###

Problem Statement :-) Online Payments Fraud Detection using Python

1.1 Description :

The introduction of online payment systems has helped a lot in the ease of payments. But, at the same time, it increased in payment frauds. Online payment frauds can happen with anyone using any payment system, especially while making payments using a credit card. That is why detecting online payment fraud is very important for credit card companies to ensure that the customers are not getting charged for the products and services they never paid. If you want to learn how to detect online payment frauds, this article is for you. In this article, I will take you through the task of online payments fraud detection with machine learning using Python.

1.2 Method:

To identify online payment fraud with machine learning, we need to train a machine learning model for classifying fraudulent and non-fraudulent payments. For this, we need a dataset containing information about online payment fraud, so that we can understand what type of transactions lead to fraud. For this task, I collected a dataset from Kaggle, which contains historical information about fraudulent transactions which can be used to detect fraud in online payments. Below are all the columns from the dataset I'm using here:

2. Importing All Necessary Libraries

```
# Basic Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings(action='ignore')

# Preprocessing Libraries
from sklearn.preprocessing import RobustScaler

# Model Training Libraries
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedGroupKFold
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from collections import Counter
from imblearn.under_sampling import NearMiss
from imblearn.over_sampling import RandomOverSampler
from imblearn.combine import SMOTETomek
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
```

2.1 Loading Dataset

1.3 Datset Information:

The below column reference:

1. step: represents a unit of time where 1 step equals 1 hour
2. type: type of online transaction
3. amount: the amount of the transaction
4. nameOrig: customer starting the transaction
5. oldbalanceOrg: balance before the transaction
6. newbalanceOrig: balance after the transaction
7. nameDest: recipient of the transaction
8. oldbalanceDest: initial balance of recipient before the transaction
9. newbalanceDest: the new balance of recipient after the transaction
10. isFraud: fraud transaction

- Dataset link---https://www.kaggle.com/datasets/rupakroy/online-payments-fraud-detection-dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
df=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/ONLINE_PAYMENT_FRAUD_DETECTION/Online_Fraud.csv")
df.head()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFrau
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	

```
df.shape
```

```
(6362620, 11)
```

Here Observation =>6362620, Feature => 11

3. EDA

```
df.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
```

3.1 Handling Null value

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

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

- **No Null value present**

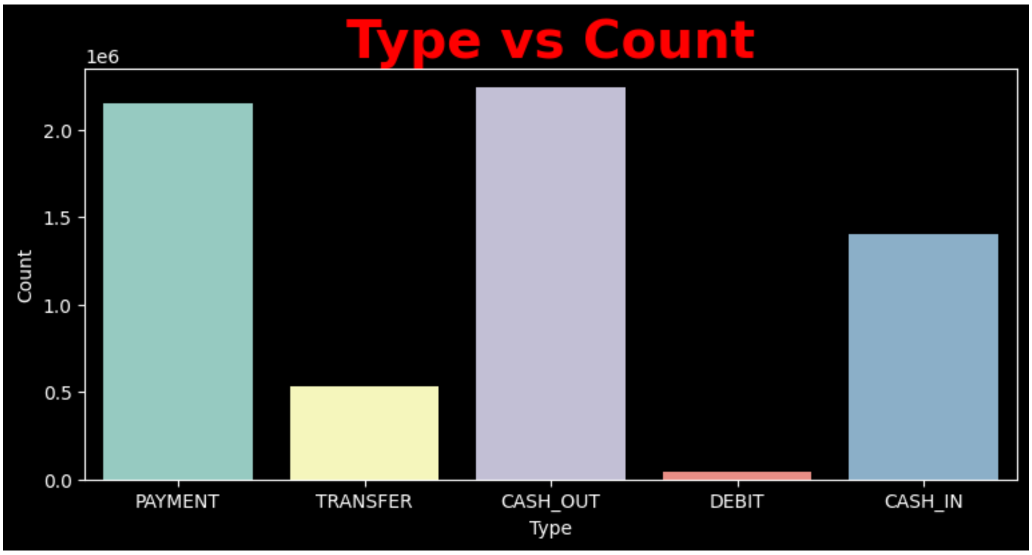
3.2 Visualization

```
df.columns.unique()
```

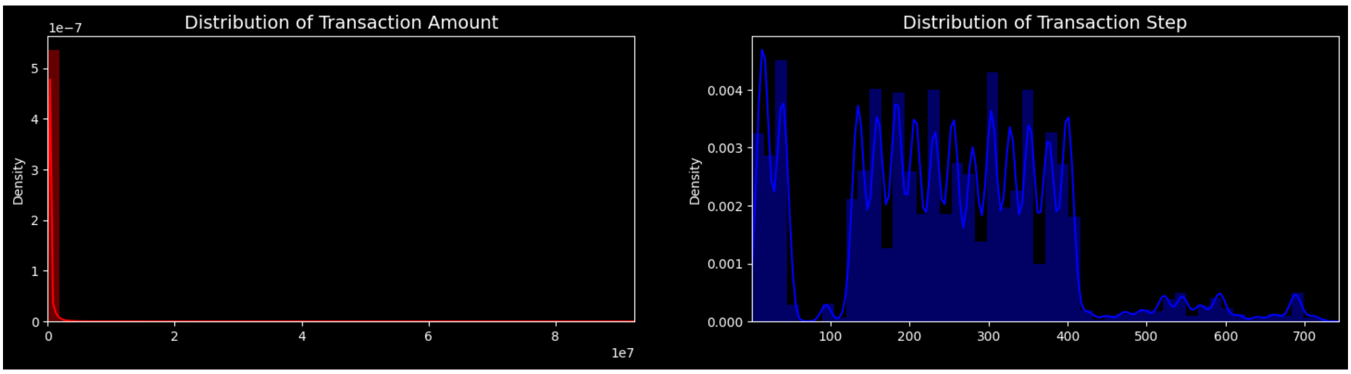
```
Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOrig',
       'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud',
```

```
             'isFlaggedFraud'],
            dtype='object')
```

```
# type of payment
plt.style.use('dark_background')
plt.rcParams.update({'text.color':'white'})
plt.figure(figsize=(9,4))
plt.title('Type vs Count', fontsize=28, fontweight='bold', color='red')
sns.countplot(data=df, x='type')
plt.xlabel('Type')
plt.ylabel('Count')
plt.show()
```

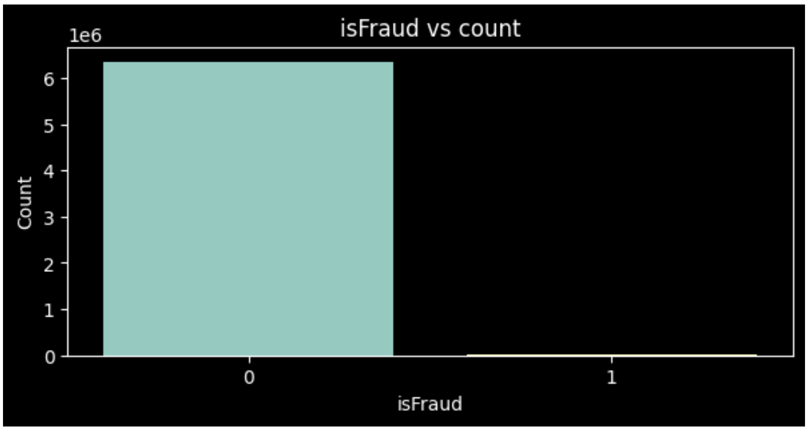



```
# Plotting subplot for amount and time column
plt.style.use('dark_background')
plt.rcParams.update({'text.color':'white'})
fig, ax = plt.subplots(1, 2, figsize=(18,4))
amount_val = df['amount'].values
time_val = df['step'].values

sns.distplot(amount_val, ax=ax[0], color='r')
ax[0].set_title('Distribution of Transaction Amount', fontsize=14)
ax[0].set_xlim([min(amount_val), max(amount_val)])

sns.distplot(time_val, ax=ax[1], color='b')
ax[1].set_title('Distribution of Transaction Step', fontsize=14)
ax[1].set_xlim([min(time_val), max(time_val)])
plt.show()
```




```
# Countplot of 'isFraud'
plt.style.use('dark_background')
plt.rcParams.update({'text.color':'white'})
plt.figure(figsize=(7,3))
plt.title('isFraud vs count')
sns.countplot(data=df,x='isFraud')
plt.xlabel('isFraud')
```

```
plt.ylabel('Count')
plt.show()
```



Note: We can see from above visualization it is an imbalanced dataset, look 0 and 1

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

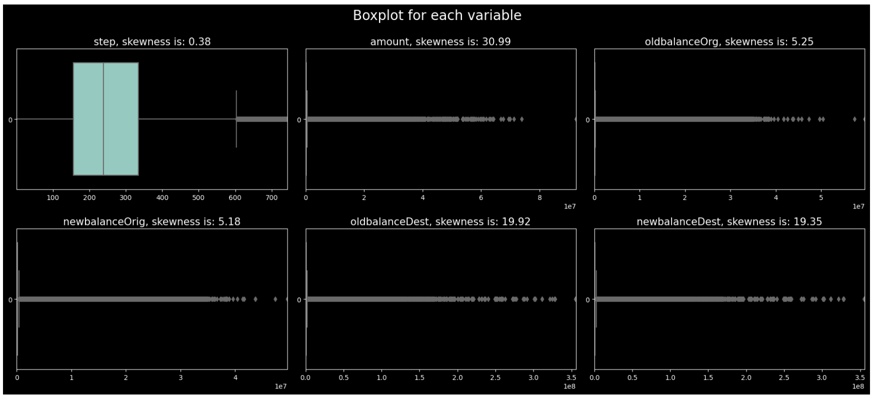
```
<bound method IndexOpsMixin.value_counts of 0          0
1          0
2          1
3          1
4          0
          ..
6362615    1
6362616    1
6362617    1
6362618    1
6362619    1
Name: isFraud, Length: 6362620, dtype: int64>
```

```
# Let's look at the percentage of each category in isFraud column(target column)
print("No Frauds:",df['isFraud'].value_counts()[0]/len(df['isFraud'])*100)
print("Frauds:",df['isFraud'].value_counts()[1]/len(df['isFraud'])*100)
```

```
No Frauds: 99.87091795518198
Frauds: 0.12908204481801522
```

```
numerical=['step','amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest']
```

```
# Boxplot for each variable in numerical list
plt.style.use('dark_background')
plt.rcParams.update({'text.color':'white'})
def boxplots_visual(data,column):
    fig, ax = plt.subplots(2,3,figsize=(18,8))
    fig.suptitle('Boxplot for each variable',y=1, size=20)
    ax=ax.flatten()
    for i,feature in enumerate(column):
        sns.boxplot(data=data[feature],ax=ax[i], orient='h')
        ax[i].set_title(feature+ ', skewness is: '+str(round(data[feature].skew(axis = 0, skipna = True),2)),fontsize=15)
        ax[i].set_xlim([min(data[feature]), max(data[feature])])
boxplots_visual(data=df,column=numerical)
plt.tight_layout()
```




```
# Checking nameOrig,nameDest column
nameOrig=df['nameOrig'].unique()
print("Unique in nameOrig:",len(nameOrig))
print(nameOrig)

nameDest=df['nameDest'].unique()
print("Unique in nameDest:",len(nameDest))
print(nameDest)
```

```
Unique in nameOrig: 6353307
['C1231006815' 'C1666544295' 'C1305486145' ... 'C1162922333' 'C1685995037'
 'C1280323807']
Unique in nameDest: 2722362
['M1979787155' 'M2044282225' 'C553264065' ... 'C1850423904' 'C1881841831'
 'C2080388513']
```

```
# Checking isFlaggedFraud column
df['isFlaggedFraud'].value_counts()
```

```
0    6362604
1         16
Name: isFlaggedFraud, dtype: int64
```

```
# Dropping columns that are not needed
df.drop(['nameOrig','nameDest','isFlaggedFraud'],axis=1,inplace=True)
```

```
# Applying onehot encoding on type column
df=pd.get_dummies(data=df,columns=['type'],drop_first=True)
df.head()
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
0	1	9839.64	170136.0	160296.36	0.0	0.C
1	1	1864.28	21249.0	19384.72	0.0	0.C
2	1	181.00	181.0	0.00	0.0	0.C
3	1	181.00	181.0	0.00	21182.0	0.C
4	1	11668.14	41554.0	29885.86	0.0	0.C

```
# We are using RobustScaler to scale down the numerical features as RobustScaler is less prone to outliers
scale=RobustScaler()
for feature in numerical:
    df[feature]=scale.fit_transform(df[feature].values.reshape(-1, 1))
df.head()
```

step amount oldbalanceOrg newbalanceOrig oldbalanceDest newbalance

Model Training

```
# Splitting our data into independent and dependent features
x=df.drop('isFraud',axis=1)
y=df['isFraud']
```

```
x.columns
```

```
Index(['step', 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest',
       'newbalanceDest', 'type_CASH_OUT', 'type_DEBIT', 'type_PAYMENT',
       'type_TRANSFER'],
      dtype='object')
```

```
df[df['isFraud']==1]
```

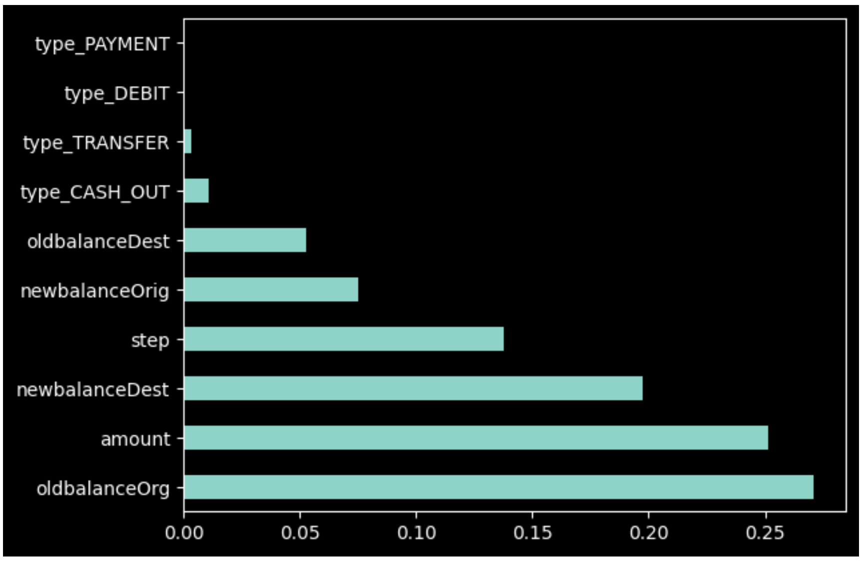
	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	type_CASH_OUT	type_DE
2	-1.329609	-0.382380	-0.130708	0.0	-0.140722	-0.193057	1	0	
3	-1.329609	-0.382380	-0.130708	0.0	-0.118260	-0.193057	1	1	
251	-1.329609	-0.368941	-0.106248	0.0	-0.140722	-0.193057	1	0	
252	-1.329609	-0.368941	-0.106248	0.0	-0.112937	-0.193057	1	1	
680	-1.329609	-0.280261	0.055165	0.0	-0.140722	-0.193057	1	0	
...	...	...	...	...	...	...	...	...	
6362615	2.815642	1.355693	3.032881	0.0	-0.140722	0.112438	1	1	
6362616	2.815642	31.927899	58.679504	0.0	-0.140722	-0.193057	1	0	
6362617	2.815642	31.927899	58.679504	0.0	-0.068096	5.544730	1	1	
6362618	2.815642	3.968274	7.788223	0.0	-0.140722	-0.193057	1	0	
6362619	2.815642	3.968274	7.788223	0.0	6.762614	6.426280	1	1	

8213 rows × 11 columns

```
# Feature Importance
from sklearn.ensemble import ExtraTreesRegressor
model = ExtraTreesRegressor()
model.fit(x,y)
print(model.feature_importances_)
```

```
[0.13762845 0.25149849 0.27122025 0.07511738 0.05286839 0.1973012
 0.01076672 0.         0.         0.00359912]
```

```
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=x.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```




```
from sklearn.model_selection import StratifiedKFold
```

```
# Doing train_test_split
X_train,X_test,y_train,y_test=train_test_split(x,y,train_size=0.7)
# Applying StratifiedKFold
skf=StratifiedKFold(n_splits=3, shuffle=False, random_state=None)
```

```
model1=LogisticRegression()
param={'C':10.0 **np.arange(-1,2)}
lrs=RandomizedSearchCV(model1,param,cv=skf,n_jobs=-1,scoring='accuracy')
lrs.fit(X_train,y_train)
```

```
▸ RandomizedSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

```
y_pred=lrs.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[1906210     176]
 [   1203    1197]]
0.9992775512812856
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1906386
           1       0.87      0.50      0.63      2400

    accuracy                           1.00   1908786
   macro avg       0.94      0.75      0.82   1908786
weighted avg       1.00      1.00      1.00   1908786
```