SMOTE

Case Study on "Ether Fraud Transaction

In [28]:
```
import pandas as pd
```

In [29]:
```
data=pd.read_excel(r'C:\Users\Admin\Desktop\Ether Fraud Transaction case study.xlsx')
data.head()
```

Out[29]:

	FLAG	Avg _Min_Sent_txn	Avg _Min_Recv_txn	Time_Diff	Min value received	Max value received	Min value sent	Max value sent	Total transactions
0	0	21.80	0.00	65.40	101.000000	101.000000	1.000000	84.176518	4
1	0	1073.76	1476.59	24563.10	0.216334	6.999559	0.000000	7.557834	21
2	0	13.57	10125.40	638755.18	0.010053	20.000000	0.008583	19.999769	126
3	0	163765.17	0.00	327530.33	0.002457	0.002457	0.000000	0.000508	3
4	0	22371.43	29004.65	596613.23	0.005441	1000.000000	0.000000	1000.000000	25

In [34]:
```
# Shape of the data

data.shape
```

Out[34]:
```
(9840, 9)
```

In [31]:
```
data.isnull().sum()
```

Out[31]:
```
FLAG                  0
Avg _Min_Sent_txn     0
Avg _Min_Recv_txn     0
Time_Diff             0
Min value received    0
Max value received    0
Min value sent        0
Max value sent        0
Total transactions    0
dtype: int64
```

In [35]:
```
# count the values in FLAG

data.FLAG.value_counts()
```

Out[35]:
```
0    7661
1    2179
Name: FLAG, dtype: int64
```

In [37]:
```
# Dependent and Independents variables
x = data.iloc[:,1:]
y = data.FLAG
```

In [38]:
```
x.head()
```

Out[38]:

	Avg _Min_Sent_txn	Avg _Min_Recv_txn	Time_Diff	Min value received	Max value received	Min value sent	Max value sent	Total transactions
0	21.80	0.00	65.40	101.000000	101.000000	1.000000	84.176518	4
1	1073.76	1476.59	24563.10	0.216334	6.999559	0.000000	7.557834	21
2	13.57	10125.40	638755.18	0.010053	20.000000	0.008583	19.999769	126
3	163765.17	0.00	327530.33	0.002457	0.002457	0.000000	0.000508	3
4	22371.43	29004.65	596613.23	0.005441	1000.000000	0.000000	1000.000000	25

In [8]:
```
# Normalizing using Standard Scaler
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
```

In [40]:
```
scaled_x = scale.fit_transform(x)
```

In [41]:
```
scaled_x
```

Out[41]:
```
array([[-0.23563546, -0.34683151, -0.67581383, ..., -0.02741928,
        -0.03472053, -0.20658626],
       [-0.18667347, -0.28285902, -0.59994049, ..., -0.03463378,
        -0.04627836, -0.19401603],
       [-0.23601852,  0.09184615,  1.30231167, ..., -0.03457186,
        -0.04440151, -0.11637637],
       ...,
       [-0.23665011, -0.34683151, -0.67601638, ..., -0.03463378,
        -0.04741845, -0.20880453],
       [-0.21206718, -0.33911628, -0.6710901 , ..., -0.03067802,
        -0.04733556, -0.20584683],
       [-0.23665011, -0.24418326,  0.22656596, ..., -0.03463378,
        -0.04741845, -0.11785522]])
```

In [42]:
```
# Train and Test splitting

from sklearn.model_selection import train_test_split
```

In [43]:
```
x_train, x_test, y_train, y_test = train_test_split(scaled_x,y, test_size = 0.2,random_state =10)
```

In [44]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [45]:
```
model_DT = DecisionTreeClassifier()
```

In [47]:
```
# Fit the data into the model

model_DT.fit(x_train,y_train)
y_predict = model_DT.predict(x_test)
```

In [52]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

Out[52]:
```
0.9171747967479674
```

col_0	0	1
FLAG		
0	1476	77
1	86	329

Now this model is wihout applying SMOTE or we can say witot balancing the data. So will use SMOTE to create data identical, so that model can't be biased one.

It will oversample the fraud values & give us accurate result without getting biased.

In [54]:
```
# Introduce SMOTE

!pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\admin\anaconda3\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\admin\anaconda3\lib\site-packages (from imblearn) (0.9.1)
Requirement already satisfied: numpy>=1.17.3 in c:\users\admin\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (1.20.3)
Requirement already satisfied: scikit-learn>=1.1.0 in c:\users\admin\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (1.1.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\admin\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (2.2.0)
Requirement already satisfied: scipy>=1.3.2 in c:\users\admin\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (1.7.1)
Requirement already satisfied: joblib>=1.0.0 in c:\users\admin\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (1.1.0)
```

In [55]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
```

In [23]:
```
x_train_SMOTE, y_train_SMOTE = smote.fit_resample(x_train.astype('float'),y_train)
```

In [59]:
```
# COmparison with counter
from collections import Counter
```

In [60]:
```
print('Before SMOTE:',Counter(y_train))
print('After SMOTE:', Counter(y_train_SMOTE))
```

```
Before SMOTE: Counter({0: 6108, 1: 1764})
After SMOTE: Counter({0: 6108, 1: 6108})
```

In [61]:
```
model_DT.fit(x_train_SMOTE,y_train_SMOTE)
y_predict = model_DT.predict(x_test)
```

In [62]:
```
print(accuracy_score(y_test,y_predict))
pd.crosstab(y_test,y_predict)
```

Out[62]:
```
0.9019308943089431
```

col_0	0	1
FLAG		
0	1422	131
1	62	353

So here we can see whatever is the accuracy it will no be biased one.

In [ ]:
```

```