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

Out[2]:

	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	0.098698	0.3
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.2
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.5
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.8

5 rows × 31 columns

In [3]:
```
credit_card_data.tail()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.3
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.2
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.7
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.6
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.4

5 rows × 31 columns

In [4]:
```
# 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 [5]:
```
# checking the number of missing values in each column
credit_card_data.isnull().sum()
```

Out[5]:
```
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 [6]:
```
# distribution of legit transactions & fraudulent transactions
credit_card_data['Class'].value_counts()
```

Out[6]:
```
0    284315
1       492
Name: Class, dtype: int64
```

In [7]:
```
# separating the data for analysis
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

In [8]:
```
print(legit.shape)
print(fraud.shape)
```

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

In [9]:
```
# statistical measures of the data
legit.Amount.describe()
```

Out[9]:
```
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 [10]:
```
# compare the values for both transactions
credit_card_data.groupby('Class').mean()
```

Out[10]:

Class	Time	V1	V2	V3	V4	V5	V6	V7	
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0

2 rows × 30 columns

Under-Sampling

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

In [14]:
```
legit_sample = legit.sample(n=480)
```

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

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

Out[16]:

	Time	V1	V2	V3	V4	V5	V6	V7	
60160	49185.0	-1.654341	-0.053119	1.542421	-0.444621	-3.039142	1.434500	0.747724	0.0870
19400	30245.0	1.083586	-0.688701	0.799170	0.412378	-0.903625	0.686599	-0.948736	0.4082
4275	3756.0	1.455736	-0.593967	-0.883533	-1.639203	1.486036	3.264616	-1.207737	0.7176
59294	48780.0	0.827041	-0.451685	1.249578	1.884651	-0.885191	0.482966	-0.293657	0.1702
20162	30831.0	-0.437671	1.048584	1.701315	0.046411	-0.234154	-1.088927	0.705312	-0.0833

5 rows × 31 columns

In [17]:
```
new_dataset['Class'].value_counts()
```

Out[17]:
```
1    492
0    480
Name: Class, dtype: int64
```

In [18]:
```
new_dataset.groupby('Class').mean()
```

Out[18]:

Class	Time	V1	V2	V3	V4	V5	V6	V7	
0	91679.389583	-0.064846	-0.011129	0.050175	0.027084	-0.086043	0.043617	-0.021643	-0.(
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.5

2 rows × 30 columns

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

print(X)
```

```
           Time        V1        V2        V3        V4        V5        V6  \
60160    49185.0 -1.654341 -0.053119  1.542421 -0.444621 -3.039142  1.434500
19400    30245.0  1.083586 -0.688701  0.799170  0.412378 -0.903625  0.686599
4275      3756.0  1.455736 -0.593967 -0.883533 -1.639203  1.486036  3.264616
59294    48780.0  0.827041 -0.451685  1.249578  1.884651 -0.885191  0.482966
20162    30831.0 -0.437671  1.048584  1.701315  0.046411 -0.234154 -1.088927
...          ...       ...       ...       ...       ...       ...       ...
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  \
60160   0.747724  0.087031 -0.184475  ... -0.337265 -0.110846  0.376021
19400  -0.948736  0.408295 -0.853436  ... -0.567866 -0.122934  0.018052
4275   -1.207737  0.717662  0.363424  ...  0.164725 -0.295169 -0.883400
59294  -0.293657  0.170273  1.048281  ...  0.094102 -0.309147 -0.632329
20162   0.705312 -0.083351 -0.407322  ...  0.082403 -0.207132 -0.540157
...          ...       ...       ...  ...       ...       ...       ...
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
60160   -0.453309  0.138548 -0.066428 -0.148535 -0.785276 -0.987261  360.60
19400    0.036336 -0.352484  0.149118 -0.258495  0.081270  0.026250   56.00
4275     0.106058  0.918577  0.367366 -0.505379 -0.014611  0.010539   13.81
59294    0.006434  0.443969  0.375803 -0.546334  0.076379  0.055209  130.00
20162    0.019404  0.690586 -0.239900  0.051199  0.265246  0.124208    2.67
...           ...       ...       ...       ...       ...       ...     ...
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

[972 rows x 30 columns]
```

In [21]:
```
print(Y)
```

```
60160     0
19400     0
4275      0
59294     0
20162     0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 972, dtype: int64
```

Split the data into Training data & Testing Data

In [22]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify
```

In [23]:
```
print(X.shape, X_train.shape, X_test.shape)
```

```
(972, 30) (777, 30) (195, 30)
```

Model Training

Logistic Regression

In [24]:
```
model = LogisticRegression()
```

In [25]:
```
# training the Logistic Regression Model with Training Data
model.fit(X_train, Y_train)
```

Out[25]:
```
LogisticRegression()
```

Model Evaluation

Accuracy Score

In [26]:
```
# accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

In [27]:
```
print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.9407979407979408
```

In [28]:
```
# accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

In [29]:
```
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.9333333333333333
```

In [ ]:
```

```