Machine Learning for Fraud Detection Using Data from IEEE-CIS Financial Transactions

Analyst Team
Team Lead - Jazmin Cervantes
Analyst - Cem Kazan
Analyst - Isabel Romero
Analyst - Jesse Dotson

Table of Contents

Abstract

The increasing prevalence of financial fraud presents a plethora of challenges to businesses and their consumers. This project, conducted by BitWizards, aims to enhance fraud detection methods using advanced machine learning techniques on the "IEEE-CIS Fraud Detection" dataset, sourced from Kaggle and provided by Vesta. The dataset includes over 140,000 transactions, containing detailed features such as transaction amounts, card information, addresses, and Vesta-engineered features for fraud detection.

Research Objectives:

Our project focused on three key research questions:

1.) **Feature Engineering and Selection for Fraud Detection (RQ1):**

- *Objective:* To improve fraud detection accuracy using techniques such as Recursive Feature Elimination (RFE), Gradient Boosting, and Principal Component Analysis (PCA).
- *Hypothesis:* Targeted feature selection will significantly improve model accuracy.

2.) **Predicting Transaction Distances (RQ2):**

- *Objective:* To develop models that can accurately predict transaction distances based on various factors in order to identify geographic anomalies, indicating fraud.

- *Hypothesis:* High-risk transactions based on long distance are more likely to be fraudulent.

3.) **Clustering for Coordinated Fraud Detection (RQ3):**

- *Objective:* To utilize clustering techniques in an attempt to identify groups of transactions that may be associated with coordinated fraud activities.
- *Hypothesis:* Fraudulent transactions that occur frequently within the same geographical area are more than likely to be a part of a coordinated fraud group.

Methodology:

1.) **Data Preparation:** To prepare our data for analysis, we first merged the transaction and identity datasets, then cleaned and imputed the data to handle any missing values. Relevant features we selected, numerical features were standardized, and categorical features were encoded. We then created a balanced sample of both fraudulent and non-fraudulent transactions in order to ensure robust model training and evaluation.

2.) **Feature Engineering and Selection:** We utilized advanced techniques such as Recursive Feature Elimination, Gradient Boosting, and PCA in order to improve fraud detection accuracy. Our results showed significant enhancements in model performance, with visuals such as ROC curves and feature importance rankings that illustrated our improvements.

3.) **Distance Prediction:** We developed regression models to predict the distance of transactions from the cardholder's expected location, aiding in the identification of high-risk transactions. Scatter plots and heatmaps highlighted any geographic anomalies indicative of potential fraud.

4.) **Clustering Analysis:** In order to identify patterns of coordinated fraud, we utilized a variety of clustering algorithms, including K-Means and Hierarchical Clustering. Visuals such as scatter plots and dendrograms helped us uncover patterns within the transaction data.

Our project's findings successfully demonstrated that advanced feature engineering, recession modeling, and clustering techniques could significantly enhance the detection of fraudulent transactions. These insights could provide Vesta with improved fraud detection models, offering better protection against fraudulent activities. BitWizards recommends leveraging these results in a campaign in order to showcase Vesta's enhanced capabilities, targeting financial institutions and eCommerce retailers.

Project Plan

May 29, 2024

Profile of the organization and background of the opportunity

Primary Company Details:

Founded - January 1, 1995
Founder(s) - Doug Fieldhouse
Headquarters - Lake Oswego, Oregon
Categories - Fraud prevention, eCommerce Safety, Payment Guarantee, Customer Experience

Address:

5400 Meadows Road
Lake Oswego, OR 97034
United States

Company Communications:

Phone Number: (888) 880-24002
E-mail: support@vesta.io
Website: https://www.vesta.io/

Business Description:

Vesta is a comprehensive payment and risk platform. They provide real-time decisioning and fraud protection for various industries, including eCommerce retail, fintech and financial services, digital travel, hospitality, e-ticketing, and more.

Vesta operates as a comprehensive payment and risk platform. They offer a variety of services aimed at enhancing the security and efficiency of online transactions. They are known for their rich expertise in understanding fraud and have developed a modern platform for detecting and preventing fraud. Their platform works continuously, monitoring for new transactions, datapoints and providing real-time fraud detection alerts. This allows Vesta to

maintain a high level of security and efficiency in online transactions. Vesta offers a variety of services aimed at enhancing the security and efficiency of online transactions. Their integrated payment and risk platform includes balance approvals and fraud, and self-service chargeback refunds. They use advanced AI models to detect anomalies and inconsistencies catching even hard to detect anomalies.Their customers and partner networks rely on Vesta to safeguard millions of transactions and billions of dollars worth of volume every single year.

Financials:

Latest Financial Data - December 2023
Peak Revenue - $200 M
Total Funding - $ 125 M
No. of Employees - 600

Key Executives:

Todd Curry, CEO
Ted Truong, COO
Shimon Steinmets, CFO
Mercedes Klemen, Chief People Officer
Paddy Beagan, General Manager

Major Competitors:

Sift
Signifyd
SEON. Fraud fighter
Riskified
ClearSale
NoFraud

Business/Analysis Opportunity:

The increasing occurrences of financial fraud poses a significant challenge to businesses and consumers alike. Leveraging the "IEEE-CIS Fraud Detection" dataset, containing over

140,232 data points for transaction and identity features. It includes transaction details such as IDs, timestamps, amounts, and product codes, along with payment card information (card1-card6), addresses (addr1-addr2), and distances (dist1-dist2). It also contains purchaser and recipient email domains, various counts (C1-C14), time deltas (D1-D15), match indicators (M1-M9), and Vesta-engineered features (Vxxx) for ranking and entity relationships. These models hold the potential to significantly benefit financial institutions and online platforms by Reducing financial losses, improving customer trust and optimizing operational efficiency.

Research Questions

New methods of protecting financial transactions are required due to the growing complexity of fraudulent activity. In order to tackle this difficulty, the following research questions will be looked at in this project.

RQ1: How can advanced feature engineering and selection techniques, such as Recursive Feature Elimination (RFE), Feature Importance from Gradient Boosting, and Principal Component Analysis (PCA), enhance the accuracy of fraud detection models?

Financial transaction data provides a wealth of potential predictors, but not all features contribute equally to accurate fraud detection. For example, the email of the recipient may not be a factor to consider in our models. This study topic explores the crucial step of determining which qualities are the most informative and turning them into reliable predictions. We want to identify the features that most effectively distinguish fraudulent transactions from real ones by utilizing methods such as PCA, RFE, and Feature Importance from Gradient Boosting. This

refined feature set will be used to train more accurate fraud detection models, ultimately improving their ability to identify fraudulent activity for real-time applications.

RQ2: How can machine learning models predict the distance where the transaction was made based on various factors, including transaction amount (TransactionAmt), card information (card1 - card6), address (addr1, addr2), time between transaction(D1-D15) and security features (Vxxx)?

Location-based anomalies can be strong indicators of fraudulent behavior. This research question focuses on building machine learning models that can accurately estimate the distance of a transaction from the cardholder's expected location. By analyzing a combination of transaction details, card information, address data, time between transactions, and Vesta's engineered security features, the models will learn to identify unusual geographic discrepancies. This information can be used to flag potentially fraudulent transactions, providing an additional layer of security for financial institutions and online platforms.

RQ3: How can clustering techniques be used to identify groups of transactions that may be part of coordinated fraud activities?

Coordinated fraud often involves multiple transactions exhibiting similar characteristics or following distinct patterns. This research question explores the power of clustering techniques in uncovering hidden relationships within transaction data. We do this by clustering transactions according to common characteristics and closeness in an effort to find anomalies in their behavior. By examining these clusters, investigators can get insights into coordinated fraud schemes, comprehend changing strategies, and create more potent defenses against them.

Hypotheses

H1: Feature engineering and selection will significantly improve the accuracy of fraud detection models compared to models using all transaction data.

We believe that carefully selecting and engineering the right features will lead to more accurate fraud detection. By using techniques like RFE, Feature Importance, and PCA, we can pinpoint the specific transaction attributes that best distinguish between legitimate and fraudulent activity. We intend to test this hypothesis.

H2: Transactions flagged as high-risk based on distance will have a higher probability of being fraudulent.

Transactions that are marked as high-risk based on distance have a greater likelihood of being fraudulent. By predicting the possible distance of fraudulent transactions from cardholders' typical location, If a transaction occurs far from where we'd expect based on the cardholder's usual activity, it is more likely to be fraudulent. We intend to test this hypothesis.

H3: Transactions that have similar traits of questionable characteristics have a high likelihood of being fraudulent.

Organized fraud cases have common characteristics that could be identified. Grouping unusual characteristics, such as multiple transactions occurring in quick succession from geographically distant locations, transactions involving unusually large amounts or unusual product categories, or transactions that deviate from typical spending patterns, we can uncover fraudulent transactions. We intend to test this hypothesis.

Data

This project leverages the Fraud Detection dataset, sourced from a Computational Intelligence Society hosted by Vesta. This dataset captures a range of online transactions from various outlets. The data includes categorical, datetime and numerical features for 140,232 transactions. The dataset is divided into two primary files: one focused on transaction details and another providing supplementary identity information.

Transactions

Majority of the data rely on the transactions with 434 features. *TransactionID* and *TransactionDT* uniquely identify each transaction and provide a relative timestamp. *isFraud*, *TransactionAMT* and *ProductCD* are the transaction details for the fraud flag, amount of dollars spent and the product or service name. *card1* through *card6* denotes the anonymized information about the payment card used, such as card type, category, issuing bank, and country. *addr1*, *addr2*, *dist1*, *dist2* are the location data. *addr1* and *addr2* are billing region and country respectively and *dist1* and *dist2* capture distances related to the transaction. *P_emaildomain* and *R_emaildomain* features provide the email domains of the purchaser and recipient. *C1-C14* represent counts of different entities linked to the purchaser and recipient, such as addresses, phone numbers, and devices. *D1-D15* captures time deltas between transactions and other events, providing insights into transaction frequency and timing. *M1-M9* indicates matches between information elements, such as names and addresses. Lastly *Vxxx* are the undisclosed Vesta engineered features for rankings, counts, and entity relationships.

Identity

Identity data includes 41 features about the individuals or accounts behind some of the transactions. We have information about the type of device used for the transaction (*DeviceType*) and more detailed device information (*DeviceInfo*), such as the operating system and browser. Features like *id_12* through *id_38* represent anonymized identifiers that can help link related transactions or accounts, potentially revealing patterns of fraudulent activity.

Measurements

There are three critical measurements needed in order to successfully perform a comprehensive analytical report of this spectrum. Based on our objectives, these key measurements include fraud detection accuracy, transaction location anomalies, and coordinated fraud activities.

Fraud detection accuracy measures how well a model can distinguish between fraudulent and non-fraudulent transactions. This is done using metrics such as precision, recall, and F1 score. Transaction location anomalies are measured by predicting the total distance the transaction location and the cardholder's expected location, analyzing several features such as transaction amount, card information, and location data. Coordinated fraud activities are identified by clustering transactions based on several shared characteristics. These clustered groups are then examined for any signs indicative of fraud, such as simultaneous transactions from different locations or unusual spending behaviors.

Through the Kaggle platform, Vesta has graciously provided users with real-world e-commerce transaction data, containing a plethora of features which align with our project's key measurement focuses.

Methodology

For research question 1, we are trying to get insight into our features and attempting to select the features with the best model prediction. We will first establish a performance baseline by training a classification model. We will then engineer new features based on domain expertise and insights gleaned from exploratory data analysis. This might involve creating interaction terms, time-based features, or aggregated features derived from card, user, or location data. Next, we will apply feature selection techniques to pinpoint the most predictive features and reduce the data's dimensionality using PCA. Finally, we will train and evaluate various classification models on both the engineered feature set and the full dataset. We will assess the impact of feature engineering and selection on model accuracy, ultimately identifying the most effective techniques.

For research question 2, we are trying to analyze how transaction distance from the customer's actual location plays a role in fraud detection. To predict transaction distances and identify potentially fraudulent geographic anomalies, we will first address missing values in location-related features. We will then select relevant features, including transaction amount, card information, time between transactions, and potentially Vesta's engineered features. Categorical features will undergo appropriate encoding transformations. Regression models will be trained to predict the distance of a transaction from the cardholder's expected location. By defining a risk threshold based on predicted distance, we can flag potentially fraudulent

transactions. The effectiveness of distance prediction in identifying fraud will be evaluated by analyzing the proportion of fraudulent transactions within the high-risk group.

For research question 3, we are interested in grouping the data to see if there are any hidden characteristics that make a transaction fraudulent. This involves selecting relevant features, transforming data to address skewness and normalize features, and experimenting with various clustering algorithms to group transactions based on shared characteristics. The resulting clusters will be analyzed for patterns indicative of coordinated fraud, such as simultaneous transactions from different locations or unusual spending patterns. Visualization techniques will be employed to interpret and communicate these findings.

Computational Methods and Outputs

We will measure the effectiveness of different feature engineering and selection techniques for question one using the Area Under the ROC Curve (AUC-ROC). AUC-ROC is a reliable metric for evaluating a model's ability to distinguish between fraudulent and legitimate transactions, especially when dealing with imbalanced datasets. To prevent overfitting and test our model’s generalizability, we will use multi-fold cross-validation during model training and evaluation. The main output of this research question will be a comparison of different classification models, which could include but is not limited to, XGBosst and RandomForest trained on all and selected feature sets. We will compare metrics like accuracy, precision, recall, F1-score, and AUC-ROC to see how much impact feature engineering and selection have on fraud detection accuracy.

To identify potentially fraudulent transactions based on unusual geographic patterns, we will build a model to predict a transaction's distance from the cardholder's typical location. We

will start by addressing missing values in location-related features and selecting relevant features for prediction, including transaction amount, card information, time between transactions, and potentially Vesta's engineered features. Categorical features will be encoded using one-hot encoding. We will experiment with a regression model like Linear Regression and Random Forest, selecting the best-performing model based on metrics like Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). Once we have a reliable model, we'll define a distance threshold, the mean of the fraudulent distances to flag high-risk transactions. Transactions exceeding this threshold will be considered potentially fraudulent. We will evaluate our approach by analyzing the proportion of fraudulent transactions within the high-risk group compared to the overall dataset.

For research question 3, we will begin by selecting features relevant for clustering, including but not limited to transaction amount, product code, time between transactions, location data, and potentially Vesta's engineered features. We will then transform the data to address skewness and normalize features, ensuring optimal performance of our clustering algorithms.

We will experiment with various clustering algorithms, such as K-Means, DBSCAN, and Hierarchical Clustering, to determine the most effective approach for grouping transactions based on shared characteristics. The choice of the best algorithm will depend on the structure of the data and the specific patterns we aim to uncover. Once we have identified distinct clusters of transactions, we'll analyze their characteristics to look for patterns indicative of coordinated fraud. This might include geographic anomalies, transactions involving unusually large amounts, unusual product categories, or deviations from typical spending habits and clusters involving transactions from multiple accounts with similar attributes. Visualization techniques will play a

crucial role in interpreting and communicating our findings. We'll use scatter plots, dendrograms (for hierarchical clustering), and other appropriate visualizations to explore the relationships between transactions within and across clusters.

Output Summaries

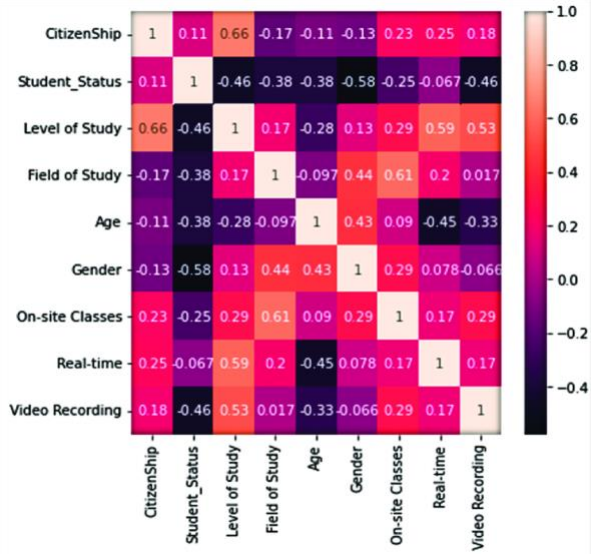
RQ1: How can advanced feature engineering and selection techniques, such as Recursive Feature Elimination (RFE), Feature Importance from Gradient Boosting, and Principal Component Analysis (PCA), enhance the accuracy of fraud detection models?

This analysis will consist of identifying the most effective feature engineering and selection techniques that improve fraud detection accuracy. Our outputs will include tables and visuals that clearly and accurately represent our findings.

A table summarizing the classification models (e.g. XGBoost, Random Forest) trained on all features vs. the selected feature sets will be included. The metrics for this table will include accuracy, precision, recall, F1-score, and AUC-ROC. We will also be including a table listing the top features ranked by their importance, as determined by techniques such as Recursive Feature Elimination (RFE) and Gradient Boosting feature importance. We will also include a detailed comparison of model performance using multi-fold cross-validation to ensure generalizability and also to prevent any overfitting.

A graphical representation of ROC curves for each model to visualize their ability to distinguish between fraudulent vs. legitimate transactions will be necessary to include. Other visuals will include a heat map demonstrating how different features contribute to model predictions, highlighting which features within the dataset are most indicative of fraud, as well as a feature correlation matrix demonstrating the correlation between different features in order to

identify potential feature interaction. The following is a visual representation of how our feature correlation matrix may look:



This feature correlation matrix represents a matrix that shows how different features affected students' performance in college during the COVID-19 pandemic. Extracted from: https://www.researchgate.net/figure/Correlation-matrix-for-feature-selection_fig2_368261486.

These outputs will provide us with valuable insight into how feature engineering and selection impact fraud detection, aiding in the future development of more reliable and accurate models.

RQ2: How can machine learning models predict the distance where the transaction was made based on various factors, including transaction amount (TransactionAmt), card information (card1 - card6), address (addr1, addr2), time between transaction(D1-D15) and security features (Vxxx)?

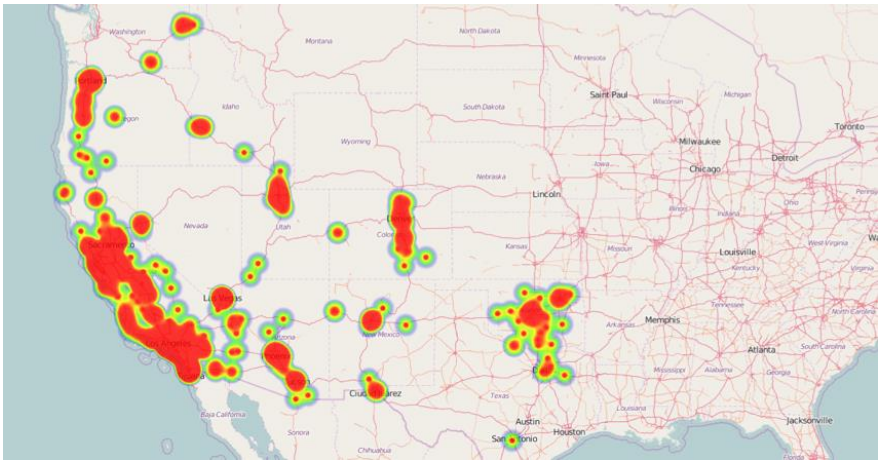
This analysis will build a regression model that predicts transaction distances and harnesses the ability to predict transaction distances and identify potential geographic anomalies

that may be indicative of fraudulent transactions. These outputs will include a mixture of tables and visuals that will aid in understanding the role of geographic patterns in fraud detection.

We will be creating a table summarizing the performance of regression models (e.g. Linear Regression, Random Forest) using metrics like Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). We will also be including a summary table that lists transactions flagged as high-risk based on surpassing the distance threshold, with details on their predicted distances and relevant features.

A scatter plot will be made comparing the predicted transaction distances to actual distances, for both fraudulent and legitimate transactions. Histogram and density plots, demonstrating the distribution of predicted distances for fraudulent transactions, will also be provided in order to help define the risk threshold.

We will also be creating a heat map that demonstrates the areas of high concentration in regards to geographically anomalous flagged transactions, showing any potential fraud hotspots. The following is a visualization of what our geographic heat map may look like:



This geographic heat map is showing the locations of Carl's Jr. restaurants. Red areas represent where there are a high volume of Carl's Jr. restaurants in close proximity, while cooler areas show where there are fewer Carl's Jr. restaurants. Extracted from: https://mode.com/example-gallery/geographic-heat-map.

Bar charts showing the importance of various features in predicting transaction distances will also be provided.

These outputs will assist in the understanding of geographical patterns found in fraud detection, enhancing the effectiveness of flagging suspicious transactions based on location data.

RQ3: How can clustering techniques be used to identify groups of transactions that may be part of coordinated fraud activities?

This analysis will utilize clustering techniques to uncover any hidden patterns indicative of coordinated fraud. We will be including several types of visualizations, such as tables, maps and scatter plots.

We will be incorporating different table outputs, such as a table that summarizes the performance of different clustering algorithms (e.g. K-Means, DBSCAN, Hierarchical Clustering). We will also create tables of the distinct clusters we have identified, highlighting any key characteristics such as transaction amounts and geographic locations.

Scatter plots will be included, which will be utilized to demonstrate the relationships between transactions, both within and between clusters.

A heat map that shows the geographic distribution of different clusters will be provided, identifying any potential regions with a high probability of coordinated fraudulent activity. An interactive map that allows for the exploration of clustered data and identification of any suspicious patterns will be important to include.

These outputs will assist us in providing insight into the structure and characteristics of fraudulent transactions, which may aid in the identification and prevention of any future coordinated fraud activity.

Campaign Implementation

Fraud detection and prevention are crucial components for businesses in the financial sector, especially companies such as Vesta, whose sole purpose is to provide fraud protection to its clients. By developing a campaign that effectively utilizes the insights gained from our research questions, we can provide Vesta with valuable insights to further enhance security and promote trust among their consumers.

By utilizing advanced feature engineering and selection techniques, as mentioned in Research Question #1, we can gain further insight on how to develop more accurate fraud detection models, benefiting Vesta. These upgraded fraud detection models can provide better protection against fraudulent transactions, resulting in higher satisfaction rates among Vesta's consumers. Some of these target clients that would greatly benefit from these enhancements are financial institutions and eCommerce retailers. These significant upgrades would give Vesta a competitive advantage.

As discussed in Research Question #2, predicting transaction distances and identifying any potential geographic anomalies would provide an additional layer of protection for Vesta's clients. By providing the capability to identify and flag suspicious transactions occurring in unexpected locations, Vesta could more accurately prevent fraudulent activities before they occur. Any clients concerned about transaction security, such as financial institutions, online platforms and businesses, could rest assured knowing that Vesta's platform is properly equipped to identify any potential fraud, using their advanced geographic anomaly detection capabilities.

Identifying groups of transactions that may be indicative of coordinated fraudulent activity would further improve Vesta's fraud detection efforts. Serving as the focus of Research Question #3, this clustering of suspicious transactions, strongly indicating coordinated fraud

schemes, would demonstrate to clients Vesta's ability to detect and disrupt coordinated fraud activities through advanced clustering techniques. The target audience for this technique would be risk management professionals, fraud investigators, and compliance officers in financial institutions. Vesta's early fraud detection capabilities would assist in the elimination of these coordinated fraud schemes.

By utilizing the insights and solutions derived from this project, we can provide Vesta with an effective campaign that would showcase their improved capabilities, attract potential clients, and provide trust in their platform's enhanced security measures.

Literature Review

When the topic of data science in retail comes up the focus is usually on increasing revenue. But losses are always present, and reducing them is always an opportunity to increase operational efficiency and profitability. A particularly valuable method of loss prevention is the timely detection of fraudulent transactions. With physical goods, detecting fraud at the point of sale or shortly thereafter prevents material losses, wasted time, and effort. Delivering a unit for free is a total loss and unlike replacing damaged products, there is no fringe benefit like increasing customer satisfaction. However when it comes to ecommerce, timely detection of fraudulent transactions can be challenging since the time frame is smaller. By incorporating machine learning models to fraud detection into ecommerce, it can potentially help to find the fraudulent transactions in a timely manner.

While it is not difficult to find discussions of using machine learning for fraud detection in general, ecommerce specifically is not well covered. Multiple literature reviews on the topic had to resort to covering the related but distinct credit card (CC) fraud. In fact, *Rodrigues et al*

declared in their 2022 literature review "...we did not find any articles focusing specifically on e-commerce strategies for fraud detection." An earlier literature review of generalized fraud detection with data mining discussed the lack of published research on mortgage and securities fraud, positing that the sensitive and private nature of the datasets make published research less likely (*Ngai et al* 2011). This makes sense in our context, as within our dataset much of the information has been "pre-engineered" and masked to keep potentially identifying nature secret. Besides privacy concerns, customer data is now a valuable commodity that businesses aren't likely to give up freely.

Even a very recent literature review had to analyze articles on other types of fraud detection with machine learning, finding neural networks were the most common method of combating CC fraud (*Mutemi et al* 2024). In this area, advances over the standard methods are being explored. Promising results have been found with improving neural networks by transforming features with deep autoencoders (*Fanai et al* 2023).

While many instances of ecommerce fraud are also CC fraud, the reviews point out that CC fraud is from the perspective of financial institutions and it is their datasets being used to fight it, missing the picture seen by electronic retailers (*Rodrigues* et al 2022).

Our search wasn't entirely empty handed, Marchal and Szyller specifically tackled fraud detection in ecommerce in a 2019 paper. Trailblazers that they are, they developed a novel approach to categorical clustering. The results were a method fast enough to categorize a day's worth of transactions in a couple hours. While this produces results in time to stop shipment, it depends on clustering transactions into "fraudulent campaigns." The authors note the shortcoming of their algorithm in being unable to detect singleton instances of fraud. This necessary clustering and complex algorithm prevents real time detection.

This search has revealed there is quite a lot of room for new research without straying too far from traditional approaches. Another one of the very few papers (*Tang* 2023) looking specifically at ecommerce fraud detection employed the classic multilayer perceptron, using a Reinforcement-Learning Driven Artificial Bee Colony algorithm for determining weights. With the dearth of competing research they had little to compare it to but their own implementations of KNN, SVM, Logistic Regression, Decision Trees, and all the familiar favorites of an undergraduate Data Science program.

When a purchase is made online, the bank doesn't see the location, but the retailer has information about the connection. Atypical connection details, like geographic location, from a known customer might indicate fraud. We intend to investigate that detail. However, since the body of research doesn't eliminate very many avenues, it's worth exploring unsupervised clustering methods to search for less obvious characteristics of fraudulent transactions.

Research Questions

1) How can advanced feature engineering and selection techniques, such as Recursive Feature Elimination (RFE), Feature Importance from Gradient Boosting, and Principal Component Analysis (PCA), enhance the accuracy of fraud detection models?

2) How can machine learning models predict the distance where the transaction was made based on various factors, including transaction amount (TransactionAmt), card information (card1 - card6), address (addr1, addr2), time between transaction(D1-D15) and security features (Vxxx)?

3) How can clustering techniques be used to identify groups of transactions that may be part of coordinated fraud activities?

Exploratory Data Analysis

The dataset for this project was found on kaggle.com provided by Vesta for a competition, it includes two files transaction and identity sets that are divided into training and testing sets. After merging the files, it included over one million observations, 434 features for training, and 433 features for testing. After cross referencing the features between the two datasets, it was found that the testing set was missing the isFraud feature, which is important to validate our results after building the model. Resulting in dropping the testing set and focusing on only the training set, leaving around 540 thousand instances and 434 features. Furthermore, after consideration, it was decided to only focus on the transactions set to narrow down the amount of features from 434 to 394. However, for analysis purposes some features from the identity will be used. As for missing values we are using -999 for numerical features and unknown for categorical features, except for descriptive statistics.

Since Recursive Feature Elimination (RFE), Feature Importance from Gradient Boosting, and Principal Component Analysis will be applied for the selection of the features, we are going to look at all features, but mainly the ones we think would be more impactful for our model which include:

ProductCD: product code

TransactionAmt: the amount spent on the transaction

TransactionDT: timedelta from a given reference datetime

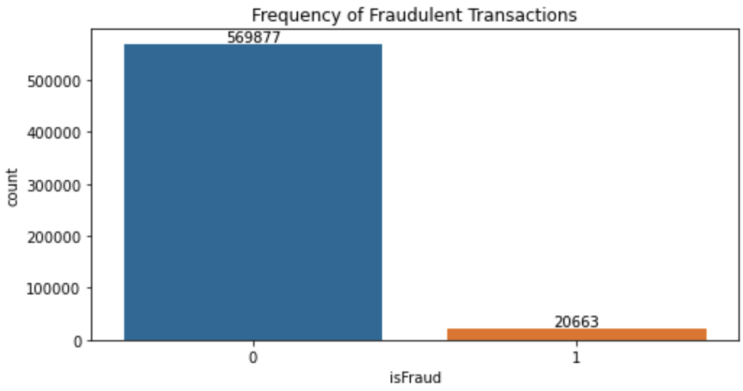
card4 and card6: payment card information (card type, card category, issue bank, country, etc)

dist1 - dist2: distance

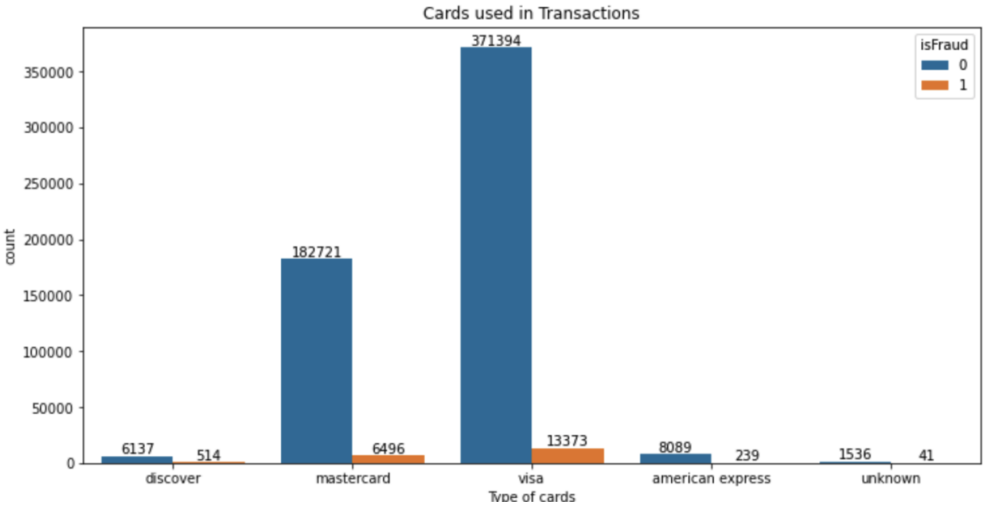
Devicetype: Device used for transaction

isFraud: if transaction is fraudulent(1) or not(0)

We implemented the use of python to create visualization for this analysis with the help of matplotlib and seaborn libraries. From our literature review, one of the challenges found was that fraud datasets are unbalanced because of the following reasons: clients don't notice the fraudulent transaction, clients do notice the fraudulent transaction and don't report it or there aren't many fraudulent transactions. So to start out we wanted to see how our data was distributed which resulted as follows:

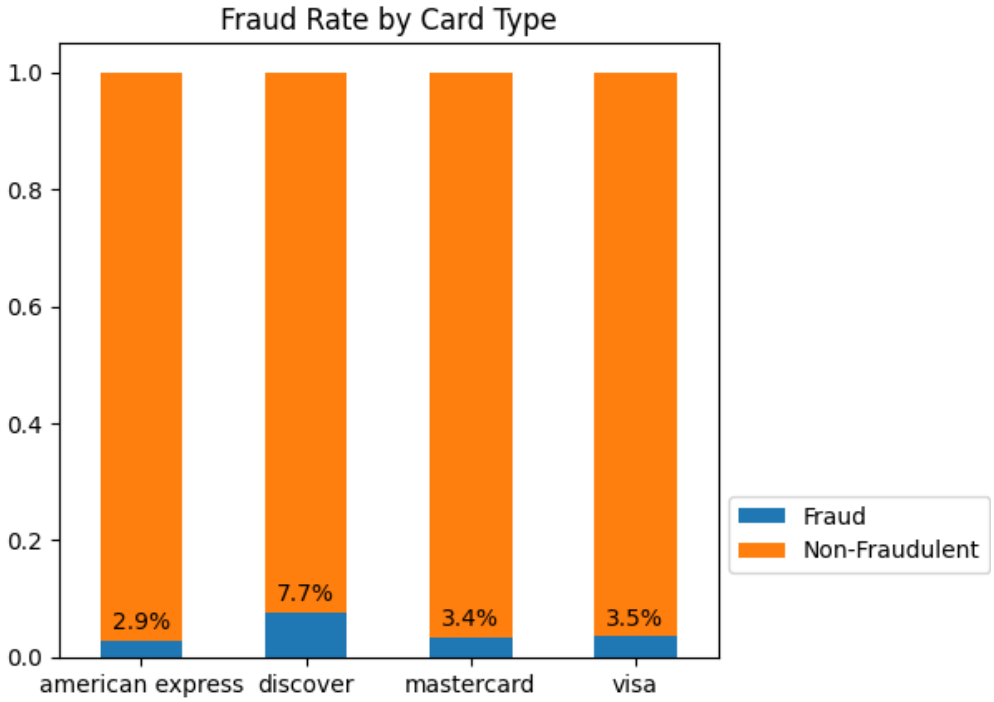


Based on the graph, we can see that 96.5 percent of the transactions are not fraudulent, this tells us that we are dealing with an unbalanced data set which can cause a problem with the training of the model. From this we wanted to see how the transactions were divided based on the type of card used as shown in the following diagram:



Referencing the barplot, we found that there are four card types (visa, mastercard, discover, american express, and unknown) and out of those the most frequent are visa and mastercard. As expected, it also shows that the most fraudulent transactions come from the same type of cards. We went a step further, based on appendix g, the majority of the transactions were done by visa and mastercard's debit cards while the rest of the cards the majority of transactions were done by credit cards. According to appendix h, based on account type, the most popular is debit and credit cards, even though the difference in amount of transactions between the two is significant, the amount of fraudulent transactions were close to each other with 9950 from credit and 10674.

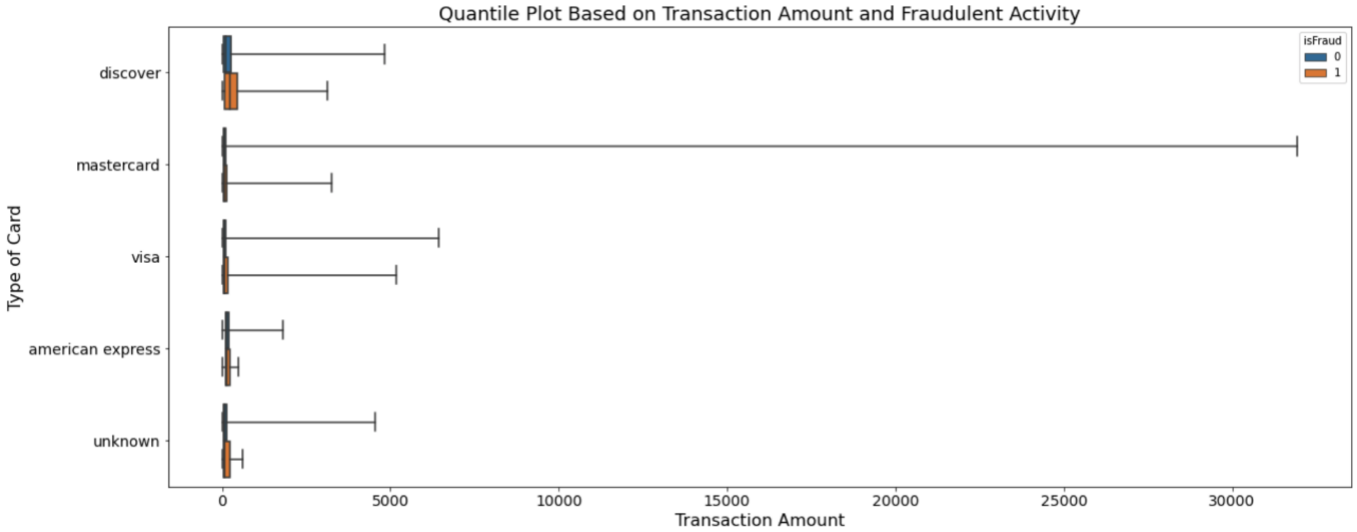
In addition to observing the quantity of transactions by card type, we also plotted the fraud rate of each card. From below, we can see that the average fraud rate of ~3.5% is strongly driven by the most common card types, but the lesser used Discover card actually has a higher rate of fraud.



Next we broke it down further to see what products were bought using the different types of cards and to see the product that was more likely to have a fraudulent transaction. As shown in appendix c and d, there are five types of products named as follows: W, H, C, S, and R. The product code W had most sales, based on appendix c and d, and it also had the same result as before since it's the most sold, it showed that it had the most fraudulent transactions. Then we

applied a descriptive statistics on the transaction amount and fraudulent transaction that resulted in this boxplot:

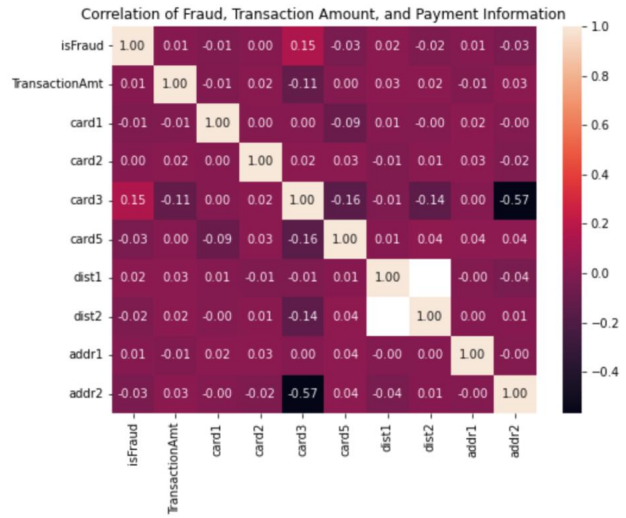
From this we found that the mean of the transaction amount was $135.03 dollars, the least amount spent was $0.25 dollars and the most expensive transaction was $31937.39 dollars. Surprisingly, the most expensive transaction was made with mastercard not with Visa, the most popular type of card in the dataset. We can also see that the greatest mean amount for fraudulent and non-fraudulent transactions was made using a Discover card, but this can be explained by the



fact that Discover didn't have as many transactions as Visa and Mastercard as shown in appendix b.

Additionally, we analyzed the type of device that was used during the transaction and spilled it up into fraudulent and non-fraudulent transactions which resulted in appendix f. We learn that there are only three types of devices which are desktop, mobile, and unknown. The most transactions were done by an unknown device and it also had the highest fraudulent transactions, then its desktop and lastly mobile. As for fraudulent transactions, desktop and mobile were close to each other with 5554 and 5657(respectively).

We created three different correlations only using the numerical features to find if we could start reducing more features from the raw database based on their correlation value as follows:



The heatmap above only takes into account fraud, transaction amount, and some payment information. From this we can see that most features don't have a strong correlation with isFraud by having a value close to zero or negative. The only one that stood out was card3 but it wasn't a significant correlation since it is closer to zero than one. Also for the distance features we found out that dist2 has the least amount of missing values, with a mean 231, farest distance was 11623, and least distance was zero. Then we looked at the relationship between fraud and counting which associated the amount of addresses associated with the card. Referencing appendix j, the relationship between fraud and all of the counting features is not strong while all of the counting features had a strong relationship between them. Lastly, based on appendix k, there was no relationship between fraud and time delta since the majority of features were negative. From all of the correlations that we made we couldn't start reducing since there are a lot of missing values which caused the correlation values to be so low for some of the heatmaps.

In conclusion, the majority of fraudulent transactions were located in the most popular type of card, type of product and type of account. Based on our analysis it is a transaction is a visa debit card, product W, and comes from an unknown device it has a high probability of being a fraudulent transaction.

Ethical Recommendations

The ethics of accurately detecting and stopping a fraudulent transaction are fairly unambiguous. Outside of favorite edge courses of philosophy and ethics courses like stealing loaves of bread to feed the starving, theft is an innate harm. In addition to the material losses, fraudulent transactions erode trust, a necessary pillar of society. The key word here is "accurately." The inherent value of identifying a fraudulent transaction comes from the follow up action of denying service or material goods to the fraudster. However, acting on a false positive victimizes an innocent person who may be trying to acquire any number of necessary goods or services. Furthermore, utilizing engineered features introduces an opaque layer whose effects are difficult to measure.

To classify an attempted purchase as a criminal act or not based on the characteristics of the customer is a form of criminal profiling. Our model is asking "does this actor fit the profile of a criminal?" While being falsely accused by a company's Point of Sale in the context of an online transaction is less severe than in a legal setting, this lessened severity brings with it less scrutiny and oversight. Businesses should err on the side of caution so as not to unduly disrupt the lives of innocent customers. It is in their own interests to limit false positives as well, even if it creates more false negatives. When picking a threshold for the model a business might be

tempted to minimize predicted losses even if it invalidates more legitimate transactions. But this ignores the potential alienation of customers, who might come to view a business as unreliable.

In addition to the problems of falsely accusing anyone, one of the most problematic uses of criminal profiling has been to target minorities on the basis of race. The justification for this was purported to be statistical evidence of a higher likelihood to commit a crime among certain demographics. Studies showed these demographics were being oversampled by store security and police, and the underlying statistics were systematic confirmation bias (*Dabney* 2006). The difference here is that our data is devoid of apparent demographic information such as race. Ironically, this makes it *more* difficult to determine if there's an inherent bias or skew in a model's determinations. The data provided was anonymized for privacy concerns, and pre-engineered features were also included. The lack of transparent meaning, further compounded by methods like PCA, make it difficult to determine if the model is influenced by factors correlated with race. Full access to the original dataset doesn't guarantee that such bias could be detected, either. Great care should be taken to ensure models such as these don't disproportionately target certain segments of the population due to some oddity or non-causal correlation in the dataset.

Businesses must be careful not to over apply such methods. The reality is that shrink is a constant that all businesses must live with and account for. In our study the most effective models are capturing only ~60% of fraud cases, and mislabeling about 10% of the flagged transactions to achieve that hit rate. Businesses should not rely on an imperfect solution at the cost of their customers. If a model is going to be the sole input, they should restrict the threshold to limit false positives as much as possible.

Like other forms of criminal profiling, in isolation these methods are not sufficient for certainty. In summarizing the debate surrounding generalized criminal profiling, Michael Boylan

notes that most critics find it unscientific, but useful when combined with other forms of evidence (2011). Businesses should follow the model of card issuers, who generally provide an avenue for card holders to immediately confirm a flagged transaction through an app or automated phone call.

Methodology

RQ1: How can advanced feature engineering and selection techniques, such as Recursive Feature Elimination (RFE), Feature Importance from Gradient Boosting, and Principal Component Analysis (PCA), enhance the accuracy of fraud detection models?

Our investigation into the impact of feature engineering and selection on fraud detection accuracy begins with a comprehensive data preparation process. We decided to remove features that have more than 60% of the data missing and remaining data will be imputed based on specific feature information using methods of mean,median or KNN imputation. In addition feature transformation methods of scaling and one-hot encoding to get our data ready for feature selection.

We will first establish a performance baseline by training a Dummy Classifier model on the raw transaction data without any feature engineering or selection. We will use the isFraud binary feature on our data as our target variable. This will provide us with scores that we can use to compare our improved models. In addition, we will train our raw data on Random Forest, XGBoost and decision tree models to get base AUC and accuracy scores. Our data already involves Vestas engineered features that will help us with the feature selection process. An

example for one of the features would be how many times the payment card associated with an IP and email or address appeared in a 24 hours time range.

Feature Selection

To identify the most predictive features and reduce dimensionality, we will apply feature selection techniques such as Recursive Feature Elimination (RFE), Feature Importance from Gradient Boosting (e.g. XGBoost), and Principal Component Analysis (PCA). RFE iteratively removes features with the least predictive power, starting with the least important feature and progressively removing features until a desired number of features remains. This process is computationally heavy but yields the best features for the models. Feature Importance from Gradient Boosting models assigns a score to each feature based on its contribution to the model's prediction accuracy. Features with higher importance scores are considered more predictive. PCA transforms the original features into a smaller set of uncorrelated components, capturing the most significant variations in the data. This can be used to reduce dimensionality and improve model performance by removing redundant information.However, a key limitation of PCA is that it loses the interpretability of the original features. Since we are focusing on improving the model performance, retaining feature interoperability is less important to us.

Modeling Techniques:

- **Random Forest:**
    - Bootstrap Aggregating a.k.a Bagging builds multiple decision trees each trained on a random subset of the data and features. The predictions from all the trees are then combined to make a final prediction. Random Forest is known for its

robustness and ability to handle high-dimensional data with complex relationships. It is particularly well-suited for our fraud detection task, as it can capture nonlinear interactions between features and improve prediction accuracy.

- **XGBoost:**
    - This is a gradient boosting algorithm that sequentially builds an ensemble of decision trees, each correcting the errors of the previous trees. XGBoost is known for its high performance and ability to handle large datasets with complex relationships thus we expect it to perform well in our fraud detection analysis.
- **Decision Tree:**
    - This model creates a tree-like structure where each node represents a split based on a specific feature. The splits are chosen to maximize the separation of fraudulent and legitimate transactions. We will use Decision Trees to explore the relationships between features and fraud, and to gain insights into the decision-making process of the model.
- **Dummy Classifier:**
    - This is a simple baseline model that makes predictions based on the most frequent class in the training data. It will serve us as a reference point to compare the performance of more complex models.

To evaluate our model we will use AUC-ROC score since AUC represents the degree of separability between fraud and non-fraud cases. A higher AUC indicates a better performance in distinguishing between the two classes. Real fraud cases are unbalanced in terms of transactions being fraud is less likely thus accuracy score can be misleading (our data contain 96.5%

non_fraud cases). We will apply 10-fold cross-validation during model training and evaluation. This will decrease the risk of overfitting and ensure the generalizability of our findings.

We will assess how feature engineering and selection strategies affect fraud detection comparing the performance of several classification models trained on different feature sets. We will identify the most effective approaches for increasing model performance and discuss the value of various indicators in predicting fraudulent transactions.

RQ2: How can machine learning models predict the distance where the transaction was made based on various factors, including transaction amount (TransactionAmt), card information (card1 - card6), address (addr1, addr2), time between transaction(D1-D15) and security features (Vxxx)?

First step of this analysis is to separate the fraudulent and non fraudulent transactions to decide on a risk threshold that we will use to flag if the prediction exceeds the decided threshold. We decided to use the following features in our regression models

- *Transaction Amount (TransactionAmt):* Unusual transaction amounts (our dataset contains outliers with most expensive transactions $31937.39) might indicate fraudulent activity, particularly when combined with location information.
- *Card Information (card1 - card6):* Features related to the payment card, such as card type, category, issuing bank, and country, can provide insights into potential fraud patterns.

- *Address Data (addr1, addr2):* Billing and shipping addresses can help identify transactions occurring far from the cardholder's usual location.
- *Time Between Transactions (D1 - D15):* The frequency and timing of transactions can be indicative of fraudulent activity, especially when combined with location data.
- *Vesta Engineered Features (Vxxx):* These features, engineered by Vesta, may capture complex patterns and risk factors related to transaction distance and fraud.

To predict transaction distances, we'll explore both Linear Regression and XGBoost, two powerful regression models with distinct strengths. Linear Regression, a classic approach, assumes a linear relationship between features and the target variable (distance). While simple, it provides a baseline for comparison. XGBoost, a more advanced gradient boosting algorithm, excels at handling complex relationships and is often considered state-of-the-art for regression tasks. We'll use hyperparameter optimization technique using bayesian search, to fine-tune the parameters of each model, finding the optimal configuration for our specific dataset and chosen features. This optimization will involve exploring different combinations of regularization parameters, learning rates, tree depth, and other model-specific settings to maximize prediction accuracy. By comparing the performance of both models, we aim to identify the best approach for predicting transaction distances and uncovering geographic anomalies that might indicate fraudulent activity.

To assess model performance, we will primarily use Root Mean Squared Error (RMSE) as our evaluation metric. RMSE measures the average difference between predicted and actual distances, providing a valuable indicator of the model's accuracy in predicting transaction distances.

We will analyze the proportion of fraudulent transactions within the high-risk group (transactions exceeding the distance threshold) compared to the overall dataset. This will allow us to assess the model's ability to correctly identify fraudulent transactions based on geographic anomalies.

RQ3: How can clustering techniques be used to identify groups of transactions that may be part of coordinated fraud activities?

The objective of our third research question is to apply clustering techniques to the transaction data in order to identify any patterns that may indicate coordinated fraud activities.

Data Preparation

The first step in our data preparation process will be feature selection and engineering, followed by cleaning and normalization/scaling of the data.

Feature Selection & Engineering:

- *Transaction Amount (TransactionAmnt):* Normalize transaction amounts to ensure consistent scaling across our dataset.
- *Card Information (card1 - card6)*: Encode categorical card information using techniques such as one-hot encoding to convert this data into numerical features.
- *Address Data (addr1, addr2):* Normalize address data and encode geographic locations.

- *Time Between Transactions (D1 - D15):* Engineer features in order to capture the time that has elapsed between transactions, as well as the frequency of transaction within specific periods of time.
- *Vesta Engineered Features (Vxxx):* Utilize these features directly as they capture the many complex patterns related to fraud.

After our feature selection and engineering steps are completed, we will then move on to cleaning our data, which will include handling any missing values utilizing imputation methods, and identifying/handling any outliers that may skew our results.

Normalizing and scaling will be the next crucial step, in order to ensure all features contribute equally to the clustering process.

Modeling Techniques

We will apply and compare several different clustering algorithms in order to identify any potential fraud patterns. Each technique will be evaluated afterwards for its effectiveness in identifying coordinated fraud activities.

Clustering Techniques:

- **K-Means Clustering:**
    - This algorithm will group the data into k-clusters by minimizing the variance found within each cluster. It is simple and efficient, however requires a specified number of clusters to be stated in advance. The steps we will take are as follows:

1.) Initialize K-cluster centroids (the center point of a cluster) randomly.
2.) Assign each transaction to its nearest centroid.
3.) Recalculate centroids based on the assigned transactions.
4.) Repeat steps 2 and 3 until convergence is reached.

Evaluation: We will determine the ideal number of clusters by plotting the within-cluster sum of squares (WCSS) against the number of clusters, then identifying the point where the rate of decrease slows.

- **DBSCAN (Density-Based Spatial Clustering of Applications with Noise):**
    - This algorithm identifies potential clusters based on the density size of their points, allowing for the detection of abnormally-shaped clusters, otherwise known as outliers. It varies from K-Means Clustering in the sense that the number of clusters does NOT need to be specified beforehand. The procedure will be as follows:

1.) Define *epsilon* (the maximum distance between two samples that considers them to be in the same neighborhood), and *min_samples* (the number of samples in a neighborhood in order for a point to be considered a core point).
2.) Identify the core, border and noise points.
3.) Form our clusters by connecting the core points to their corresponding neighbors.

Evaluation: We will properly adjust ‘epsilon’ and ‘min_samples’ to balance the detection of meaningful clusters, as well as identify any noise.

- **Hierarchical Clustering:**
    - This method builds a hierarchy of clusters by either splitting or merging them. The steps for this method are as follows:

        1.) Treat each transaction as its own separate cluster.
        2.) Merge the closest pair(s) of clusters.
        3.) Repeat step 2 until all transactions are found within a single cluster.
        4.) Use a tree diagram to visualize the hierarchy and determine the ideal level to cut the tree for the most ideal clusters.

    Evaluation: We will analyze our tree diagram and use silhouette scores to determine the quality of the clusters at different levels within the hierarchy.

Analysis

After applying the clustering algorithms, we will analyze these clusters and identify patterns that may be indicators of coordinated fraud activities. This analysis will involve examining the distribution of fraudulent transactions found within each cluster, as well as identifying any found common characteristics among these transactions. Some of these characteristics may include similar geographic locations, time intervals, and/or transaction

amounts. We will also analyze any outliers that do not fit well into any clusters, as this may be indicative of individual instances of fraud or other suspicious activities that may be worth further investigating.

We will be creating several visualizations, such as cluster heatmaps, scatter plots, and tree diagrams, in order to visualize the relationships between the transactions found both within and between clusters.

By applying various clustering techniques and analyzing the resulting clusters, we strive to uncover any hidden patterns or relationships within the transaction data that may indicate coordinated fraud activities. The insight gained from this analysis will contribute to developing more effective fraud detection strategies, as well as improving the security of e-commerce platforms.

Data Visualization and Analysis

Data Analysis

Model Description

We began by merging the train_transaction.csv and train_identity.csv files using a left join on the TransactionID column. This created a comprehensive dataset containing both transaction and identity information for each record. Merged dataset included 434 features. Data containing missing values that are greater than 60% of the whole feature does not provide enough information for our analysis. Thus, features with more than 60% missing data were removed from the created dataset. The dataset then were separated into numerical and categorical variables. The remaining missing values were imputed using KNN imputation for numerical

features and mode imputation for categorical features. KNN imputation replaces missing values with the average of the values from the 5 neighbors based on other features, while mode imputation replaces missing values with the most frequent category in that feature. After removing features with high missing values, we randomly sampled 25% of the remaining records (reducing from approximately 540,000 to 150,000) due to computational limitations.

We first trained five models (XGBoost, Random Forest, Decision Tree, Logistic Regression, and a Dummy Classifier) using all available features. To further improve model performance, we identified and removed highly correlated features, creating a reduced feature set. We then retrained the models on this reduced dataset. To assess model generalizability, we employed 10-fold cross-validation. 80% of the data was selected randomly for training and 20 % of the data were selected for testing our models. We then applied three feature selection techniques (SFS, FI, and PCA) using selected models to identify the most relevant features for fraud detection. Models are then evaluated using 10 fold cross validation the asses model generalizability. Features are selected using sequential features selection, feature importance from XGboost and principal component analysis.

For our regression models, the data was transformed by applying standardization to our numerical features and a dummy classifier to our categorical features to create our linear regression and xboost regressors. We also applied the use of Recursive Feature Elimination to reduce our features and applied parameter hypertuning with the help of bayesian optimization and randomized search to improve our models.

For our clustering analysis supporting RQ3, we utilized various techniques, including K-Means, HDBSCAN, and Hierarchical Clustering. Each method required preprocessing steps, such as feature selection, normalization, and dimensionality reduction. The dataset was

transformed in order to address any skewness, as well as to ensure optimal performance of clustering algorithms. K-Means Clustering was performed on the reduced dataset, and the number of clusters was determined using the Elbow Method. HDBSCAN was applied to aid in the identification of clusters with various densities. Hierarchical Clustering was used to understand the hierarchical structure of the data, and dendrograms were created for visualization. The resulting clusters were then analyzed for any patterns indicative of coordinated fraud, such as simultaneous transactions from different locations or unusual spending behaviors.

Model Results

RQ1: How can advanced feature engineering and selection techniques, such as Recursive Feature Elimination (RFE), Feature Importance from Gradient Boosting, and Principal Component Analysis (PCA), enhance the accuracy of fraud detection models?

The following table represents the results from each technique for feature selection using XGboost model:

Metric Model	Accuracy	Precision	ROC AUC	Recall
Feature Importance	0.866	0.519	0.479	0.122
PCA	0.890	0.741	0.826	0.323
Sequential Feature Selection	0.863	0.468	0.481	0.105

Main focus for the accuracy of the test was ROC AUC score which denotes area under receiver operating characteristic curve. An area of 1 represents the perfect test and 0.5 represents a test that is not better than randomly guessing if a transaction is fraud or not. Only PCA

technique achieved a score (.826) that had accurate predictions. Other methods to check model performance are the following scores:

Accuracy: It defines the ratio of correctly predicted instances. Higher the score the better the model yet it does not differentiate between fraudulent and non fraudulent cases.

Precision: High precision indicates that when the model predicts a transaction as fraudulent, it is likely correct. This is essential for fraud detection in order to reduce false positives, which may be expensive and inconvenient for clients and the company.

Recall: High recall ensures that the model catches most of the fraudulent transactions, which is critical to avoid missing any fraud cases. It is the ratio of true positive predictions (fraud) to the total actual positives.

In order to select the best performing model, we evaluated DummyClassifier, DecisionTree, XGBoost and Random Forest using 10 fold cross validation on training and test sets with 168 features that were left after removing 121 highly correlated features. The features are shown in Appendix.

Sequential Feature Selection (SFS)

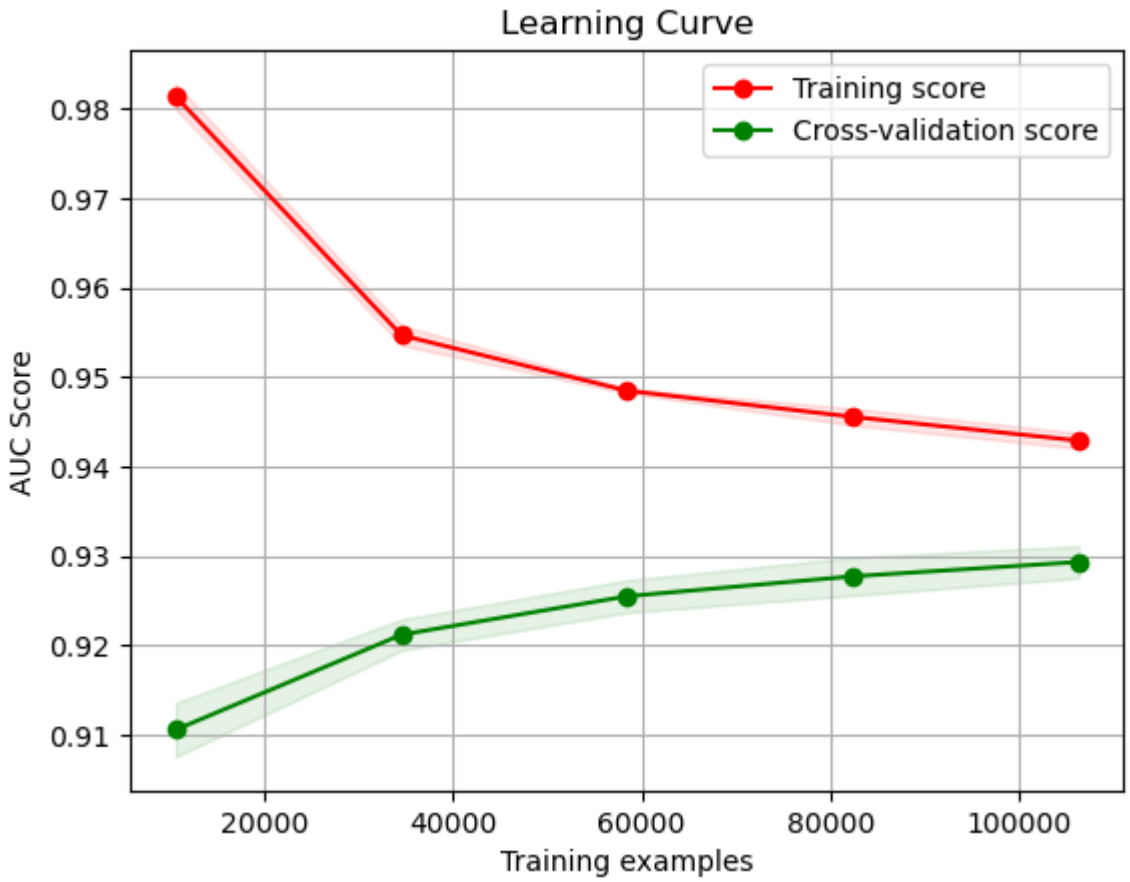
We employed Sequential Feature Selection (SFS) with backward elimination to identify a subset of features that would optimize the performance of the XGBoost model. Using 5-fold cross-validation and the 'roc_auc' scoring metric, SFS selected 43 features. A new model is trained using those features using 10 fold cross validation. Despite reducing the feature space, the resulting model exhibited poor performance. For this model the mean AUC score was .48,

precision was .46 and recall was .1. These scores show that sequential feature selection is ineffective for selecting features from a large large pool to create an effective model. The model performed worse than random chance in distinguishing between fraudulent and non-fraudulent transactions.

The Learning curve and ROC graph for each fold can be examined in Figures 2 and Figure 3 respectively. The learning curve suggests the model is overfitting with selected features, which can be also explained by the large gap between training (.96) and test (.48) AUC scores. This can explain the suboptimal performance of the model. Additionally, since SFS is a greedy algorithm that selects features based on immediate benefits rather than overall performance, our model's performance could be further compromised by these suboptimal features, especially in high-dimensional datasets with complex interactions like fraud detection.

Figure 2

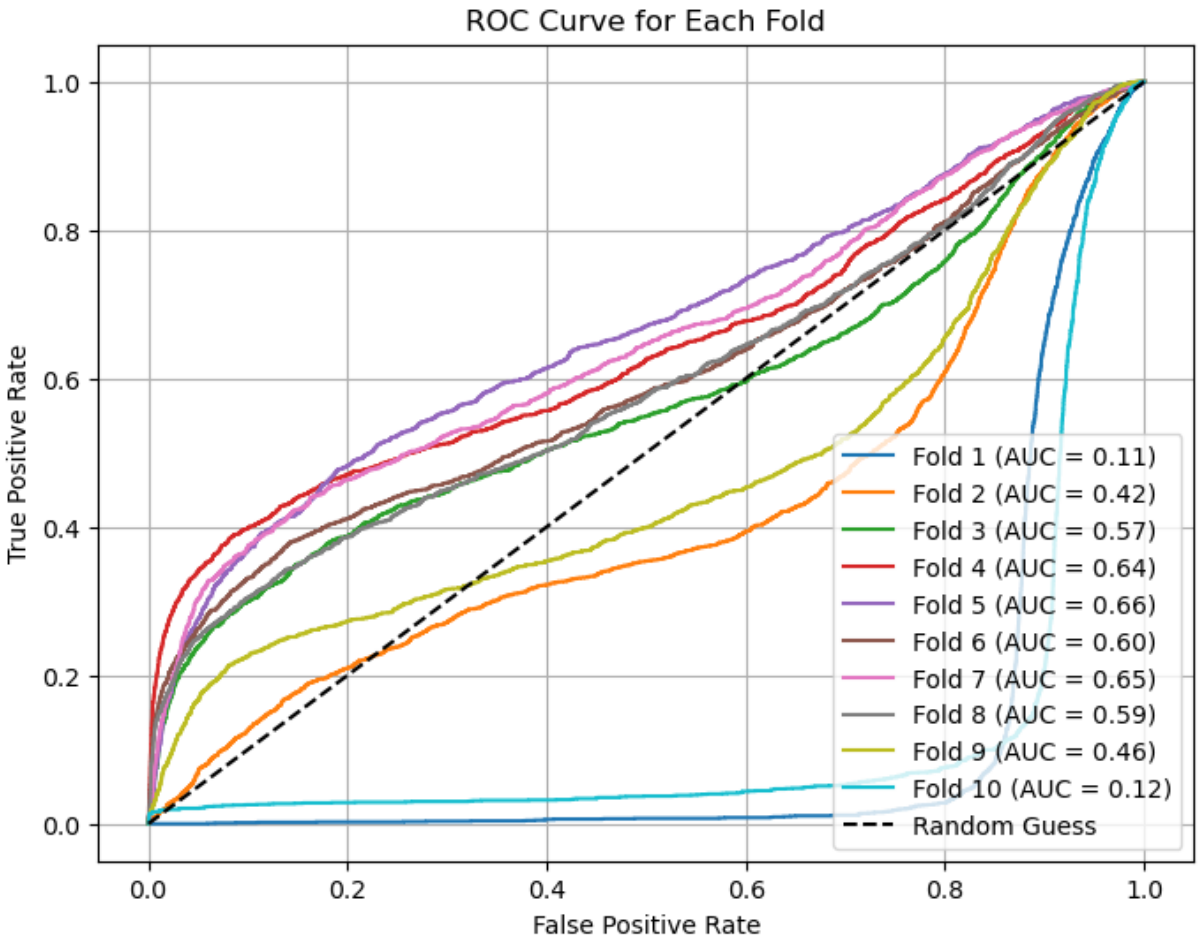
Learning Curve of SFS.



The graph shows a learning curve. The learning curve demonstrates overfitting in the model, as evidenced by the high performance on training data and significantly lower performance on cross-validation data.

Figure 3

ROC_AUC Graph



This graph presents the ROC curves for ten different folds in a model validation process. AUC values, also provided, quantify the overall performance of the model for each fold. The dashed line represents a random classifier, serving as a baseline for comparison. The closer a curve is to the top left corner, the better the model's performance for that fold.

Feature Importance (FI)

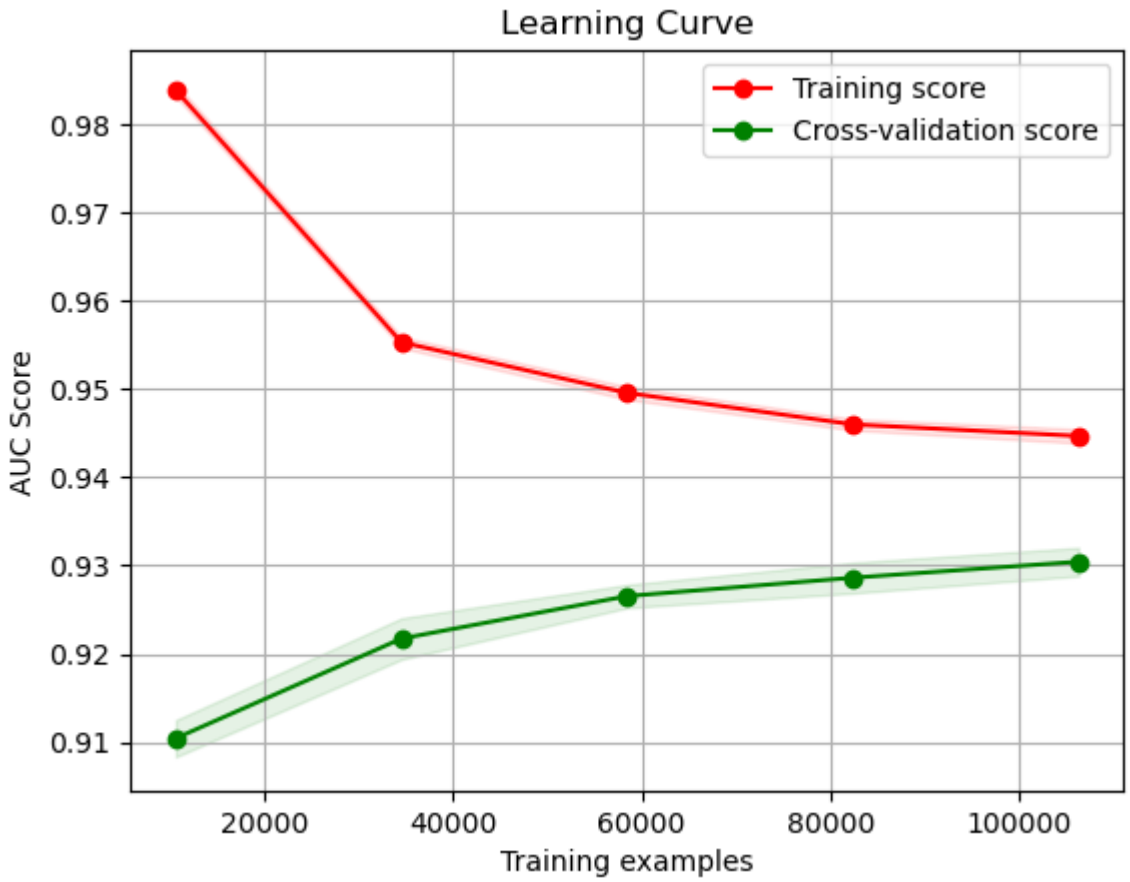
For feature importance, an XGBoost classifier was trained with all features (168) using AUC score evaluation metric. The feature importance scores were extracted from the trained

XGBoost model. These scores indicate the significance of each feature in predicting the target variable. A cumulative importance threshold of 90% was set to select the most influential features. This threshold ensures that the selected features contribute to 90% of the total importance. Selected features with their scores can be seen in appendix N. A new model with selected features was trained and evaluated using AUC and 10 fold cross validation. For this model the mean AUC score was .48, precision was .46 and recall was .1. However, similar to SFS, the XGBoost model trained on these features did not exhibit satisfactory performance. This score, again, falls below the random chance baseline of 0.5, indicating that the selected features were not effective in improving the model's discriminatory power.

The Learning curve and ROC graph for each fold can be examined in Figure 3 and Figure 4 respectively. The learning curve suggests the model is overfitting with selected features, which can be also explained by the large gap between training (.96) and test (.48) AUC scores. Tree-based models like XGBoost can bias feature importance scores towards features with many unique values. This bias might have led to the selection of features that don't genuinely contribute to fraud detection. Additionally, Feature importance scores typically reflect the individual contribution of each feature, without considering potential interactions between features. Important interactions might have been missed during the selection process.

Figure 3

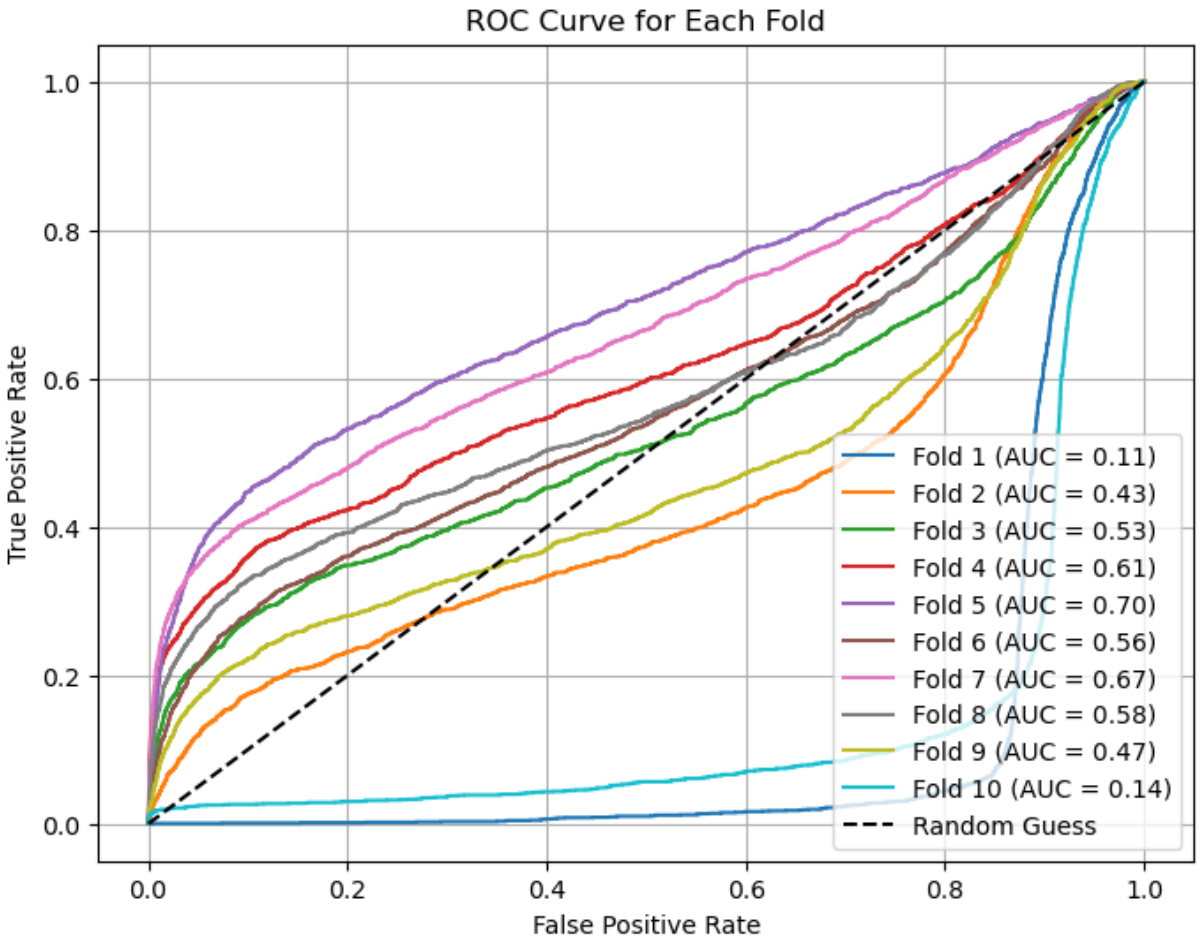
Learning Curve of SFS.



The graph shows a learning curve. The learning curve demonstrates overfitting in the model, as evidenced by the high performance on training data and significantly lower performance on cross-validation data.

Figure 4

ROC_AUC Graph



This graph presents the ROC curves for ten different folds in a model validation process. AUC values, also provided, quantify the overall performance of the model for each fold. The dashed line represents a random classifier, serving as a baseline for comparison. The closer a curve is to the top left corner, the better the model's performance for that fold.

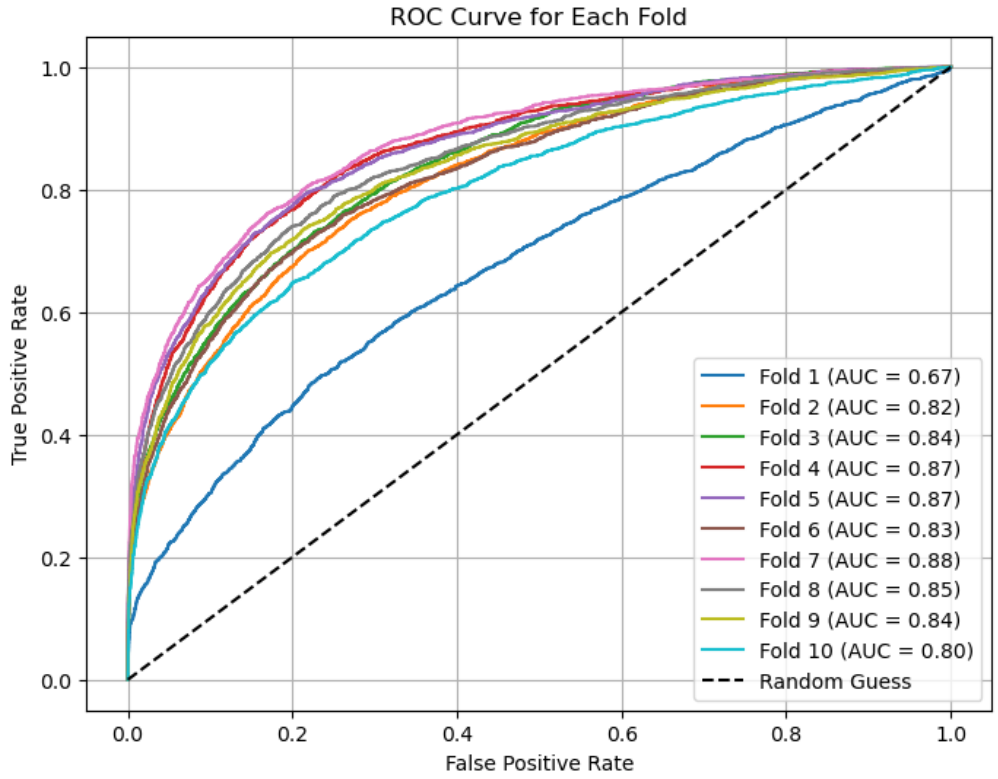
Principal Component Analysis (PCA)

For PCA, we trained the PCA model using all our feature sets. By setting a threshold of retaining 98% of the cumulative variance, seen in Figure 5, we selected the optimal number of principal components 86 to maintain a high level of data variability while reducing

dimensionality. The XGBoost classifier model was trained using the selected components and evaluated using 10 fold cross validation. For this model AUC score was .83, precision was .74 and recall was .3. Those scores are strong support for how PCA improved the effectiveness of the model.

The ROC graph for each fold can be examined in Figure 6. The gap between training (.95) and test (.83) is smaller than the other models. PCA provided the strongest model for predicting Fraud cases.

Figure 6
ROC_AUC Graph



This graph presents the ROC curves for ten different folds in a model validation process. AUC values, also provided, quantify the overall performance of the model for each fold. The dashed line represents a random classifier, serving as a baseline for comparison. The closer a curve is

to the top left corner, the better the model's performance for that fold.

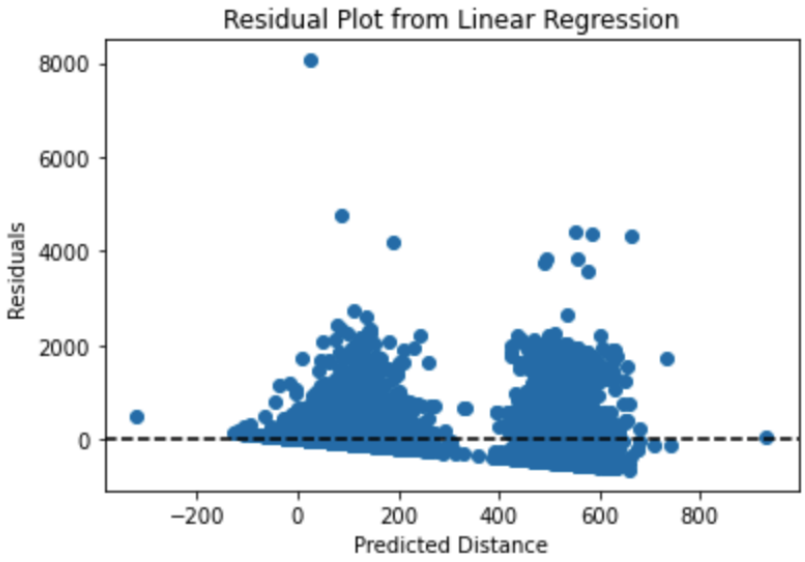
RQ2: How can machine learning models predict the distance where the transaction was made based on various factors, including transaction amount (TransactionAmt), card information (card1 - card6), address (addr1, addr2), time between transaction(D1-D15) and security features (Vxxx)?

Linear Regression

To start off with our baseline, we made a linear regression and fitted our data as it was to see how well the model would do. The regression model had a test set RSME of 248.99 which means that model is not fitting the data well.

Figure 1

Linear Regression Residual Plot:



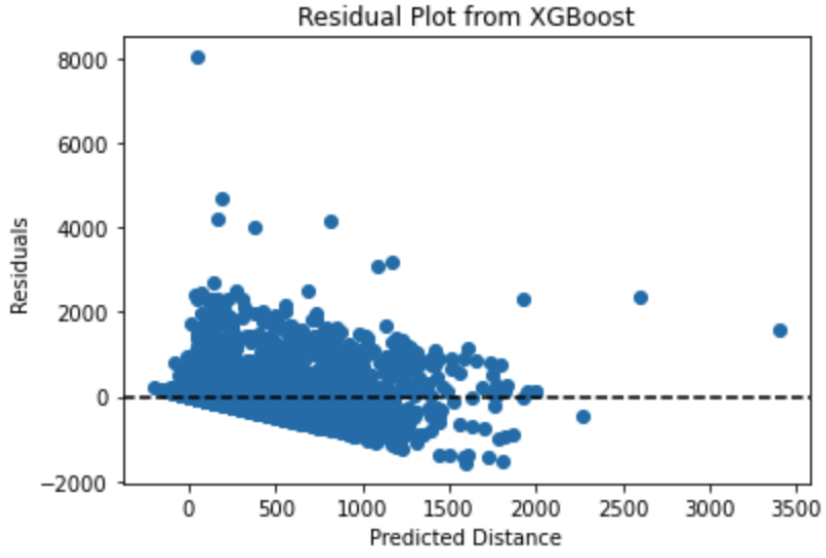
As we can see from the residual plot, the majority of the points are far from the y= 0 line and by having an accuracy score 0.17 confirms us that the model is not doing well and tells us that the model is guessing randomly by having an accuracy score lower than .5. After reviewing the discovered information, we decided to reduce the features used in our regression, to see if we could reduce our RMSE. However, it wasn't helpful since it increased the RMSE value instead of lowering it.

XGBoost Models

After the creation of the linear regression, we made an XGBoost model with the same specifications as the linear regression to compare them before performing parameter hypertuning. After the fitting of the model, the RMSE value was 223.79.

Figure 2

XGBoost Residual Plot:



Compared to the linear regression, this model did a little better, the residual looks more clustered together. However, we still have the same problem since the RMSE is too high, the points in residual are still far from the y=0 line and it has an accuracy of .33.

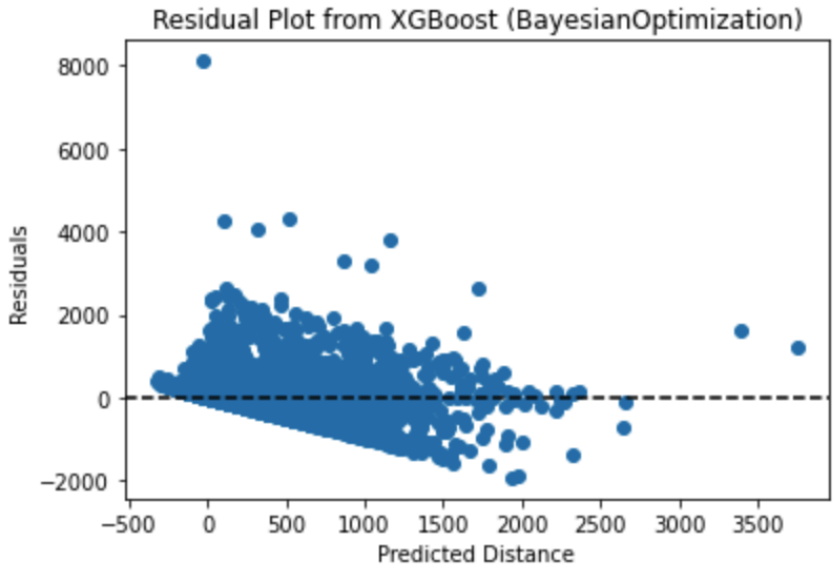
One of the biggest problems in creating a xgboost model without any hyper tuning is overfitting or underfitting the data so instead of having a generalized model, it creates a memorized model. To solve this issue we are going to hypertune our xgboost with bayesian optimization with the following parameters and values:

- n_estimators:(3, 10)
- learning_rate:(0.01, 0.3)
- max_depth:(100, 1000)
- subsample:(0.6, 1.0)
- colsample_bytree:(0.6, 1.0)

We are also using RMSE for evaluating our model with 10 iterations and 3 cross-validations. After the compiling of our model, RMSE being 231.45 tells us that it is having a performance difference between our linear and our first xgboost models.

Figure 3

XGBoost(BayesianOptimization) Residual Plot:



The residual plot and an accuracy of .29 tells us that the model it's not performing well, but it is a bit less spread out and the accuracy score is less compared to the first xgboost model.

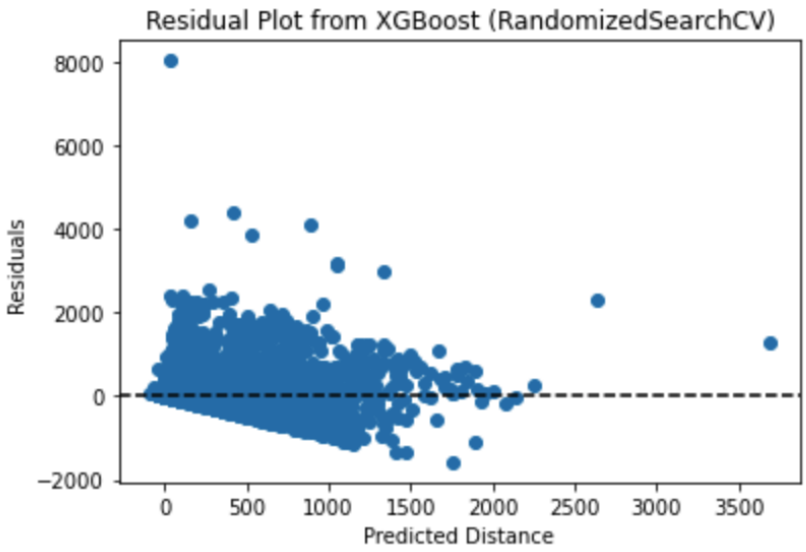
Since both of our models were close to each other and the improvement wasn't significant, we decided to try another method of hypertuning so for our last model we are using randomized search with the following parameters and values:

- n_estimators:[1000, 2000, 3000]
- learning_rate:[0.01, 0.1, 0.3]
- max_depth:[3, 5, 7]
- subsample:[0.7, 0.8, 0.9]
- colsample_bytree:[0.7, 0.8, 0.9]

As for evaluation scoring, iterations, and cross-validation were left the same for this model.

Figure 4

XGBoost(RandomizedSearchCV) Residual Plot:



As a result this model had the lowest RMSE score for the testing data with a value of 217.68. Even though this model had the lowest score, we still faced the same issues as the previous models. Based on figure 4, the residuals are closer but far from the x-axis and the accuracy value is 0.36 which is the highest one we had so far.

RQ3: How can clustering techniques be used to identify groups of transactions that may be part of coordinated fraud activities?

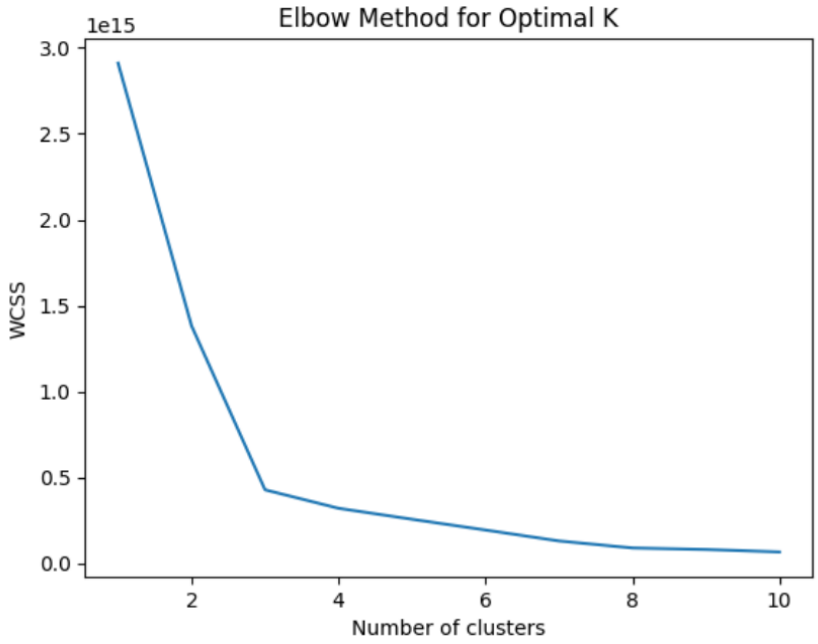
Clustering Analysis

K-Means Clustering

K-Means Clustering was applied to the PCA-reduced dataset with five clusters. The cluster labels were then stored in the dataframe. The Elbow Method plot indicated that five clusters provided an optimal balance between within-cluster sum of squares (WCSS) and the number of clusters.

Figure 1

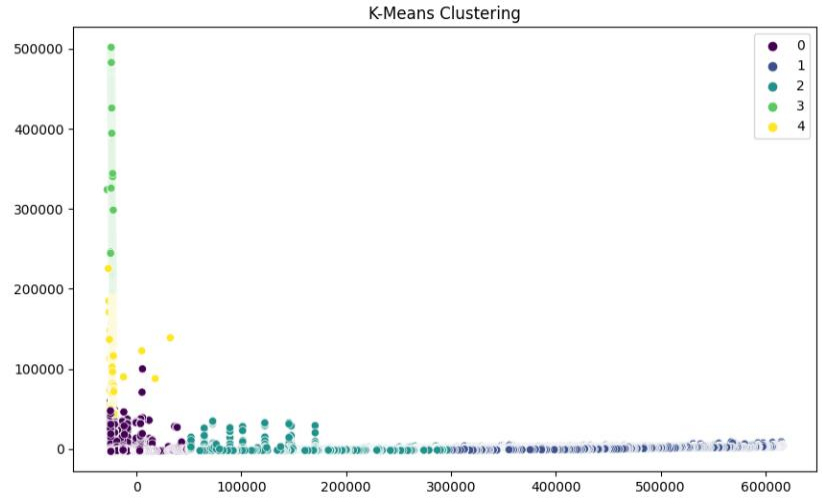
Elbow Method plot showing the optimal number of clusters for K-Means:



The silhouette score for K-Means clustering was calculated as 0.856, which indicates a good clustering structure containing well-separated clusters.

Figure 2

K-Means Clustering Scatter Plot:

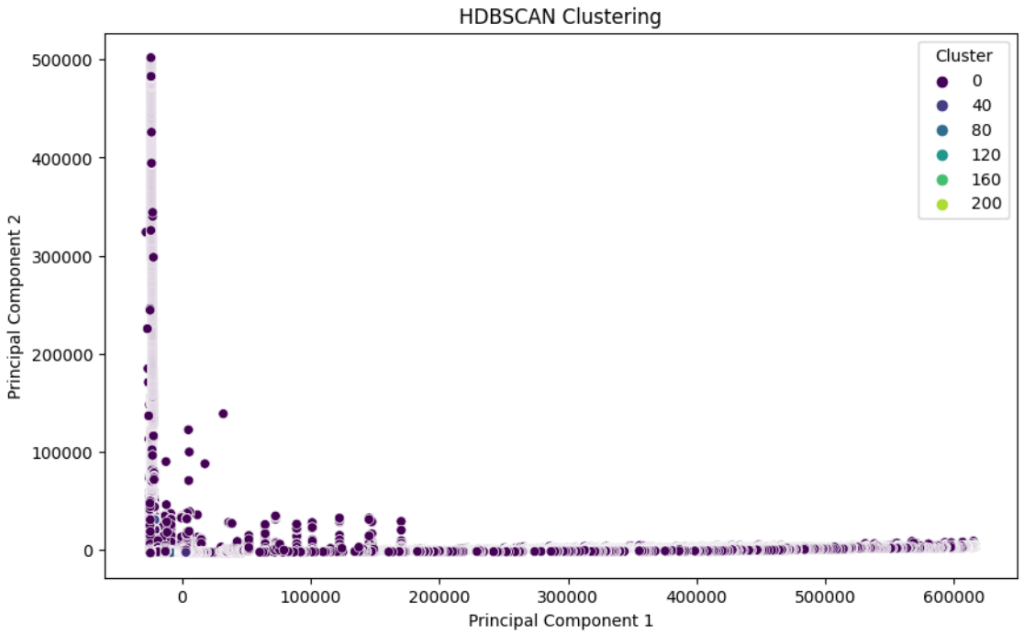


HDBSCAN Clustering

HDBSCAN was used to identify clusters of varying density with ‘min_samples’ set to 10 and ‘min_cluster_size’ set to 500. This density-based algorithm identified a large number of clusters, including noise points (labeled as -1). The silhouette score for HDBSCAN was -0.184, suggesting that the clustering did not perform well and may need further parameter tuning.

Figure 3

HDBSCAN cluster plot with varying densities:



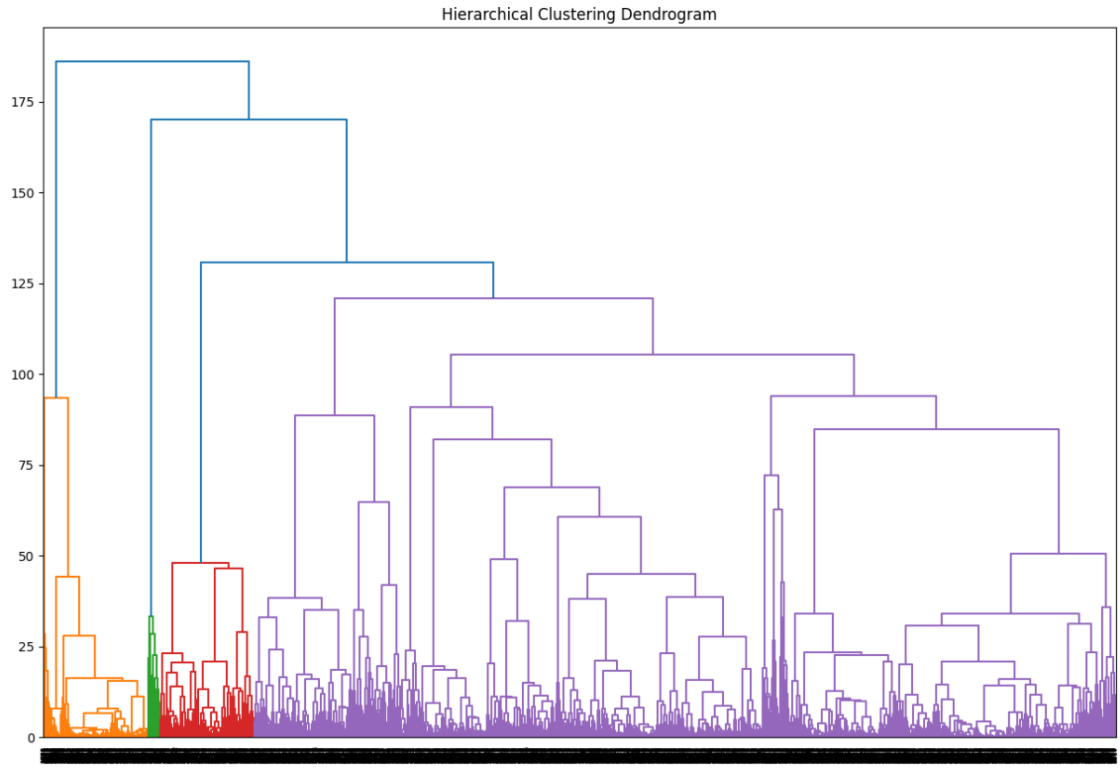
The scatter plot shows clusters in similar positions to those identified by K-Means, suggesting similar groupings despite HDBSCAN's ability to handle varying densities.

Hierarchical Clustering

Due to memory limitations, we randomly sampled 20,000 data points for hierarchical clustering. Agglomerative clustering was applied with five clusters, and the cluster labels were stored inside the sampled dataframe. The hierarchical clustering dendrogram provided insights into the hierarchical structure of the clusters.

Figure 4

Dendrogram from hierarchical clustering:



The silhouette score for hierarchical clustering was calculated as -0.064, indicating that this method also requires further refinement in order to improve clustering performance.

Figure 5

Image listing the silhouette scores for each of the three clustering methods used:

```
K-Means Silhouette Score: 0.8564298897506002
HDBSCAN Silhouette Score: -0.18430016878234834
Hierarchical Silhouette Score: -0.0637329143989308
```

Evaluation and Visualization

Evaluating Clustering with WCSS

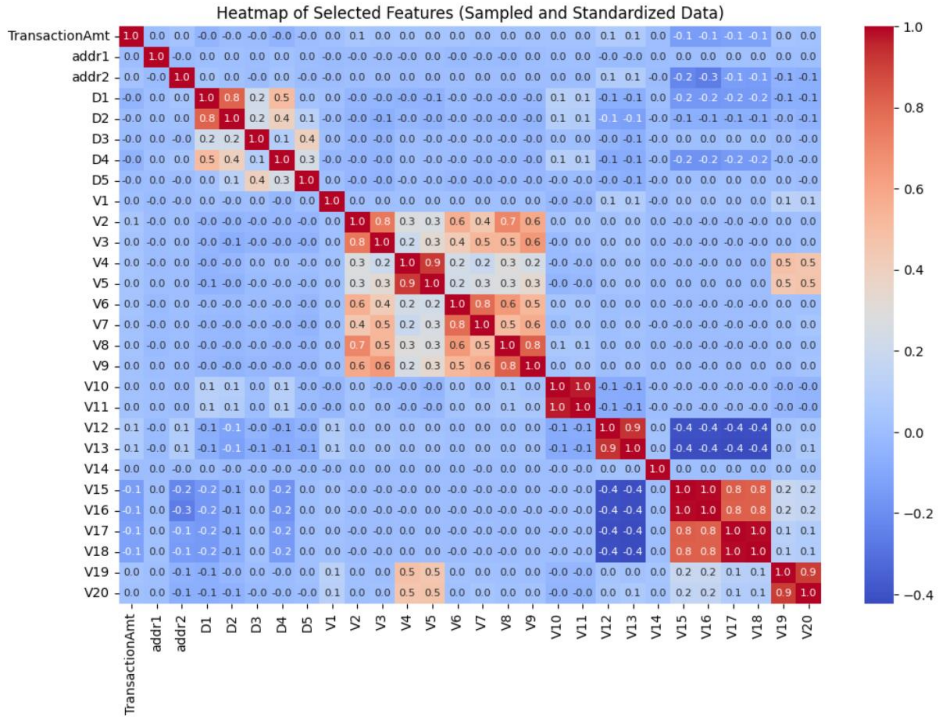
The WCSS was calculated for different numbers of clusters in order to determine the optimal number using the Elbow Method. This method aids in identifying the point where adding more clusters does not significantly reduce the WCSS, suggesting an optimal number of clusters. **Refer to Figure 1 for the WCSS for different numbers of clusters.*

Visualizing the Clusters

- **Scatter Plot for K-Means:** A scatter plot of the first two principal components was created, colored by K-Means cluster labels, showing distinct clusters.
    - **Refer to Figure 2 for K-Means clustering scatter plot*.
- **Scatter Plot for HDBSCAN:** A scatter plot of the first two principal components was created, colored by HDBSCAN cluster labels, showing clusters with varying densities.
    - **Refer to Figure 3 for HDBSCAN clustering scatter plot.*
- **Heatmap of Selected Features:** A heatmap was plotted to visualize the correlations between a subset of selected features, providing insight into the relationships within the data.

Figure 6

Heatmap of Selected Features (Sampled and Standardized Data):



- **Dendrogram for Hierarchical Clustering:** A dendrogram was created to visualize the hierarchical structure of the clusters, highlighting the clustering relationships within the dataset.
    - **Refer to figure 4 for a diagram of hierarchical clustering.*

Conclusion

The analysis provided us with valuable insights into the clustering structure of the dataset. K-Means clustering showed promising results with a high silhouette score of 0.856, indicating well-separated clusters. However, hierarchical clustering and HDBSCAN, with respective silhouette scores of -0.184 and -0.064, require further parameter tuning or alternative approaches in order to improve their performance.

Results

RQ1: How can advanced feature engineering and selection techniques, such as Recursive Feature Elimination (RFE), Feature Importance from Gradient Boosting, and Principal Component Analysis (PCA), enhance the accuracy of fraud detection models?

Our analysis reveals a complex relationship between feature engineering techniques and fraud detection model performance. We learned that some feature selection techniques like sequential feature selection and feature importance, while not improving the model performance, lowers model performance to less than random chance.Their scores indicate a failure to effectively capture the signals of fraud within the data. Given the results seen in the following table, the XGBoost model trained on all features shows its effectiveness in correctly identifying fraudulent transactions and minimizing false negatives.While its Test AUC (0.94) is only slightly higher than its Train AUC (0.93), suggesting potential underfitting, it still outperforms all other feature engineering approaches in terms of overall performance. This suggests that leveraging all available features, despite potential noise, can be beneficial for maximizing fraud detection performance.

Feature Selection Technique	Number of Features/Components	Test Accuracy	Test Precision	Test Recall	Test ROC AUC
All Features	168	0.93	**0.88**	**0.63**	0.94
Sequential Feature Selection (SFS)	43	0.86	0.47	0.11	0.48
Feature Importance (FI)	85	0.87	0.52	0.12	0.48
Principal Component Analysis (PCA)	86	0.89	0.74	0.32	**0.83**

To enhance fraud detection accuracy, we recommend considering PCA as a feature engineering technique. PCA, while not achieving the same raw performance as the "all features"

model, is the only technique that provided improvements. It scored .83 AUC using only 86 components proving its generalizability. This highlights the effectiveness of PCA in reducing dimensionality and capturing underlying data structures relevant to fraud. Reduced features increase computational efficiency which would be beneficial for handling real time data. However, the lower Test Precision (0.74) and Recall (0.32) compared to the "all features" model indicate a potential trade-off between generalizability and precise fraud identification. In addition, it brings a struggle communicating the resulting features to non-technical executives.

RQ2: How can machine learning models predict the distance where the transaction was made based on various factors, including transaction amount (TransactionAmt), card information (card1 - card6), address (addr1, addr2), time between transaction(D1-D15) and security features (Vxxx)?

After completing our four different models, we got the following results:

```
RMSE Based on Model:
-----------------------------------------------------------------
Linear Regression: 248.9945910454159
XGBoost: 223.796282183509
XGBoost(BayesianOptimization): 231.4596745982548
XGBBoost(RandomizedSearchCV): 217.68759073220858


Proportion of fraudulent transactions in high-risk group:
-----------------------------------------------------------------
Entire data set: 0.15749598224535089
Test set: 0.14036418816388468
Linear Regression: 0.11415044713308785
XGBoost: 0.12021439509954059
XGBoost(BayesianOptimization): 0.14961101137043686
XGBBoost(RandomizedSearchCV): 0.11370967741935484
```

Since our main evaluation score for our models is RMSE, we can choose XGBoost with randomized search since it has the lowest value. However if consider our high risk threshold that was measured by taking the mean distance of the fraudulent transaction plus one standard

deviation so that we can flag high risk transaction that might be fraudulent depending on their distance then we can choose XGBoost with bayesian optimization since it had the best proportion of fraudulent transaction and was the third best RMSE.

Based on the result, we could recommend using XBoost with Bayesian optimization to predict the distance of a transaction. Also since our attempt to reduce the amount of features was a failure, we can recommend trying PCA to reduce the RMSE value because it did better in our models to predict if a transaction was fraudulent or not.

RQ3: How can clustering techniques be used to identify groups of transactions that may be part of coordinated fraud activities?

Our analysis employed K-Means, HDBSCAN, and hierarchical clustering techniques to identify coordinated fraud activities. K-Means clustering, applied after PCA, identified five distinct clusters with a silhouette score of 0.856, indicating well-separated and meaningful clusters. This suggests that K-Means effectively grouped transactions, potentially highlighting coordinated fraudulent activities.

HDBSCAN, a density-based clustering algorithm, identified a large number of clusters with varying densities, including noise points. Despite its ability to detect clusters of different shapes and sizes, HDBSCAN received a poor silhouette score of -0.184, indicating poorly defined clusters. HDBSCAN received the lowest silhouette score out of the three clustering methods. The clusters defined by HDBSCAN appeared in similar positions to those from K-Means, but the high number of clusters and poor silhouette score suggest it may not be suitable without further parameter tuning.

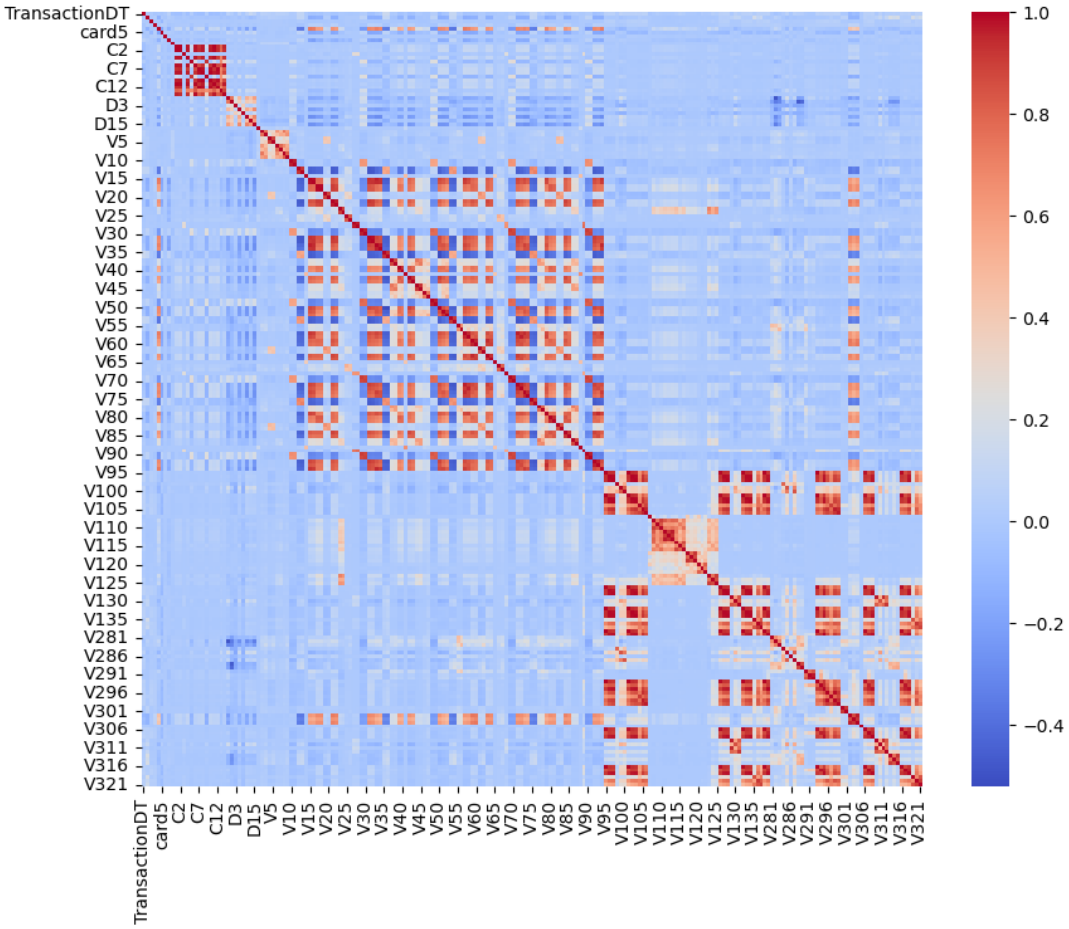
Hierarchical clustering, performed on a random sample due to computational limitations, showed clusters in similar positions to K-Means and HDBSCAN, but with less clarity. The silhouette score for hierarchical clustering was -0.064, indicating poorly separated clusters.

Overall, K-Means provided the most effective and distinct clustering results, making it the preferred method for identifying coordinated fraud activities in this dataset.

Data Visualization

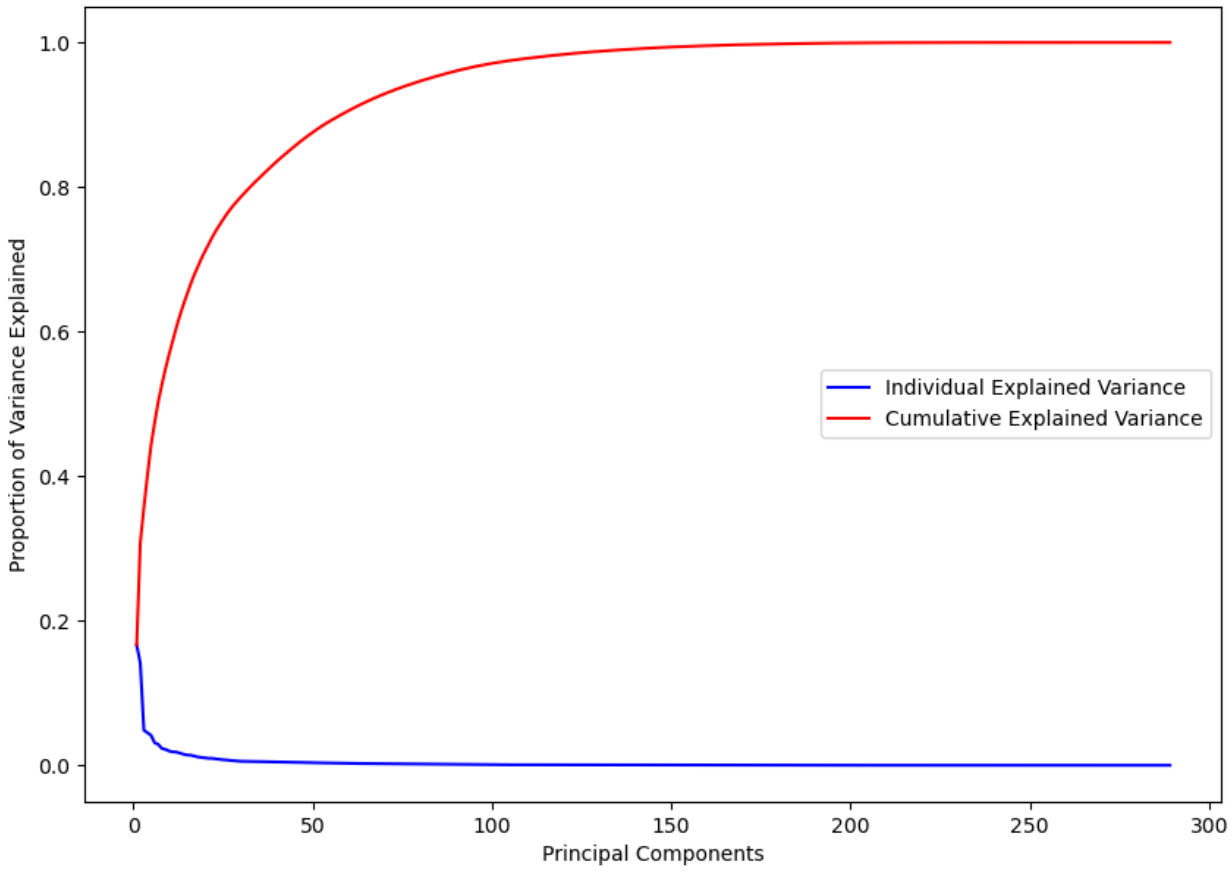
Feature Selection for Better Prediction

Our methodology included various feature selection techniques (Sequential Feature Selection (SFS), feature importance (FI) using XGboost, Principal Component Analysis (PCA)). Our data included lots of feature engineered variables our Vesta developed over the years. The first graph is a Heatmap that shows the highly correlated features that needed to be removed that will hinder feature selection techniques performance.



We were able to remove 121 features that had really high positive or negative correlation (Higher than 0.8 or lower than -0.8). After initial processing we proceed with feature importance using the XGboost model. Our model selected 85 features that contributed to the 90% of cumulative importance. Those features selected can be seen in appendix N. Sequential feature selection technique selected 43 features that provided the best model. We wanted to compare the features selected by both these models to see potential overlap. The following venn diagram shows the names of the features that are in either only SFS or FI or on both:

The table explaining the features that appeared in both SFS and FI are shown in appendix O. 114 components were chosen to capture 98% variance, as shown in the figure:



PCA being dimensionality reduction , there is a downside of not knowing which features are included. I check the loadings which indicate the importance of each feature in explaining the variance in the data captured by each principal component, as seen in the figure.

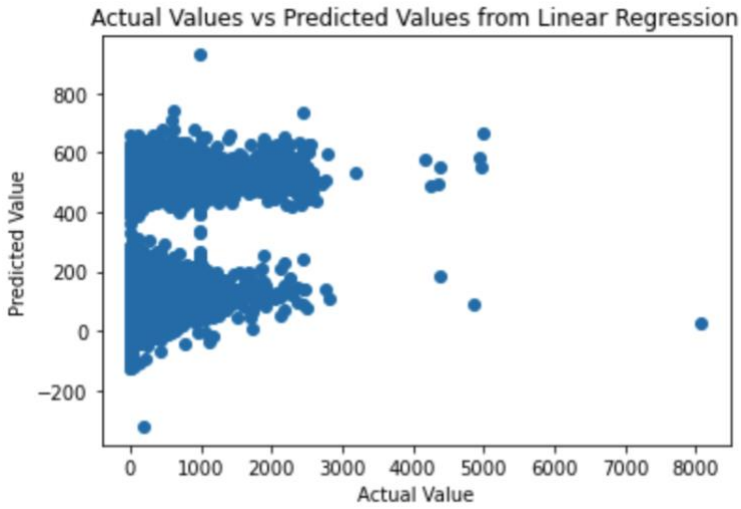
	PC1	PC2	PC3	PC4	PC5	PC6	\
TransactionDT	−0.023080	0.017519	0.051594	0.056723	0.039941	0.024736	
TransactionAmt	−0.038611	0.067510	0.040597	−0.010419	0.008633	0.011234	
card1	−0.003685	0.009580	0.003142	0.010351	−0.006163	−0.014145	
card2	0.017061	0.005308	−0.013879	0.011037	−0.009156	−0.029518	
card3	0.225130	−0.092173	−0.102390	−0.021157	−0.006552	−0.074573	
...	...	...	...	...	...	...	
M5_T	−0.016768	0.034901	0.020419	−0.004749	−0.006624	0.001596	
M6_T	−0.051531	0.001912	0.030103	0.023642	0.008246	0.033180	
M7_T	−0.009635	0.002858	0.006193	0.003255	−0.001914	0.007512	
M8_T	−0.026088	0.005000	0.014299	0.002968	0.000308	0.024277	
M9_T	0.008201	−0.011777	−0.009543	0.001731	−0.019787	−0.000813	

High absolute values in the loadings suggest that the feature has a significant impact on the component. Taking some of squares of each feature we saw some similar features with SFS

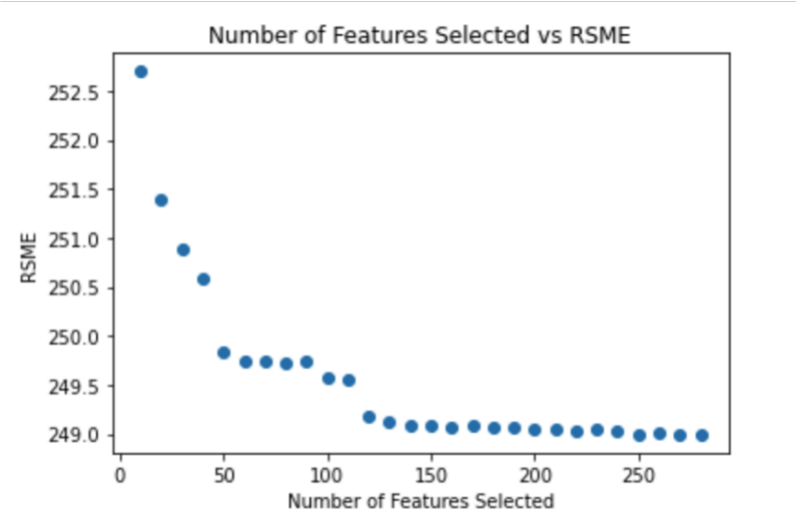
and FI intersection yet some top features that contributed to 98% variance were different, as seen in appendix P.

Linear Regression

For our second research question, it was decided to create a linear regression as a based model and to indicate a threshold for high risk fraudulent distance, we used the mean of the distance of the fraudulent transaction plus one standard deviation from our overall dataset. All of the features were included for our based model, which resulted in the following scatter plot:



As we can see the predicted values(red) are all together at the bottom with a significant difference with the actual values(blue). From this scatter plot we can conclude that the base model is not performing well and we can confirm it by having an Root Squared Mean Estimation (RMSE) of 248.99. Before moving on to our next model, we used Recursive Feature Elimination(RFE) to see if we could improve our model by eliminating some of the features.



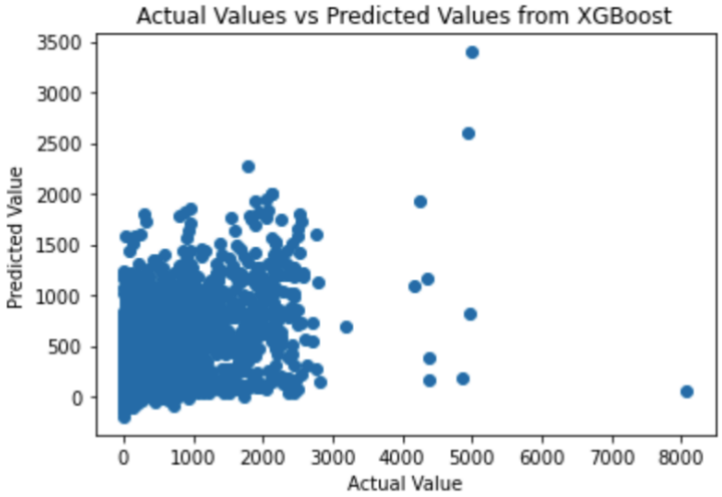
From the graph, we can see that the fewer features selected the greater the RSME became so it would be best to use all of our features for the following models. The actual values of the graph can be seen in appendix Q.

XGBoost Models

For this step, we created three different XGBoost: one without hypertuning, one with bayesian search, and one with randomized search. As we know XGBoost performs better that a

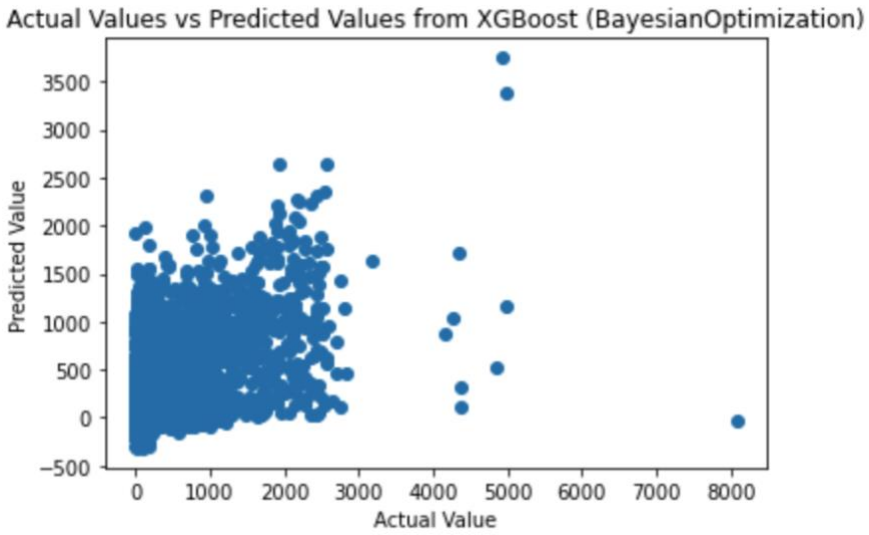
linear regression so we wanted to see the difference between them, which resulted in the following scatter plot for XGBoost without hypertuning:

Comparing this scatter plot with the linear regression's scatter plot, we can see that the

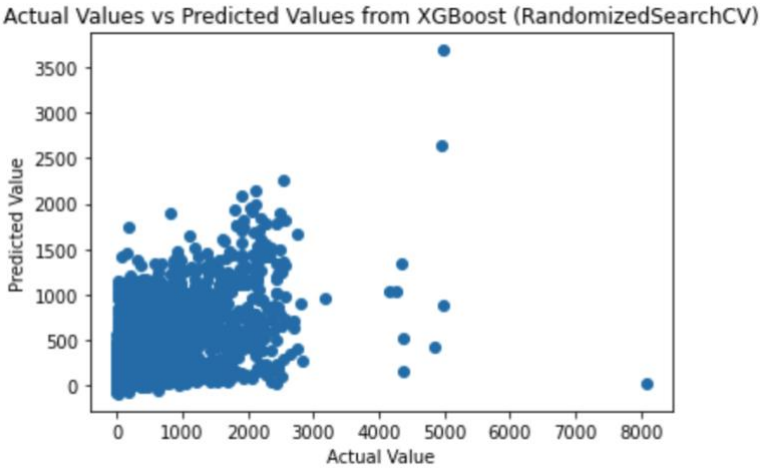


predicted values are a bit more spread out, however we can still see that the predicted values are far from the actual values but a bit closer than the linear regression. As for RMSE, since the actual and predicted are a bit closer we can say that the RMSE is a bit lower with 223.79 compared to 248.99 from the linear regression.

Next we used bayesian search to hypertune the parameters, which includes n_estimators, learning_rate, max_depth, subsample, and colsample_bytree of the XGBoost model , after the training of out model this was the result:



Comparing this scatter plot to the first XGBoost model, they look the same at first sight, however comparing their RMSE values the difference is shown. By using bayesian search, the RMSE value increased to 231.45 compared to our first XGBoost model, but it is still better than the linear regression by being 17 lower. Lastly, the third XGBoost model created was using randomized search using the same parameters and range from the second model as well as folds and iterations.



Looking at all of the scatter plots from the three XGBoost models, they look similar, the only difference from the first two to the last one is that the points look closer. The randomized search had a lower RMSE compared to the bayesian search.

After the training of four different models based on proportion of high risk and RMSE, the best model to predict the distance would be the XGBoost with bayesian search since it had the closest proportion to the test set and the RMSE was the third lowest.

Clustering Analysis

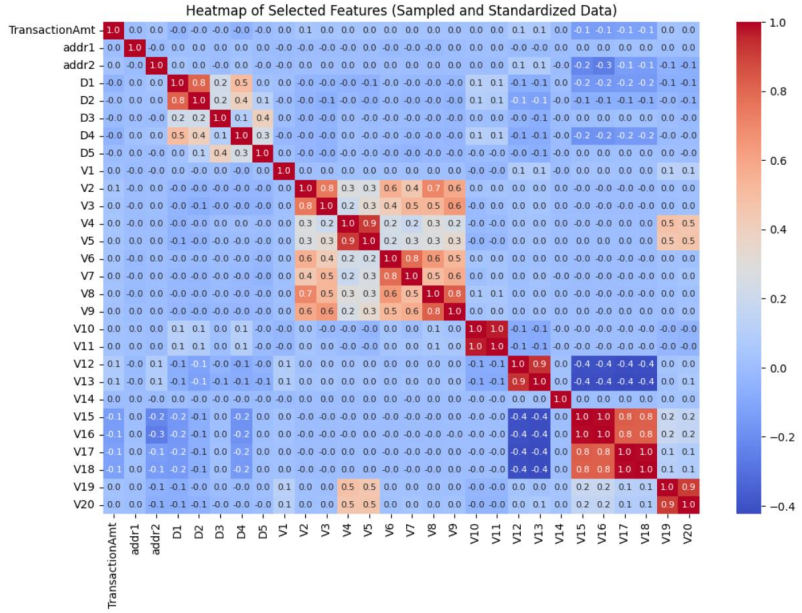
For our third research question, we aimed to apply clustering techniques in order to uncover any patterns or groupings within our transaction data. We used three different clustering methods: K-Means, HDBSCAN, and Hierarchical Clustering. The following visualizations provide insights into the clusters identified by these methods.

Feature Selection for Clustering

We began by selecting the relevant features for clustering, focusing on transaction amount, card information, address details, and specific D and V variables. This selection was based on their importance, as well as the potential impact they may have on the clustering results. We then sampled 20,000 instances, in order to ensure manageable memory shortage and standardize the data without centering to handle the sparse data appropriately.

Heatmap of Selected Features

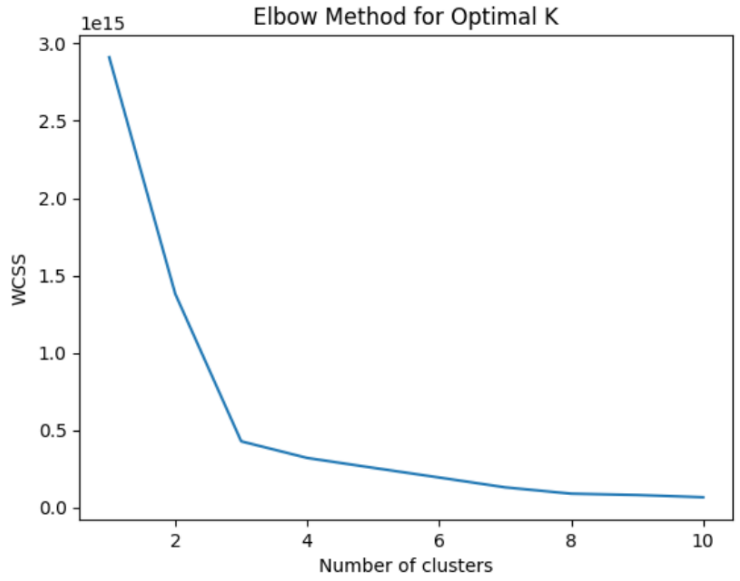
In order to visualize the various correlations between the selected features in the dataset, we created a heatmap. This visualization helps to identify the relationships between variables, which is crucial for understanding the structure of our data and informing further data analysis. We then standardized the data without centering, and computed the correlation matrix on the standardized data. Our heatmap revealed some strong correlations between certain card features and transaction amounts, indicating some potential patterns in transaction behavior. High correlations between address-related features suggest geographic factors in the area.



*The full-size correlation matrix is included in Appendix R for further reference.

Elbow Method for Optimal Clusters

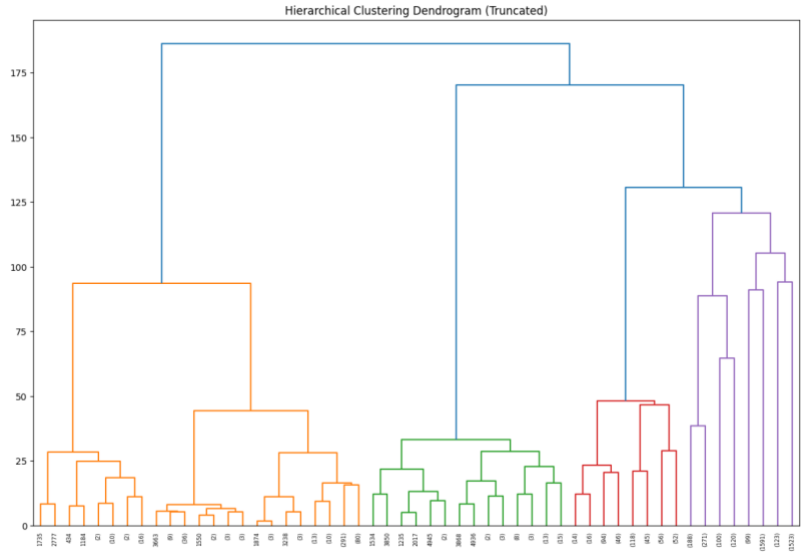
The elbow method was employed to determine the optimal number of clusters for the K-Means algorithm. This method involves plotting the within-cluster sum of squares (WCSS) against the number of clusters. The point where the WCSS starts to decrease more slowly (forming an 'elbow') indicates the optimal number of clusters. Our elbow plot revealed that the optimal number of clusters was around 5, as the reduction in WCSS began to level off beyond this point.



Hierarchical Clustering Dendrogram

For our next step, we created a dendrogram which illustrated the hierarchical relationships between our data points. This visualization helps to identify the number of clusters as well as understand the hierarchical structure of the data. We sampled 5,000 instances to manage memory storage, as well as standardized the data without centering. Using the linkage method with 'ward', we created our dendrogram. The dendrogram shows clear clusters,

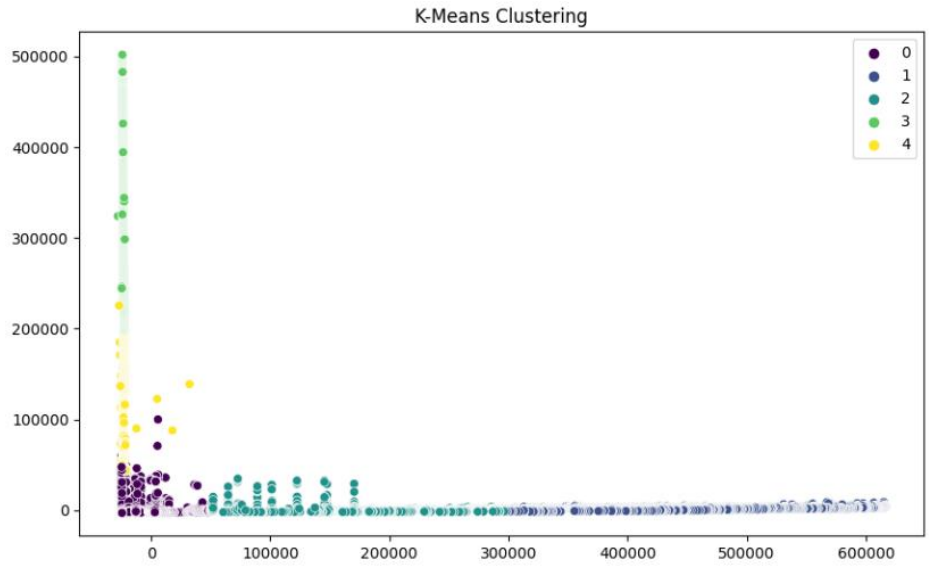
suggesting distinct groups within the data. The hierarchical structure of our dendrogram provided us with valuable insights into the relationships between clusters and their relative dissimilarities.



*The full-sized diagram is included in Appendix S for detailed examination.

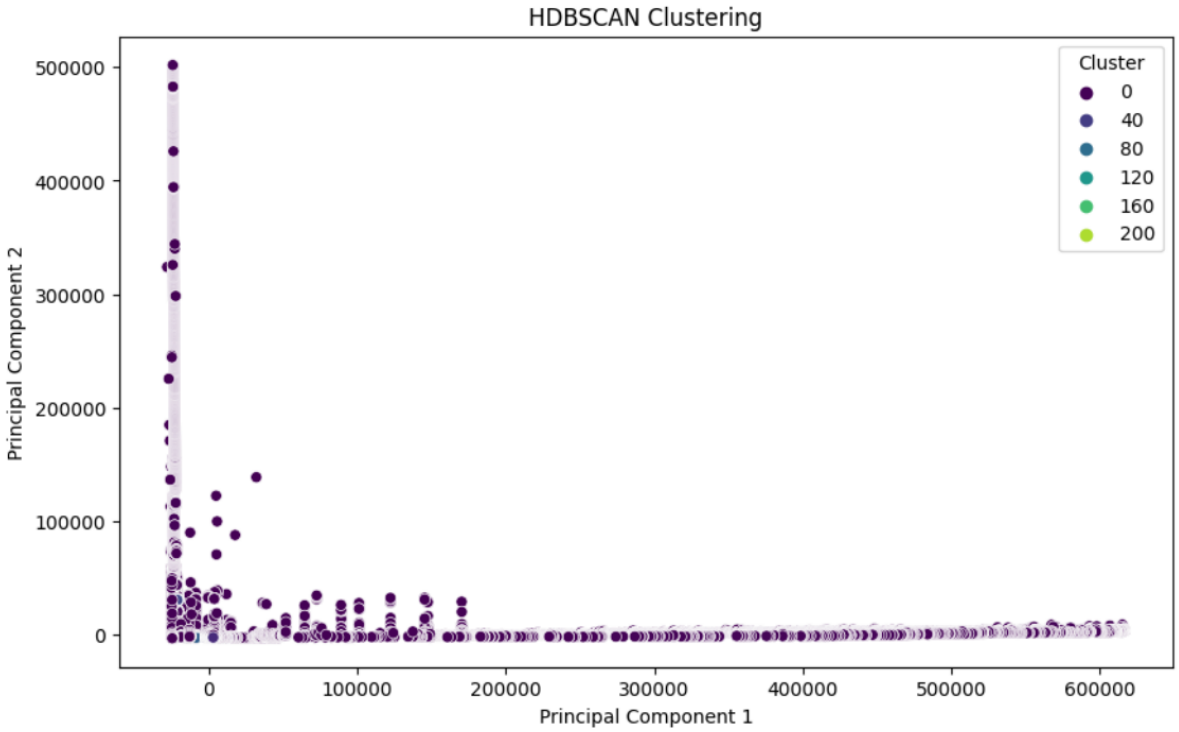
K-Means Clustering Scatter Plot

A scatter plot was created to visualize the clusters identified by the K-Means algorithm. This helped us better understand the distribution of data points within each cluster, as well as the separation between each cluster. We reduced the data to two principal components for visualization, using Principal Component Analysis (PCA) and applied K-Means clustering on the reduced data. The scatter plot shows well-defined clusters, indicating the K-Means effectively grouped similar data points together. The separation between clusters suggests distinct transaction patterns within the data.



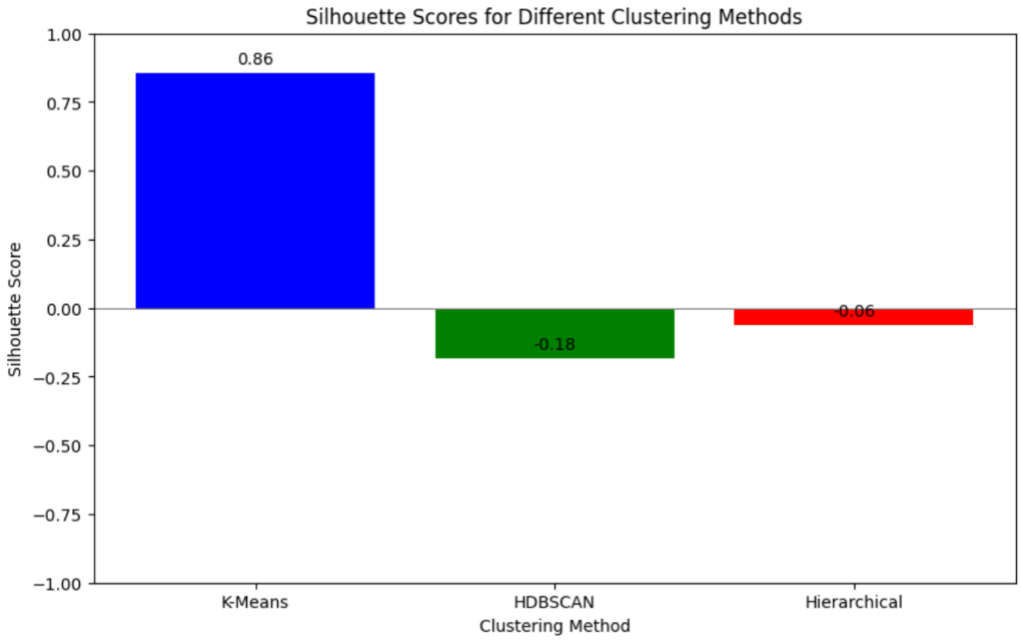
HDBSCAN Clustering Scatter Plot

In order to provide a visual representation of the clusters identified by the HDBSCAN algorithm, we created an HDBSCAN clustering scatter plot. Each point on the plot represents a transaction, projected onto the first two principal components, which aim to capture the most significant variance in the data. The different colors indicate distinct clusters, allowing us to observe how the data points are grouped based on their density. While HDBSCAN was able to identify numerous clusters, the scatter plot shows clusters in similar positions as those identified by K-Means. The lack of variation in colors suggests that both algorithms detected similar groupings within the data. However, the poor silhouette score for HDBSCAN indicates that its clustering performance is not effective, potentially due to noise or an overestimation of clusters. This visualization helps to compare the clustering results of HDBSCAN with K-Means, highlighting any areas where HDBSCAN's density-based approach might not have provided clear advantages in the dataset.



Silhouette Scores for Clustering Methods

Finally, we calculated silhouette scores, in order to evaluate the quality of the clustering created by K-Means, HDBSCAN, and Hierarchical Clustering algorithms. The silhouette score ranges from -1 to 1, with higher values indicating better-defined clusters. For K-Means clustering, we calculated silhouette scores and found a score of 0.8597, indicating well-defined clusters with good separation. The HDBSCAN clustering method had a silhouette score of -0.1846, suggesting some possible overlapping clusters or noise within the data. Hierarchical clustering showed us a silhouette score of -0.3428, implying a poor clustering performance. This may potentially be due to the hierarchical clustering method's sensitivity to data structure.



*Detailed silhouette score calculations are included in Appendix T.

Conclusion

The clustering analysis provided us with very valuable insights into the patterns and relationships between the transaction data. Our heatmap highlighted significant correlations, the dendrogram illustrated hierarchical relationships, and the scatter plots and silhouette scores evaluated the effectiveness of different clustering methods. The K-Means algorithm showed the best performance in terms of well-defined clusters, as indicated by the silhouette scores.

Ethical Recommendations

The ethics of accurately detecting and stopping a fraudulent transaction are fairly unambiguous. Outside of favorite edge courses of philosophy and ethics courses like stealing loaves of bread to feed the starving, theft is an innate harm. In addition to the material losses, fraudulent transactions erode trust, a necessary pillar of society. The key word here is “accurately.” The

inherent value of identifying a fraudulent transaction comes from the follow up action of denying service or material goods to the fraudster. However, acting on a false positive victimizes an innocent person who may be trying to acquire any number of necessary goods or services. Furthermore, utilizing engineered features introduces an opaque layer whose effects are difficult to measure.

To classify an attempted purchase as a criminal act or not based on the characteristics of the customer is a form of criminal profiling. Our model is asking "does this actor fit the profile of a criminal?" While being falsely accused by a company's Point of Sale in the context of an online transaction is less severe than in a legal setting, this lessened severity brings with it less scrutiny and oversight. Businesses should err on the side of caution so as not to unduly disrupt the lives of innocent customers. It is in their own interests to limit false positives as well, even if it creates more false negatives. When picking a threshold for the model a business might be tempted to minimize predicted losses even if it invalidates more legitimate transactions. But this ignores the potential alienation of customers, who might come to view a business as unreliable.

In addition to the problems of falsely accusing anyone, one of the most problematic uses of criminal profiling has been to target minorities on the basis of race. The justification for this was purported to be statistical evidence of a higher likelihood to commit a crime among certain demographics. Studies showed these demographics were being oversampled by store security and police, and the underlying statistics were systematic confirmation bias (*Dabney* 2006). The difference here is that our data is devoid of apparent demographic information such as race. Ironically, this makes it *more* difficult to determine if there's an inherent bias or skew in a model's determinations. The data provided was anonymized for privacy concerns, and pre-engineered features were also included. The lack of transparent meaning, further compounded by

methods like PCA, make it difficult to determine if the model is influenced by factors correlated with race. Full access to the original dataset doesn't guarantee that such bias could be detected, either. Great care should be taken to ensure models such as these don't disproportionately target certain segments of the population due to some oddity or non-causal correlation in the dataset.

Businesses must be careful not to over apply such methods. The reality is that shrink is a constant that all businesses must live with and account for. In our study the most effective models are capturing only ~60% of fraud cases, and mislabeling about 10% of the flagged transactions to achieve that hit rate. Businesses should not rely on an imperfect solution at the cost of their customers. If a model is going to be the sole input, they should restrict the threshold to limit false positives as much as possible.

Like other forms of criminal profiling, in isolation these methods are not sufficient for certainty. In summarizing the debate surrounding generalized criminal profiling, Michael Boylan notes that most critics find it unscientific, but useful when combined with other forms of evidence (2011). Businesses should follow the model of card issuers, who generally provide an avenue for card holders to immediately confirm a flagged transaction through an app or automated phone call.

Challenges

Working with datasets that have highly confidential information proved very challenging. Having 434 features with masked meanings possessed a challenge for interoperability. While information about the general meaning of feature sets was provided, we needed to guess each feature's exact meaning. In addition, engineered features while being valuable possessed the same problem of explainability to peers.

Second major problem was missing values and class imbalance in fraud cases. Not every feature has the data on each transaction which makes our combined data set packed with missing values. We handled missing values by removing features that had records with more than 60% of its data missing which puts less value on the feature, inputting rest using KNN and mode imputations. We also carefully selected XGBoost, known for its robustness to class imbalance, as our primary model and prioritized the ROC AUC score as our evaluation metric, as it is less sensitive to class imbalance than accuracy. The dataset exhibited a significant class imbalance, with fraudulent transactions representing only about 3.8% of the total records. Even Though class imbalance is an issue for machine learning models, the real world fraud data is always imbalanced due to its nature. We used tools that are suitable to use with imbalanced classes without losing performance.

The large dataset (over 540,000 records and 434 features) exceeded the computational capacity of our local machines. To overcome this, we randomly sampled 147,635 records (25% of the original data) for analysis. In addition, we removed 204 features just by removing missing values, mentioned above. While this allowed us to perform the necessary computations, it's important to acknowledge that sampling could have introduced bias and limited the representativeness of our findings. Future research could explore using cloud computing resources or more efficient algorithms to analyze the full dataset.

One significant challenge we faced during the exploration of RQ3 was the inability to visually identify the geographical location of the fraudulent transaction clusters. The transactions were encrypted by Vesta to ensure user privacy, making it impossible for us to find their latitudinal and longitudinal locations. We had planned on creating a geographical heat map that visually demonstrated where there was a high concentration of fraud occurring; however, we

were ultimately unable to provide this visualization. Instead, we utilized alternative visualization techniques, such as scatter plots, numbered heatmaps and dendrograms.

Recommendation and Next Steps

For further projects we can implement hybrid approaches to feature selection. This involves leveraging Feature Importance to pre-select the most influential features, followed by applying SFS for further refinement. To combat overfitting, we will incorporate early stopping during the SFS process. This involves monitoring the model's performance on a validation set at each step and halting the selection if no improvement is observed for a predefined number of iterations (patience). Additionally, to further enhance the model's generalization ability, we could leverage regularization techniques within the XGBoost algorithm. Specifically, we could explore the use of L1 (Lasso) and L2 (Ridge) regularization, which introduce penalties for large feature weights, thus encouraging a simpler and more robust model. Early stopping combined with regularization could further improve model performance and reduce the risk of overfitting that we faced.

To improve future projects, we recommend using more advanced clustering methods and visualization techniques, such as t-SNE or UMAP, in order to find any hidden patterns. In addition, incorporating time-series analysis could also help track fraud trends over time. Adding external data also may improve accuracy. Regularizing and tuning clustering algorithms can also improve overfitting. Finally, creating an automated pipeline for data processing and model evaluation may make workflows more efficient and adaptable to new fraud patterns.

References

Dataset website. (n.d.). Description for the features. Retrieved from https://www.kaggle.com/c/ieee-fraud-detection/discussion/101203#583227

Vesta Corporation. (n.d.). Competitors. Retrieved from https://www.g2.com/products/vesta-corporation-vesta/competitors/alternatives

Vesta Corporation. (n.d.). Corporate information. Retrieved from https://www.vesta.io/about-vesta

Vesta Corporation. (n.d.). Funding. Retrieved from https://www.vesta.io/news/vesta-secures-125-million-investment-from-goldfinch-partners

Vesta Corporation. (n.d.). Funding. Retrieved from https://www.zippia.com/vesta-careers-43571/demographics/

Vesta Corporation. (n.d.). Location. Retrieved from https://pitchbook.com/profiles/company/52849-45#faqs

ResearchGate. (n.d.). Feature Correlation Matrix. Retrieved from https://www.researchgate.net/figure/Correlation-matrix-for-feature-selection_fig2_368261486

Mode. (n.d.). Heat Map. Retrieved from https://mode.com/example-gallery/geographic-heat-map

Rodrigues, V. F., et al. (2022). Fraud Detection and Prevention in E-Commerce: A Systematic Literature Review. Electronic Commerce Research and Applications, 56, 101207. https://doi.org/10.1016/j.elerap.2022.101207

Ngai, E. W. T., et al. (2011). The Application of Data Mining Techniques in Financial Fraud Detection: A Classification Framework and an Academic Review of Literature. DECISION SUPPORT SYSTEMS, 50(3), 559–69. https://doi.org/10.1016/j.dss.2010.08.006

Mutemi, A., & Bacao, F. (2024). E-Commerce Fraud Detection Based on Machine Learning Techniques: Systematic Literature Review. Big Data Mining and Analytics, 7(2), 419–44. https://doi.org/10.26599/BDMA.2023.9020023

Fanai, H., & Abbasimehr, H. (2023). A Novel Combined Approach Based on Deep Autoencoder and Deep Classifiers for Credit Card Fraud Detection. Expert Systems with Applications, 217, 119562. https://doi.org/10.1016/j.eswa.2023.119562

Marchal, S., & Szyller, S. (2019). Detecting Organized eCommerce Fraud Using Scalable Categorical Clustering. ACM International Conference Proceeding Series, 215–28. https://doi.org/10.1145/3359789.3359810

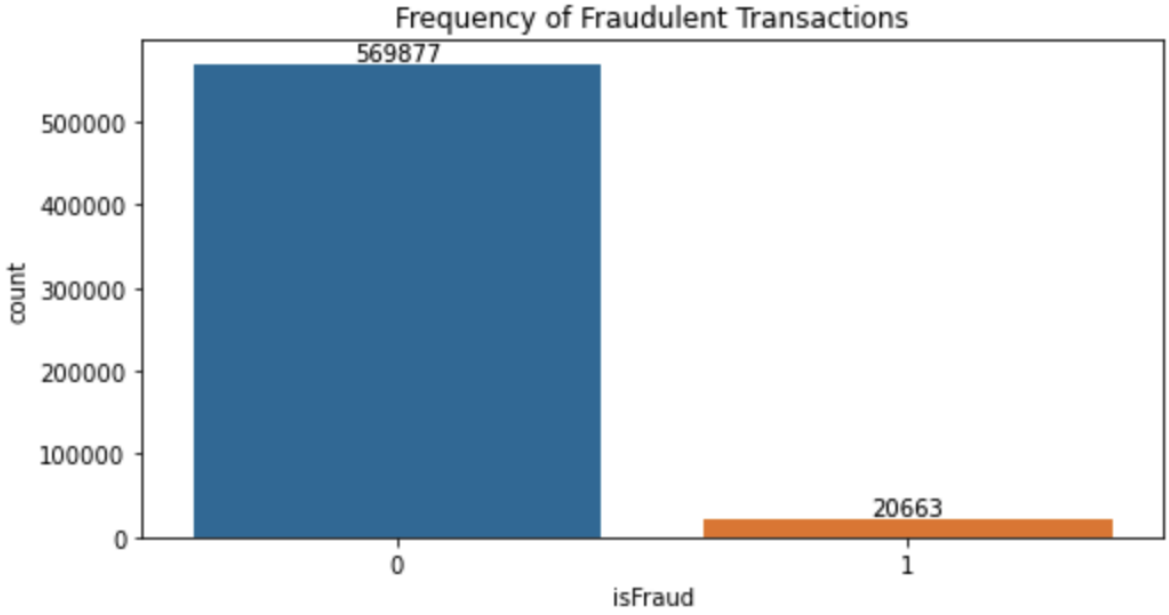
Tang, Y. (2023). Automatic Fraud Detection in E-Commerce Transactions Using Deep Reinforcement Learning and Artificial Neural Networks. International Journal of Advanced Computer Science & Applications, 14(7), 1047–58. https://doi.org/10.14569/IJACSA.2023.01407113

Boylan, M. (2011). Ethical Profiling. The Journal of Ethics, 15(1/2), 131–45. https://doi.org/10.1007/s10892-010-9092-9

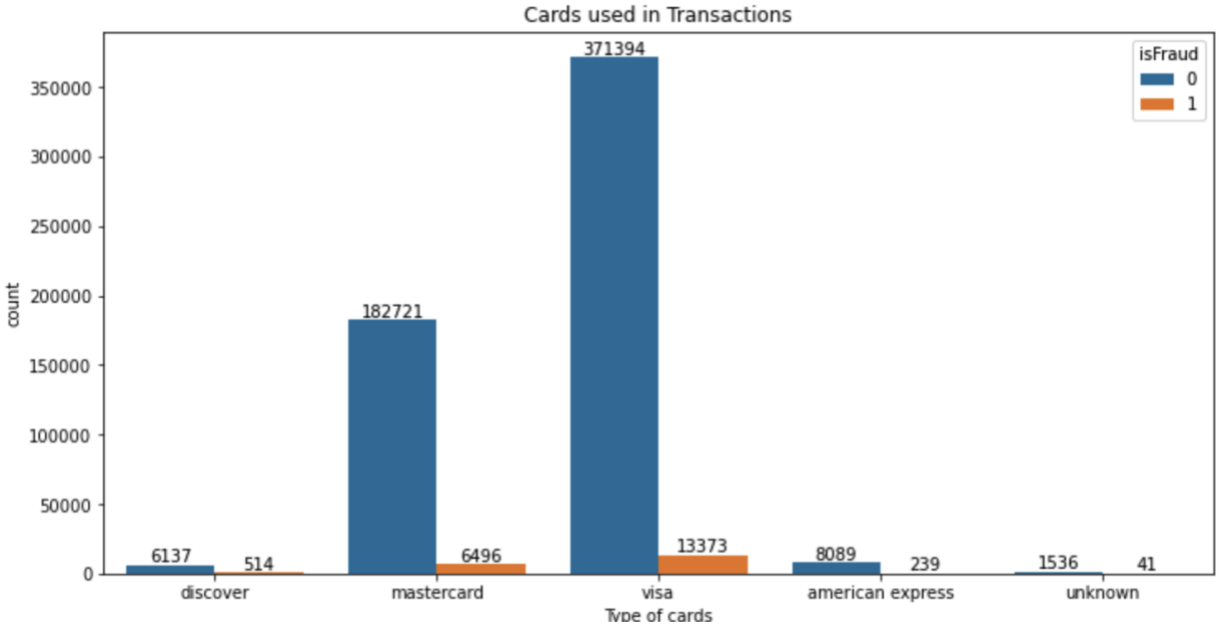
Dabney, D. A., et al. (2006). The Impact of Implicit Stereotyping on Offender Profiling: Unexpected Results From an Observational Study of Shoplifting. Criminal Justice and Behavior, 33(5), 646–74. https://doi.org/10.1177/0093854806288942

Appendix

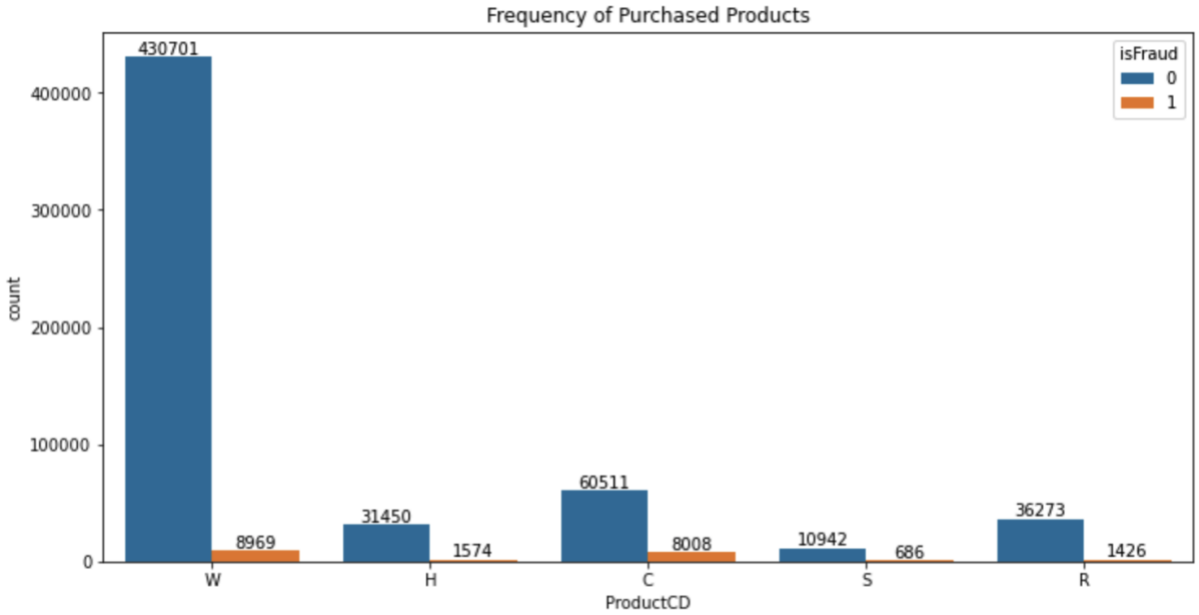
A. “Frequency of Fraudulent Transactions”



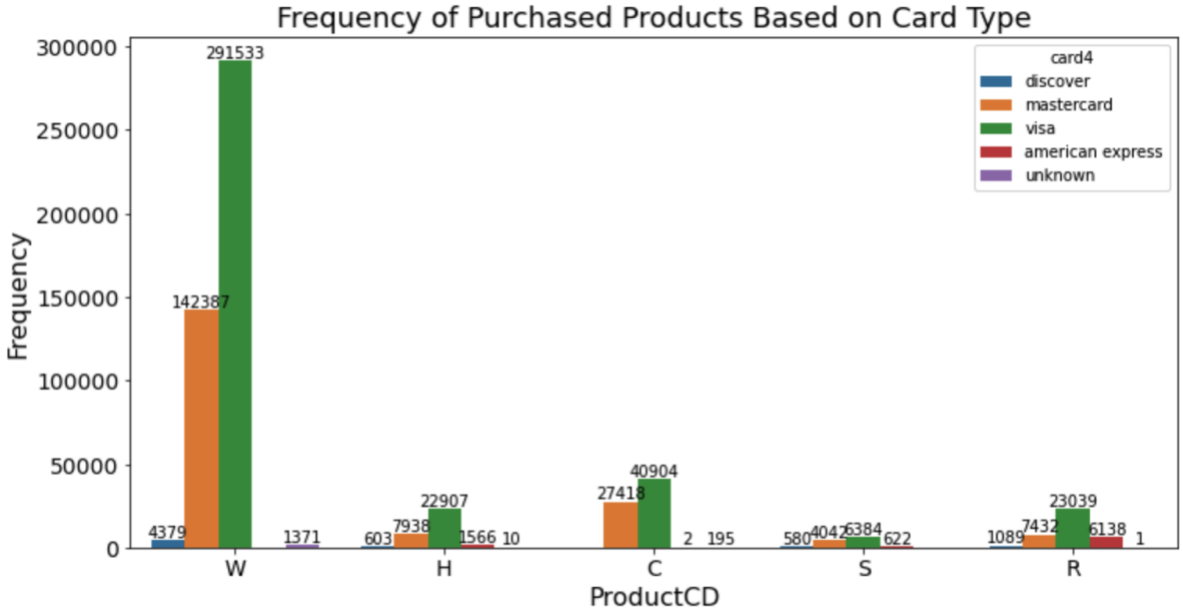
B. “Cards used in Transactions”



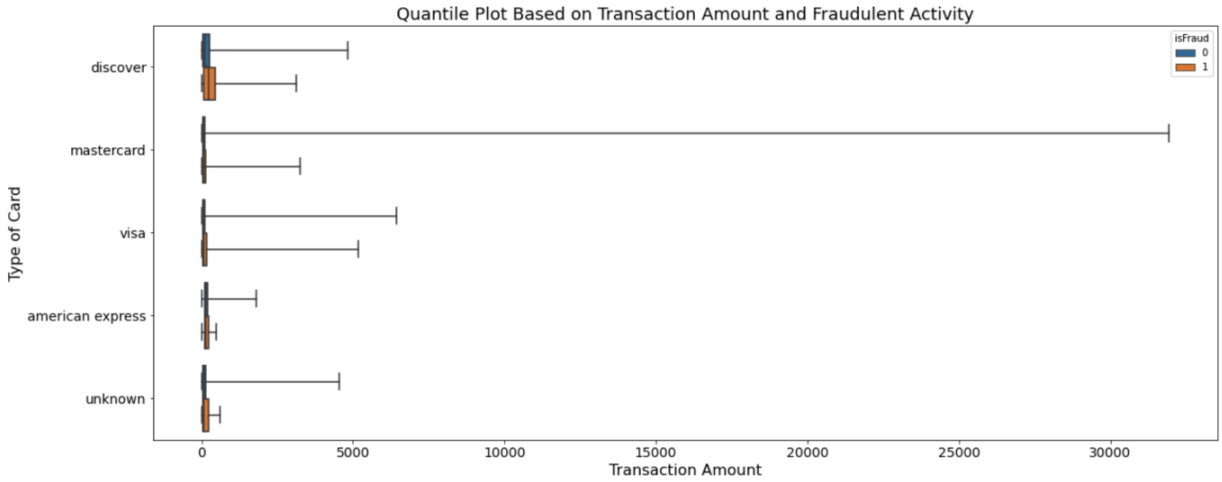
C. “Frequency of Purchased Products”



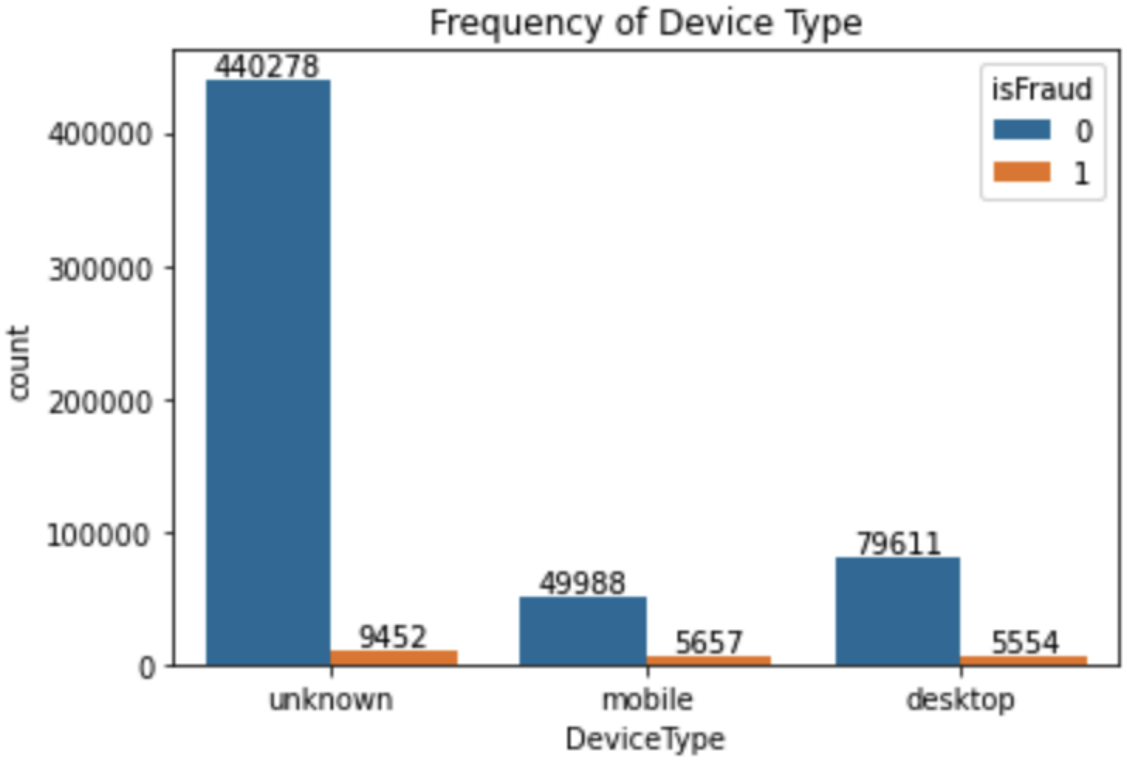
D. "Frequency of Purchased Products Based on Card Type"



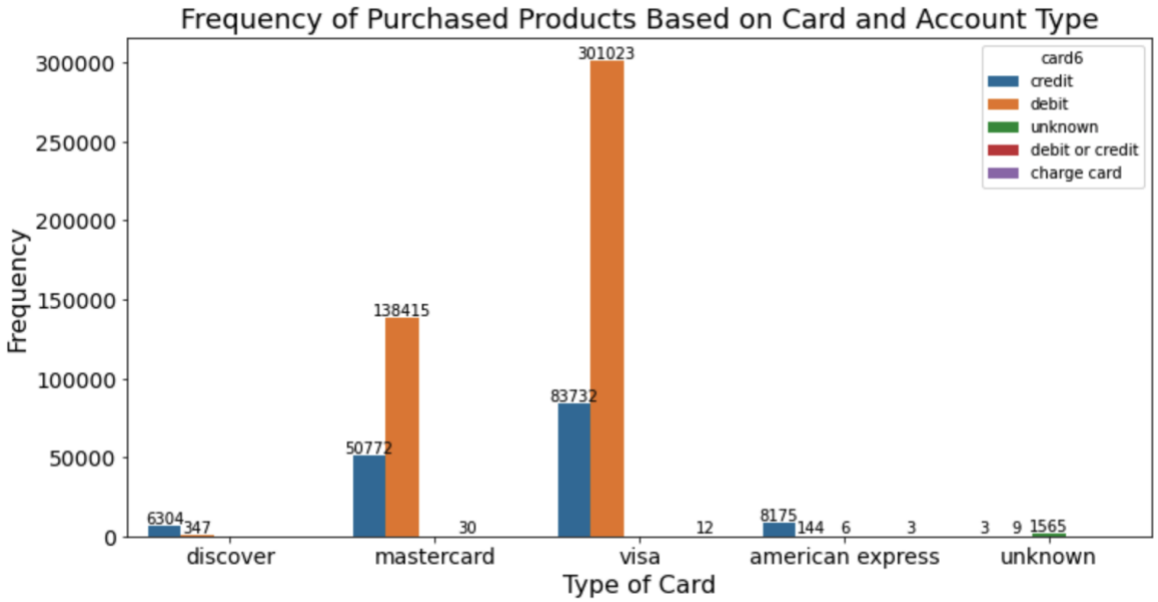
E. "Quantile Plot Based on Transaction Amount and Fraudulent Activity"



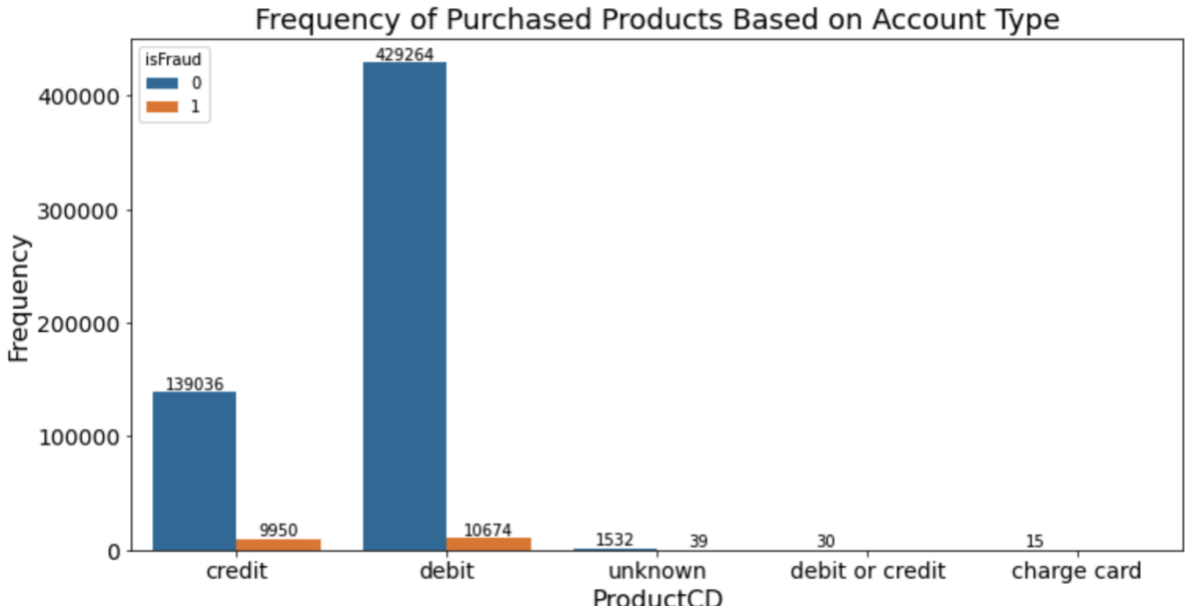
F. "Frequency of Device Type"



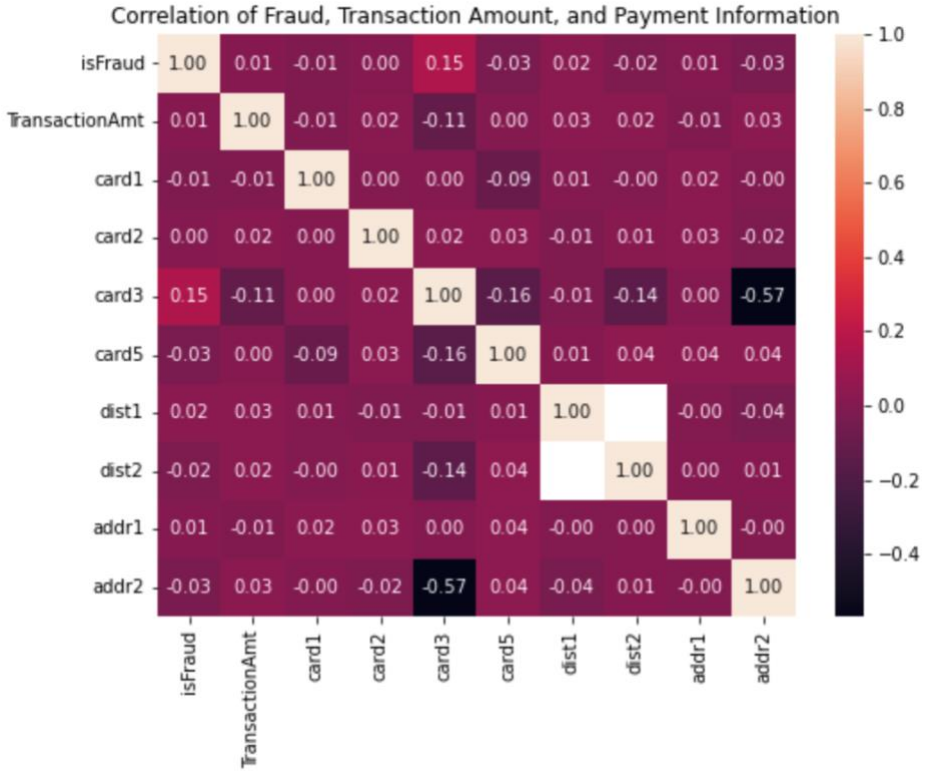
G. "Frequency of Purchased Products Based on Card and Account Type"



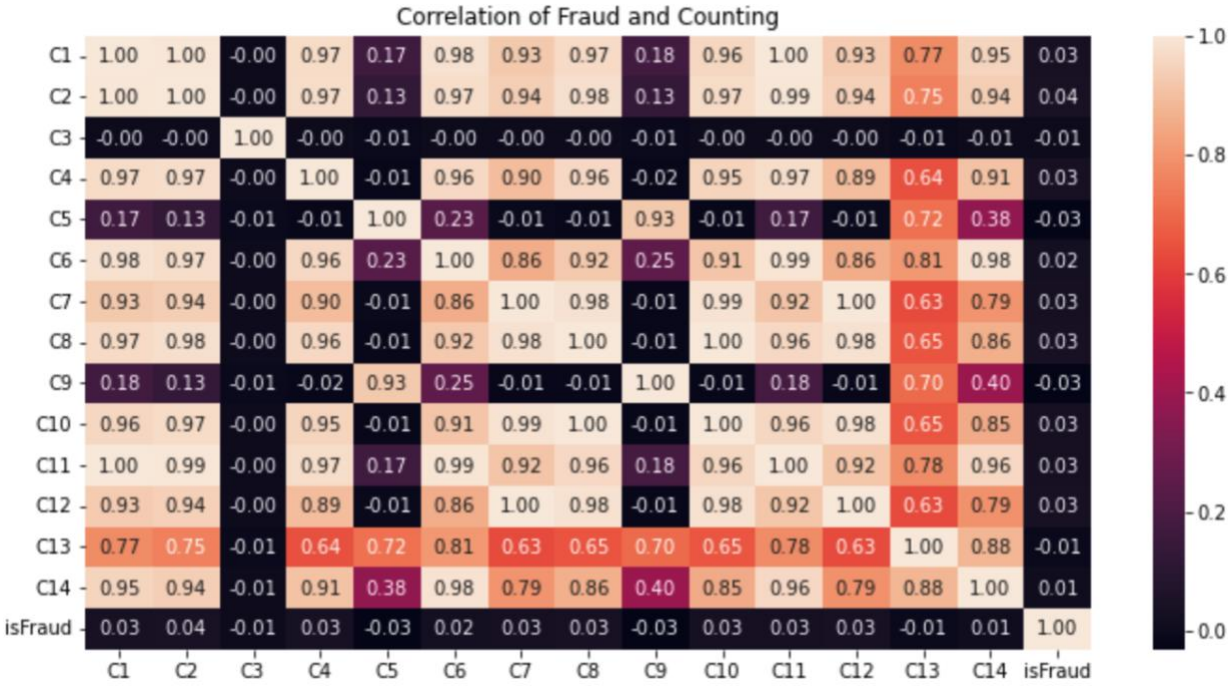
H. "Frequency of Purchased Products Based on Account Type"



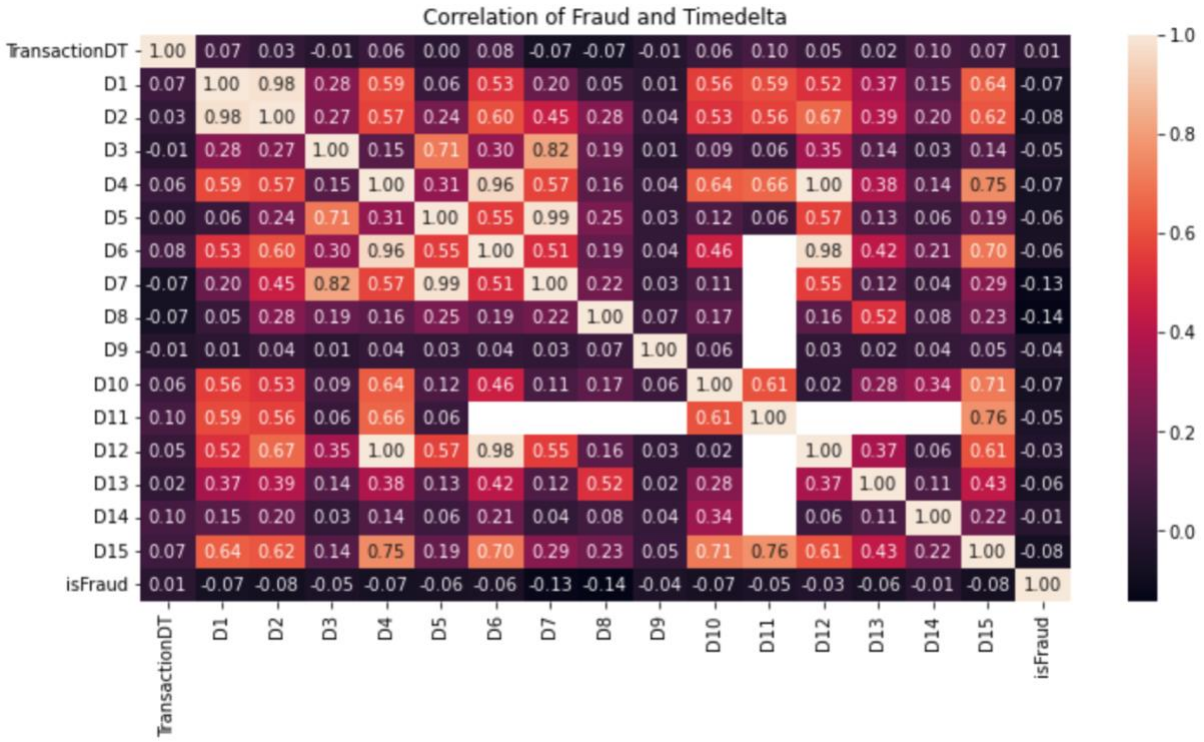
I. "Correlation of Fraud, Transaction Amount, and Payment Information"



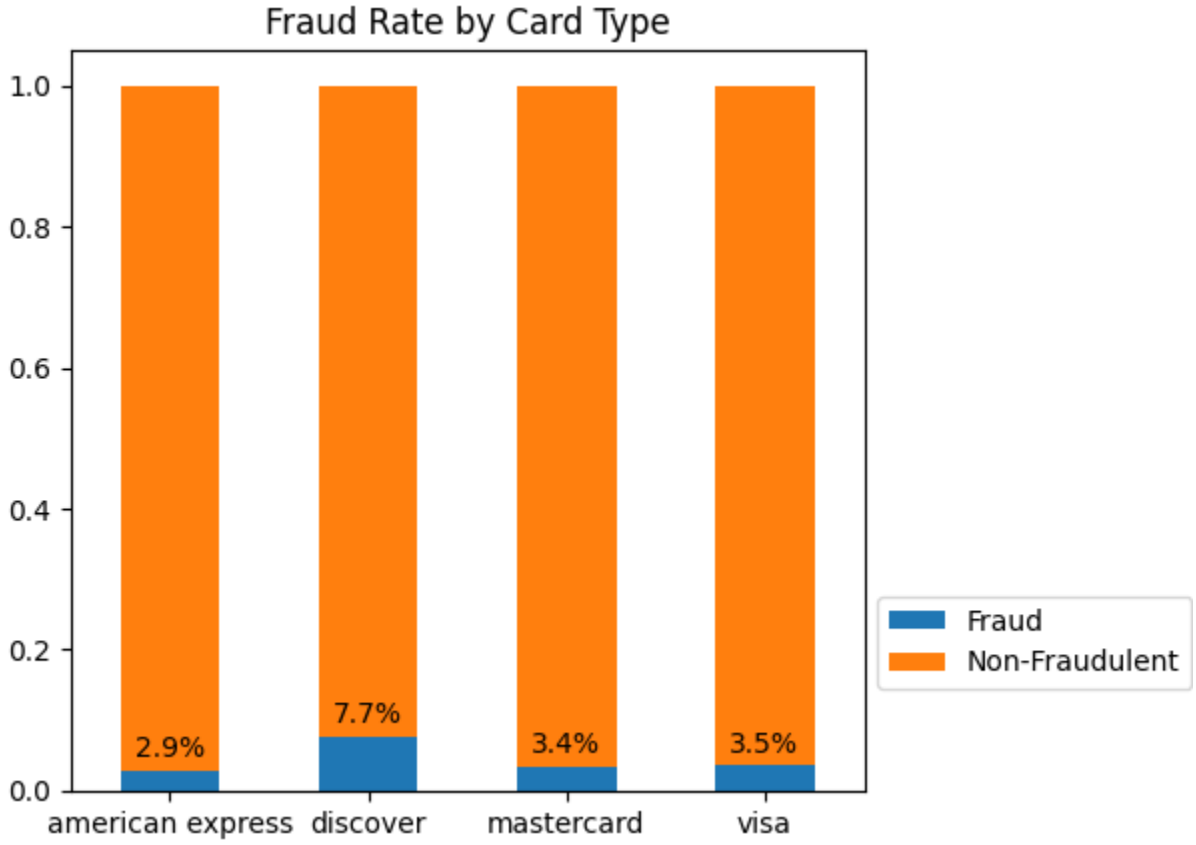
J. "Correlation of Fraud and Counting"



K. "Correlation of Fraud and Timedelta"

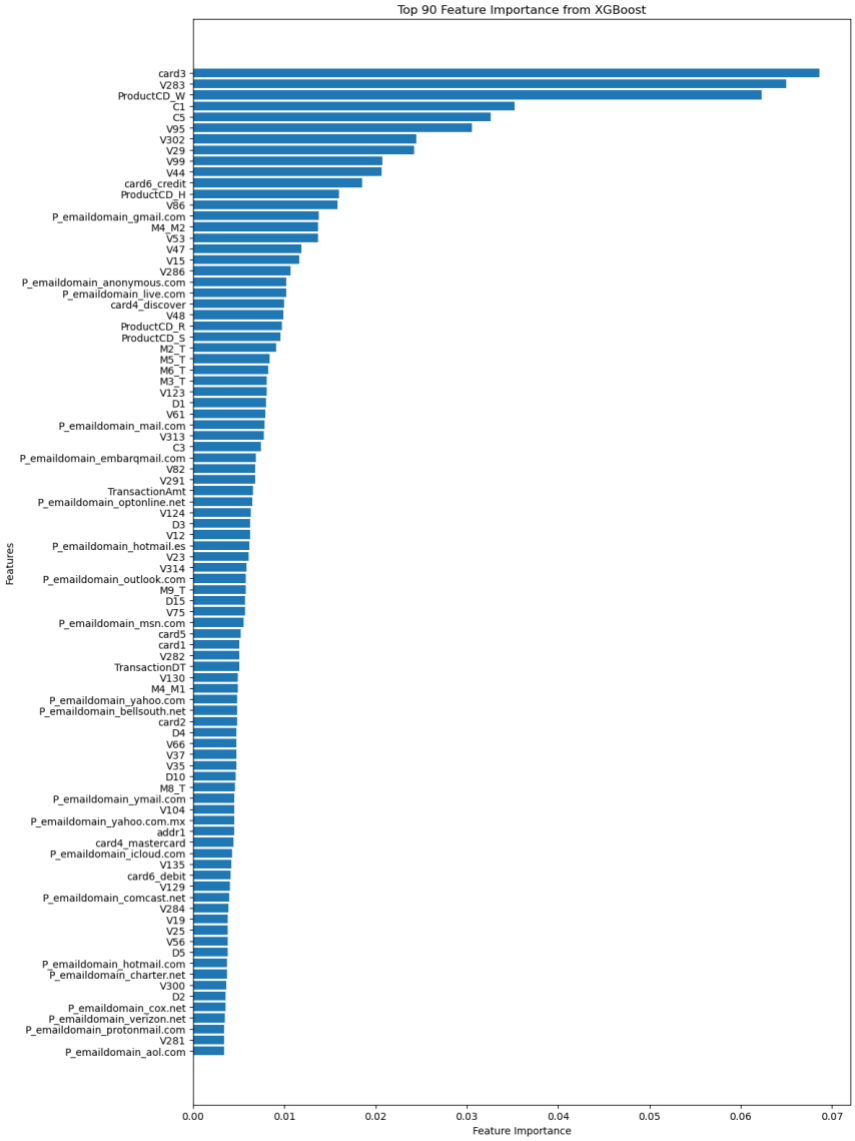


L. "Fraud Rate by Card Type"



M.

N. “Top 90 Feature Importance from XGBoost”



O.”Feature Group Table”

Feature Group	**Features**	**Explanation**
C	C1, C3, C5	Counting, such as how many addresses are found to be associated with the payment card.
Transaction	TransactionAmt, TransactionDT	TransactionAmt is the transaction payment amount in USD. TransactionDT is timedelta from a given reference datetime (not an actual timestamp).
M	M4_M1, M5_T, M6_T, M8_T	Match, such as names on card and address, etc.
card	card1, card2, card3, card5, card6_credit	Payment card information, such as card type, card category, issue bank, country, etc.
addr	addr1	Address (billing region).
V	V23, V29, V35, V53, V61, V66, V82, V124, V302, V314	Vesta engineered rich features, including ranking, counting, and other entity relations.
P_emaildomain	P_emaildomain_gmail.com, P_emaildomain_hotmail.com, P_emaildomain_yahoo.com	Purchaser email domain.
D	D1, D4, D10, D15	Timedelta, such as days between previous transactions, etc.
ProductCD	ProductCD_R, ProductCD_H, ProductCD_W	Product code, the product for each transaction.

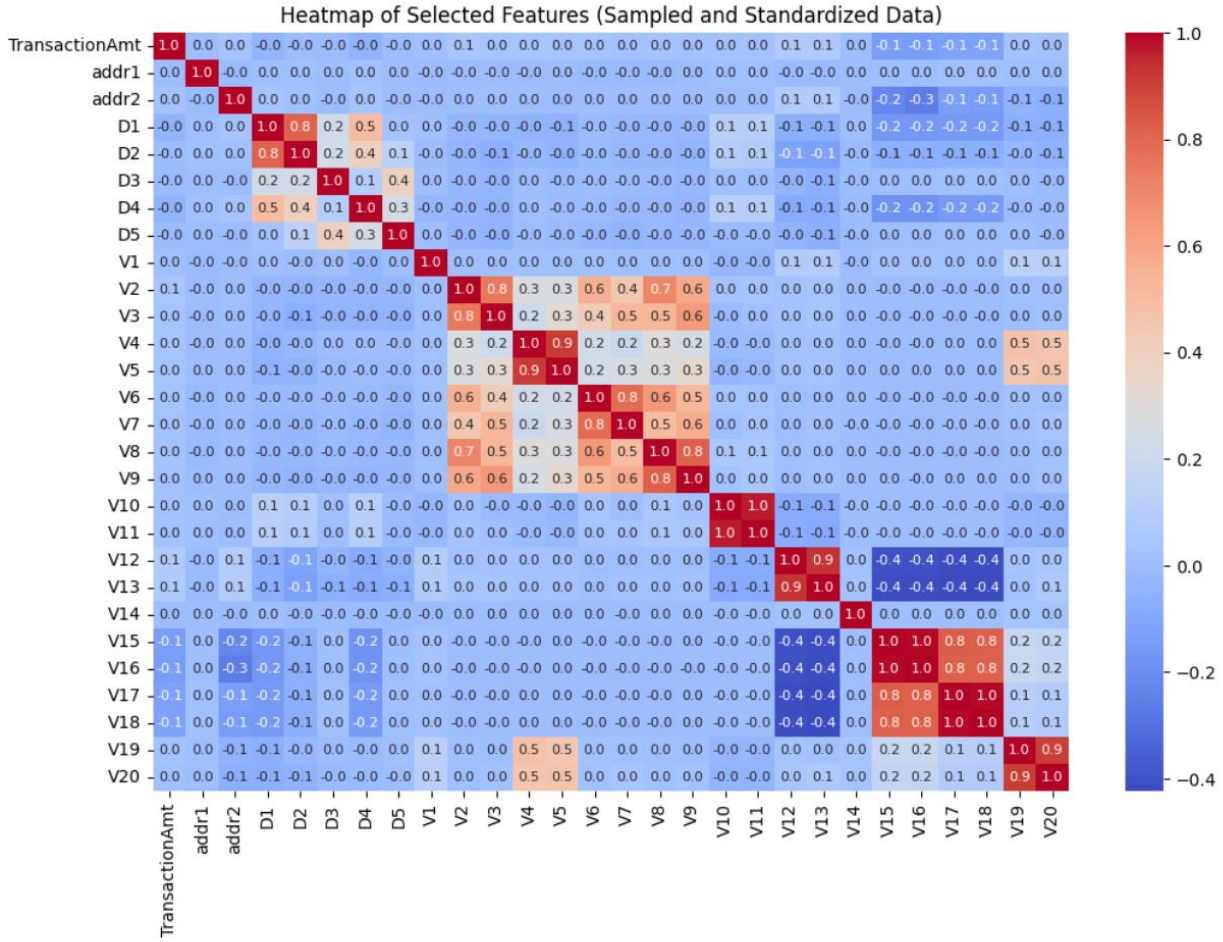
P.”Feature Overall Contribution”

	Feature	Overall Contribution
0	V305	0.999985
1	V1	0.999953
2	V65	0.999920
3	V14	0.999893
4	V68	0.999866
5	card2	0.999855
6	addr1	0.999851
7	C5	0.999846
8	V41	0.999807
9	V135	0.999794
10	V88	0.999703
11	card1	0.999553
12	D3	0.999424
13	V4	0.999359
14	V10	0.999283
15	V47	0.998991
16	C3	0.998979
17	addr2	0.998957
18	V27	0.998883
19	card5	0.998880
20	D5	0.998708

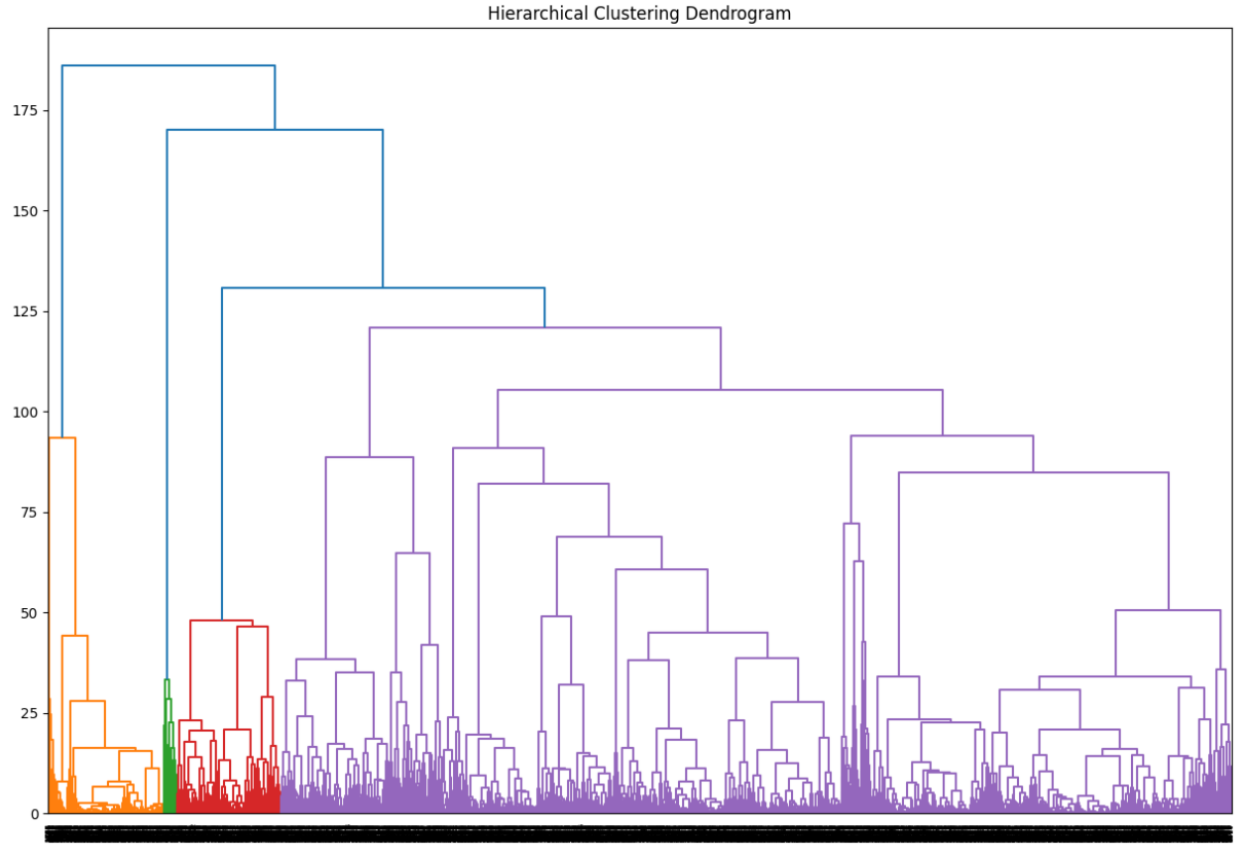
Q."Evaluation from Different numbers of Selected Features"

```
   Evaluation from different numbers of selected features:
-----------------------------------------------------------------
Num of selected features:10   Linear RMSE: 252.6991995375728
Num of selected features:20   Linear RMSE: 251.38688647012748
Num of selected features:30   Linear RMSE: 250.88800361060729
Num of selected features:40   Linear RMSE: 250.58234650509962
Num of selected features:50   Linear RMSE: 249.84196742944687
Num of selected features:60   Linear RMSE: 249.74482580947395
Num of selected features:70   Linear RMSE: 249.73613956808256
Num of selected features:80   Linear RMSE: 249.7231650525724
Num of selected features:90   Linear RMSE: 249.74698840697482
Num of selected features:100   Linear RMSE: 249.57694358602953
Num of selected features:110   Linear RMSE: 249.54592716356316
Num of selected features:120   Linear RMSE: 249.17860533189554
Num of selected features:130   Linear RMSE: 249.1296552285396
Num of selected features:140   Linear RMSE: 249.08457704718447
Num of selected features:150   Linear RMSE: 249.0780362959285
Num of selected features:160   Linear RMSE: 249.0654580893683
Num of selected features:170   Linear RMSE: 249.08026954491189
Num of selected features:180   Linear RMSE: 249.05980048531168
Num of selected features:190   Linear RMSE: 249.05973817366802
Num of selected features:200   Linear RMSE: 249.04163037463516
Num of selected features:210   Linear RMSE: 249.04342676809495
Num of selected features:220   Linear RMSE: 249.03348441548547
Num of selected features:230   Linear RMSE: 249.04622780762344
Num of selected features:240   Linear RMSE: 249.03841949670488
Num of selected features:250   Linear RMSE: 248.99352649093433
Num of selected features:260   Linear RMSE: 249.0017966892745
Num of selected features:270   Linear RMSE: 248.99463536842313
Num of selected features:280   Linear RMSE: 248.99911929321462
```

R. “Heatmap of Selected Features(Sampled and Standardized Data)”



S. “Hierarchical Clustering Dendrogram”



T.”Silhouette Scores”

```
K-Means Silhouette Score: 0.8564298897506002
HDBSCAN Silhouette Score: -0.18430016878234834
Hierarchical Silhouette Score: -0.0637329143989308
```

Code

EDA:

In [ ]:
```
import pandas as pd
```

In [ ]:
```
train_t = pd.read_csv('ieee-fraud-detection/train_transaction.csv')
test_t = pd.read_csv('ieee-fraud-detection/test_transaction.csv')
train_i = pd.read_csv('ieee-fraud-detection/train_identity.csv')
test_i = pd.read_csv('ieee-fraud-detection/test_identity.csv')
```

In [ ]:
```
merged = pd.merge(train_t,train_i[['TransactionID','DeviceType']],
                  on= 'TransactionID', how='left')
merged = merged.drop('TransactionID', axis=1)
```

In [ ]:
```
col_name = list(merged.columns)
numeric = list(merged._get_numeric_data().columns)
```

In [ ]:
```
for i in col_name:
    if i in numeric:
        merged[i] = merged[i].fillna(-999)
    else:
        merged[i] = merged[i].fillna('unknown')
```

In [ ]:
```
merged.head(20)
```

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [ ]:
```
plt.figure(figsize=(8, 4))
ax = sns.countplot(x = 'isFraud',data = merged)
ax.bar_label(ax.containers[0])
plt.title('Frequency of Fraudulent Transactions')
plt.show()
```

In [ ]:
```
plt.figure(figsize=(12, 6))
ax = sns.countplot(x ='card4', hue ='isFraud', data = merged)
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])
plt.title('Cards used in Transactions')
plt.xlabel('Type of cards')
plt.show()
```

In [ ]:
```
plt.figure(figsize=(12, 6))
ax = sns.countplot(x ='ProductCD', hue ='isFraud', data = merged)
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])
plt.title('Frequency of Purchased Products')
plt.show()
```

In [ ]:
```
plt.figure(figsize=(20, 8))
ax = sns.boxplot(data = merged, x='TransactionAmt', y='card4',hue='isFraud',
                 whis=(0, 100))
ax.tick_params(labelsize=14)
plt.title('Quantile Plot Based on Transaction Amount and Fraudulent Activity',
          fontsize = 18)
plt.ylabel('Type of Card', fontsize = 16)
plt.xlabel('Transaction Amount', fontsize = 16)
plt.show()
```

In [ ]:
```
merged['TransactionAmt'].describe()
```

In [ ]:
```
plt.figure(figsize=(12, 6))
ax = sns.countplot(x ='ProductCD', hue ='card4', data = merged)
ax.tick_params(labelsize=14)
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])
ax.bar_label(ax.containers[2])
ax.bar_label(ax.containers[3])
ax.bar_label(ax.containers[4])
plt.title('Frequency of Purchased Products Based on Card Type', fontsize = 18)
plt.ylabel('Frequency', fontsize = 16)
plt.xlabel('ProductCD', fontsize = 16)
plt.show()
```

In [ ]:
```
sns.boxplot(data = merged, x ='dist1', y ='card4',hue='isFraud', whis=(0, 100))
plt.title('Frequency of Device Type')
plt.show()
```

In [ ]:
```
plt.figure(figsize=(12, 6))
ax = sns.countplot(x ='ProductCD', hue ='isFraud', data = merged)
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])
plt.title('Frequency of Purchased Products')
plt.show()
```

In [ ]:
```
plt.figure(figsize=(8, 6))
ax = sns.countplot(x ='DeviceType', hue ='isFraud', data = merged)
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])
plt.title('Frequency of Device Type')
plt.show()
```

In [ ]:
```
plt.figure(figsize=(12, 6))
ax = sns.countplot(x ='card4', hue ='card6', data = merged)
ax.tick_params(labelsize=14)
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])
ax.bar_label(ax.containers[2])
ax.bar_label(ax.containers[3])
ax.bar_label(ax.containers[4])
plt.title('Frequency of Purchased Products Based on Card and Account Type',
          fontsize = 18)
plt.ylabel('Frequency', fontsize = 16)
plt.xlabel('Type of Card', fontsize = 16)
plt.show()
```

In [ ]:
```
plt.figure(figsize=(12, 6))
ax = sns.countplot(x ='card6', hue ='isFraud', data = merged)
ax.tick_params(labelsize=14)
ax.bar_label(ax.containers[0])
ax.bar_label(ax.containers[1])
plt.title('Frequency of Purchased Products Based on Account Type', fontsize = 18)
plt.ylabel('Frequency', fontsize = 16)
plt.xlabel('ProductCD', fontsize = 16)
plt.show()
```

In [ ]:
```
m = pd.merge(train_t,train_i[['TransactionID','DeviceType']],on= 'TransactionID',
             how='left')
v = [i for i in numeric if 'V' in i]
v.append('isFraud')
c = [i for i in numeric if 'C' in i]
c.append('isFraud')
d = [i for i in numeric if 'D' in i]
d.append('isFraud')
```

In [ ]:
```
plt.figure(figsize=(8, 6))
sns.heatmap(m[['isFraud', 'TransactionAmt', 'card1', 'card2',
               'card3', 'card5', 'dist1','dist2', 'addr1', 'addr2']].corr(),
            annot=True, fmt=".2f")
plt.title('Correlation of Fraud, Transaction Amount, and Payment Information')
plt.show()
```

In [ ]:
```
plt.figure(figsize=(12, 6))
sns.heatmap(m[c].corr(), annot=True, fmt=".2f")
plt.title('Correlation of Fraud and Counting')
plt.show()
```

In [ ]:
```
plt.figure(figsize=(12, 6))
sns.heatmap(m[d].corr(), annot=True, fmt=".2f")
plt.title('Correlation of Fraud and Timedelta')
plt.show()
```

Research Questions:

In [ ]:
```
import pandas as pd
import os

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
# FOR GOOGLE COLLAB Required Packages
# !pip install scikit-learn matplotlib seaborn xgboost mlxtend pandas numpy scipy,scikit-optimize,scikit-plot
```

In [ ]:
```
# FOR GOOGLE COLLAB

# from google.colab import drive
# drive.mount('/content/drive')
# file_path = '/content/drive/MyDrive/DAT490 BitWizards/Data/Original dataset/'
# file1 ='train_transaction.csv'
# file2='train_identity.csv'
# train_df_identity = pd.read_csv(os.path.join(file_path, file1))
# train_df_transaction = pd.read_csv(os.path.join(file_path, file2))
```

In [ ]:
```
train_df_identity = pd.read_csv('Data/train_identity.csv')
train_df_transaction = pd.read_csv('Data/train_transaction.csv')
```

In [ ]:
```
train_set = pd.merge(train_df_transaction, train_df_identity, on='TransactionID', how='left')
```

In [ ]:
```
train_set.head()
```

In [ ]:
```
threshold = 0.4 * len(train_set) # Data that have 60% or more missing values (40% of rows have a value)
train_set_clean = train_set.dropna(thresh=threshold, axis=1)
train_set_clean
```

In [ ]:
```
train_set_clean.isnull().sum()
```

In [ ]:
```
pd.set_option('display.max_rows',300)
print(train_set_clean.isnull().sum())
print(train_set_clean.dtypes)
pd.reset_option('display.max_rows')
```

In [ ]:
```
train_set_clean['isFraud'].value_counts()
```

In [ ]:
```
sample_size = int(0.25 * len(train_set_clean)) # I selected 25% of the data.

fraud_df = train_set_clean[train_set_clean['isFraud'] == 1]
non_fraud_df = train_set_clean[train_set_clean['isFraud'] == 0]

non_fraud_sample_size = sample_size - len(fraud_df) # How many samples we need to fulfill 25%

non_fraud_sample = non_fraud_df.sample(n=non_fraud_sample_size, random_state=42) # Select that amount

final_df = pd.concat([fraud_df, non_fraud_sample])
```

In [ ]:
```
print(final_df['isFraud'].value_counts())
print(len(final_df))
```

In [ ]:
```
pd.set_option('display.max_rows',300)
print(final_df.isnull().sum())
print(final_df.dtypes)
pd.reset_option('display.max_rows')
```

In [ ]:
```
# IMPUTING THE MISSING VALUES USING KNN IMPUTER
from sklearn.impute import KNNImputer
```

In [ ]:
```
numerical_cols = final_df.select_dtypes(include=['int64', 'float64']).columns
categorical_cols = final_df.select_dtypes(include=['object']).columns
```

In [ ]:
```
file = 'final_df_imputed.csv'
# file_name = os.path.join(file_path, 'final_df_imputed.csv')

if os.path.exists(file): # Change it to file_name if using Collab and comment out file
    final_df = pd.read_csv('final_df_imputed.csv')
else:
    # Apply KNN imputation for numerical features
    imputer = KNNImputer(n_neighbors=5)
    final_df[numerical_cols] = imputer.fit_transform(final_df[numerical_cols])

    # Apply mode imputation for categorical features
    for col in categorical_cols:
        final_df[col] = final_df[col].fillna(final_df[col].mode()[0])

    # Check if there are any remaining missing values
    print(final_df.isnull().sum().sum())

    final_df.to_csv('final_df_imputed.csv', index=False)
```

In [ ]:
```
final_df.describe()
```

In [ ]:
```
final_df['isFraud'].value_counts()
```

In [ ]:
```
X = final_df.drop(['isFraud','TransactionID'], axis=1)
y = final_df['isFraud']
```

In [ ]:
```
X_numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns
X_categorical_cols = final_df.select_dtypes(include=['object']).columns
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
scaler = StandardScaler()
X[X_numerical_cols] = scaler.fit_transform(X[X_numerical_cols])
```

In [ ]:
```
X = pd.get_dummies(X, columns=X_categorical_cols, drop_first=True)
```

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2, random_state=42)
```

RQ1 - How can advanced feature engineering and selection techniques, such as Recursive Feature Elimination (RFE), Feature Importance from Gradient Boosting, and Principal Component Analysis (PCA), enhance the accuracy of fraud detection models?

In [ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.dummy import DummyClassifier
from sklearn.pipeline import Pipeline
from xgboost import XGBClassifier
from sklearn.model_selection import cross_validate, KFold
from sklearn.model_selection import learning_curve
```

In [ ]:
```
def plot_learning_curve(model, X, y, cv):
    # Generate the training set sizes and the training and test scores
    train_sizes, train_scores, test_scores = learning_curve(model, X, y, cv=cv, n_jobs=-1)

    # Calculate the mean and standard deviation of the training and test scores
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    # Plot the learning curve
    plt.figure()
    plt.title("Learning Curve")
    plt.xlabel("Training examples")
    plt.ylabel("AUC Score")
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scores_mean + train_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")

    plt.legend(loc="best")
    plt.show()
```

In [ ]:
```
def evaluate_models(X_train, X_test, y_train, y_test):
    # Define models
    models = {
        'Base Model': DummyClassifier(strategy='stratified', random_state=42),
        "Decision Tree": DecisionTreeClassifier(random_state=0, criterion="entropy"),
        "XGBoost": XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='auc'),
        "Random Forest": RandomForestClassifier(n_estimators=10, random_state=0),
    }
```

```
    # Define scoring metrics
    scoring = ['accuracy', 'precision', 'recall', 'roc_auc']

    # Initialize cross-validation
    cv = KFold(n_splits=10, shuffle=True, random_state=0)

    # Initialize results storage
    results = {}

    # Evaluate each model
    for name, model in models.items():
        # Cross-validation results on training set
        cv_results = cross_validate(model, X_train, y_train, cv=cv, scoring=scoring)

        # Fit the model on the training data
        model.fit(X_train, y_train)

        # Evaluate on the test set
        test_accuracy = model.score(X_test, y_test)
        test_precision = precision_score(y_test, model.predict(X_test))
        test_recall = recall_score(y_test, model.predict(X_test))
        test_roc_auc = roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])

        results[name] = {
            'Train Accuracy': np.mean(cv_results['test_accuracy']),
            'Train Precision': np.mean(cv_results['test_precision']),
            'Train Recall': np.mean(cv_results['test_recall']),
            'Train ROC AUC': np.mean(cv_results['test_roc_auc']),
            'Test Accuracy': test_accuracy,
            'Test Precision': test_precision,
            'Test Recall': test_recall,
            'Test ROC AUC': test_roc_auc,
        }

    # Convert results to DataFrame for display
    results_df = pd.DataFrame(results).transpose()

    return results_df
```

```
[ ]: results_df = evaluate_models(X_train, X_test, y_train, y_test)
print(results_df)
```

In [ ]:
```
# Calculate the correlation matrix
corr_matrix = X[X_numerical_cols].corr()

# Plot the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=False, cmap='coolwarm')
plt.show()
```

In [ ]:
```
# Find correlations greater than 0.8 or less than -0.8
highly_correlated_features = []
for i in range(len(corr_matrix.columns)):
    for j in range(i):
        if abs(corr_matrix.iloc[i, j]) > 0.8:
            # colname_i = corr_matrix.columns[i]
            # colname_j = corr_matrix.columns[j]
            # score = corr_matrix.iloc[i, j]
            # print(f"{colname_i} - {colname_j} : score = {score}")
            colname = corr_matrix.columns[i]
            highly_correlated_features.append(colname)

# Print highly correlated features
unique_cols = []
unique_cols = list(set(highly_correlated_features))
print(unique_cols)
print(len(unique_cols))
```

In [ ]:
```
X_droped = X.drop(columns=unique_cols)
print(X_droped.shape)
display(X_droped.head())
display(X_droped.describe())
```

In [ ]:
```
X_train_corr, X_test_corr, y_train_corr, y_test_corr = train_test_split(X_droped, y,test_size=0.2, random_state=42)
```

In [ ]:
```
results_df_corr = evaluate_models(X_train_corr, X_test_corr, y_train_corr, y_test_corr)
print(results_df_corr)
```

In [ ]:
```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.model_selection import cross_validate, StratifiedKFold
import pickle
from sklearn.metrics import roc_curve, auc
```

In [ ]:
```
def evaluate_model(model, X, y, cv=10):
    # Use StratifiedKFold to ensure each fold has the same proportion of classes
    skf = StratifiedKFold(n_splits=cv)
    cv_results = cross_validate(model, X, y, cv=skf, scoring=['accuracy', 'precision', 'recall', 'roc_auc'],
                                return_train_score=True, return_estimator=True)

    results = pd.DataFrame({
        'Metric': ['Accuracy', 'Precision', 'Recall', 'ROC AUC'],
        'CV Mean': [np.mean(cv_results['test_accuracy']), np.mean(cv_results['test_precision']),
                    np.mean(cv_results['test_recall']), np.mean(cv_results['test_roc_auc'])],
        'Train Mean': [np.mean(cv_results['train_accuracy']), np.mean(cv_results['train_precision']),
                       np.mean(cv_results['train_recall']), np.mean(cv_results['train_roc_auc'])]
    })
```

```
    results = results.round(3)
    return results, cv_results['estimator'], skf
```

In [ ]:
```
def plot_roc_curves(estimators, skf, X, y):
    X = np.array(X)
    y = np.array(y)

    plt.figure(figsize=(8, 6))
    for i, (train_idx, test_idx) in enumerate(skf.split(X, y)):
        estimator = estimators[i]
        X_test, y_test = X[test_idx], y[test_idx]
        probs = estimator.predict_proba(X_test)[:, 1]
        fpr, tpr, _ = roc_curve(y_test, probs)
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f'Fold {i+1} (AUC = {roc_auc:.2f})')

    plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve for Each Fold')
    plt.legend(loc='lower right')
    plt.grid(True)
    plt.show()
```

In [ ]:
```
# Define the pickle file path
pickle_file_path = 'sfs_model.pkl'

# Check if the pickle file exists
if os.path.exists(pickle_file_path):
    # Load the sfs object from the pickle file
    with open(pickle_file_path, 'rb') as f:
        sfs = pickle.load(f)
    print("Loaded SFS from pickle file.")
else:
    # Initialize the XGBoost model
    xgb_model = XGBClassifier(random_state=0, use_label_encoder=False, eval_metric='auc')

    # Initialize SFS with the XGBoost model
    sfs = SFS(estimator=xgb_model,
              k_features='best',
              forward=False,  # Perform RFE
              floating=False,
              scoring='roc_auc',
              n_jobs=-1,
              cv=5)

    # Fit SFS
    sfs = sfs.fit(X_train_corr, y_train_corr)

    # Save the sfs object to a pickle file
    with open(pickle_file_path, 'wb') as f:
        pickle.dump(sfs, f)
    print("SFS object created and saved to pickle file.")
```

In [ ]:
```
selected_features_sfs = list(sfs.k_feature_names_)
print(len(selected_features_sfs))
# Create datasets with selected features
X_train_sfs_selected = X_train_corr[selected_features_sfs]
X_test_sfs_selected = X_test_corr[selected_features_sfs]

X_sfs_selected = X_droped[selected_features_sfs]

# Initialize a new XGBoost model for sfs selected feature
xgb_model_sfs_selected = XGBClassifier(random_state=0, use_label_encoder=False, eval_metric='auc')
```

In [ ]:
```
plot_learning_curve(xgb_model_sfs_selected, X_train_sfs_selected, y_train_corr, cv=10)
```

In [ ]:
```
# Evaluate the model
sfs_results, sfs_estimators, sfs_skf = evaluate_model(xgb_model_sfs_selected, X_sfs_selected, y, cv=10)
print(sfs_results)
```

In [ ]:
```
plot_roc_curves(sfs_estimators, sfs_skf, X_sfs_selected, y)
```

Feature Importance

In [ ]:
```
# Train an XGBoost model for feature importance
xgb_model = XGBClassifier(random_state=0, use_label_encoder=False, eval_metric='auc')
xgb_model.fit(X_train_corr, y_train_corr)

# Get feature importance scores
feature_importances = xgb_model.feature_importances_
```

In [ ]:
```
indices = np.argsort(feature_importances)[::-1][:90][::-1] # Get the indices of the top 90 features

plt.figure(figsize=(12, 16))  # Adjust the figure size for better readability
plt.barh(range(90), feature_importances[indices], align='center')
plt.yticks(range(90), X_train_corr.columns[indices])
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Top 90 Feature Importance from XGBoost')
plt.tight_layout()
plt.show()
```

In [ ]:
```
#  Convert to a DataFrame for easy manipulation
feature_importances_df = pd.DataFrame({
    'Feature': X_train_corr.columns,
    'Importance': feature_importances
})

# Sort features by importance
feature_importances_df = feature_importances_df.sort_values(by='Importance', ascending=False)

# Calculate cumulative importance
feature_importances_df['Cumulative Importance'] = feature_importances_df['Importance'].cumsum()
```

```
# Set a cumulative importance threshold (e.g., 90%)
cumulative_threshold = 0.90
selected_features = feature_importances_df[feature_importances_df['Cumulative Importance'] <= cumulative_threshold]['Feature'].tolist()

# Create datasets with selected features
X_train_selected = X_train_corr[selected_features]
X_test_selected = X_test_corr[selected_features]

X_fi_selected = X_droped[selected_features]

print(f"Selected features names: {selected_features}")
print(f"Number of selected features: {len(selected_features)}")

# Initialize a new XGBoost model for evaluation
xgb_model_selected = XGBClassifier(random_state=0, use_label_encoder=False, eval_metric='auc')
```

In [ ]:
```
plot_learning_curve(xgb_model_selected, X_train_selected, y_train_corr, cv=10)
```

In [ ]:
```
fi_results, fi_estimator, fi_skf = evaluate_model(xgb_model_selected, X_fi_selected, y, cv=10)
print(sfs_results)
```

In [ ]:
```
plot_roc_curves(fi_estimator, fi_skf, X_fi_selected, y)
```

In [ ]:
```
sfs_set = set(selected_features_sfs)
fi_set = set(selected_features)

# Identify features exclusively selected by each method and overlapping features
sfs_only = sfs_set - fi_set
fi_only = fi_set - sfs_set
both = sfs_set & fi_set

# Create a DataFrame for the counts
df_counts = pd.DataFrame({
    'Method': ['SFS Only', 'FI Only', 'Both'],
    'Count': [len(sfs_only), len(fi_only), len(both)]
})

# Bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x='Method', y='Count', data=df_counts, palette='viridis')
plt.title('Count of Selected Features by SFS and FI')
plt.xlabel('Selection Method')
plt.ylabel('Count of Features')
plt.show()

# Print the features for verification
print(f"Features selected only by SFS: {list(sfs_only)}")
print(f"Features selected only by FI: {list(fi_only)}")
print(f"Features selected by both SFS and FI: {list(both)}")
```

In [ ]:
```
from matplotlib_venn import venn2
```

In [ ]:
```
plt.figure(figsize=(18, 18))
venn = venn2([sfs_set, fi_set], set_labels = ('SFS', 'FI'))

for text in venn.set_labels:
    text.set_fontsize(20)

# Move the labels to the top
venn.get_label_by_id('A').set_position((-0.4, 0.5))
venn.get_label_by_id('B').set_position((0.5, 0.5))


# Customize the plot
venn.get_label_by_id('10').set_text('\n'.join(sfs_set - fi_set))
venn.get_label_by_id('01').set_text('\n'.join(fi_set - sfs_set))
venn.get_label_by_id('11').set_text('\n'.join(sfs_set & fi_set))

venn.get_label_by_id('10').set_fontsize(10)
venn.get_label_by_id('01').set_fontsize(10)
venn.get_label_by_id('11').set_fontsize(10)

venn.get_patch_by_id('10').set_color('xkcd:light teal')
venn.get_patch_by_id('01').set_color('xkcd:dark orange')
venn.get_patch_by_id('11').set_color('xkcd:plum')

plt.title('Selected Features based on tehcnique used', fontsize=16)
plt.show()
```

PCA

In [ ]:
```
from sklearn.decomposition import PCA
```

In [ ]:
```
#Create a PCA object
pca = PCA()

# Fit PCA on your training data
pca.fit(X_droped)

# Get explained variance ratios
explained_variance_ratio = pca.explained_variance_ratio_

# Calculate cumulative explained variance
cumulative_explained_variance = np.cumsum(explained_variance_ratio)
```

In [ ]:
```
# Plot cumulative variance over principalcomponents
plt.figure(figsize=(10, 7))
plt.plot(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio, 'b-', label='Individual Explained Variance')
plt.plot(range(1, len(cumulative_explained_variance) + 1), cumulative_explained_variance, 'r-', label='Cumulative Explained Variance')
plt.xlabel('Principal Components')
plt.ylabel('Proportion of Variance Explained')
plt.legend(loc='best')
plt.show()
```

In [ ]:
```python
# Define the number of components you want to keep
n_components = 0.98

# Initialize the PCA model
pca = PCA(n_components=n_components)

# Fit and transform the data on the training set
X_pca = pca.fit_transform(X_droped)


print("Number of components chosen by PCA: ", pca.n_components_)
```

In [ ]:
```python
# Initialize XGBoost classifier for PCA
xgb_model_pca = XGBClassifier(random_state=0, use_label_encoder=False, eval_metric='auc')

# Evaluate the model
pca_results, pca_estimator, pca_skf = evaluate_model(xgb_model_pca, X_pca, y, cv=10)
print(pca_results)
```

In [ ]:
```python
loadings = pd.DataFrame(pca.components_.T, columns=[f'PC{i+1}' for i in range(pca.n_components_)], index=X_droped.columns)

print(loadings)
```

In [ ]:
```python
# Calculate squared loadings
squared_loadings = loadings ** 2

# Sum the squared loadings for each feature
overall_contribution = squared_loadings.sum(axis=1)

# Sort by overall contribution
overall_contribution_sorted = overall_contribution.sort_values(ascending=False)

# Create a DataFrame with feature names and their overall contribution
importance_df = pd.DataFrame({
    'Feature': overall_contribution_sorted.index,
    'Overall Contribution': overall_contribution_sorted.values
})

print(importance_df.head(21))
```

In [ ]:
```python
plot_roc_curves(pca_estimator, pca_skf, X_pca, y)
```

In [ ]:
```python
fi_results['Model'] = 'Feature Importance'
sfs_results['Model'] = 'Sequential Feature Selection'
pca_results['Model'] = 'PCA'

# Concatenate all DataFrames into one
all_results_df = pd.concat([fi_results, sfs_results, pca_results])

# Pivot the DataFrame
all_results_pivot = all_results_df.pivot(index='Model', columns='Metric', values='CV Mean')

print(all_results_pivot)
```

RQ 2 - How can machine learning models predict the distance where the transaction was made based on various factors, including transaction amount (TransactionAmt), card information (card1 - card6), address (addr1, addr2), time between transaction(D1-D15) and security features (Vxxx)?

```python
import pandas as pd
import os

import numpy as np
import matplotlib.pyplot as plt
```

```python
df = pd.read_csv('final_df_imputed.csv')
df.head(5)
```

```python
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
categorical_cols = df.select_dtypes(include=['object']).columns
```

```python
file = 'final_df_imputed.csv'
# file_name = os.path.join(file_path, 'final_df_imputed.csv')

if os.path.exists(file): # Change it to file_name if using Collab and comment out file
    final_df = pd.read_csv('final_df_imputed.csv')
else:
    # Apply KNN imputation for numerical features
    imputer = KNNImputer(n_neighbors=5)
    final_df[numerical_cols] = imputer.fit_transform(final_df[numerical_cols])

    # Apply mode imputation for categorical features
    for col in categorical_cols:
        final_df[col] = final_df[col].fillna(final_df[col].mode()[0])

    # Check if there are any remaining missing values
    print(final_df.isnull().sum().sum())

    final_df.to_csv('final_df_imputed.csv', index=False)
```

```python
final_df.describe()
```

```
[ ]  final_df = final_df.drop('TransactionID', axis=1)
     fraud = final_df[final_df["isFraud"] == 1]
     notfraud = final_df[final_df["isFraud"] == 0]
```

```
     fraud['dist1'].shape[0]/final_df.shape[0]
```

```
[ ]  notfraud['dist1'].shape[0]/final_df.shape[0]
```

```
[ ]  threshold = fraud['dist1'].mean() + fraud['dist1'].std()
```

```
[ ]  X = final_df.drop(['dist1'], axis=1)
     y = final_df['dist1']
     X_numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns.drop('isFraud')
     X_categorical_cols = final_df.select_dtypes(include=['object']).columns
     from sklearn.preprocessing import StandardScaler
     X = pd.get_dummies(X, columns=X_categorical_cols, drop_first=True)

     scaler = StandardScaler()
     X[X_numerical_cols] = scaler.fit_transform(X[X_numerical_cols])

     from sklearn.model_selection import train_test_split

     X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2, random_state=42)
```

```
[ ]  from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_squared_error
     import numpy as np

     lr_model = LinearRegression()
     lr_model.fit(X_train, y_train)

     y_pred = lr_model.predict(X_test)
     rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred))
     print(f'Linear Regression RMSE: {rmse_lr}')
```

```python
from sklearn.feature_selection import RFE

def rfe_evaluation(n, X_test, X_train, y_train):

    model = LinearRegression()

    # Initialize RFE with the model and number of features to select
    rfe = RFE(model, n_features_to_select=n)

    # Fit RFE
    fit = rfe.fit(X_train, y_train)

    # Get the selected features
    selected_features = X_train.columns[fit.support_]
    model.fit(X_train[selected_features], y_train)
    y_pred = model.predict(X_test[selected_features])
    rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred))
    return [fit.n_features_, selected_features, rmse_lr]
```

```python
rfe_selected = []
for i in range(10,X.shape[1], 10):
   rfe_selected.append(rfe_evaluation(i, X_test, X_train, y_train))
```

```python
x_rfe = []
y_rfe = []
print("   Evaluation from different numbers of selected features:")
print("-----------------------------------------------------------------")
for i in range(len(rfe_selected)):
    x_rfe.append(rfe_selected[i][0])
    y_rfe.append(rfe_selected[i][2])
    print(f"Num of selected features:{rfe_selected[i][0]}   Linear RMSE: {rfe_selected[i][2]}")
```

```python
plt.scatter(x_rfe, y_rfe)
plt.xlabel("Number of Features Selected")
plt.ylabel("RSME")
plt.title("Number of Features Selected vs RSME")
plt.show()
```

```python
import xgboost as xgb
model_xgb = xgb.XGBRegressor()
model_xgb.fit(X_train, y_train)
y_pred_xgb = model_xgb.predict(X_test)
rmse_xgb = np.sqrt(mean_squared_error(y_test, y_pred_xgb0))
print(f'XGBoost RMSE: {rmse_xgb}')
```

```python
import xgboost as xgb
from bayes_opt import BayesianOptimization
from sklearn.model_selection import cross_val_score
def xgb_evaluate(max_depth, learning_rate, n_estimators, subsample, colsample_bytree):
    params = {
        'max_depth': int(max_depth),
        'learning_rate': learning_rate,
        'n_estimators': int(n_estimators),
        'subsample': subsample,
        'colsample_bytree': colsample_bytree,
        'objective': 'reg:squarederror'
    }
    model = xgb.XGBRegressor(**params)
    cv_result = cross_val_score(model, X_train, y_train, cv=3, scoring='neg_mean_squared_error')
    return np.sqrt(-cv_result.mean())

# Define the parameter space
param_bounds = {
    'max_depth': (3, 10),
    'learning_rate': (0.01, 0.3),
    'n_estimators': (100, 1000),
    'subsample': (0.6, 1.0),
    'colsample_bytree': (0.6, 1.0)
}

# Initialize Bayesian Optimization
optimizer = BayesianOptimization(f=xgb_evaluate, pbounds=param_bounds, random_state=42)

# Run the optimizer
optimizer.maximize(init_points=10, n_iter=10)

# Best parameters
print("Best Parameters: ", optimizer.max)
best_params = optimizer.max['params']
best_params['max_depth'] = int(best_params['max_depth'])
best_params['n_estimators'] = int(best_params['n_estimators'])

# Train the final model
final_model = xgb.XGBRegressor(**best_params)
final_model.fit(X_train, y_train)

# Evaluate the final model
from sklearn.metrics import mean_squared_error

y_pred_xgbbo = final_model.predict(X_test)
rmse_xgbbo = np.sqrt(mean_squared_error(y_test, y_pred_xgbbo))
print(f'XGBoost(BayesianOptimization) RMSE: {rmse_xgbbo}')
```

```python
from sklearn.model_selection import RandomizedSearchCV

# Train XGBoost
xgb_model = xgb.XGBRegressor(objective='reg:squarederror')

# Define hyperparameter grid
param_dist = {
    'n_estimators': [1000, 2000, 3000],
    'learning_rate': [0.01, 0.1, 0.3],
    'max_depth': [3, 5, 7],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 0.9]
}

# Fitting training values to RandomizedSearchCV
random_search = RandomizedSearchCV(xgb_model, param_distributions=param_dist, n_iter=10, scoring='neg_mean_squared_error', cv=3, verbose=1, random_state=42)
random_search.fit(X_train, y_train)

# Best model
best_xgb_model = random_search.best_estimator_

# Predict and evaluate
y_pred_xgbr = best_xgb_model.predict(X_test)
rmse_xgbr = np.sqrt(mean_squared_error(y_test, y_pred_xgbr))
print(f'Final XGBoost(RandomizedSearchCV) RMSE: {rmse_xgbr}')
```

```python
def highrisk(X, y):
    df = X.join(y)
    df['risk_flag'] = df['dist1'].apply(lambda x: 1 if x > threshold else 0)
    high_risk_group = df[df['risk_flag'] == 1]
    fraud_proportion = high_risk_group['isFraud'].mean()
    return fraud_proportion

```

```python
print(f'Proportion of fraudulent transactions in high-risk group:')
print("-----------------------------------------------------------------")
print(f"Entire data set: {highrisk(X, y)}")
print(f"Test set: {highrisk(X_test, y_test)}")
print(f'Linear Regression: {highrisk(X_test, pd.DataFrame(y_pred, index=X_test.index, columns= ["dist1"]))}')
print(f'XGBoost: {highrisk(X_test, pd.DataFrame(y_pred_xgb, index=X_test.index, columns= ["dist1"]))}')
print(f'XGBoost(BayesianOptimization): {highrisk(X_test, pd.DataFrame(y_pred_xgbbo, index=X_test.index, columns= ["dist1"]))}')
print(f'XGBBoost(RandomizedSearchCV): {highrisk(X_test,pd.DataFrame(y_pred_xgbr, index=X_test.index, columns= ["dist1"]))}')
```

```python
print(f'RMSE Based on Model:')
print("-----------------------------------------------------------------")
print(f'Linear Regression: {rmse_lr}')
print(f'XGBoost: {rmse_xgb}')
print(f'XGBoost(BayesianOptimization): {rmse_xgbbo}')
print(f'XGBBoost(RandomizedSearchCV): {rmse_xgbr}')
```

In [ ]:
```python
def actualPredictedPlot(y_test, y_pred, model):
    plt.scatter(y_test, y_pred)
    plt.title(f'Actual Values vs Predicted Values from {model}')
    plt.xlabel('Actual Value')
    plt.ylabel('Predicted Value')
    plt.show()
```

In [ ]:
```python
actualPredictedPlot(y_test, y_pred, "Linear Regression")
```

In [ ]:
```python
actualPredictedPlot(y_test, y_pred_xgb, "XGBoost")
```

In [ ]:
```python
actualPredictedPlot(y_test, y_pred_xgbbo, "XGBoost (BayesianOptimization)")
```

In [ ]:
```python
actualPredictedPlot(y_test, y_pred_xgbr, "XGBoost (RandomizedSearchCV)")
```

In [ ]:
```
def residualsPlot(y_test, y_pred, model):
    residuals = y_test - y_pred
    plt.scatter(y_pred, residuals)
    plt.xlabel('Predicted Distance')
    plt.ylabel('Residuals')
    plt.title(f'Residual Plot from {model}')
    plt.axhline(y=0, color='black', linestyle='--')
    plt.show()
```

In [ ]:
```
residualsPlot(y_test, y_pred, "Linear Regression")
```

In [ ]:
```
residualsPlot(y_test, y_pred_xgb, "XGBoost")
```

In [ ]:
```
residualsPlot(y_test, y_pred_xgbbo, "XGBoost (BayesianOptimization)")
```

In [ ]:
```
residualsPlot(y_test, y_pred_xgbr, "XGBoost (RandomizedSearchCV)")
```

RQ3: How can clustering techniques be used to identify groups of transactions that may be part of coordinated fraud activities?

```
]   1 import os
    2 os.environ['PATH'] += os.pathsep + os.path.expanduser('~/.local/bin')
    3 import sys
    4 sys.path.append(os.path.expanduser('~/.local/lib/python3.10/site-packages'))
```

```
    1 import pandas as pd
    2 import numpy as np
    3 from sklearn.preprocessing import StandardScaler, OneHotEncoder
    4 from sklearn.impute import SimpleImputer
    5 from sklearn.decomposition import PCA
    6 from sklearn.cluster import KMeans, AgglomerativeClustering
    7 from sklearn.metrics import silhouette_score
    8 import matplotlib.pyplot as plt
    9 import seaborn as sns
   10 import hdbscan
   11 import gc
   12 from scipy.sparse import csr_matrix, vstack
   13 from scipy.cluster.hierarchy import dendrogram, linkage
   14 from joblib import Parallel, delayed
   15 from sklearn.metrics import silhouette_samples, silhouette_score
   16 import random
```

Data Loading and Merging

- **Objective**: Load the transaction and identity datasets and merge them based on `TransactionID`.
- **Steps**:
  - Loaded `train_transaction.csv` and `train_identity.csv`.
  - Merged the datasets using an inner join on `TransactionID`.

```
]   1 #Loading the datasets
    2 data_transaction = pd.read_csv('train_transaction.csv')
    3 data_identity = pd.read_csv('train_identity.csv')
```

```
]   1 #Merging datasets on TransactionID
    2 data = pd.merge(data_transaction, data_identity, on='TransactionID', how='left')
```

Feature Selection

- **Objective**: Select relevant features based on the project plan and methodology.
- **Selected Features**:
  - Numerical: `TransactionAmt`
  - Categorical: `card1`, `card2`, `card3`, `card4`, `card5`, `card6`, `addr1`, `addr2`
  - Derived: `D1` to `D15`, `V1` to `V339`

```
[ ]  1 #Feature selection based on our project plan and methodology
     2 selected_features = ['TransactionAmt', 'card1', 'card2', 'card3', 'card4', 'card5', 'card6', 'addr1', 'addr2'] + [f'D{i}' for i in range(1, 16)] + [f'V{i}' for i in range(1, 340)]
     3 data = data[selected_features]
```

Optimizing Data Types

- **Objective**: Optimize data types to save memory and improve computational efficiency.
- **Steps**:
  - Converted float64 columns to float32.
  - Converted int64 columns to int32.

```
[ ]  1 #Optimizing data types to save memory
     2 def optimize_data_types(df):
     3     for col in df.select_dtypes(include=['float64']).columns:
     4         df[col] = df[col].astype('float32')
     5     for col in df.select_dtypes(include=['int64']).columns:
     6         df[col] = df[col].astype('int32')
     7     return df
```

```
[ ]  1 data = optimize_data_types(data)
```

Normalization

- **Objective**: Standardize numerical features to have a mean of 0 and a standard deviation of 1.
- **Steps**:
  - Applied `StandardScaler` to `TransactionAmt`, `addr1`, and `addr2`.

```
]  1 #Normalizing numerical features
   2 scaler = StandardScaler()
   3 data[['TransactionAmt', 'addr1', 'addr2']] = scaler.fit_transform(data[['TransactionAmt', 'addr1', 'addr2']])
```

)Encoding Categorical Features

- **Objective**: Convert categorical variables into a numerical format suitable for machine learning algorithms.
- **Steps**:
  - Used `OneHotEncoder` to encode `card1`, `card2`, `card3`, `card4`, `card5`, and `card6`.
  - Converted the encoded features into a sparse DataFrame for memory efficiency.
  - Concatenated the encoded features back with the original DataFrame and dropped the original categorical columns.

```
]  1 #One-hot encoding categorical features using sparse matrices
   2 categorical_cols = ['card1', 'card2', 'card3', 'card4', 'card5', 'card6']
   3 encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=True)
   4 encoded_cols = encoder.fit_transform(data[categorical_cols])
```

```
]  1 #Converting to sparse df (more memory efficient)
   2 encoded_df = pd.DataFrame.sparse.from_spmatrix(encoded_cols, columns=encoder.get_feature_names_out(categorical_cols))
   3
   4 #Concatenating with original df
   5 data = pd.concat([data.reset_index(drop=True), encoded_df.reset_index(drop=True)], axis=1)
   6 data.drop(categorical_cols, axis=1, inplace=True)
```

Imputing Missing Values

- **Objective**: Handle missing values in the dataset to ensure complete data for analysis.
- **Steps**:
  - Implemented a function to impute missing values in chunks to handle large datasets efficiently.
  - Used `SimpleImputer` with the mean strategy to fill missing values.

```python
#Imputing the missing values manually for sparse data
def impute_sparse_data(df, chunk_size=10000):
    imputer = SimpleImputer(strategy='mean')
    sparse_chunks = []

    for start in range(0, df.shape[0], chunk_size):
        end = min(start + chunk_size, df.shape[0])
        chunk = df.iloc[start:end]

        #Imputing the chunk
        imputed_chunk = imputer.fit_transform(chunk.values)

        #Converting back to sparse format
        sparse_chunk = csr_matrix(imputed_chunk)
        sparse_chunks.append(sparse_chunk)

    #Combining all sparse chunks
    imputed_sparse_data = vstack(sparse_chunks)
    return imputed_sparse_data
```

```python
#Using the function to impute the data
imputed_data_sparse = impute_sparse_data(data)
```

```python
#Converting the imputed sparse matrix back to a df
data = pd.DataFrame.sparse.from_spmatrix(imputed_data_sparse, columns=data.columns)

#Clearing unused variables to free up memory
gc.collect()
```

```
0
```

Dimensionality Reduction

- **Objective**: Reduce the dimensionality of the dataset to facilitate clustering.
- **Steps**:
  - Applied PCA to reduce the dataset to 10 principal components.

```
1 pca = PCA(n_components=10)
2 data_reduced = pca.fit_transform(data)
```

K-Means Clustering

- **Objective**: Partition the dataset into distinct clusters using K-Means.
- **Steps**:
  - Applied K-Means clustering with 5 clusters.
  - Stored the cluster labels in the DataFrame.

```
[ ]  1 #K-Means Clustering
     2 kmeans = KMeans(n_clusters=5, random_state=42)
     3 kmeans_labels = kmeans.fit_predict(data_reduced)
     4 data['KMeans_Cluster'] = kmeans_labels
```

HDBSCAN Clustering

- **Objective**: Identify clusters of varying density using HDBSCAN.
- **Steps**:
  - Applied HDBSCAN with `min_samples` of 10 and `min_cluster_size` of 500.
  - Stored the cluster labels in the DataFrame.

```
[ ]  1 #HDBSCAN Clustering
     2 clusterer = hdbscan.HDBSCAN(min_samples=10, min_cluster_size=500)
     3 hdbscan_labels = clusterer.fit_predict(data_reduced)
     4 data['HDBSCAN_Cluster'] = hdbscan_labels
```

Hierarchical Clustering

- **Objective**: Understand the hierarchical structure of the data through clustering.
- **Steps**:
  - Randomly sampled 20,000 data points due to memory constraints.
  - Applied Agglomerative Clustering with 5 clusters.
  - Stored the cluster labels in the sampled DataFrame.

```
#Randomly sampling the data for hierarchical clustering
sample_size = 20000
sampled_indices = np.random.choice(data_reduced.shape[0], sample_size, replace=False)
sampled_data_reduced = data_reduced[sampled_indices]
```

+ Code + Te

Add code
⌘/Ctrl+M

```
#Hierarchical Clustering on the sampled data
hierarchical = AgglomerativeClustering(n_clusters=5)
hierarchical_labels = hierarchical.fit_predict(sampled_data_reduced)

#Creating a df for the sampled data to store the labels
sampled_data_df = pd.DataFrame(sampled_data_reduced, columns=[f'PC{i+1}' for i in range(sampled_data_reduced.shape[1])])
sampled_data_df['Hierarchical_Cluster'] = hierarchical_labels
```

Evaluation and Visualization

Evaluating Clustering with WCSS

- **Objective**: Determine the optimal number of clusters using the Elbow Method.
- **Steps**:
  - Calculated WCSS for different numbers of clusters.
  - Plotted the Elbow Method graph to identify the optimal number of clusters.

```
#Evaluating K-Means with WCSS
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(data_reduced)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

Silhouette Score Calculation

- **Objective**: Evaluate the quality of clustering using the Silhouette Score.
- **Steps**:
  - Randomly sampled 20,000 data points to calculate the Silhouette Score for K-Means, HDBSCAN, and Hierarchical Clustering.
  - Reported the Silhouette Scores for each clustering method.

```
#Silhouette Score Sampling
sample_size_for_silhouette = 20000
random_indices_silhouette = np.random.choice(data_reduced.shape[0], sample_size_for_silhouette, replace=False)
sampled_data_reduced_silhouette = data_reduced[random_indices_silhouette]
sampled_kmeans_labels = kmeans_labels[random_indices_silhouette]
sampled_hdbscan_labels = hdbscan_labels[random_indices_silhouette]
```

```
[ ]
#Silhouette Score for K-Means
kmeans_silhouette = silhouette_score(sampled_data_reduced_silhouette, sampled_kmeans_labels)
print(f'K-Means Silhouette Score: {kmeans_silhouette}')

#Silhouette Score for HDBSCAN
hdbscan_silhouette = silhouette_score(sampled_data_reduced_silhouette, sampled_hdbscan_labels)
print(f'HDBSCAN Silhouette Score: {hdbscan_silhouette}')

#Silhouette Score for Hierarchical Clustering
#Ensuring sampled_data_reduced_silhouette is used instead of sampled_data_reduced for hierarchical clustering
sampled_hierarchical_labels = hierarchical_labels[:sample_size_for_silhouette]
hierarchical_silhouette = silhouette_score(sampled_data_reduced_silhouette, sampled_hierarchical_labels)
print(f'Hierarchical Silhouette Score: {hierarchical_silhouette}')
```

```
#Silhouette Scores
kmeans_silhouette = 0.8564298897506002
hdbscan_silhouette = -0.18430016878234834
hierarchical_silhouette = -0.0637329143989308

#Labels and Scores
labels = ['K-Means', 'HDBSCAN', 'Hierarchical']
scores = [kmeans_silhouette, hdbscan_silhouette, hierarchical_silhouette]

#Plotting the Silhouette Scores
plt.figure(figsize=(10, 6))
bars = plt.bar(labels, scores, color=['blue', 'green', 'red'])

#Adding the scores above the bars
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 0.02, round(yval, 2), ha='center', va='bottom')

plt.ylim(-1, 1)
plt.axhline(0, color='gray', linewidth=0.8)
plt.title('Silhouette Scores for Different Clustering Methods')
plt.xlabel('Clustering Method')
plt.ylabel('Silhouette Score')
plt.show()
```

Visualizing the Clusters

Scatter Plot for K-Means

- **Objective**: Visualize the clusters formed by K-Means.
- **Steps**:
    - Created a scatter plot for the first two principal components colored by K-Means cluster labels.

```
[ ]
#Scatter plot for K-Means
plt.figure(figsize=(10, 6))
sns.scatterplot(x=data_reduced[:, 0], y=data_reduced[:, 1], hue=kmeans_labels, palette='viridis')
plt.title('K-Means Clustering')
plt.show()
```

Heatmap of Selected Features

- **Objective**: Visualize the correlations between a subset of selected features to understand the relationships within the data.
- **Steps**:
  - Selected a subset of features for the correlation matrix to focus on key variables.
  - Sampled 20,000 data points to reduce memory usage and ensure computational efficiency.
  - Ensured the sampled data is in a dense format for processing.
  - Standardized the selected features without centering to prepare for correlation analysis.
  - Computed the correlation matrix on the standardized data.
  - Plotted a heatmap with adjusted font size and fewer decimal places to visualize the correlations between the selected features.

This approach allows us to visually assess the relationships between the selected features, providing insights into potential patterns and interactions within the data.

```
[ ]  1 #Selecting a subset of features for the correlation matrix
     2 selected_features_subset = ['TransactionAmt', 'card1', 'card2', 'card3', 'addr1', 'addr2'] + [f'D{i}' for i in range(1, 6)] + [f'V{i}' for i in range(1, 21)]
     3
     4 #Ensuring the selected features are in the dataset
     5 selected_features_subset = [feature for feature in selected_features_subset if feature in data.columns]
     6
     7 #Sampling the data to reduce memory usage
     8 sample_size = 20000
     9 sampled_data_for_corr = data[selected_features_subset].sample(n=sample_size, random_state=42)
    10
    11 #Ensuring the sampled data is dense
    12 sampled_data_for_corr_dense = sampled_data_for_corr.sparse.to_dense()
    13
    14 #Standardizing the data without centering (with_mean=False)
    15 scaler = StandardScaler(with_mean=False)
    16 sampled_data_for_corr_scaled = pd.DataFrame(scaler.fit_transform(sampled_data_for_corr_dense), columns=selected_features_subset)
    17
    18 #Computing the correlation matrix on the standardized data
    19 corr_matrix_sampled_scaled = sampled_data_for_corr_scaled.corr()
    20
    21 #Plotting the heatmap with adjusted font size and fewer decimal places
    22 plt.figure(figsize=(12, 8))
    23 sns.heatmap(corr_matrix_sampled_scaled, annot=True, fmt='.1f', annot_kws={"size": 8}, cmap='coolwarm')
    24 plt.title('Heatmap of Selected Features (Sampled and Standardized Data)')
    25 plt.show()
```

Dendrogram for Hierarchical Clustering

- **Objective**: Visualize the hierarchical structure of the clusters to understand the clustering relationships within the dataset.
- **Steps**:
  - Selected a subset of features for the correlation matrix and dendrogram to focus on key variables.
  - Sampled 5000 data points to reduce memory usage and ensure computational efficiency.
  - Standardized the selected features without centering to prepare for clustering.
  - Computed the correlation matrix and plotted a heatmap to visualize correlations between selected features.
  - Created a dendrogram using the linkage method with 'ward' to display the hierarchical clustering.
  - Increased the figure size and rotated x-axis labels for better readability.
  - Optionally truncated the dendrogram to show only the top levels of the hierarchy, making it more readable.

This approach allows us to visually assess the hierarchical relationships between clusters and understand how the selected features contribute to the clustering structure.

```
 1 #Selecting a subset of features for the correlation matrix
 2 selected_features_subset = ['TransactionAmt', 'card1', 'card2', 'card3', 'addr1', 'addr2'] + [f'D{i}' for i in range(1, 6)] + [f'V{i}' for i in range(1, 21)]
 3
 4 #Ensuring the selected features are in the dataset
 5 selected_features_subset = [feature for feature in selected_features_subset if feature in data.columns]
 6
 7 #Sampling the data to reduce memory usage
 8 sample_size = 5000
 9 sampled_data_for_corr = data[selected_features_subset].sample(n=sample_size, random_state=42)
10
11 #Ensuring the sampled data is dense
12 sampled_data_for_corr_dense = sampled_data_for_corr.sparse.to_dense()
13
14 #Standardizing the data without centering (with_mean=False)
15 scaler = StandardScaler(with_mean=False)
16 sampled_data_for_corr_scaled = pd.DataFrame(scaler.fit_transform(sampled_data_for_corr_dense), columns=selected_features_subset)
17
18 #Computing the correlation matrix on the standardized data
19 corr_matrix_sampled_scaled = sampled_data_for_corr_scaled.corr()
20
21 #Plotting the heatmap with adjusted font size and fewer decimal places
22 plt.figure(figsize=(12, 8))
23 sns.heatmap(corr_matrix_sampled_scaled, annot=True, fmt='.1f', annot_kws={"size": 8}, cmap='coolwarm')
24 plt.title('Heatmap of Selected Features (Sampled and Standardized Data)')
25 plt.show()
26
27 #Dendrogram for Hierarchical Clustering on the sampled data
28 linked = linkage(sampled_data_for_corr_scaled, method='ward')
29
```

HDBSCAN Scatter Plot

[26]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.scatterplot(x=data_reduced[:, 0], y=data_reduced[:, 1], hue=hdbscan_labels,␣
 ↪palette='viridis')
plt.title('HDBSCAN Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Cluster')
plt.show()
```

[ ]
```
#Plotting the dendrogram with increased figure size and rotated labels
plt.figure(figsize=(15, 10))
dendrogram(linked)
plt.xticks(rotation=90)
plt.title('Hierarchical Clustering Dendrogram')
plt.show()

#Truncating the dendrogram to make it more readable
plt.figure(figsize=(15, 10))
dendrogram(linked, truncate_mode='level', p=5)
plt.xticks(rotation=90)
plt.title('Hierarchical Clustering Dendrogram (Truncated)')
plt.show()
```

Conclusion

- **Summary**:
  - The analysis provided insights into the clustering structure of the dataset.
  - K-Means clustering showed promising results with a high silhouette score.
  - Hierarchical clustering and HDBSCAN require further parameter tuning or alternative approaches to improve performance.