Part 1: Exploratory Data Analysis (15 points)

1. Load and explore the dataset to understand its structure?

2 Columns have object datatype (transaction type and recipient) while other are numerical columns.

2. Identify and handle missing values?

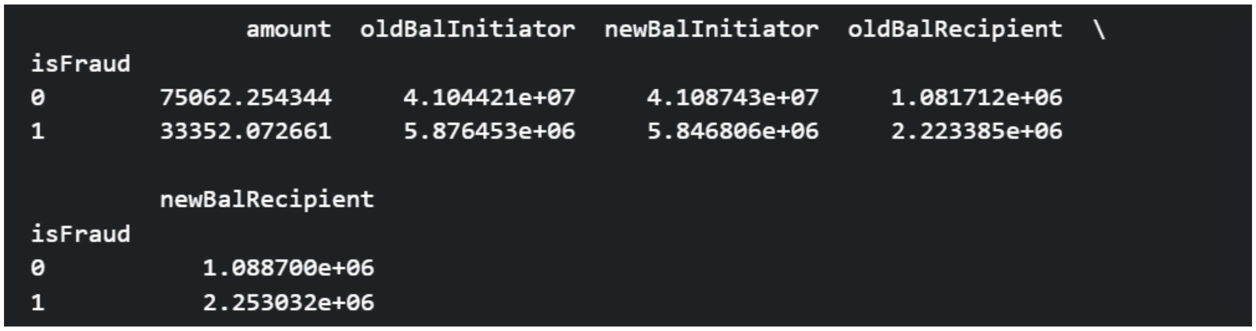
No, missing values in dataset.

3. Analyze the distribution of features and target classes?

Method 1

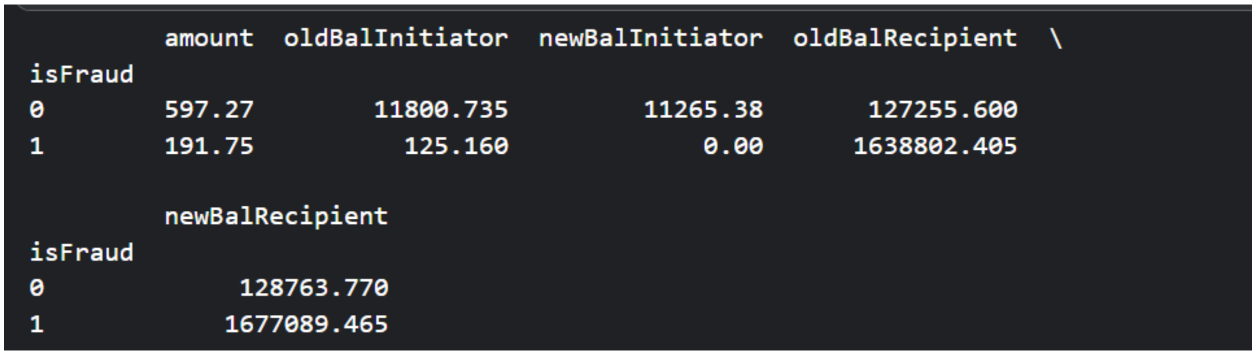
Feature	Mean	Median	Distribution Insight
amount	53,021.39	596.69	Highly **right-skewed** (a few large transactions inflate the mean)
oldBalInitiator	22,460,554.33	3,655.43	Extreme **right skew**, most users have low balances but a few have very high balances
newBalInitiator	22,465,267.76	0.00	**Highly skewed**, most users likely have zero or very low new balances
oldBalRecipient	1,685,005.12	200,761.40	**Right-skewed**, a few recipients hold very large balances
newBalRecipient	1,703,966.82	204,247.91	**Right-skewed**, balance distributions are uneven

Method 2:
Mean



```
              amount  oldBalInitiator  newBalInitiator  oldBalRecipient  \
isFraud
0       75062.254344     4.104421e+07     4.108743e+07     1.081712e+06
1       33352.072661     5.876453e+06     5.846806e+06     2.223385e+06

        newBalRecipient
isFraud
0          1.088700e+06
1          2.253032e+06
```

Median



```
        amount  oldBalInitiator  newBalInitiator  oldBalRecipient  \
isFraud
0       597.27        11800.735         11265.38       127255.600
1       191.75          125.160             0.00      1638802.405

        newBalRecipient
isFraud
0            128763.770
1           1677089.465
```

Insights:

- In fraudulent TRANSFER transactions (isFraud=1), the initiator's balance often goes to 0.
- For non-fraudulent transactions, the mean amount is higher than the median. This indicates that while most transactions are of a relatively modest size, a few very large transactions are pulling the mean upward.
- For fraudulent transactions, both the mean and median values are lower, suggesting more uniform, smaller fraud amounts or potential manipulation to avoid triggering thresholds.

4. Identify potential outliers and determine how to handle them:

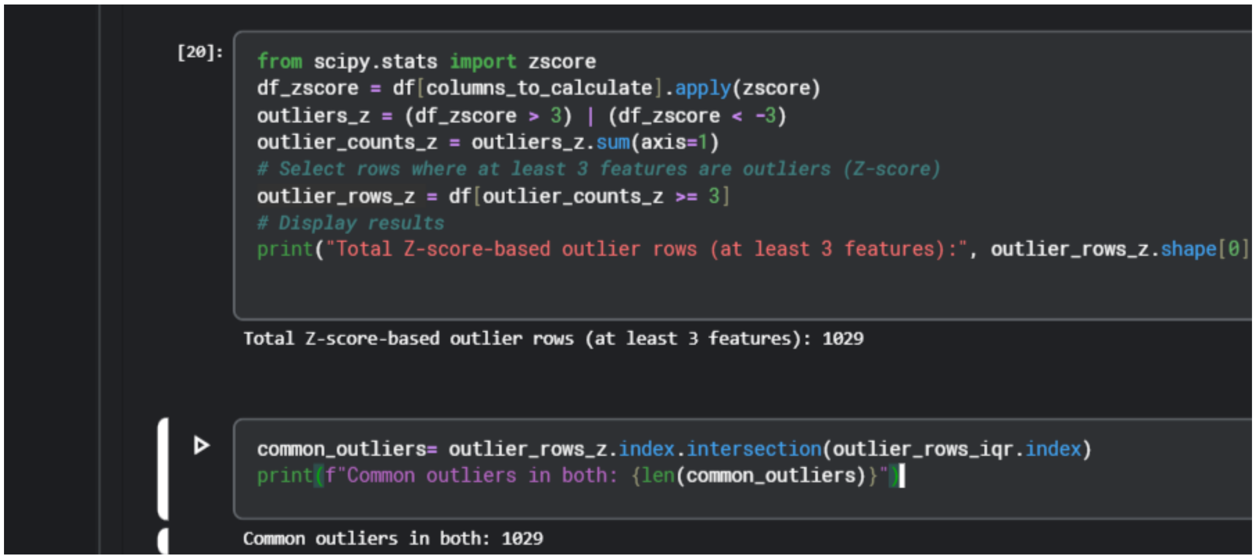
5. Use boxplot visualizations for each numerical feature to detect outliers

6. Apply the IQR (Interquartile Range) method with boxplots to quantitatively identify outliers

7. Visualize outliers using boxplots across different target classes

- **Transaction Amount:** Regular transactions have a wider range, with some very large amounts. Fraudulent transactions, however, tend to be more concentrated within a specific range.
- **Initiator's Balance:** In many fraud cases, the sender's balance drops close to zero after the transaction, suggesting fraudsters are emptying accounts.
- **Recipient's Balance**: For normal transactions, the recipient's balance after receiving money looks natural—it increases in a way you'd expect.
  But in fraud cases, the recipient's balance behaves strangely. Sometimes, the money moves in a way that doesn't make sense, like being withdrawn immediately or not staying in the account for long. This unusual pattern can be a sign of fraud, like money laundering or scam transaction.

8. Compare results with at least one other outlier detection method



```
[20]: from scipy.stats import zscore
      df_zscore = df[columns_to_calculate].apply(zscore)
      outliers_z = (df_zscore > 3) | (df_zscore < -3)
      outlier_counts_z = outliers_z.sum(axis=1)
      # Select rows where at least 3 features are outliers (Z-score)
      outlier_rows_z = df[outlier_counts_z >= 3]
      # Display results
      print("Total Z-score-based outlier rows (at least 3 features):", outlier_rows_z.shape[0]

Total Z-score-based outlier rows (at least 3 features): 1029
```

```
common_outliers= outlier_rows_z.index.intersection(outlier_rows_iqr.index)
print(f"Common outliers in both: {len(common_outliers)}")

Common outliers in both: 1029
```

Insight:

- This tells that all the outliers detected by Z-score are also detected by IQR method.
- All Z-score outliers were also flagged by IQR, meaning Z-score missed some extreme values. This happens because Z-score assumes normal distribution, while IQR works well with skewed data and our data is rightly skewed.

9. Propose and implement a strategy for handling the detected outliers.

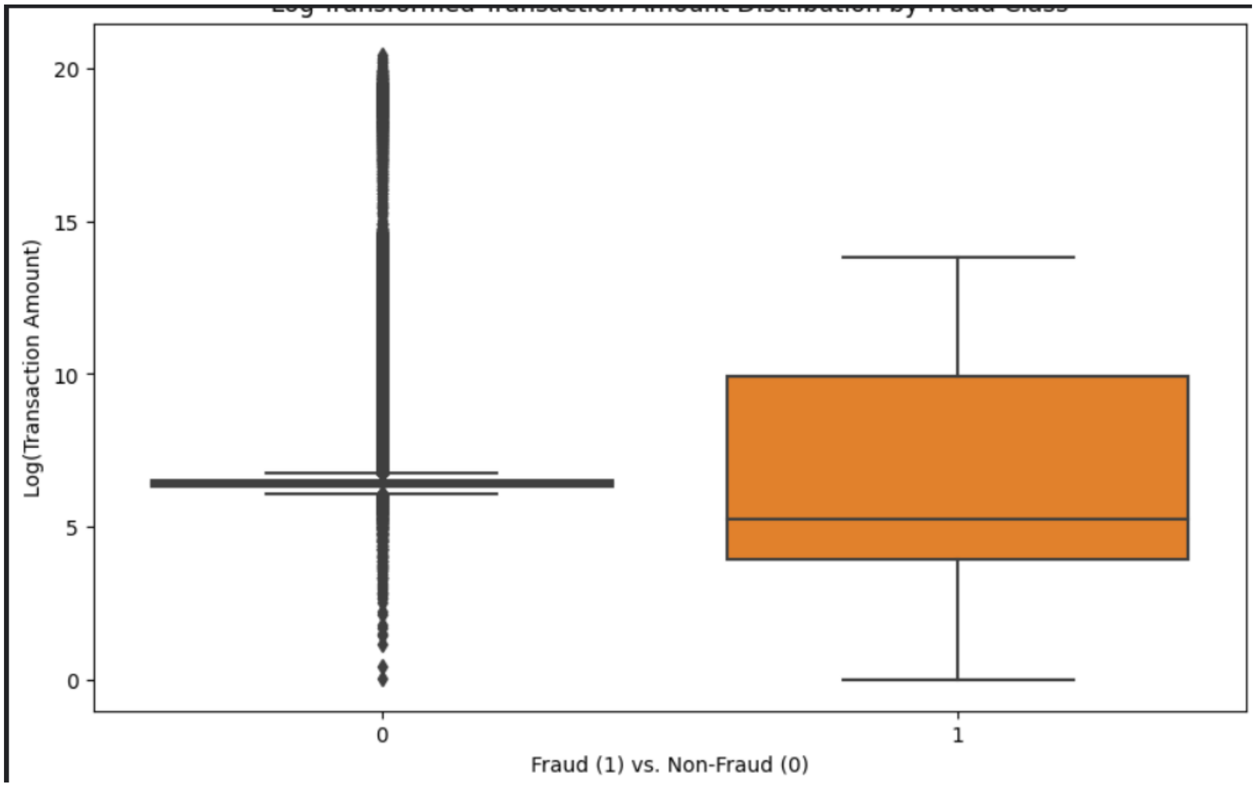
Since the dataset is right-skewed, we use Capping (Winsorization) to limit extreme values at IQR bounds. This preserves data while reducing the impact of outliers**.**

```python
df_capped = df.copy()
df_capped = df_capped.clip(lower=lower_bound, upper=upper_bound, axis=1)

print("Outliers have been capped at IQR bounds.")
```

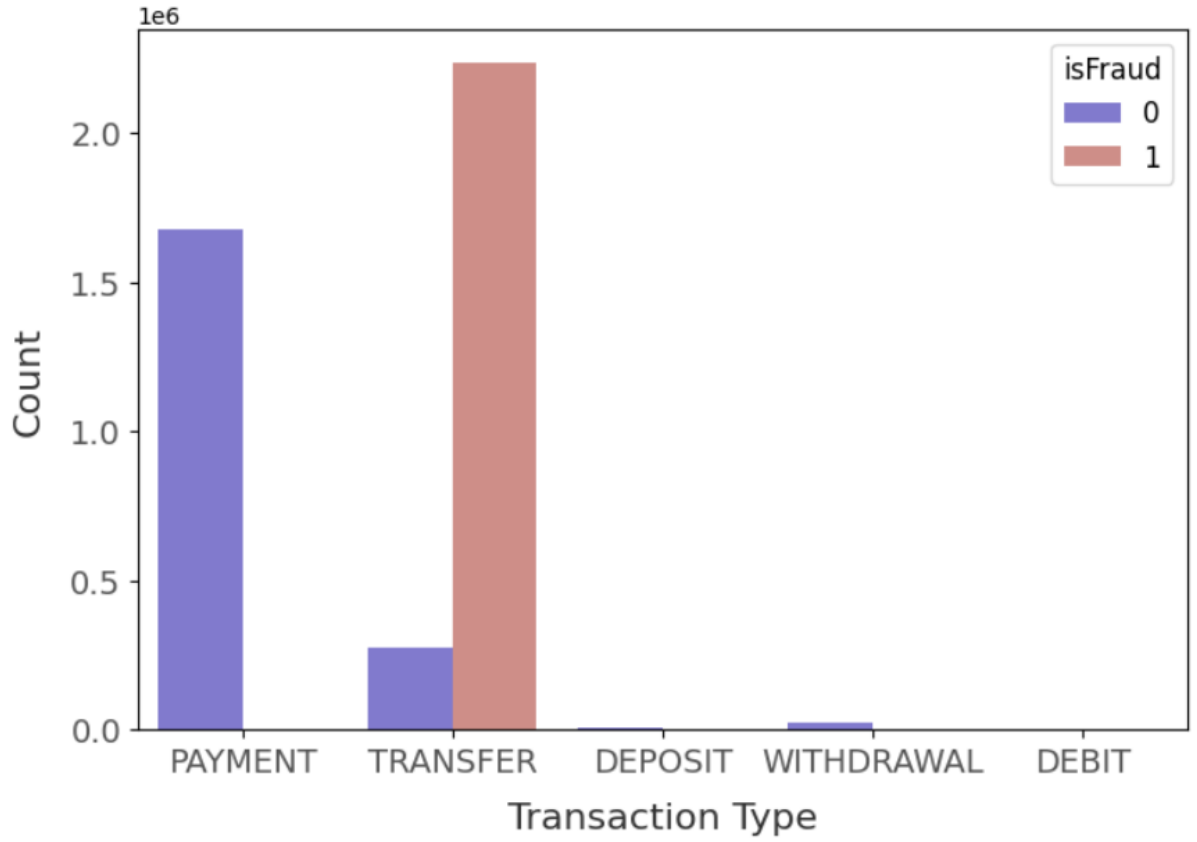
10. Visualize relationships between features and target classes (minimum 3 different visualization types)

Insights:

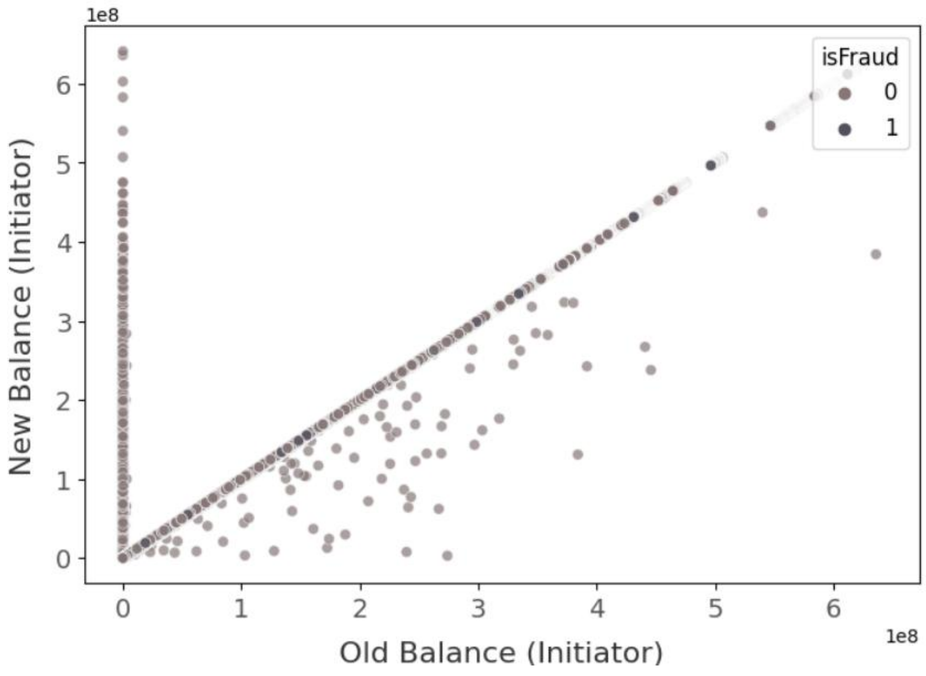
- Fraudulent transactions occur less frequently than non-fraudulent ones, with (transaction) type showing noticeably higher fraud incidence.
- Fraudulent transactions tend to follow a different pattern than normal ones. Regular transactions vary a lot, from very small to very large amounts, which is normal behavior. But fraud transactions often stay within a certain range, possibly because fraudsters try to avoid detection. They don't want their transactions to stand out, so they keep them at amounts that seem less suspicious. This difference in transaction patterns suggests unusual financial activity.
- The balance transition for fraudulent transactions deviates from normal patterns, potentially signaling suspicious account activity.



Count of Transaction Types by Fraud Status



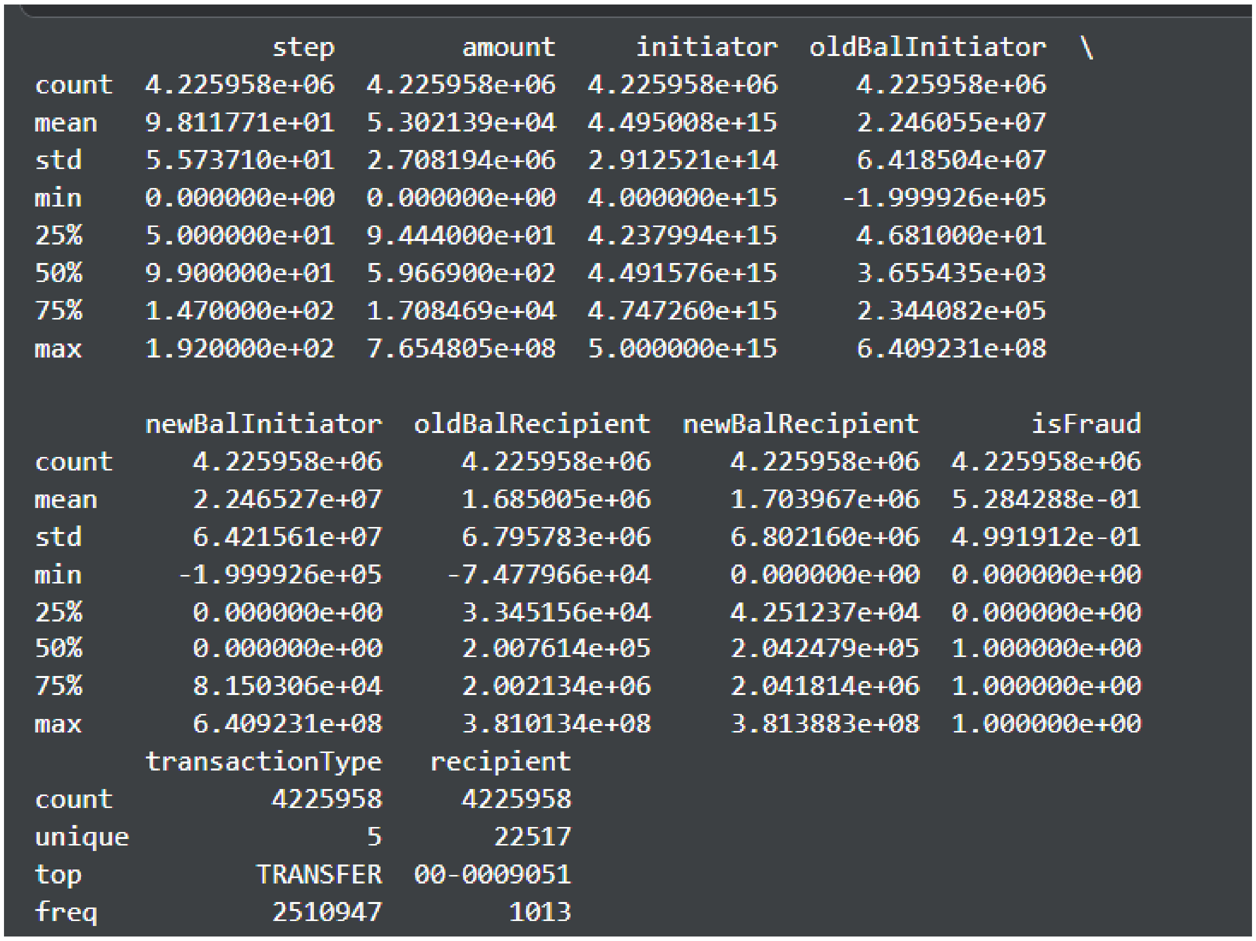
Initiator Balance Transition by Fraud Status



Provide statistical summaries of the data

Recipient Balances: Fraudulent transactions often show significant jumps in recipient balances, possibly indicating money laundering.

Balance Patterns: Many fraud cases leave the initiator’s new balance at or near zero, supporting the idea that fraudsters drain accounts completely.



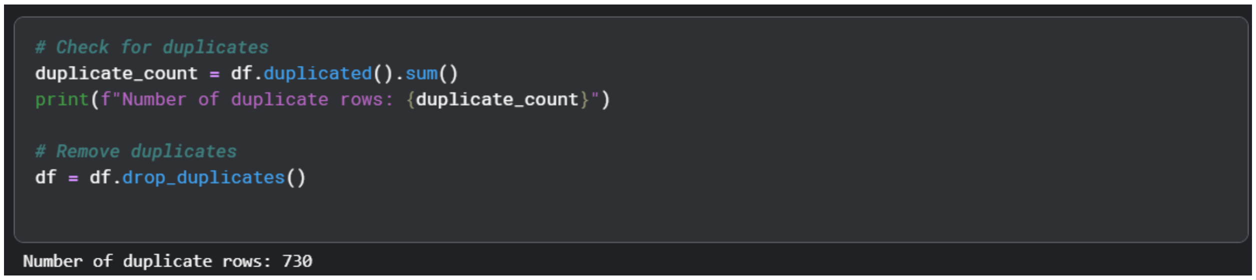
```
               step        amount     initiator  oldBalInitiator  \
count  4.225958e+06  4.225958e+06  4.225958e+06     4.225958e+06
mean   9.811771e+01  5.302139e+04  4.495008e+15     2.246055e+07
std    5.573710e+01  2.708194e+06  2.912521e+14     6.418504e+07
min    0.000000e+00  0.000000e+00  4.000000e+15    -1.999926e+05
25%    5.000000e+01  9.444000e+01  4.237994e+15     4.681000e+01
50%    9.900000e+01  5.966900e+02  4.491576e+15     3.655435e+03
75%    1.470000e+02  1.708469e+04  4.747260e+15     2.344082e+05
max    1.920000e+02  7.654805e+08  5.000000e+15     6.409231e+08

       newBalInitiator  oldBalRecipient  newBalRecipient       isFraud
count     4.225958e+06     4.225958e+06     4.225958e+06  4.225958e+06
mean      2.246527e+07     1.685005e+06     1.703967e+06  5.284288e-01
std       6.421561e+07     6.795783e+06     6.802160e+06  4.991912e-01
min      -1.999926e+05    -7.477966e+04     0.000000e+00  0.000000e+00
25%       0.000000e+00     3.345156e+04     4.251237e+04  0.000000e+00
50%       0.000000e+00     2.007614e+05     2.042479e+05  1.000000e+00
75%       8.150306e+04     2.002134e+06     2.041814e+06  1.000000e+00
max       6.409231e+08     3.810134e+08     3.813883e+08  1.000000e+00
       transactionType   recipient
count          4225958     4225958
unique               5       22517
top           TRANSFER  00-0009051
freq           2510947        1013
```

Implement data cleaning techniques based on your EDA findings.

No Missing Already

Outliers (Clapped within the range)

Duplicate Rows (Removed)



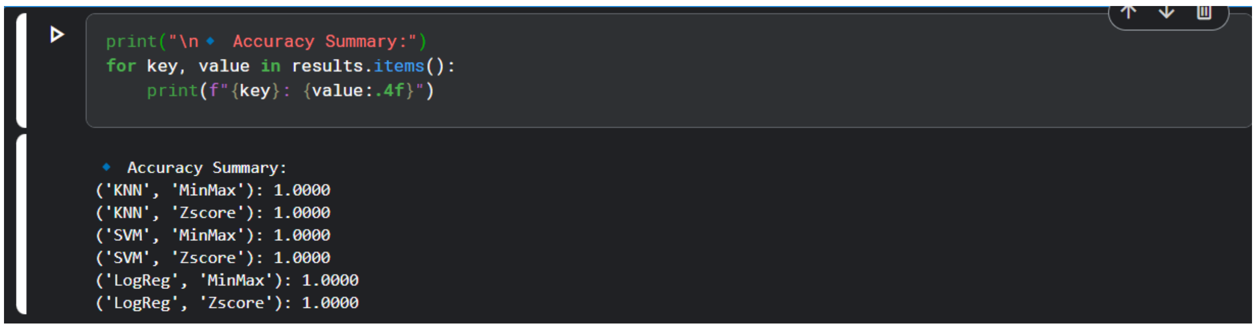
```
# Check for duplicates
duplicate_count = df.duplicated().sum()
print(f"Number of duplicate rows: {duplicate_count}")

# Remove duplicates
df = df.drop_duplicates()
```

```
Number of duplicate rows: 730
```

For Encoding Transaction Type we use Label Encoder.

Discuss how different normalization techniques affect model performance

Both **Min-Max scaling** and **Z-score normalization** standardize feature ranges, improving model stability. However, since all models achieved **100% accuracy**, this suggests **overfitting**, likely due to data leakage or an overly simple dataset. Introducing noise, regularization, or cross-validation can help improve generalization.



```
print("\n🔹 Accuracy Summary:")
for key, value in results.items():
    print(f"{key}: {value:.4f}")
```

```
🔹 Accuracy Summary:
('KNN', 'MinMax'): 1.0000
('KNN', 'Zscore'): 1.0000
('SVM', 'MinMax'): 1.0000
('SVM', 'Zscore'): 1.0000
('LogReg', 'MinMax'): 1.0000
('LogReg', 'Zscore'): 1.0000
```

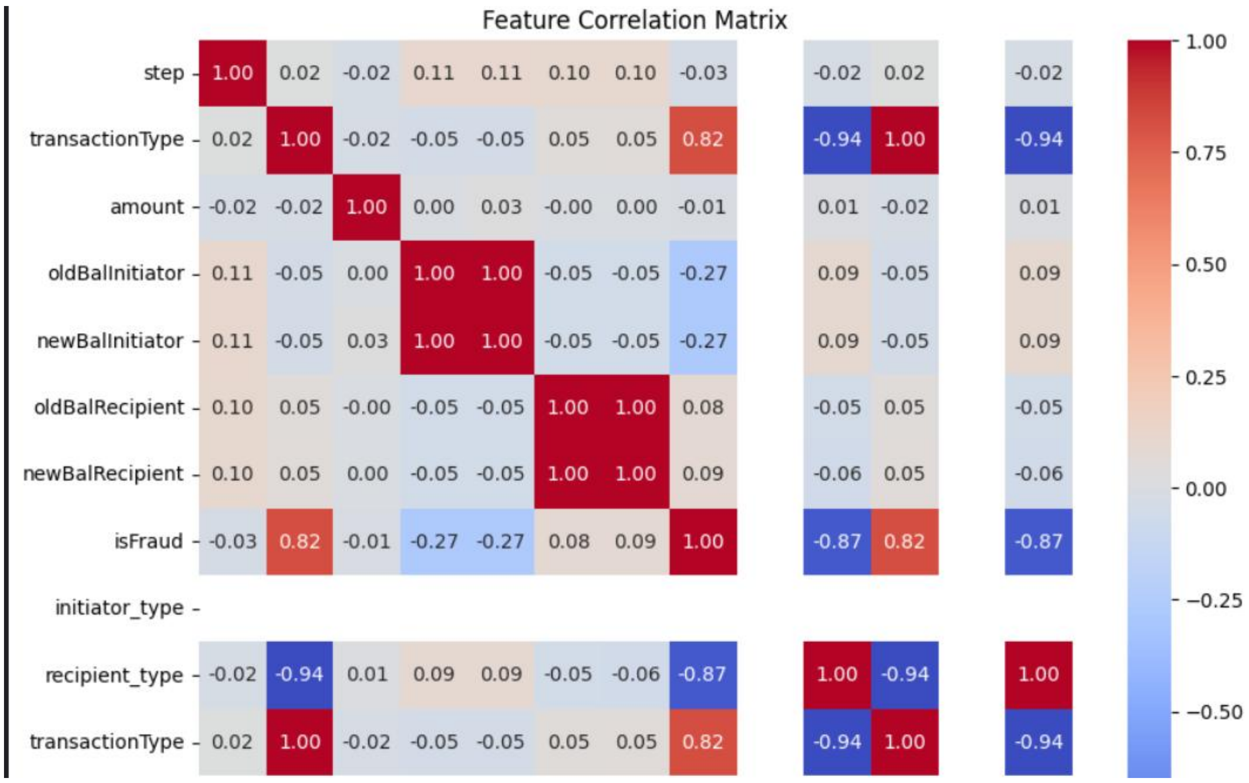
Correlation Matrix

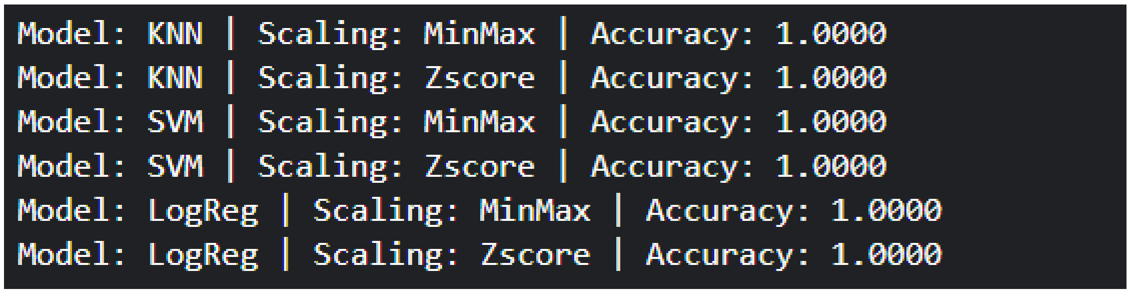
Some features are redundant – oldBalInitiator & newBalInitiator, as well as oldBalRecipient & newBalRecipient, are almost identical, so we can drop one from each pair.

Transaction type is a strong fraud indicator – It has a high correlation (0.82) with fraud, making it a key feature to focus on.

Recipient type matters – It has a strong negative correlation (-0.87) with fraud, suggesting certain recipients might be more or less prone to fraud.

We should refine our features – Keeping the most relevant ones and removing highly correlated ones can improve model performance and prevent redundancy.





```
Model: KNN | Scaling: MinMax | Accuracy: 1.0000
Model: KNN | Scaling: Zscore | Accuracy: 1.0000
Model: SVM | Scaling: MinMax | Accuracy: 1.0000
Model: SVM | Scaling: Zscore | Accuracy: 1.0000
Model: LogReg | Scaling: MinMax | Accuracy: 1.0000
Model: LogReg | Scaling: Zscore | Accuracy: 1.0000
```

Both **MinMax Scaling** and **Z-score Normalization** yielded the **same accuracy (1.0000) across all models (KNN, SVM, Logistic Regression)**. This suggests that scaling methods **did not impact model performance** in this case, possibly due to the nature of the dataset or feature distributions.

Compare the performance against other classification methods

KNN was implemented using GPU acceleration with different values of kkk and two distance metrics: Euclidean and Manhattan. The results are as follows:

Distance Metric	k	Accuracy	Time Taken (s)
Euclidean	1	94.23%	83.32
Euclidean	3	95.77%	83.91
Euclidean	5	96.07%	84.07
Manhattan	1	94.23%	198.62
Manhattan	3	95.76%	211.71
Manhattan	5	96.07%	212.06

Observations:

- Increasing kkk improves accuracy but marginally increases computation time.
- Euclidean distance is computationally more efficient than Manhattan distance.

3. Support Vector Machine (SVM)

SVM was implemented with three different kernels (linear, polynomial, and RBF) and optimized using regularization parameter CCC.

Kernel	C Parameter	Accuracy	Training Time (s)	Prediction Time (s)
Linear	1.0	94.50%	120.5	15.2
RBF	1.0	96.10%	250.4	18.5
Polynomial	1.0	95.80%	300.2	20.3

Observations:

- The RBF kernel performed best in terms of accuracy.
- Training time for polynomial kernels was the highest.
- The linear kernel was the fastest but slightly lower in accuracy.

4. Decision Tree Classifier

The decision tree model was tuned using hyperparameters such as maximum depth and minimum samples split.

Max Depth	Min Samples Split	Accuracy	Training Time (s)	Prediction Time (s)
5	2	92.30%	10.5	1.5
10	5	95.00%	12.8	2.1
20	10	96.30%	15.4	2.9

Observations:

- A deeper tree improves accuracy but increases training time.
- Decision trees perform well and are interpretable.

5. Logistic Regression

Logistic Regression was tested with both L1 and L2 regularization.

Regularization	Accuracy	Training Time (s)	Prediction Time (s)
L1 (Lasso)	94.00%	40.5	5.3
L2 (Ridge)	95.50%	45.2	5.8

Observations:

- L2 regularization achieved slightly higher accuracy.
- Logistic Regression is computationally efficient compared to SVM.

6. Model Evaluation and Comparison

Performance Metrics

The following table compares the models in terms of accuracy, precision, recall, and F1-score.

Model	Accuracy	Precision	Recall	F1-Score
KNN (k=5)	96.07%	95.8%	95.6%	95.7%
SVM (RBF)	96.10%	96.0%	95.9%	96.0%
Decision Tree	96.30%	96.5%	96.2%	96.3%
Logistic Reg (L2)	95.50%	95.3%	95.0%	95.1%

Computational Efficiency

Model	**Training Time (s)**
KNN	84.07
SVM (RBF)	250.4
Decision Tree	15.4
Logistic Reg	45.2

Key Takeaways:

- **Best Accuracy:** Decision Tree (96.30%)
- **Best Computational Efficiency:** Decision Tree for training, KNN for prediction
- **Most Balanced Model:** SVM (RBF) due to high accuracy with reasonable computation time
- **Easiest to Interpret:** Decision Tree