In [24]:
```
# Data cleaning including missing values, outliers and multi-collinearity.
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler
```

In [25]:
```
df = pd.read_csv("Fraud.csv")
```

In [26]:
```
df
```

Out[26]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	n
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M19
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M20
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C5
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M12
...	...	...	...	...	...	...	
28292	8	CASH_OUT	7270.37	C457003860	0.0	0.00	C2
28293	8	CASH_OUT	113043.31	C1845952463	0.0	0.00	C4
28294	8	CASH_OUT	89346.62	C140193335	0.0	0.00	C13
28295	8	CASH_OUT	138651.85	C297851161	0.0	0.00	C10
28296	8	CASH_OUT	61553.92	C1612091270	0.0	0.00	

28297 rows × 11 columns

In [27]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28297 entries, 0 to 28296
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   step            28297 non-null  int64
 1   type            28297 non-null  object
 2   amount          28297 non-null  float64
 3   nameOrig        28297 non-null  object
 4   oldbalanceOrg   28297 non-null  float64
 5   newbalanceOrig  28297 non-null  float64
 6   nameDest        28297 non-null  object
 7   oldbalanceDest  28297 non-null  float64
 8   newbalanceDest  28297 non-null  float64
 9   isFraud         28296 non-null  float64
 10  isFlaggedFraud  28296 non-null  float64
dtypes: float64(7), int64(1), object(3)
memory usage: 2.4+ MB
None
```

In [28]:
```
print(df.describe())
```

```
               step        amount  oldbalanceOrg  newbalanceOrig  \
count  28297.000000  2.829700e+04   2.829700e+04    2.829700e+04
mean       6.508252  1.357405e+05   7.667026e+05    7.823551e+05
std        2.291090  3.013167e+05   2.126123e+06    2.166615e+06
min        1.000000  1.770000e+00   0.000000e+00    0.000000e+00
25%        6.000000  5.966520e+03   0.000000e+00    0.000000e+00
50%        8.000000  1.950669e+04   1.963654e+04    3.682140e+03
75%        8.000000  1.601022e+05   1.386575e+05    1.407606e+05
max        8.000000  1.000000e+07   2.235231e+07    2.246600e+07

       oldbalanceDest  newbalanceDest       isFraud  isFlaggedFraud
count    2.829700e+04    2.829700e+04  28296.000000         28296.0
mean     8.483811e+05    1.191306e+06      0.002969             0.0
std      2.513869e+06    3.106440e+06      0.054405             0.0
min      0.000000e+00    0.000000e+00      0.000000             0.0
25%      0.000000e+00    0.000000e+00      0.000000             0.0
50%      0.000000e+00    0.000000e+00      0.000000             0.0
75%      3.654323e+05    6.670935e+05      0.000000             0.0
max      2.495524e+07    2.878359e+07      1.000000             0.0
```

In [29]:
```
print(df.head())
```

```
   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             0.0
1  M2044282225             0.0             0.0      0.0             0.0
2   C553264065             0.0             0.0      1.0             0.0
3    C38997010         21182.0             0.0      1.0             0.0
4  M1230701703             0.0             0.0      0.0             0.0
```

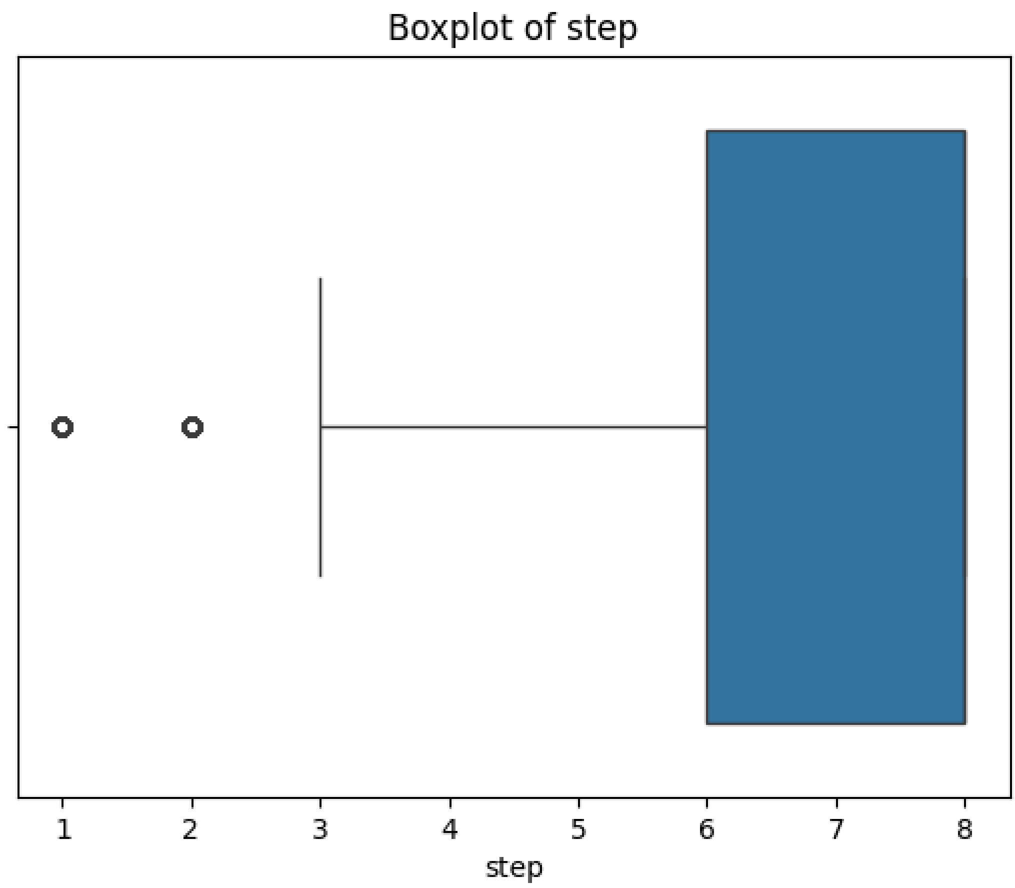
In [30]:
```
# Check missing values
print("\nMissing Values:\n", df.isnull().sum())
```

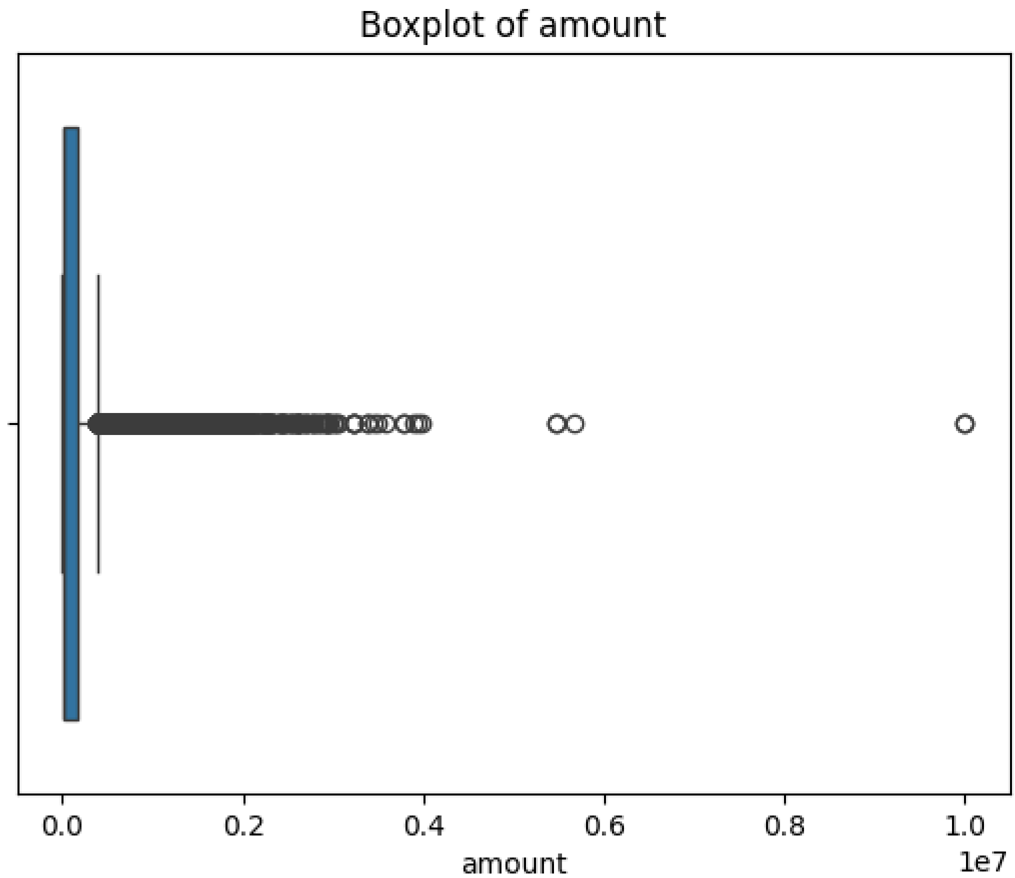
```
Missing Values:
 step             0
type             0
amount           0
nameOrig         0
oldbalanceOrg    0
newbalanceOrig   0
nameDest         0
oldbalanceDest   0
newbalanceDest   0
isFraud          1
isFlaggedFraud   1
dtype: int64
```

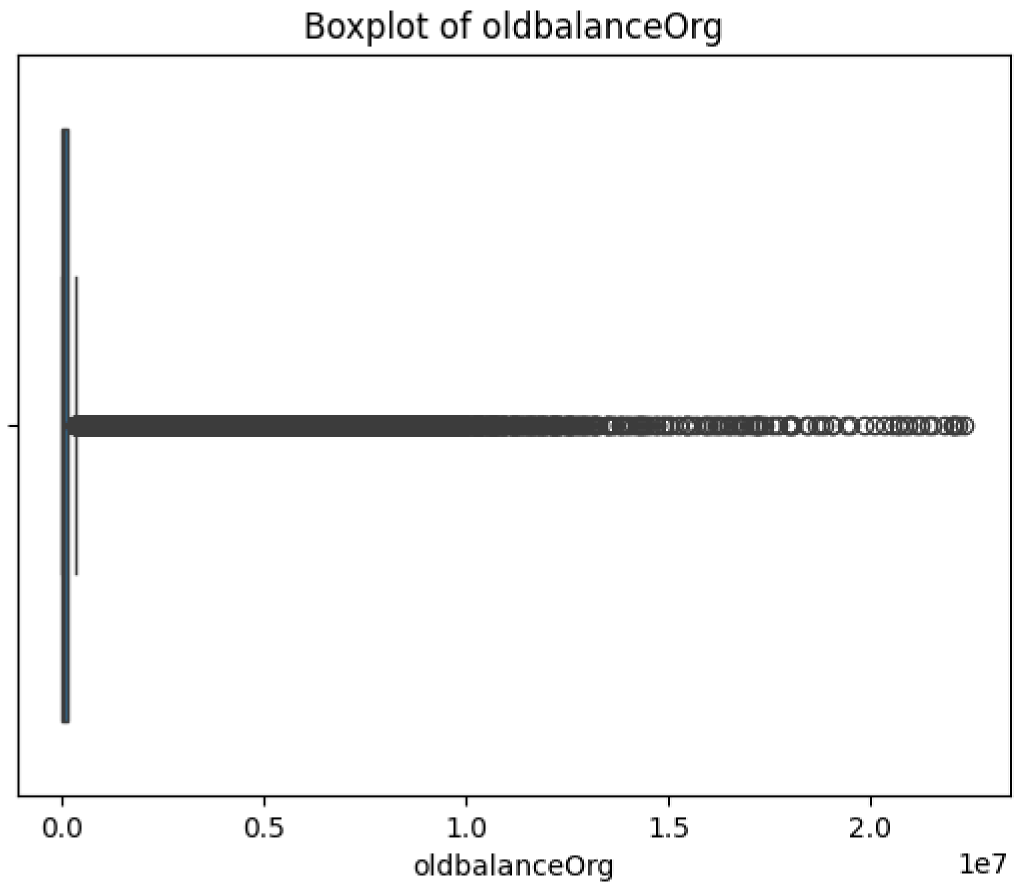
In [31]:
```
# Option 1: Drop rows with missing values (if few)
df_cleaned = df.dropna()
```

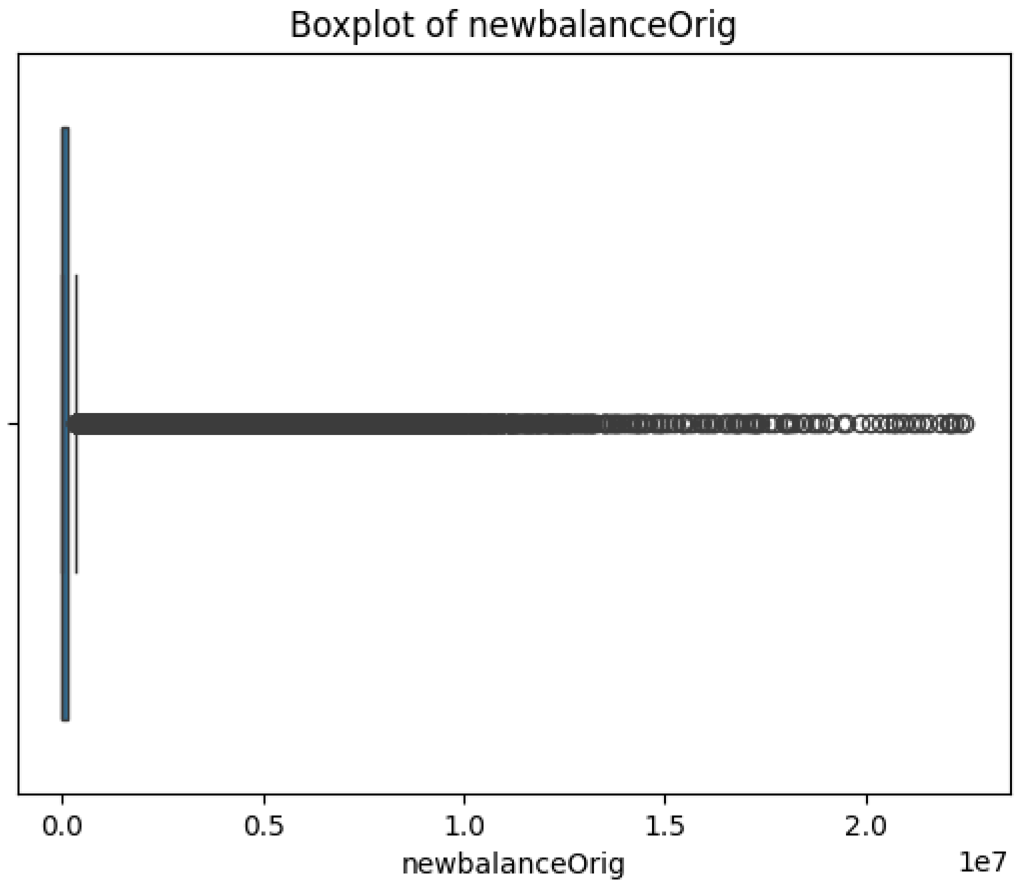
In [32]:
```
# Visualize outliers using boxplot
numeric_cols = df_cleaned.select_dtypes(include=np.number).columns
```

In [33]:
```
for col in numeric_cols:
    sns.boxplot(x=df_cleaned[col])
    plt.title(f'Boxplot of {col}')
    plt.show()
```

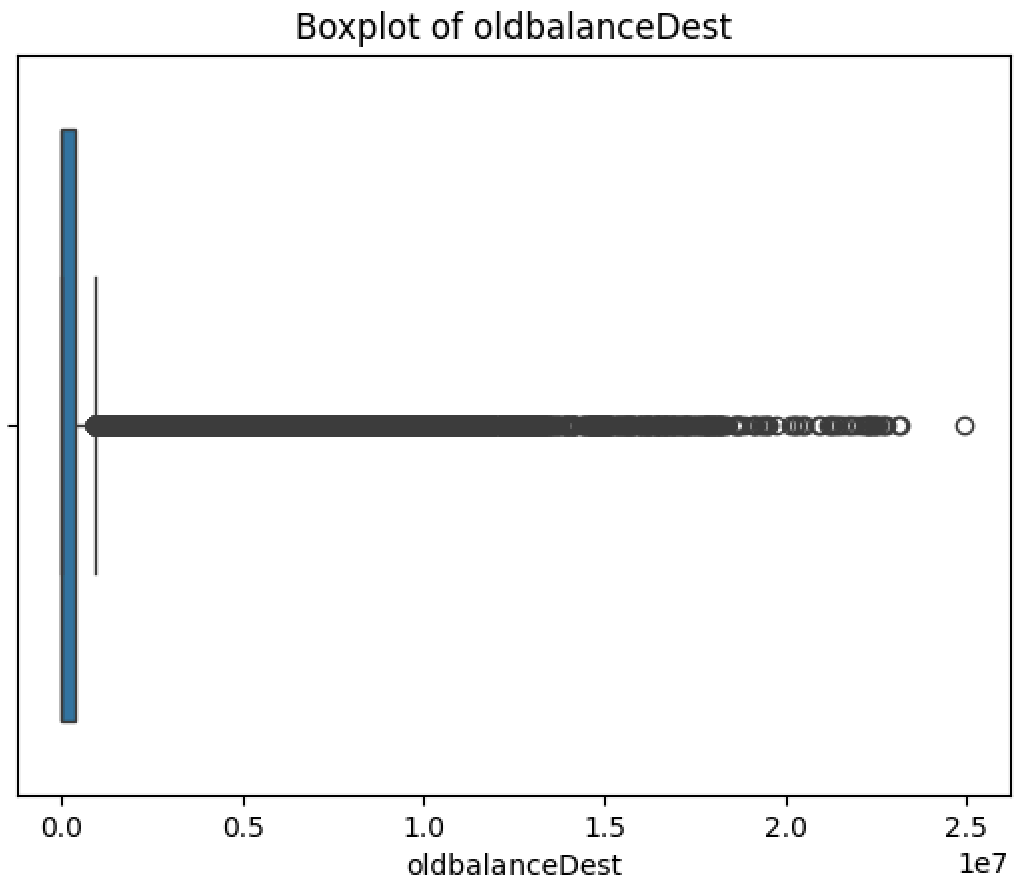



Boxplot of amount
0.0
0.2
0.4
0.6
0.8
1.0
amount
1e7

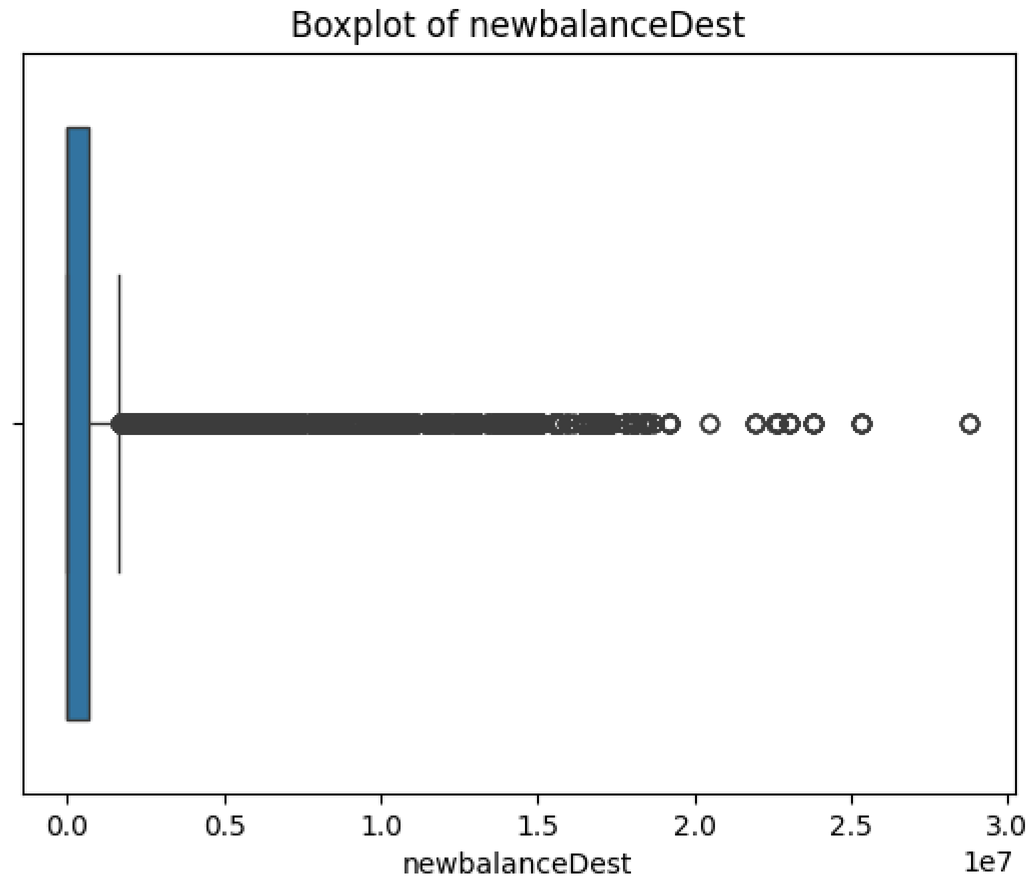

Boxplot of oldbalanceOrg
0.0
0.5
1.0
1.5
2.0
oldbalanceOrg
1e7



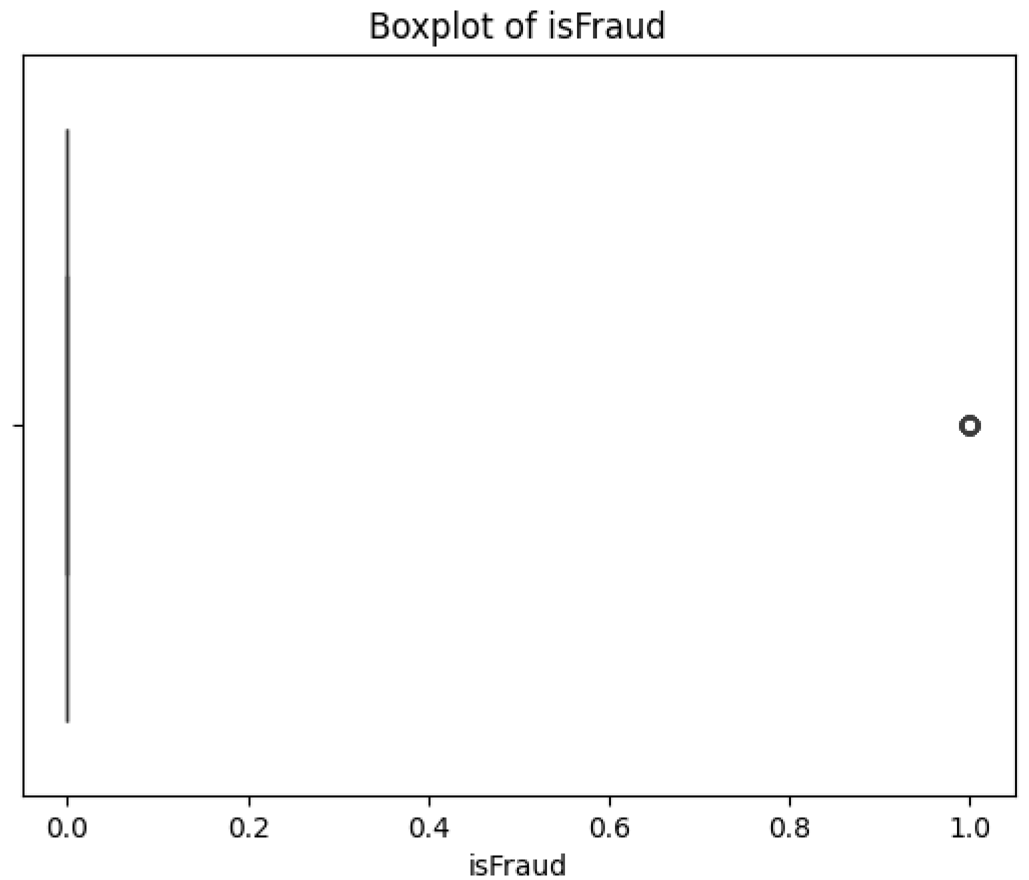
Boxplot of newbalanceOrig
0.0
0.5
1.0
1.5
2.0
1e7
newbalanceOrig



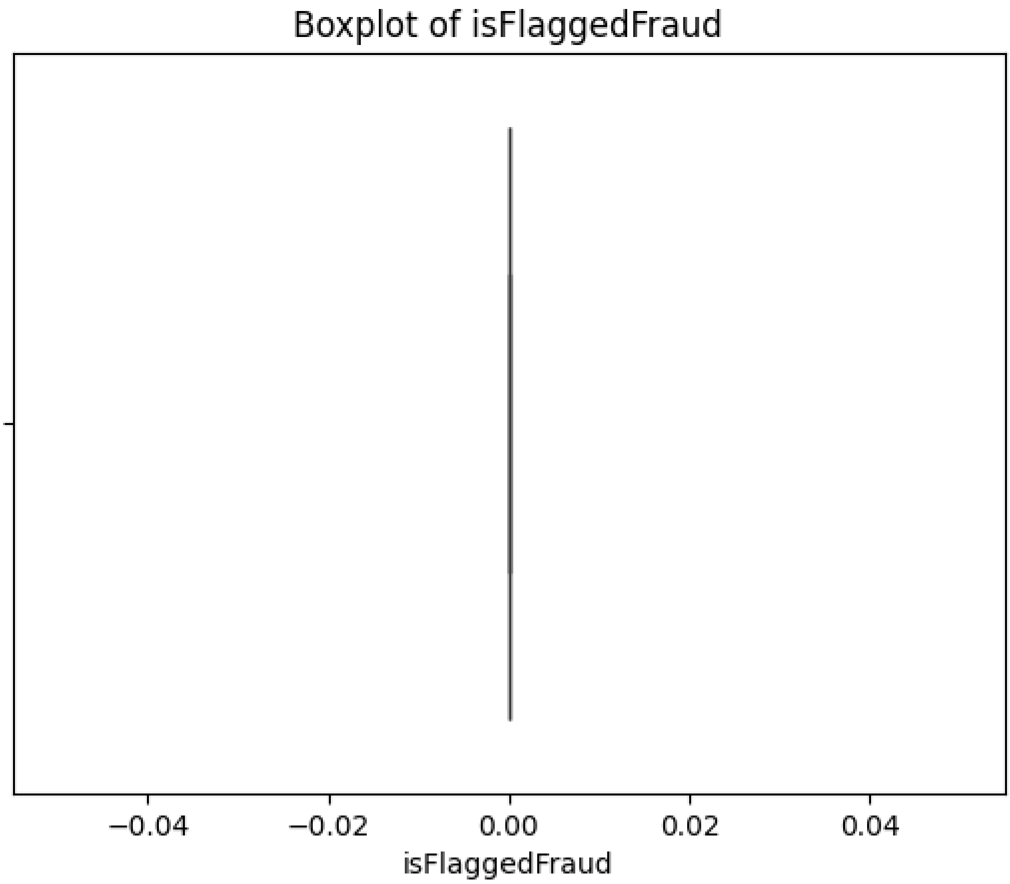
Boxplot of oldbalanceDest
0.0
0.5
1.0
1.5
2.0
2.5
1e7
oldbalanceDest



Boxplot of newbalanceDest
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e7
newbalanceDest



Boxplot of isFraud
0.0
0.2
0.4
0.6
0.8
1.0
isFraud



In [34]:
```
# Remove outliers using IQR
def remove_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    return data[(data[column] >= lower) & (data[column] <= upper)]
```

In [35]:
```
# Apply outlier removal for numeric columns
for col in numeric_cols:
    df_cleaned = remove_outliers_iqr(df_cleaned, col)
```

In [36]:
```
# 3. Check for Multicollinearity (VIF)
# Standardize numeric features
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df_cleaned[numeric_cols]), columns
```

In [37]:
```
# Calculate VIF
vif_data = pd.DataFrame()
vif_data["Feature"] = df_scaled.columns
vif_data["VIF"] = [variance_inflation_factor(df_scaled.values, i) for i in range
```

In [38]:
```
print("\nVIF (Variance Inflation Factor):\n", vif_data)
```

```
VIF (Variance Inflation Factor):
           Feature       VIF
0             step  1.007076
1           amount  1.638127
2    oldbalanceOrg  5.723853
3   newbalanceOrig  5.657809
4   oldbalanceDest  1.269601
5   newbalanceDest       NaN
6          isFraud       NaN
7   isFlaggedFraud       NaN
```

In [39]:
```python
# Optional: Drop features with high VIF (>10)
high_vif_cols = vif_data[vif_data["VIF"] > 10]["Feature"].tolist()
df_final = df_cleaned.drop(columns=high_vif_cols)
```

In [40]:
```python
# Final cleaned data
print("\nFinal Cleaned Data Shape:", df_final.shape)
print(df_final.head())
```

```
Final Cleaned Data Shape: (9903, 11)
      step     type    amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
3722     3  PAYMENT  20971.00  C1415812333        19462.0            0.00
3723     3  PAYMENT  27659.67   C647218712        37057.0         9397.33
3725     3  PAYMENT    501.11   C174999703         8934.5         8433.39
3726     3  PAYMENT  11866.89   C431939256        10565.0            0.00
3727     3  PAYMENT   2108.36  C1677115089            0.0            0.00

         nameDest  oldbalanceDest  newbalanceDest  isFraud  isFlaggedFraud
3722  M1715606187             0.0             0.0      0.0             0.0
3723   M876864630             0.0             0.0      0.0             0.0
3725   M854977732             0.0             0.0      0.0             0.0
3726   M463759298             0.0             0.0      0.0             0.0
3727  M2130242983             0.0             0.0      0.0             0.0
```

In [41]:
```python
#Demonstrate the performance of the model by using best set of tools.
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_scc
from imblearn.over_sampling import SMOTE
import seaborn as sns
import matplotlib.pyplot as plt
```

In [42]:
```python
# Load data
df = pd.read_csv("Fraud.csv")
```

In [43]:
```python
# Drop ID-like columns that won't help
df = df.drop(columns=["nameOrig", "nameDest"], errors='ignore')
```

In [44]:
```python
df
```

Out[44]:

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	ne
0	1	PAYMENT	9839.64	170136.0	160296.36	0.00	
1	1	PAYMENT	1864.28	21249.0	19384.72	0.00	
2	1	TRANSFER	181.00	181.0	0.00	0.00	
3	1	CASH_OUT	181.00	181.0	0.00	21182.00	
4	1	PAYMENT	11668.14	41554.0	29885.86	0.00	
...	...	...	...	...	...	...	
42266	9	CASH_OUT	195364.06	0.0	0.00	506957.59	
42267	9	CASH_OUT	546075.62	0.0	0.00	5075471.31	
42268	9	CASH_OUT	111003.87	0.0	0.00	2533159.94	
42269	9	CASH_OUT	101025.44	0.0	0.00	156646.32	
42270	9	CASH_OUT	271441.28	0.0	0.00	NaN	

42271 rows × 9 columns

In [45]:
```
# Encode 'type' column
df['type'] = LabelEncoder().fit_transform(df['type'])
```

In [46]:
```
# Define features and target
X = df.drop(['isFraud', 'isFlaggedFraud'], axis=1)
y = df['isFraud']
```

In [47]:
```
# Combine X and y into one DataFrame temporarily
df_combined = pd.concat([X, y], axis=1)

# Drop rows where isFraud is missing
df_combined = df_combined.dropna(subset=["isFraud"])

# Separate again
X = df_combined.drop("isFraud", axis=1)
y = df_combined["isFraud"]
```

In [48]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)
```

In [49]:
```
# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [51]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, classification_report
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from imblearn.over_sampling import SMOTE
import warnings
warnings.filterwarnings("ignore")
```

In [50]:
```
# Balance data using SMOTE
sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train_scaled, y_train)
```

In [53]:
```
best_model_name = max(accuracies, key=accuracies.get)
print(f"\n🏆 Best Model: {best_model_name}")
```

```
🏆 Best Model: Decision Tree
```

In [55]:
```
# Define models
models = {
    "Decision Tree": DecisionTreeClassifier(),
    "SVC": SVC(probability=True),
    "Naive Bayes": GaussianNB(),
    "MLP": MLPClassifier(max_iter=300)
}

accuracies = {}
for name, model in models.items():
    model.fit(X_train_res, y_train_res)
    acc = accuracy_score(y_test, model.predict(X_test_scaled))
    accuracies[name] = acc
    print(f"\n {name} Accuracy: {acc:.4f}")
    print(classification_report(y_test, model.predict(X_test_scaled)))

# Pick the best model by accuracy
# best_model_name = max(accuracies, key=accuracies.get)
# print(f"\n🏆 Best Model: {best_model_name}")
print(accuracies)
# Define GridSearchCV param grid based on best model
if best_model_name == "Decision Tree":
    model = DecisionTreeClassifier()
    param_grid = {
        "max_depth": [5, 10, 15],
        "min_samples_split": [2, 5, 10]
    }

elif best_model_name == "SVC":
    model = SVC(probability=True)
    param_grid = {
        "C": [0.1, 1, 10],
        "kernel": ["linear", "rbf"]
    }

elif best_model_name == "MLP":
    model = MLPClassifier(max_iter=300)
    param_grid = {
        "hidden_layer_sizes": [(50,), (100,), (100, 50)],
        "activation": ["relu", "tanh"],
        "alpha": [0.0001, 0.001]
    }
```

```
elif best_model_name == "Naive Bayes":
    model = GaussianNB()
    param_grid = {}  # no tunable hyperparameters

# Run GridSearchCV if applicable
if param_grid:
    print(f"\n Tuning {best_model_name}...")
    grid = GridSearchCV(model, param_grid, cv=3, scoring='accuracy', n_jobs=1)
    grid.fit(X_train_res, y_train_res)
    best_model = grid.best_estimator_
    print(f" Best Params: {grid.best_params_}")
else:
    best_model = model
    best_model.fit(X_train_res, y_train_res)

# Final evaluation
final_pred = best_model.predict(X_test_scaled)
final_acc = accuracy_score(y_test, final_pred)

print(f"\n Final Accuracy (after tuning): {final_acc:.4f}")
print(classification_report(y_test, final_pred))
```

```
 Decision Tree Accuracy: 0.9962
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      8435
         1.0       0.33      0.68      0.45        19

    accuracy                           1.00      8454
   macro avg       0.67      0.84      0.72      8454
weighted avg       1.00      1.00      1.00      8454


 SVC Accuracy: 0.9347
              precision    recall  f1-score   support

         0.0       1.00      0.93      0.97      8435
         1.0       0.03      0.89      0.06        19

    accuracy                           0.93      8454
   macro avg       0.51      0.91      0.51      8454
weighted avg       1.00      0.93      0.96      8454


 Naive Bayes Accuracy: 0.2459
              precision    recall  f1-score   support

         0.0       1.00      0.24      0.39      8435
         1.0       0.00      1.00      0.01        19

    accuracy                           0.25      8454
   macro avg       0.50      0.62      0.20      8454
weighted avg       1.00      0.25      0.39      8454


 MLP Accuracy: 0.9860
              precision    recall  f1-score   support

         0.0       1.00      0.99      0.99      8435
         1.0       0.13      0.89      0.22        19

    accuracy                           0.99      8454
   macro avg       0.56      0.94      0.61      8454
weighted avg       1.00      0.99      0.99      8454

{'Decision Tree': 0.9962148095576059, 'SVC': 0.9347054648687012, 'Naive Bayes':
0.2459190915542938, 'MLP': 0.9860421102436716}

 Tuning Decision Tree...
 Best Params: {'max_depth': 15, 'min_samples_split': 2}

 Final Accuracy (after tuning): 0.9943
              precision    recall  f1-score   support

         0.0       1.00      0.99      1.00      8435
         1.0       0.25      0.74      0.37        19

    accuracy                           0.99      8454
   macro avg       0.62      0.87      0.68      8454
weighted avg       1.00      0.99      1.00      8454
```

In [ ]:

```

```