Ethereum Blockchain Transaction Analysis

Nastaran Darabi, Sanmitha Shetty, Shreyash Kadam, Dhiraj Shelke, Frank Wang, Victor Escudero
CS Department, University of Illinois at Chicago, IL, USA
Emails: {ndarab2, sshet32, skada29, dshel4, xwang246, vescu2 }@uic.edu

Abstract*—The blockchain is an emerging field, and analyzing Ethereum transactions can provide valuable insights into transaction behaviors and gas fees. Our goal is to understand these behaviors by identifying anomalies such as irregular gas fees or potential fraudulent activities. By examining transaction patterns from the last 20 blocks of Ethereum, we aim to reveal how transactional behaviors change over short periods. The project will begin by formulating questions about Ethereum transactions. After collecting and analyzing the data using techniques like clustering and anomaly detection, the final deliverable will be an interactive visual dashboard. This dashboard will enable users to explore trends and correlations, filtering data by gas fees, block numbers, and transaction types. Our code is available here: *https://github.com/cs418-fa24/project-check-in-team-2

I. Introduction

The blockchain is a rapidly growing field, and analyzing Ethereum transactions can provide valuable insights into transaction behaviors, gas fees, and patterns that may signal trends or anomalies. Our objective is to understand transaction behaviors on the Ethereum blockchain, particularly by identifying anomalies such as irregular gas fees or potential fraudulent activities. By examining transaction patterns from the last 20 blocks of Ethereum, this project aims to determine how transactional behaviors change over short periods and whether some patterns are not immediately visible [1]–[5].

Data will be collected through web scraping from the Etherscan block explorer at *https://etherscan.io/txs*. Specifically, we will extract data from the last 20 Ethereum blocks, including transaction hashes, gas fees, sender and receiver addresses, and timestamps. This data is publicly available and can be gathered in real-time. Each block typically contains hundreds of transactions, so we expect to handle approximately 2,000 to 5,000 transactions for analysis. Given the nature of web scraping, this data will be organized in tables featuring various attributes, such as transaction hash, gas fees, and addresses.

This project will start by posing questions about Ethereum transactions. We will collect transaction data from the last 20 blocks using web scraping techniques. After gathering the data, we will analyze it to identify patterns and anomalies. The analysis will employ techniques such as clustering, anomaly detection, and trend analysis. The final deliverable will be an interactive visual dashboard that allows users to explore trends and correlations within the Ethereum blockchain data. Users will be able to filter the data by gas fees, block numbers, and transaction types.

Key Questions We Aim to Answer:

1) What patterns or trends can be identified in Ethereum transactions?
2) Are there specific metrics that can help predict movements in Ethereum prices?

II. Data

A. Data Collection

Data is collected from *https://etherscan.io/txs* using the automation tool Selenium, which facilitates the extraction of key information related to Ethereum transactions. This process focuses specifically on gathering various attributes including transaction hashes, block numbers, timestamps, wallet addresses, and transaction fees.

Through this collection, we obtain valuable insights into the patterns and behaviors of Ethereum transactions. The data allows for in-depth analysis of several aspects, such as the frequency of transactions, the values associated with each transaction, and the fees incurred during these transactions.

By employing Selenium, we streamline the process of automating data collection, which enhances efficiency and accuracy. This automation involves systematically navigating the Etherscan website, extracting the necessary transactional data, and organizing it for further analysis. The end result is a comprehensive dataset that supports a thorough exploration of transaction dynamics on the Ethereum network.

B. Data Cleaning

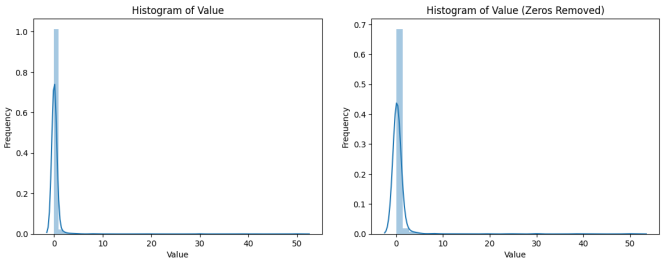

Fig. 1: Data Cleaning

We have completed these data-cleaning steps:

- **Incomplete Records Removal:** Rows with missing or incomplete data are removed to ensure a consistent dataset.
- **Standardizing Columns:** Timestamps are standardized, and columns that are not useful for analysis (e.g., transaction hash, method, age, from, to) are dropped to streamline the dataset.
- **Outliers Removal:** Outliers are identified and removed based on values outside 3 standard deviations from the mean, assuming a normal distribution.

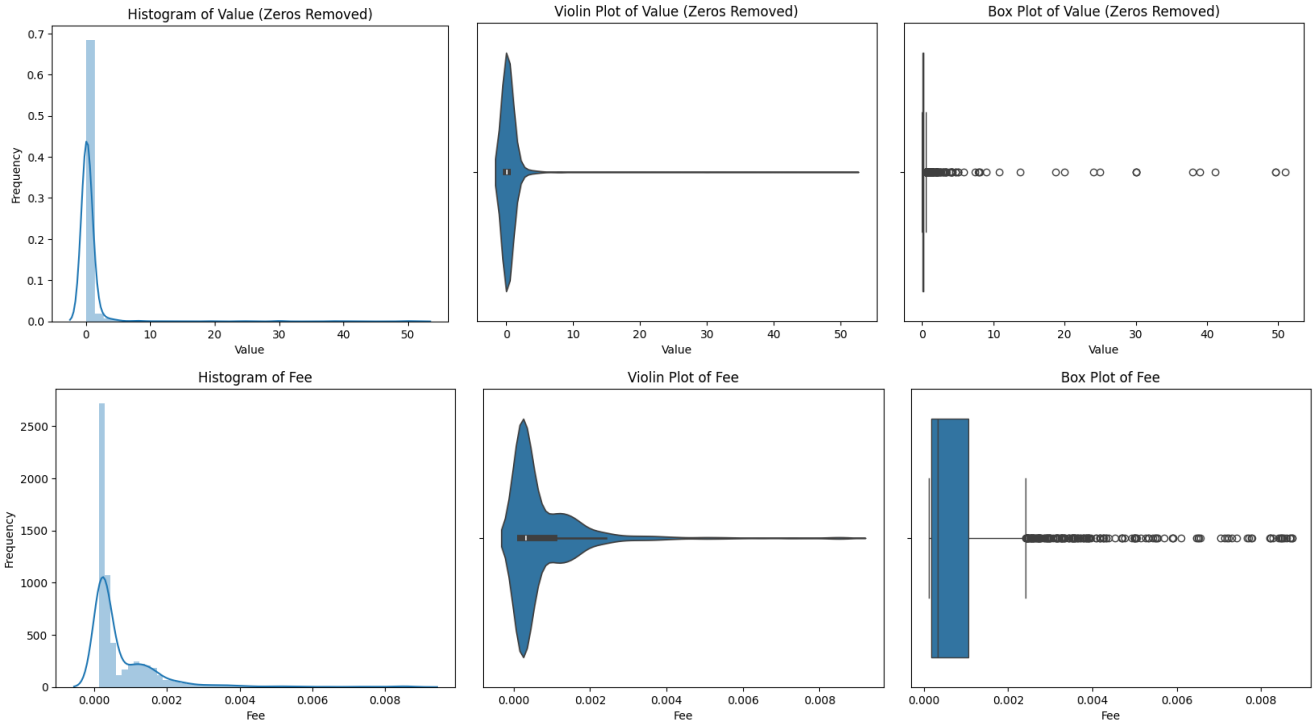

Fig. 2: Visualization of transaction data distribution using histograms, violin plots, and box plots. The histogram illustrates the frequency distribution of transaction fees, with bin sizes determined by the square root of the dataset size for optimal clarity and balance. The violin plot provides insights into the density and skewness of transaction values and fees, while the box plot highlights central tendencies, variability, and the presence of outliers. Together, these visualizations offer a robust analysis of transaction behavior, enabling the identification of significant patterns and anomalies.

Once the data is cleaned, we calculate basic statistics to gain valuable insights from it.

- **Mean and Standard Deviation:** The mean and standard deviation for 'value' and 'txnFee' columns give insight into the central tendency and variability of transaction values and fees after outlier removal.
- **Histogram Binning Strategy:** Using the square root of the data size as the bin count is a good approach for balancing visual interpretability and detail without overfitting to outliers.

III. Data Visualization & Distribution Analysis

We are implementing a method for determining the bin size based on the square root of the dataset size. This approach is effective in creating a balanced number of optimally sized bins. It allows us to achieve a robust and visually clear representation of the data distribution, ensuring that the results are interpretable. Additionally, this method helps reduce the influence of outliers, which can skew the data representation and lead to misleading interpretations.

Histograms of Transaction Fees: The histogram for transaction values is important in our analysis, as it enables us to observe the spread and frequency of varying transaction sizes within the dataset. By carefully examining the histogram after removing outliers and excluding zero-value transactions, we can effectively unveil significant patterns in transaction behavior. This focused analysis reveals notable insights, highlighting that transaction values and fees typically are skewed. Such detailed examination is crucial for understanding typical transaction sizes and the associated fees, which can inform decision-making and strategy development.

Violin Plot: This plot offers a comprehensive view of the distribution and density of transaction values and fees. It helps identify whether the data follows a normal distribution or exhibits skewness and illustrates where the data clusters.

Box Plot: This plot highlights central values, variability, and outliers within the transaction values and fees range. It visually emphasizes extreme values and the spread of the data for further analysis.

IV. Exploratory Data Analysis (EDA)

Our exploratory data analysis (EDA) aims to gain insights into the dataset structure and uncover key characteristics, particularly focusing on transaction fees and their correlation with other features. By employing a variety of visualizations, we illustrate important trends and distributions that reveal significant patterns in the data. To begin our analysis, we performed an initial exploration of the dataset by calculating descriptive statistics and visualizing numeric feature distributions and relationships. A summary table was generated, providing an overview of the dataset, including column names, data types, and the count of null values. For numeric columns, key statistics such as mean, standard deviation, minimum, and maximum values were computed to capture the range and variability of the data. To complement this summary, histograms were used to visualize the distributions of numeric features,

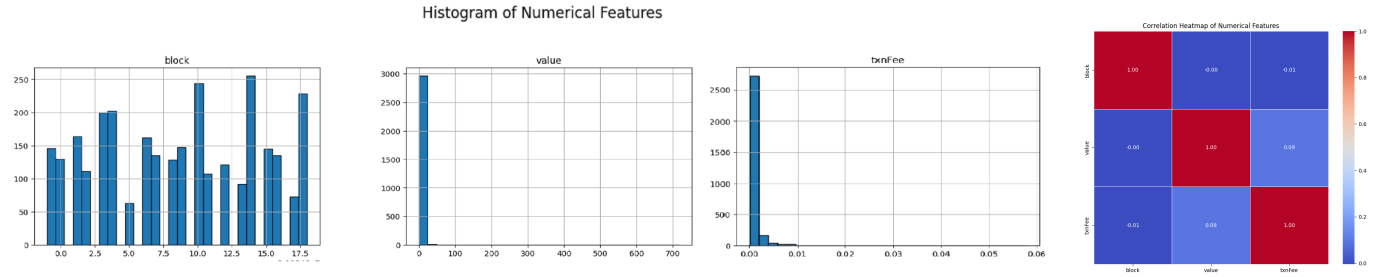

Fig. 3: **Comprehensive dataset overview and initial exploratory analysis:** Histograms visualize the distributions of numeric features, identifying spread, skewness, and outliers. A correlation heatmap highlights relationships between features, uncovering patterns and dependencies within the data.

Column Names	Null Values	Unique Values	Data Types	Mean	Standard Deviation	Min	Max
Method	0	293	object	NaN	NaN	NaN	NaN
Block	0	20	int64	2.112491×10^7	5.803496	2.112490×10^7	2.112492×10^7
Value	0	1207	float64	1.451123	20.199170	0.000000	717.000000
txnFee	0	2155	float64	0.000962	0.002638	0.000126	0.057719

TABLE I: **EDA Summary Table:** Overview of key statistics and metadata for the dataset. Includes column names, null values, unique values, data types, mean, standard deviation, minimum, and maximum values for numeric columns.

offering insights into their spread and identifying potential skewness or outliers. Additionally, a correlation heatmap was created to highlight relationships between features, aiding in the detection of patterns or dependencies within the data. This combined approach ensures a comprehensive understanding of the dataset's structure and facilitates targeted downstream analyses.

A. Transaction Fee Distribution

We used a histogram overlaid with a kernel density estimate (KDE) to analyze the distribution of transaction fees. This visualization effectively illustrates the spread of transaction fees, highlighting both the most common values and the presence of outliers. By examining this plot, we identified typical ranges of transaction fees and detected anomalies that may indicate irregularities or exceptional cases in transaction behavior. This analysis serves as a basis for further investigation into the underlying factors that influence transaction costs.

B. Block Number Correlation

To examine how transaction fees change with different block numbers, we created a scatter plot that displays transaction fees in relation to block numbers. This visualization allows us to identify potential trends or patterns, such as whether transaction fees increase, decrease, or stay consistent across various blocks. Understanding this relationship is essential for evaluating the dynamics of fee structures within the dataset and pinpointing any block-specific factors that may influence costs.

C. Average Transaction Fee per Block

A line plot illustrating the average transaction fee per block provides a clear perspective on trends in transaction costs over time. By aggregating fees within each block and plotting these trends, we can identify periods with higher or lower transaction fees. This analysis is especially useful for detecting systematic changes or anomalies in transaction fee trends that may correlate with external factors or shifts in the network's behavior.

These visualizations offer a thorough understanding of transaction fees within the dataset. They reveal the overall distribution and variability of fees, as well as relationships and trends related to block numbers and time patterns. These insights are crucial for identifying systemic behaviors, optimizing transaction strategies, and making informed decisions about fee management in blockchain systems.

V. Hypothesis Testing Visualization

To test the hypothesis that "Higher gas fees indicate higher transaction volume and network congestion," we analyzed the relationship between gas fees and transaction volume across different traffic conditions. This analysis aimed to uncover patterns and potential correlations that could validate or challenge the hypothesis.

To explore this relationship, we calculated each block's average gas fee and transaction volume, where transaction volume is defined as the count of transactions within a block. We categorized the blocks into three traffic tiers:

- **Peak Traffic:** Blocks with transaction volumes in the top 25% of the dataset.
- **Low Traffic:** Blocks with transaction volumes in the bottom 25%.
- **Normal Traffic:** Blocks with transaction volumes between these thresholds.

Using this categorization, we created a scatter plot to visualize the relationship between transaction volume and average gas fees. Each point in the plot represents a block, with its position determined by the block's average gas fee and transaction volume. The points are color-coded based on the traffic tier to differentiate between peak, low, and normal traffic periods. The scatter plot revealed mixed findings:

- **Peak Traffic Periods:** Several high-transaction-volume blocks were associated with higher gas fees, supporting the hypothesis. These instances suggest that higher

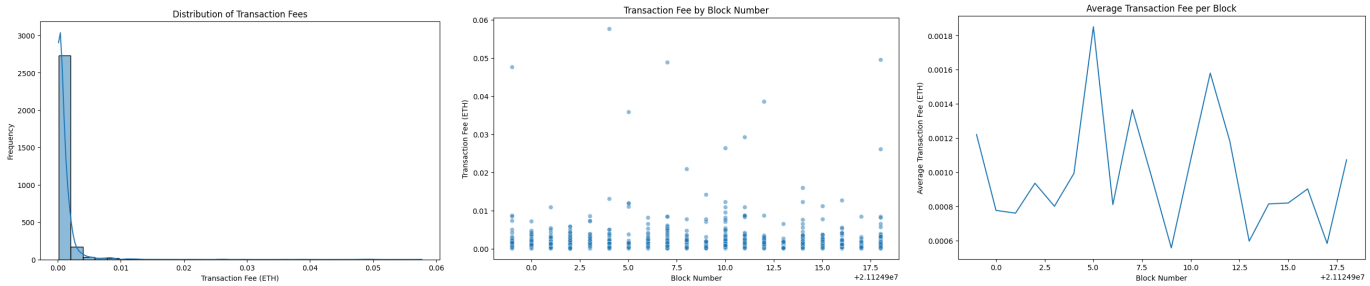

Fig. 4: **Exploratory Data Analysis (EDA) visualizations of transaction fees:** (a) Histogram with kernel density estimate (KDE) illustrating the distribution of transaction fees, highlighting typical values and outliers. (b) Scatter plot showing the relationship between transaction fees and block numbers, revealing potential correlations and block-specific patterns. (c) Line plot depicting the trend of average transaction fees per block, identifying temporal changes and trends in transaction costs.

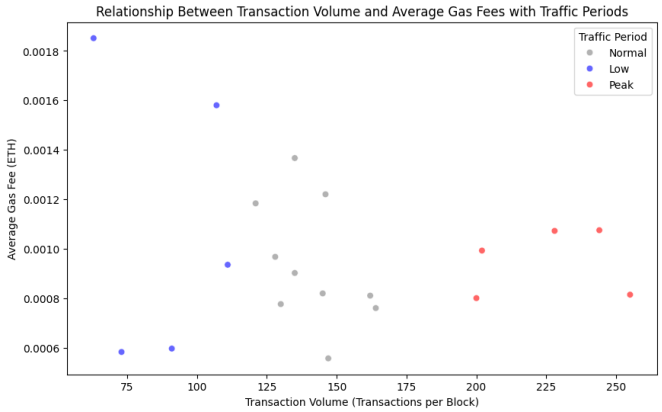

Fig. 5: Scatter plot illustrating the relationship between transaction volume and average gas fees across traffic tiers. Each point represents a block, categorized into 'Peak' , 'Low', or 'Normal' traffic periods. The plot highlights that while higher gas fees are often associated with peak traffic periods, high fees can also occur during low traffic periods, and normal traffic blocks exhibit significant variability.

transaction volumes may contribute to increased network congestion and, consequently, elevated gas fees.

- **Low Traffic Periods:** Interestingly, some blocks in the low traffic tier also displayed high gas fees. This indicates that factors other than transaction volume—such as temporary network congestion, prioritized transactions, or fee structure adjustments—might influence gas fees.
- **Normal Traffic Periods:** Blocks categorized as normal traffic showed a wide range of gas fees, with no distinct linear correlation between transaction volume and gas fees. This variability suggests the influence of additional factors that disrupt the direct relationship between transaction volume and gas fees.

While the visualization supports the hypothesis to some degree—indicating a trend of higher gas fees during peak traffic periods—it also reveals complexities that prevent a straightforward conclusion. The occurrence of high gas fees during low traffic times, along with fluctuations in normal traffic blocks, suggests that factors beyond transaction volume alone influence network congestion and gas fee dynamics. To enhance our understanding, further statistical analysis is needed. This could involve regression modeling to account for potential confounding variables, time-series analysis to examine temporal trends or additional data concerning network conditions and fee structures to establish causal relationships.

VI. Machine Learning Analysis

We initiated a preliminary machine learning (ML) analysis to better understand the data and improve predictive capabilities. This section outlines the key steps undertaken, including feature engineering, model selection, hyperparameter tuning, and evaluation metrics.

A. Feature Engineering

To enhance the predictive power of the models, we incorporated additional features and engineered new ones. These include:

- **Average Transaction Value and Gas Price:** Adding these contextual features provides a more comprehensive understanding of the data.
- **Rolling Averages:** Rolling averages of transaction volume or fees over varying window sizes were calculated to capture trends and recent patterns.
- **Transaction Value Ratios:** Ratios between transaction volume and average transaction value or fees were created to highlight spikes or abnormalities in the dataset.

B. Model Selection

We explored multiple regression models to determine the most effective approach for prediction:

- **Random Forest Regressor:** This model was chosen for its ability to capture nonlinear relationships and handle complex feature interactions.
- **Gradient Boosting Regressor (e.g., XGBoost):** Known for its strong performance on tabular data, this model was tested for its ability to balance bias and variance effectively.
- **Lasso and Ridge Regression:** These models were included for their regularization properties, which are useful in preventing overfitting, especially on smaller datasets.

C. Hyperparameter Tuning

To optimize model performance, hyperparameter tuning was performed using cross-validation. This ensured robust evaluation and fine-tuning of parameters to maximize accuracy and generalization.

Model	MAE	MSE	RMSE	R^2
Linear Regression	12.897428	297.946121	17.261116	0.897390
Ridge Regression	16.724501	355.498103	18.854657	0.877570
Lasso Regression	16.507441	355.025766	18.842127	0.877733
Random Forest	30.807500	1234.710025	35.138441	0.574779
Best Parameters for Random Forest: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 200}				
Best Random Forest Model Performance: MAE: 30.31875 RMSE: 34.82251 R^2: 0.58239				

TABLE II: **Model Performance Comparison:** Evaluation of different regression models using Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2). The table also includes the best parameters and performance metrics for the Random Forest model.

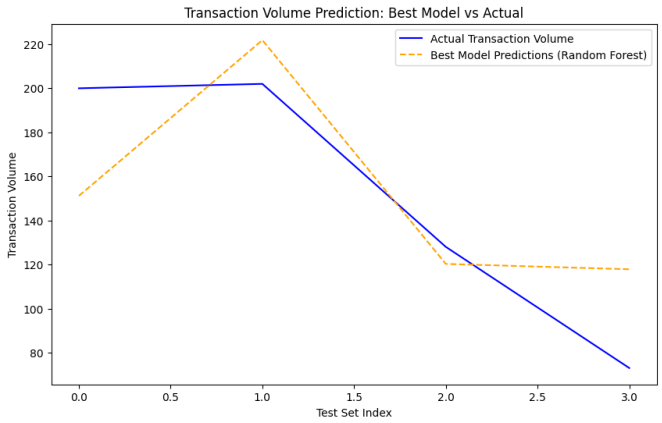

Fig. 6: **Comparison of actual transaction volume and predictions from the best model (Random Forest):** The plot highlights a significant discrepancy, with the model's predictions failing to capture the upward trend in transaction volume and remaining relatively flat and lower than the actual values. This suggests that the model requires further refinement to better align with the observed data patterns.

D. Evaluation Metrics

To assess model performance comprehensively, multiple metrics were used:

- **Mean Absolute Error (MAE):** Measures the average magnitude of errors, providing an intuitive understanding of prediction accuracy.
- **Mean Squared Error (MSE):** Penalizes larger errors more heavily, emphasizing outliers.
- **Root Mean Squared Error (RMSE):** The square root of MSE, offering interpretability in the original units of the target variable.
- **R-squared (R^2):** Indicates the proportion of variance in the target variable explained by the model, providing a measure of goodness-of-fit.

These steps establish a foundation for further refinement and optimization of the machine learning models, ensuring a comprehensive approach to predictive analysis.

VII. Conclusion

This study provided an in-depth analysis of Ethereum blockchain transactions, focusing on identifying patterns, trends, and anomalies. Through systematic data collection and cleaning, we established a robust dataset from the last 20 blocks of Ethereum, enabling the exploration of transaction behaviors and their relationship with gas fees. Our visualization techniques, including histograms, scatter plots, and line plots, revealed significant insights into the distribution of transaction fees, correlations with block numbers, and temporal trends in transaction costs.

The hypothesis testing uncovered the complex relationship between transaction volume and gas fees, showing that while higher transaction volumes often correspond to elevated fees, other factors, such as network congestion and fee structure changes, play a significant role. The variability observed in "Normal" traffic periods and high fees during low traffic periods underscores the need for further statistical analysis to establish causal relationships.

Preliminary machine learning analysis, incorporating feature engineering and testing multiple regression models, highlighted challenges in predicting transaction volumes accurately. Although models like Random Forest showed some promise, their predictions failed to fully capture the upward trends in transaction volume, indicating the necessity for further refinement, hyperparameter tuning, and potentially more advanced modeling approaches.

VIII. Peer Review

All team members contributed equally to the work during every phase of the project. Detailed contributions are outlined below:

1) Data Collection and Web Scraping: Nastaran Darabi & Shreyash Kadam & Sanmitha Shetty
2) Data Cleaning and Preprocessing: Frank Wang & Victor Escudero & Dhiraj Shelke
3) Exploratory Data Analysis (EDA) and Data Sampling: Frank Wang & Victor Escudero & Dhiraj Shelke
4) Machine Learning Application: Nastaran Darabi & Shreyash Kadam & Sanmitha Shetty
5) Data Visualization and Reporting: All team members
6) Conclusion, Final Presentation, and Project Documentation: All team members

References

[1] M. Hasan, M. S. Rahman, H. Janicke, and I. H. Sarker, "Detecting anomalies in blockchain transactions using machine learning classifiers and explainability analysis," *Blockchain: Research and Applications*, p. 100207, 2024.

[2] C. Butler and M. Crane, "Blockchain transaction fee forecasting: a comparison of machine learning methods," *Mathematics*, vol. 11, no. 9, p. 2212, 2023.

[3] V. Patel, L. Pan, and S. Rajasegarar, "Graph deep learning based anomaly detection in ethereum blockchain network," in *International conference on network and system security*. Springer, 2020, pp. 132–148.

[4] S. Jamwal, J. Cano, G. M. Lee, N. H. Tran, and N. Truong, "A survey on ethereum pseudonymity: Techniques, challenges, and future directions," *Journal of Network and Computer Applications*, p. 104019, 2024.

[5] J. Ahn, E. Yi, and M. Kim, "Ethereum 2.0 hard fork: Consensus change and market efficiency," *The Journal of The British Blockchain Association*, 2024.

APPENDIX

The following code was used for the Analysis:

```
!pip install selenium

from bs4 import BeautifulSoup as bs
import requests as req
from selenium.webdriver.common.action_chains import ActionChains
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import pandas as pd
import math
import time
from tqdm import tqdm
import re
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from scipy.stats import norm
import numpy as np
from math import sqrt
import random

"""We have disabled the data collection process by commenting it out, as the data has already been saved in a CSV file."""

# # Initialize WebDriver
# driver = webdriver.Chrome()

# # Navigate to the blocks page to retrieve the latest block ID
# driver.get("https://etherscan.io/blocks")
# time.sleep(1)
# lastBlock = driver.find_element(by=By.XPATH,
    value="//*[@id=\"content\"]/section[2]/div[2]/div[2]/table/tbody/tr[1]/td[1]/a")
# lastBlockID = int(lastBlock.text)

# # Generate URLs for the last 20 blocks
# blockURLs = []
# for blockID in range(lastBlockID, lastBlockID-20, -1):
#     blockURLs.append("https://etherscan.io/txs?block=" + str(blockID))

# # Function to read each row's data in the transactions table
# def readRow(rowIndex):
#     row = {}
#     xPath = "//*[@id=\"ContentPlaceHolder1_divTransactions\"]/div[2]/table/tbody/tr[" + str(rowIndex) + "]/"
#     row["txnHash"] = driver.find_element(by=By.XPATH, value=xPath + "td[2]/div/span/a").text
#     row["method"] = driver.find_element(by=By.XPATH, value=xPath + "td[3]/span").text
#     row["block"] = driver.find_element(by=By.XPATH, value=xPath + "td[4]/a").text
#     row["age"] = driver.find_element(by=By.XPATH, value=xPath + "td[6]/span").text
#     row["from"] = driver.find_element(by=By.XPATH, value=xPath + "td[8]/div/a[1]").get_attribute("href").split("/")[-1]
#     row["to"] = driver.find_element(by=By.XPATH, value=xPath + "td[10]/div").find_elements(by=By.CSS_SELECTOR,
    value="a")[-1].get_attribute("data-clipboard-text")
#     row["value"] = driver.find_element(by=By.XPATH, value=xPath + "td[11]/span").text
#     row["txnFee"] = driver.find_element(by=By.XPATH, value=xPath + "td[12]").text
#     return row

"""*We used time.sleep(1) to avoid being blocked by the website.*"""

# # Collect data for each block
# table = []
# for blockUrl in tqdm(blockURLs, desc="Collecting Data Blocks ("):
#     driver.get(blockUrl)
#     tranCount = int(driver.find_element(by=By.XPATH,
    value="//*[@id=\"ContentPlaceHolder1_divDataInfo\"]/div/div[1]/span").text.split(" ")[3])
#     pageCount = math.ceil(tranCount / 50)
#     for pageIndex in range(1, pageCount + 1):
#         url = blockUrl + "&p=" + str(pageIndex)
#         driver.get(url)
#         time.sleep(1)
#         rowBound = (tranCount - (pageCount - 1) * 50 + 1) if (pageIndex == pageCount) else 51
#         for rowIndex in range(1, rowBound):
#             table.append(readRow(rowIndex))
#     time.sleep(1)

"""*We're creating a CSV file to use later, so we don't have to run this code again.*"""

# Convert to DataFrame and save to CSV
# dataFrame = pd.DataFrame(table)
# dataFrame.to_csv("DataFrame.csv", index=False)
```

```
77
78 # # Close the driver
79 # driver.quit()
80
81 """ Data Cleaning"""
82
83 cleanedDataFrame = pd.read_csv("DataFrame.csv")
84 cleanedDataFrame = cleanedDataFrame.drop(["txnHash", "method", "age", "from", "to"], axis=1)
85
86 """*We are changing the data type of the 'Value' and 'txnFee' columns to float.*"""
87
88 cleanedDataFrame["txnFee"] = cleanedDataFrame["txnFee"].apply(lambda x: float(x))
89 pattern = r"(\d+\.?\d*)"
90 cleanedDataFrame["value"] = cleanedDataFrame["value"].apply(lambda x: float(re.findall(pattern, x)[0]))
91
92 """*We are removing outliers by identifying and excluding data points that fall outside the range of 3 standard
       deviations from the mean, based on the assumption that the dataset follows a normal distribution.*"""
93
94 threshold = 3
95 columnsForProcess = ['value', 'txnFee']
96 outliersMask = ~((cleanedDataFrame[columnsForProcess] - cleanedDataFrame[columnsForProcess].mean()).abs() > threshold *
       cleanedDataFrame[columnsForProcess].std()).any(axis=1)
97 cleanedDataFrameWithoutOutlier = cleanedDataFrame[outliersMask]
98
99 """Basic Statistics"""
100
101 valueMean = cleanedDataFrameWithoutOutlier['value'].mean()
102 valueStd = cleanedDataFrameWithoutOutlier['value'].std()
103 feeMean = cleanedDataFrameWithoutOutlier['txnFee'].mean()
104 feeStd = cleanedDataFrameWithoutOutlier['txnFee'].std()
105 print("Value Column:")
106 print("Mean:", valueMean)
107 print("Standard Deviation:", valueStd)
108 print("\ntxnFee Column:")
109 print("Mean:", feeMean)
110 print("Standard Deviation:", feeStd)
111
112 """Data Visualization & Distribution Analysis"""
113
114 binValue=int(sqrt(len(cleanedDataFrameWithoutOutlier["value"])))
115 sns.distplot(cleanedDataFrameWithoutOutlier["value"],bins=binValue)
116 plt.title('Histogram of Value')
117 plt.xlabel('Value')
118 plt.ylabel('Frequency')
119 plt.show()
120
121 """Removing Transactions with zero value:"""
122
123 removedZeroesDF = cleanedDataFrameWithoutOutlier["value"]
124 removedZeroesDF = removedZeroesDF[removedZeroesDF != 0]
125 removedZeroesDF = removedZeroesDF.reset_index()
126
127 binValue=int(sqrt(len(removedZeroesDF["value"])))
128 sns.distplot(removedZeroesDF["value"],bins=binValue)
129 plt.title('Histogram of Value (Zeros Removed)')
130 plt.xlabel('Value')
131 plt.ylabel('Frequency')
132 plt.show()
133
134 binValue=int(sqrt(len(cleanedDataFrameWithoutOutlier["txnFee"])))
135 sns.distplot(cleanedDataFrameWithoutOutlier["txnFee"],bins=binValue)
136 plt.title('Histogram of Fee')
137 plt.xlabel('Fee')
138 plt.ylabel('Frequency')
139 plt.show()
140
141 """Violin and Box Plots for Transaction Value and Fees:"""
142
143 sns.violinplot(x=cleanedDataFrameWithoutOutlier["value"])
144 plt.xlabel("Value")
145 plt.title("Violin Plot of Value")
146 plt.show()
147
148 sns.violinplot(x=removedZeroesDF["value"])
149 plt.xlabel("Value")
150 plt.title("Violin Plot of Value (Zeros Removed)")
151 plt.show()
152
153 sns.violinplot(x=cleanedDataFrameWithoutOutlier["txnFee"])
154 plt.xlabel("Fee")
155 plt.title("Violin Plot of Fee")
156 plt.show()
157
```

```
sns.boxplot(x=cleanedDataFrameWithoutOutlier["value"])
plt.xlabel("Value")
plt.title("Box Plot of Value")
plt.show()

sns.boxplot(x=removedZeroesDF["value"])
plt.xlabel("Value")
plt.title("Box Plot of Value (Zeros Removed)")
plt.show()

sns.boxplot(x=cleanedDataFrameWithoutOutlier["txnFee"])
plt.xlabel("Fee")
plt.title("Box Plot of Fee")
plt.show()

"""Exploratory Data Analysis (EDA)"""

cleanedDataFrame = pd.read_csv("DataFrame.csv")
cleanedDataFrame = cleanedDataFrame.drop(["txnHash", "age", "from", "to"], axis=1)

# Convert 'txnFee' and 'value' columns to numeric types, handling any potential formatting issues
cleanedDataFrame["txnFee"] = cleanedDataFrame["txnFee"].apply(lambda x: float(x))
pattern = r"(\d+\.?\d*)"
cleanedDataFrame["value"] = cleanedDataFrame["value"].apply(lambda x: float(re.findall(pattern, x)[0]))

# 1. Transaction Fee Distribution
plt.figure(figsize=(10, 6))
sns.histplot(cleanedDataFrame["txnFee"], bins=30, kde=True)
plt.title("Distribution of Transaction Fees")
plt.xlabel("Transaction Fee (ETH)")
plt.ylabel("Frequency")
plt.show()

# 2. Block Number Correlation with Transaction Fee
plt.figure(figsize=(10, 6))
sns.scatterplot(data=cleanedDataFrame, x="block", y="txnFee", alpha=0.5)
plt.title("Transaction Fee by Block Number")
plt.xlabel("Block Number")
plt.ylabel("Transaction Fee (ETH)")
plt.show()

# 3. Line plot of average transaction fee per block
# Group by block and calculate the mean transaction fee for each block
block_fee_avg = cleanedDataFrame.groupby("block")["txnFee"].mean().reset_index()

plt.figure(figsize=(10, 6))
sns.lineplot(data=block_fee_avg, x="block", y="txnFee")
plt.title("Average Transaction Fee per Block")
plt.xlabel("Block Number")
plt.ylabel("Average Transaction Fee (ETH)")
plt.show()

"""initial exploration of the dataset"""

numeric_cols = cleanedDataFrame.select_dtypes(include=['float64', 'int64']).columns

eda_summary = pd.DataFrame({
    "Column Names": cleanedDataFrame.columns,
    "Null Values": cleanedDataFrame.isnull().sum(),
    "Unique Values": cleanedDataFrame.nunique(),
    "Data Types": cleanedDataFrame.dtypes,
})

# Add mean, standard deviation, min, and max
eda_summary_numeric = pd.DataFrame({
    "Mean": cleanedDataFrame[numeric_cols].mean(),
    "Standard Deviation": cleanedDataFrame[numeric_cols].std(),
    "Min": cleanedDataFrame[numeric_cols].min(),
    "Max": cleanedDataFrame[numeric_cols].max(),
})

# Combine both summaries
eda_summary = pd.concat([eda_summary, eda_summary_numeric], axis=1)


print("First Few Rows of the Data:")
print(cleanedDataFrame.head())
print("\nEDA Summary:")
print(eda_summary)



# 1. Histogram of Numerical Features
```

```
cleanedDataFrame[numeric_cols].hist(figsize=(15, 10), bins=30, edgecolor='black')
plt.suptitle('Histogram of Numerical Features')
plt.show()

# 2. Correlation Heatmap for Numerical Features
plt.figure(figsize=(10, 8))
sns.heatmap(cleanedDataFrame[numeric_cols].corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap of Numerical Features')
plt.show()

"""Hypothesis Testing Visualization"""

block_stats = cleanedDataFrame.groupby('block').agg(
    avg_gas_fee=('txnFee', 'mean'),   # Average transaction fee as a proxy for gas fee
    transaction_volume=('txnFee', 'size')  # Transaction count per block as volume
).reset_index()

volume_threshold_high = block_stats['transaction_volume'].quantile(0.75)
volume_threshold_low = block_stats['transaction_volume'].quantile(0.25)

# Categorize blocks based on transaction volume
block_stats['traffic_period'] = np.where(
    block_stats['transaction_volume'] >= volume_threshold_high, 'Peak',
    np.where(block_stats['transaction_volume'] <= volume_threshold_low, 'Low', 'Normal')
)

# Visualization: Scatter plot with different colors for peak and low-traffic periods
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=block_stats, x='transaction_volume', y='avg_gas_fee', hue='traffic_period',
    palette={'Peak': 'red', 'Low': 'blue', 'Normal': 'gray'}, alpha=0.6
)
plt.title("Relationship Between Transaction Volume and Average Gas Fees with Traffic Periods")
plt.xlabel("Transaction Volume (Transactions per Block)")
plt.ylabel("Average Gas Fee (ETH)")
plt.legend(title="Traffic Period")
plt.show()

"""Machine Learning Analysis"""

from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# 1. Feature Engineering

block_stats['rolling_avg_volume'] = block_stats['transaction_volume'].rolling(window=5).mean()
block_stats['volume_to_fee_ratio'] = block_stats['transaction_volume'] / (block_stats['avg_gas_fee'] + 1e-9)  # Avoid
    division by zero
block_stats.dropna(inplace=True)
features = block_stats[['avg_gas_fee', 'block', 'rolling_avg_volume', 'volume_to_fee_ratio']]
target = block_stats['transaction_volume']


X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

# 2. Model Training and Comparison


models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'Lasso Regression': Lasso(),
    'Random Forest': RandomForestRegressor(random_state=42)
}

results = {}
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)

    results[model_name] = {
        'MAE': mae,
        'MSE': mse,
        'RMSE': rmse,
        'R  ': r2
    }

```

```
results_df = pd.DataFrame(results).T
print("Model Performance Comparison:")
print(results_df)

# 3. Hyperparameter Tuning

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(RandomForestRegressor(random_state=42), param_grid, cv=5, scoring='neg_mean_absolute_error')
grid_search.fit(X_train, y_train)
best_rf_model = grid_search.best_estimator_
print("Best Parameters for Random Forest:", grid_search.best_params_)

# 4. Evaluate the best model on test set
y_best_pred = best_rf_model.predict(X_test)
best_mae = mean_absolute_error(y_test, y_best_pred)
best_rmse = np.sqrt(mean_squared_error(y_test, y_best_pred))
best_r2 = r2_score(y_test, y_best_pred)

print("Best Random Forest Model Performance:")
print("MAE:", best_mae)
print("RMSE:", best_rmse)
print("R  :", best_r2)

# 5. Visualization

plt.figure(figsize=(10, 6))
plt.plot(y_test.values, label="Actual Transaction Volume", color="blue")
plt.plot(y_best_pred, label="Best Model Predictions (Random Forest)", color="orange", linestyle='--')
plt.legend()
plt.title("Transaction Volume Prediction: Best Model vs Actual")
plt.xlabel("Test Set Index")
plt.ylabel("Transaction Volume")
plt.show()
```