In [1]:
```
#Importing all the required packages
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
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 [2]:
```
#importing the file
data = pd.read_csv(r"E:\onlinefraud.csv")
```

In [3]:
```
data.head()
```

Out[3]:

	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 [4]:
```
data.tail()
```

Out[4]:

	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 [5]:
```
data.describe()
```

Out[5]:

	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 [6]:
```
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 [7]:
```
data.isnull().sum()
```

Out[7]:
```
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 [11]:
```
data.dtypes
```

Out[11]:
```
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 [12]:
```
#normalizing numerical coloumns
numeric_cols = ['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest','newbalanceDest']
scaler = StandardScaler()
data[numeric_cols] = scaler.fit_transform(data[numeric_cols])
```

In [15]:
```
data.head()
```

Out[15]:

	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 [16]:
```
#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 [17]:
```
#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 [18]:
```
# LOGISTIC REGRESSION #
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train, y_train)
```

Out[18]:
```
LogisticRegression
LogisticRegression(max_iter=1000)
```

In [19]:
```
y_pred_logreg = logreg.predict(X_test)
print(y_pred_logreg)
```

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

In [20]:
```
print("Logistic Regression Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_logreg))
print("Classification Report:\n", classification_report(y_test, y_pred_logreg))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_logreg))
```

```
Logistic Regression Model Performance:
Accuracy: 0.9988188827872795
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00   1270904
           1       0.84      0.09      0.16      1620

    accuracy                           1.00   1272524
   macro avg       0.92      0.54      0.58   1272524
weighted avg       1.00      1.00      1.00   1272524

Confusion Matrix:
 [[1270876      28]
 [   1475     145]]
```

In [21]:
```
# DESICION TREE CLASSIFIER #
DT = DecisionTreeClassifier(random_state=42)
DT.fit(X_train, y_train)
```

Out[21]:
```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

In [22]:
```
#Model evaluation
y_pred_DT = DT.predict(X_test)
print(y_pred_DT)
```

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

In [23]:
```
print("Decision Tree Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_DT))
print("Classification Report:", classification_report(y_test, y_pred_DT))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_DT))
```

```
Decision Tree Model Performance:
Accuracy: 0.9997029525572798
Classification Report:               precision    recall  f1-score   support

           0       1.00      1.00      1.00   1270904
           1       0.90      0.87      0.88      1620

    accuracy                           1.00   1272524
   macro avg       0.95      0.93      0.94   1272524
weighted avg       1.00      1.00      1.00   1272524

Confusion Matrix:
 [[1270743     161]
 [    217    1403]]
```

In [24]:
```
# RANDOM FOREST CLASSIFIER #
RF = RandomForestClassifier(n_estimators=100, random_state=42)
RF.fit(X_train, y_train)
```

Out[24]:
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [25]:
```
#Model evaluation
y_pred_RF = RF.predict(X_test)
print(y_pred_RF)
```

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

In [26]:
```
print("Random Forest Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_RF))
print("Classification Report:", classification_report(y_test, y_pred_RF))
```

```
print("Confusion Matrix:", confusion_matrix(y_test, y_pred_RF))
```

```
Random Forest Model Performance:
Accuracy: 0.9997100251154398
Classification Report:               precision    recall  f1-score   support

           0       1.00      1.00      1.00   1270904
           1       0.98      0.79      0.87      1620

    accuracy                           1.00   1272524
   macro avg       0.99      0.89      0.94   1272524
weighted avg       1.00      1.00      1.00   1272524

Confusion Matrix: [[1270883      21]
 [    348    1272]]
```

In [ ]: