ENHANCING ONLINE PAYMENT FRAUD DETECTION

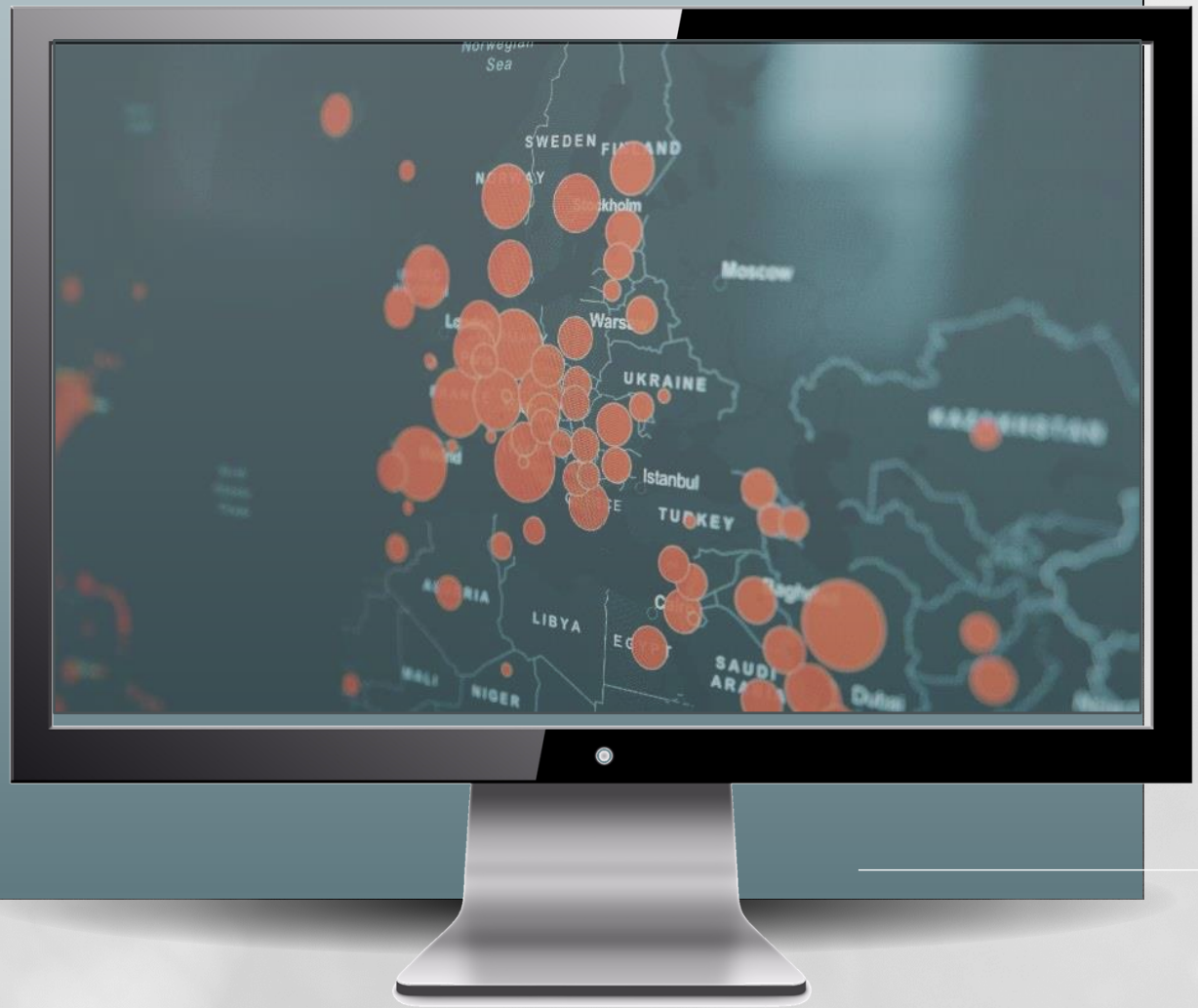
AGENDA

- INTRODUCTION TO ONLINE PAYMENTS
- OBJECTIVE
- DATASET OVERVIEW
- FRAUD DETECTION FOR THE DATA
- TRANSACTION TYPES
- EXPLORATORY DATA ANALYSIS
- MODEL SELECTION
- UNIVARIATE ANALYSIS
- DECISION TREE
- ADABOOST AND GRADIENT BOOST
- CONCLUSION
- FUTURE SCOPE

ONLINE PAYMENTS IN THE DIGITAL ERA

In today's digital age, online payments have become A fundamental aspect of daily life, offering unparalleled convenience for financial transactions. This shift from traditional methods has transformed the way we handle money, providing swift and efficient ways to make payments and conduct transactions globally.

However, this convenience comes with the challenge of ensuring security in the face of online payment fraud. As we delve into this presentation, we'll explore how data-driven methodologies can bolster the security of online payments by analyzing patterns and anomalies in historical fraudulent transactions.



OBJECTIVE

Our primary objective is to bolster online payment security through A meticulous, data-driven approach to fraud detection.

By leveraging advanced data analytics and classification techniques, we aim to enhance the accuracy and efficiency of identifying fraudulent transactions.

This initiative is crucial in fortifying the resilience of online payment systems, mitigating risks, and ultimately fostering a safer digital financial ecosystem.

DATASET OVERVIEW

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 11 columns):
 #   Column           Non-Null Count    Dtype
---  ------           --------------    -----
 0   step             1048575 non-null  int64
 1   type             1048575 non-null  object
 2   amount           1048575 non-null  float64
 3   nameOrig         1048575 non-null  object
 4   oldbalanceOrg    1048575 non-null  float64
 5   newbalanceOrig   1048575 non-null  float64
 6   nameDest         1048575 non-null  object
 7   oldbalanceDest   1048575 non-null  float64
 8   newbalanceDest   1048575 non-null  float64
 9   isFraud          1048575 non-null  int64
 10  isFlaggedFraud   1048575 non-null  int64
dtypes: float64(5), int64(3), object(3)
memory usage: 88.0+ MB
```

FRAUD DETECTION FOR THE DATASET

EXPLORATORY DATA ANALYSIS

In this Exploratory Data Analysis (EDA), categorical variables like transaction types and fraud labels are numerically encoded for analysis. Visualizations, including a pie chart for transaction type distribution and bar plots for type and fraud labels, offer insights into data characteristics. The Pearson correlation heatmap illuminates relationships between features, aiding in understanding potential patterns and informing subsequent steps in fraud detection.

MODEL SELECTION

In the realm of online payment fraud detection, Decision Trees are employed to identify complex patterns, such as specific transaction sequences indicative of fraudulent activity.

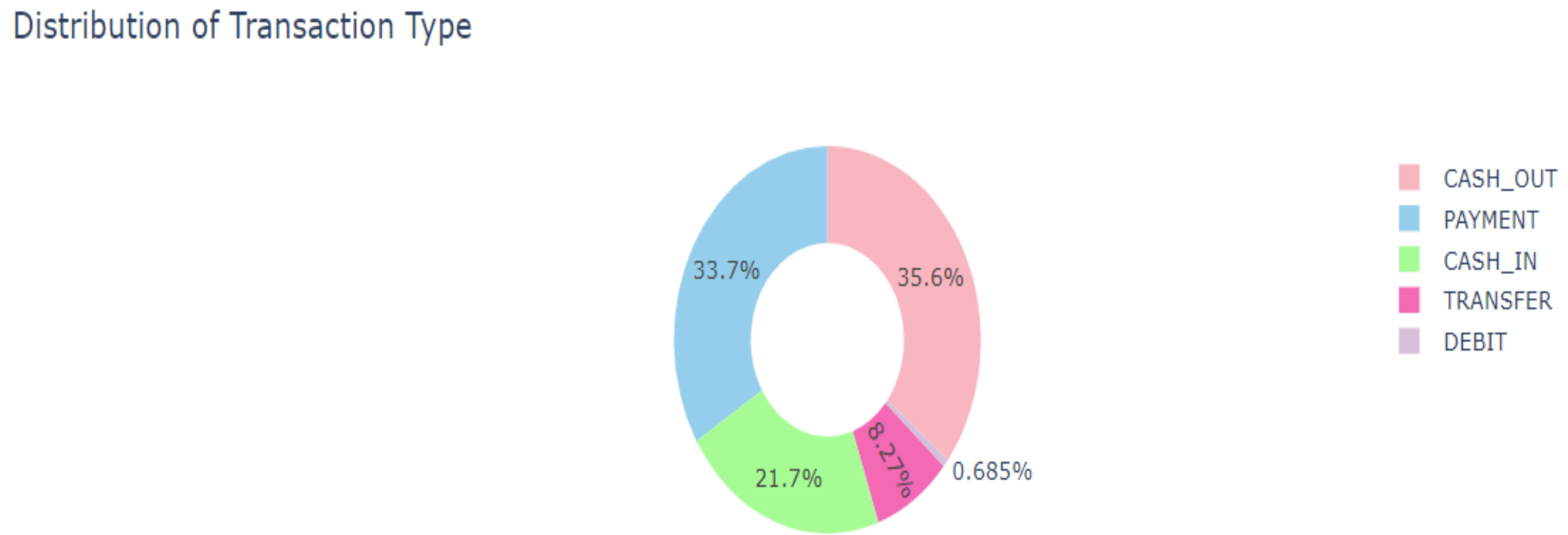
Bivariate analysis, examining relationships between two variables like transaction type and amount, reveals nuanced insights into potential fraud patterns.

AdaBoost, an ensemble learning method, iteratively corrects errors of previous models, enhancing overall accuracy. Accuracy, a common metric, must be considered cautiously in imbalanced datasets to avoid overlooking crucial instances of fraud.

Evaluating individual predictions provides detailed insights into model strengths and weaknesses, guiding continuous refinement for effective fraud detection in the dynamic landscape of online payments.

```
type       isFraud
CASH_OUT   0          373063
PAYMENT    0          353873
CASH_IN    0          227130
TRANSFER   0           86189
DEBIT      0            7178
CASH_OUT   1             578
TRANSFER   1             564
Name: count, dtype: int64
```

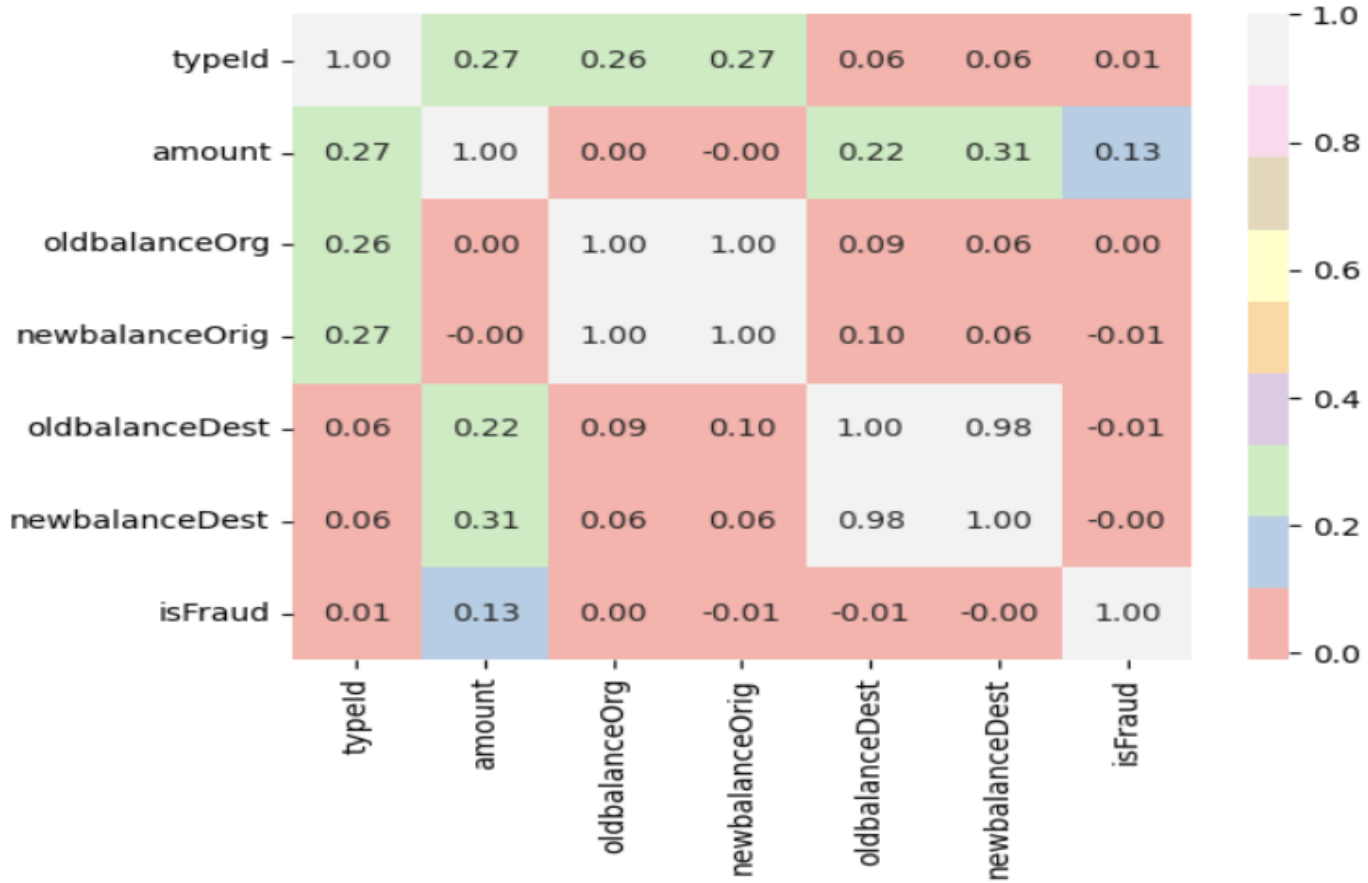
FRAUD DATA IN EACH COLUMN



DISTRIBUTION OF TRANSACTION IN THE 'TYPE' COLUMN

	typeId	amount	oldbalanceOrg	newbalanceOrig	\
typeId	1.000000	0.265263	0.262623	0.272537	
amount	0.265263	1.000000	0.004864	-0.001133	
oldbalanceOrg	0.262623	0.004864	1.000000	0.999047	
newbalanceOrig	0.272537	-0.001133	0.999047	1.000000	
oldbalanceDest	0.057842	0.215558	0.093305	0.095182	
newbalanceDest	0.059061	0.311936	0.064049	0.063725	
isFraud	0.014645	0.128862	0.003829	-0.009438	

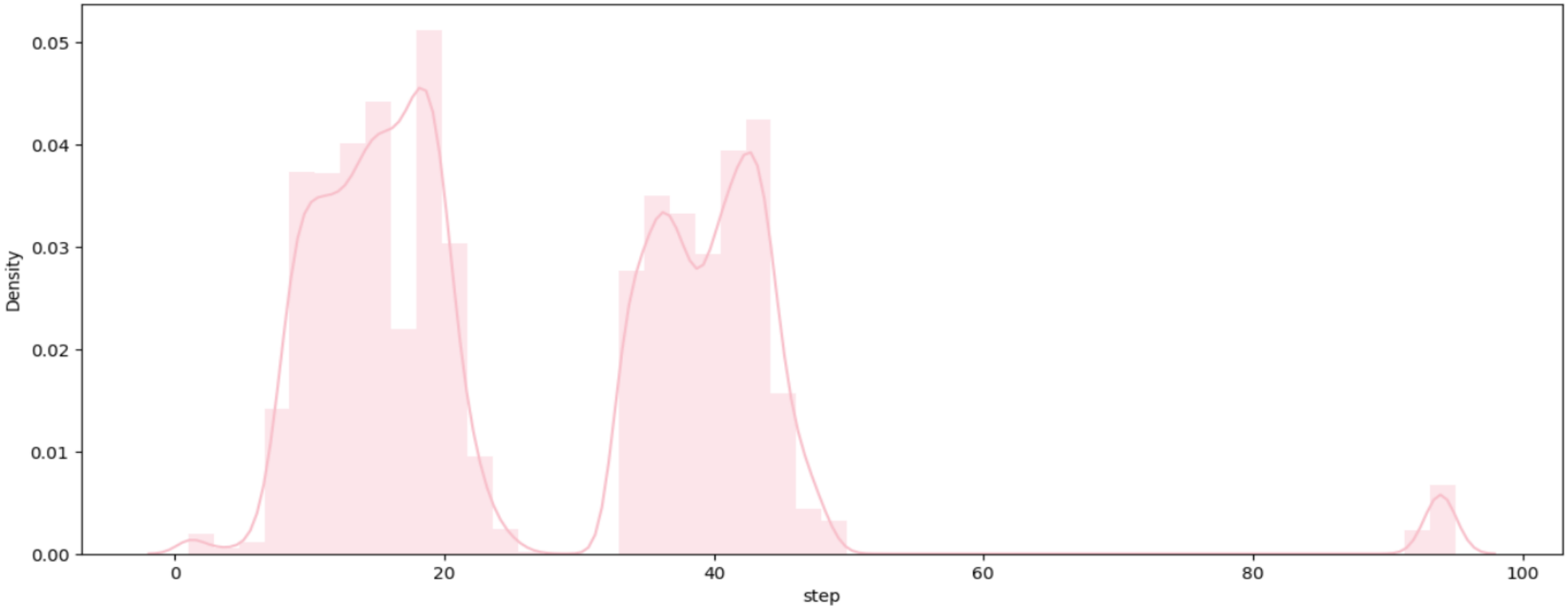
	oldbalanceDest	newbalanceDest	isFraud
typeId	0.057842	0.059061	0.014645
amount	0.215558	0.311936	0.128862
oldbalanceOrg	0.093305	0.064049	0.003829
newbalanceOrig	0.095182	0.063725	-0.009438
oldbalanceDest	1.000000	0.978403	-0.007552
newbalanceDest	0.978403	1.000000	-0.000495
isFraud	-0.007552	-0.000495	1.000000

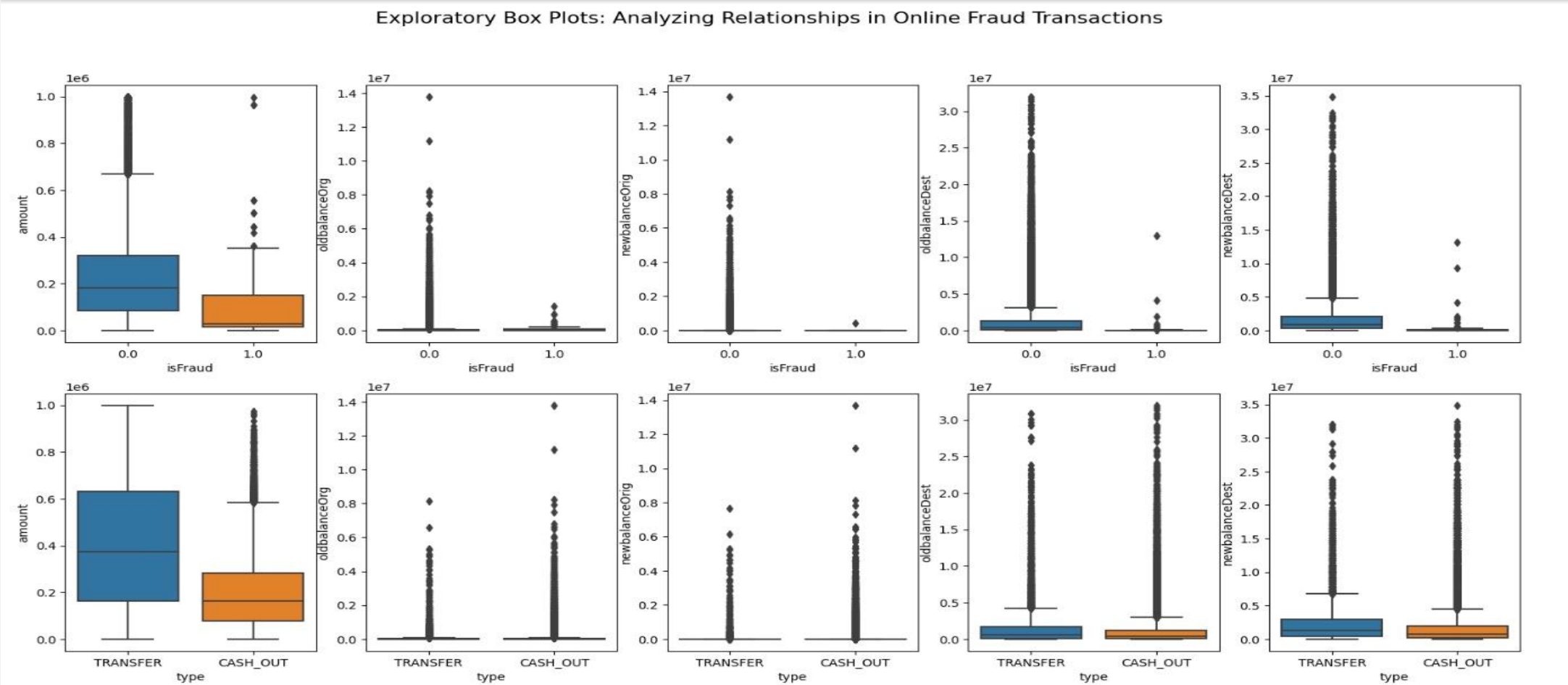


PEARSON CORRELATION PLOT

SEABORN DISTPLOT

```
The amount of loss due to fraud transaction:
<Axes: xlabel='step', ylabel='Density'>
```




Exploratory Box Plots: Analyzing Relationships in Online Fraud Transactions
amount
oldbalanceOrg
newbalanceOrig
oldbalanceDest
newbalanceDest
isFraud
type
TRANSFER
CASH_OUT

UNIVARIATE ANALYSIS FOR NUMERICAL

Univariate analysis is a fundamental component of exploratory data analysis, providing valuable insights into the characteristics of individual variables within a dataset.

In the context of our project, univariate analysis serves as the initial step to comprehend the distribution, central tendency, and variability of each feature independently.

The code evaluates numerical variables in several of ways :

Histogram : a graphic representation of the distribution of data.

Boxplot : provides a summary of the distribution of data, including percentiles and any outliers.

Loop of univariate analysis : The code then iterates through each numerical variable (opfd2_num).

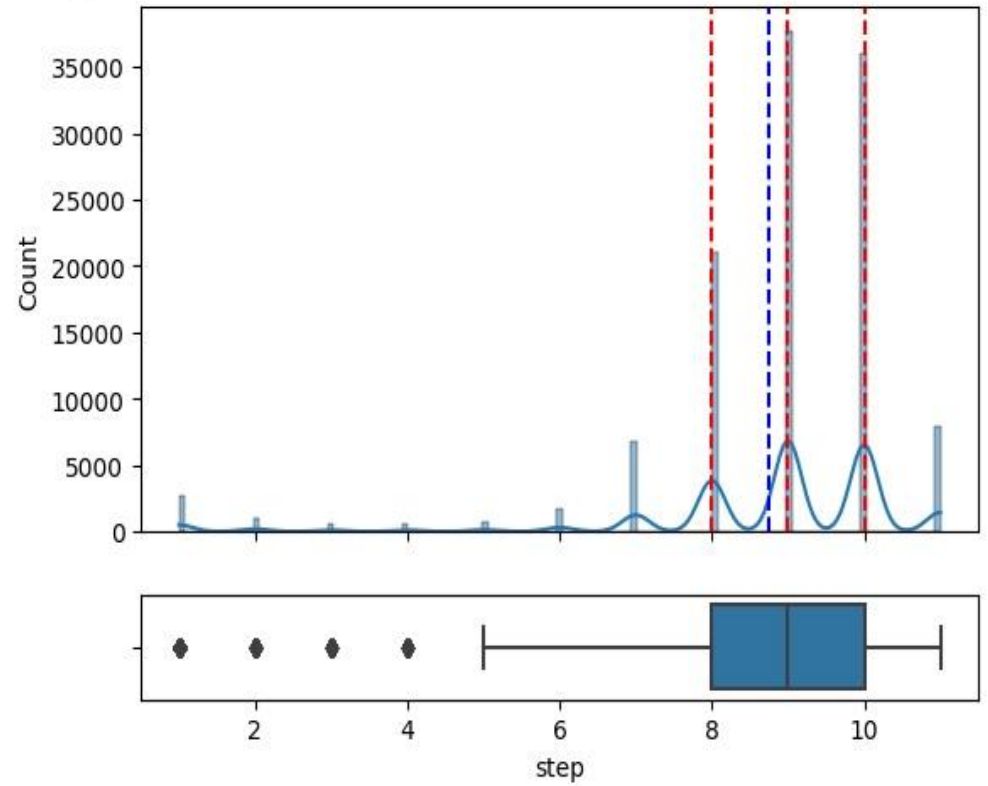
It generates a subplot with two axes for each variable : one for a histogram with percentiles and another for a boxplot. Descriptive statistics tests provide a numerical summary of the distribution's central tendency, variability, and form.

Outlier detection : identifies extreme values that deviate from the normal distribution.

Skewness test : determines whether the distribution of data is symmetric or skewed.

The Normality test : determines if the data has a normal distribution.

1. step



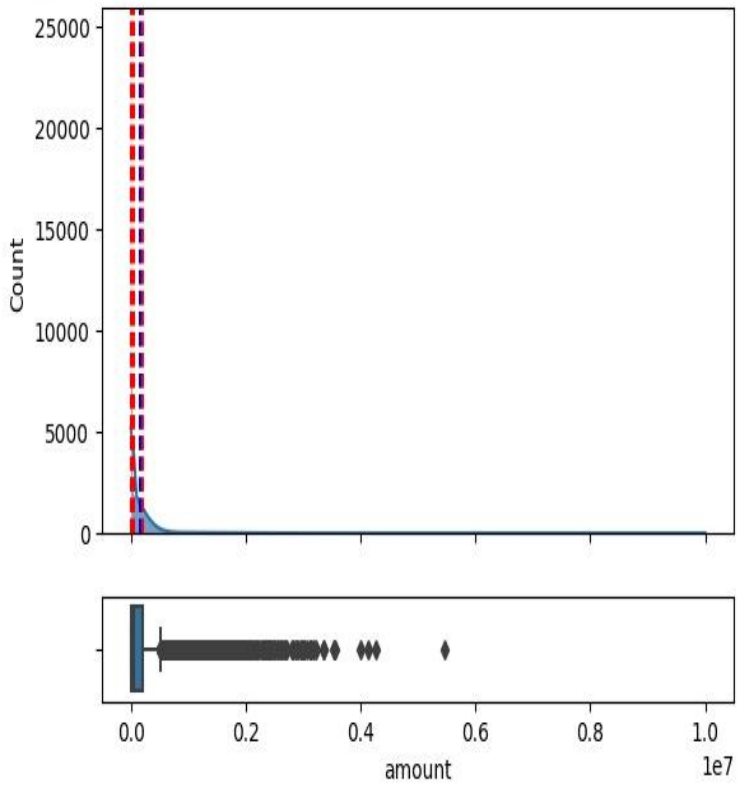

```
count    10000.000000
mean         8.753400
std          1.865888
min          1.000000
25%          8.000000
50%          9.000000
75%         10.000000
max         11.000000
Name: step, dtype: float64

lower_whisker: 5
upper_whisker: 11
outlier counts: 431

Data skewed with skew is -59.54572376723032

Data not normal
```

2. amount



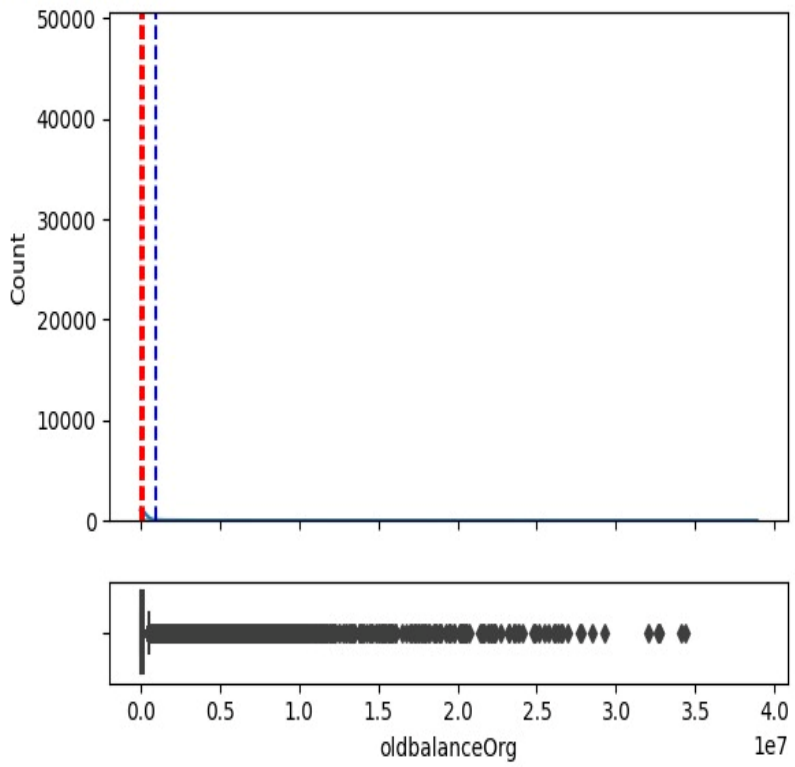

```
count    1.000000e+04
mean     1.718288e+05
std      3.407603e+05
min      3.830000e+00
25%      1.031730e+04
50%      5.113901e+04
75%      2.122844e+05
max      5.460003e+06
Name: amount, dtype: float64

lower_whisker: 3.83
upper_whisker: 515087.36
outlier counts: 598

Data skewed with skew is 84.80077175512173

Data not normal
```

3. oldbalanceOrg



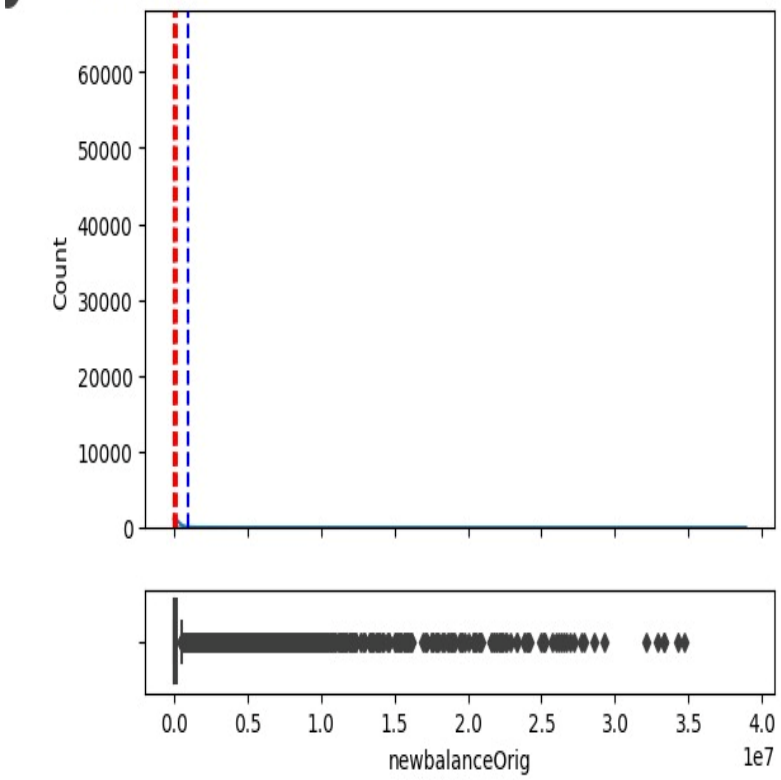

```
count    1.000000e+04
mean     9.404521e+05
std      2.924417e+06
min      0.000000e+00
25%      0.000000e+00
50%      2.044400e+04
75%      2.028375e+05
max      3.440000e+07
Name: oldbalanceOrg, dtype: float64

lower_whisker: 0.0
upper_whisker: 506032.71
outlier counts: 1798

Data skewed with skew is 84.35573949805044

Data not normal
```

4. newbalanceOrig



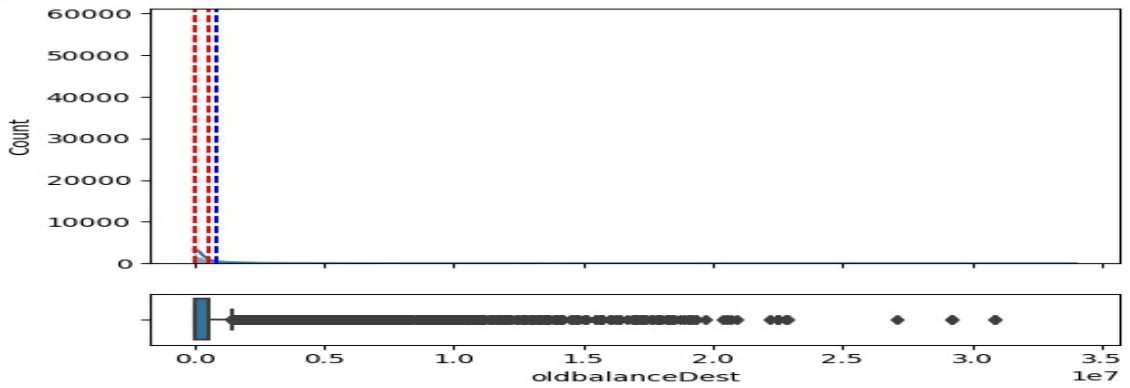

```
count    1.000000e+04
mean     9.568765e+05
std      2.963412e+06
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      2.320521e+05
max      3.470000e+07
Name: newbalanceOrig, dtype: float64

lower_whisker: 0.0
upper_whisker: 579684.04
outlier counts: 1743

Data skewed with skew is 83.97838532131851

Data not normal
```

5. oldbalanceDest

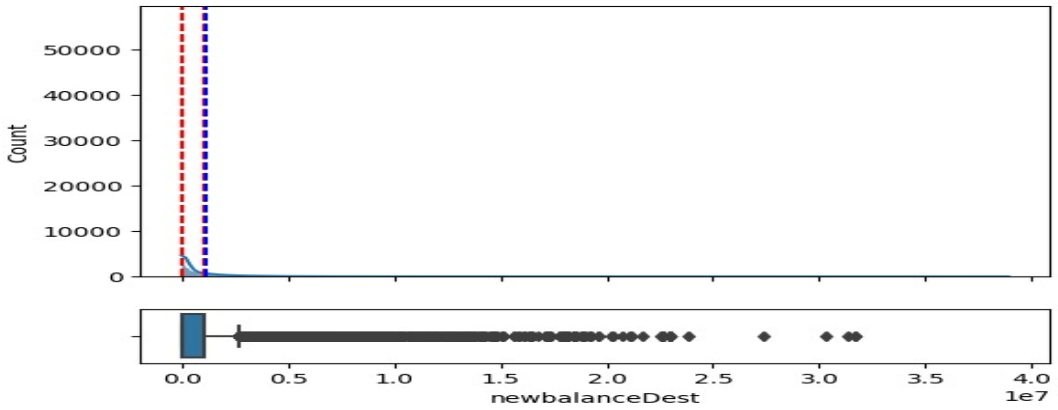



```
count     1.000000e+04
mean      8.673768e+05
std       2.373717e+06
min       0.000000e+00
25%       0.000000e+00
50%       1.987272e+04
75%       5.744560e+05
max       3.090000e+07
Name: oldbalanceDest, dtype: float64

lower_whisker: 0.0
upper_whisker: 1435519.32
outlier counts: 1428

Data skewed with skew is 84.73635645179978
```

6. newbalanceDest



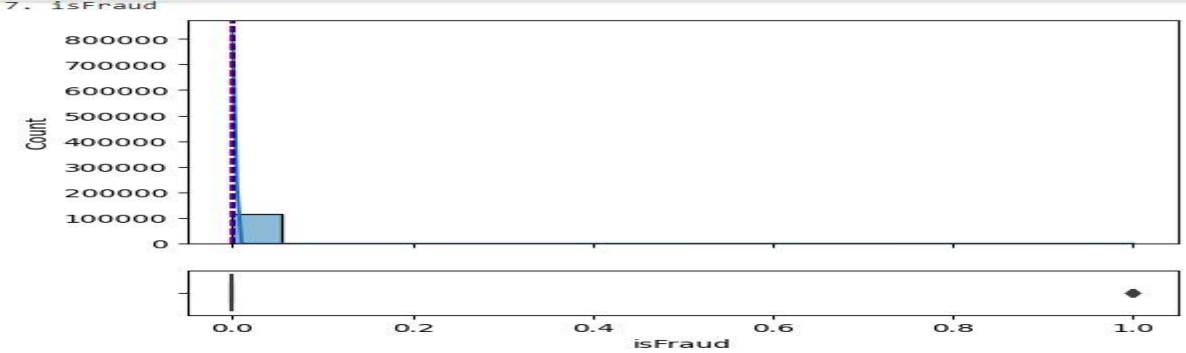

```
count     1.000000e+04
mean      1.149508e+06
std       2.719558e+06
min       0.000000e+00
25%       0.000000e+00
50%       4.882145e+04
75%       1.066982e+06
max       3.170000e+07
Name: newbalanceDest, dtype: float64

lower_whisker: 0.0
upper_whisker: 2662150.68
outlier counts: 1181

Data skewed with skew is 79.48139687875155

Data not normal
```

7. isFraud



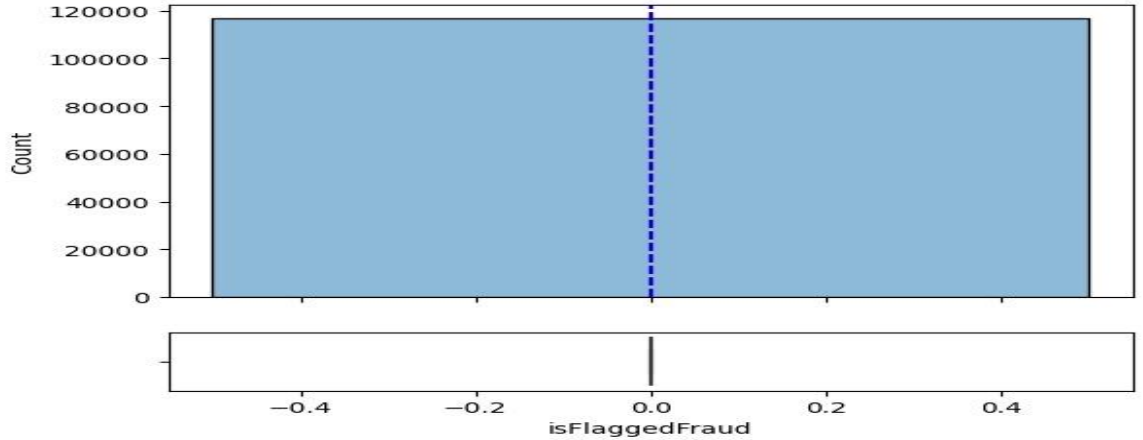

```
count     10000.000000
mean          0.001000
std           0.031609
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max           1.000000
Name: isFraud, dtype: float64

lower_whisker: 0.0
upper_whisker: 0.0
outlier counts: 10

Data skewed with skew is 145.181559536849

Data not normal
```

8. isFlaggedFraud




```
count     10000.0
mean          0.0
std           0.0
min           0.0
25%           0.0
50%           0.0
75%           0.0
max           0.0
Name: isFlaggedFraud, dtype: float64

lower_whisker: 0.0
upper_whisker: 0.0
outlier counts: 0

Data not skewed

Data normal
```

```
# Get user input for a transaction
transaction_type = input("Enter transaction type (CASH_OUT, PAYMENT, CASH_IN, TRANSFER, DEBIT): ")
amount = float(input("Enter transaction amount: "))
old_balance = float(input("Enter old balance of the origin account: "))
new_balance = float(input("Enter new balance of the origin account: "))
```

```
Enter transaction type (CASH_OUT, PAYMENT, CASH_IN, TRANSFER, DEBIT):  CASH_OUT
Enter transaction amount:  18627.02
Enter old balance of the origin account:  18627.02
Enter new balance of the origin account:  0
```

```
# Map transaction type to numeric value
transaction_type_mapping = {"CASH_OUT": 1, "PAYMENT": 2, "CASH_IN": 3, "TRANSFER": 4, "DEBIT": 5}
mapped_transaction_type = transaction_type_mapping.get(transaction_type.upper(), -1)

# Check if the transaction type is valid
if mapped_transaction_type == -1:
    print("Invalid transaction type.")
else:
    # Make a prediction for the user's input
    user_input_features = np.array([[mapped_transaction_type, amount, old_balance, new_balance]])
    prediction = model.predict(user_input_features)

    # Output the prediction
    if prediction == 0:
        print("The transaction is predicted to be not a fraud.")
    else:
        print("The transaction is predicted to be fraud.")
```

```
The transaction is predicted to be fraud.
```

```
# Get user input for a transaction
transaction_type = input("Enter transaction type (CASH_OUT, PAYMENT, CASH_IN, TRANSFER, DEBIT): ")
amount = float(input("Enter transaction amount: "))
old_balance = float(input("Enter old balance of the origin account: "))
new_balance = float(input("Enter new balance of the origin account: "))
```

```
Enter transaction type (CASH_OUT, PAYMENT, CASH_IN, TRANSFER, DEBIT):  PAYMENT
Enter transaction amount:  30000
Enter old balance of the origin account:  49999
Enter new balance of the origin account:  2368
```

```
# Map transaction type to numeric value
transaction_type_mapping = {"CASH_OUT": 1, "PAYMENT": 2, "CASH_IN": 3, "TRANSFER": 4, "DEBIT": 5}
mapped_transaction_type = transaction_type_mapping.get(transaction_type.upper(), -1)

# Check if the transaction type is valid
if mapped_transaction_type == -1:
    print("Invalid transaction type.")
else:
    # Make a prediction for the user's input
    user_input_features = np.array([[mapped_transaction_type, amount, old_balance, new_balance]])
    prediction = model.predict(user_input_features)

    # Output the prediction
    if prediction == 1:
        print("The transaction is predicted to be a fraud.")
    else:
        print("The transaction is predicted to be not a fraud.")
```

```
The transaction is predicted to be not a fraud.
```

USER INTERACTIVE FRAUD DETECTION USING DECISION TREE

BOOSTING ALGORITHMS

AdaBoost Model Accuracy: 0.9990367926147743

	precision	recall	f1-score	support
Fraud	0.81	0.18	0.29	117
No Fraud	1.00	1.00	1.00	104741
accuracy			1.00	104858
macro avg	0.90	0.59	0.65	104858
weighted avg	1.00	1.00	1.00	104858

In the provided code, Adaboost and Gradient boosting are used for classification tasks. These ensemble methods are employed to enhance the performance of decision tree classifiers.

Let's break down why these techniques are used :

Adaboost (Adaptive Boosting) :

Adaboost is used to improve the classification performance, especially in the presence of imbalanced data. In the context of fraud detection in online transactions, the adaboost algorithm plays a pivotal role as an ensemble learning method.

This algorithm excels in combining the strengths of multiple weak learners, typically decision tree classifiers, to form a robust and accurate predictive model.

Adaboost assigns weights to each data point, focusing on misclassified instances in successive iterations to iteratively improve the model's performance.

In the provided python code for fraud detection, adaboost is applied with 50 weak learners to create a classification model using features like transaction type, amount, old balance, and new balance.

The model is trained, evaluated for accuracy, and then used to make predictions. Adaboost's ability to handle imbalanced data and enhance the predictive capabilities of the model makes it a valuable asset in effectively identifying fraudulent activities within online transactions.

Gradient Boosting :

Similar to AdaBoost, Gradient Boosting is employed for enhancing classification accuracy and handling imbalanced data. Gradient Boosting, another powerful ensemble learning technique, is employed in the fraud detection code to enhance the accuracy and predictive capabilities of the model.

Unlike AdaBoost, Gradient Boosting builds a series of decision trees sequentially, with each tree aiming to correct the errors of its predecessor. In the context of fraud detection, this method is highly effective in capturing complex relationships and patterns within the data.

The Python code utilizes the Gradient Boosting Classifier with 100 trees, incorporating features such as transaction type, amount, old balance, and new balance.

The model is trained iteratively, with each tree refining the predictions of the previous ones.

The final model is then evaluated for its accuracy and employed to detect Potential fraudulent activities in online transactions. Gradient Boosting, with its ability to handle non-linear relationships and high predictive accuracy, stands as a robust tool in the realm of fraud detection.

Gradient Boosting Model Accuracy: 0.9989223521333613

	precision	recall	f1-score	support
Fraud	0.62	0.09	0.15	117
No Fraud	1.00	1.00	1.00	104741
accuracy			1.00	104858
macro avg	0.81	0.54	0.57	104858
weighted avg	1.00	1.00	1.00	104858

CONCLUSION

The success of online fraud detection relies on an effective blend of exploratory data analysis, machine learning algorithms, and user engagement which is achieved in this project.

The journey begins with an in-depth analysis of the dataset, with the goal of learning about its structure, characteristics, and potential issues.

Understanding the distribution and statistical features of individual numerical variables relies heavily on univariate analysis. Distplots, boxplots, and descriptive statistics can help discover patterns, outliers, and the overall nature of the data.

Bivariate analysis broadens our understanding of relationships between numerical variables by revealing potential correlations and dependencies.

EDA provides a comprehensive picture of the dataset by integrating visuals and statistical approaches to find patterns and trends that may be suggestive of fraudulent activity.

The combination of Machine learning algorithms gradient boosting with AdaBoost shows the effectiveness of ensemble methods in capturing complex connections across data improve prediction capacities for fraud detection.

For user-interactive fraud detection, decision trees are used because of their interpretability.

User involvement with decision trees encourages collaboration while leveraging expertise for more informed decision-making.

As fraud trends evolve, Continuous development and adaptation is important for staying ahead of emerging threats in the continually changing world of online transactions.

FUTURE SCOPE

By employing a dual-pronged approach of undersampling and oversampling, the dataset can be modified to create a balanced representation of both fraudulent and non-fraudulent instances.

Undersampling ensures a reduced representation of non-fraudulent transactions, while oversampling generates synthetic instances of the minority class, mitigating the impact of class imbalance.

Subsequently, the application of ensemble learning techniques, specifically AdaBoost on the undersampled data and Gradient Boosting on the oversampled data, serves to enhance the predictive power of the model. This approach leverages the strengths of both undersampling and oversampling methods to create robust models capable of detecting fraudulent activities more effectively.

Through thorough evaluation, comparison, and optimization, this future scope aims to yield a well-balanced and finely tuned ensemble model for deployment, contributing to more accurate and reliable fraud detection in real-world scenarios.

THANK YOU

KEERTHI YALAMANCHILI
PRABHITHA VEERAMACHANENI
DEEKSHITHA CHIKKALA