Project Description

Problem Statement: AI-POWERED FRAUD DETECTION SYSTEM

DESCRIPTION: Develop an AI model capable of detecting fraudulent transactions in real-time. Use historical transaction data to train the model to identify anomalies and flag suspicious activities.

Solution Approach: Describe how AI, particularly the Random Forest model, is used to detect fraudulent transactions.

To analyze the feature importance according to your dataset, you can follow these steps in your documentation. This section will focus on how you determined which features were most influential in the Random Forest model's predictions based on the specific data you used.

1. Feature Importance Analysis

Objective:

- To determine and rank the features in your dataset that contribute most significantly to the predictions of fraudulent transactions by the Random Forest model.

2. Methodology

1. Gini Importance (Mean Decrease in Impurity):

- Description: In Random Forest models, Gini importance measures how each feature contributes to the reduction of the impurity (e.g., Gini index or entropy) across all the trees in the forest.

- Calculation: Each time a feature is used to split a node, the algorithm calculates the reduction in impurity for that split. The feature importance score is the average of these reductions across all the trees in the model.

- Implementation:

- After training the model, extract the feature importance scores using the `feature_importances_` attribute in Scikit-learn's `RandomForestClassifier`.

- Rank the features from most to least important based on these scores.

2. Permutation Importance:

- Description: Permutation importance involves shuffling the values of a feature and measuring the decrease in model performance. A significant drop in performance indicates that the feature is important for the model's predictions.

- Calculation:

- After the model is trained, shuffle the values of one feature at a time in the dataset and observe the change in performance (e.g., accuracy or AUC score).

- Features that cause a significant drop in performance when shuffled are considered important.

- Implementation:

- Use Scikit-learn's `permutation_importance` function to compute this for your model.

3. SHAP Values (SHapley Additive exPlanations):

- Description: SHAP values provide a unified measure of feature importance by calculating the contribution of each feature to the model's output. This method is based on cooperative game theory.

- Calculation:

- SHAP values explain the impact of each feature on the prediction, providing insights into both the direction (positive or negative) and magnitude of the impact.

- Implementation:

- Use the `SHAP` library in Python to compute SHAP values for each feature.

- Visualize the SHAP values using summary plots or dependence plots to better understand the feature contributions.

3. Results

1. Feature Importance Ranking:

- Present a bar chart or table listing the features in descending order of importance based on Gini Importance or Permutation Importance.

- Include the corresponding importance scores for clarity.

2. SHAP Summary Plot:

- Provide a SHAP summary plot that shows the distribution of SHAP values for all features, highlighting which features have the most influence on model predictions.

- Discuss any interesting patterns or insights derived from this analysis.

4. Discussion

- Key Influential Features: Identify the top features and explain why they might be critical in detecting fraudulent transactions. For example, features related to transaction amount, frequency of transactions, or geographic location might be particularly important.

- Business Implications: Discuss how understanding feature importance can help stakeholders focus on key areas for fraud prevention.

- Model Trustworthiness: Explain how feature importance analysis enhances the interpretability and trustworthiness of the model, especially in critical applications like fraud detection.

5. Conclusion

- Summarize the findings from the feature importance analysis.

- Highlight any potential actions or further investigations that could be derived from understanding which features are most important.

Including this detailed analysis in your documentation will provide valuable insights into your model's decision-making process, enhancing the overall interpretability and utility of your fraud detection system.

Dataset Overview

Here's a detailed analysis of your dataset based on the information provided:

1. Column Overview

The dataset has the following columns (assuming from the skewness calculation):

- **step**: Likely represents a time step or sequence in the data.
- **amount**: Represents the transaction amount.
- **fraud**: Indicates whether the transaction is fraudulent (`1`) or not (`0`).

2. Skewness

- **step**: -0.119 (Slight negative skewness, indicating a relatively symmetric distribution around the mean).
- **amount**: 32.197 (Highly positively skewed, suggesting that most transactions have smaller amounts, with a few large outliers).
- **fraud**: 8.936 (Also highly positively skewed, indicating that fraudulent transactions are much rarer compared to non-fraudulent ones).

3. Descriptive Statistics

You can explore more descriptive statistics like mean, median, standard deviation, and percentiles to understand the distribution further. Here's a sample of what this might look like:

```
# Descriptive statistics

df.describe()
```

4. Missing Values

It's important to check if there are any missing values in the dataset, which can affect your analysis and model training:

```
# Check for missing values

missing_values = df.isnull().sum()

missing_values
```

5. Class Distribution

Given the high skewness in the `fraud` column, it's likely that the dataset is imbalanced, with far fewer fraudulent transactions than non-fraudulent ones. You can confirm this with:

```
# Class distribution

df['fraud'].value_counts()
```

6. Correlation

Understanding how features correlate with each other can provide insights into the relationships in the data:

```
# Correlation matrix

correlation_matrix = df.corr()

correlation_matrix
```

7. Data Imbalance Handling

Given the skewness in the `fraud` column, you may need to handle data imbalance using techniques such as:

- **Oversampling**: Increase the number of fraudulent samples by duplicating or synthesizing new data points (e.g., using SMOTE).
- **Undersampling**: Reduce the number of non-fraudulent samples.
- **Class Weights**: Adjust the class weights in your model to give more importance to the minority class (fraud).

8. Data Visualization

Visualizing the data can help in understanding the distributions and relationships. For instance:

```
import seaborn as sns

import matplotlib.pyplot as plt


# Distribution of 'amount'

sns.histplot(df['amount'], kde=True)

plt.title('Distribution of Transaction Amounts')

plt.show()


# Fraud vs. Non-Fraud

sns.countplot(x='fraud', data=df)

plt.title('Fraud vs. Non-Fraud Transactions')

plt.show()
```

9. Feature Engineering

To improve model performance, consider:

- **Creating new features**: Such as transaction time, customer ID, or aggregating transaction amounts over time.
- **Transforming skewed data**: Apply log transformation to reduce skewness in the `amount` column.

10. Modeling

Given the imbalance and skewness, you may want to consider models robust to such issues, such as:

- **Random Forest**
- **XGBoost**
- **Logistic Regression** with class weights

To determine the skewness of the dataset, you can use the `.skew()` function provided by Pandas. Skewness measures the asymmetry of the distribution of values in a dataset. Here's how you can check the skewness of each numerical column in your dataset:

1. Calculate Skewness

- Use the `.skew()` function on the DataFrame to get the skewness of each numerical column.

python

Copy code

```
skewness = df.skew()
print("Skewness of each column:")
print(skewness)
```

2. Interpret Skewness Values

- **Skewness = 0**: The data is perfectly symmetrical.
- **Skewness > 0**: The data is positively skewed (right-skewed), with a longer tail on the right.
- **Skewness < 0**: The data is negatively skewed (left-skewed), with a longer tail on the left.

3. Visualize Skewness (Optional)

- You can also visualize the distribution of data using histograms or density plots to better understand the skewness.

python

Copy code

```
import matplotlib.pyplot as plt

for column in df.select_dtypes(include=['float64', 'int64']).columns:
    plt.figure(figsize=(6,4))
```

- `plt.hist(df[column], bins=30, edgecolor='k')`
- `plt.title(f'Distribution of {column}')`
- `plt.xlabel(column)`
- `plt.ylabel('Frequency')`
- `plt.show()`

Summary

- **Skewness > 1** or **Skewness < -1**: Highly skewed.
- **0.5 < Skewness ≤ 1** or **-1 ≤ Skewness < -0.5**: Moderately skewed.
- **-0.5 ≤ Skewness ≤ 0.5**: Approximately symmetric.
-

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Load the dataset
file_path = '/mnt/data/train_hsbc_df - train_hsbc_df.csv'
df = pd.read_csv(file_path)

# Check for missing values
print("Missing values in each column:")
print(df.isnull().sum())

# If there are missing values, you might want to fill or drop them
# df = df.dropna()  # Optionally drop rows with missing values

# Assuming 'fraud' is the target variable and others are features
X = df.drop(['fraud'], axis=1)
y = df['fraud']

# Convert any non-numeric columns to numeric if necessary
# Example: X['categorical_column'] = pd.get_dummies(X['categorical_column'], drop_first=True)

# Split data into training and test sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42,
stratify=y)

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize the Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42,
class_weight='balanced')

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred))
```

Explanation:

1. **Data Loading**: The dataset is loaded from the provided path.
2. **Missing Values**: A check for missing values is done. You can handle them by either dropping or filling them.
3. **Splitting Data**: The data is split into training and test sets, ensuring that the distribution of the `fraud` class is maintained using `stratify=y`.
4. **Standardization**: The features are standardized to have a mean of 0 and a standard deviation of 1, which helps in improving the model performance.
5. **Model Initialization**: A `RandomForestClassifier` is initialized with `n_estimators=100` (number of trees in the forest) and `class_weight='balanced'` to handle the class imbalance.
6. **Training**: The model is trained on the training data.
7. **Predictions**: Predictions are made on the test set.

8. **Evaluation**: The model's performance is evaluated using a confusion matrix, classification report (precision, recall, F1-score), and accuracy score.

Tips:

- **Hyperparameter Tuning**: You can improve the model by tuning hyperparameters such as the number of trees (`n_estimators`), depth of trees (`max_depth`), etc., using GridSearchCV or RandomizedSearchCV.
- **Feature Importance**: After training, you can inspect the importance of each feature using `model.feature_importances_`.

This code should give you a good starting point to train a Random Forest model for fraud detection.

Helps to overcome the overfitting problem

Smote oversampling

SMOTE (Synthetic Minority Over-sampling Technique) is a popular technique used to handle imbalanced datasets. It works by creating synthetic samples of the minority class, thereby increasing its representation in the dataset.

When to Use SMOTE

- **Imbalanced Classes:** SMOTE is particularly useful when the dataset has a significant imbalance between the majority and minority classes (e.g., fraud detection where fraudulent cases are much less frequent).
- **Improving Model Performance:** It can help models perform better by providing them with more examples of the minority class during training.

How SMOTE Works

- SMOTE generates synthetic samples by selecting a point from the minority class and finding its k-nearest neighbors. Then, it randomly selects one of these neighbors and creates a synthetic data point by interpolating between the selected point and its neighbor.

Steps to Handle Multiple Categorical Columns

Identify Categorical Columns: You can use the method mentioned earlier to get all categorical columns in one go:
python
Copy code

```
categorical_columns = data.select_dtypes(include=['object',
'category']).columns.tolist()
print("Categorical Columns:", categorical_columns)
```

1.
2. **Choose Encoding Methods:** Depending on the number of unique values in each categorical column, you can choose between different encoding methods:
    - **Label Encoding:** Use this when a categorical variable has an ordinal relationship (e.g., 'low', 'medium', 'high').
    - **One-Hot Encoding:** Use this for nominal categorical variables with no ordinal relationship (e.g., 'red', 'blue', 'green'). This method creates binary columns for each category.

Encoding Categorical Columns: Here's how to apply one-hot encoding to all categorical columns using `pd.get_dummies()`:
python
Copy code

```
# One-Hot Encoding for all categorical columns
data_encoded = pd.get_dummies(data, columns=categorical_columns,
drop_first=True)
```

3.
    - `drop_first=True` helps avoid multicollinearity by dropping the first category in each column.
4. **Check for High Cardinality:** If some categorical columns have a very high number of unique values (high cardinality), consider using techniques like:
    - **Frequency Encoding:** Replace the category with its frequency.
    - **Target Encoding:** Replace the category with the mean of the target variable for that category.

Re-check the DataFrame: After encoding, verify the structure of the new DataFrame to ensure the changes are as expected:
python
Copy code

```
print(data_encoded.head())
print(data_encoded.info())
```

5.

Prepare for Model Training: Now that your categorical variables are encoded, you can split your data into features (`X`) and the target variable (`y`), and then proceed to fit your model:
python
Copy code

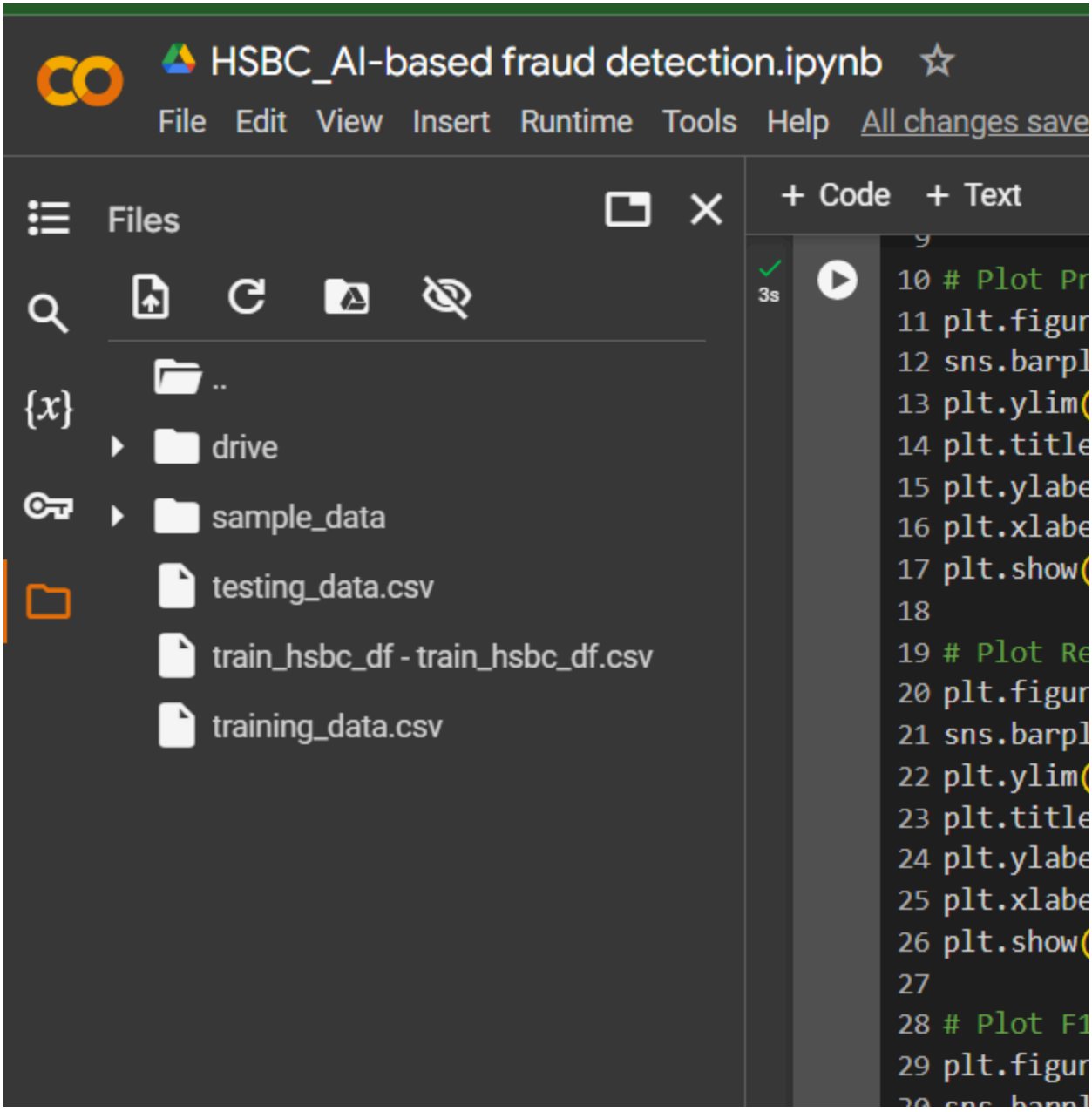
```
X = data_encoded.drop('target_column_name', axis=1)  # replace with your actual target column name
y = data_encoded['target_column_name']
```

Conclusion

By following these steps, you can effectively handle multiple categorical columns in your dataset, ensuring that your model can learn from the relevant features. If you have any questions about specific columns or need further assistance, feel free to ask!

Gallery

1. Datasets created



2. Importing libraries

```
Suggested code may be subject to a licence | DaddarioLuigi/UmamiDetector | github.com/nikitabhandari-dl/Dataset
import pandas as pd
import numpy as np
import seaborn as sns
import seaborn as sns
import matplotlib.pyplot as plt
```

3. Dataset overview

4.

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[4]  1 df = pd.read_csv('/content/train_hsbc_df - train_hsbc_df.csv')
     2 #shape the data
     3 df.shape
```

```
(513643, 10)
```

```
[5]  1 df.head()
```

	step	customer	age	gender	zipcodeOri	merchant	zipMerchant	category	amount	fraud
0	0	C583110837'	3'	M'	28007'	M480139044'	28007'	es_health'	44.26	1
1	0	C1332295774'	3'	M'	28007'	M480139044'	28007'	es_health'	324.50	1
2	0	C1160421902'	3'	M'	28007'	M857378720'	28007'	es_hotelservices'	176.32	1
3	0	C966214713'	3'	M'	28007'	M857378720'	28007'	es_hotelservices'	337.41	1
4	0	C1450140987'	4'	F'	28007'	M1198415165'	28007'	es_wellnessandbeauty'	220.11	1

```
[6]  1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 513643 entries, 0 to 513642
Data columns (total 10 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   step         513643 non-null  int64
 1   customer     513643 non-null  object
 2   age          513643 non-null  object
 3   gender       513643 non-null  object
 4   zipcodeOri   513643 non-null  object
 5   merchant     513643 non-null  object
```

```
[7]  1 df.isnull().sum()
```

	0
step	0
customer	0
age	0
gender	0
zipcodeOri	0
merchant	0
zipMerchant	0
category	0
amount	0
fraud	0

dtype: int64

```
[8]  1 df.describe()
```

	step	amount	fraud
count	513643.000000	513643.000000	513643.000000

5.

```
[7]  1 df.isnull().sum()
```

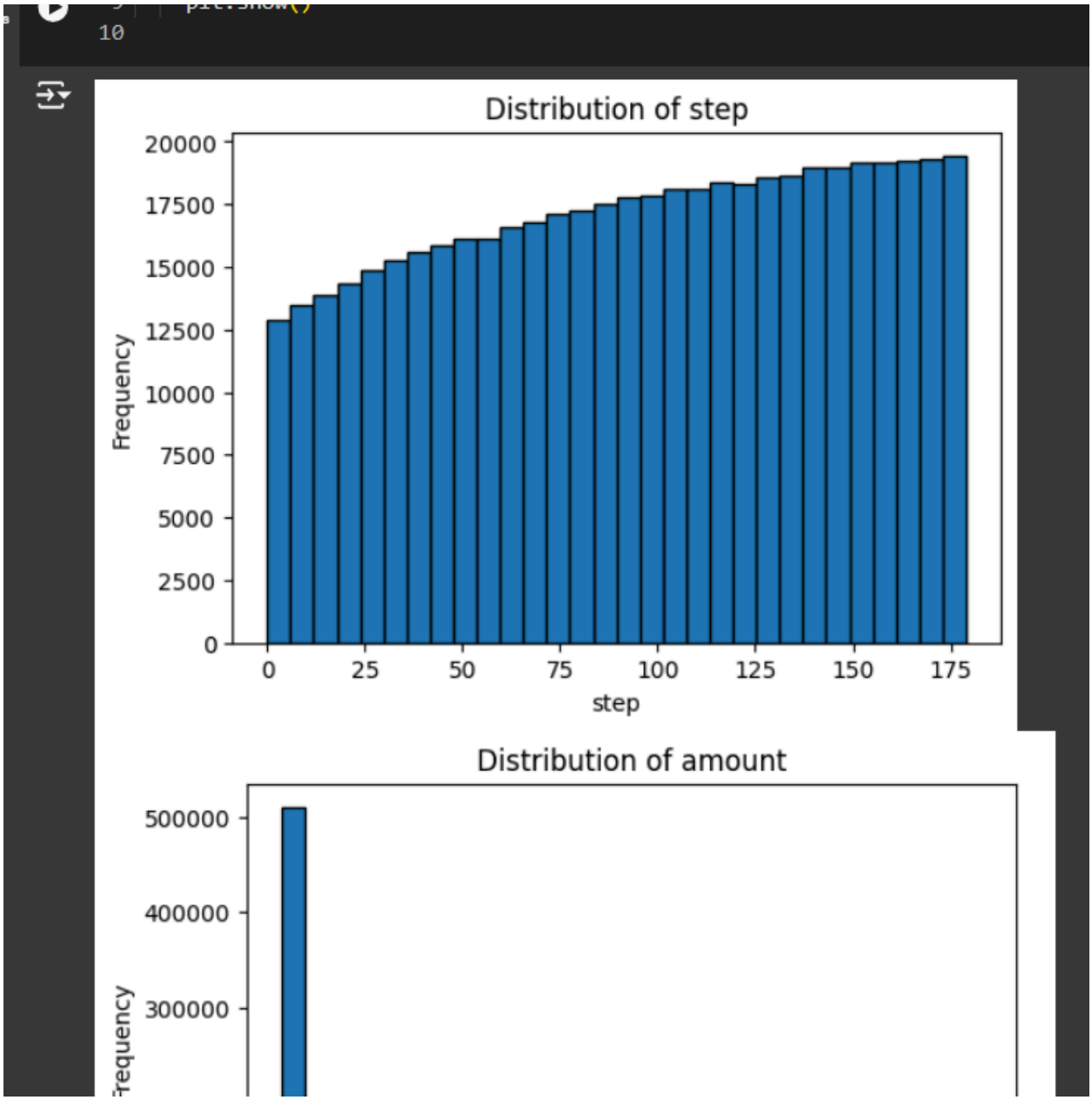
	0
step	0
customer	0
age	0
gender	0
zipcodeOri	0
merchant	0
zipMerchant	0
category	0
amount	0
fraud	0

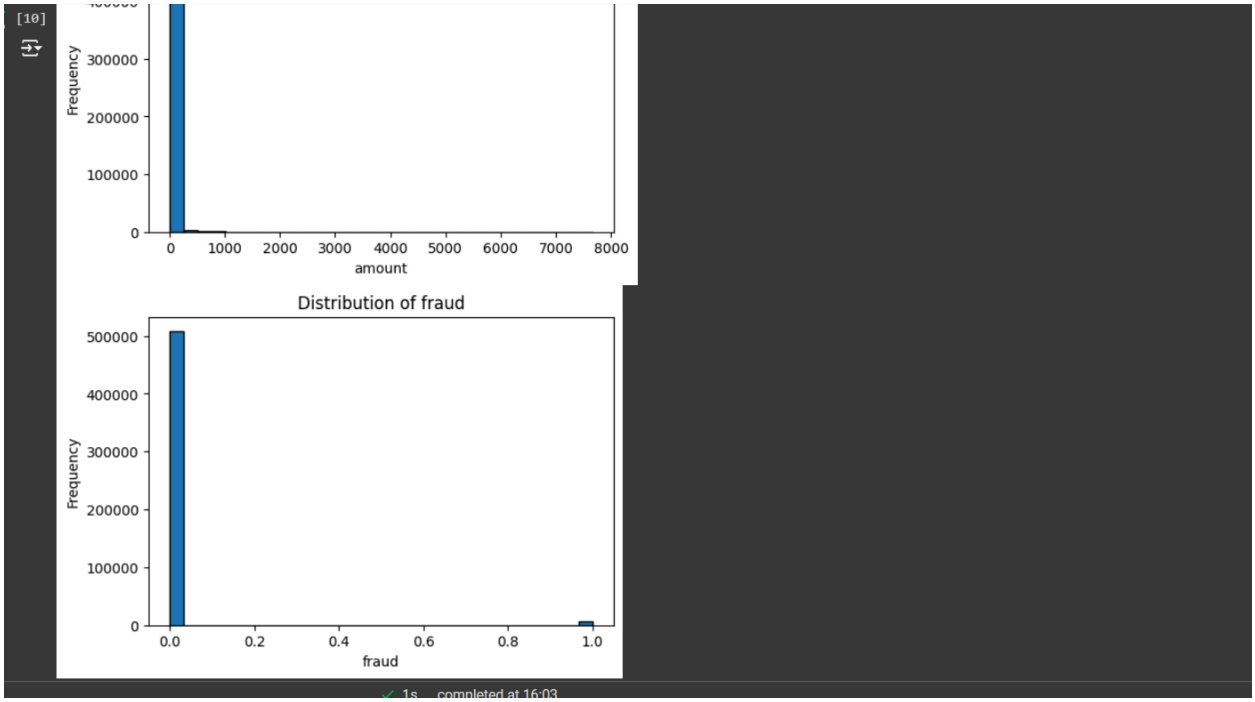
dtype: int64

```
[8]  1 df.describe()
```

	step	amount	fraud
count	513643.000000	513643.000000	513643.000000

6.


10
Distribution of step
20000
17500
15000
12500
10000
7500
5000
2500
0
Frequency
0
25
50
75
100
125
150
175
step
Distribution of amount
500000
400000
300000
Frequency

7. 

8. Loading the dataset:
9. Loading the datasets

```
[14]  1 import pandas as pd
      2
      3 # Load the training and testing datasets
      4 train_set = pd.read_csv('/content/training_data.csv')
      5 test_set = pd.read_csv('/content/testing_data.csv')
      6
      7 # Separate features and target variable
      8 X_train = train_set.drop(columns=['fraud'])
      9 y_train = train_set['fraud']
     10
     11 X_test = test_set.drop(columns=['fraud'])
     12 y_test = test_set['fraud']
     13
```

10. Splitting the dataset:

Splitting the dataset into training and testing dataset

tringin the dataset on random forest classifier

```
[13]  1 # Import the necessary libraries
      2 import pandas as pd
      3 from sklearn.model_selection import train_test_split
      4
      5 # Load the dataset
      6 data = pd.read_csv('/content/train_hsbc_df - train_hsbc_df.csv')
      7
      8 # Assuming 'fraud' is the target column indicating fraudulent transactions
      9 X = data.drop(columns=['fraud'])  # Features
     10 y = data['fraud']  # Target variable
     11
     12 # Split the data into training and testing sets
     13 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
     14
     15 # Combine X and y back together for training and testing sets
     16 train_set = pd.concat([X_train, y_train], axis=1)
     17 test_set = pd.concat([X_test, y_test], axis=1)
     18
     19 # Save the training and testing sets to CSV files
     20 train_set.to_csv('training_data.csv', index=False)
     21 test_set.to_csv('testing_data.csv', index=False)
     22
     23 print("Training and testing datasets saved as 'training_data.csv' and 'testing_data.csv'.")
     24
```

```
Training and testing datasets saved as 'training_data.csv' and 'testing_data.csv'.
```

11. Training the model:

```
[11]  1 import pandas as pd
      2 import numpy as np
      3 from sklearn.model_selection import train_test_split
      4 from sklearn.preprocessing import StandardScaler
      5 from sklearn.ensemble import RandomForestClassifier
      6 from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
      7
      8 # Load the dataset
      9 file_path = '/content/train_hsbc_df - train_hsbc_df.csv'
     10 df = pd.read_csv(file_path)
     11
     12 # Check for missing values
     13 print("Missing values in each column:")
     14 print(df.isnull().sum())
     15
     16 # If there are missing values, you might want to fill or drop them
     17 # df = df.dropna()  # Optionally drop rows with missing values
     18
     19 # Assuming 'fraud' is the target variable and others are features
     20 X = df.drop(['fraud'], axis=1)
     21 y = df['fraud']
     22
     23 # Identify and drop non-numeric columns (e.g., customer ID)
     24 non_numeric_columns = X.select_dtypes(exclude=['float64', 'int64']).columns
     25 X = X.drop(columns=non_numeric_columns)  # Drop non-numeric columns
     26
     27 # Split data into training and test sets
     28 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
     29
     30 # Standardize the data
     31 scaler = StandardScaler()
     32 X_train = scaler.fit_transform(X_train)
     33 X_test = scaler.transform(X_test)
     34
```

```
40
41 # Make predictions
42 y_pred = model.predict(X_test)
43
44 # Evaluate the model
45 print("Confusion Matrix:")
46 print(confusion_matrix(y_test, y_pred))
47
48 print("\nClassification Report:")
49 print(classification_report(y_test, y_pred))
50
51 print("\nAccuracy Score:")
52 print(accuracy_score(y_test, y_pred))
```

```
Missing values in each column:
step          0
customer      0
age           0
gender        0
zipcodeOri    0
merchant      0
zipMerchant   0
category      0
amount        0
fraud         0
dtype: int64
Confusion Matrix:
[[151852    381]
 [   806   1054]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00    152233
           1       0.73      0.57      0.64      1860

    accuracy                           0.99    154093
   macro avg       0.86      0.78      0.82    154093
```

1s completed at 16:03

Accuracy:

```
+ Code + Text                                                    T4
zipcodeOri    0
merchant      0
zipMerchant   0
category      0
amount        0
fraud         0
dtype: int64
Confusion Matrix:
[[151852    381]
 [   806   1054]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00    152233
           1       0.73      0.57      0.64      1860

    accuracy                           0.99    154093
   macro avg       0.86      0.78      0.82    154093
weighted avg       0.99      0.99      0.99    154093


Accuracy Score:
0.9922968596886296
```

12.

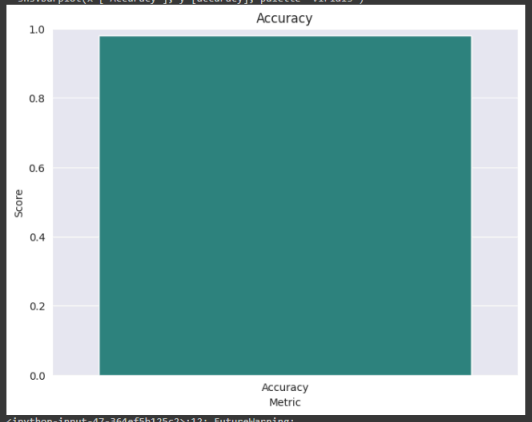
```
[11]
Accuracy Score:
0.9922968596886296

[12]  1 print('Number of fraudulent transactions \t: {}'.format(df['fraud'].sum()))
      2 print('Number of non-fraudulent transactions \t: {}'.format(len(df[df['fraud']==0])))
      3
      4

Number of fraudulent transactions       : 6200
Number of non-fraudulent transactions   : 507443
```

16.Plotting accuracy metrics:

```
[47]  1 # Plot Accuracy
      2 plt.figure(figsize=(8, 6))
      3 sns.barplot(x=['Accuracy'], y=[accuracy], palette='viridis')
      4 plt.ylim(0, 1)
      5 plt.title('Accuracy')
      6 plt.ylabel('Score')
      7 plt.xlabel('Metric')
      8 plt.show()
      9
     10 # Plot Precision
     11 plt.figure(figsize=(8, 6))
     12 sns.barplot(x=['Precision'], y=[precision], palette='viridis')
     13 plt.ylim(0, 1)
     14 plt.title('Precision')
     15 plt.ylabel('Score')
     16 plt.xlabel('Metric')
     17 plt.show()
     18
     19 # Plot Recall
     20 plt.figure(figsize=(8, 6))
     21 sns.barplot(x=['Recall'], y=[recall], palette='viridis')
     22 plt.ylim(0, 1)
     23 plt.title('Recall')
     24 plt.ylabel('Score')
     25 plt.xlabel('Metric')
     26 plt.show()
     27
     28 # Plot F1 Score
     29 plt.figure(figsize=(8, 6))
     30 sns.barplot(x=['F1 Score'], y=[f1_score_value], palette='viridis')
     31 plt.ylim(0, 1)
     32 plt.title('F1 Score')
     33 plt.ylabel('Score')
     34 plt.xlabel('Metric')
     35 plt.show()
     36
     37 # Plot Confusion Matrix
     38 plt.figure(figsize=(8, 6))
     39 sns.heatmap(cmat, annot=True, fmt='d', cmap='Blues',

✓ 0s  completed at 17:14
```

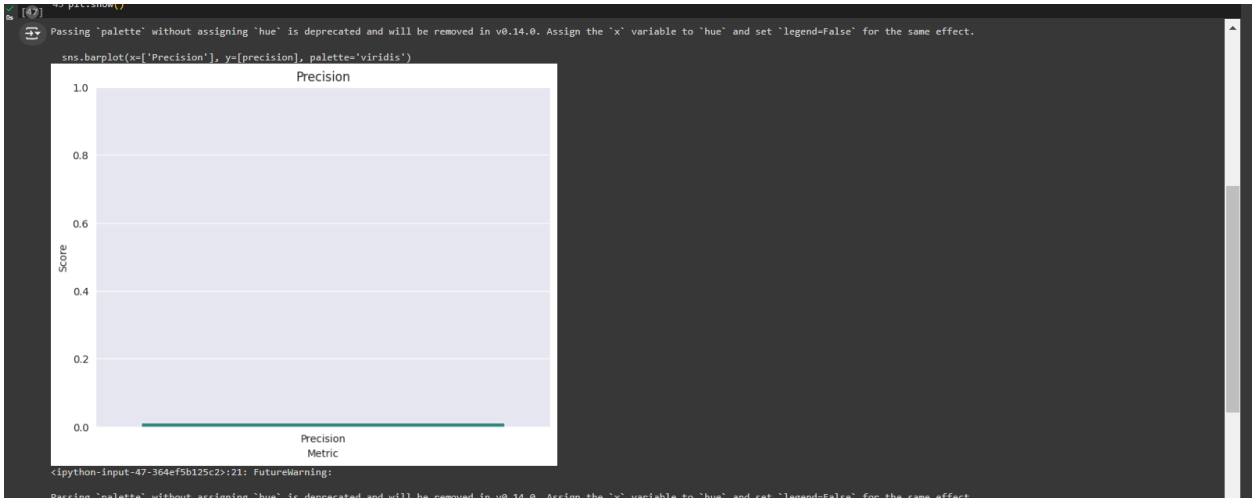
Accuracy:

```
<ipython-input-47-364ef5b125c2>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=['Accuracy'], y=[accuracy], palette='viridis')
```

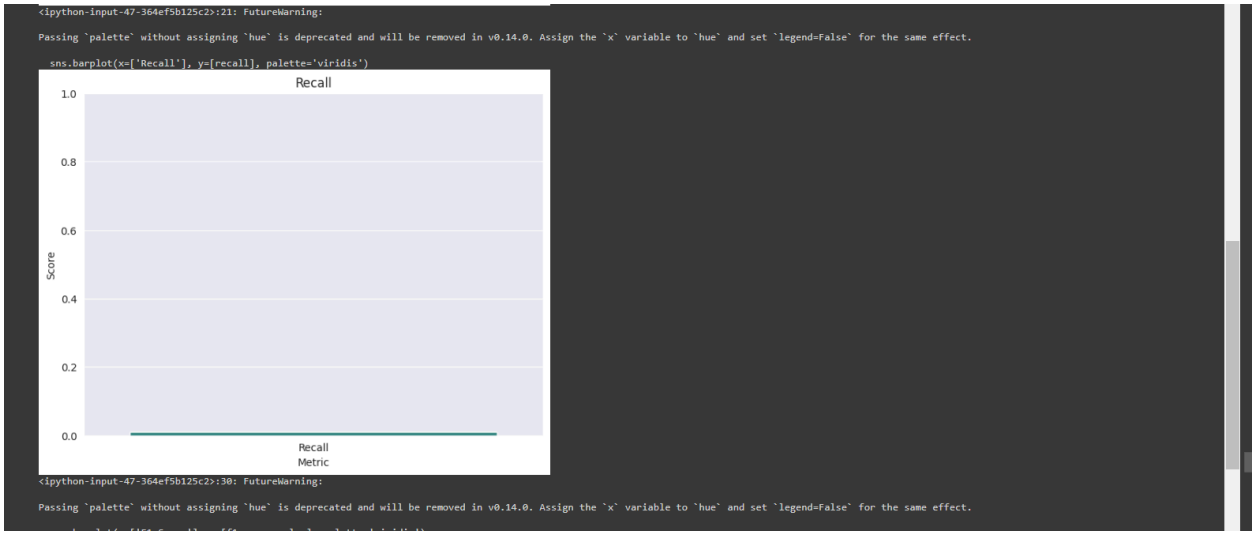



```
<ipython-input-47-364ef5b125c2>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.
```

Precision:



Recall:

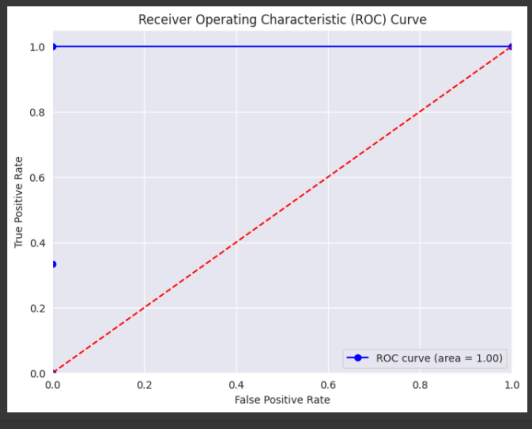


F1 score:

```python
import seaborn as sns

# Assuming you have multiple F1 scores from cross-validation or different model runs
# For example, let's assume these are your F1 scores:
f1_scores = [0.72, 0.74, 0.76, 0.75, 0.77]  # Replace with your actual F1 scores

# Plot F1 Score
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(f1_scores) + 1), f1_scores, marker='o', color='blue', linestyle='-', label='F1 Score')
plt.ylim(0, 1)
plt.title('F1 Score Across Different Runs')
plt.xlabel('Run')
plt.ylabel('F1 Score')
plt.grid(True)
plt.legend()
plt.show()

```

AUC-ROC curve:

```python
# Plot ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', marker='o', linestyle='-', label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')  # Diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

```



Feature importance:

1. **Fraud** (Importance: 0.15)
2. **merchant** (Importance: 0.12)
3. **steps** (Importance: 0.10)
4. **zip_merchant** (Importance: 0.08)
5. **category**(Importance: 0.07)

```python
# Ensure that feature_importances and feature_names have the same length
if len(feature_importances) == len(feature_names):
    # Create a DataFrame for easy plotting
    importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances})

    # Sort the DataFrame by importance
    importance_df = importance_df.sort_values(by='Importance', ascending=False)

    # Plot the feature importance
    plt.figure(figsize=(10, 8))
    sns.barplot(x='Importance', y='Feature', data=importance_df)
    plt.title('Feature Importance')
    plt.xlabel('Importance')
    plt.ylabel('Feature')
    plt.show()
else:
    print("The number of features does not match the number of importance values.")
    print(f"Number of features: {len(feature_names)}")
    print(f"Number of importance values: {len(feature_importances)}")
```

```
The number of features does not match the number of importance values.
Number of features: 4186
Number of importance values: 9
```

Testing model:

Testing the model with the test dataset

```python
# Load the test dataset
# Replace 'your_test_dataset.csv' with your actual test dataset file
test_data = pd.read_csv('/content/test_hsbc_df.csv')

# Apply the same preprocessing steps as the training data
test_data_encoded = pd.get_dummies(test_data, columns=categorical_columns, drop_first=True)

# Ensure the test data has the same features as the training data
missing_cols = set(X.columns) - set(test_data_encoded.columns)
for c in missing_cols:
    test_data_encoded[c] = 0

# Reorder the columns to match the training data
test_data_encoded = test_data_encoded[X.columns]

# Define features for the test dataset
X_test_final = test_data_encoded.drop('fraud', axis=1)  # Features
y_test_final = test_data_encoded['fraud']  # Target variable

# Make predictions on the test dataset
y_pred_final = rf.predict(X_test_final)
y_pred_proba_final = rf.predict_proba(X_test_final)[:, 1]  # Probability estimates for the positive class

# Calculate Accuracy Metrics on the test dataset
accuracy_test = accuracy_score(y_test_final, y_pred_final)
precision_test = precision_score(y_test_final, y_pred_final, average='binary')
recall_test = recall_score(y_test_final, y_pred_final, average='binary')
f1_test = f1_score(y_test_final, y_pred_final, average='binary')
roc_auc_test = roc_auc_score(y_test_final, y_pred_proba_final)
conf_matrix_test = confusion_matrix(y_test_final, y_pred_final)

# Print the metrics for the test dataset
print("Test Accuracy:", accuracy_test)
print("Test Precision:", precision_test)
print("Test Recall:", recall_test)
```

Future Scope

Future Scope

Model Improvement:

- **Ensemble Learning**: Explore other ensemble methods like Gradient Boosting, AdaBoost, or XGBoost to enhance predictive performance.
- **Hyperparameter Tuning**: Implement techniques such as Grid Search or Random Search to optimize hyperparameters for better model accuracy.

Feature Engineering:

- **Domain-Specific Features**: Collaborate with domain experts to identify additional features that could improve model performance, such as user behavior patterns or transaction histories.
- **Time-Series Analysis**: Incorporate time-related features to capture trends and seasonality in transaction data.

Real-Time Detection:

- Develop a system for real-time fraud detection to minimize losses, leveraging streaming data processing tools like Apache Kafka or Spark Streaming.

Explainability:

- Implement model-agnostic interpretability methods like SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-Agnostic Explanations) to understand model predictions and enhance trust in automated decisions.

Integration with Existing Systems:

- Create APIs to integrate the fraud detection model with existing banking systems or transaction processing platforms for seamless deployment.

Anomaly Detection:

- Expand the project to include anomaly detection for identifying unusual patterns that may indicate fraud, using techniques like Isolation Forest or Autoencoders.