In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report,confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

In [4]:
```
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 [8]:
```
data.dtypes
```

Out[8]:
```
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 [9]:
```
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 [10]:
```
data.describe()
```

Out[10]:

	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 [11]:
```
data.isnull().sum()
```

Out[11]:
```
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 [13]:
```
#Normalize numerical columns
numeric_cols = ['amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest']
data[numeric_cols] = data[numeric_cols].apply(lambda x: (x - x.min()) / (x.max() - x.min()))
```

In [14]:
```
data.head()
```

Out[14]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFra
0	1	PAYMENT	0.000106	C1231006815	0.002855	0.003233	M1979787155	0.000000	0.0	
1	1	PAYMENT	0.000020	C1666544295	0.000357	0.000391	M2044282225	0.000000	0.0	
2	1	TRANSFER	0.000002	C1305486145	0.000003	0.000000	C553264065	0.000000	0.0	
3	1	CASH_OUT	0.000002	C840083671	0.000003	0.000000	C38997010	0.000059	0.0	
4	1	PAYMENT	0.000126	C2048537720	0.000697	0.000603	M1230701703	0.000000	0.0	

In [15]:
```
#Encode categorical columns
data['type'] = data['type'].map({'CASH_OUT': 0, 'CASH_IN': 1, 'DEBIT': 2,'PAYMENT': 3, 'TRANSFER': 4})
data['nameOrig'] = data['nameOrig'].astype('category').cat.codes
data['nameDest'] = data['nameDest'].astype('category').cat.codes
```

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

Out[16]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlagge
0	1	3	0.000106	757869	0.002855	0.003233	1662094	0.000000	0.0	0	
1	1	3	0.000020	2188998	0.000357	0.000391	1733924	0.000000	0.0	0	
2	1	4	0.000002	1002156	0.000003	0.000000	439685	0.000000	0.0	1	
3	1	0	0.000002	5828262	0.000003	0.000000	391696	0.000059	0.0	1	
4	1	3	0.000126	3445981	0.000697	0.000603	828919	0.000000	0.0	0	

In [18]:
```
 #Split data into features (X) and target variable (y)
X = data.drop('isFraud', axis=1)
y = data['isFraud']
```

In [19]:
```
#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=42)
```

In [20]:
```
#Scale data using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [21]:
```
#Define SVM classifier
svm_model = SVC()
#Train SVM model
svm_model.fit(X_train_scaled, y_train)
```

Out[21]:
```
SVC()
```

In [22]:
```
y_pred_svm = svm_model.predict(X_test_scaled)
print(y_pred_svm)
```

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

In [23]:
```
#Evaluate model performance
print("SVM Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Classification Report:", classification_report(y_test, y_pred_svm))
print("Confusion Matrix:", confusion_matrix(y_test, y_pred_svm))
```

```
SVM Model Performance:
Accuracy: 0.9992668114707464
Classification Report:               precision    recall  f1-score   support

           0       1.00      1.00      1.00   1270904
           1       1.00      0.43      0.60      1620

    accuracy                           1.00   1272524
   macro avg       1.00      0.71      0.80   1272524
weighted avg       1.00      1.00      1.00   1272524

Confusion Matrix: [[1270901       3]
 [    930     690]]
```

In [24]:
```
# Gradient Boosting classifier #
gb_model = GradientBoostingClassifier(
n_estimators=100,
learning_rate=0.1,
max_depth=5,
random_state=42
)
#Train Gradient Boosting model
gb_model.fit(X_train, y_train)
```

Out[24]:
```
GradientBoostingClassifier(max_depth=5, random_state=42)
```

In [25]:
```
 #Make predictions on test set
y_pred_gb = gb_model.predict(X_test)
print(y_pred_gb)
```

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

In [26]:
```
#Evaluate model performance
print("Gradient Boosting Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_gb))
print("Classification Report:", classification_report(y_test, y_pred_gb))
print("Confusion Matrix:", confusion_matrix(y_test, y_pred_gb))
```

```
Gradient Boosting Model Performance:
Accuracy: 0.9995394978798042
Classification Report:               precision    recall  f1-score   support

           0       1.00      1.00      1.00   1270904
           1       0.93      0.69      0.79      1620

    accuracy                           1.00   1272524
   macro avg       0.96      0.85      0.90   1272524
weighted avg       1.00      1.00      1.00   1272524

Confusion Matrix: [[1270815      89]
 [    497    1123]]
```

In [27]:
```
# Naive Bayes classifier #
nb_model = GaussianNB()
#Train Naive Bayes model
nb_model.fit(X_train, y_train)
```

Out[27]:

▾ GaussianNB

GaussianNB()

In [28]:

```
#Make predictions on test set
y_pred_nb = nb_model.predict(X_test)
print(y_pred_nb)
```

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

In [29]:

```
#Evaluate model performance
print("Naive Bayes Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Classification Report:", classification_report(y_test, y_pred_nb))
print("Confusion Matrix:", confusion_matrix(y_test, y_pred_nb))
import warnings
warnings.filterwarnings('ignore')
```

```
Naive Bayes Model Performance:
Accuracy: 0.9987269395311994
Classification Report:               precision    recall  f1-score   support

           0       1.00      1.00      1.00   1270904
           1       0.00      0.00      0.00      1620

    accuracy                           1.00   1272524
   macro avg       0.50      0.50      0.50   1272524
weighted avg       1.00      1.00      1.00   1272524

Confusion Matrix: [[1270904       0]
 [   1620       0]]
```

```
C:\Users\swaro\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1565
: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\swaro\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1565
: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\swaro\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1565
: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

In [ ]: