Credit Card Fraud Detection - Logistic Regression

Importing the Dependencies

In [3]:

```
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
```

In [4]:

```
# loading the dataset to a Pandas DataFrame
credit_card_data = pd.read_csv('creditcard.csv')
```

In [5]:

```
# first 5 rows of the dataset
credit_card_data.head()
```

Out[5]:

	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	0.09869
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08510
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24767
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37743
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27053

5 rows × 31 columns

In [6]:

```
credit_card_data.tail()
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.91821
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.02433
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.29682
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.68618
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.57700

5 rows × 31 columns

In [7]:

```python
# dataset informations
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 [8]:

```
# checking the number of missing values in each column
credit_card_data.isnull().sum()
```

Out[8]:

```
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 [9]:

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

Out[9]:

```
0    284315
1       492
Name: Class, dtype: int64
```

This Dataset is highly unblanced

0 --> Normal Transaction

1 --> fraudulent transaction

In [10]:

```
# separating the data for analysis
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

In [11]:

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

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

In [12]:

```
# statistical measures of the data
legit.Amount.describe()
```

Out[12]:

```
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 [13]:

```
fraud.Amount.describe()
```

Out[13]:

```
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 [14]:

```
# compare the values for both transactions
credit_card_data.groupby('Class').mean()
```

Out[14]:

	Time	V1	V2	V3	V4	V5	V6	
Class								
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.0096
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.5687

2 rows × 30 columns

Under-Sampling

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

Number of Fraudulent Transactions --> 492

In [15]:

```
legit_sample = legit.sample(n=492)
```

Concatenating two DataFrames

In [16]:

```
new_dataset = pd.concat([legit_sample, fraud], axis=0)
```

In [17]:

```
new_dataset.head()
```

Out[17]:

	Time	V1	V2	V3	V4	V5	V6	V7
123161	76832.0	-1.087438	2.017740	-0.567000	0.713958	-0.023615	-0.810597	0.479397
166343	118015.0	0.050322	0.402958	0.789783	-0.627845	0.051557	0.049537	0.216446
72886	54897.0	1.138338	0.097058	0.511077	1.488158	-0.455488	-0.412689	-0.031305
169988	119962.0	1.989623	-0.340611	-0.275218	0.569683	-0.780929	-0.742158	-0.552439
28652	35098.0	1.295819	0.356457	0.085803	0.564738	-0.092013	-0.761407	0.087178

5 rows × 31 columns

In [18]:

```
new_dataset.tail()
```

Out[18]:

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

5 rows × 31 columns

In [19]:

```
new_dataset['Class'].value_counts()
```

Out[19]:

```
0    492
1    492
Name: Class, dtype: int64
```

In [20]:

```
new_dataset.groupby('Class').mean()
```

Out[20]:

	Time	V1	V2	V3	V4	V5	V6	
Class								
0	95031.329268	-0.040423	-0.091994	-0.045086	-0.059471	-0.022841	0.099992	0.0363
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.5687

2 rows × 30 columns

Splitting the data into Features & Targets

In [21]:

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

In [22]:

```
print(X)
```

```
            Time        V1        V2        V3        V4        V5
V6  \
123161   76832.0 -1.087438  2.017740 -0.567000  0.713958 -0.023615 -0.810
597
166343  118015.0  0.050322  0.402958  0.789783 -0.627845  0.051557  0.049
537
72886    54897.0  1.138338  0.097058  0.511077  1.488158 -0.455488 -0.412
689
169988  119962.0  1.989623 -0.340611 -0.275218  0.569683 -0.780929 -0.742
158
28652    35098.0  1.295819  0.356457  0.085803  0.564738 -0.092013 -0.761
407
...          ...       ...       ...       ...       ...       ...
...
279863  169142.0 -1.927883  1.125653 -4.518331  1.749293 -1.566487 -2.010
494
280143  169347.0  1.378559  1.289381 -5.004247  1.411850  0.442581 -1.326
536
280149  169351.0 -0.676143  1.126366 -2.213700  0.468308 -1.120541 -0.003
346
281144  169966.0 -3.113832  0.585864 -5.399730  1.817092 -0.840618 -2.943
548
281674  170348.0  1.991976  0.158476 -2.583441  0.408670  1.151147 -0.096
695

              V7        V8        V9  ...       V20       V21       V22
\
123161  0.479397  0.385982  0.178915  ...  0.369416  0.076340  0.560859
166343  0.216446  0.040670  0.492662  ... -0.114646  0.325914  1.031268
72886  -0.031305  0.053101  0.511696  ... -0.302089 -0.098039 -0.127215
169988 -0.552439 -0.098062  1.043559  ... -0.196929  0.191288  0.688134
28652   0.087178 -0.180047  0.040541  ... -0.052741 -0.321231 -0.912854
...          ...       ...       ...  ...       ...       ...       ...
279863 -0.882850  0.697211 -2.064945  ...  1.252967  0.778584 -0.319189
280143 -1.413170  0.248525 -1.127396  ...  0.226138  0.370612  0.028234
280149 -2.234739  1.210158 -0.652250  ...  0.247968  0.751826  0.834108
281144 -2.208002  1.058733 -1.632333  ...  0.306271  0.583276 -0.269209
281674  0.223050 -0.068384  0.577829  ... -0.017652 -0.164350 -0.295135

             V23       V24       V25       V26       V27       V28  Amoun
t
123161  0.141607  0.052542 -0.583965 -0.362817  0.751282  0.396964    2.3
0
166343 -0.081635  0.677562 -0.400256  0.531929 -0.108827  0.006508   27.0
0
72886  -0.003904  0.364371  0.564557 -0.327169  0.029701  0.015961    9.4
6
169988  0.177762  0.081085 -0.292403  0.353617 -0.014223 -0.043190   11.0
0
28652   0.061328 -0.187906  0.294568  0.127188 -0.022380  0.025644    0.9
9
...          ...       ...       ...       ...       ...       ...
...
279863  0.639419 -0.294885  0.537503  0.788395  0.292680  0.147968  390.0
0
280143 -0.145640 -0.081049  0.521875  0.739467  0.389152  0.186637    0.7
6
280149  0.190944  0.032070 -0.739695  0.471111  0.385107  0.194361   77.8
9
281144 -0.456108 -0.183659 -0.328168  0.606116  0.884876 -0.253700  245.0
0
```

```
281674 -0.072173 -0.450261  0.313267 -0.289617  0.002988 -0.015309   42.5
3

[984 rows x 30 columns]
```

In [23]:

```python
print(Y)
```

```
123161    0
166343    0
72886     0
169988    0
28652     0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

Split the data into Training data & Testing Data

In [24]:

```python
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, rar
```

In [25]:

```python
print(X.shape, X_train.shape, X_test.shape)
```

```
(984, 30) (787, 30) (197, 30)
```

Model Training

Logistic Regression

In [26]:

```python
model = LogisticRegression()
```

In [27]:

```python
# training the Logistic Regression Model with Training Data
model.fit(X_train, Y_train)
```

Out[27]:

```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Model Evaluation

Accuracy Score

In [28]:

```
# accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

In [29]:

```
print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.9351969504447268
```

In [30]:

```
# accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

In [31]:

```
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.9187817258883249
```