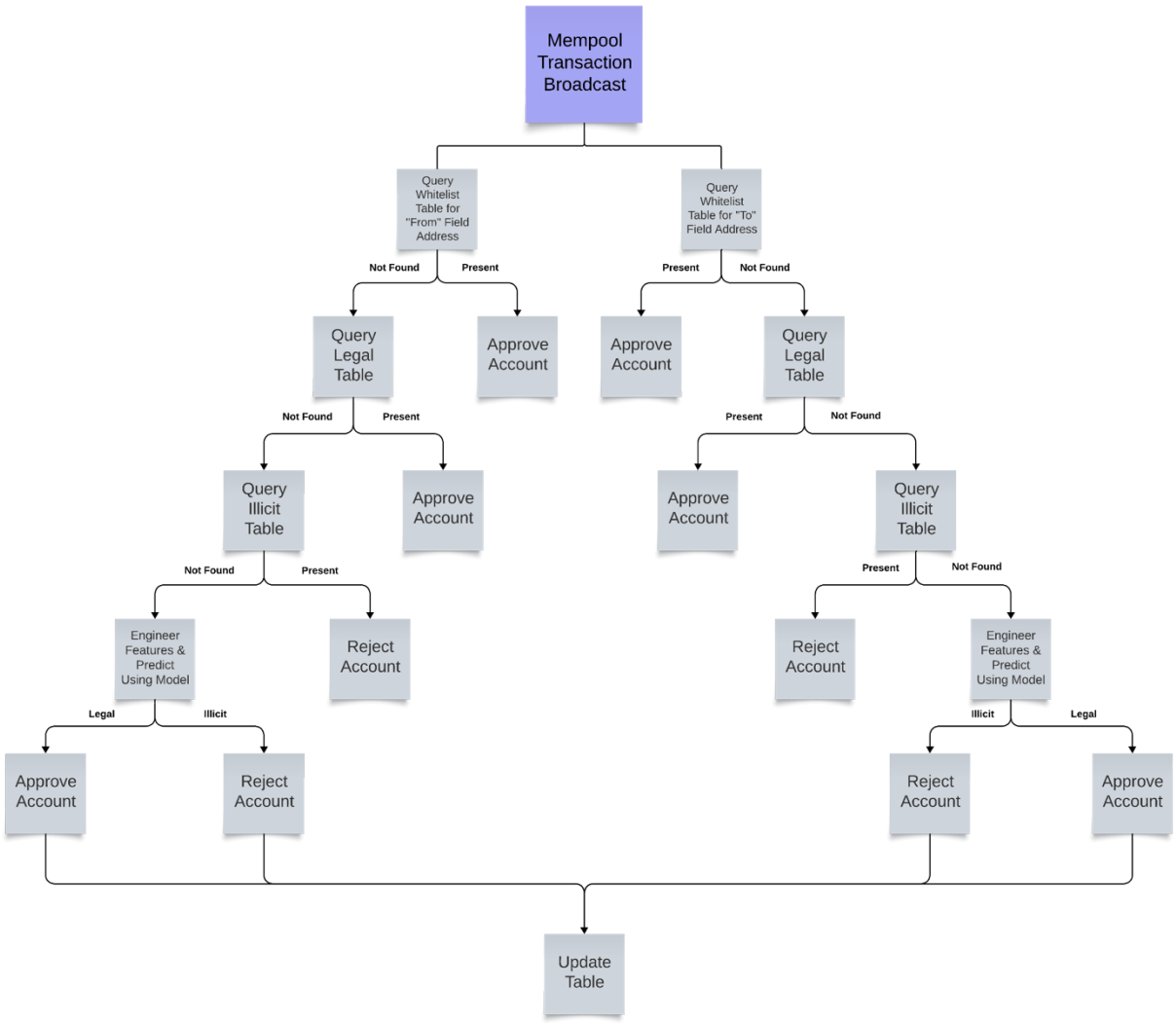
Due to the utilization of a transaction-based model failing to accurately identify illicit activity, our cohort decided to revisit our successful address-based models and solve our feature engineering speed impediment in an alternative way. Our proposed solution is the introduction of a SQL database that would store the model decisions on the addresses that the model classifies. The decision to run this database in tandem with our machine learning model would allow us to increase our classification speed and reduce the number of addresses we are required to engineer features for over time. When our model encounters an address, before we automatically begin the feature engineering process, we can query our database to see whether or not the model has ever made a classification on the address in question in the past. If the address is already in the database, we can simply take the result associated with the model's prior decision. If the address isn’t present within the database, we will engineer the features for that address, make a classification using our model, and store the classification into our SQL database for later use on future encounters. While this system would undoubtedly help us increase the rate at which we

are able to process transactions for a real-time use-case, there are limitations that come along with this proposed methodology as well. While ideally a separate process would be run on the side in order to update the addresses held within the database periodically, our current implementation has no built-in functionality to update the classification of an address over time. This lack of functionality leads us into a slippery assumption of continued address behavior. When we consider illicit addresses, the assumption that addresses flagged as illicit will continue to make illicit transactions was deemed by our cohort as an acceptable assumption to make. However, the assumption that addresses determined to be legal in nature will continue to operate legally into the future degrades the preventative performance of our model. Secondly, in order for our database to speed up transaction classification in a meaningful way, our database would need to accumulate a very large number of addresses in order to possess the breadth of address coverage needed to be applicable in a real-time scenario. Essentially, we would need to accumulate enough addresses within our database so that the unseen addresses requiring feature engineering would represent the vast minority of addresses assessed by the model. A remedy to this limitation could be to run our database in conjunction with our model before full deployment to validators to allow the database to accumulate a sizable number of addresses and reach an acceptable classification speed.

Our database schema is currently structured into three different tables with distinct contents. Our first table is the whitelist table, which will contain addresses commonly interacted with on the Ethereum blockchain that we can verify as legal in nature, such as addresses associated with prominent crypto exchanges and the Beacon Chain deposit contract. This list of addresses was accumulated from Etherscan's display of top addresses by transaction volume that were tagged as verifiably legal entities. In total, this table holds the Etherscan tag, Ethereum

address, and flag of 189 addresses associated with whitelisted entities. Our flag in this table will always be zero due to our assumption that whitelisted addresses will always act legally. Our second table is the table that stores the addresses deemed to be legal in nature by our machine learning model. The data stored in this table will be the address itself, the flag, and the timestamp at which the model classification was made. Due to this table's role of tracking legal addresses, our flag values will always be zero as well. Lastly, we have a table to store all of the addresses that our model deems to be illicit. This table shares all of the same fields as our legal address table, but our flag value in this table will always equal one due to the table's role of tracking illicit addresses. A depiction of our database schema and transaction monitoring system can be viewed below:





Assessment of Model Performance on Real-Time Mempool Transactions

When implementing our ensemble machine learning models on real-time mempool transactions being broadcast to our Ethereum node utilizing our account-based database deployment method, our cohort noticed two major disparities between our expectations and our

results. The first disparity we noticed was that our ensemble model built off of our dataset utilizing addresses which interacted with Coinbase as our legal addresses classified a much greater percentage of the overall transactions it analyzed as illicit than the model built off of randomly sampled addresses as our legal addresses. Over a 5,000-transaction sample size, our Coinbase model determined that 35% of the transactions it was exposed to were illicit in nature, while our random sample model classified 25% of the transactions it encountered as illicit over an equal sample size. After evaluating our models results on mempool transactions, our cohort made the observation that the percentage of transactions deemed to be illicit by our models seems to capture a much higher degree of illicit activity on the Ethereum blockchain than we believe to be feasibly present.

Juxtaposition to External Ethereum Illicit Activity Estimations

The illicit activity figures obtained by our models called for further investigation by our cohort into external sources who have also made estimations of the percentage of illicit activity present on the Ethereum blockchain. We observed that there are a wide range of estimates from multiple external sources. The lowest estimate we could find was made by blockchain analysis firm Chainalysis, which pegged the percentage of illicit transactions throughout all of crypto traffic to be around 0.25%. For our middle ground estimate, the Financial Action Task Force (FATF) has estimated with a high variance that 0.6% to 9.9% of all Bitcoin transactions are illicit. Lastly, for the highest external estimates that our cohort was able to find, Solidus Labs found that 8% of all Ethereum ERC-20 tokens were illicit in nature, and that 12% of all BEP-20 tokens on Binance's blockchain are illicit as well. While these percentages describe different blockchains, these figures serve to give us a ballpark estimate of what our results should be if our

findings are to be considered comparable. Our model estimations of 25% and 35% are about 2-3 times higher respectively than the highest external estimates we could find. The percentages of illicit activity shown by our models also suggests that our thesis that our Coinbase model would perform more effectively than our random sampling model due to more stringent criteria of what constitutes a legal address was proven to be incorrect. While not always the case, our group believes that the random sampling model being closer in proximity to realistic estimations could be indicative of it performing better than the Coinbase model on mempool transactions.

Model Performance Reflection, Improvements, & Limitations

After assessing the performance of our models on real-time mempool transaction data, we have determined that the novel methodologies proposed in aforementioned studies do not accurately identify illicit activity as effectively as once believed. When we juxtapose the estimations of external sources with the illicit activity percentages produced by our model, it is apparent that our model likely possesses a false positive rate that we cannot accept for a real-time use case. Our cohort believes that this issue could stem from many different causes and that there is more work to be done at all stages of the deployment process to refine our approach.

We believe the primary reason the machine learning models created by contemporary researchers failed to perform as well on mempool transactions as they did on their designated testing splits is due to the composition of addresses within our dataset. During our exploratory phase of implementing descriptive statistics analysis on our dataset, we found that many of our addresses labeled as illicit held low aggregate values in features signifying the total Ether balance of the address, the total number of transactions that the address had been implicated in, and the time difference between the first and last recorded transactions for the address, while

these values were typically much higher when evaluating their legal counterparts. Unsurprisingly, these features were among the most important classification determinators identified by our SHAP value and ELI5 feature importance analysis due to the aforementioned stark difference in aggregate feature values. When evaluating the addresses implicated in mempool transactions that were identified as illicit (the majority of which were likely false positives), our cohort realized that our model had essentially come to recognize low activity addresses as illicit and higher activity addresses as operating legally. This undue classification bias is problematic and likely the root of the issue of our ensemble models classifying an infeasibly high proportion of transactions as illicit, as there are many addresses with a low amount of activity that operate legally.

Our cohort believes that this rampant classification of false positives could be remedied by including legal addresses that have low activity volume into our training dataset in order to reduce our models reliance on the volume of transaction activity to make classifications. While our group is adamant that features correlated to transaction activity levels should continue to be included in a models decision-making process, we believe that our models reliance on such measures has caused the high false positive rate on mempool transaction data that has rendered our models unusable for a real-time transaction monitoring use case.

Despite our aforementioned feature engineering speed issue, we also believe a potential improvement to our dataset that would bolster our models efficacy on mempool transactions could be to engineer a greater number of aggregate features per address that cover a wider breadth of transaction activity present on Ethereum. There are fundamental differences between the various transaction types and token standards on the Ethereum blockchain, all of which are encountered when streaming real-time mempool transactions. While we engineered 22 features

to help us make address classifications, these features only aggregated specific information pertaining to simple Ether and ERC-20 transactions conducted by each address, akin to the methodology utilized in research we initially considered to be efficacious. Aside from simple Ether transactions and the ERC-20 token standard, there are many other token standards such as ERC-721 and ERC-1155, which allow for the usage of non-fungible tokens (NFTs) on the Ethereum blockchain, that could be utilized to derive further classification insights from our machine learning models. Our cohort believes that there could remain unexplored valuable features to engineer for each address that would provide even greater conclusively when conducting classification of illicit activity on real-time mempool data that is representative of the entirety of traffic on the Ethereum blockchain.

While limitations such as our feature engineering speed impediment and dubious assumptions made in our flagging criteria and current database deployment strategy have already been discussed in prior sections, there is a final limitation our cohort was forced to address during our implementation process as well. Another issue lies with the Etherscan API's ability to only provide a maximum of 10,000 transactions per query. While this issue isn't relevant for the vast majority of addresses on Ethereum and could be remedied in the future by using multiple API calls when necessary, this is a limitation that increased the difficulty and time required to calculate features based on highly active addresses, which can in some cases have millions of transactions in their history.

Multicategorical Classification: Delineating Illicit Ethereum Transactions

Introduction

Building on the established framework of identifying illicit transactions, this section introduces a multicategorical classification system, moving beyond the binary 'illicit or not' determination to pinpoint the specific type of scam associated with a transaction. This expansion not only broadens the understanding of threats and illicit addresses but also allows for the possibility of more precise safety measures. We also ventured into this multi-categorical classification to attempt to give more depth and meaning into the illicit address features and information we gather, and have it be a new useful addition to the strong machine learning models that we had created to detect licit and illicit addresses.

This was a multi-step process where we began from scraping addresses with their categories from websites, then creating machine learning models to learn from and accurately predict the types of illicit addresses, and then fine-tuning the models using hyperparameter tuning and ensemble models to achieve better accuracy from the models. The models would initially detect between two categories: phishing and non-phishing. This would later change to us further classifying the models between phishing, impersonation, and other_scam categories, and also creating a hierarchical model where the model predicts whether an address is fraudulent or not first before getting into the categorical model.

Methodology & Fraudulent Scam Categories

Part of this new classification system is the addition of the "Scam_Types" column to the existing dataset. This column categorizes the 17 distinct scams into a specific number of categories, which we detail later in this section. This information was scraped from databases that contained reported and flagged illicit Ethereum addresses, leveraging the data available in resources such as Etherscan.io and Chainabuse.com.

Utilizing the expanded dataset allows for an exploration into various scam categories prevalent in Ethereum transactions. Chainabuse.com and Etherscan.io combined had 17 categorized scam types, which is what we used in this section. Here are the 17 categories:

#	Scam Type	#	Scam Type
1	Other Blackmail	11	Sextortion
2	Contract Exploit	12	SIM Swap
3	Other Hack	13	Ransomware
4	Fake Returns	14	Other Investment Scam
5	Airdrop	15	Other
6	Pig Butchering	16	Other Scam
7	Rug Pull	17	Phishing
8	Fake Project		
9	Romance		
10	Donation Scam		

Enhanced Understanding of Threats

This transition to a multicategorical system offers a more granular insight into the different scams, revealing specific patterns and tactics associated with each type of scam.

Recognizing the distinct scam categories enables the development of targeted safety initiatives, allowing preventative measures to address the unique patterns and possible anti-fraud measures presented by analyzing each scam type.

Acquisition of Illicit Accounts by Category

Objective

The main objective in this phase of the project was to extract and store illicit Ethereum addresses to facilitate safety and analysis, aiming to further the understanding of illicit activities taking place in the Ethereum network. Through a careful extraction process, approximately 4,000 addresses categorized into 16 unique scam categories were retrieved, the Chainabuse.com website being the primary data source. The ‘Phishing’ addresses would be extracted and compiled from the Etherscan.io website, as that website had a very large number of phishing categorized addresses, but not many other categories were present. The Chainabuse website itself had nearly 23,000 addresses, but scraping those as well would create a heavily imbalanced dataset in favor of the Phishing category alone.

Methodology

The extraction process demanded a manual login procedure to allow Selenium to scrape past page 10. If the useragent is not logged in, the website will not allow it to go past page 10. where individual pages were navigated to gather pertinent addresses. Various selectors facilitated this process, where XPath, CSS Selector, and Element were employed to locate the specific data on the web pages.

We initially attempted to use BeautifulSoup to scrape the HTML content of the web page, including using the Async function instead of requests, since the requests function from the BeautifulSoup library was not successful in scraping across pages and would give us duplicated addresses. Async also did not help scrape correctly, as it led to the same issue of addresses being duplicated; our sets function would only output addresses from one page. We then switched to Selenium, which did a significantly better job of interacting with the webpage and getting us addresses from each page.

The specific code that helped us retrieve the addresses from each page is as follows: the elements with the CSS class 'create-ResponsiveAddress__text', which contained Ethereum addresses on the page. By using this class, the code was able to identify and extract the Ethereum addresses from each page's HTML content, allowing it to collect the relevant data for further processing.

To enhance the efficiency of data retrieval while avoiding bans, randomized delays and scroll techniques were tested and later utilized. In addition to this, the logging() function replaced the print() function to track errors seamlessly and maintain a clean output. An example of the output of the logging() function can be seen below. After extensive debugging, this function allowed us to pinpoint issues in the scraping process and helped us solve them.

To organize the extracted data methodically for easy access and analysis, and to utilize in our machine learning models, data was stored in a CSV file, specifying each address alongside

the respective page number and scam type. The IMPERSONATION dataset became a focal point in this extraction, where the Selenium bot scraped 221 pages of data seamlessly.

Data

Data acquired through this process included:

- **Website**: Chainabuse.com
- **Total Addresses Extracted**: ~4,000
- **Scam Categories**: 16
- **Data Points / Columns**: Address, Page, Scam Type
- **Example**: IMPERSONATION Dataset had 221 pages of addresses, which was successfully scraped through

Scraping Tools Utilized: Multi-Categorical Classification

Tools Utilized

In this phase, the Selenium tool was utilized on the Chrome browser to automate the browsing process. Here, multiple user agents were employed to mimic various browsers, while emulating a diverse range of browsers, helping to seamlessly scrape the data. This approach ensured the continuous and efficient gathering of data without arousing suspicion from web servers. Note that the scraping performed from Chainabuse was with due respect to their terms of service.

Rotating user agents utilized:

```
USER_AGENTS = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36",
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36",
```

Logging() function output example:

```
Once the browser opens, please login to chainabuse.com.
After logging in, press Enter to start scraping.
INFO:root:Scraped 15 addresses from page 168.
INFO:root:Scraped 15 addresses from page 169.
INFO:root:Scraped 15 addresses from page 170.
INFO:root:Scraped 15 addresses from page 171.
INFO:root:Scraped 15 addresses from page 172.
INFO:root:Scraped 15 addresses from page 173.
INFO:root:Scraped 15 addresses from page 174.
INFO:root:Scraped 15 addresses from page 175.
INFO:root:Scraped 15 addresses from page 176.
```

Data Validation

A stringent data validation process was used here, where checkpoints were established at various stages to identify and eliminate duplicate entries. Sets were utilized to ensure that only unique addresses, i.e no duplicate addresses were imported to the CSV. The Python script is designed to append new data to a CSV file, tracking Ethereum addresses and avoiding duplicate entries. It operates by looping over a set of Ethereum addresses, each associated with a page and scam type. When it encounters a unique address (one not already in a set of known addresses), it adds the address to the CSV file and the set. If the address is a duplicate, it isn't added to the file; instead, a counter tracking the number of times that specific address has appeared is increased by the number.

This ensures the CSV file maintains a record of unique entries while still keeping track of how many duplicates were found.The cleaned data was then archived in a CSV file, ensuring that the data stored was immediately able to be utilized. The cleaned data was then archived in a CSV file, ensuring that the data stored was immediately able to be utilized.

The appended column and finalized CSV would look like this:

ERC20_Avg_Time_B…	Unique_Received_Fr…	Unique_Sent_To_Add…	ERC20_Uniq_Sent_A…	ERC20_Uniq_Rec_A…	Scam_Types
0.0	0	0	0	0	IMPERSONATION
25799522.916666668	0	0	0	1	IMPERSONATION
162903.31666666668	0	0	0	2	IMPERSONATION
0.0	0	0	0	0	IMPERSONATION
44504.51845238095	5	11	7	12	CONTRACT_EXPLOIT
1904.9386473429952	371	360	289	215	OTHER_HACK

Roadblocks and Solutions

During the data extraction process, several hurdles were encountered which led us to consider utilizing Virtual Machines (VM) to increase processing power, thereby accommodating larger datasets with ease. The error handling mechanisms such as logging() was used to manage issues arising during the data scraping process and allow for debugging and error-tracking processes, ultimately helping us devise a path to automated scraping without issues.

Hyperparameter Tuning and Ensemble Modeling for Multi-Categorical Classification

Hyperparameter Tuning: Multi-Categorical Classification

In our study, we utilized several ML models including RandomForest, GradientBoosting, AdaBoost, and Decision Trees to achieve an accurate classification of our Scam_Type categories. We also tested Logistic Regression, KNN Classifiers, and some others but ultimately decided on the 4 mentioned. Our hyperparameter tuning involved setting the RandomForest and Decision Tree models with a number of estimators to 100 and a max depth of 10, and both the GradientBoosting and AdaBoost with 100 estimators and a learning rate of 0.1. This fine-tuning helped us attain our results.

Our initial focus was binary, categorizing data into phishing and non-phishing groups due to a significant data imbalance favoring non-phishing with roughly a 10:1 ratio. Due to this imbalance and the complexity of the dataset, we chose to reduce the number of phishing category addresses to match the number of the remaining categories, and began to train our models with 1,000 phishing addresses versus 1,000 non-phishing addresses. The precision and recall metrics reveal that our model doesn't just favor the dominant class, which indicates a balanced model.

We also performed grid searches individually in an attempt to bring the score using optimal configuration, leveraging the benefit of averaging model biases to boost overall predictability. This approach helped us achieve the scores that we did, as the models were initially achieving scores from roughly 62% to 74% before performing individual grid searches.

Individual scores of the models, utilizing hyper parameter tuning and performing grid search:

Tuned Model Results:

ML Model	Accuracy
Decision Tree	0.782
Gradient Boosting	0.724
AdaBoost	0.664
Random Forest	0.777

Ensemble Modeling: Multi-Categorical Classification

To optimize decision-making in identifying the categories of activities, we used voting and stacking techniques. The ensemble models included the same 4 models, RandomForest,

GradientBoosting, AdaBoost, and Decision Trees. We also kept in mind while attempting this, that we were creating ensemble models on top of ensemble models, all of these 4 models using ensemble methods. We used Voting Classifier and Stacking Classifiers, evaluating based on accuracy, precision, and recall metrics, attaining impressive results with the accuracies being 74.18% and 77.94%, respectively.

Ensemble Techniques Results:

Technique	Accuracy (%)
Voting Classifier	74.2
Stacking Classifier	77.9

- **Voting Classifier**: Combined the predictions of all the above models with 'hard' voting.
- **Stacking Classifier**: Stacked the base models with a logistic regression as the meta-learner.

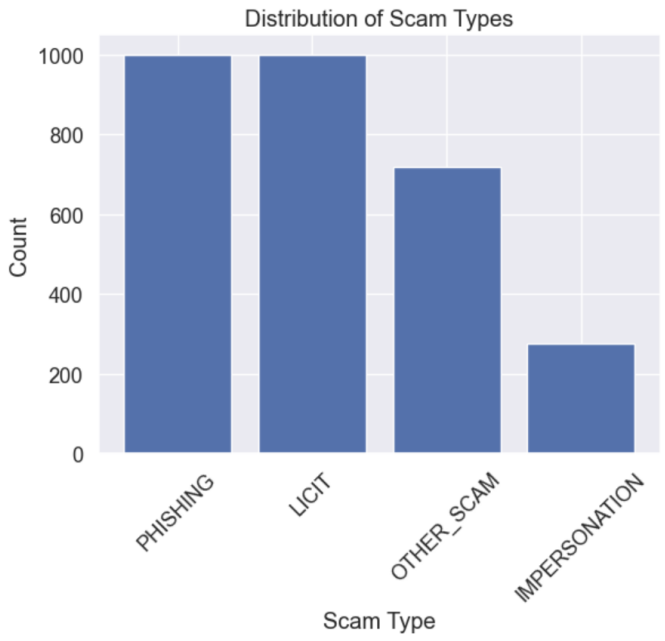
After increasing the max_depth from 10 to 15 on our Stacking Classifier model, we then managed to obtain the highest accuracy of 78.2%. This appears to be the exact score we achieved from our Decision Tree model, and was the highest score we achieved among our models. We also attempted to use RandomizedSearchCV on the Stacking Classifier to achieve up to an 80% score on our model, but since we ran this locally, we did not have enough compute power on our local machines to do so.

Further Classification of Ethereum Addresses

Background & Objective

In the previous stages of this project, the classification of Ethereum addresses was restricted to a binary categorization represented through a FLAG column, with '0' denoting licit and '1' indicating illicit activities. We then added another column, Scam_Types, and while also binary (PHISHING vs NON-PHISHING labeled categories). While this additional insight was useful, where we were able to classify between two categories and not just stay within the licit and illicit categorization, we thought that we may be able to further classify the phishing and non-phishing to impersonation as well.

The dataset here would be imbalanced due to the lack of addresses from the impersonation category, but we thought it may help venture into this area to see if we can get a model to predict this additional category accurately. We also chose to add 1,000 licit addresses and 1,000 illicit, uncategorized addresses into this dataset, to train the model based on these two new categories of “LICIT” and “NA”, respectively. The final goal here would be to include a level of granularity that extends beyond the distinction that’s made currently in the classification of datasets, i.e adding the impersonation category.



Data Integration and Enhancement

Datasets Description

- **Dataset 1**: Contained detailed scam categories aligned with Ethereum addresses.
- **Dataset 2** (test data): Had Ethereum addresses marked as licit or illicit but lacked detailed scam categorizations.

Integration Process

To improve our classification system, we did the following:

- **Scam Types Unification:** To maintain as much a balanced dataset as possible, we grouped the remaining 15 less common scam categories under the label "OTHER_SCAM".
- **Adding the 'IMPERSONATION' Category:** We identified and isolated all the values under the Scam_Type column tagged as "IMPERSONATION" to now have three main categories: PHISHING, NON-PHISHING, and IMPERSONATION.

Further Classification: Model Performance Metrics

Methodology

Moving further, we classified data derived from the ‘merged_dataset.csv’ file (which had all the categories of addresses mentioned earlier). This involved cleaning the data, managing 'NA' and NaN values which were found in the new 1,000 uncategorized illicit addresses (gathered from the previous datasets used in this project), and creating a new ‘Binary_Scam_Type’ column to differentiate between 'LICIT' and 'Illicit' categories. Utilizing

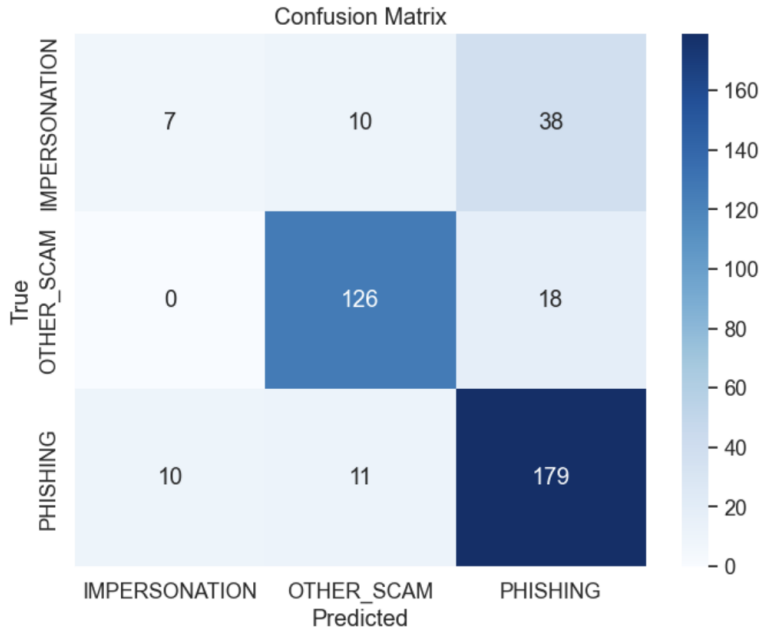
RandomForestClassifier, we separately developed binary and multiclass models which were then observed using precision, recall, and the f1-score. We then utilized heat maps and confusion matrix to visualize the outputs.

Results

We achieved the following results:

Category	Precision	Recall
Impersonation	41%	13%
Other Scam	86%	88%
Phishing	76%	90%

Actual \ Predicted	Impersonation	Other Scam	Phishing
Impersonation	7	10	38
Other Scam	0	126	18
Phishing	10	11	179



The model's overall accuracy stood at a decent 78%, with the averages being as we can see below:

- Unweighted Average across all classes: 68%

- Weighted Average considering class support: 75%

We see here that the impersonation category did not receive a high enough score. This means that there is either discrepancy or that the data was not balanced enough.

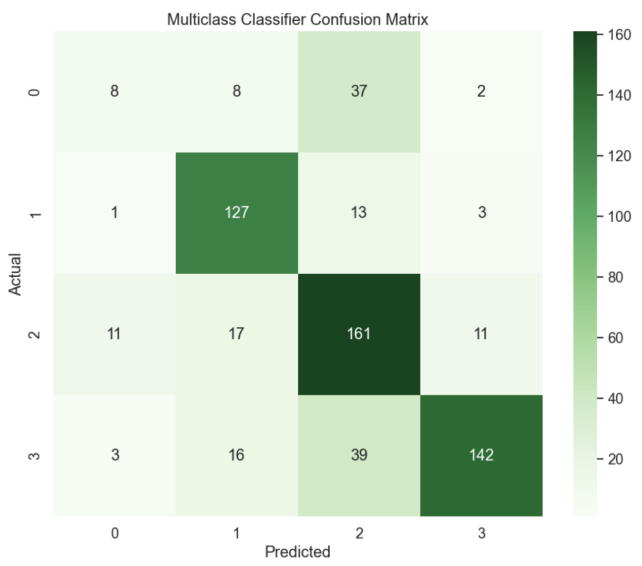
Further Classification: Hierarchical Model

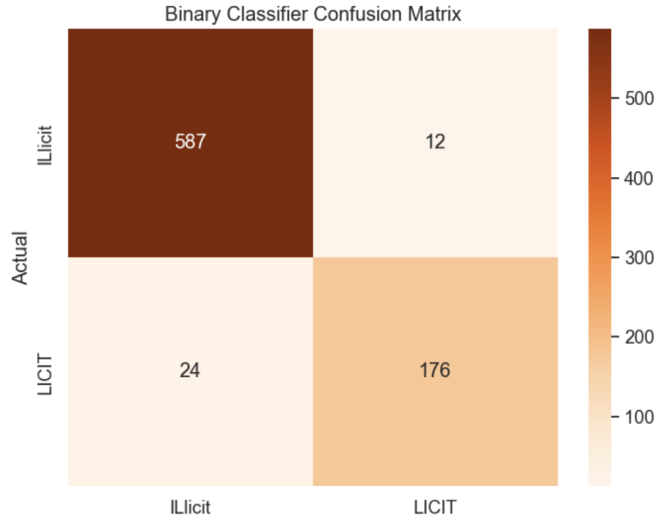
Objective & Methodology

Our objective here was to change the classification strategy to a two-tier hierarchical model. The initial step focused on differentiating between licit and illicit categories. This method involved importing data from 'merged_dataset.csv' and checking for inconsistencies, following which a RandomForestClassifier was engaged for both binary and multi-class classifications. We then utilized heat maps and confusion matrix again to check the model accuracies.

Results

- Binary Classification Accuracy: 95.49%
- Illicit category also showcasing a great performance with a 97% f1-score
- Multiclass Classification Accuracy: 73.12%, with a performance spectrum ranging from an excellent 79% f1-score for ‘Unknown’ to a relatively low 21% for 'Impersonation'.





The results suggest that there was significant success in the initial binary classification phase, and with a large room for improvement in multiclass categorization, especially in enhancing the accuracy for the lower-performing category, 'IMPERSONATION'. The model accuracy can likely be increased using the SMOTE for balancing the impersonation dataset. Due to constraints and kernels crashing and various bugs we ran into while running the code on our local machines, we decided to not use SMOTE for this dataset but it remains as an option to explore in the future.

Multicategorical Classification Conclusion

At the beginning of our project, we started by classifying Ethereum addresses into two simple groups: phishing and non-phishing, using a basic binary system of '0' and '1' to indicate licit and illicit activities. As we delved deeper, we added a new dimension to our classification by introducing an 'impersonation' category and later expanding to include a detailed hierarchical model. This was supported by a dataset balanced with 1000 licit and 1000 illicit uncategorized addresses, allowing us to perform trial and error on various predictive models. Despite facing challenges such as imbalances in the data and the risk of overfitting, the addition of the 'Scam_Type' column marked a significant stride in evolving our approach from a simple binary classification to a more nuanced understanding of different types of scams.

Looking ahead, we aim to further refine and train our models by running the model on VMs, using balancers, and deploying Monte Carlo and similar system performance checking systems. Lastly, we aim to leverage libraries like joblib to cache the model learnings and to create an automated system that learns over time as we are able to extract and feature-engineer more categorized data. This development from a simple start to a more detailed analysis

represents a significant step towards creating a more secure environment in the crypto and blockchain landscape.

Tech Stack

The following graphic displays the various technologies our team utilized in order to build the infrastructure necessary to complete this project. Our four main utilities were our Ethereum Geth Client through Quicknode, the Jupyter Notebook Python environment, a PostgreSQL Google Cloud SQL database, and Power Bi:

Geth	Utilized Geth execution client hosted by Quicknode for mempool data	
Python	**Libraries** web3 psycopg2 pandas joblib etherscan-python sklearn datetime selenium	
PostgreSQL	Google Cloud SQL Database	
Power Bi	Statistical Analysis and Visualization Software	

References

"Specially Designated Nationals and Blocked Persons List (SDN) Human Readable Lists." *Office of Foreign Assets Control | U.S. Department of the Treasury*, ofac.treasury.gov/specially-designated-nationals-and-blocked-persons-list-sdn-human-readable-lists. Accessed 20 Sept. 2023.

"CryptoScamDB." *Home*, cryptoscamdb.org/. Accessed 20 Sept. 2023.

"Farrugia Dataset." *GitHub*, github.com/sfarrugia15/Ethereum_Fraud_Detection/blob/master/Account_Stats/Complete.csv. Accessed 20 Sept. 2023.

Aliyev, Vagif. "Ethereum Fraud Detection Dataset." *Kaggle*, 3 Jan. 2021, www.kaggle.com/datasets/vagifa/ethereum-frauddetection-dataset.

Escobero, Guille. "Ethereum Fraud Dataset." *Kaggle*, 14 Sept. 2022, www.kaggle.com/datasets/gescobero/ethereum-fraud-dataset.

Ethereum (ETH) Blockchain Explorer - Etherscan, etherscan.io/. Accessed 21 Sept. 2023.

Alarab, Ismail, and Simant Prakoonwit. "Effect of Data Resampling on Feature Importance in Imbalanced ..." *Research Gate*, www.researchgate.net/publication/360191975_Effect_of_data_resampling_on_feature_importance_in_imbalanced_blockchain_data_Comparison_studies_of_resampling_techniques. Accessed 21 Sept. 2023.

"Documentation for RPC and REST API, NFT API, Token API." *QuickNode*, www.quicknode.com/docs/welcome?_gl=1%2A1bdjafm%2A_ga%2ANzU1Mzc5NTY1LjE2OTAwNDEwNDc.%2A_ga_DYE4XLEMH3%2AMTY5NTI1MjYzMy4xLjEuMTY5NTI1MjY5NS42MC4wLjA. Accessed 20 Sept. 2023.

"ETH Mempool." *Blockchain.Com*, www.blockchain.com/explorer/mempool/eth. Accessed 20 Sept. 2023.

"Ethereum Development Documentation." *Ethereum.Org*, ethereum.org/en/developers/docs/. Accessed 20 Sept. 2023.

Farrugia, Steven. "Detection of Illicit Accounts over the Ethereum Blockchain." *Expert Systems with Applications*, Pergamon, 17 Feb. 2020, www.sciencedirect.com/science/article/abs/pii/S0957417420301433?via%3Dihub.

Hayes, Adam. "Blockchain Facts: What Is It, How It Works, and How It Can Be Used." *Investopedia*, Investopedia, www.investopedia.com/terms/b/blockchain.asp. Accessed 20 Sept. 2023.

"How Does the New Ethereum Work?" *RSS*, www.preethikasireddy.com/post/how-does-the-new-ethereum-work. Accessed 20 Sept. 2023.

"How to Access Ethereum Mempool." *QuickNode*, 18 Aug. 2023, www.quicknode.com/guides/ethereum-development/transactions/how-to-access-ethereum-mempool.

"Mev Watch." *MEV Watch*, www.mevwatch.info/. Accessed 20 Sept. 2023.

Pahuja, Lavina, and Ahamad Kamal. “EnLEFD‐DM: Ensemble Learning Based Ethereum Fraud Detection Using Crisp ...” *Research Gate*, www.researchgate.net/publication/371696262_EnLEFD-DM_Ensemble_Learning_based_Ethereum_Fraud_Detection_using_CRISP-DM_framework. Accessed 21 Sept. 2023.

“Web3.Py Documentation.” *Gm - Web3.Py 6.9.0 Documentation*, web3py.readthedocs.io/en/stable/. Accessed 20 Sept. 2023.

“What Is the Mempool? - How Blockchain Transactions Work.” *What Is the Mempool? - How Blockchain Transactions Work*, Blocknative, www.blocknative.com/blog/mempool-intro. Accessed 20 Sept. 2023.

Chainabuse Data Extraction

“Ethereum Scam Reports.” *Chainabuse*, www.chainabuse.com/chain/ETH. Accessed 20 Sept. 2023.

SHAP value

“Shap.Treeexplainer.” Shap.TreeExplainer - SHAP Latest Documentation, www.shap-lrjball.readthedocs.io/en/latest/generated/shap.TreeExplainer.html. Accessed 20 Sept. 2023.

Gopinath, Divya. “The Shapley Value for ML Models.” Medium, Towards Data Science, 26 Oct. 2021, www.towardsdatascience.com/the-shapley-value-for-ml-models-f1100bff78d1.

ELI5 Value

Meena, Priyanka. “Demystifying Model Interpretation Using ELI5.” Analytics Vidhya, 28 Sept. 2022, www.analyticsvidhya.com/blog/2020/11/demystifying-model-interpretation-using-eli5/.

“ELI5 Top-Level API¶.” ELI5 Top-Level API - ELI5 0.11.0 Documentation, eli5.readthedocs.io/en/latest/autodocs/eli5.html. Accessed 20 Sept. 2023.

Hyperparameter Tuning

Tuning the Hyper-Parameters of an Estimator.” Scikit, scikit-learn.org/stable/modules/grid_search.html. Accessed 20 Sept. 2023.

Brownlee, Jason. “Binary Classification Tutorial with the Keras Deep Learning Library.” MachineLearningMastery.Com, 5 Aug. 2022, www.machinelearningmastery.com/binary-classification-tutorial-with-the-keras-deep-learning-library/.

Nyuytiymbiy, Kizito. “Parameters and Hyperparameters in Machine Learning and Deep Learning.” *Medium*, Towards Data Science, 28 Mar. 2022, www.towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac.

Ensembling Modelling

Kalirane, Mbali. “Ensemble Learning Methods: Bagging, Boosting and Stacking.” *Analytics Vidhya*, 9 Aug. 2023, www.analyticsvidhya.com/blog/2023/01/ensemble-learning-methods-bagging-boosting-and-stacking/.

Lutins, Evan. “Ensemble Methods in Machine Learning: What Are They and Why Use Them?” *Medium*, Towards Data Science, 2 Aug. 2017, www.towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f.

Kanwar, Ankush. “ML: Voting Classifier Using Sklearn.” *GeeksforGeeks*, GeeksforGeeks, 25 Nov. 2019, www.geeksforgeeks.org/ml-voting-classifier-using-sklearn/.

Afolabi, Samson. “Ensemble Methods: Comparing Scikit Learn’s Voting Classifier to the Stacking Classifier.” *Medium*, Towards Data Science, 17 Sept. 2022, www.towardsdatascience.com/ensemble-methods-comparing-scikit-learns-voting-classifier-to-the-stacking-classifier-f5ab1ed1a29d.