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')
```

In [3]:
```
# first 5 rows of the dataset
credit_card_data.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.2
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.6
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.7
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.7

5 rows × 31 columns

In [4]:
```
credit_card_data.tail()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.21
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.21
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.23
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.26
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.26

5 rows × 31 columns

In [5]:
```
# 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 [6]:
```
# checking the number of missing values in each column
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 [7]:
```
# distribution of legit transactions & fraudulent transactions
credit_card_data['Class'].value_counts()
```

Out[7]:
```
Class
0    284315
1       492
Name: count, dtype: int64
```

```
This Dataset is highly unblanced

0 --> Normal Transaction

1 --> fraudulent transaction
```

In [8]:
```
# separating the data for analysis
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

In [9]:
```
print(legit.shape)
print(fraud.shape)
```

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

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

Out[12]:

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

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

In [14]:
```
new_dataset = pd.concat([legit_sample, fraud], axis=0)
```

In [15]:
```
new_dataset.head()
```

Out[15]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
2777	2317.0	-0.555676	0.787932	1.028709	-0.336876	0.968243	1.555526	0.223955	0.742251	-0.520564	...
244239	152262.0	-2.932310	2.763792	-2.017536	0.241667	1.293422	0.063131	0.902822	-1.036658	2.639671	...
265596	161954.0	-2.048560	2.977981	-0.789285	4.389680	-0.939254	0.266194	-0.924117	1.887244	-2.116808	...
271112	164417.0	1.759853	-0.962090	-1.951554	0.448426	0.165768	-0.036202	0.185521	-0.183478	-1.154987	...
235049	148224.0	-0.683103	-0.297774	-0.147114	-4.342052	1.116575	0.896010	0.795852	0.244868	-0.546613	...

5 rows × 31 columns

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

Out[16]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	...	0.778
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	...	0.370
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...	0.751
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	...	0.583
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	...	-0.164

5 rows × 31 columns

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

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

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

Out[18]:

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
0	98118.768293	-0.005525	-0.021394	-0.033001	-0.038024	0.125218	0.025210	0.028481	-0.011847	0.090251	...
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...

2 rows × 30 columns

Splitting the data into Features & Targets

In [19]:
```
X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']
```

In [20]:
```
print(X)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
2777      2317.0 -0.555676  0.787932  1.028709 -0.336876  0.968243  1.555526
244239  152262.0 -2.932310  2.763792 -2.017536  0.241667  1.293422  0.063131
265596  161954.0 -2.048560  2.977981 -0.789285  4.389680 -0.939254  0.266194
271112  164417.0  1.759853 -0.962090 -1.951554  0.448426  0.165768 -0.036202
235049  148224.0 -0.683103 -0.297774 -0.147114 -4.342052  1.116575  0.896010
...          ...       ...       ...       ...       ...       ...       ...
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  \
2777    0.223955  0.742251 -0.520564  ... -0.046087 -0.121522 -0.213457
244239  0.902822 -1.036658  2.639671  ...  1.200995 -0.324365 -1.700986
265596 -0.924117  1.887244 -2.116808  ...  0.172994  0.490979  1.315604
271112  0.185521 -0.183478 -1.154987  ... -0.240877 -0.116941 -0.234963
235049  0.795852  0.244868 -0.546613  ... -0.402263 -0.124435  0.156713
...          ...       ...       ...  ...       ...       ...       ...
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
2777    0.080522 -1.425727 -0.488227  0.206451  0.305507  0.068461    7.99
244239  0.140249 -0.617718  0.693403 -0.712085  0.998987  0.461615   17.46
265596 -0.029158  0.091585 -0.327946  0.615306  0.294226  0.182546   10.62
271112 -0.089028  0.179577  0.197563 -0.557929 -0.028528 -0.025278  203.52
235049 -0.418199 -0.960328  1.043281 -0.497336  0.075504  0.026186   64.98
...          ...       ...       ...       ...       ...       ...     ...
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]:
```
print(Y)
```

```
2777      0
244239    0
265596    0
271112    0
235049    0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

In [22]:
```
_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)
```

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

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

Model Training

Logistic Regression

In [26]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=1000)  # You can adjust the value as needed
model.fit(X_train, Y_train)
```

Out[26]:
```
LogisticRegression
LogisticRegression(max_iter=1000)
```

```
Model Evaluation

Accuracy Score
```

In [27]:
```
# accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

In [28]:
```
print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.9466327827191868
```

In [29]:
```
# accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

In [30]:
```
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.9441624365482234
```

In [ ]: