CreditCard Fraud Detection

In [1]:
```
# Step 1 - Import all the libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
```

In [2]:
```
df=pd.read_csv(r"C:\Users\swaro\OneDrive\Desktop\data science swaroop ky 1\Project
print(df)
```

```
         step      type      amount     nameOrig  oldbalanceOrg  \
0           1   PAYMENT     9839.64  C1231006815      170136.00
1           1   PAYMENT     1864.28  C1666544295       21249.00
2           1  TRANSFER      181.00  C1305486145         181.00
3           1  CASH_OUT      181.00   C840083671         181.00
4           1   PAYMENT    11668.14  C2048537720       41554.00
...       ...       ...         ...          ...            ...
6362615   743  CASH_OUT   339682.13   C786484425      339682.13
6362616   743  TRANSFER  6311409.28  C1529008245     6311409.28
6362617   743  CASH_OUT  6311409.28  C1162922333     6311409.28
6362618   743  TRANSFER   850002.52  C1685995037      850002.52
6362619   743  CASH_OUT   850002.52  C1280323807      850002.52

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud  \
0             160296.36  M1979787155            0.00            0.00        0
1              19384.72  M2044282225            0.00            0.00        0
2                  0.00   C553264065            0.00            0.00        1
3                  0.00    C38997010        21182.00            0.00        1
4              29885.86  M1230701703            0.00            0.00        0
...                 ...          ...             ...             ...      ...
6362615            0.00   C776919290            0.00       339682.13        1
6362616            0.00  C1881841831            0.00            0.00        1
6362617            0.00  C1365125890        68488.84      6379898.11        1
6362618            0.00  C2080388513            0.00            0.00        1
6362619            0.00   C873221189      6510099.11      7360101.63        1

         isFlaggedFraud
0                     0
1                     0
2                     0
3                     0
4                     0
...                 ...
6362615               0
6362616               0
6362617               0
6362618               0
6362619               0

[6362620 rows x 11 columns]
```

In [3]:
```
df=pd.read_csv(r"C:\Users\swaro\OneDrive\Desktop\data science swaroop ky 1\Project
df2=pd.DataFrame(df)
label_encoder = LabelEncoder()
# Iterate through all columns in the dataframe
for col in df2.columns:
    if df2[col].dtype == 'object':  # Check if the column is of object type
        df2[col] = label_encoder.fit_transform(df2[col])
print(df2)
```

```
         step  type      amount  nameOrig  oldbalanceOrg  newbalanceOrig  \
0           1     3     9839.64    757869      170136.00       160296.36
1           1     3     1864.28   2188998       21249.00        19384.72
2           1     4      181.00   1002156         181.00            0.00
3           1     1      181.00   5828262         181.00            0.00
4           1     3    11668.14   3445981       41554.00        29885.86
...       ...   ...         ...       ...            ...             ...
6362615   743     1   339682.13   5651847      339682.13            0.00
6362616   743     4  6311409.28   1737278     6311409.28            0.00
6362617   743     1  6311409.28    533958     6311409.28            0.00
6362618   743     4   850002.52   2252932      850002.52            0.00
6362619   743     1   850002.52    919229      850002.52            0.00

         nameDest  oldbalanceDest  newbalanceDest  isFraud  isFlaggedFraud
0         1662094            0.00            0.00        0               0
1         1733924            0.00            0.00        0               0
2          439685            0.00            0.00        1               0
3          391696        21182.00            0.00        1               0
4          828919            0.00            0.00        0               0
...           ...             ...             ...      ...             ...
6362615    505863            0.00       339682.13        1               0
6362616    260949            0.00            0.00        1               0
6362617    108224        68488.84      6379898.11        1               0
6362618    319713            0.00            0.00        1               0
6362619    534595      6510099.11      7360101.63        1               0

[6362620 rows x 11 columns]
```

In [4]:
```
df2.head()
```

Out[4]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalance
0	1	3	9839.64	757869	170136.0	160296.36	1662094	
1	1	3	1864.28	2188998	21249.0	19384.72	1733924	
2	1	4	181.00	1002156	181.0	0.00	439685	
3	1	1	181.00	5828262	181.0	0.00	391696	21
4	1	3	11668.14	3445981	41554.0	29885.86	828919	

In [5]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            int32
 2   amount          float64
 3   nameOrig        int32
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        int32
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64
 10  isFlaggedFraud  int64
dtypes: float64(5), int32(3), int64(3)
memory usage: 461.2 MB
```

In [6]:
```
# step 2  - checking the missing value
df2.isnull().sum()
```

Out[6]:
```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

In [7]:
```
# Step 4 - distribution of legit and fraudulent transaction
df2['isFraud'].value_counts()
```

Out[7]:
```
isFraud
0    6354407
1       8213
Name: count, dtype: int64
```

In [8]:
```
#This data set is highly unbalanced, The label 0 = normal transaction, 1=fradulant
```

In [9]:
```
# step 5 - creating a new data frame
df3 = df2[df2['isFraud'] == 0]
df4 = df2[df2['isFraud'] == 1]
```

In [10]:
```
df3.shape
```

Out[10]:
```
(6354407, 11)
```

In [11]:
```
df4.shape
```

Out[11]: (8213, 11)

In [12]:
```
# step6:Statistical measure of the data
df3.amount.describe()
```

Out[12]:
```
count    6.354407e+06
mean     1.781970e+05
std      5.962370e+05
min      1.000000e-02
25%      1.336840e+04
50%      7.468472e+04
75%      2.083648e+05
max      9.244552e+07
Name: amount, dtype: float64
```

In [13]:
```
df4.amount.describe()
```

Out[13]:
```
count    8.213000e+03
mean     1.467967e+06
std      2.404253e+06
min      0.000000e+00
25%      1.270913e+05
50%      4.414234e+05
75%      1.517771e+06
max      1.000000e+07
Name: amount, dtype: float64
```

In [14]:
```
#step7
# Under sampling
# Build a sample dataset containing a similar distribution of normal transactions a

df5=df3.sample(n= 8213)
```

In [15]:
```
# step 8 Concatenating two data frame
df6 = pd.concat([df4, df5], axis=0)
```

In [16]:
```
df6.head()
```

Out[16]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalan
2	1	4	181.0	1002156	181.0	0.0	439685	
3	1	1	181.0	5828262	181.0	0.0	391696	2
251	1	4	2806.0	1379875	2806.0	0.0	563886	
252	1	1	2806.0	3619815	2806.0	0.0	2134	2
680	1	4	20128.0	1232211	20128.0	0.0	251089	

In [17]:
```
# Step 9 - Checking distribution of legit and fraudulent transaction
df6['isFraud'].value_counts()
```

Out[17]:
```
isFraud
1    8213
0    8213
Name: count, dtype: int64
```

In [18]:
```
# step 10
# spliting the data into features and target
X=df6.drop(columns='isFraud',axis=1)
Y=df6['isFraud']
```

In [19]:
```
print(X)
```

```
         step  type     amount  nameOrig  oldbalanceOrg  newbalanceOrig  \
2           1     4     181.00   1002156         181.00            0.00
3           1     1     181.00   5828262         181.00            0.00
251         1     4    2806.00   1379875        2806.00            0.00
252         1     1    2806.00   3619815        2806.00            0.00
680         1     4   20128.00   1232211       20128.00            0.00
...       ...   ...        ...       ...            ...             ...
2111092   183     3    8624.04   2942145           0.00            0.00
2466311   203     4  344290.83   1478060           0.00            0.00
3832869   282     1  124174.25   3551676           0.00            0.00
4997230   352     0  199580.83   5305907      555743.85       755324.67
2102295   182     3   17082.40   5877535      279417.30       262334.90

         nameDest  oldbalanceDest  newbalanceDest  isFlaggedFraud
2          439685            0.00            0.00               0
3          391696        21182.00            0.00               0
251        563886            0.00            0.00               0
252          2134        26202.00            0.00               0
680        251089            0.00            0.00               0
...           ...             ...             ...             ...
2111092   1412612            0.00            0.00               0
2466311    324446      1430601.76      1774892.59               0
3832869    317107      1156506.45      1280680.70               0
4997230    450802       373871.24       174290.42               0
2102295    614115            0.00            0.00               0

[16426 rows x 10 columns]
```

In [20]:
```
print(Y)
```

```
2          1
3          1
251        1
252        1
680        1
          ..
2111092    0
2466311    0
3832869    0
4997230    0
2102295    0
Name: isFraud, Length: 16426, dtype: int64
```

In [21]:
```
# step11= split the data into training data and splitting data
```

In [22]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,train_size=0.33,stratify=Y,rando
```

In [23]:
```
print(X.shape,X_train.shape,X_test.shape)
```

```
(16426, 10) (5420, 10) (11006, 10)
```

In [24]:
```
# step 12
model=LogisticRegression()
```

In [25]:
```
# training the logistic regression model with training data
model.fit(X_train, Y_train)
```

```
C:\interr .ai\Lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWa
rning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[25]:

LogisticRegression

LogisticRegression()

In [26]:
```
# Model evaluation
# Accuracy score on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
print("Accuracy on training data:", training_data_accuracy)
```

```
Accuracy on training data: 0.9400369003690037
```

In [27]:
```
from sklearn.metrics import accuracy_score
```

In [28]:
```
# Accuracy score on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
print("Accuracy on test data:", test_data_accuracy)
```

```
Accuracy on test data: 0.9373069234962748
```

In [29]:
```
# Generate classification report
report = classification_report(Y_test, X_test_prediction, target_names=['No Fraud',
print("\nClassification Report:")
print(report)
```

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

    No Fraud       0.92      0.95      0.94      5503
       Fraud       0.95      0.92      0.94      5503

    accuracy                           0.94     11006
   macro avg       0.94      0.94      0.94     11006
weighted avg       0.94      0.94      0.94     11006
```

XGBOOST MODEL

In [30]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
```

In [31]:
```
data=pd.read_csv(r"C:\Users\swaro\OneDrive\Desktop\data science swaroop ky 1\Projec
data1=pd.DataFrame(data)
label_encoder = LabelEncoder()
# Iterate through all columns in the dataframe
for col in data1.columns:
    if data1[col].dtype == 'object':  # Check if the column is of object type
        data1[col] = label_encoder.fit_transform(data1[col])
print(data1)
```

```
         step  type      amount  nameOrig  oldbalanceOrg  newbalanceOrig  \
0           1     3     9839.64    757869      170136.00       160296.36
1           1     3     1864.28   2188998       21249.00        19384.72
2           1     4      181.00   1002156         181.00            0.00
3           1     1      181.00   5828262         181.00            0.00
4           1     3    11668.14   3445981       41554.00        29885.86
...       ...   ...         ...       ...            ...             ...
6362615   743     1   339682.13   5651847      339682.13            0.00
6362616   743     4  6311409.28   1737278     6311409.28            0.00
6362617   743     1  6311409.28    533958     6311409.28            0.00
6362618   743     4   850002.52   2252932      850002.52            0.00
6362619   743     1   850002.52    919229      850002.52            0.00

         nameDest  oldbalanceDest  newbalanceDest  isFraud  isFlaggedFraud
0         1662094            0.00            0.00        0               0
1         1733924            0.00            0.00        0               0
2          439685            0.00            0.00        1               0
3          391696        21182.00            0.00        1               0
4          828919            0.00            0.00        0               0
...           ...             ...             ...      ...             ...
6362615    505863            0.00       339682.13        1               0
6362616    260949            0.00            0.00        1               0
6362617    108224        68488.84      6379898.11        1               0
6362618    319713            0.00            0.00        1               0
6362619    534595      6510099.11      7360101.63        1               0

[6362620 rows x 11 columns]
```

In [32]:
```
# step 2
data1.head()
```

Out[32]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalance
0	1	3	9839.64	757869	170136.0	160296.36	1662094	
1	1	3	1864.28	2188998	21249.0	19384.72	1733924	
2	1	4	181.00	1002156	181.0	0.00	439685	
3	1	1	181.00	5828262	181.0	0.00	391696	21
4	1	3	11668.14	3445981	41554.0	29885.86	828919	

In [33]:
```
data1.duplicated()
```

Out[33]:
```
0          False
1          False
2          False
3          False
4          False
           ...
6362615    False
6362616    False
6362617    False
6362618    False
6362619    False
Length: 6362620, dtype: bool
```

In [34]:
```
data1.drop_duplicates()
```

Out[34]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	o
0	1	3	9839.64	757869	170136.00	160296.36	1662094	
1	1	3	1864.28	2188998	21249.00	19384.72	1733924	
2	1	4	181.00	1002156	181.00	0.00	439685	
3	1	1	181.00	5828262	181.00	0.00	391696	
4	1	3	11668.14	3445981	41554.00	29885.86	828919	
...	...	...	...	...	...	...	...	
6362615	743	1	339682.13	5651847	339682.13	0.00	505863	
6362616	743	4	6311409.28	1737278	6311409.28	0.00	260949	
6362617	743	1	6311409.28	533958	6311409.28	0.00	108224	
6362618	743	4	850002.52	2252932	850002.52	0.00	319713	
6362619	743	1	850002.52	919229	850002.52	0.00	534595	

6362620 rows × 11 columns

In [35]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            int32
 2   amount          float64
 3   nameOrig        int32
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        int32
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64
 10  isFlaggedFraud  int64
dtypes: float64(5), int32(3), int64(3)
memory usage: 461.2 MB
```

In [36]:
```
#distribution of legit and fraudulent transaction
data1['isFraud'].value_counts()
```

Out[36]:
```
isFraud
0    6354407
1       8213
Name: count, dtype: int64
```

In [37]:
```
# step 3
# Split the dataset into features and target variable
X = data1.drop('isFraud', axis=1)  # Features
y = data1['isFraud']  # Target variable
```

In [38]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Apply SMOTE to oversample the minority class
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

In [39]:
```
# Scale the features
from imblearn.over_sampling import SMOTE
scaler = StandardScaler()
X_resampled = scaler.fit_transform(X_resampled)
X_test = scaler.transform(X_test)
```

In [40]:
```
y_pred = model.predict(X_test)

# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
C:\interr .ai\Lib\site-packages\sklearn\base.py:493: UserWarning: X does not have va
lid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
```

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

           0       1.00      0.96      0.98   1270904
           1       0.02      0.67      0.04      1620

    accuracy                           0.96   1272524
   macro avg       0.51      0.81      0.51   1272524
weighted avg       1.00      0.96      0.98   1272524
```

In [41]:
```
# Check if the dataset is balanced after applying SMOTE
print("Class distribution after SMOTE:")
print(pd.Series(y_resampled).value_counts())
```

```
Class distribution after SMOTE:
isFraud
0    5083503
1    5083503
Name: count, dtype: int64
```

In [42]:
```
# step 4
# Create a new data frame with the resampled data
data2 = pd.DataFrame(X_resampled, columns=X.columns)
data2['is_fraud'] = y_resampled
```

In [43]:
```
data2.head()
```

Out[43]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest
0	1.821917	-1.432743	-0.236534	0.700215	1.043921	1.808651	-0.390259
1	-0.826861	-1.432743	-0.328597	1.443853	-0.310801	-0.030243	-0.534572
2	0.442224	0.616687	-0.430486	1.474768	-0.381364	-0.208630	1.409412
3	-0.733717	-0.749600	-0.161778	1.052650	-0.327424	-0.208630	-0.121700
4	-1.572012	1.299831	-0.331980	1.304323	-0.382474	-0.208630	-0.306431

In [44]:
```
data2['is_fraud'].value_counts()
```

Out[44]:
```
is_fraud
0    5083503
1    5083503
Name: count, dtype: int64
```

In [ ]:

Exploratory data analysis

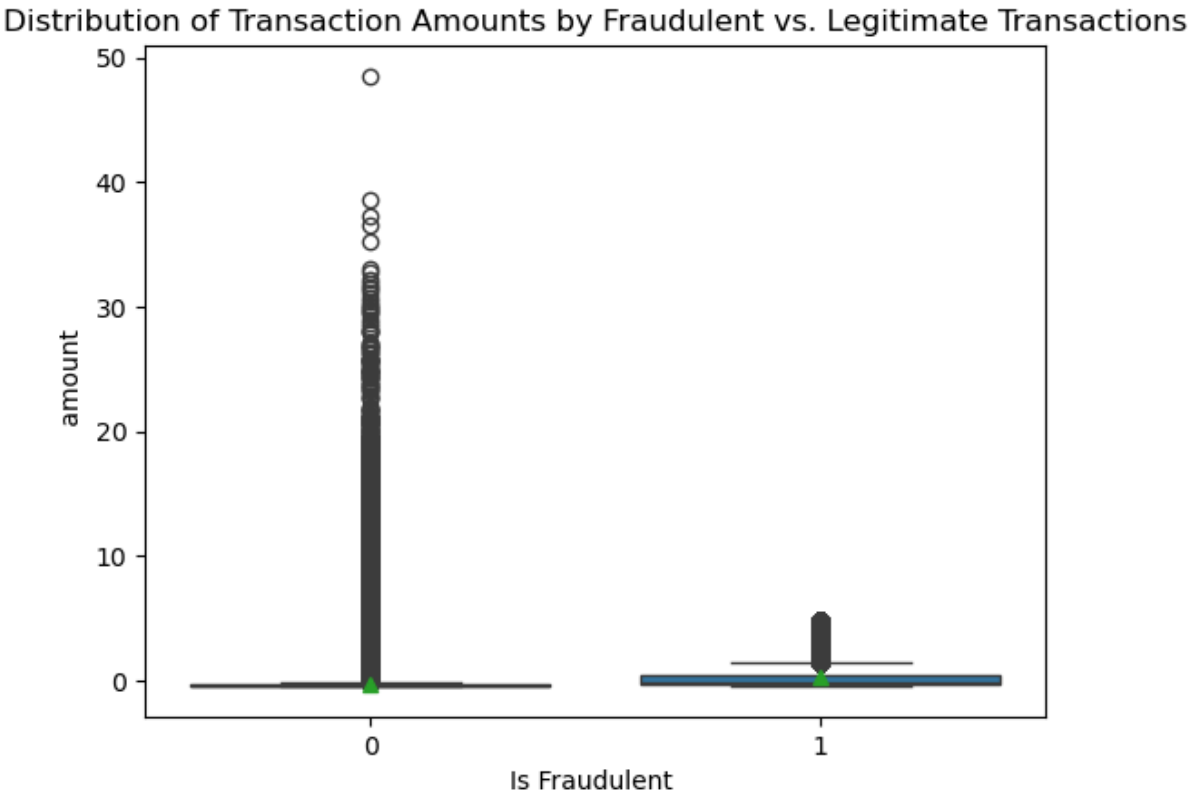
In [45]:
```
# step 5
data2.columns
```

Out[45]:
```
Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOrig',
       'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFlaggedFraud',
       'is_fraud'],
      dtype='object')
```

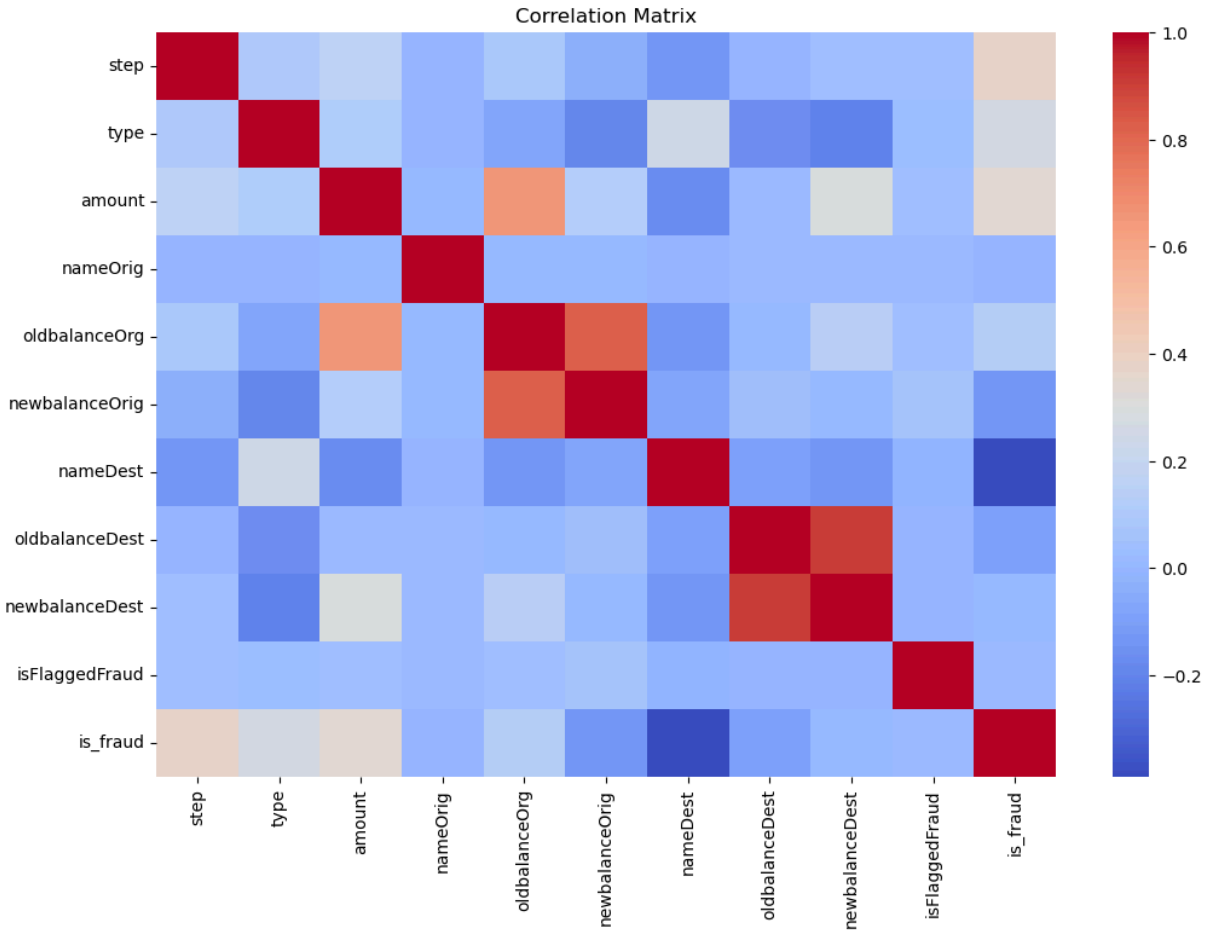
In [46]:
```
data2.is_fraud.describe()
```

Out[46]:
```
count    10167006.0
mean             0.5
std              0.5
min              0.0
25%              0.0
50%              0.5
75%              1.0
max              1.0
Name: is_fraud, dtype: float64
```

In [47]:
```
# Pictorial representation of the outlier by drawing the box plot

sns.boxplot(
    x = "is_fraud",
    y = "amount",
    showmeans=True,
    data=data2
)
plt.title("Distribution of Transaction Amounts by Fraudulent vs. Legitimate Transac
plt.xlabel("Is Fraudulent ")
plt.ylabel(" amount")
plt.show()
```



In [48]:
```
# step  6
# Correlation matrix
corr_matrix = data2.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=False, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



XGBOOST Model

In [49]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\interr .ai\lib\site-packages (2.1.0)
Requirement already satisfied: numpy in c:\interr .ai\lib\site-packages (from xgboos
t) (1.26.4)
Requirement already satisfied: scipy in c:\interr .ai\lib\site-packages (from xgboos
t) (1.13.1)
Note: you may need to restart the kernel to use updated packages.
```

In [50]:
```
# step 7
from sklearn.metrics import roc_auc_score, accuracy_score, confusion_matr
```

In [51]:
```
# Load the preprocessed dataset
X= data2.drop('is_fraud', axis=1)
y = data2['is_fraud']
```

In [52]:
```
data2['is_fraud'].value_counts()
```

Out[52]:
```
is_fraud
0    5083503
1    5083503
Name: count, dtype: int64
```

In [53]:
```
# step8 Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [54]:
```
# Create an XGBoost classifier
model = xgb.XGBClassifier(objective='binary:logistic',
                          max_depth=3,
                          learning_rate=0.1,
                          n_estimators=100,
                          min_child_weight=1,
                          gamma=0,
                          subsample=0.8,
                          colsample_bytree=0.8,
                          reg_alpha=0,
                          reg_lambda=1)
```

In [55]:
```
# Fit the model to the training data
model.fit(X_train, y_train)
```

Out[55]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=
None,
              enable_categorical=False, eval_metric=None, feature_types
=None,
              gamma=0, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=
None,
```

In [56]:
```
# Evaluate the model on the testing data
y_pred = model.predict(X_test)
```

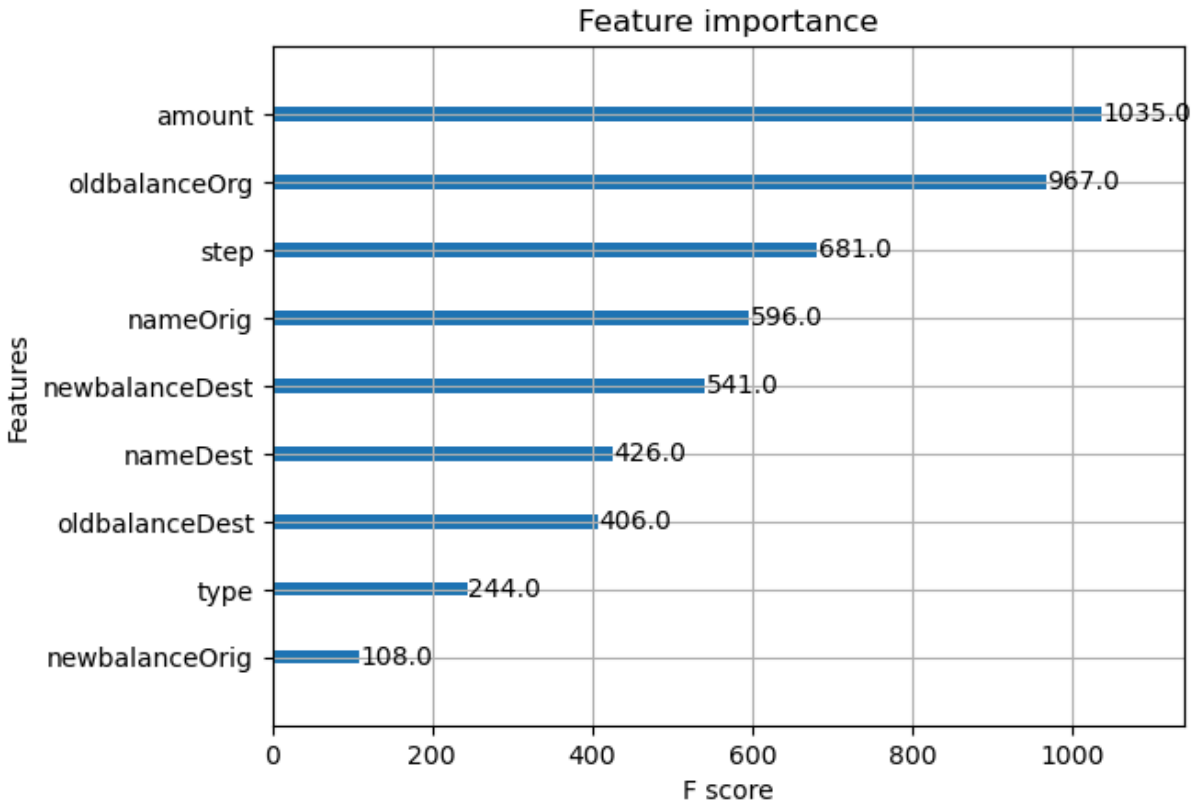
In [57]:
```
def fit_and_evaluate_model(X_train, X_test, y_train, y_test,xgb):
    xgb.fit(X_train, y_train)
    xgb_predict = xgb.predict(X_test)
    xgb_conf_matrix = confusion_matrix(y_test, xgb_predict)
    xgb_acc_score = accuracy_score(y_test, xgb_predict)
    print("confussion matrix")
    print(xgb_conf_matrix)
    print("\n")
    print("Accuracy of XGBoost:",xgb_acc_score*100,'\n')
    print(classification_report(y_test,xgb_predict))
    return xgb
```

In [58]:
```
xgb =  XGBClassifier(random_state=0)
model = fit_and_evaluate_model(X_train, X_test, y_train, y_test,xgb)
```

```
confussion matrix
[[1014715    2313]
 [    846 1015528]]


Accuracy of XGBoost: 99.84464459069086

              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1017028
           1       1.00      1.00      1.00   1016374

    accuracy                           1.00   2033402
   macro avg       1.00      1.00      1.00   2033402
weighted avg       1.00      1.00      1.00   2033402
```

The model shows high accuracy (96%) in detecting fraud (low precision and recall for class 1), indicating overfitting.

In [59]:
```
# Step 9  Check Feature Importance

import xgboost as xgb
import matplotlib.pyplot as plt

# Assuming 'model' is your trained XGBoost model
xgb.plot_importance(model)
plt.show()

# Get feature importance as a dictionary
importance = model.get_booster().get_score(importance_type='weight')
importance = sorted(importance.items(), key=lambda x: x[1], reverse=True)

# Print feature importance
for feature, score in importance:
    print(f"Feature: {feature}, Importance Score: {score}")
```




```
Feature: amount, Importance Score: 1035.0
Feature: oldbalanceOrg, Importance Score: 967.0
Feature: step, Importance Score: 681.0
Feature: nameOrig, Importance Score: 596.0
Feature: newbalanceDest, Importance Score: 541.0
Feature: nameDest, Importance Score: 426.0
Feature: oldbalanceDest, Importance Score: 406.0
Feature: type, Importance Score: 244.0
Feature: newbalanceOrig, Importance Score: 108.0
```

In [66]:
```
# Define the features to remove
leaking_features = ['nameDest', 'oldbalanceOrg', 'step']
```

In [68]:
```
# Remove leaking features from training and test sets
X_train_cleaned = X_train.drop(columns=leaking_features)
X_test_cleaned = X_test.drop(columns=leaking_features)

# Retrain the model without the leaking features
model = xgb.XGBClassifier()
model.fit(X_train_cleaned, y_train)
```

Out[68]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds
=None,
              enable_categorical=False, eval_metric=None, feature_types
=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin
=None,
```

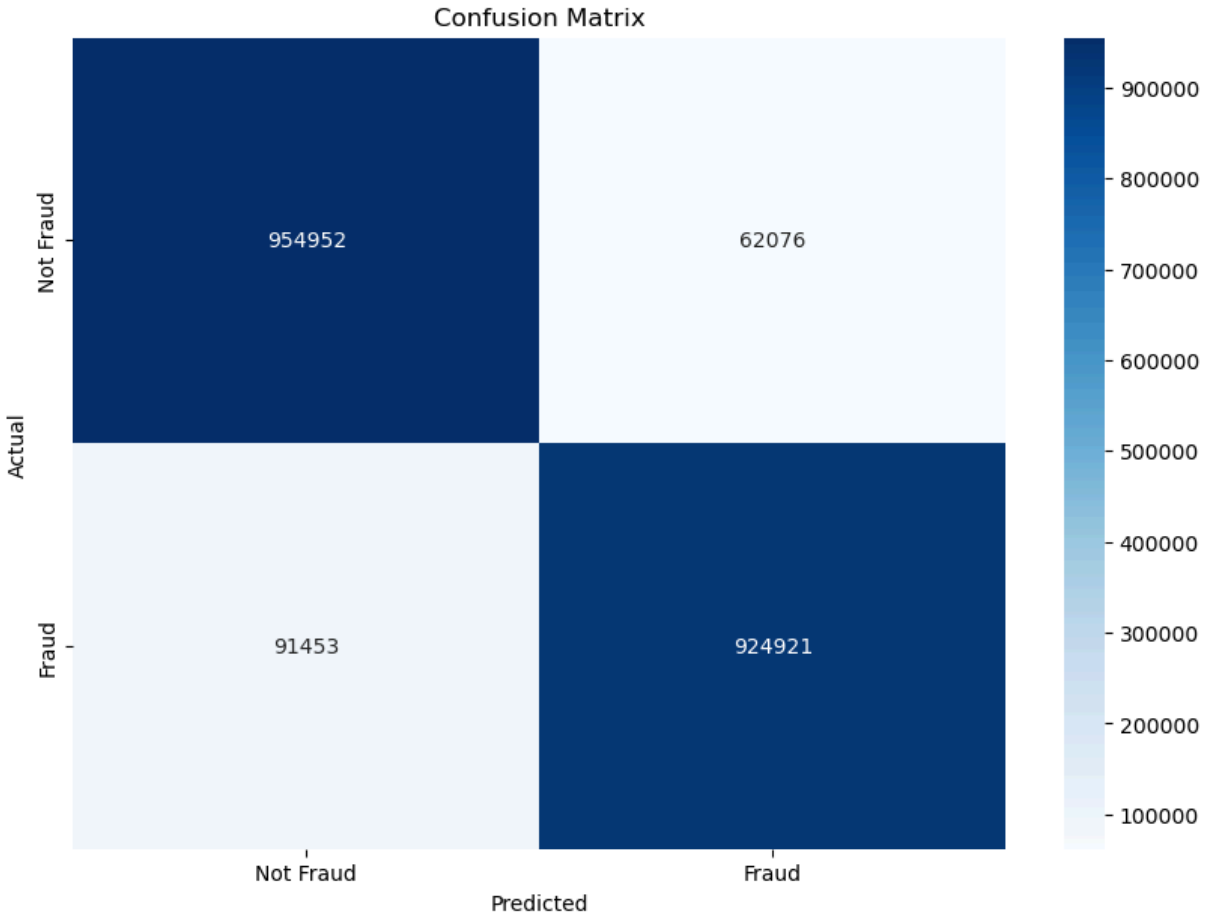
In [69]:

```
#  step 10 Evaluate the model
y_pred = model.predict(X_test_cleaned)
accuracy = (y_pred == y_test).mean()
print(f"Accuracy of XGBoost after removing leaking features: {accuracy}")

# Generate the classification report
report = classification_report(y_test, y_pred, target_names=['Not Fraud', 'Fraud'])
print("Classification Report:\n")
print(report)

# Generate and print the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n")
print(conf_matrix)
```

```
Accuracy of XGBoost after removing leaking features: 0.9244964842170904
Classification Report:

              precision    recall  f1-score   support

   Not Fraud       0.91      0.94      0.93   1017028
       Fraud       0.94      0.91      0.92   1016374

    accuracy                           0.92   2033402
   macro avg       0.92      0.92      0.92   2033402
weighted avg       0.92      0.92      0.92   2033402

Confusion Matrix:

[[954952  62076]
 [ 91453 924921]]
```

In [70]:

```
# Plot the confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Fraud
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```


Confusion Matrix
954952
62076
91453
924921
Not Fraud
Fraud
Actual
Not Fraud
Fraud
Predicted
900000
800000
700000
600000
500000
400000
300000
200000
100000

In [ ]: