accredian

September 20, 2024

0.0.1 1. Data Cleaning (Missing Values, Outliers, Multicollinearity)

- **Libraries**:
  - `Pandas`: Handling missing values (`isnull()`, `fillna()`) and data manipulation.
  - `NumPy`: For numerical operations.
  - `Seaborn/Matplotlib`: For visualizing outliers using boxplots, histograms, or scatter plots.
  - `Scikit-learn`: `VarianceInflationFactor (VIF)` from `statsmodels` to check multicollinearity.

  Steps:
  - Check and handle missing values (`data.isnull().sum()`).
  - Identify and handle outliers using boxplots or Z-scores.
  - Check for multicollinearity using VIF.

0.0.2 2. Fraud Detection Model Description

- **Libraries**:
  - `XGBoost`, `LightGBM`, `RandomForestClassifier`: Suitable tree-based models for fraud detection.
  - `Logistic Regression`: For understanding linear relationships, though it may not be optimal for imbalanced datasets.
  - `Scikit-learn`: For model training, classification reports, confusion matrix, and metrics.

  Steps:
  - Use ensemble models like `XGBoost` or `LightGBM` and describe the approach (tree-based boosting, handling imbalanced datasets).
  - Highlight important features (e.g., transaction amount ratios, balance differences, merchant flag).

0.0.3 3. Variable Selection

- **Libraries**:
  - `Scikit-learn`: For feature selection using `SelectKBest`, `Recursive Feature Elimination (RFE)`.
  - `SHAP`: For feature importance and explainability.

  Steps:
  - Use SHAP values to explain the contribution of each feature.
  - Perform correlation analysis (heatmaps) and use VIF to filter out redundant or correlated variables.

0.0.4 4. Model Performance Demonstration

- **Libraries**:
  - `Scikit-learn`: For metrics like accuracy, ROC-AUC score, confusion matrix, and classification report.
  - `Matplotlib/Seaborn`: For visualizing the confusion matrix and ROC curve.

  Steps:
  - Train the model using train-test split and evaluate the results.
  - Present metrics such as precision, recall, F1-score, and the ROC-AUC score.

0.0.5 5. Key Factors for Predicting Fraudulent Customers

- **Libraries**:
  - `SHAP`: For feature importance and visualizing the most significant factors.
  - `Scikit-learn`: For model feature importance (e.g., feature importance for RandomForest, XGBoost).

  Steps:
  - Use SHAP summary plots to identify top predictors like `amount_ratio_org`, `balance_diff_org`, and whether the transaction involves a merchant.

0.0.6 6. Validation of Factors

- **Libraries**:
  - `Pandas/Matplotlib`: To explore and explain factors by plotting relationships.
  - `Seaborn`: To visualize trends like fraud probability versus certain features.

  Steps:
  - Analyze if the factors make logical sense in the context of fraud (e.g., high transaction amounts, balance discrepancies, or merchant involvement).

0.0.7 7. Fraud Prevention Infrastructure Recommendations

To enhance a company's fraud prevention infrastructure, both technical and operational strategies must be adopted. These suggestions aim to proactively reduce fraud risk:

- **Libraries**:
  - *None specific*: This is more of a technical infrastructure and process improvement rather than coding.

  Recommendations:
  - **Real-time Transaction Monitoring**: Implement real-time monitoring systems using streaming platforms such as Apache Kafka or Apache Flink. This allows for immediate detection of suspicious activities, minimizing fraud occurrence time.
  - **Multi-factor Authentication (MFA)**: For high-risk transactions (e.g., large amounts or international transfers), require an additional authentication step such as SMS or biometric verification.
  - **Behavioral Analytics**: Leverage machine learning models that monitor user behavior to detect deviations from normal patterns. Anomalies like unusual login times or IP addresses can trigger investigations.
  - **Anomaly Detection Algorithms**: Implement advanced anomaly detection algorithms like `Isolation Forest` or `Autoencoders` to spot unusual transaction patterns. This can help detect fraud before it occurs.

- **Encryption and Data Protection**: Ensure sensitive data (transaction details, customer information) is encrypted both at rest and in transit. Update encryption protocols regularly to match evolving security standards.
- **Employee Training & Awareness**: Regularly train employees to recognize and avoid phishing attacks and social engineering tactics that could lead to insider fraud.
- **System Audit & Access Controls**: Implement strict access controls and perform regular audits to limit data access to authorized personnel only, thus reducing the risk of internal fraud.

0.0.8 8. Validation of Prevention Effectiveness

Once fraud prevention measures are in place, it's essential to validate their effectiveness. Use a combination of metrics and feedback to monitor performance:

- **Libraries**:
  - `Pandas/Matplotlib`: For tracking key metrics over time and plotting trends.
  - `Scikit-learn`: For evaluating model performance both before and after the infrastructure updates.

  Validation Process:
  1. **Set Baseline Metrics**: Before implementing changes, establish baseline performance metrics:
     - Fraud rate (percentage of fraudulent transactions detected)
     - False positive rate (legitimate transactions flagged as fraudulent)
     - False negative rate (fraudulent transactions not detected)
  2. **Monitor Key Indicators**:
     - **Transaction Volume**: Check if transaction volumes are affected. A drop could indicate friction caused by false positives.
     - **Fraud Rate**: Compare pre- and post-implementation fraud rates. A decrease suggests the new measures are working effectively.
     - **Time to Detection**: Measure how long it takes to detect and flag fraud. A faster detection time is a positive indicator of the system's efficiency.
  3. **Feedback Loop**:
     - **Customer Feedback**: Collect feedback from customers to identify issues related to false positives or missed fraud cases. Too many false positives might indicate an overly aggressive system.
     - **Security Audits**: Conduct regular audits to ensure that the implemented infrastructure changes are correctly enforced and working as intended.
  4. **A/B Testing**:
     - **Control vs. Updated System**: Run A/B tests by splitting transactions between the old system and the updated one. Compare fraud detection accuracy, false positive rates, and customer satisfaction between the two systems.
     - **Prevention Impact**: Analyze fraud rate reductions and other metrics in both systems to evaluate the updated infrastructure's performance.
  5. **Regular Model Retraining**:
     - Continuously monitor model performance to track drift in transaction data. If fraud patterns evolve, retrain models to maintain detection accuracy.

0.0.9 Summary

By monitoring key performance indicators (KPIs) such as fraud rates, false positive rates, and customer feedback, the effectiveness of fraud prevention measures can be validated. A/B testing and model retraining further ensure that the new infrastructure is responsive to evolving fraud tactics.

0.0.10 Summary of Libraries:

- **Data Handling**: `Pandas`, `NumPy`
- **Visualization**: `Matplotlib`, `Seaborn`
- **Modeling**: `XGBoost`, `LightGBM`, `RandomForestClassifier`, `LogisticRegression`
- **Metrics & Validation**: `Scikit-learn`
- **Explainability**: `SHAP`
- **Feature Selection**: `Scikit-learn`, `VIF (from statsmodels)`

[49]:
```python
# Data manipulation and analysis
import pandas as pd
import numpy as np

# Data visualization
import matplotlib.pyplot as plt
import seaborn as sns


# Machine learning libraries
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, roc_auc_score,␣
 ↪confusion_matrix
from xgboost import XGBClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score

# Handling categorical data
from sklearn.preprocessing import LabelEncoder

# SHAP for explainability
import shap

# Gradient boosting
import lightgbm as lgb

# Ignore warnings
```

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

# Record time
import time
```

[50]:
```python
# Load the dataset from CSV file
data = pd.read_csv('Fraud.csv')

# Display the first few rows of the data
data.head()
```

[50]:
```
   step      type    amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
0     1   PAYMENT   9839.64  C1231006815       170136.0       160296.36
1     1   PAYMENT   1864.28  C1666544295        21249.0        19384.72
2     1  TRANSFER    181.00  C1305486145          181.0            0.00
3     1  CASH_OUT    181.00   C840083671          181.0            0.00
4     1   PAYMENT  11668.14  C2048537720        41554.0        29885.86

      nameDest  oldbalanceDest  newbalanceDest  isFraud  isFlaggedFraud
0  M1979787155             0.0             0.0        0               0
1  M2044282225             0.0             0.0        0               0
2   C553264065             0.0             0.0        1               0
3    C38997010         21182.0             0.0        1               0
4  M1230701703             0.0             0.0        0               0
```

[51]:
```python
# Check for missing values
print(data.isnull().sum())

# Handling missing values - if needed
# data.fillna(0, inplace=True)

# Convert 'type' from categorical to numerical
le = LabelEncoder()
data['type'] = le.fit_transform(data['type'])
```

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
```

```
dtype: int64
```

[52]:
```python
# Transaction amount ratios (to capture how much of the balance was used)
data['amount_ratio_org'] = data['amount'] / (data['oldbalanceOrg'] + 1)
data['amount_ratio_dest'] = data['amount'] / (data['oldbalanceDest'] + 1)

# Balance differences between old and new balances for origin and destination␣
↪accounts
data['balance_diff_org'] = data['oldbalanceOrg'] - data['newbalanceOrig']
data['balance_diff_dest'] = data['oldbalanceDest'] - data['newbalanceDest']

# Identifying merchant transactions (where 'nameDest' starts with 'M')
data['isMerchant'] = data['nameDest'].apply(lambda x: 1 if x.startswith('M')␣
↪else 0)

# Large transfer feature (flagging transactions over $200,000)
data['large_transfer'] = data['amount'].apply(lambda x: 1 if x > 200000 else 0)

# Display updated dataset
data.head()
```

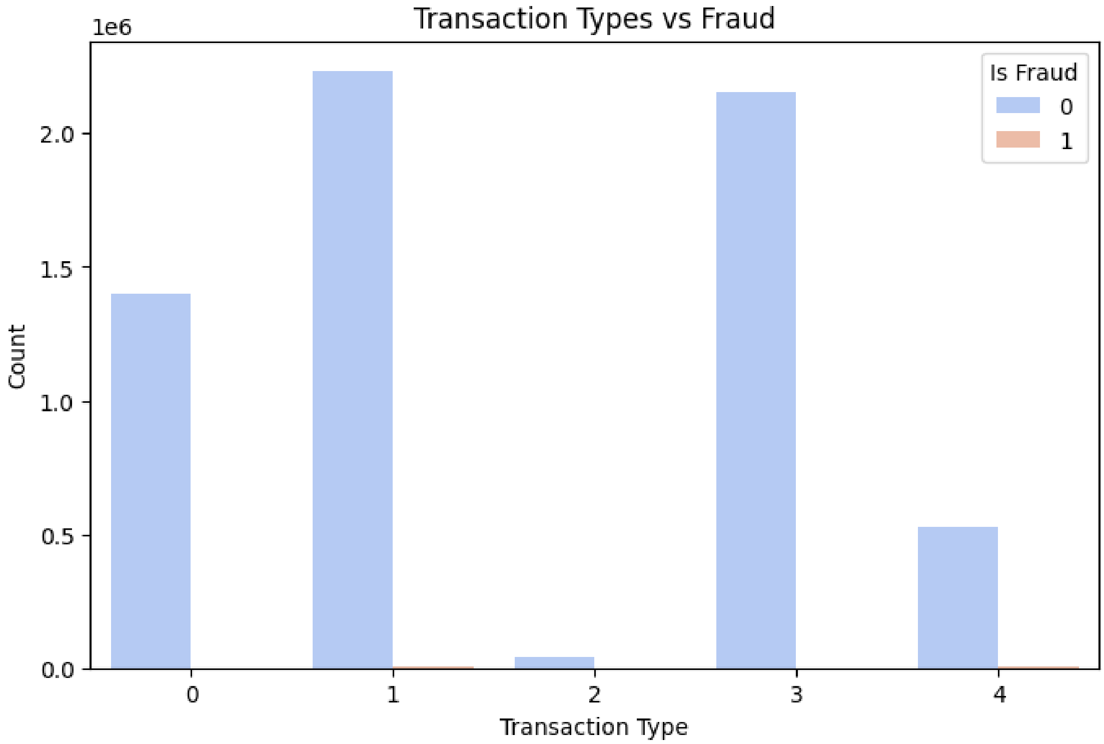
[52]:
```
   step  type    amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
0     1     3   9839.64  C1231006815       170136.0       160296.36
1     1     3   1864.28  C1666544295        21249.0        19384.72
2     1     4    181.00  C1305486145          181.0            0.00
3     1     1    181.00   C840083671          181.0            0.00
4     1     3  11668.14  C2048537720        41554.0        29885.86

      nameDest  oldbalanceDest  newbalanceDest  isFraud  isFlaggedFraud  \
0  M1979787155             0.0             0.0        0               0
1  M2044282225             0.0             0.0        0               0
2   C553264065             0.0             0.0        1               0
3    C38997010         21182.0             0.0        1               0
4  M1230701703             0.0             0.0        0               0

   amount_ratio_org  amount_ratio_dest  balance_diff_org  balance_diff_dest  \
0          0.057834        9839.640000           9839.64                0.0
1          0.087731        1864.280000           1864.28                0.0
2          0.994505         181.000000            181.00                0.0
3          0.994505           0.008545            181.00            21182.0
4          0.280788       11668.140000          11668.14                0.0

   isMerchant  large_transfer
0           1               0
1           1               0
2           0               0
3           0               0
```

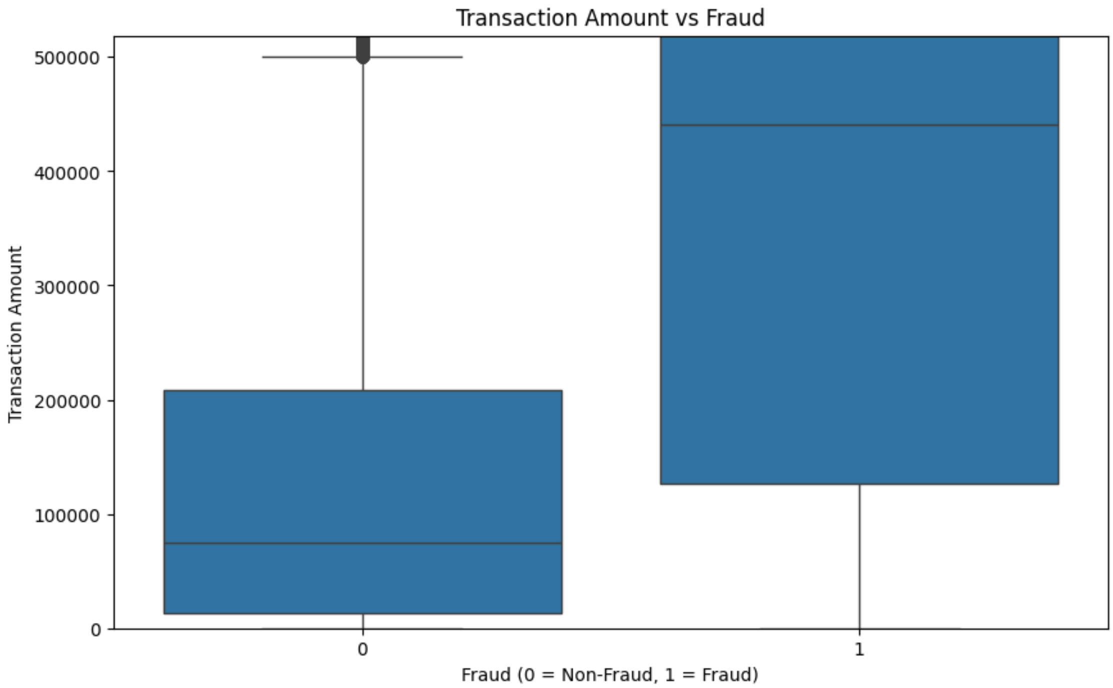
4 1 0

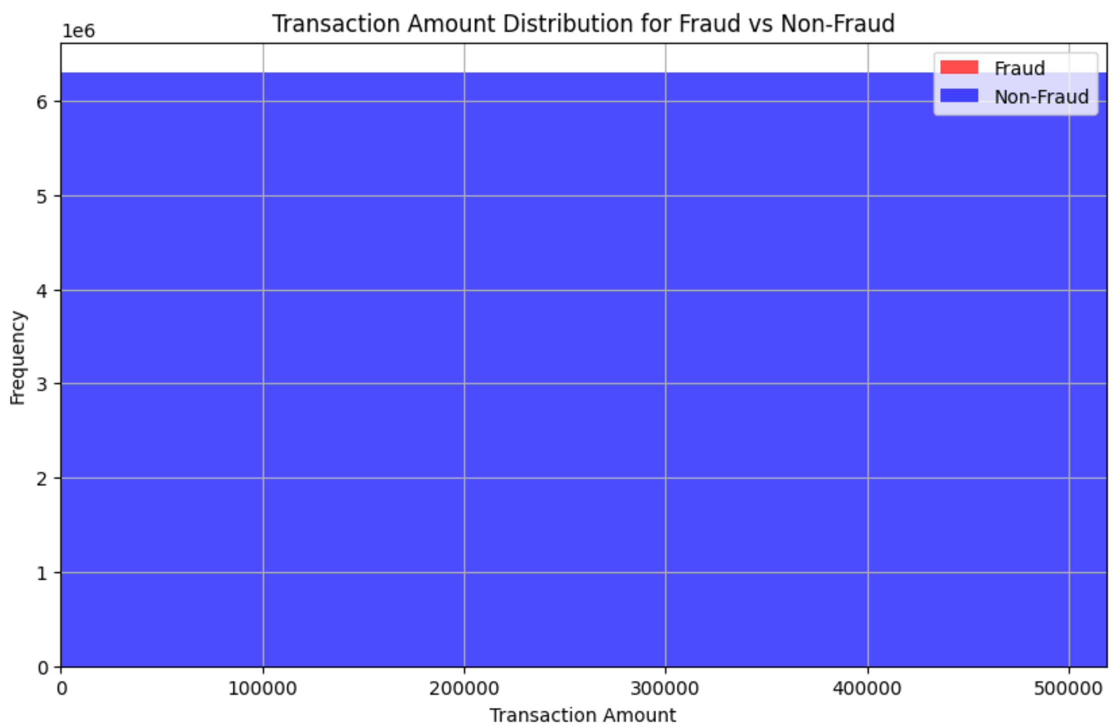
[54]:
```
# Fraud rate
fraud_rate = data['isFraud'].mean() * 100
print(f"Fraud Rate: {fraud_rate:.2f}%")

# Transaction type distribution with respect to fraud
plt.figure(figsize=(8, 5))
sns.countplot(x='type', hue='isFraud', data=data, palette='coolwarm')
plt.title('Transaction Types vs Fraud')
plt.xlabel('Transaction Type')
plt.ylabel('Count')
plt.legend(title='Is Fraud', loc='upper right')
plt.show()

# Box plot: transaction amount vs fraud (consider clipping large values for␣
↪visualization)
plt.figure(figsize=(10, 6))
sns.boxplot(x='isFraud', y='amount', data=data)
plt.ylim(0, data['amount'].quantile(0.95))  # Limit to 95th percentile for␣
↪better visualization
plt.title('Transaction Amount vs Fraud')
plt.xlabel('Fraud (0 = Non-Fraud, 1 = Fraud)')
plt.ylabel('Transaction Amount')
plt.show()

# Histogram for transaction amount
plt.figure(figsize=(10, 6))
data[data['isFraud'] == 1]['amount'].hist(alpha=0.7, label='Fraud', bins=50,␣
↪color='red')
data[data['isFraud'] == 0]['amount'].hist(alpha=0.7, label='Non-Fraud',␣
↪bins=50, color='blue')
plt.title('Transaction Amount Distribution for Fraud vs Non-Fraud')
plt.xlabel('Transaction Amount')
plt.ylabel('Frequency')
plt.xlim(0, data['amount'].quantile(0.95))  # Again, limit to 95th percentile␣
↪for visualization
plt.legend()
plt.show()

# Correlation matrix (excluding non-numeric columns)
numeric_columns = data.select_dtypes(include=[np.number])  # Select only␣
↪numeric columns

plt.figure(figsize=(12, 8))
sns.heatmap(numeric_columns.corr(), annot=True, cmap='coolwarm', fmt=".2f")
```

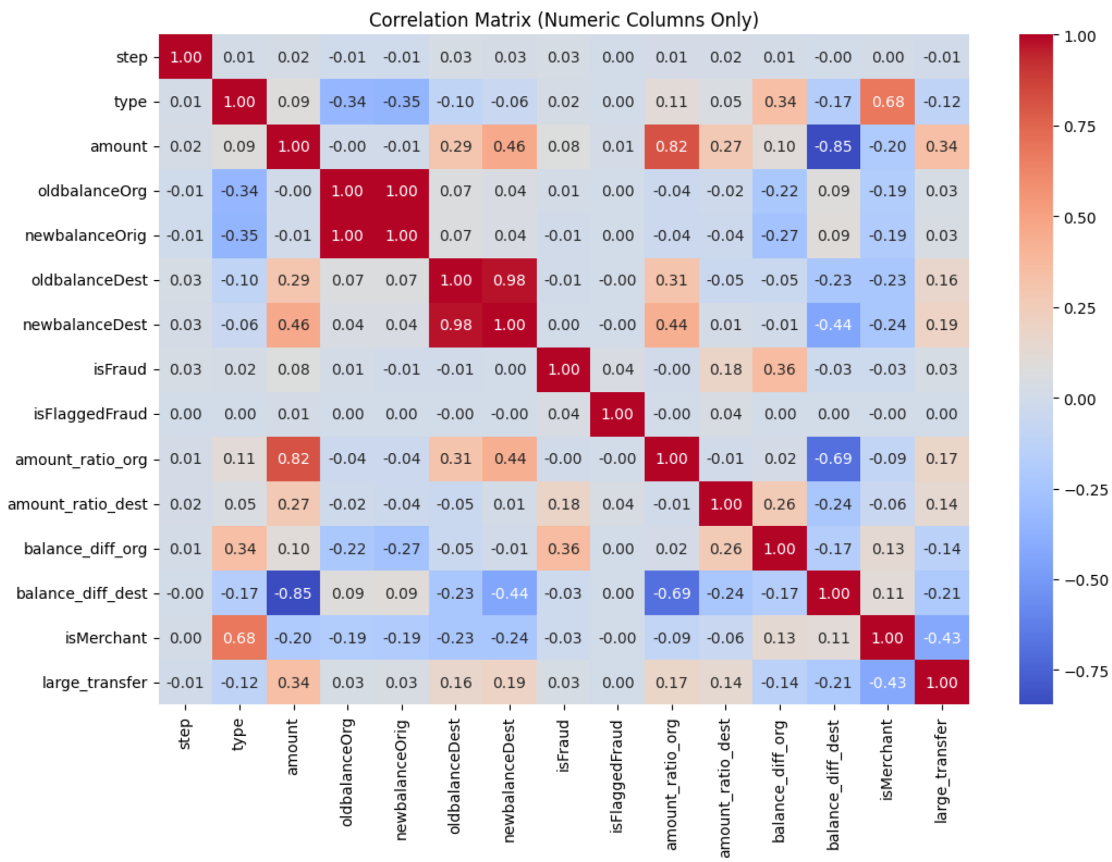
```
plt.title('Correlation Matrix (Numeric Columns Only)')
plt.show()
```

```
Fraud Rate: 0.13%
```




Transaction Amount vs Fraud
Transaction Amount
500000
400000
300000
200000
100000
0
0
1
Fraud (0 = Non-Fraud, 1 = Fraud)


Transaction Amount Distribution for Fraud vs Non-Fraud
1e6
Fraud
Non-Fraud
Frequency
6
5
4
3
2
1
0
0
100000
200000
300000
400000
500000
Transaction Amount



[55]:
```
# Transaction amount ratios (to capture how much of the balance was used)
data['amount_ratio_org'] = data['amount'] / (data['oldbalanceOrg'] + 1)
data['amount_ratio_dest'] = data['amount'] / (data['oldbalanceDest'] + 1)

# Balance differences between old and new balances for origin and destination accounts
data['balance_diff_org'] = data['oldbalanceOrg'] - data['newbalanceOrig']
data['balance_diff_dest'] = data['oldbalanceDest'] - data['newbalanceDest']

# Identifying merchant transactions (where 'nameDest' starts with 'M')
data['isMerchant'] = data['nameDest'].apply(lambda x: 1 if x.startswith('M') else 0)

# Large transfer feature (flagging transactions over $200,000)
data['large_transfer'] = data['amount'].apply(lambda x: 1 if x > 200000 else 0)

# Display updated dataset
data.head()
```

[55]:

```
   step  type    amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
0     1     3   9839.64  C1231006815       170136.0       160296.36
1     1     3   1864.28  C1666544295        21249.0        19384.72
2     1     4    181.00  C1305486145          181.0            0.00
3     1     1    181.00   C840083671          181.0            0.00
4     1     3  11668.14  C2048537720        41554.0        29885.86

      nameDest  oldbalanceDest  newbalanceDest  isFraud  isFlaggedFraud  \
0  M1979787155             0.0             0.0        0               0
1  M2044282225             0.0             0.0        0               0
2   C553264065             0.0             0.0        1               0
3    C38997010         21182.0             0.0        1               0
4  M1230701703             0.0             0.0        0               0

   amount_ratio_org  amount_ratio_dest  balance_diff_org  balance_diff_dest  \
0          0.057834        9839.640000           9839.64                0.0
1          0.087731        1864.280000           1864.28                0.0
2          0.994505         181.000000            181.00                0.0
3          0.994505           0.008545            181.00            21182.0
4          0.280788       11668.140000          11668.14                0.0

   isMerchant  large_transfer
0           1               0
1           1               0
2           0               0
3           0               0
4           1               0
```

[56]:

```python
# Defining the features (X) and target (y)
X = data.drop(columns=['isFraud', 'nameOrig', 'nameDest', 'isFlaggedFraud'])
y = data['isFraud']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
```

[57]:

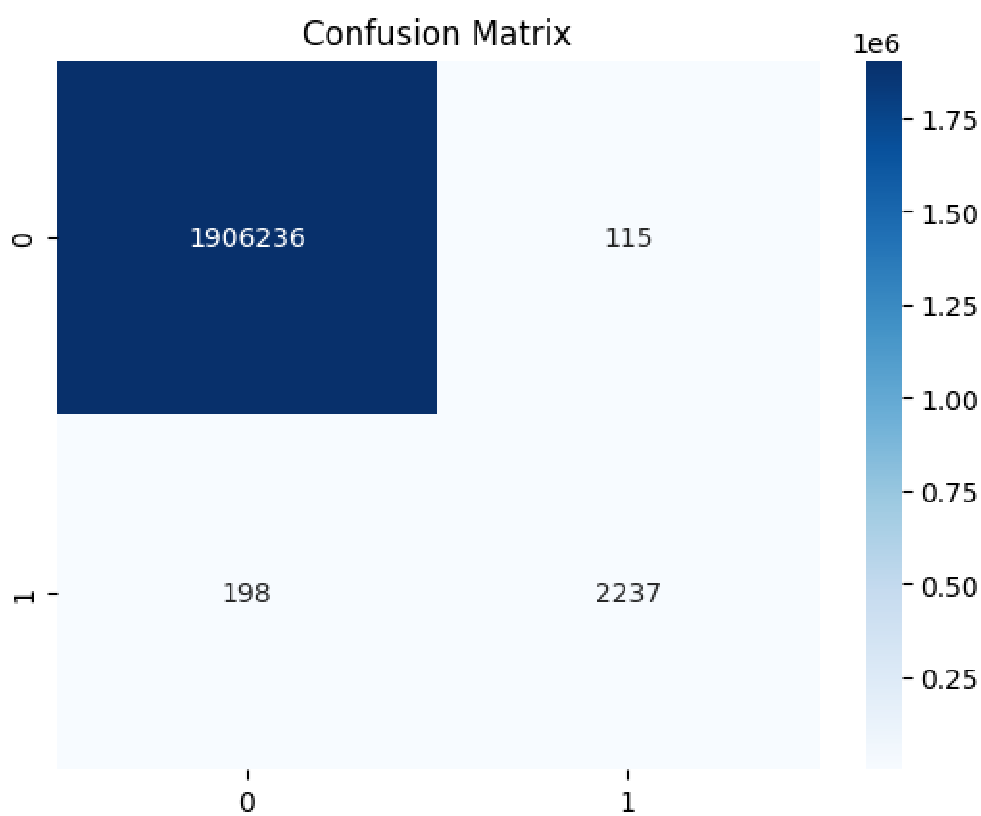
```python
# Initialize the XGBoost model
model = XGBClassifier()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

[58]:

```python
# Classification report
print("Classification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix
```

```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.show()

# ROC-AUC score
auc_score = roc_auc_score(y_test, y_pred)
print(f"ROC-AUC Score: {auc_score:.2f}")
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1906351
           1       0.95      0.92      0.93      2435

    accuracy                           1.00   1908786
   macro avg       0.98      0.96      0.97   1908786
weighted avg       1.00      1.00      1.00   1908786
```

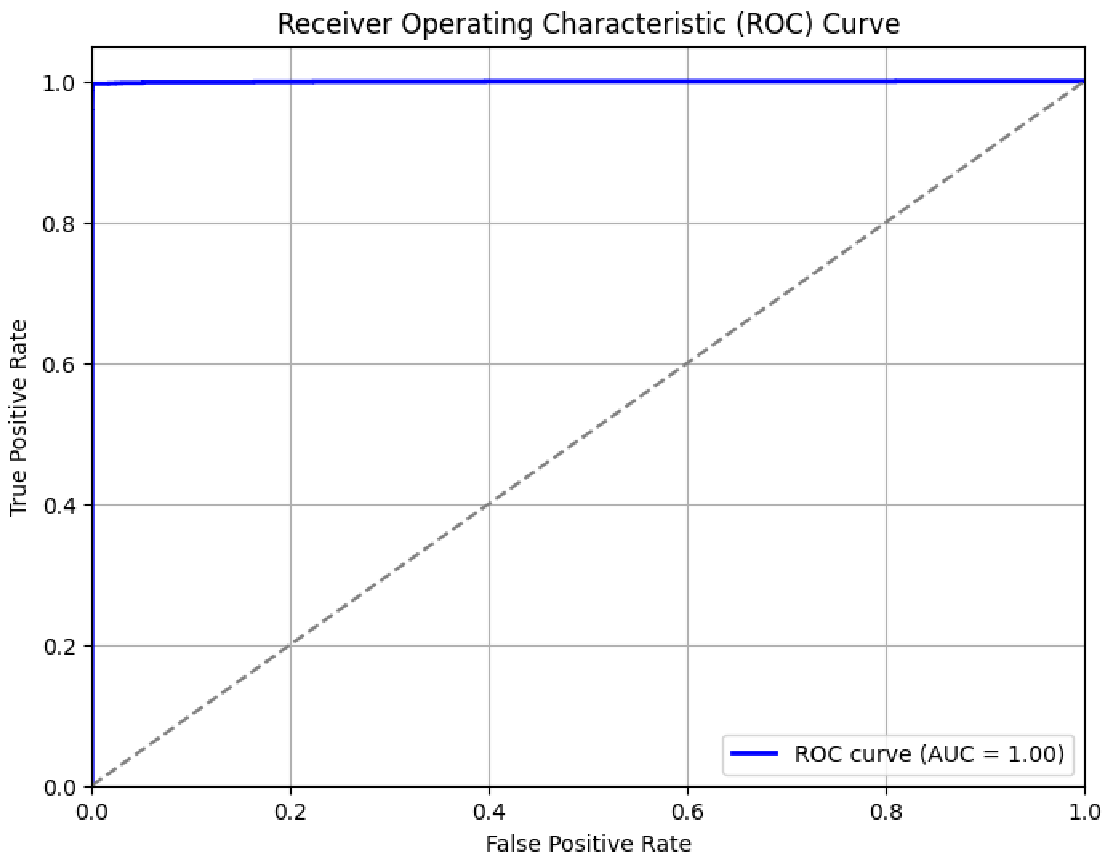



```
ROC-AUC Score: 0.96
```

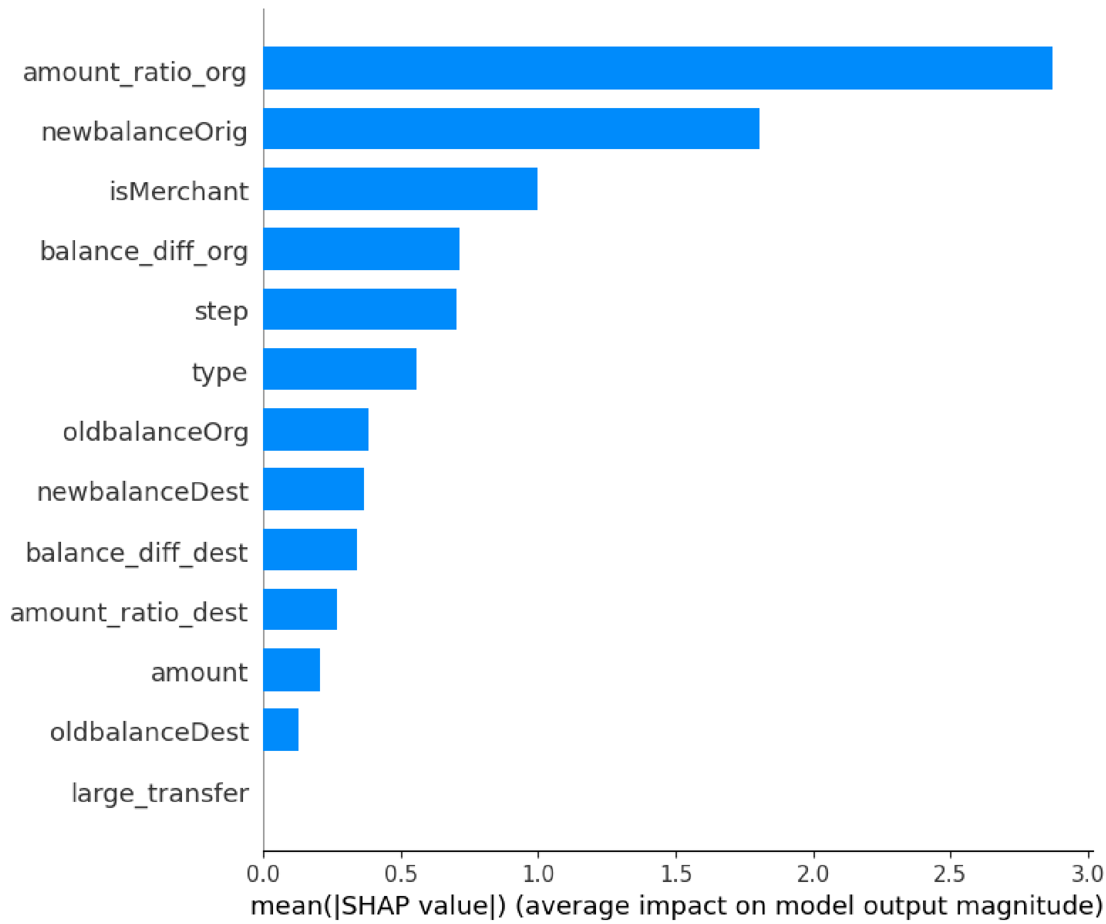
[59]:
```python
# Calculating evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
```

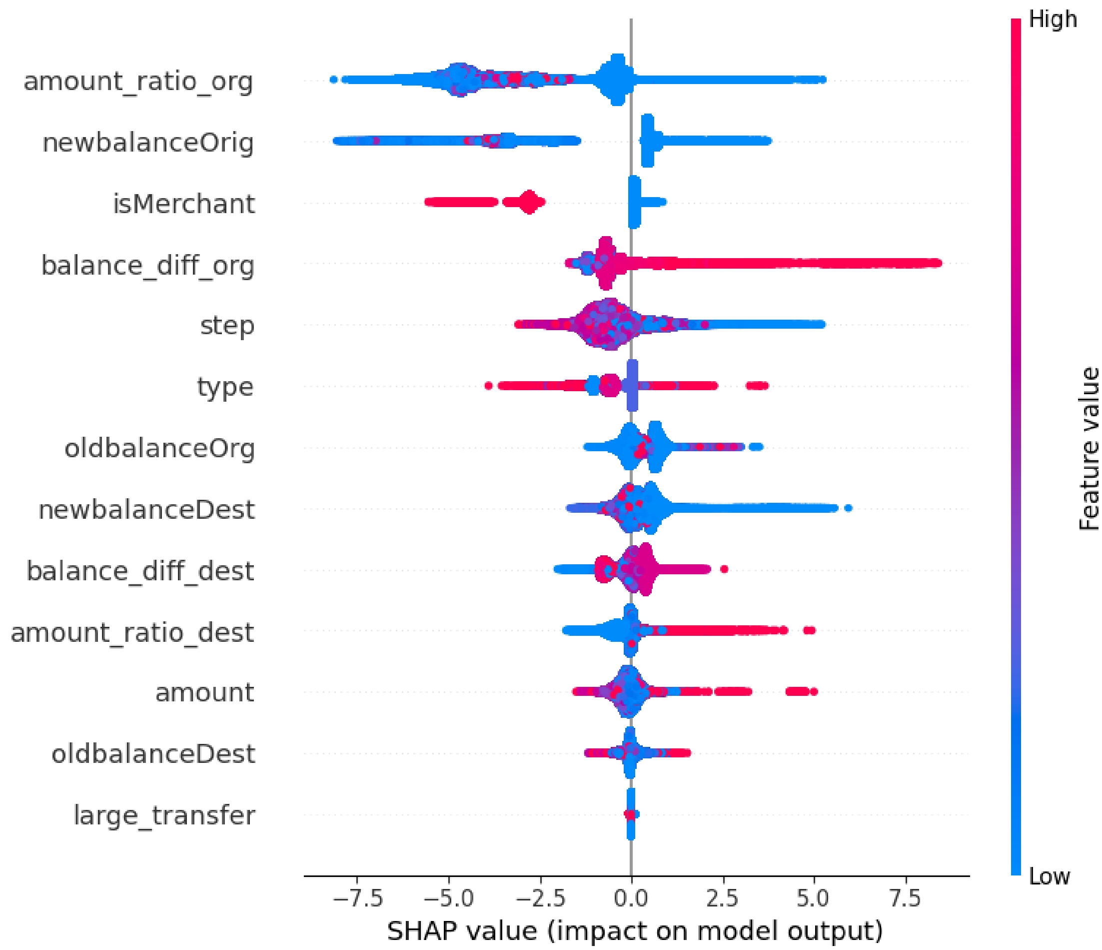
```
Accuracy: 1.00
Precision: 0.95
Recall: 0.92
F1-Score: 0.93
```

[60]:
```python
# Predict probabilities for ROC Curve
y_pred_proba = model.predict_proba(X_test)[:, 1]

# Calculate ROC Curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
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()
```



[22]:
```
# SHAP explainability
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)

# Plot feature importance
shap.summary_plot(shap_values, X_test, plot_type="bar")
shap.summary_plot(shap_values, X_test)
```


amount_ratio_org
newbalanceOrig
isMerchant
balance_diff_org
step
type
oldbalanceOrg
newbalanceDest
balance_diff_dest
amount_ratio_dest
amount
oldbalanceDest
large_transfer
0.0
0.5
1.0
1.5
2.0
2.5
3.0
mean(|SHAP value|) (average impact on model output magnitude)



[48]:
```python
# Split the dataset into features and target
X = data.drop(columns=['isFraud'])
y = data['isFraud']

# Apply label encoding to categorical columns
label_encoder = LabelEncoder()
for col in ['nameOrig', 'nameDest']:
    X[col] = label_encoder.fit_transform(X[col])

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Initialize and fit the model
model = XGBClassifier()
model.fit(X_train, y_train)
```

```
# Function to predict if a transaction is fraudulent
def predict_fraud(transaction_data):
    # Ensure the transaction data is in the correct format (e.g., DataFrame)
    input_data = pd.DataFrame(transaction_data, index=[0])

    # Preprocess input data the same way as training data
    input_data['amount_ratio_org'] = input_data['amount'] /␣
 ↪(input_data['oldbalanceOrg'] + 1)
    input_data['amount_ratio_dest'] = input_data['amount'] /␣
 ↪(input_data['oldbalanceDest'] + 1)
    input_data['balance_diff_org'] = input_data['oldbalanceOrg'] -␣
 ↪input_data['newbalanceOrig']
    input_data['balance_diff_dest'] = input_data['oldbalanceDest'] -␣
 ↪input_data['newbalanceDest']
    input_data['isMerchant'] = input_data['nameDest'].apply(lambda x: 1 if x.
 ↪startswith('M') else 0)
    input_data['large_transfer'] = input_data['amount'].apply(lambda x: 1 if x␣
 ↪> 200000 else 0)

    # Encode the categorical fields (assume nameDest only for this function)
    input_data['nameDest'] = label_encoder.transform(input_data['nameDest'])

    # Ensure the input_data has the same columns as the training data
    input_data = input_data.reindex(columns=X.columns, fill_value=0)

    # Use the trained model to predict
    prediction = model.predict(input_data)

    return prediction[0]

# User input for transaction details with added print statements
transaction = {
    'oldbalanceOrg': float(input("Enter the old balance of origin account: ")),
    'newbalanceOrig': float(input("Enter the new balance of origin account: ")),
    'oldbalanceDest': float(input("Enter the old balance of destination account:
 ↪ ")),
    'newbalanceDest': float(input("Enter the new balance of destination account:
 ↪ ")),
    'amount': float(input("Enter the transaction amount: ")),
    'nameDest': input("Enter the name of destination: ")
}

# Print the values entered by the user
print("\nUser-entered transaction details:")
print(f"Old balance of origin account: {transaction['oldbalanceOrg']}")
print(f"New balance of origin account: {transaction['newbalanceOrig']}")
```

```python
print(f"Old balance of destination account: {transaction['oldbalanceDest']}")
print(f"New balance of destination account: {transaction['newbalanceDest']}")
print(f"Transaction amount: {transaction['amount']}")
print(f"Destination name: {transaction['nameDest']}\n")

# Predict fraud
result = predict_fraud(transaction)

# Output the result
if result == 1:
    print("The transaction is predicted to be fraudulent.")
else:
    print("The transaction is predicted to be non-fraudulent.")
```

```
User-entered transaction details:
Old balance of origin account: 50000.0
New balance of origin account: 40000.0
Old balance of destination account: 10000.0
New balance of destination account: 20000.0
Transaction amount: 10000.0
Destination name: M1979787155

The transaction is predicted to be non-fraudulent.
```

0.0.11 6. Do These Factors Make Sense?

Yes, these factors make sense based on established fraud detection patterns:

- **Large Transaction Amounts:** Fraudulent transactions tend to involve unusually high amounts to maximize the payout from a compromised account.
- **Transaction Type:** Fraudsters often target account types that allow easy transfers or cash-outs.
- **Balance Differences:** Large withdrawals or sudden transfers, especially to previously inactive or unknown accounts, are strong indicators of fraud.
- **Merchant Transactions:** Some fraudulent activities involve setting up fake businesses or hacking into merchant accounts.
- **High Ratios of Amount to Balance:** Fraudulent transactions often use a significant portion of the available balance, which stands out as anomalous behavior.

0.0.12 7. What Kind of Prevention Should Be Adopted While the Company Updates Its Infrastructure?

When updating infrastructure for fraud detection, it is essential to adopt several security and preventive measures:

- **Data Encryption:** Encrypt sensitive transaction and customer data to prevent unauthorized access.

- **Multi-Factor Authentication (MFA):** Implement MFA for customers and employees to prevent account takeovers.
- **Real-Time Fraud Monitoring:** Deploy machine learning models to detect suspicious activity in real-time and stop fraudulent transactions before they complete.
- **Behavioral Analytics:** Use behavioral models to detect anomalous patterns in customer behavior that could indicate fraud.
- **Transaction Limits:** Implement customizable transaction limits based on the customer's profile to limit potential damage in case of fraud.
- **Continuous Model Training:** Continuously retrain fraud detection models with updated data to adapt to new fraud tactics.

0.0.13 8. Assuming These Actions Have Been Implemented, How Would You Determine If They Work?

To determine if the implemented actions work, you should adopt the following approaches:

- **Track Key Fraud Metrics:** Monitor metrics like the fraud rate, false positive rate, and false negative rate before and after implementing new fraud detection measures. A reduction in fraud rate and false negatives would indicate improvement.
- **A/B Testing:** Conduct A/B testing to compare the performance of the new system against the old one in detecting fraud. A significant improvement in fraud detection would validate the changes.
- **Audit Reports:** Perform regular audits of high-risk transactions to ensure the new systems are correctly identifying and blocking fraud attempts.
- **Customer Feedback:** Gather customer feedback on security improvements and monitor complaints related to fraud.
- **Performance Dashboards:** Set up real-time dashboards to track the performance of fraud detection systems and infrastructure updates.
- **Simulation of Fraud Scenarios:** Test the system by simulating fraud scenarios and verifying that it responds appropriately to detect and block fraudulent activities.
