In [3]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.metrics import accuracy_score, classification_report,confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
import xgboost as xgb
from sklearn.naive_bayes import GaussianNB
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.metrics.cluster import silhouette_score
import matplotlib.pyplot as plt
import seaborn as sns
```

In [4]:

```
#importing the file
data = pd.read_csv(r"E:\onlinefraud.csv")
```

In [5]:

```
data.head()
```

Out[5]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFra
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	

In [6]:

```
data.tail()
```

Out[6]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceD
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.0	C776919290	0.00	339682
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.0	C1881841831	0.00	0
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.0	C1365125890	68488.84	6379898
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.0	C2080388513	0.00	0
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.0	C873221189	6510099.11	7360101

In [7]:

```
data.dtypes
```

Out[7]:

```
step                int64
type               object
amount            float64
nameOrig           object
oldbalanceOrg     float64
newbalanceOrig    float64
nameDest           object
oldbalanceDest    float64
newbalanceDest    float64
isFraud             int64
isFlaggedFraud      int64
dtype: object
```

In [8]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype  
---  ------          -----  
 0   step            int64  
 1   type            object 
 2   amount          float64
 3   nameOrig        object 
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        object 
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64  
 10  isFlaggedFraud  int64  
dtypes: float64(5), int64(3), object(3)
memory usage: 534.0+ MB
```

In [9]:

```
data.describe()
```

Out[9]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFrau
count	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+0
mean	2.433972e+02	1.798619e+05	8.338831e+05	8.551137e+05	1.100702e+06	1.224996e+06	1.290820e-03	2.514687e-0
std	1.423320e+02	6.038582e+05	2.888243e+06	2.924049e+06	3.399180e+06	3.674129e+06	3.590480e-02	1.585775e-0
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+0
25%	1.560000e+02	1.338957e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+0
50%	2.390000e+02	7.487194e+04	1.420800e+04	0.000000e+00	1.327057e+05	2.146614e+05	0.000000e+00	0.000000e+0
75%	3.350000e+02	2.087215e+05	1.073152e+05	1.442584e+05	9.430367e+05	1.111909e+06	0.000000e+00	0.000000e+0
max	7.430000e+02	9.244552e+07	5.958504e+07	4.958504e+07	3.560159e+08	3.561793e+08	1.000000e+00	1.000000e+0

In [10]:

```
#Normalizing numerical columns
numeric_cols = ['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest','newbalanceDest']
scaler = StandardScaler()
data[numeric_cols] = scaler.fit_transform(data[numeric_cols])
```

In [11]:

```
data.head()
```

Out[11]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFr
0	1	PAYMENT	-0.281560	C1231006815	-0.229810	-0.237622	M1979787155	-0.323814	-0.333411	
1	1	PAYMENT	-0.294767	C1666544295	-0.281359	-0.285812	M2044282225	-0.323814	-0.333411	
2	1	TRANSFER	-0.297555	C1305486145	-0.288654	-0.292442	C553264065	-0.323814	-0.333411	
3	1	CASH_OUT	-0.297555	C840083671	-0.288654	-0.292442	C38997010	-0.317582	-0.333411	
4	1	PAYMENT	-0.278532	C2048537720	-0.274329	-0.282221	M1230701703	-0.323814	-0.333411	

In [12]:

```
#doing the Label Encoder to transform the below columns
le = LabelEncoder()
data['type'] = le.fit_transform(data['type'])
data['nameOrig'] = le.fit_transform(data['nameOrig'])
data['nameDest'] = le.fit_transform(data['nameDest'])
```

In [13]:

```
#Splitting data into training and testing sets:
X = data.drop('isFraud', axis=1) # features
y = data['isFraud'] # target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)
```

In [14]:

```
#Define XGBoost classifier #
xgb_model = xgb.XGBClassifier(
max_depth=6, # Maximum tree depth
learning_rate=0.3, # Learning rate (step size)
n_estimators=150, # Number of trees
gamma=0, # Minimum loss reduction
subsample=0.8, # Fraction of samples for each tree
colsample_bytree=0.8, # Fraction of features for each tree
reg_alpha=0.1, # L1 regularization term
reg_lambda=0.1 # L2 regularization term
)
#Train XGBoost model
xgb_model.fit(X_train, y_train)
```

Out[14]:

```
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,
              feature_weights=None, gamma=0, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.3, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=6,
              max_leaves=None, min_child_weight=None, missing=nan,
```

In [15]:

```
#Make predictions on test set
y_pred_xgb = xgb_model.predict(X_test)
print(y_pred_xgb)
```

```
[0 0 0 ... 0 0 0]
```

In [16]:

```
print("XGBoost Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_xgb))
print("Classification Report:", classification_report(y_test, y_pred_xgb))
print("Confusion Matrix:", confusion_matrix(y_test, y_pred_xgb))
```

```
XGBoost Model Performance:
Accuracy: 0.9996196535389509
Classification Report:               precision    recall  f1-score   support

           0       1.00      1.00      1.00   1270904
           1       0.89      0.80      0.84      1620

    accuracy                           1.00   1272524
   macro avg       0.94      0.90      0.92   1272524
weighted avg       1.00      1.00      1.00   1272524

Confusion Matrix: [[1270741     163]
 [    321    1299]]
```

In [17]:

```
# ADABOOST CLASSIFIER #
adaboost_model = AdaBoostClassifier(
n_estimators=100,
learning_rate=0.5,
random_state=42
)
```

In [18]:

```
#Train AdaBoost model
adaboost_model.fit(X_train, y_train)
```

Out[18]:

```
AdaBoostClassifier

AdaBoostClassifier(learning_rate=0.5, n_estimators=100, random_state=42)
```

In [19]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [20]:

```
 #Make predictions on test set
y_pred_adaboost = adaboost_model.predict(X_test)
print(y_pred_adaboost)
```

```
[0 0 0 ... 0 0 0]
```

In [21]:

```
#Evaluate model performance
print("AdaBoost Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_adaboost))
print("Classification Report:", classification_report(y_test, y_pred_adaboost))
print("Confusion Matrix:", confusion_matrix(y_test, y_pred_adaboost))
```

```
AdaBoost Model Performance:
Accuracy: 0.9988676048546039
Classification Report:               precision    recall  f1-score   support

           0       1.00      1.00      1.00   1270904
           1       1.00      0.11      0.20      1620

    accuracy                           1.00   1272524
   macro avg       1.00      0.56      0.60   1272524
weighted avg       1.00      1.00      1.00   1272524

Confusion Matrix: [[1270904       0]
 [   1441     179]]
```

In [22]:

```
#Scale data using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [23]:

```
#K-Nearest Neighbors (KNN) classification #
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train_scaled, y_train)
```

Out[23]:

```
▾ KNeighborsClassifier ⓘ ?

KNeighborsClassifier()
```

In [24]:

```
#Make predictions on test set
y_pred_knn = knn_model.predict(X_test_scaled)
print(y_pred_knn)
```

```
[0 0 0 ... 0 0 0]
```

In [25]:

```
#Evaluate model performance
print("KNN Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Classification Report:", classification_report(y_test, y_pred_knn))
print("Confusion Matrix:", confusion_matrix(y_test, y_pred_knn))
```

```
KNN Model Performance:
Accuracy: 0.9991866558115996
Classification Report:               precision    recall  f1-score   support

           0       1.00      1.00      1.00   1270904
           1       0.97      0.37      0.54      1620

    accuracy                           1.00   1272524
   macro avg       0.98      0.69      0.77   1272524
weighted avg       1.00      1.00      1.00   1272524

Confusion Matrix: [[1270883      21]
 [   1014     606]]
```

In [ ]: