Credit Card Fraud Detection Project

Importing the Dependencies

In [62]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')
```

Loading the datasets to a pandas dataframe

In [2]:
```
credit_card_data= pd.read_csv(r"C:\Users\hp\Downloads\creditcard.csv")
```

In [3]:
```
# First 5 rows from the datasets
credit_card_data.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22
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
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
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
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
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

5 rows × 31 columns

In [4]:
```
# Last 5 rows from the datasets
credit_card_data.tail()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057

5 rows × 31 columns

In [5]:
```
# Dataset information
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 [78]:

```python
# Descriptive Statistics of the Dataset
credit_card_data.describe()
```

Out[78]:

	Time	V1	V2	V3	V4	V5	V6	V7	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2

8 rows × 31 columns

In [6]:

```python
# Checking the missing values in each columns
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 [8]:
```
# Distribution of legit transaction and fraudulant transaction
credit_card_data['Class'].value_counts()
```

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

This dataset is highly unbalanced

0 --> Normal transaction

1 --> Fraudulant 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 [15]:
```
# Statistical measures of the data
legit.Amount.describe()
```

Out[15]:
```
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 [21]:
```
# Compare the value for both transactions
credit_card_data.groupby('Class').mean()
```

Out[21]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20
Class												
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	...	-0.000644
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319

2 rows × 30 columns

Under-sampling

Build a sample dataset containing similar distribution of normal transaction and fraudulant transaction

Number of Fraudulant transaction --> 492

In [22]:
```
legit_sample = legit.sample(n=492)
```

Concatenating two Data Frame

In [24]:
```
new_dataset=pd.concat([legit_sample, fraud], axis=0)
```

In [25]:
```
new_dataset.head()
```

Out[25]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	
166220	117935.0	-0.590713	1.017427	-0.499623	-0.498109	0.982834	-0.511261	0.942722	-0.452153	0.540771	...	0.081814	0.
56895	47657.0	-0.524949	-3.099488	-1.625054	0.636279	-0.466524	0.517939	1.396547	-0.325619	0.174358	...	0.440857	-0.
234311	147918.0	-1.291577	0.041407	0.864071	-1.643596	-0.927824	-0.651869	0.407948	0.293575	0.539491	...	0.303216	0.
139817	83374.0	-0.322899	1.215585	0.741049	0.881526	-0.167682	-0.505084	0.309599	0.370825	-1.053370	...	0.234576	0.
147568	88691.0	-1.109847	0.986439	0.304282	-1.368567	1.868140	1.294716	1.024818	0.465826	-0.626328	...	0.344322	1.

5 rows × 31 columns

In [26]:
```
new_dataset.tail()
```

Out[26]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	...	0.778584	-0.
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	...	0.370612	0.
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...	0.751826	0.
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	...	0.583276	-0.
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	...	-0.164350	-0.

5 rows × 31 columns

In [27]:
```
new_dataset['Class'].value_counts()
```

Out[27]:
```
Class
0    492
1    492
Name: count, dtype: int64
```

In [28]:
```
new_dataset.groupby('Class').mean()
```

Out[28]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20
Class												
0	91563.686992	0.000617	-0.011978	-0.043459	-0.042650	-0.079911	-0.023273	-0.071837	0.077173	-0.035296	...	-0.006109
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319

2 rows × 30 columns

Splitting the data into features and target

In [31]:
```
x=new_dataset.drop(columns='Class',axis=1)
y=new_dataset['Class']
```

In [34]:
```
print(x)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
166220  117935.0 -0.590713  1.017427 -0.499623 -0.498109  0.982834 -0.511261
56895    47657.0 -0.524949 -3.099488 -1.625054  0.636279 -0.466524  0.517939
234311  147918.0 -1.291577  0.041407  0.864071 -1.643596 -0.927824 -0.651869
139817   83374.0 -0.322899  1.215585  0.741049  0.881526 -0.167682 -0.505084
147568   88691.0 -1.109847  0.986439  0.304282 -1.368567  1.868140  1.294716
...          ...       ...       ...       ...       ...       ...       ...
279863  169142.0 -1.927883  1.125653 -4.518331  1.749293 -1.566487 -2.010494
280143  169347.0  1.378559  1.289381 -5.004247  1.411850  0.442581 -1.326536
280149  169351.0 -0.676143  1.126366 -2.213700  0.468308 -1.120541 -0.003346
281144  169966.0 -3.113832  0.585864 -5.399730  1.817092 -0.840618 -2.943548
281674  170348.0  1.991976  0.158476 -2.583441  0.408670  1.151147 -0.096695

              V7        V8        V9  ...       V20       V21       V22  \
166220  0.942722 -0.452153  0.540771  ... -0.254721  0.081814  0.394565
56895   1.396547 -0.325619  0.174358  ...  1.920297  0.440857 -0.737504
234311  0.407948  0.293575  0.539491  ... -0.144028  0.303216  0.732074
139817  0.309599  0.370825 -1.053370  ... -0.080045  0.234576  0.583835
147568  1.024818  0.465826 -0.626328  ... -0.270733  0.344322  1.059389
...          ...       ...       ...  ...       ...       ...       ...
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  Amount
166220 -0.225234 -0.582452 -0.219667  0.475617 -0.931328 -0.173388    6.99
56895  -1.090459 -1.165015  0.314591  1.101588 -0.261089  0.145519  977.28
234311 -0.046195  0.054990 -0.182179  0.701335 -0.127487  0.017725  150.00
139817 -0.067777  0.356406 -0.141238 -0.347433 -0.044694 -0.026809   12.90
147568 -0.563945 -1.840395  0.816531  0.060419 -0.052302  0.033631   32.90
...          ...       ...       ...       ...       ...       ...     ...
279863  0.639419 -0.294885  0.537503  0.788395  0.292680  0.147968  390.00
280143 -0.145640 -0.081049  0.521875  0.739467  0.389152  0.186637    0.76
280149  0.190944  0.032070 -0.739695  0.471111  0.385107  0.194361   77.89
281144 -0.456108 -0.183659 -0.328168  0.606116  0.884876 -0.253700  245.00
281674 -0.072173 -0.450261  0.313267 -0.289617  0.002988 -0.015309   42.53

[984 rows x 30 columns]
```

In [35]:
```
print(y)
```

```
166220    0
56895     0
234311    0
139817    0
147568    0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

Split the data into training data and testing data

In [37]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2, stratify=y, random_state=2)
```

In [39]:
```
print(x.shape,x_train.shape, x_test.shape)
```

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

Model Training

In [40]:
```
model= LogisticRegression()
```

Training the logistic regression model with training data

In [63]:
```
model.fit(x_train,y_train)
```

Out[63]:

▾ LogisticRegression

```
LogisticRegression()
```

Model Evaluation

Accuracy Score

In [64]:

```
# Acuuracy on training data
x_train_prediction=model.predict(x_train)
training_data_accuracy=accuracy_score(x_train_prediction, y_train)
```

In [73]:

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

```
Accuracy on training data: 0.9555273189326556
```

In [66]:

```
# Acuuracy on test data
x_test_prediction=model.predict(x_test)
testing_data_accuracy=accuracy_score(x_test_prediction, y_test)
```

In [74]:

```
print('Accuracy on testing data:',testing_data_accuracy)
```

```
Accuracy on testing data: 0.934010152284264
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js