In [3]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [5]:
```python
# Loading the dataset to a pandas Dataframe

credit_card_data = pd.read_csv("creditcard.csv")
```

In [6]:
```python
# Print first 5 rows of the dataset

credit_card_data.head()
```

Out[6]:

	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 [7]:
```python
#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 [8]:
```python
#checking the no 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]:
```python
# check the distribution legit transection & fraudulent tracsaction

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

Out[9]:
```
Class
0    284315
1       492
Name: count, dtype: int64
```

This dataset is highly unblanced

0--> Normal Transaction

1--> Fraudulant Transaction

In [10]:
```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 [11]:
```python
print(legit.shape)
print(fraud.shape)
```

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

In [12]:
```python
#Statistical measure of the dataa

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 [14]:
```python
#statistical measure of the fraudulent data

fraud.Amount.describe()
```

Out[14]:
```
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 [15]:
```python
# compare the values for both transections

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

Out[15]:

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amoun
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	88.29102
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	122.21132

2 rows × 30 columns

Under Sampling

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

Number of fraudulant transactio --> 492

In [19]:
```python
legit_sample = legit.sample(n=492)
```

In [20]:
```python
#Concatenating two dataframes

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

In [22]:
```python
new_dataset.head()
```

Out[22]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
50958	44723.0	-0.149770	-0.484074	1.780925	-2.175589	-1.229287	-0.194473	-0.628454	-0.018248	-1.825910	...	-0.065327	0.347721	-0.177822	-0.044061	-0.099446	-0.157829	0.014495	-0.151310	12.00	0
106540	70004.0	-1.113875	0.805525	0.991393	-1.237974	-0.677030	-0.954057	0.696629	0.441609	-0.554530	...	-0.025454	-0.400844	0.244328	0.527070	-0.344054	0.627906	0.079758	0.107471	96.40	0
133620	80465.0	-4.566588	2.265938	-2.081287	0.022018	-0.808003	-1.983682	-0.747484	1.968856	-0.584346	...	0.149212	-0.087656	-0.476314	0.288504	0.220049	-0.772300	-0.803539	-0.318038	1.00	0
111845	72393.0	-0.557169	0.712440	-0.601470	-1.652010	2.618591	3.237918	0.618573	0.487756	-0.014197	...	-0.024861	-0.030212	-0.212911	1.027933	-0.060787	0.136017	-0.259648	-0.133428	50.00	0
171830	120839.0	1.968808	0.567574	-2.150139	1.479389	0.847584	-0.895213	0.380771	-0.183443	0.244382	...	-0.361074	-1.032240	0.279658	0.321827	-0.094958	-0.944798	0.027039	0.016592	24.49	0

5 rows × 31 columns

In [23]:
```python
new_dataset.tail()
```

Out[23]:

	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 [24]:
```python
new_dataset["Class"].value_counts()
```

Out[24]:
```
Class
0    492
1    492
Name: count, dtype: int64
```

In [25]:
```python
#check the mean value of new dataset

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

Out[25]:

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount
0	94303.760163	-0.026661	0.050488	0.144314	-0.038754	-0.014703	0.018385	-0.028544	-0.002668	-0.060944	...	0.008597	0.007827	-0.006465	-0.031066	0.018525	0.024956	-0.044479	-0.017016	-0.011264	83.847398
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

Spliting the data info Features & Targets

In [29]:
```python
X=new_dataset.drop(columns='Class',axis=1)
Y= new_dataset['Class']
```

In [30]:
```python
print(X)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
50958    44723.0 -0.149770 -0.484074  1.780925 -2.175589 -1.229287 -0.194473   
106540   70004.0 -1.113875  0.805525  0.991393 -1.237974 -0.677030 -0.954057   
133620   80465.0 -4.566588  2.265938 -2.081287  0.022018 -0.808003 -1.983682   
111845   72393.0 -0.557169  0.712440 -0.601470 -1.652010  2.618591  3.237918   
171830  120839.0  1.968808  0.567574 -2.150139  1.479389  0.847584 -0.895213   
...          ...       ...       ...       ...       ...       ...       ...   
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  \
50958  -0.628454 -0.018248 -1.825910  ... -0.162027 -0.065327  0.347721   
106540  0.696629  0.441609 -0.554530  ...  0.063886 -0.025454 -0.400844   
133620 -0.747484  1.968856 -0.584346  ... -0.601376  0.149212 -0.087656   
111845  0.618573  0.487756 -0.014197  ...  0.041162 -0.024861 -0.030212   
171830  0.380771 -0.183443  0.244382  ... -0.165824 -0.361074 -1.032240   
...          ...       ...       ...  ...       ...       ...       ...   
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  
50958  -0.177822 -0.044061 -0.099446 -0.157829  0.014495 -0.151310   12.00  
106540  0.244328  0.527070 -0.344054  0.627906  0.079758  0.107471   96.40  
133620 -0.476314  0.288504  0.220049 -0.772300 -0.803539 -0.318038    1.00  
111845 -0.212911  1.027933 -0.060787  0.136017 -0.259648 -0.133428   50.00  
171830  0.279658  0.321827 -0.094958 -0.944798  0.027039  0.016592   24.49  
...          ...       ...       ...       ...       ...       ...     ...  
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 [31]:
```python
print(Y)
```

```
50958     0
106540    0
133620    0
111845    0
171830    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 [32]:
```python
X_train, X_test, Y_train, Y_test =  train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)
```

In [33]:
```python
print(X.shape, X_train.shape, X_test.shape)
```

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

Model Training

Logistic Regression

In [34]:
```python
model = LogisticRegression()
```

In [35]:
```python
# Training the logistic Regression model with Training data

model.fit(X_train, Y_train)
```

Out[35]:
```
▾ LogisticRegression
LogisticRegression()
```

Model Evaluation

Accuracy Score

In [37]:
```python
#accuracy of training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

In [38]:
```python
print('Accuracy on Training Data: ',training_data_accuracy)
```

```
Accuracy on Training Data:  0.9466327827191868
```

In [39]:
```python
#accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

In [41]:
```python
print('Accuracy on Test Data: ', training_data_accuracy)
```

```
Accuracy on Test Data:  0.9466327827191868
```

In [ ]: