

INTERNATIONAL UNIVERSITY- VIETNAM NATIONAL UNIVERSITY, HCM CITY

School of Computer Science & Engineering

-----□ □ □-----



REPORT

FRAUD DETECTION IN BANK TRANSACTION

Machine Learning Course

Course by Dr. Ho Van Long

2024 - 2025

MEMBER LIST

No	Full Name	Student ID	Position
1	Phan Thế Thiện	ITDSIU20084	Leader
2	Trịnh Tiến Đạt	ITDSIU20109	Member
3	Phạm Quang Tường	ITDSIU20108	Member

WORK DISTRIBUTION

No	Full Name	Workload
1	Phan Thế Thiện	34%
2	Trịnh Tiến Đạt	33%
3	Phạm Quang Tường	33%

TABLE OF CONTENT

Abstract

I. Introduction

1. Purpose
2. Goal
3. Algorithms selection

II. Dataset Overview

- Key Features
- Objectives

III. Implementation

1. Import and setup
2. Load and Explore Dataset
3. Unique Value Exploration
4. Exploratory Data Analysis (EDA)

IV. Anomaly Detection

1. Identifying Potential Frauds with K-means Clustering
2. DBSCAN Clustering for Anomaly Detection
3. Isolation Forest for Anomaly Detection

V. Conclusion

VI. Reference

Abstract

This report explores the application of machine learning algorithms for detecting potential fraudulent transactions in banking systems. Specifically, we employ three distinct methods for anomaly detection: K-means clustering, DBSCAN clustering, and Isolation Forest. These algorithms are applied to analyze transaction patterns and identify anomalies that may indicate fraudulent activities.

The study emphasizes handling challenges such as imbalanced data and high-dimensionality in financial datasets. Our results demonstrate that the combined use of clustering and isolation-based techniques provides robust and efficient detection of outliers, achieving high performance in identifying frauds. This approach

offers significant potential for real-world applications in enhancing security and reducing financial losses in the banking industry.

I. Introduction

1. Purpose:

Fraud detection has become a critical challenge in the finance, e-commerce, and telecommunications industries. The surge in digital transactions, while offering convenience, has also led to more sophisticated fraudulent activities. This complexity necessitates the use of advanced techniques to effectively combat these threats. Fraudulent activities not only cause significant financial losses but also erode consumer trust, making early detection and prevention paramount.

2. Goal:

The primary goal of this project is to identify fraudulent activities within a transaction dataset using clustering and anomaly detection techniques. These methods offer a data-driven approach to uncovering suspicious patterns or irregularities that may indicate fraud. Unlike traditional rule-based systems, clustering and anomaly detection algorithms dynamically identify deviations and are particularly effective for large-scale and evolving datasets.

3. Algorithms selection:

To address this problem, we employ three machine learning algorithms: K-Means Clustering, DBSCAN (Density-Based Spatial Clustering of Applications with Noise), and Isolation Forest. These algorithms are chosen for their strengths in identifying anomalies in complex datasets. Through the application of these methods, this project aims to demonstrate their effectiveness in improving fraud detection systems by uncovering valuable insights, enhancing detection accuracy, and reducing false positives.

II. Dataset Overview

This dataset contains transaction data intended for the analysis and detection of potential fraudulent activities. The data includes various attributes that provide insights into transactional behavior, customer profiles, and other contextual details, making it suitable for applying different clustering and anomaly detection techniques.

Key Features:

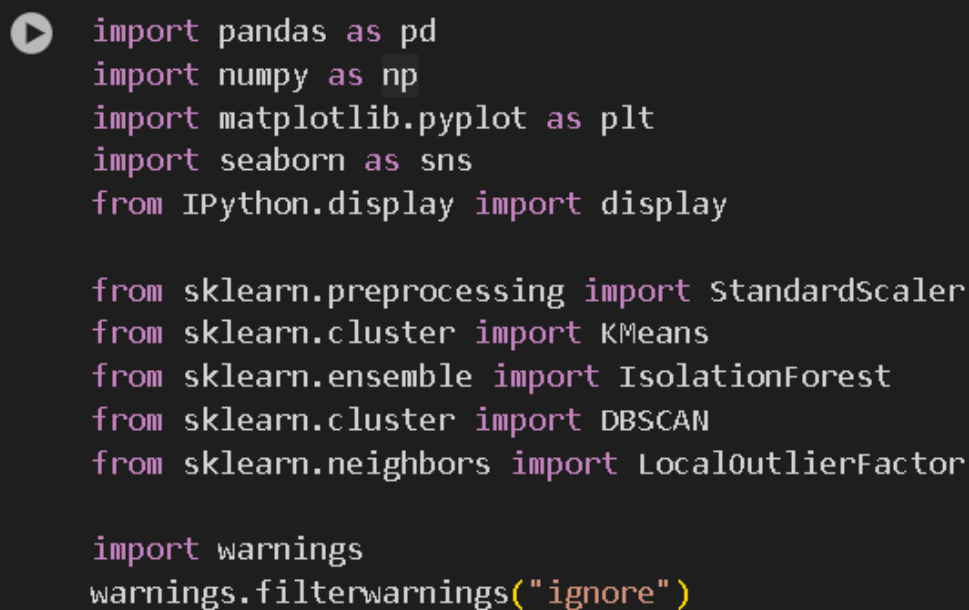
- **TransactionID:** A unique identifier for each transaction.
- **AccountID:** A unique identifier for the account associated with the transaction.
- **TransactionAmount:** The monetary value of each transaction, varying from small expenses to large purchases.
- **TransactionDate:** The date and time when the transaction occurred.
- **TransactionType:** Categorical value indicating whether the transaction was a 'Credit' or 'Debit'.
- **Location:** The geographic location where the transaction took place.
- **DeviceID:** A unique identifier for the device used to perform the transaction.
- **IP Address:** The IP address associated with the transaction.
- **MerchantID:** A unique identifier for merchants involved in the transaction.
- **Channel:** Indicates the channel through which the transaction was conducted (e.g., Online, ATM, Branch).
- **AccountBalance:** The balance remaining in the account after the transaction.
- **TransactionDuration:** Duration of the transaction in seconds.
- **LoginAttempts:** The number of login attempts made before the transaction.

Objectives:

- **Fraud Detection:** Utilize clustering algorithms and anomaly detection techniques to identify potential fraudulent transactions.
- **Behavioral Insights:** Gain insights into transactional behavior patterns to understand normal vs. anomalous behavior.

III. Implementation

1. Imports and Setup



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.ensemble import IsolationForest
from sklearn.cluster import DBSCAN
from sklearn.neighbors import LocalOutlierFactor

import warnings
warnings.filterwarnings("ignore")
```

Figure 1: Import library

We utilized key Python libraries such as Pandas and NumPy for data processing, Matplotlib, and Seaborn for visualization, and various clustering and anomaly detection algorithms from Scikit-learn for model implementation. These imports collectively supported the project's data analysis and fraud detection goals.

2. Load and Explore Dataset

In this phase, the dataset is loaded, and its structure is examined to understand the variables and their distributions. The dataset used in this project is *bank_transactions_data_2.csv*, and it is loaded into a data frame named *df*. This

step provides an overview of the data, helping to identify potential issues such as missing values, outliers, and the nature of the features.

Steps for Exploration:

- **Check Dataset Dimensions:** using `df.shape()`, we retrieve the number of rows and columns in the dataset. This helps to gauge the dataset's scale and complexity.

```
Shape of the dataset: (2512, 16)
```

- **df.head()** function is used to display the first few rows of the dataset. This provides a glance at the structure and content of the data, helping us understand feature names and their possible values.

	0	1	2	3	4
TransactionID	TX000001	TX000002	TX000003	TX000004	TX000005
AccountID	AC00128	AC00455	AC00019	AC00070	AC00411
TransactionAmount	14.09	376.24	126.29	184.5	13.45
TransactionDate	2023-04-11 16:29:14	2023-06-27 16:44:19	2023-07-10 18:16:08	2023-05-05 16:32:11	2023-10-16 17:51:24
TransactionType	Debit	Debit	Debit	Debit	Credit
Location	San Diego	Houston	Mesa	Raleigh	Atlanta
DeviceID	D000380	D000051	D000235	D000187	D000308
IP Address	162.198.218.92	13.149.61.4	215.97.143.157	200.13.225.150	65.164.3.100
MerchantID	M015	M052	M009	M002	M091
Channel	ATM	ATM	Online	Online	Online
CustomerAge	70	68	19	26	26
CustomerOccupation	Doctor	Doctor	Student	Student	Student
TransactionDuration	81	141	56	25	198
LoginAttempts	1	1	1	1	1
AccountBalance	5112.21	13758.91	1122.35	8569.06	7429.4
PreviousTransactionDate	2024-11-04 08:08:08	2024-11-04 08:09:35	2024-11-04 08:07:04	2024-11-04 08:09:06	2024-11-04 08:06:39
BalanceChange	5098.12	13382.67	996.06	8384.56	7442.85
TransactionHour	16	16	18	16	17
AverageTransactionAmount	460.698571	304.622857	237.0475	236.48375	280.796667
UniqueLocationsCount	7	7	4	7	6
Cluster	1	2	1	1	2
DistanceToCentroid	0.627683	0.912412	0.276223	0.616155	0.639987
DBSCAN_Cluster	0	0	0	0	0
Cluster_Description	Normal	Normal	Normal	Normal	Normal
AnomalyScore	0.184358	0.097301	0.178514	0.150973	0.142493
IsAnomaly	1	1	1	1	1
AnomalyLabel	Normal	Normal	Normal	Normal	Normal

Figure 2: Transaction list

- **df.info()** gives a summary of the columns, non-null values, data types, and memory usage.

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2512 entries, 0 to 2511
Data columns (total 16 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   TransactionID                         2512 non-null   object
1   AccountID                           2512 non-null   object
2   TransactionAmount                    2512 non-null   float64
3   TransactionDate                      2512 non-null   object
4   TransactionType                      2512 non-null   object
5   Location                             2512 non-null   object
6   DeviceID                            2512 non-null   object
7   IP Address                          2512 non-null   object
8   MerchantID                          2512 non-null   object
9   Channel                             2512 non-null   object
10  CustomerAge                          2512 non-null   int64
11  CustomerOccupation                  2512 non-null   object
12  TransactionDuration                 2512 non-null   int64
13  LoginAttempts                      2512 non-null   int64
14  AccountBalance                     2512 non-null   float64
15  PreviousTransactionDate             2512 non-null   object
dtypes: float64(2), int64(3), object(11)
memory usage: 314.1+ KB
None
```

Figure 3: Data information

- **df.describe().T** provides key statistics, including count, mean, min, and max values for numerical columns, helping to understand data distribution.

Statistical Summary:								
	count	mean	std	min	25%	50%	75%	max
TransactionAmount	2512.0	297.593778	291.946243	0.26	81.885	211.14	414.5275	1919.11
CustomerAge	2512.0	44.673965	17.792198	18.00	27.000	45.00	59.0000	80.00
TransactionDuration	2512.0	119.643312	69.963757	10.00	63.000	112.50	161.0000	300.00
LoginAttempts	2512.0	1.124602	0.602662	1.00	1.000	1.00	1.0000	5.00
AccountBalance	2512.0	5114.302966	3900.942499	101.25	1504.370	4735.51	7678.8200	14977.99

Figure 4: Statistical summary table

After display, we have an overview about this dataset:

- Shape of the Dataset:

- Number of Rows: 2,512
- Number of Columns: 16


- Dataset Information:

- Total Columns: 16
- Numeric columns include TransactionAmount, CustomerAge, TransactionDuration, LoginAttempts, and AccountBalance.
- Date columns like TransactionDate and PreviousTransactionDate will likely need to be converted to DateTime objects for time-series analysis.
- Categorical columns include fields like TransactionType, Location, Channel, etc.

- Statistical Summary:


- The mean transaction amount is approximately 297.59.
- CustomerAge ranges from 18 to 80, with an average age of approximately 44.67.
- AccountBalance ranges significantly, with a maximum of around 14,978.

Before proceeding with the analysis, we conducted a thorough check for potential data quality issues.



```
# Check for missing and duplicated values
print(f'\nMissing values: {df.isna().sum().sum()}')
print(f'Duplicated values: {df.duplicated().sum()}')
```

Figure 5: Check missing and duplicate values



```
Missing values: 0
Duplicated values: 0
```

Figure 6: Results checking missing values and duplicate values

The dataset was confirmed to have no missing values or duplicated entries, ensuring the reliability of subsequent processing and analysis.

3. Unique Value Exploration

In this step, the unique values of each column in the dataset were analyzed to uncover meaningful insights and prepare for further processing. The key objectives of this analysis were as follows:

- The number of unique values in each column was examined to determine potential categorical features and to detect any inconsistencies within the data.
- Specific key columns were evaluated for their value distributions, providing a deeper understanding of the data's characteristics and trends.
- Potential missing or anomalous values were identified to facilitate targeted preprocessing and ensure data integrity.

Insights from the unique value analysis:

- Column Types:

- **Numerical Columns:** ['TransactionAmount', 'CustomerAge', 'TransactionDuration', 'LoginAttempts', 'AccountBalance']
- **Categorical Columns:** ['TransactionID', 'AccountID', 'TransactionDate', 'TransactionType', 'Location', 'DeviceID', 'IP Address', 'MerchantID', 'Channel', 'CustomerOccupation', 'PreviousTransactionDate']

4. Exploratory Data Analysis (EDA)

4.1 Histogram of Numeric Columns

```
def plot_histograms(df, numerical_columns):  
    for column in numerical_columns:  
        plt.figure(figsize=(8, 5))  
        # Create a histogram with KDE  
        sns.histplot(df[column], bins=30, kde=True, color='#ff8c00')  
        plt.title(f'Distribution of {column}')  
        plt.xlabel(column)  
        plt.ylabel('Frequency')  
        plt.show()  
  
numerical_columns = ['TransactionAmount', 'CustomerAge', 'TransactionDuration', 'LoginAttempts', 'AccountBalance']  
plot_histograms(df, numerical_columns)
```

Figure 7: Code to plot histograms of numerical columns

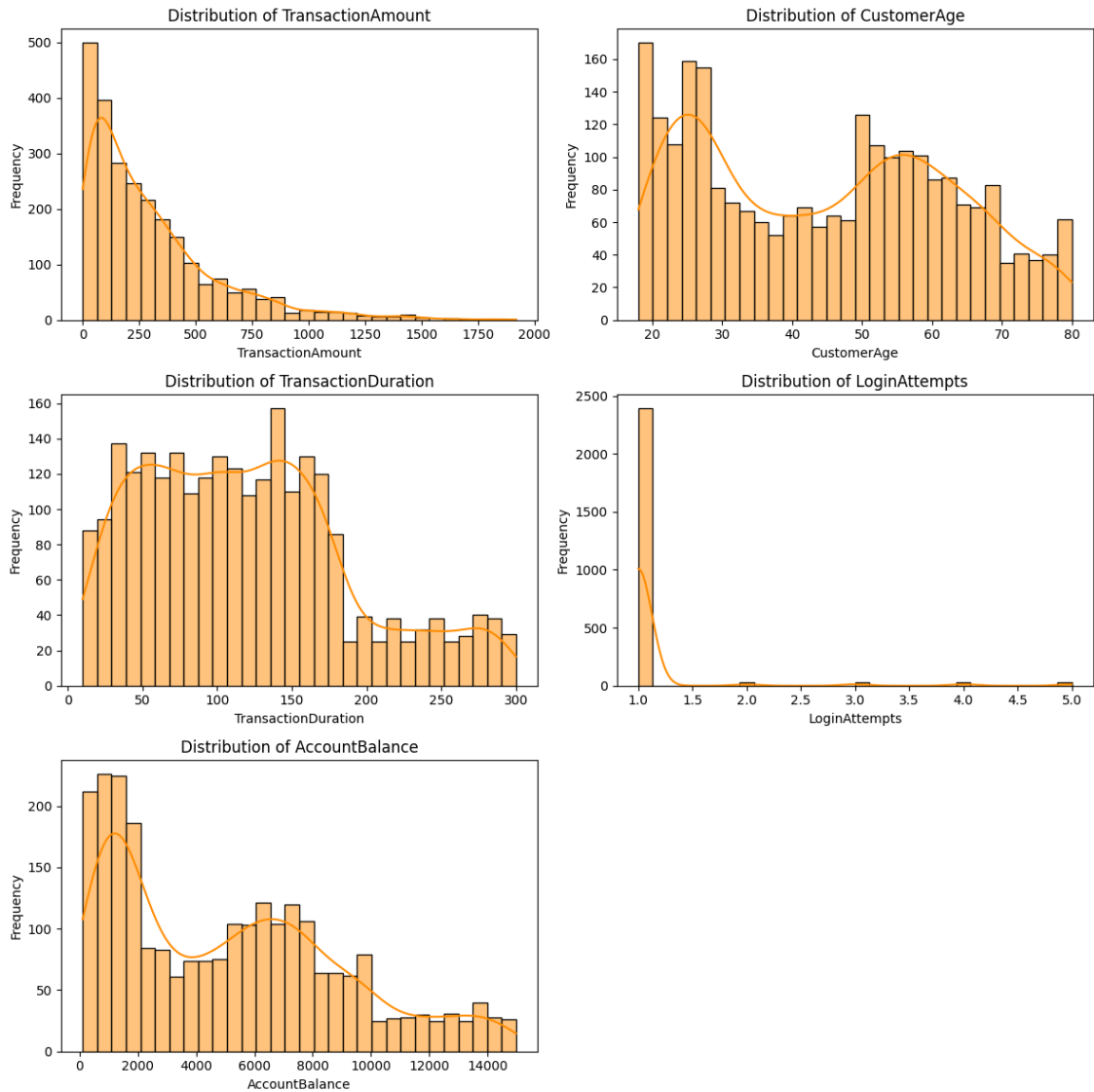


Figure 8: Histograms of numerical columns

In conclusion, the analysis highlights that most numerical data has a skewed distribution with a significant number of outliers for features like TransactionAmount, TransactionDuration, and AccountBalance. CustomerAge is relatively stable, while LoginAttempts tends to cluster around a single value.

4.2 Correlation Analysis

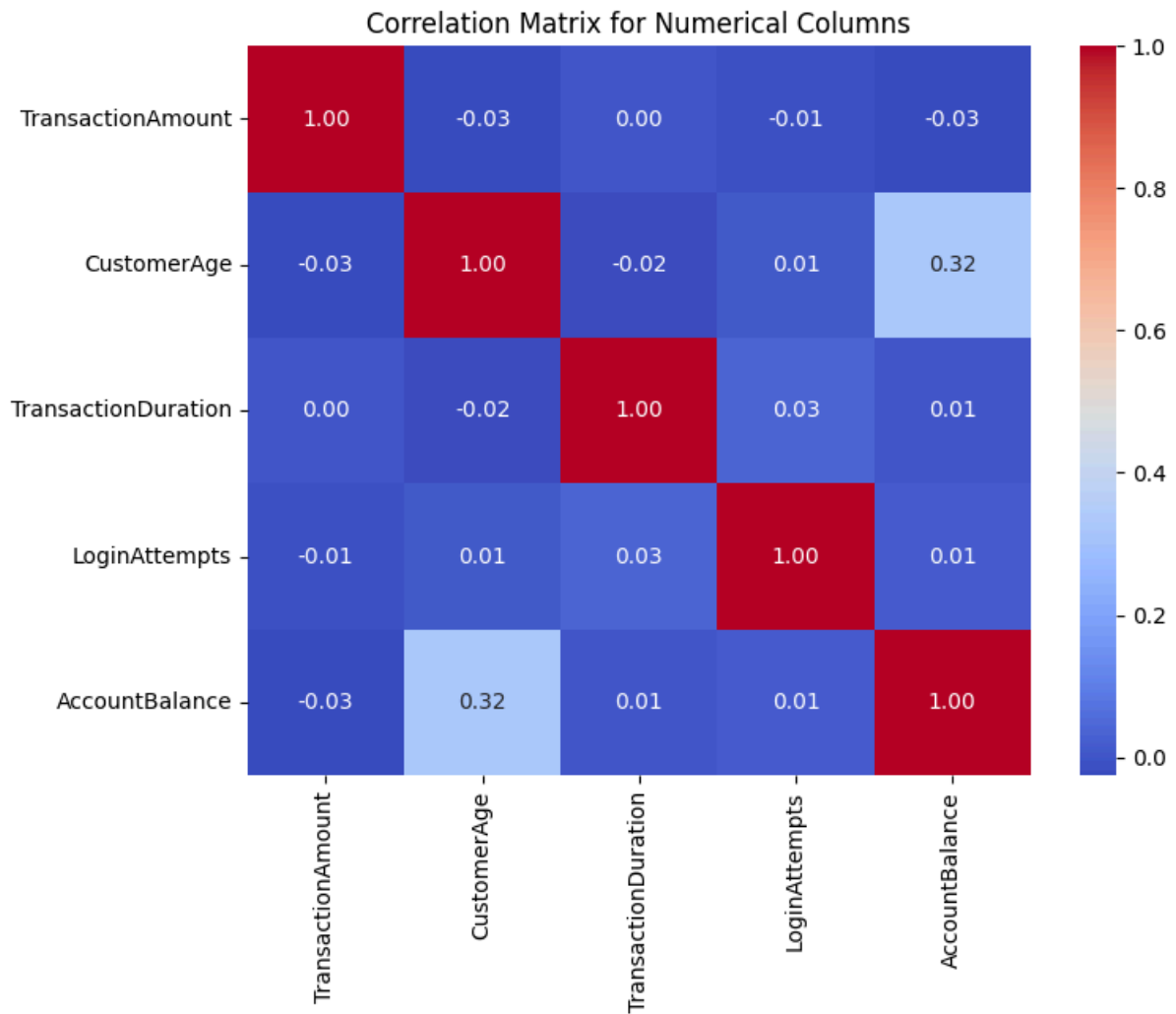


Figure 9: Correlation matrix of numerical columns

Most of the numerical variables show weak to moderate correlations with one another, indicating that the variables are relatively independent in terms of linear relationships.

4.3 Pie chart of categorical columns

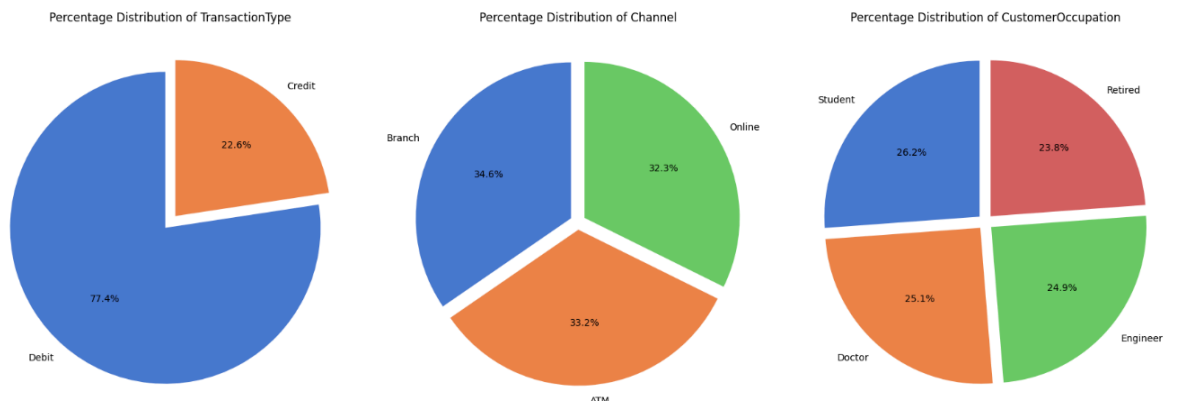


Figure 10: Pie chart of Transaction Type, Channel, Customer Occupation categorical columns

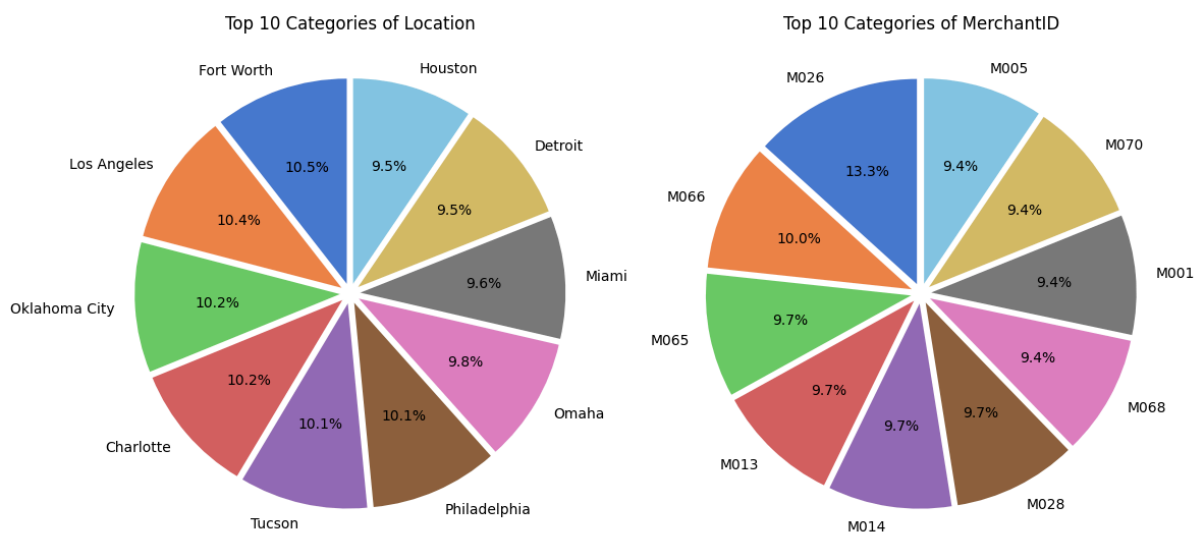


Figure 11: Pie chart of top 10 categories in Location, MerchantID columns

4.4 Visualize high-value transaction distribution by TransactionType

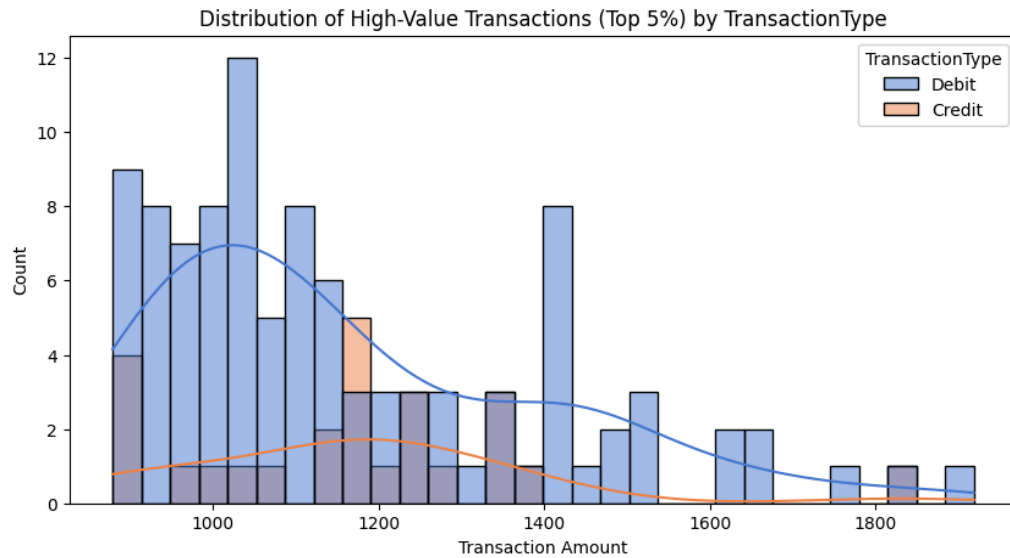


Figure 12: Histogram for the Transaction Amount column of high-value transactions by Transaction Type

- Frequency: High-value Debit transactions occur more frequently than Credit.
- Distribution: Debit is concentrated in the range of 1000-1200, Credit is more evenly distributed.
- Concentration: Debit has higher concentration, Credit is spread evenly

4.5 Visualize high-value transaction distribution by Channel

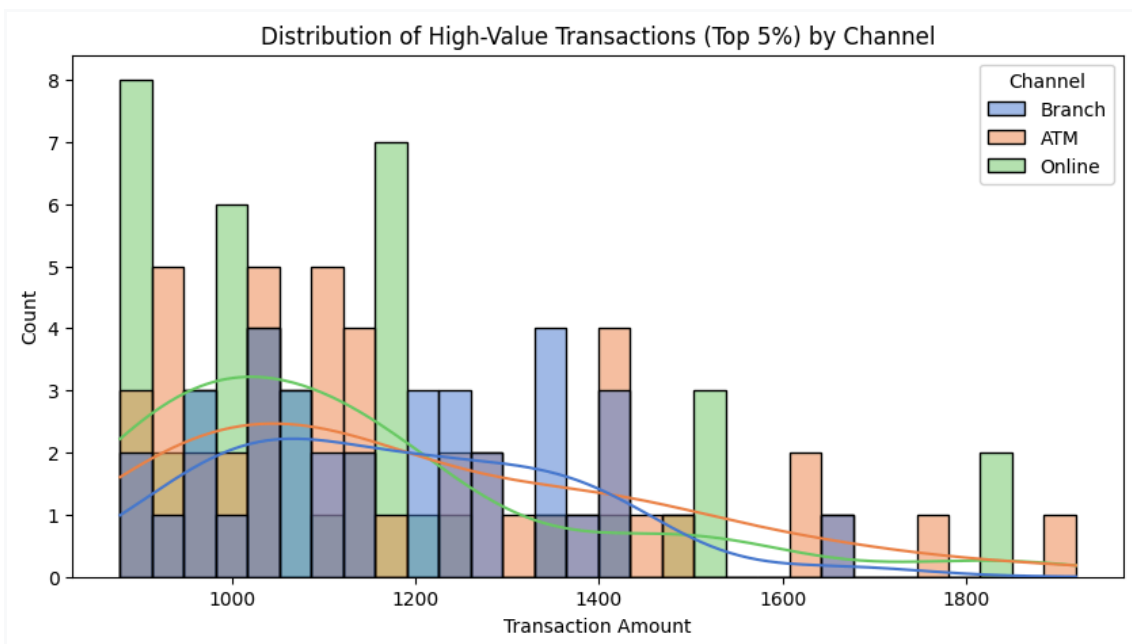


Figure 13: Histogram for the Transaction Amount column of high-value transactions by Channel

- Analyze the distribution of high-value transactions by channel: This chart helps better understand the distribution of high-value transactions, divided by different Channels
- Identify the differences between transaction channels so you can easily see which channel has the highest value transactions, as well as their distribution. This can help identify channels with high transaction frequency or unusual transactions in specific channels.

4.6 Visualize consecutive transactions with short time differences

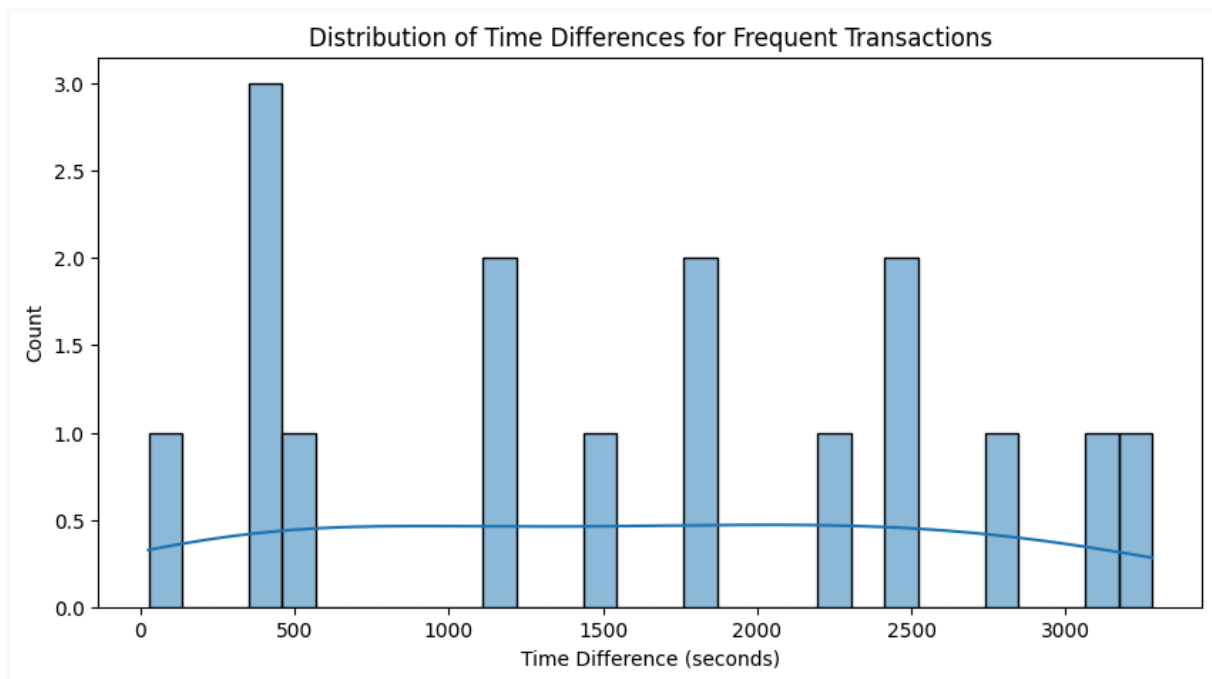


Figure 14: Histogram of short time intervals between consecutive transactions

- High frequency: The number of consecutive transactions within a short period with the highest frequency around 500 seconds.
- Secondary peaks: Several consecutive trades have secondary peaks at time intervals of 1000, 2000, and 2500 seconds.
- Evenly distributed: The time between consecutive transactions is evenly distributed with a certain number of time clusters.
- Detect high-frequency, short-duration transactions (i.e., transactions that may have occurred within the same trading session or indicate fraud).
- Detecting consecutive transactions over a short period can help detect fraudulent activities, such as multiple withdrawals in a short period or suspicious transactions over a small period, which could be a sign of fraud. signs of unusual transactions.

4.7 Visualize balance changes

- Calculate account balance change (BalanceChange): Based on the transaction type (TransactionType) whether Debit (withdrawal) or Credit (deposit), the program will calculate the change in account balance after each transaction.
 - If it is Debit, the balance will be reduced by the transaction amount.
 - If it is Credit, the balance will increase by the transaction amount.
- Identify transactions with large balance changes: The code uses a threshold calculated as the average plus 2 times the standard deviation (to identify transactions with unusual changes larger than normal levels). Transactions with balance changes greater than this threshold are considered potentially unusual or fraudulent.
- Visualize balance changes: A histogram is drawn to visualize the distribution of changes in account balances. This helps to better understand the characteristics of balance changes and identify transactions with large changes.

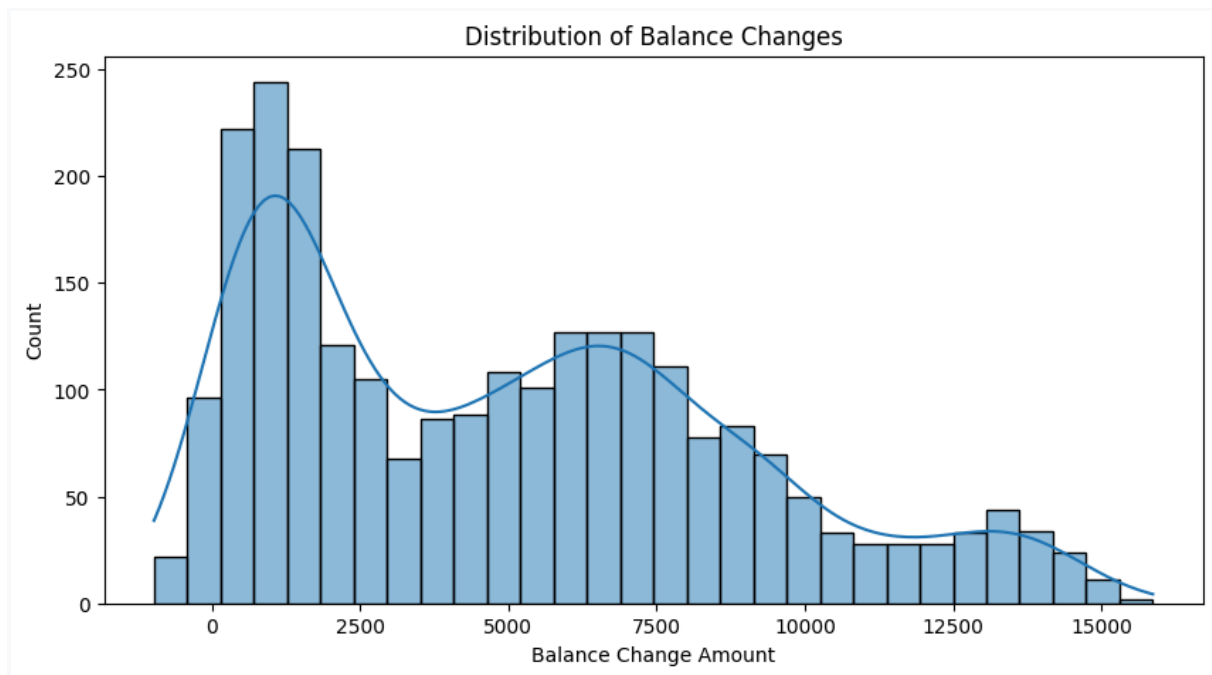


Figure 15: Histogram of Balance Changes in Transactions

4.8 Exploring the Impact of Login Attempts on Transactions

- Analyze transactions with multiple login attempts (LoginAttempts) and determine the relationship between the number of login attempts and other transaction characteristics such as TransactionAmount, TransactionType, and Channel. This is an important step in detecting fraudulent or suspicious activity, as multiple login attempts can be a sign of account attacks.
- Filter transactions with more than one login attempt: The first snippet filters out transactions with login attempts (LoginAttempts) greater than 1. This could be a sign of suspicious behavior, like an attack. brute-force.
- Show the distribution of login attempts: A histogram is created to show the distribution of login attempts across the entire dataset. This helps to better understand how login attempts are distributed and detect outliers.
- Relationship between login attempts and transaction amount (TransactionAmount): Boxplot is used to find the relationship between login attempts and transaction amount. It helps to check if there is any association between making multiple login attempts and transaction value.
- Analyze transactions with multiple login attempts by transaction type and channel: Count Plots are drawn to analyze the frequency of transactions with multiple login attempts based on factors such as TransactionType, Channel.

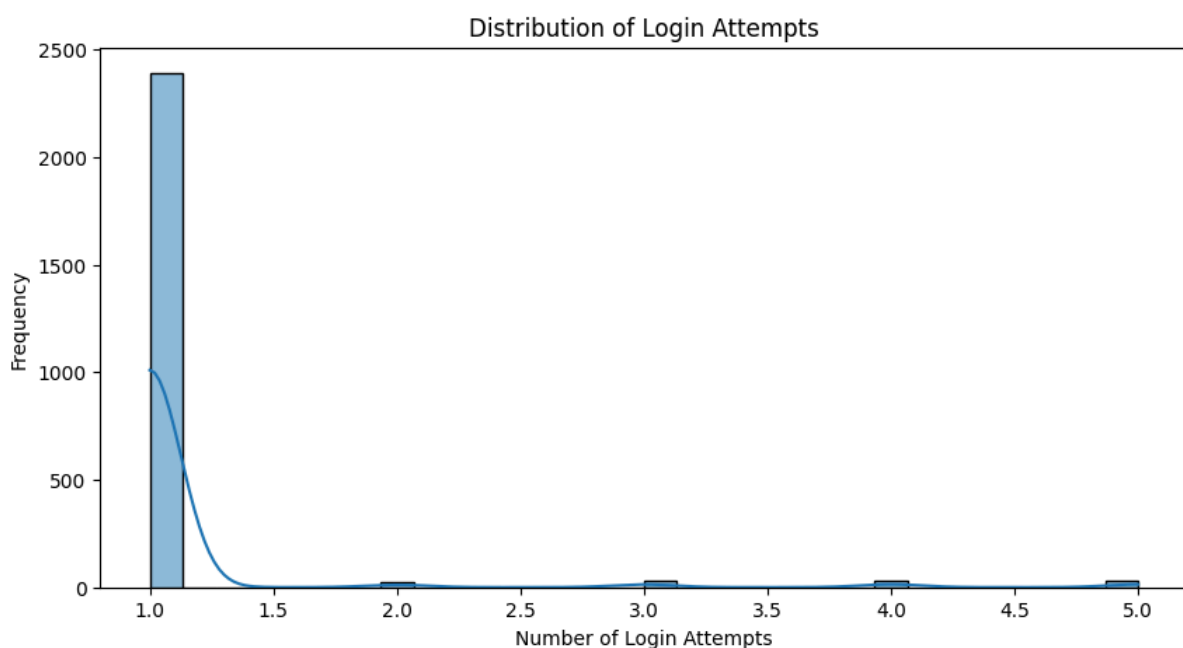


Figure 16: Distribution of Login Attempts

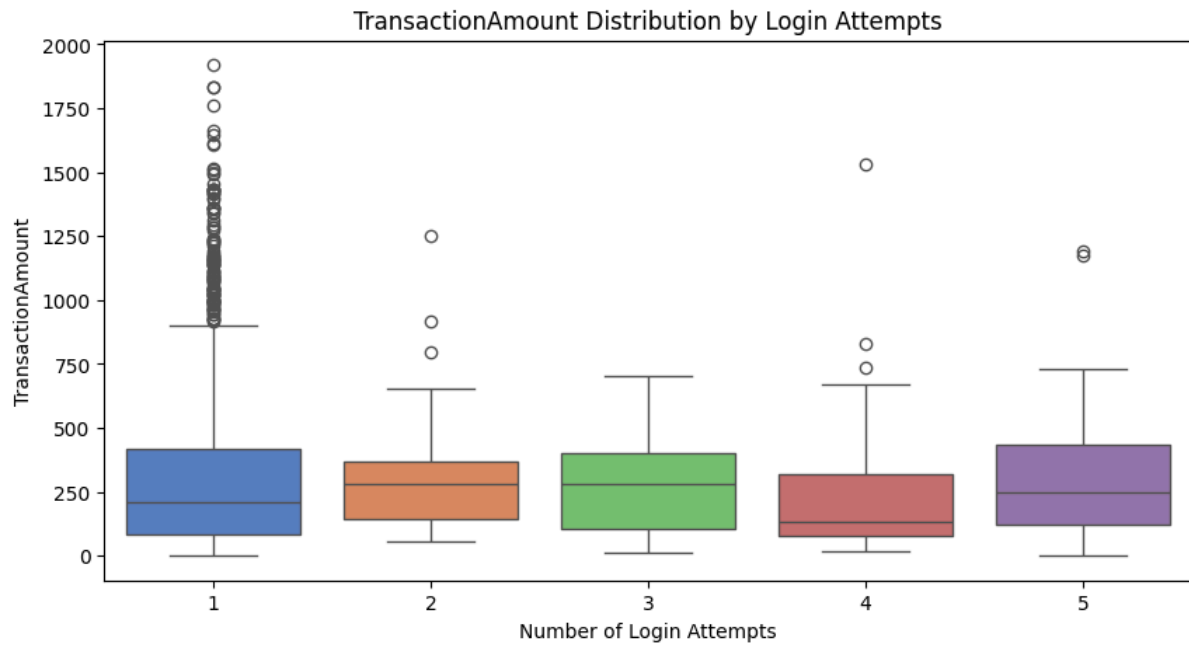


Figure 17: Transaction Amount Distribution by Login Attempts

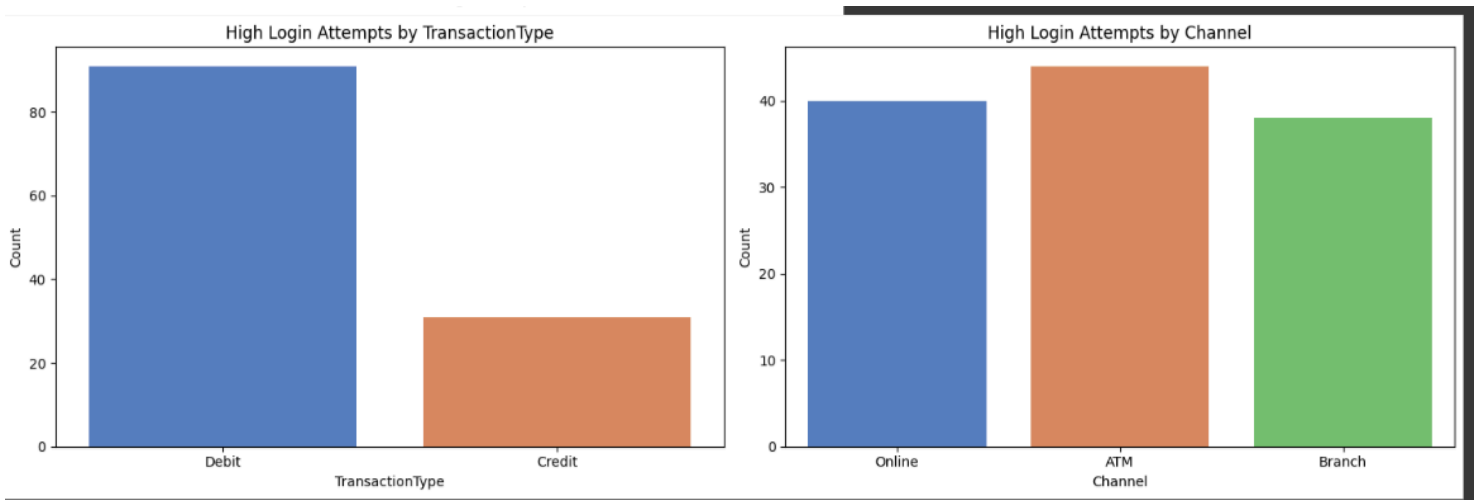


Figure 18: High Login Attempts by TransactionType and Channel

4.9 Analyze transactions by location

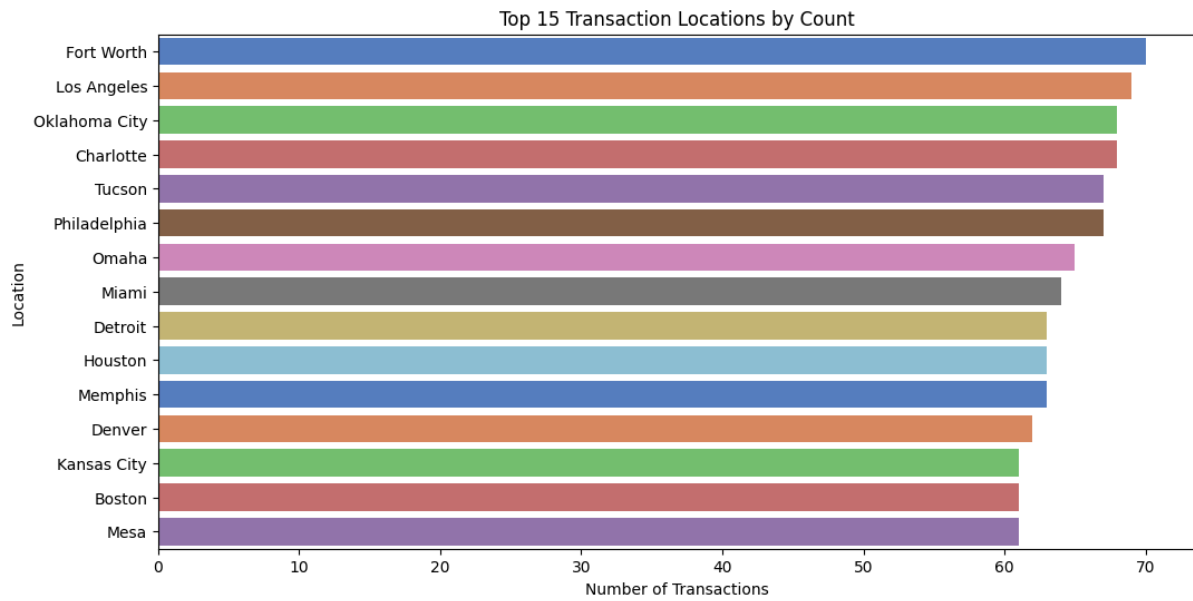


Figure 19: Top 15 Transaction Locations by Count

- Identify the areas or locations with the most transactions.
- Calculate the number of unique positions each account has traded (which can be different locations where that account trades). Draw a histogram to show the distribution of the number of unique locations visited by each account.

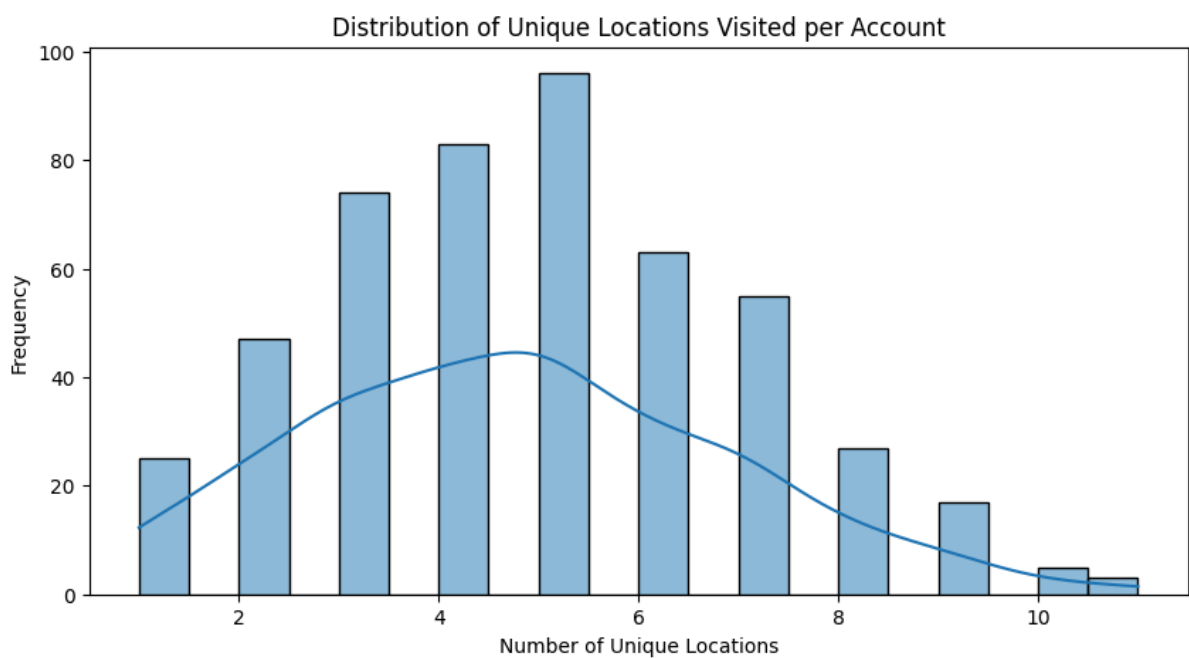
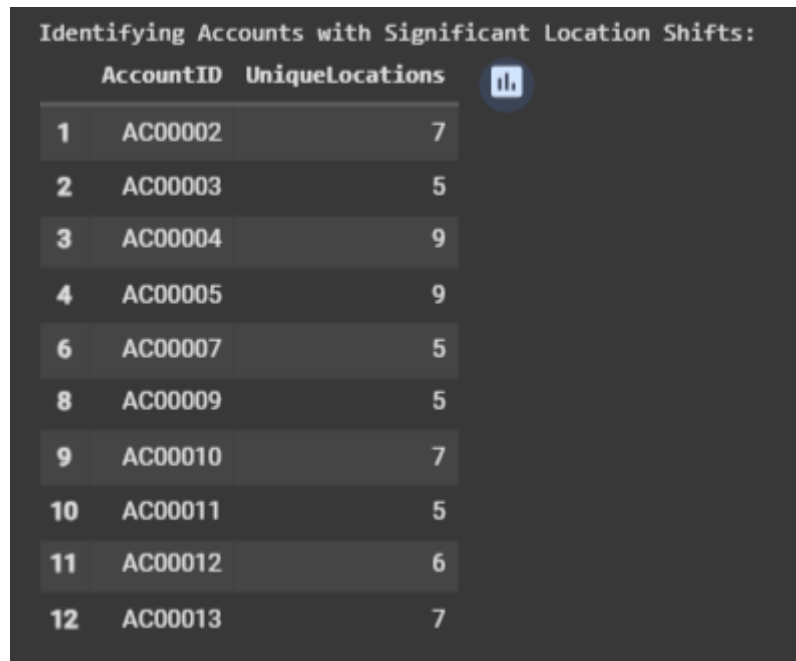


Figure 20: Histogram the distribution of the number of unique locations visited by each account.

- Filter and display accounts with large location changes, i.e. more than 3 unique locations. This can help identify accounts with unusual behavior, such as suspicious account movements.



	AccountID	UniqueLocations
1	AC00002	7
2	AC00003	5
3	AC00004	9
4	AC00005	9
6	AC00007	5
8	AC00009	5
9	AC00010	7
10	AC00011	5
11	AC00012	6
12	AC00013	7

Figure 21: Filter and display accounts with large location changes

In conclusion, analyze transactions by location to find unusual or behavioral patterns related to trading from multiple locations or significant location changes of accounts.

4.10 Analyze transactions made during unusual hours

Identify transactions made during irregular hours (before 9 AM and after 6 PM) and analyze their number and type of transactions. This helps detect transactions that may be associated with unusual or fraudulent behavior, as transactions outside of business hours can be a sign of suspicious activity.

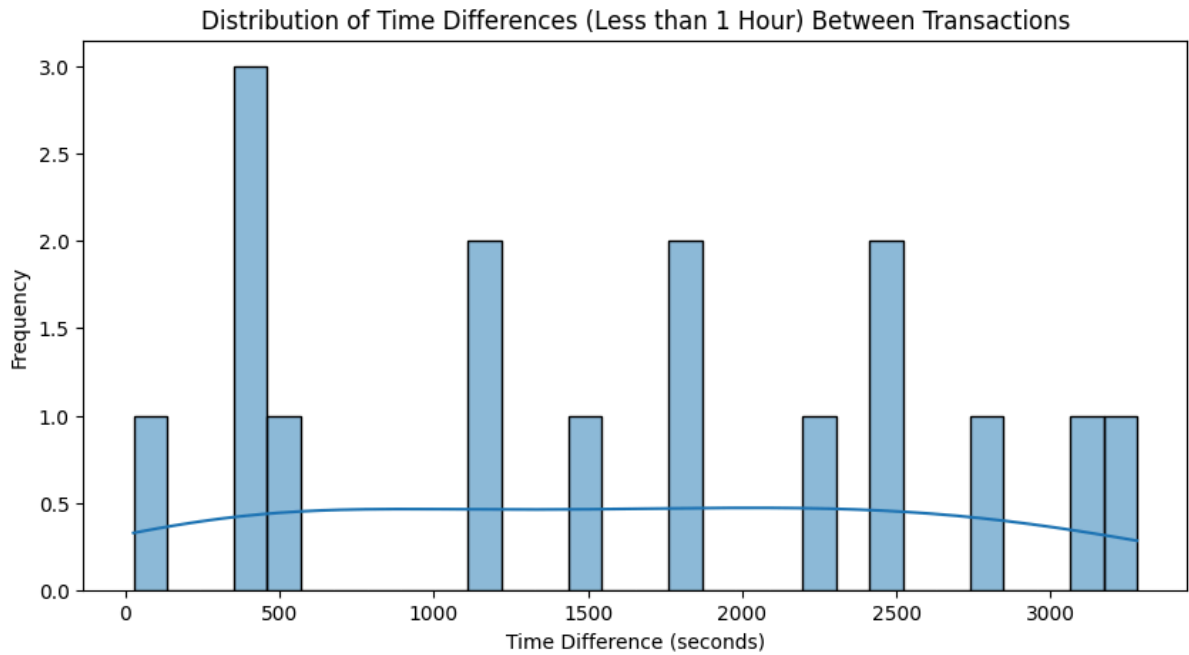


Figure 22: display the number of transactions by each transaction type (*TransactionType*) during unusual hours.

4.11 Analyze High-Value Transactions

Identify transactions that are significantly higher than the average transaction amount for each account. This helps in detecting potential fraudulent activities, as unusually high-value transactions can be a sign of suspicious behavior.

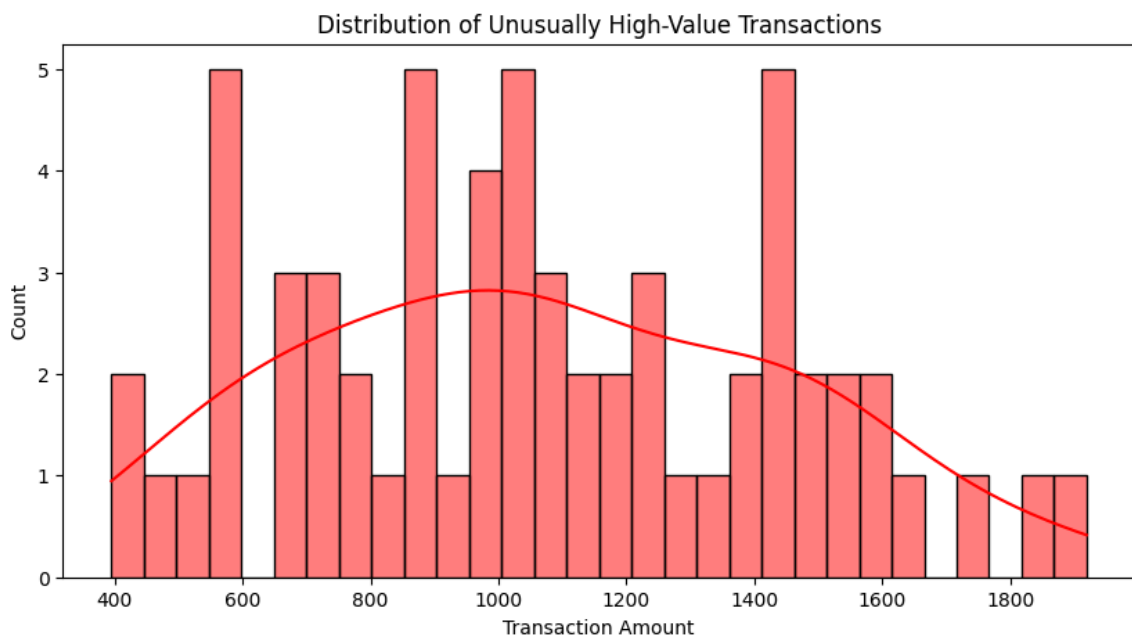


Figure 23: displays the distribution of unusually high-value transactions

The chart shows the distribution of unusually high-value transactions, indicating that there are peaks around certain transaction amounts. This visualization helps in understanding the range and frequency of these high-value transactions.

4.12 Accounts with High Location Variability

Identify accounts with high variability in transaction locations. This helps in detecting potential fraudulent activities, as high variability in transaction locations can be a sign of suspicious behavior.

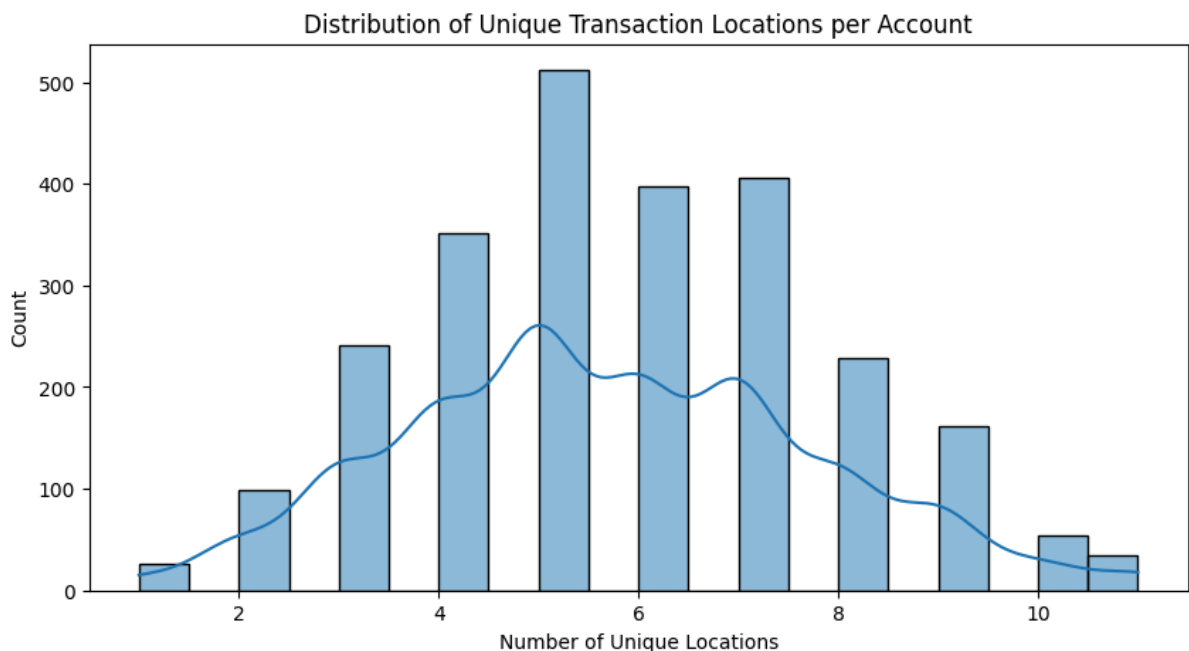


Figure 24: displays distribution of unique transaction locations per account

Each bar in the chart represents the number of accounts having a specific range of unique locations. The KDE line provides a smooth estimate of the distribution, highlighting peaks and trends in the number of unique locations. The highest peak is at 5 unique locations, indicating most accounts have transactions in 5 different locations. This visualization helps identify patterns in transaction behavior, which can be useful for detecting anomalies or understanding customer behavior.

4.13 Analyze Transaction Distribution by Channel and Transaction Type

Analyze the distribution of transactions by different channels (ATM, Branch, Online) and transaction types (Credit, Debit). This analysis helps understand the patterns and volumes of transactions across different channels, which is useful for detecting potential fraudulent activities and optimizing channel performance.

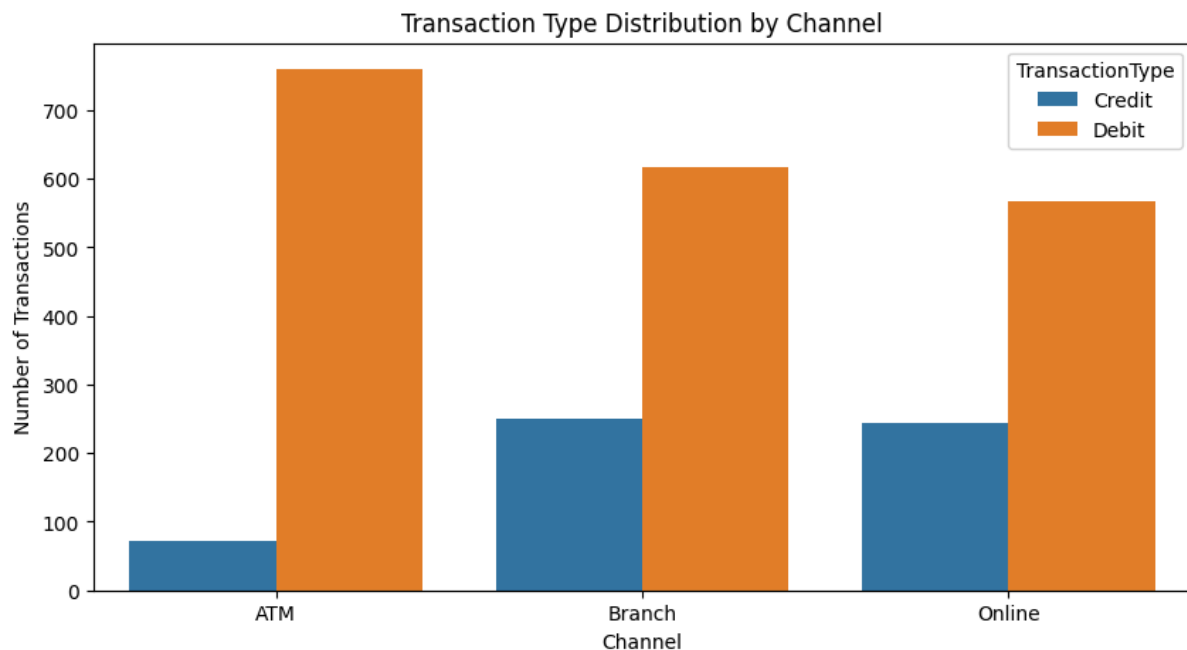


Figure 25: displays the number of transactions by each channel and transaction type

Each bar represents the number of transactions within a specific channel, categorized by transaction types (Credit and Debit). The colors in the bars differentiate between Credit (blue) and Debit (orange) transactions. This visualization helps identify which channels and transaction types are most prevalent, which can be useful for analyzing fraud likelihood, customer behavior, and channel performance.

4.14 Analyze Multiple Accounts Using the Same Device

Identify devices that are associated with multiple accounts. This helps detect potential fraudulent activities, as multiple accounts using the same device can be a sign of suspicious behavior.

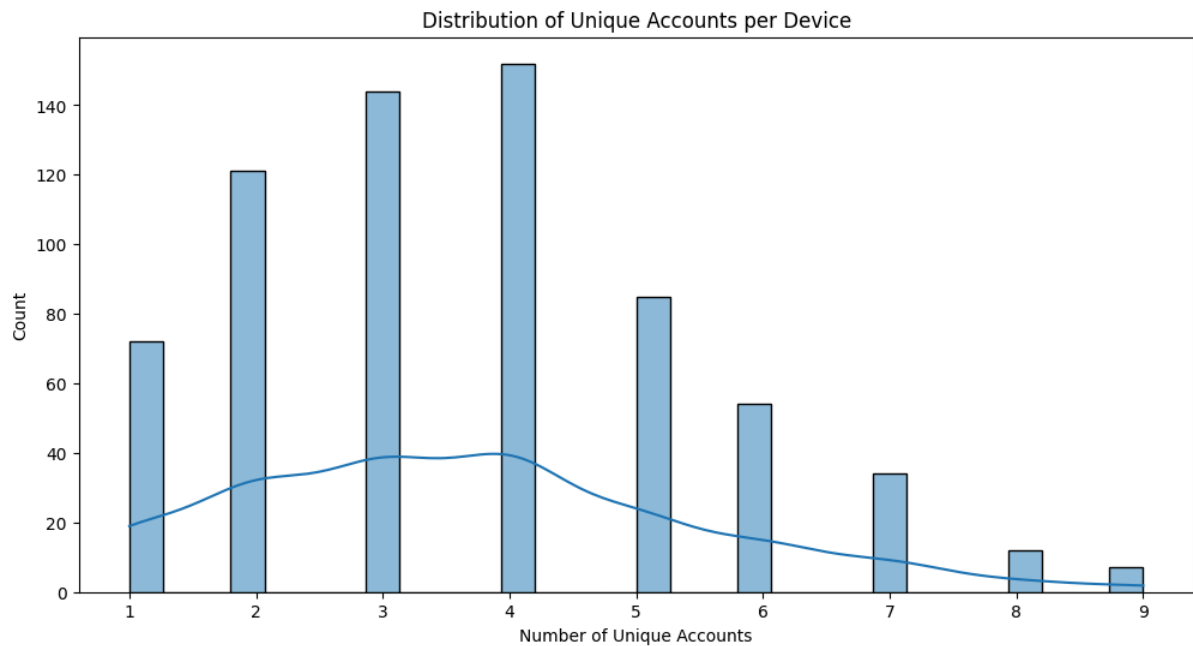


Figure 26: displays distribution of unique accounts per device

Each bar in the histogram represents the number of devices with a specific range of unique accounts. The KDE line provides a smooth estimate of the distribution, highlighting peaks and trends in the number of unique accounts. Most devices are associated with 2 to 4 unique accounts, with a few devices having as many as 9 unique accounts. This visualization helps identify the prevalence of devices being used for multiple accounts, which can be useful for detecting suspicious activity or understanding user behavior patterns.

4.15 Analyze Multiple Accounts Using the Same IP Address

Identify IP addresses associated with multiple accounts. This helps in detecting potential fraudulent activities, as multiple accounts using the same IP address can indicate suspicious behavior.

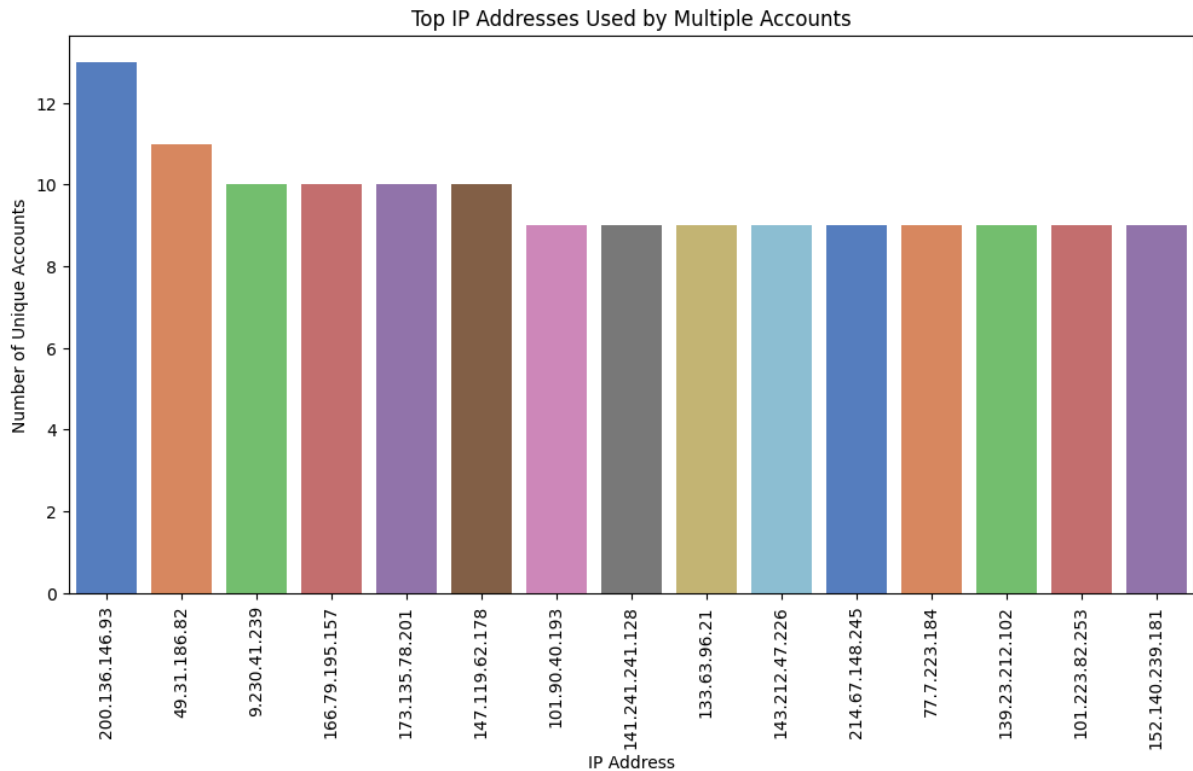


Figure 27: displays the top 15 IP addresses used by multiple unique accounts

The x-axis represents the IP addresses, and the y-axis represents the number of unique accounts associated with each IP address. The IP address "1.198.76.182" has the highest number of unique accounts, with a count of 8. The other IP addresses have between 2 and 3 unique accounts each. This visualization highlights potential cases of shared or compromised IP addresses, which could be important for security and monitoring purposes.

4.16 Analyze Long-Duration Transactions

Identify transactions that have unusually long durations, specifically those in the top 5% of transaction durations. This helps detect potential issues or suspicious activities, as unusually long transaction durations can indicate problems or fraud attempts.

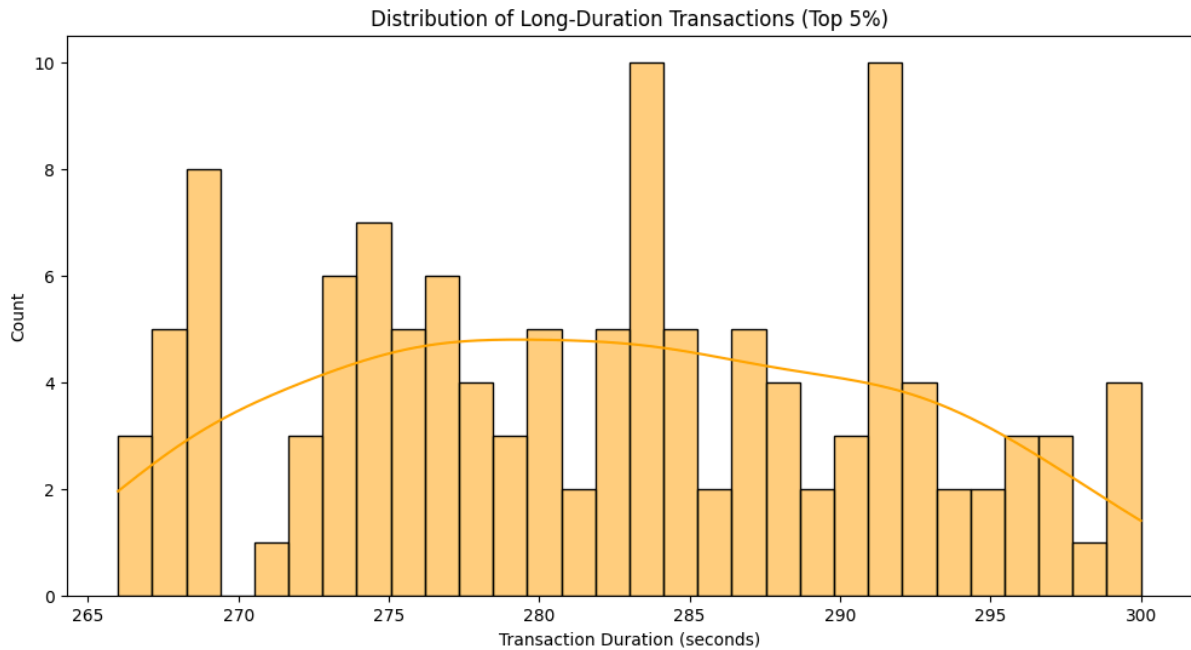


Figure 28: displays the distribution of long-duration transactions

Each bar in the histogram represents the frequency of transactions with durations in a specific range. The x-axis shows the transaction duration in seconds, while the y-axis shows the count of transactions. The KDE line provides a smooth estimate of the distribution's shape, indicating that there are peaks around certain durations. This visualization helps understand the frequency and range of long-duration transactions, which can be important for identifying anomalies or inefficiencies.

IV. Anomaly Detection

1. Identifying Potential Frauds with K-means Clustering

K-means is a popular clustering algorithm that groups data points into k clusters such that the variance within each cluster is minimized. The algorithm works as follows:

- Randomly initialize k centroids.
- Assign each data point to the nearest centroid.
- Update the centroids based on the mean of the data points in each cluster.

- Repeat until centroids stabilize or a predefined number of iterations is reached

Why K-means?

- K-means is computationally efficient, making it suitable for large datasets like transaction records.
- By clustering transactions, patterns emerge that can isolate unusual or potentially fraudulent behavior.
- Clusters and centroids provide clear insights into the dataset's structure.
- It scales well with large datasets, a key requirement for financial fraud detection.

Process:

- Preparing the Data for K-means Clustering, chosen features here are **TransactionAmount** and **TransactionDuration**.

```
features = ['TransactionAmount', 'TransactionDuration']  
X = df[features].copy()
```

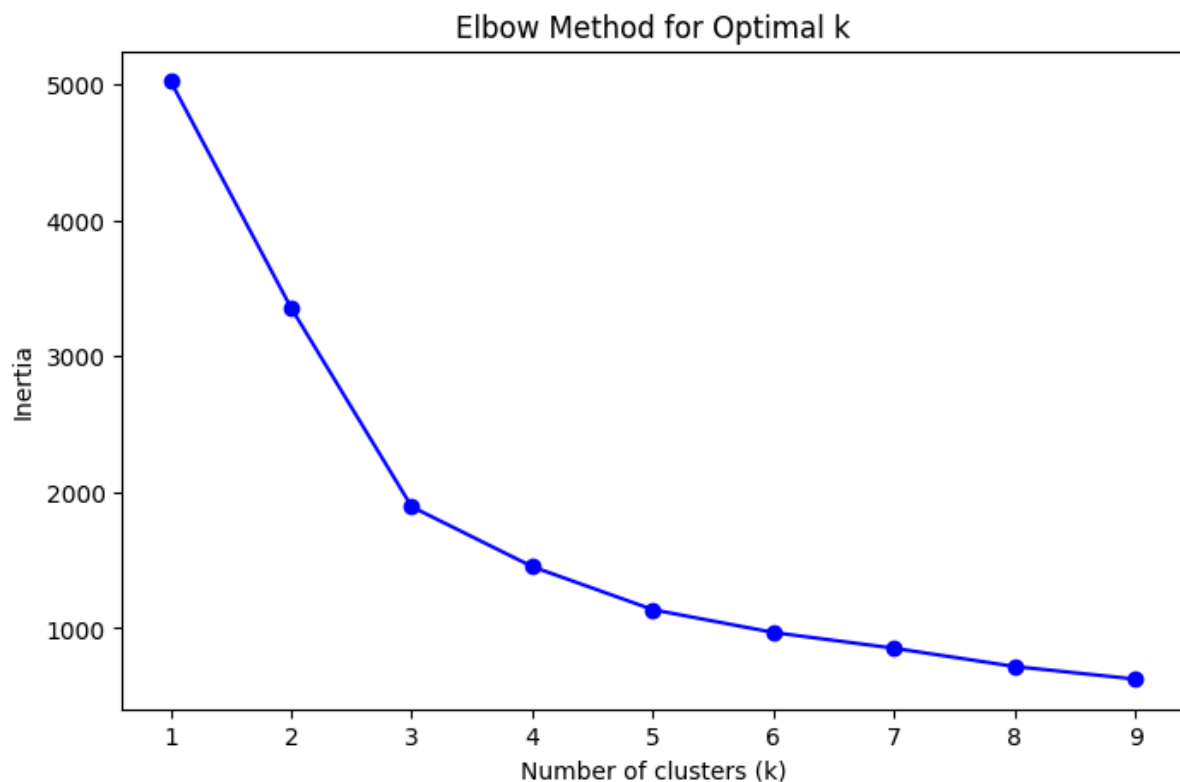
- To ensure equal contribution from all features, data is standardized using **StandardScaler**.

```
scaler = StandardScaler()  
X_scaled = scaler.fit_transform(X)
```

- The Elbow Method was used to identify the optimal number of clusters k by:
 - Running the K-means algorithm for different values of k.
 - Calculating the Within-Cluster-Sum-of-Squares (WCSS) for each value.
 - Plotting WCSS against k to find the "elbow point," where adding more clusters yields diminishing returns.

```
inertia = []
K = range(1, 10) # Test for clusters from 1 to 10
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
```

Through feature selection and standardization, the dataset is prepared into a normalized format appropriate for K-means clustering, ensuring optimal algorithm performance.



The elbow graph analysis revealed $k=3$ as the optimal number of clusters, indicated by the point where additional clusters no longer provide significant reduction in the WCSS.

Running K-means Clustering:

- Applying K-means, using the determined value of **k=3**, the algorithm assigns each data point to a cluster. A new column, **Cluster**, is added to the dataset.

```
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(X_scaled)

df['Cluster'] = kmeans.labels_
df['DistanceToCentroid'] = np.linalg.norm(X_scaled - kmeans.cluster_centers_[kmeans.labels_], axis=1)
```

- Transactions significantly distant from their cluster centroids (e.g., beyond the 95th percentile) are flagged as potential frauds.

```
threshold = df['DistanceToCentroid'].quantile(0.95)
potential_frauds = df[df['DistanceToCentroid'] > threshold]

print(f"Number of potential frauds detected: {len(potential_frauds)}")
```

After running process, the results show Number of potential frauds detected and summary table of the detected fraudulent transactions:

Number of potential frauds detected: 126

	TransactionID	AccountID	TransactionAmount	TransactionDate	TransactionType	Location	DeviceID	IP Address	MerchantID	Channel	...	TransactionDuration
74	TX000075	AC00265	1212.51	2023-10-04 16:36:29	Debit	Indianapolis	D000231	193.83.0.183	M036	Branch	...	24
85	TX000086	AC00098	1340.19	2023-09-29 17:22:10	Credit	Austin	D000574	165.114.224.47	M012	Online	...	30
141	TX000142	AC00114	1049.92	2023-10-23 16:50:33	Debit	Detroit	D000522	121.67.144.20	M052	ATM	...	21
142	TX000143	AC00163	227.14	2023-07-03 17:42:08	Debit	Charlotte	D000439	197.162.55.147	M057	ATM	...	294
146	TX000147	AC00385	973.39	2023-08-30 17:23:20	Debit	Sacramento	D000292	202.194.199.70	M026	Branch	...	296
...
2403	TX002404	AC00111	1493.00	2023-06-07 17:05:41	Debit	Colorado Springs	D000344	136.162.111.135	M096	ATM	...	151
2414	TX002415	AC00028	1664.33	2023-09-25 17:11:19	Debit	San Antonio	D000072	116.106.207.139	M064	Branch	...	65
2439	TX002440	AC00439	538.17	2023-09-26 17:27:17	Credit	Washington	D000430	116.44.12.250	M055	Branch	...	262
2445	TX002446	AC00439	403.01	2023-09-04 17:32:35	Debit	Washington	D000677	223.32.70.156	M029	Online	...	286
2458	TX002459	AC00312	430.83	2023-08-14 16:09:21	Debit	Los Angeles	D000195	87.50.72.69	M079	Branch	...	292

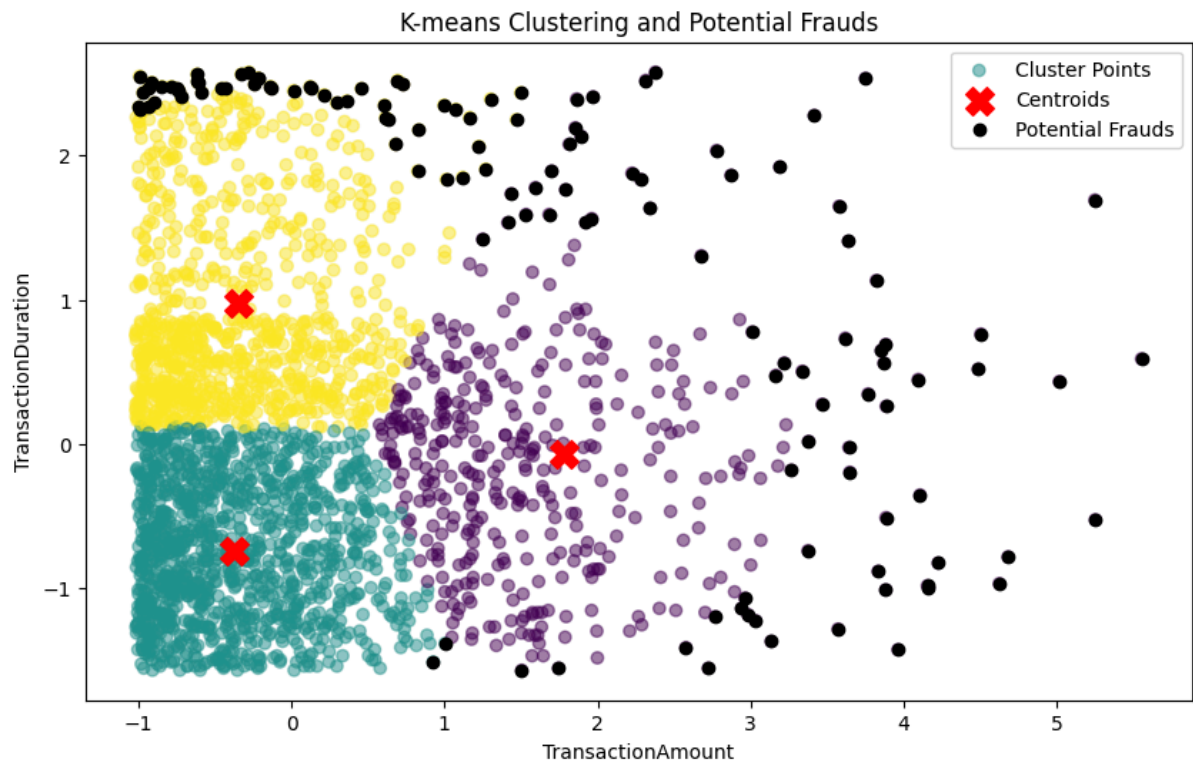
126 rows × 22 columns

LoginAttempts	AccountBalance	PreviousTransactionDate	BalanceChange	TransactionHour	AverageTransactionAmount	UniqueLocationsCount	Cluster	DistanceToCentroid
1	605.95	2024-11-04 08:06:51	-606.56	16	439.288889	9	0	1.876727
1	8654.28	2024-11-04 08:06:53	9994.47	17	642.514000	5	0	2.163279
1	2037.85	2024-11-04 08:10:34	987.93	16	317.322857	7	0	1.562140
1	341.94	2024-11-04 08:11:34	114.80	17	307.955000	2	2	1.521390
1	2042.22	2024-11-04 08:11:44	1068.83	17	591.600000	7	0	2.640557
...
1	1619.16	2024-11-04 08:07:07	126.16	17	438.818571	6	0	2.368753
1	1588.31	2024-11-04 08:07:07	-76.02	17	723.390000	3	0	2.986511
1	5908.04	2024-11-04 08:10:25	6446.21	17	465.090000	7	2	1.490003
3	10513.52	2024-11-04 08:09:32	10110.51	17	465.090000	7	2	1.573122
1	941.49	2024-11-04 08:07:49	510.66	16	489.730000	4	2	1.693355

Visualizing Clusters and Anomalies:

A scatter plot was generated to visualize clusters in two dimensions. Dimensionality reduction was applied using Principal Component Analysis (PCA).

Each data point is plotted with a different color to represent its assigned cluster. Centroids marked with a red "X", while potential frauds, identified as outliers distant from their cluster centroids, are highlighted with distinct black markers.



Scatter plot with clusters and potential frauds highlighted

By applying K-means clustering, we effectively grouped transactions into meaningful clusters and identified potential fraudulent activities through outliers significantly distant from cluster centroids. The Elbow Method ensured a robust choice of the optimal number of clusters, enhancing the reliability of results.

K-means is computationally efficient and interpretable, making it well-suited for fraud detection. However, its sensitivity to initial centroids and the assumption of spherical clusters require careful preprocessing and validation.

In summary, K-means provides a practical approach to analyzing transaction data and detecting anomalies, but understanding its limitations is essential for optimal application.

2. DBSCAN Clustering for Anomaly Detection

DBSCAN is a density-based clustering algorithm that identifies clusters of arbitrary shapes and detects anomalies (noise points) in the dataset. DBSCAN does not require the number of clusters to be pre-specified. Instead, it relies on two main parameters:

- **ϵ (eps):** The maximum distance between two points for them to be considered as part of the same cluster.
- **min_samples:** The minimum number of points required to form a dense region (cluster).

Why DBSCAN?

- Unlike K-means, DBSCAN does not require pre-specifying the number of clusters
- DBSCAN can automatically identify the number of clusters based on data density, making it suitable for datasets with diverse shapes and densities.
- DBSCAN effectively detects noise points—points outside the main clusters indicating potential fraudulent transactions.
- Its capability to handle datasets with varying densities, common in real transaction data, makes DBSCAN a powerful tool for analyzing and detecting anomalous patterns in financial data.

Process:

- Key features such as TransactionAmount, TransactionDuration, AccountBalance, and LoginAttempts were selected

```
features = ['TransactionAmount', 'TransactionDuration', 'AccountBalance', 'LoginAttempts']
x = df[features].copy()
x = x.fillna(x.mean())
```

- Standardizing features using StandardScaler ensures the data is scaled, allowing DBSCAN to operate effectively on features with varying ranges.

```
scaler = StandardScaler()  
x_scaled = scaler.fit_transform(X)
```

Running DBSCAN Clustering:

- DBSCAN is applied with parameters `eps = 1.5` and `min_samples = 5` to detect clusters and anomalies.

```
dbscan = DBSCAN(eps=1.5, min_samples=5)  
dbscan.fit(X_scaled)
```

- Cluster labels are assigned to each data point, and a mapping is applied to provide descriptive names for the clusters

```
df['DBSCAN_Cluster'] = dbscan.labels_  
df['Cluster_Description'] = df['DBSCAN_Cluster'].map(label_mapping)
```

- Points labeled as `-1` by DBSCAN are considered anomalies, representing potential fraudulent transactions.

```
potential_frauds = df[df['DBSCAN_Cluster'] == -1]  
print(f"Number of potential frauds detected by DBSCAN: {len(potential_frauds)}")  
display(potential_frauds.head())
```

Upon completion of the algorithmic processing pipeline, the resulting output demonstrates the following findings:

Number of potential frauds detected by DBSCAN: 17

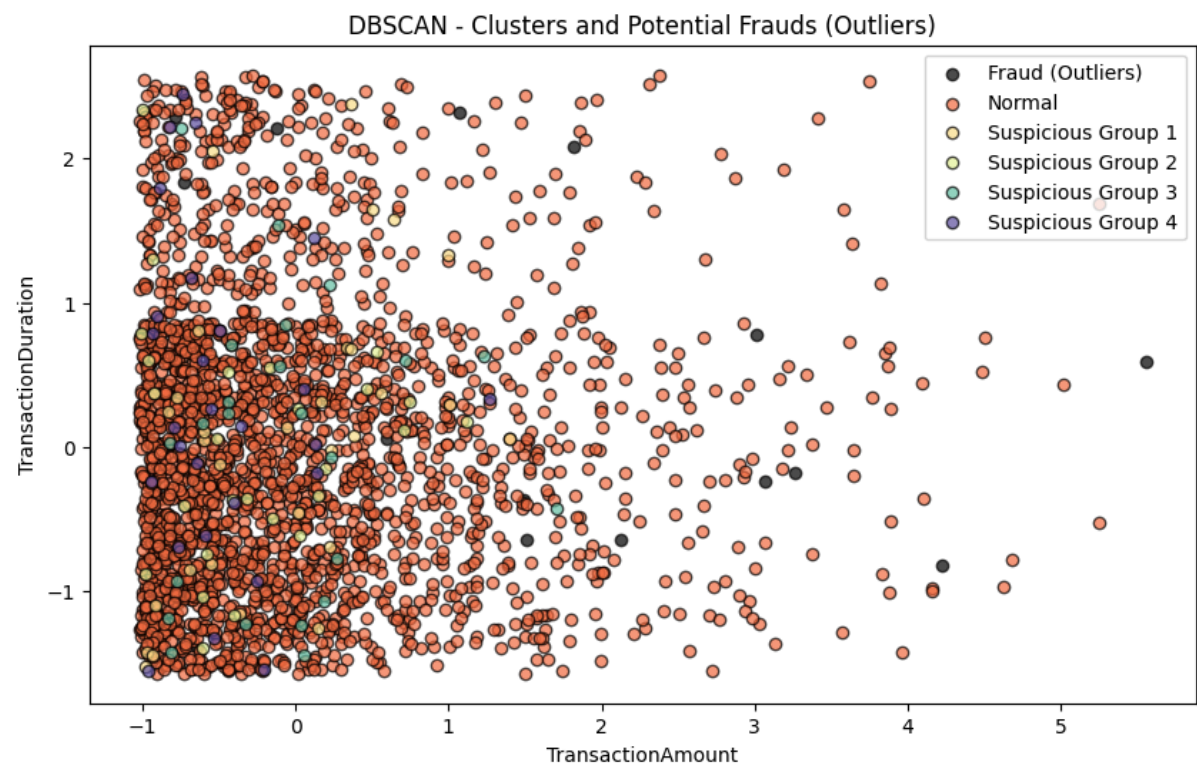
	TransactionID	AccountID	TransactionAmount	TransactionDate	TransactionType	Location	DeviceID	IP Address	MerchantID	Channel	...	AccountBalance	PreviousTransacti
274	TX000275	AC00454	1176.28	2023-12-20 16:08:02	Credit	Kansas City	D000476	50.202.8.53	M074	ATM	...	323.69	2024-11-04 01
454	TX000455	AC00264	611.11	2023-10-18 18:32:31	Debit	Detroit	D000215	141.201.46.191	M045	ATM	...	919.84	2024-11-04 01
653	TX000654	AC00423	1919.11	2023-06-27 17:48:25	Debit	Portland	D000191	207.157.126.125	M033	ATM	...	11090.24	2024-11-04 01
693	TX000694	AC00011	733.29	2023-03-15 18:42:16	Debit	Virginia Beach	D000618	16.51.235.240	M032	ATM	...	10427.00	2024-11-04 01
754	TX000755	AC00153	84.34	2023-06-08 16:27:56	Debit	Memphis	D000493	200.136.146.93	M039	Online	...	5313.97	2024-11-04 01

5 rows × 24 columns

PreviousTransactionDate	BalanceChange	TransactionHour	AverageTransactionAmount	UniqueLocationsCount	Cluster	DistanceToCentroid	DBSCAN_cluster	Cluster_Description
2024-11-04 08:11:44	1499.97	16	494.182500	4	4	1.488559	-1	Fraud (Outliers)
2024-11-04 08:11:12	308.73	18	441.255000	6	2	1.969599	-1	Fraud (Outliers)
2024-11-04 08:10:27	9171.13	17	506.356667	5	0	3.828962	-1	Fraud (Outliers)
2024-11-04 08:07:12	9693.71	18	348.322857	5	0	0.418493	-1	Fraud (Outliers)
2024-11-04 08:09:31	5229.63	16	350.270000	7	2	0.940290	-1	Fraud (Outliers)

Visualize clusters and potential frauds

The visualization below represents the results of the DBSCAN clustering applied to the dataset. It highlights both the identified clusters and the outliers (potential frauds) detected by the algorithm.



- The clusters are represented by data points with distinct colors. Each color corresponds to a specific cluster identified by DBSCAN.
- The black points represent outliers labeled as -1 by DBSCAN. These are potential fraudulent transactions or anomalies in the dataset.
- The x-axis corresponds to TransactionAmount (normalized values).
- The y-axis corresponds to TransactionDuration (normalized values).
- Fraud (Outliers): Represented by black points, these are data points that DBSCAN could not assign to any cluster, indicating their unusual nature.
- The scattered black points indicate anomalies that deviate significantly from the rest of the dataset, potentially signaling fraudulent activities

By applying DBSCAN clustering, we effectively identified potential fraudulent activities through anomalies in the transaction dataset. This method allowed us to group transactions into meaningful clusters based on density, and it accurately detected noise points as outliers.

DBSCAN's ability to find arbitrary shaped clusters made it particularly effective for detecting anomalies without requiring a predefined number of clusters. This flexibility is crucial in real-world scenarios where transaction patterns may not conform to spherical shapes. However, DBSCAN's reliance on parameters such as `eps` and `min_samples` demands careful tuning and validation to ensure accurate anomaly detection.

In summary, DBSCAN provided a valuable tool for detecting fraudulent activities in transaction data by leveraging its ability to identify noise points as anomalies. While computationally efficient and powerful for clustering complex data, understanding its limitations particularly with parameter sensitivity ensures optimal application in practical fraud detection scenarios.

3. Isolation Forest for Anomaly Detection

Process:

- **Outlier_mapping:** This code creates a dictionary (outlier_mapping) to map the output values of the Isolation Forest model. A value of 1 (not an outlier) is mapped to 'Normal', and a value of -1 (outlier, potential fraud) is mapped to 'Potential Fraud'.

```
outlier_mapping = {1: 'Normal', -1: 'Potential Fraud'}
```

- **Features:** Select the important features for fraud detection, including transaction amount (TransactionAmount), transaction duration (TransactionDuration), account balance (AccountBalance), and the number of failed login attempts (LoginAttempts).
- **X:** Creates a copy of the selected columns from the DataFrame **df** for use in training the model.

```
features = ['TransactionAmount', 'TransactionDuration', 'AccountBalance', 'LoginAttempts']  
X = df[features].copy()
```

- **X.fillna(X.mean()):** This line replaces any missing values (NaN) in the dataset with the mean value of each column. This ensures that there are no missing values when processing with machine learning models, which usually do not support missing values.

```
X = X.fillna(X.mean())
```

- **StandardScaler:** Creates a StandardScaler object and applies it to standardize the data **X**. This ensures that the features have the same scale (mean of 0 and standard deviation of 1). This step is important because models might be biased towards features with larger scales.

```
scaler = StandardScaler()  
X_scaled = scaler.fit_transform(X)
```

- **Isolation Forest:** Uses the Isolation Forest model to detect outliers (anomalies).
 - `n_estimators = 100`: The model uses 100 decision trees.
 - `contamination = 0.05`: Assumes that 5% of the data is anomalous (fraudulent).
 - `random_state = 42`: Ensures reproducibility of the results.
- **Fit:** Trains the model with the standardized data.

```
iso_forest = IsolationForest(n_estimators=100, contamination=0.05, random_state=42)
iso_forest.fit(X_scaled)
```

- **Decision_function:** Calculates the anomaly score for each sample. This score indicates how anomalous each data point is.
- **predict:** Predicts anomalies in the data. A value of **1** means normal data, and **-1** means anomalous (potential fraud).

```
df['AnomalyScore'] = iso_forest.decision_function(X_scaled)
df['IsAnomaly'] = iso_forest.predict(X_scaled)
```

- **AnomalyLabel:** Maps the **1** and **-1** values to descriptive labels 'Normal' and 'Potential Fraud' for easier interpretation.

```
df['AnomalyLabel'] = df['IsAnomaly'].map(outlier_mapping)
```

- **potential_frauds:** Filters the rows where **IsAnomaly** is **-1**, i.e., the samples predicted as anomalous (potential fraud).
- **display (potential_frauds.head()):** Displays the first 5 rows of the potential fraud transactions.

```
potential_frauds = df[df['IsAnomaly'] == -1]
print(f"Number of potential frauds detected: {len(potential_frauds)}")
display(potential_frauds.head())
```

Output of the process:

Number of potential frauds detected: 126

	TransactionID	AccountID	TransactionAmount	TransactionDate	TransactionType	Location	DeviceID	IP Address	MerchantID	Channel	...	TransactionHour
26	TX000027	AC00441	246.93	2023-04-17 16:37:01	Debit	Miami	D000046	55.154.161.250	M029	ATM	...	16
32	TX000033	AC00060	396.45	2023-09-25 16:26:00	Debit	New York	D000621	133.67.250.163	M007	ATM	...	16
85	TX000086	AC00098	1340.19	2023-09-29 17:22:10	Credit	Austin	D000574	165.114.224.47	M012	Online	...	17
91	TX000092	AC00310	223.85	2023-10-02 16:36:10	Debit	Kansas City	D000481	133.223.159.151	M009	ATM	...	16
146	TX000147	AC00385	973.39	2023-08-30 17:23:20	Debit	Sacramento	D000292	202.194.199.70	M026	Branch	...	17

5 rows x 27 columns

AverageTransactionAmount	UniqueLocationsCount	Cluster	DistanceToCentroid	DBSCAN_Cluster	Cluster_Description	AnomalyScore	IsAnomaly	AnomalyLabel
317.948333	6	2	0.461840	2	Suspicious Group 2	-0.075150	-1	Potential Fraud
388.180000	4	2	0.744993	1	Suspicious Group 1	-0.014181	-1	Potential Fraud
642.514000	5	0	2.163279	0	Normal	-0.018206	-1	Potential Fraud
439.977143	7	1	0.221557	4	Suspicious Group 4	-0.029888	-1	Potential Fraud
591.600000	7	0	2.640557	0	Normal	-0.016904	-1	Potential Fraud

Data Visualization

- **colors**: Creates an array of colors (**r** for red and **b** for blue) based on the value of **IsAnomaly** (red for fraud and blue for normal).

```
colors = np.where(df['IsAnomaly'] == -1, 'r', 'b')
```

- **plt.scatter**: Plots a scatter plot where the x-axis represents **TransactionAmount** and the y-axis represents **AccountBalance** (both standardized). The data points are colored based on whether they are normal or anomalous.

```
# Visualize potential frauds (TransactionAmount vs AccountBalance)
plt.figure(figsize=(10, 6))
plt.scatter(
    X_scaled[:, 0],
    X_scaled[:, 1],
    c=colors,
    cmap='coolwarm',
    alpha=0.7,
    edgecolors='k',
    label='Data Points'
)
```

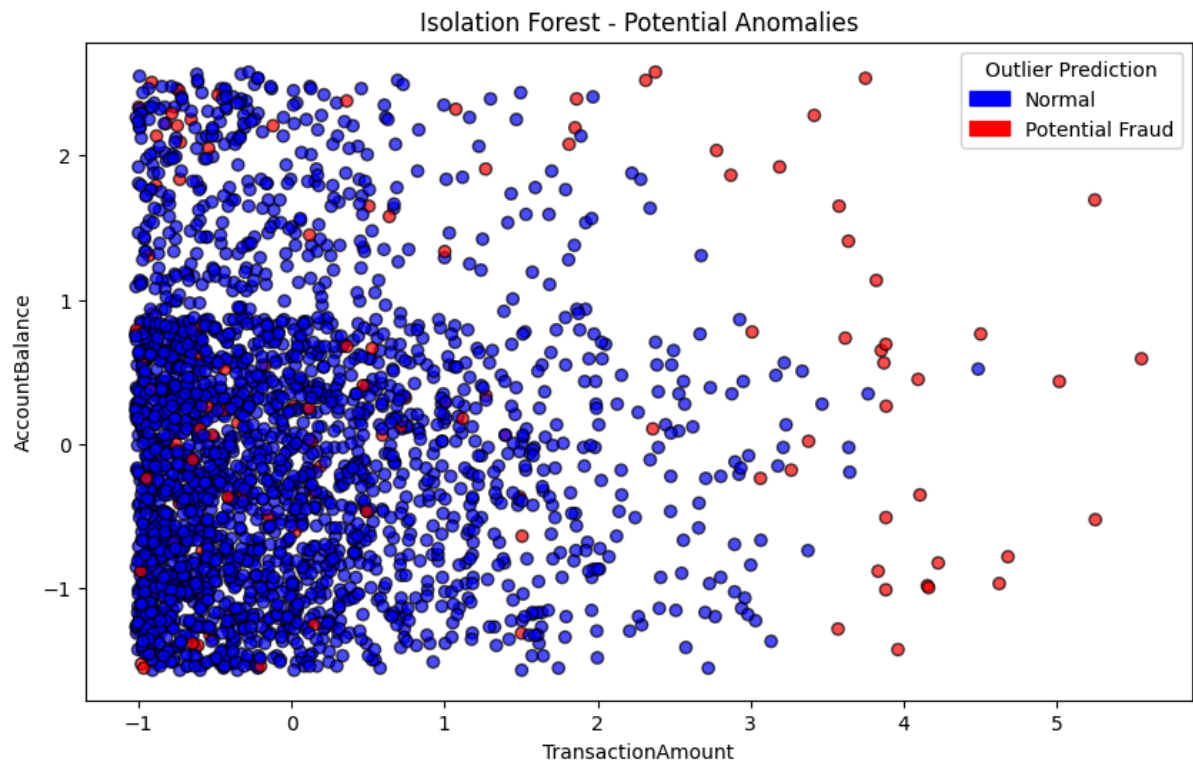
- **Custom legend:** Creates and adds a legend to the plot, so that viewers can easily distinguish between 'Normal' and 'Potential Fraud' points.

```
import matplotlib.patches as mpatches
normal_patch = mpatches.Patch(color='b', label='Normal')
fraud_patch = mpatches.Patch(color='r', label='Potential Fraud')
plt.legend(handles=[normal_patch, fraud_patch], title='Outlier Prediction')
```

- **plt.title, x-label, y-label:** Adds a title and labels for the x and y axis to make the plot more informative.

```
plt.title('Isolation Forest - Potential Anomalies')
plt.xlabel(features[0]) # TransactionAmount
plt.ylabel(features[2]) # AccountBalance
plt.show()
```

Output of the data visualization:



V. Conclusion

The project successfully explored clustering and anomaly detection techniques to identify potential fraudulent activities. Key takeaways include:

- **K-means Clustering:** Effective in grouping data but sensitive to the initial number of clusters.
- **DBSCAN:** Robust in identifying noise but requires careful parameter tuning to balance sensitivity and specificity.
- **Isolation Forest:** A powerful unsupervised method to flag anomalies based on feature isolation, particularly effective with high-dimensional data.

These methods demonstrated complementary strengths in detecting anomalies within the dataset, providing valuable insights for fraud detection efforts.

VI. Reference

 [Fraud Detection: Clustering & Anomaly Analysis](#)