Credit Card Fraud Detection

(1) Import Python Libraries

In [1]:
```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

(2) Loading the Data Set

In [2]:
```
path = r'D:\IITG\portfolio_finance\credit_fraud\creditcard.csv'
```

In [3]:
```
credit_card_data = pd.read_csv(path)
credit_card_data.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

In [4]:
```
credit_card_data.tail()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.30
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.29
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.70
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.67
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.41

5 rows × 31 columns

(3) Exploring the Data Set

In [5]:
```
credit_card_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [6]:
```
credit_card_data.isnull().sum()
```

Out[6]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [7]:
```
# distribution of legit transactions & fraudulent transactions
credit_card_data['Class'].value_counts()
```

Out[7]:
```
0    284315
1       492
Name: Class, dtype: int64
```

In [9]:
```
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]

print(legit.shape)
print(fraud.shape)
```

```
(284315, 31)
(492, 31)
```

In [10]:
```
legit.Amount.describe()
```

Out[10]:
```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

In [11]:
```
fraud.Amount.describe()
```

Out[11]:
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

In [12]:
```
credit_card_data.groupby('Class').mean()
```

Out[12]:

	Time	V1	V2	V3	V4	V5	V6	V7
Class								
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731

2 rows × 30 columns

(4) Undersampling to Balance Uneven Dataset

Building a sample dataset containing similar distribution of normal transactions and Fraudulent Transactions.

Number of Fraudulent Transactions = 492

In [13]:
```
legit_sample = legit.sample(n = 492)
```

In [14]:
```
new_dataset = pd.concat([legit_sample, fraud], axis=0)
new_dataset.head()
```

Out[14]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
195339	131009.0	1.823190	0.082713	0.000767	3.767025	-0.103623	0.763705	-0.601138	0.249
243533	151968.0	2.052367	0.314290	-1.670219	0.513186	0.315332	-1.407964	0.354787	-0.451
10153	15554.0	-0.079505	0.922168	2.103711	1.884313	-0.418929	0.143780	0.122155	0.005
32846	36989.0	1.332849	0.389198	-2.165597	-0.306873	2.641351	2.808084	-0.171627	0.683
150547	93636.0	-1.545224	-0.794062	1.909521	-2.256057	-1.701522	0.799766	1.630826	-0.269

5 rows × 31 columns

In [15]:
```
new_dataset.tail()
```

Out[15]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.6972
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.2485
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.2101
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.0587
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.0683

5 rows × 31 columns

In [16]:
```
new_dataset['Class'].value_counts()
```

Out[16]:
```
0    492
1    492
Name: Class, dtype: int64
```

In [17]:
```
new_dataset.groupby('Class').mean()
```

Out[17]:

	Time	V1	V2	V3	V4	V5	V6	V7
Class								
0	92909.794715	0.095856	-0.043130	-0.036796	0.078152	0.101458	-0.000708	-0.029993
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731

2 rows × 30 columns

(5) Building the Model

Split the Data

In [18]:
```
X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']
```

In [19]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, str
```

Create and Fit the Model

In [22]:
```
model = LogisticRegression()
model.fit(X_train, Y_train)
```

Out[22]:
```
LogisticRegression()
```

(6) Evaluating the Model

In [23]:
```
X_train_prediction = model.predict(X_train)
X_train_prediction[:5]
```

Out[23]:
```
array([0, 0, 1, 0, 1], dtype=int64)
```

In [24]:
```
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.9364675984752223
```

In [25]:
```
X_test_prediction = model.predict(X_test)
X_test_prediction[:5]
```

Out[25]:
```
array([1, 0, 0, 0, 1], dtype=int64)
```

In [26]:
```
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.9441624365482234
```