Online Payment Fraud Detection using Machine Learning.

by SURYANSHU VERMA

Importing the libraries

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from  ydata_profiling import ProfileReport
```

Importing the dataset

In [ ]:
```
dataset = pd.read_csv('Dataa.csv')
dataset.drop(columns=["nameDest", "isFlaggedFraud", "oldbalanceDest", "newbalanceDe
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [ ]:
```
dataset
```

Out[ ]:

	type	amount	oldbalanceOrg	newbalanceOrig	isFraud
0	PAYMENT	9839.64	170136.00	160296.36	0
1	PAYMENT	1864.28	21249.00	19384.72	0
2	TRANSFER	181.00	181.00	0.00	1
3	CASH_OUT	181.00	181.00	0.00	1
4	PAYMENT	11668.14	41554.00	29885.86	0
...	...	...	...	...	...
6362615	CASH_OUT	339682.13	339682.13	0.00	1
6362616	TRANSFER	6311409.28	6311409.28	0.00	1
6362617	CASH_OUT	6311409.28	6311409.28	0.00	1
6362618	TRANSFER	850002.52	850002.52	0.00	1
6362619	CASH_OUT	850002.52	850002.52	0.00	1

6362620 rows × 5 columns

YData Profiling

In [ ]:
```
profile = ProfileReport(dataset, title='Pandas Profiling Report', explorative=True)
```

In [ ]:
```
profile.to_notebook_iframe()
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	5
Number of observations	6362620
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	101134
Duplicate rows (%)	1.6%
Total size in memory	536.5 MiB
Average record size in memory	88.4 B

Variable types

Categorical	2
Numeric	3

Alerts

Dataset has 101134 (1.6%) duplicate rows	**Duplicates**
`newbalanceOrig` is highly overall correlated with `oldbalanceOrg`	**High correlation**
`oldbalanceOrg` is highly overall correlated with	**High correlation**

In [ ]:
```
print(X)
```

```
[['PAYMENT' 9839.64 170136.0 160296.36]
 ['PAYMENT' 1864.28 21249.0 19384.72]
 ['TRANSFER' 181.0 181.0 0.0]
 ...
 ['CASH_OUT' 6311409.28 6311409.28 0.0]
 ['TRANSFER' 850002.52 850002.52 0.0]
 ['CASH_OUT' 850002.52 850002.52 0.0]]
```

In [ ]:
```
print(y)
```

```
[0 0 1 ... 1 1 1]
```

Checking the missing Values

In [ ]:
```
dataset.isnull().sum()
```

Out[ ]:
```
type              0
amount            0
oldbalanceOrg     0
newbalanceOrig    0
isFraud           0
dtype: int64
```

In [ ]:
```
dataset
```

Out[ ]:

	type	amount	oldbalanceOrg	newbalanceOrig	isFraud
0	PAYMENT	9839.64	170136.00	160296.36	0
1	PAYMENT	1864.28	21249.00	19384.72	0
2	TRANSFER	181.00	181.00	0.00	1
3	CASH_OUT	181.00	181.00	0.00	1
4	PAYMENT	11668.14	41554.00	29885.86	0
...	...	...	...	...	...
6362615	CASH_OUT	339682.13	339682.13	0.00	1
6362616	TRANSFER	6311409.28	6311409.28	0.00	1
6362617	CASH_OUT	6311409.28	6311409.28	0.00	1
6362618	TRANSFER	850002.52	850002.52	0.00	1
6362619	CASH_OUT	850002.52	850002.52	0.00	1

6362620 rows × 5 columns

Encoding categorical data

Encoding the Independent Variable

In [ ]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder=
X = np.array(ct.fit_transform(X))
```

In [ ]:
```
print(pd.DataFrame(X).head(5))
```

```
     0    1    2    3    4         5         6          7
0  0.0  0.0  0.0  1.0  0.0   9839.64  170136.0  160296.36
1  0.0  0.0  0.0  1.0  0.0   1864.28   21249.0   19384.72
2  0.0  0.0  0.0  0.0  1.0     181.0     181.0        0.0
3  0.0  1.0  0.0  0.0  0.0     181.0     181.0        0.0
4  0.0  0.0  0.0  1.0  0.0  11668.14   41554.0   29885.86
```

In [ ]:
```
print(y)
```

```
[0 0 1 ... 1 1 1]
```

Splitting the dataset into the Training set and Test set

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_s
```

In [ ]:
```
print(X_train)
```

```
[[0.0 0.0 0.0 ... 1607.27 0.0 0.0]
 [0.0 0.0 0.0 ... 16958.15 416.0 0.0]
 [0.0 1.0 0.0 ... 249226.07 11335.0 0.0]
 ...
 [0.0 1.0 0.0 ... 128095.4 0.0 0.0]
 [0.0 0.0 0.0 ... 5504.7 0.0 0.0]
 [0.0 0.0 0.0 ... 41331.38 2223.0 0.0]]
```

In [ ]:
```
print(X_test)
```

```
[[1.0 0.0 0.0 ... 23557.12 8059.0 31616.12]
 [0.0 0.0 0.0 ... 6236.13 0.0 0.0]
 [0.0 0.0 0.0 ... 33981.87 18745.72 0.0]
 ...
 [0.0 1.0 0.0 ... 353801.58 0.0 0.0]
 [0.0 0.0 0.0 ... 37659.34 360829.27 323169.93]
 [0.0 0.0 0.0 ... 609.64 0.0 0.0]]
```

In [ ]:
```
print(y_train)
```

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

In [ ]:
```
print(y_test)
```

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

Model Selection

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
LRC=LogisticRegression(random_state = 0)
LRC.fit(X_train, y_train)
DTC=DecisionTreeClassifier(criterion="entropy", max_depth=25, random_state=42)
DTC.fit(X_train, y_train)
```

Out[ ]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=25, random_state=42)
```

Predicting a new result

In [ ]:
```
pd.DataFrame(X)
```

Out[ ]:

	0	1	2	3	4	5	6	7
0	0.0	0.0	0.0	1.0	0.0	9839.64	170136.0	160296.36
1	0.0	0.0	0.0	1.0	0.0	1864.28	21249.0	19384.72
2	0.0	0.0	0.0	0.0	1.0	181.0	181.0	0.0
3	0.0	1.0	0.0	0.0	0.0	181.0	181.0	0.0
4	0.0	0.0	0.0	1.0	0.0	11668.14	41554.0	29885.86
...	...	...	...	...	...	...	...	...
6362615	0.0	1.0	0.0	0.0	0.0	339682.13	339682.13	0.0
6362616	0.0	0.0	0.0	0.0	1.0	6311409.28	6311409.28	0.0
6362617	0.0	1.0	0.0	0.0	0.0	6311409.28	6311409.28	0.0
6362618	0.0	0.0	0.0	0.0	1.0	850002.52	850002.52	0.0
6362619	0.0	1.0	0.0	0.0	0.0	850002.52	850002.52	0.0

6362620 rows × 8 columns

In [ ]:
```
print(LRC.predict(X_test))
print(DTC.predict(X_test))
```

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

In [ ]:
```
y_Pred=DTC.predict(X_test)
y_pred=LRC.predict(X_test)
```

In [ ]:
```
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1))
print(".....................Below With DTC.....................")
print(np.concatenate((y_Pred.reshape(len(y_Pred),1), y_test.reshape(len(y_test),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
.........................Below With DT
C.................................................
[[0 0]
 [0 0]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
```

Confusion metrix When Model is LRC

In [ ]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[1270788      89]
 [    986     661]]
```

Accuracy Score Of LRC

In [ ]:
```
accuracy_score(y_test, y_pred)
```

Out[ ]:
```
0.9991552222197774
```

F1 Score Of LRC

In [ ]:
```
f1_score(y_test, y_pred)
```

Out[ ]:
```
0.5515227367542762
```

Confusion metrix When Model is DTC

In [ ]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score
cm = confusion_matrix(y_test, y_Pred)
print(cm)
```

```
[[1270701     176]
 [    223    1424]]
```

Accuracy Score Of DTC

In [ ]:
```
accuracy_score(y_test, y_Pred)
```

Out[ ]:
```
0.9996864499215732
```

F1 Score Of DTC

In [ ]:
```
f1_score(y_test, y_Pred)
```

Out[ ]:
```
0.8771173390822298
```

Cross Validation

In [ ]:
```
from sklearn.model_selection import cross_val_score, StratifiedKFold
# Create Stratified k-fold cross-validation iterator
stratified_kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Perform cross-validation
scores = cross_val_score(DTC, X, y, cv=stratified_kfold, scoring='accuracy')

# Print the cross-validation scores
print("Cross-validation scores:", scores)

# Calculate and print the mean and standard deviation of the scores
print("Mean accuracy:", np.mean(scores))
print("Standard deviation of accuracy:", np.std(scores))
```

```
Cross-validation scores: [0.99969509 0.99967466 0.99969038 0.99971867 0.99976425 0.9
9969981
 0.99970452 0.99969038 0.99969195 0.99971238]
Mean accuracy: 0.9997042099009527
Standard deviation of accuracy: 2.3214085314090805e-05
```

Now let's classify whether a transaction is a fraud or not by feeding about a transaction into the model:

In [ ]:
```
# prediction by DTC =  Decision Tree Classifier
#features = [type, amount, oldbalanceOrg, newbalanceOrig]
#                            payment type       amount  oldbal   newbal
features = np.array([[0.0, 0.0, 0.0, 1.0, 0.0, 9000.60, 9000.60, 0.0]])
print(DTC.predict(features))
```

```
[0]
```

Predicted value Is 0, So Its Not A Fraud Transaction

CONCLUSION

In the pursuit of identifying online fraud, the Decision Tree Classifier emerged as the standout choice. Its high F1 score and impressive accuracy underscored its effectiveness in distinguishing fraudulent transactions. This selection reflects a commitment to excellence and precision, ensuring the protection of digital integrity with sophistication and poise.