Credit Card Fraud Detection

Importing the Dependencies

In [1]:
```python
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 [2]:
```python
credit_card_data = pd.read_csv('creditcard.csv')
```

In [4]:
```python
credit_card_data.head()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Am
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	14
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	37
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	12
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	6

5 rows × 31 columns

In [5]:
```python
credit_card_data.tail()
```

Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.82
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.05
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.02
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.10
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.01

5 rows × 31 columns

In [6]:
```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 [7]:
```python
# checking the number of missing values in each column
credit_card_data.isnull().sum()
```

Out[7]:
```
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 [8]:
```python
# distribution of legit transactions & fraudulent transactions
credit_card_data['Class'].value_counts()
```

Out[8]:
```
0    284315
1       492
Name: Class, dtype: int64
```

This Dataset is highly unblanced

0 --> Normal Transaction

1 --> fraudulent transaction

In [9]:
```python
# separating the data for analysis
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

In [10]:
```python
print(legit.shape)
print(fraud.shape)
```

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

In [11]:
```python
# statistical measures of the data
legit.Amount.describe()
```

Out[11]:
```
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 [12]:
```python
fraud.Amount.describe()
```

Out[12]:
```
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 [13]:
```python
# compare the values for both transactions
credit_card_data.groupby('Class').mean()
```

Out[13]:

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	...	-0.000644	-0.001235	-0.000024	0.000070	0.000182	-0.000072	-0.000089	-(
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588	0.014049	-0.040308	-0.105130	0.041449	0.051648	(

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 [14]:
```python
legit_sample = legit.sample(n=492)
```

Concatenating two DataFrames

In [15]:
```python
new_dataset = pd.concat([legit_sample, fraud], axis=0)
```

In [16]:
```python
new_dataset.head()
```

Out[16]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V
65710	51711.0	1.092926	-0.210745	0.300661	0.206798	0.111370	0.896631	-0.361242	0.211620	0.001454	...	-0.036764	-0.198099	-0.139403	-1.295300	0.256514	0.298118	-0.000718	0.014
27348	34512.0	1.291999	-0.305899	0.592046	0.076172	-0.408032	0.480820	-0.598464	0.092763	1.077848	...	-0.309492	-0.593790	-0.109808	-0.906162	0.358527	0.992938	-0.028041	0.004
87264	61606.0	0.887448	-0.373792	2.307083	3.178615	-1.717495	0.527985	-1.108241	0.396379	1.033958	...	0.156364	0.671896	-0.086938	0.949620	0.318508	0.191768	0.070428	0.055
268854	163411.0	1.819994	-1.364167	-1.733458	-0.853660	-0.464153	-0.772680	-0.073094	-0.407604	-0.628067	...	0.570276	1.166146	-0.206510	0.668933	0.179489	0.025896	-0.077834	-0.019
29257	35381.0	-2.996508	1.860704	1.115144	-1.989952	0.207069	-0.394921	1.014649	-0.440142	2.045358	...	-0.593678	0.063342	0.167963	0.037198	0.000962	0.646262	0.319762	-0.574

5 rows × 31 columns

In [17]:
```python
new_dataset.tail()
```

Out[17]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	...	0.778584	-0.319189	0.639419	-0.294885	0.537503	0.788395	0.292680	0.147
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	...	0.370612	0.028234	-0.145640	-0.081049	0.521875	0.739467	0.389152	0.186
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...	0.751826	0.834108	0.190944	0.032070	-0.739695	0.471111	0.385107	0.194
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	...	0.583276	-0.269209	-0.456108	-0.183659	-0.328168	0.606116	0.884876	-0.253
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	...	-0.164350	-0.295135	-0.072173	-0.450261	0.313267	-0.289617	0.002988	-0.015

5 rows × 31 columns

In [18]:
```python
new_dataset['Class'].value_counts()
```

Out[18]:
```
0    492
1    492
Name: Class, dtype: int64
```

In [19]:
```python
new_dataset.groupby('Class').mean()
```

Out[19]:

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	
0	95731.180894	0.002404	0.037731	-0.049418	0.020513	0.081113	0.059102	0.093509	-0.034330	-0.064653	...	-0.023246	-0.023964	-0.032276	0.029787	-0.015999	0.006913	0.002329	-0.0
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588	0.014049	-0.040308	-0.105130	0.041449	0.051648	0.1

2 rows × 30 columns

Splitting the data into Features & Targets

In [20]:
```python
X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']
```

In [21]:
```python
X
```

Out[21]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	
65710	51711.0	1.092926	-0.210745	0.300661	0.206798	0.111370	0.896631	-0.361242	0.211620	0.001454	...	0.170307	-0.036764	-0.198099	-0.139403	-1.295300	0.256514	0.298118	-0.00
27348	34512.0	1.291999	-0.305899	0.592046	0.076172	-0.408032	0.480820	-0.598464	0.092763	1.077848	...	0.041490	-0.309492	-0.593790	-0.109808	-0.906162	0.358527	0.992938	-0.02
87264	61606.0	0.887448	-0.373792	2.307083	3.178615	-1.717495	0.527985	-1.108241	0.396379	1.033958	...	-0.094336	0.156364	0.671896	-0.086938	0.949620	0.318508	0.191768	0.07
268854	163411.0	1.819994	-1.364167	-1.733458	-0.853660	-0.464153	-0.772680	-0.073094	-0.407604	-0.628067	...	0.501482	0.570276	1.166146	-0.206510	0.668933	0.179489	0.025896	-0.07
29257	35381.0	-2.996508	1.860704	1.115144	-1.989952	0.207069	-0.394921	1.014649	-0.440142	2.045358	...	1.070273	-0.593678	0.063342	0.167963	0.037198	0.000962	0.646262	0.31
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	...	1.252967	0.778584	-0.319189	0.639419	-0.294885	0.537503	0.788395	0.29
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	...	0.226138	0.370612	0.028234	-0.145640	-0.081049	0.521875	0.739467	0.38
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...	0.247968	0.751826	0.834108	0.190944	0.032070	-0.739695	0.471111	0.38
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	...	0.306271	0.583276	-0.269209	-0.456108	-0.183659	-0.328168	0.606116	0.88
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	...	-0.017652	-0.164350	-0.295135	-0.072173	-0.450261	0.313267	-0.289617	0.00

984 rows × 30 columns

In [22]:
```python
Y
```

Out[22]:
```
65710     0
27348     0
87264     0
268854    0
29257     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, random_state=2)
```

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()
```

Model Evaluation

Accuracy Score

In [28]:
```python
# accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

In [29]:
```python
print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.9224904701397713
```

In [30]:
```python
# accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

In [31]:
```python
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.9137055837563451
```

END

In [ ]: