

ONLINE PAYMENT FRAUD DETECTION

Objective:

Create a machine learning system to detect and prevent fraudulent online transactions.

Summary:

Using a Kaggle dataset, the project involves gathering and cleaning transaction data, engineering features to differentiate legitimate and fraudulent activities, and evaluating algorithms such as Random Forest, XGBoost, and Decision Trees. Techniques like SMOTE will address class imbalance. The models will be trained on historical data and validated using cross-validation. The system will integrate a real-time detection model to flag suspicious transactions, with performance assessed using accuracy, precision, and recall metrics.

Problem Statement:

With the rise in online transactions, the incidence of fraudulent activities has also increased, leading to significant financial losses and reputational damage for businesses. The need for a robust system to detect and prevent fraud in real-time has become critical.

Dataset Features :

Step : tells about the unit of time

Type: type of online transaction

Amount: the amount of the transaction

NameOrig: customer starting the transaction

OldbalanceOrg : balance before the transaction

NewbalanceOrg : balance after the transaction

NameDest : account that receives the transaction

OldbalanceDest : initial balance of receiver before the transaction

NewbalanceDest : the new balance of the receiver after the transaction


IsFraud : fraud transaction

✓ Importing Libraries and Tools

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
import xgboost
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```


✓ Loading and Exploring the Dataset

```
df=pd.read_csv('/content/drive/MyDrive/Datasets/Online Payment Fraud Detection.csv')
df
```




	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M15
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M20
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C5
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M12
...
1048570	95	CASH_OUT	132557.35	C1179511630	479803.00	347245.65	C4
1048571	95	PAYMENT	9917.36	C1956161225	90545.00	80627.64	M6
1048572	95	PAYMENT	14140.05	C2037964975	20545.00	6404.95	M13
1048573	95	PAYMENT	10020.05	C1633237354	90605.00	80584.95	M15
1048574	95	PAYMENT	11450.03	C1264356443	80584.95	69134.92	M6

1048575 rows x 10 columns




```
df.shape
df.duplicated().sum()
df.isnull().sum()
df.describe()
```

 (1048575, 10)

```
df.duplicated().sum()
```

 0


```
df.isnull().sum()
```



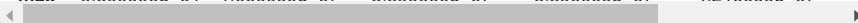
step	0
type	0
amount	0
nameOrig	0
oldbalanceOrg	0
newbalanceOrig	0
nameDest	0
oldbalanceDest	0
newbalanceDest	0
isFraud	0
dtype: int64	

There is no missing values in this dataset

```
df.describe()
```



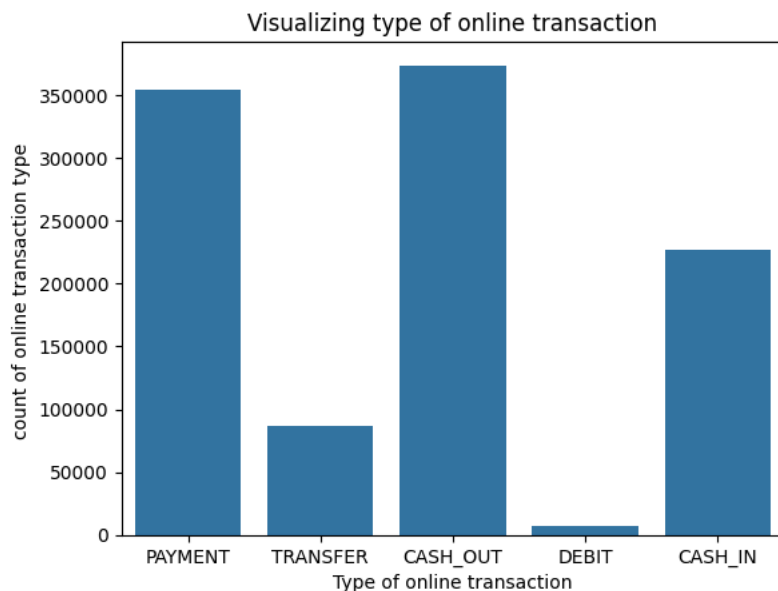
	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newt
count	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.
mean	2.696617e+01	1.586670e+05	8.740095e+05	8.938089e+05	9.781600e+05	1
std	1.562325e+01	2.649409e+05	2.971751e+06	3.008271e+06	2.296780e+06	2.
min	1.000000e+00	1.000000e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.
25%	1.500000e+01	1.214907e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.
50%	2.000000e+01	7.634333e+04	1.600200e+04	0.000000e+00	1.263772e+05	2.
75%	3.900000e+01	2.137619e+05	1.366420e+05	1.746000e+05	9.159235e+05	1.
max	9.500000e+01	1.000000e+07	3.890000e+07	3.890000e+07	4.210000e+07	4.



✓ Data Visualization and Fraud Analysis

```
sns.countplot(x="type",data=df)
plt.title ("Visualizing type of online transaction")
plt.xlabel("Type of online transaction")
plt.ylabel("count of online transaction type ")
```

```
Text(0, 0.5, 'count of online transaction type')
```



From this chart, it is seen that cash_out and payment is the most common type of online transaction that customers use.

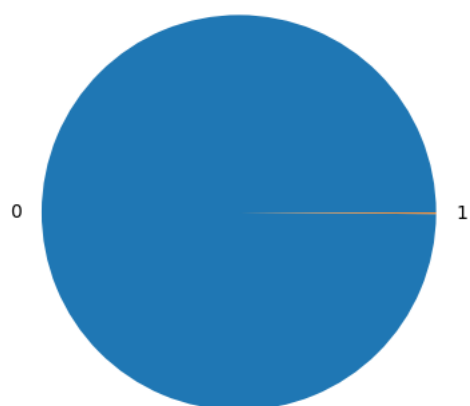
```
df['isFraud'].value_counts()
```

```
isFraud
0    1047433
1      1142
Name: count, dtype: int64
```

1,142 transactions have been tagged as fraudulent in the dataset

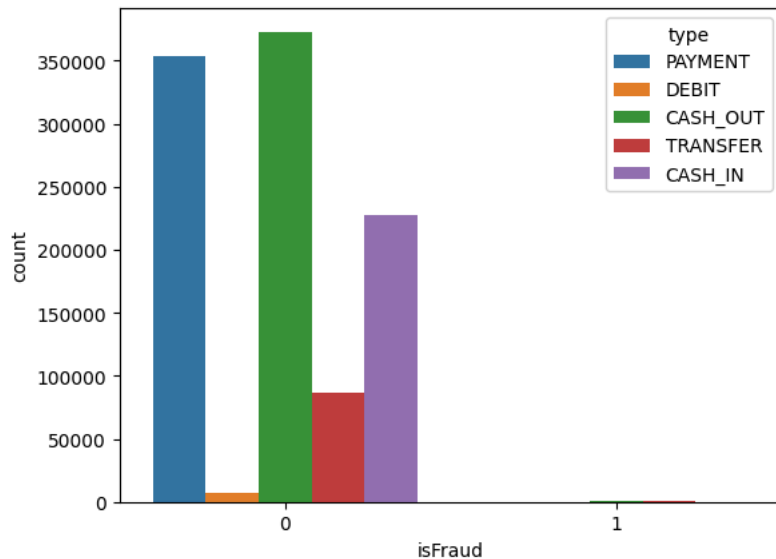
```
plt.pie(labels=[0,1],x=df['isFraud'].value_counts()) #,autopct='%1.1f%%')
```

```
([<matplotlib.patches.Wedge at 0x7f2a589f2710>,
 <matplotlib.patches.Wedge at 0x7f2a589f2620>],
 [Text(-1.0999935613563063, 0.003763638488259466, '0'),
 Text(1.0999935613686946, -0.0037636384867559027, '1')])
```



```
sns.countplot(x='isFraud',data=df,hue='type')
```

```
<Axes: xlabel='isFraud', ylabel='count'>
```



From all this, its shows that most of the online transactions customers does is not fraudulent. Also the dataset is not balance

```
sns.distplot(df['step'])
```

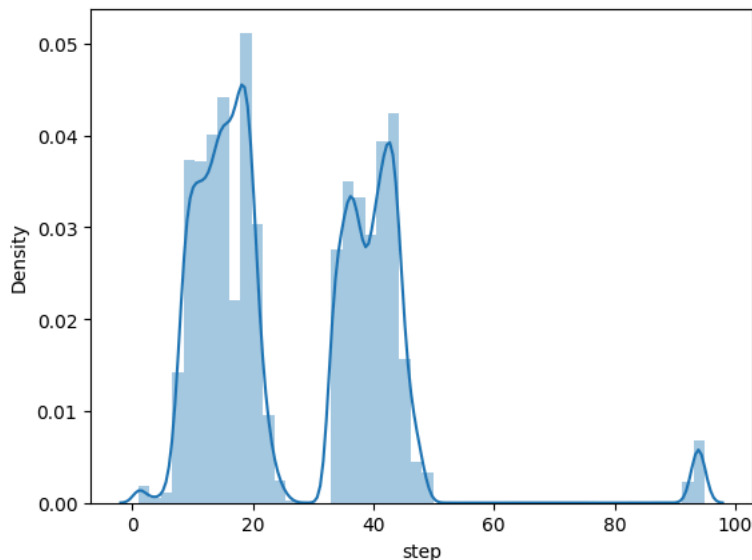
```
<ipython-input-12-664699a34125>:1: UserWarning:
```

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see <https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

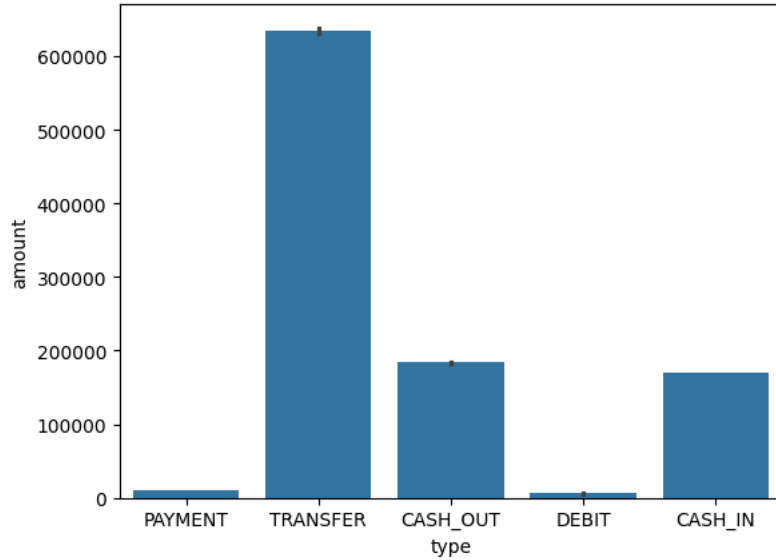
```
sns.distplot(df['step'])
<Axes: xlabel='step', ylabel='Density'>
```



The above graph indicates the distribution of the step column.

```
sns.barplot(x='type',y='amount',data=df)
```

```
<Axes: xlabel='type', ylabel='amount'>
```



In this chart, 'transfer' type has the maximum amount of money being transferred from customers to the recipient.

```
df.head()
```

	step	type	amount	nameOrig	oldbalanceOrig	newbalanceOrig	nameDes
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M197978715
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M204428222
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C55326406
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C3899701
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M12307017C

✓ PERFORMING FEATURE ENGINEERING

```
df['type'].value_counts()
```

```
type
CASH_OUT    373641
PAYMENT     353873
CASH_IN     227130
TRANSFER     86753
DEBIT        7178
Name: count, dtype: int64
```

```
df_new=pd.get_dummies(df['type'],drop_first=True)
df_new=df_new.astype(int)
df_new.head()
```

	CASH_OUT	DEBIT	PAYMENT	TRANSFER
0	0	0	1	0
1	0	0	1	0
2	0	0	0	1
3	1	0	0	0
4	0	0	1	0

```
df.drop('type',axis=1,inplace=True)
```

```
df=pd.concat([df,df_new], axis=1)
df.head()
```

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

df.shape

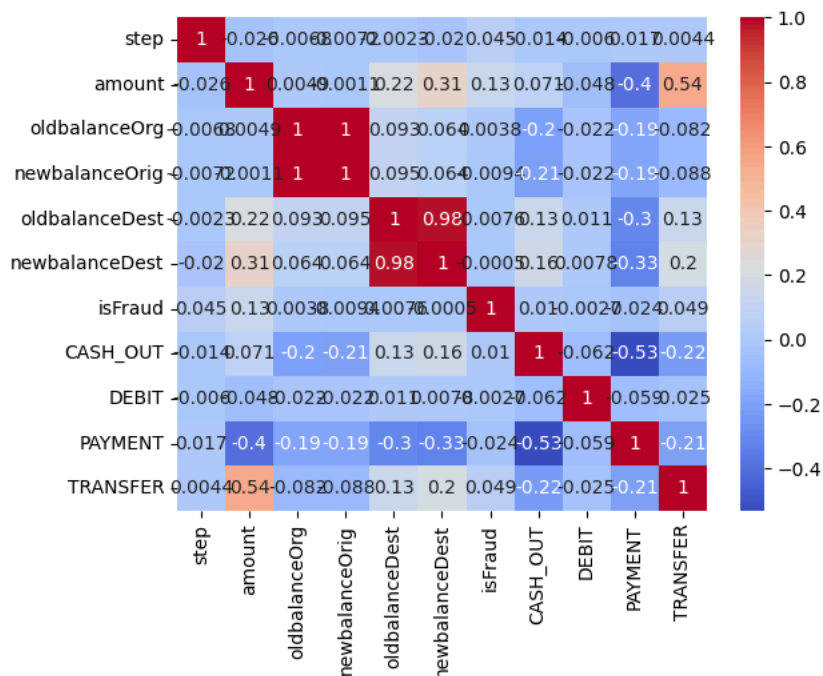
(1048575, 13)

```
df[['step', 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest', 'isFraud', 'CASH_OUT', 'DEBIT', 'PAYMENT', 'TRANSFER']
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
step	1.000000	-0.025996	-0.006780	-0.007180	-0.002251	-0.019503
amount	-0.025996	1.000000	0.004864	-0.001133	0.215558	0.311936
oldbalanceOrg	-0.006780	0.004864	1.000000	0.999047	0.093305	0.064049
newbalanceOrig	-0.007180	-0.001133	0.999047	1.000000	0.095182	0.063725
oldbalanceDest	-0.002251	0.215558	0.093305	0.095182	1.000000	0.978403
newbalanceDest	-0.019503	0.311936	0.064049	0.063725	0.978403	1.000000
isFraud	0.045030	0.128862	0.003829	-0.009438	-0.007552	0.045030
CASH_OUT	-0.013746	0.071255	-0.204549	-0.214548	0.130120	-0.013746
DEBIT	-0.005992	-0.047878	-0.022109	-0.022489	0.010704	-0.005992
PAYMENT	0.017102	-0.397464	-0.186253	-0.190113	-0.303959	0.017102
TRANSFER	0.004375	0.539278	-0.081976	-0.087814	0.130362	0.004375

```
correlation_matrix = df[['step', 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest', 'isFraud', 'CASH_OUT', 'DEBIT', 'PAYMENT', 'TRANSFER']].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
```

<Axes: >



```
x=df.drop(['isFraud', 'nameOrig', 'nameDest'],axis=1)
y=df['isFraud']
```

```
df['isFraud'].value_counts()
```

```
isFraud
0    1047433
1      1142
Name: count, dtype: int64
```

✓ Model Selection, Training and Validation

```
over_sampler=SMOTE()
x_train_sampled,y_train_sampled=over_sampler.fit_resample(x,y)
```

```
x_train,x_test,y_train,y_test=train_test_split(x_train_sampled,y_train_sampled,test_size=0.2,random_state=42)
```

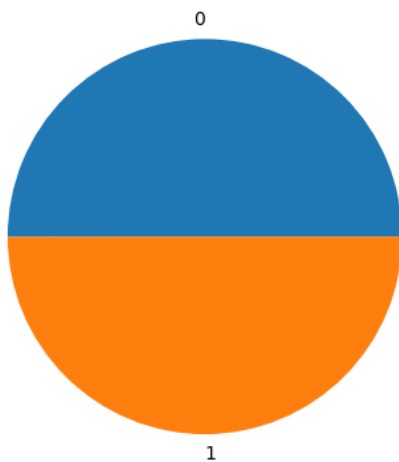
```
sc=StandardScaler()
x_train_scaled=sc.fit_transform(x_train)
x_test_scaled=sc.transform(x_test)
```

```
y_train.value_counts()
```

```
isFraud
0    838313
1    837579
Name: count, dtype: int64
```

```
plt.pie(labels=[0,1],x=y_train.value_counts())
```

```
([<matplotlib.patches.Wedge at 0x7f2a3e813f10>,
 <matplotlib.patches.Wedge at 0x7f2a3e813e20>],
 [Text(-0.0007567662923739112, 1.0999997396839596, '0'),
 Text(0.0007567662923735321, -1.0999997396839596, '1')])
```



RandomForestClassifier

```
rf=RandomForestClassifier(n_estimators=10,max_depth=10,min_samples_split=25,bootstrap=False)
rf.fit(x_train_scaled,y_train)
y_p=rf.predict(x_test_scaled)
```

```
print(accuracy_score(y_test,y_p))
```

```
0.988278508928955
```

```
print(classification_report(y_test,y_p))
```

```
precision    recall  f1-score   support

0           0.99      0.99      0.99     209120
1           0.99      0.99      0.99     209854


accuracy          0.99      0.99      0.99     418974
macro avg         0.99      0.99      0.99     418974
weighted avg      0.99      0.99      0.99     418974
```

```
print(confusion_matrix(y_test, y_p))
```

```
[[206654  2466]
 [ 2445 207409]]
```

Xgboost

```
xb=xgboost.XGBClassifier(n_estimators=1000,max_depth=10,learning_rate=0.001,min_samples_split=10)
xb.fit(x_train_scaled,y_train)
```

 /usr/local/lib/python3.10/dist-packages/xgboost/core.py:160: UserWarning: [17:02:16]
Parameters: { "min_samples_split" } are not used.

```
warnings.warn(smsg, UserWarning)
```


```

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=0.001, max_bin=None,
               max_cat_threshold=None, max_cat_to_onehot=None,
               max_delta_step=None, max_depth=10, max_leaves=None,
               min_child_weight=None, min_samples_split=10, missing=nan,
               monotone_constraints=None, multi_strategy=None, n_estimators=1000,
               n_jobs=None, num_parallel_tree=None, ...)


```

```
y_pred_xb=xb.predict(x_test_scaled)
```

```
print(accuracy_score(y_test,y_pred_xb))
```


 0.9902213502508509

```
print(classification_report(y_test,y_pred_xb))
```




	precision	recall	f1-score	support
0	0.99	0.99	0.99	209120
1	0.99	0.99	0.99	209854
accuracy			0.99	418974
macro avg	0.99	0.99	0.99	418974
weighted avg	0.99	0.99	0.99	418974

```
print(confusion_matrix(y_test, y_pred_xb))
```


 [[206345 2775]
[1322 208532]]

DecisionTreeClassifier


```
model=DecisionTreeClassifier(splitter='best',min_samples_split=4,max_depth=9,criterion='log_loss')
model.fit(x_train_scaled,y_train)
```

 **DecisionTreeClassifier**
DecisionTreeClassifier(criterion='log_loss', max_depth=9, min_samples_split=4)


```
y_pred_dc=model.predict(x_test_scaled)
y_pred_dc
```

 array([1, 1, 1, ..., 0, 0, 1])

```
print(accuracy_score(y_test,y_pred_dc))
```

 0.9861805267152616

```
print(classification_report(y_test,y_pred_dc))
```



	precision	recall	f1-score	support
0	0.99	0.98	0.99	209120
1	0.98	0.99	0.99	209854
accuracy			0.99	418974
macro avg	0.99	0.99	0.99	418974
weighted avg	0.99	0.99	0.99	418974

Accuracy

Random forest model : 0.9895602113734981

xgboost model : 0.9902595387780626

Decesion tree model:0.9863428279559113

✓ Conclusion

The development of a machine learning system to detect and prevent fraudulent online transactions has yielded highly accurate models. By utilizing a Kaggle dataset and performing thorough data preprocessing and feature engineering, we were able to create effective models for fraud detection.

The Random Forest model achieved an accuracy of 98.96%, demonstrating strong performance in identifying fraudulent transactions. The XGBoost model outperformed the other models with an impressive accuracy of 99.03%, indicating its superior capability in handling complex fraud patterns. The Decision Tree model also performed well with an accuracy of 98.63%.

These results highlight the potential of machine learning algorithms in combating online payment fraud, providing a robust solution to enhance transaction security and protect financial systems from fraudulent activities. The high accuracy rates of these models can significantly reduce financial losses and build customer trust in online payment systems.

Double click (or enter) to edit