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
# loading the dataset to a Pandas DataFrame
df = pd.read_csv(r"C:\Users\Ajith\Desktop\Project dataset\Python\creditcard.csv")
```

In [3]:

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

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
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	149.62	0
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	2.69	0
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	378.66	0
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	123.50	0
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	69.99	0

5 rows × 31 columns

In [4]:

```python
df.tail()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
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	0.823731	0.77	0
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	-0.053527	24.79	0
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	-0.026561	67.88	0
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	0.104533	10.00	0
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	0.013649	217.00	0

5 rows × 31 columns

In [5]:

```python
# dataset informations
df.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]:

```python
# checking the number of missing values in each column
df.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]:

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

Out[7]:

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

This Dataset is highly unblanced

0 --> Normal Transaction

1 --> fraudulent transaction

In [8]:

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

In [9]:

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

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

In [10]:

```python
# statistical measures of the data
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]:

```python
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]:

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

Out[12]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount
Class																					
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	-0.000295	-0.000131	88.291022
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	0.170575	0.075667	122.211321

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 [13]:

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

Concatenating two DataFrames

In [14]:

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

In [15]:

```python
new_dataset.head()
```

Out[15]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
263455	160963.0	-1.653601	1.298182	-0.406764	0.416766	1.074881	-0.876537	2.162298	-1.263610	1.082443	...	-0.132844	0.696389	-0.601603	0.001147	-0.370812	-0.749309	-1.142959	-0.717143	66.22	0
226108	144497.0	-7.488511	6.859970	-4.500524	-2.971204	-0.690314	-1.858815	0.903615	0.525302	5.595473	...	-0.707976	0.223671	0.137250	-0.467387	1.206142	0.126149	2.581028	1.041168	0.85	0
167045	118452.0	-0.059486	-0.749901	0.102727	-2.153158	0.491632	-0.258358	-0.385521	-0.077316	-2.900464	...	0.255930	1.048712	-0.149389	0.272841	-0.313043	0.144482	0.178788	0.185649	15.00	0
189503	128430.0	-0.397391	-0.818624	0.133938	-1.865391	-2.434405	1.180764	2.436894	-0.666855	-0.674239	...	0.473834	1.597321	0.364247	0.730550	-0.887322	-0.298353	0.177353	-0.097310	590.50	0
259282	159045.0	1.978972	-0.122719	-0.818335	0.437949	-0.405757	-1.223695	0.059958	-0.195546	0.319209	...	-0.206158	-0.524010	0.403332	0.535035	-0.420795	0.145174	-0.077130	-0.065443	1.98	0

5 rows × 31 columns

In [16]:

```python
new_dataset.tail()
```

Out[16]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
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	0.147968	390.00	1
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	0.186637	0.76	1
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	0.194361	77.89	1
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	-0.253700	245.00	1
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	-0.015309	42.53	1

5 rows × 31 columns

In [17]:

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

Out[17]:

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

In [18]:

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

Out[18]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount
Class																					
0	95391.481707	-0.037333	0.013865	0.031194	-0.056112	-0.104775	0.080183	0.047697	-0.037166	0.058083	...	0.000567	0.010975	0.057971	0.037114	-0.021311	0.001073	-0.018945	-0.016009	0.011314	93.008476
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	0.170575	0.075667	122.211321

2 rows × 30 columns

Splitting the data into Features & Targets

In [19]:

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

In [20]:

```python
print(X)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
263455  160963.0 -1.653601  1.298182 -0.406764  0.416766  1.074881 -0.876537
226108  144497.0 -7.488511  6.859970 -4.500524 -2.971204 -0.690314 -1.858815
167045  118452.0 -0.059486 -0.749901  0.102727 -2.153158  0.491632 -0.258358
189503  128430.0 -0.397391 -0.818624  0.133938 -1.865391 -2.434405  1.180764
259282  159045.0  1.978972 -0.122719 -0.818335  0.437949 -0.405757 -1.223695
...          ...       ...       ...       ...       ...       ...       ...
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  \
263455  2.162298 -1.263610  1.082443  ...  0.010684 -0.132844  0.696389
226108  0.903615  0.525302  5.595473  ...  3.872881 -0.707976  0.223671
167045 -0.385521 -0.077316 -2.900464  ...  0.011254  0.255930  1.048712
189503  2.436894 -0.666855 -0.674239  ...  0.266129  0.473834  1.597321
259282  0.059958 -0.195546  0.319209  ... -0.263392 -0.206158 -0.524010
...          ...       ...       ...  ...       ...       ...       ...
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
263455 -0.601603  0.001147 -0.370812 -0.749309 -1.142959 -0.717143   66.22
226108  0.137250 -0.467387  1.206142  0.126149  2.581028  1.041168    0.85
167045 -0.149389  0.272841 -0.313043  0.144482  0.178788  0.185649   15.00
189503  0.364247  0.730550 -0.887322 -0.298353  0.177353 -0.097310  590.50
259282  0.403332  0.535035 -0.420795  0.145174 -0.077130 -0.065443    1.98
...          ...       ...       ...       ...       ...       ...     ...
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 [21]:

```python
print(Y)
```

```
263455    0
226108    0
167045    0
189503    0
259282    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 [22]:

```python
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)
```

In [23]:

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

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

Model Training

Logistic Regression

In [24]:

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

In [25]:

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

```
C:\Users\Ajith\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[25]:

▾ LogisticRegression

LogisticRegression()

Model Evaluation

Accuracy Score

In [26]:

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

In [27]:

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

```
Accuracy on Training data :  0.9504447268106735
```

In [28]:

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

In [29]:

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

```
Accuracy score on Test Data :  0.934010152284264
```

In [ ]: