Importing the Dependencies

```
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
```

```
# loading the dataset to a Pandas DataFrame
credit_card_data = pd.read_csv('/content/creditcard.csv')
```

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

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

5 rows × 31 columns

```
credit_card_data.tail()
```

| | 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 |
| **284803** | 172787.0 | -0.732789 | -0.055080 | 2.035030 | -0.738589 | 0.868229 | 1.058415 | 0.024330 |
| **284804** | 172788.0 | 1.919565 | -0.301254 | -3.249640 | -0.557828 | 2.630515 | 3.031260 | -0.296827 |
| **284805** | 172788.0 | -0.240440 | 0.530483 | 0.702510 | 0.689799 | -0.377961 | 0.623708 | -0.686180 |
| **284806** | 172792.0 | -0.533413 | -0.189733 | 0.703337 | -0.506271 | -0.012546 | -0.649617 | 1.577006 |

5 rows × 31 columns

```
# 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
```

```
# checking the number of missing values in each column
credit_card_data.isnull().sum()
```

| | 0 |
|---|---|
| **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

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

| **Class** | **count** |
|---|---|
| **0** | 284315 |
| **1** | 492 |

**dtype:** int64

This Dataset is highly unblanced

0 --> Normal Transaction

1 --> fraudulent transaction

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

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

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

```
# statistical measures of the data
legit.Amount.describe()
```

| | Amount |
|---|---|
| **count** | 284315.000000 |
| **mean** | 88.291022 |
| **std** | 250.105092 |
| **min** | 0.000000 |
| **25%** | 5.650000 |
| **50%** | 22.000000 |
| **75%** | 77.050000 |
| **max** | 25691.160000 |

**dtype:** float64

```
fraud.Amount.describe()
```

| | Amount |
|---|---|
| **count** | 492.000000 |
| **mean** | 122.211321 |
| **std** | 256.683288 |
| **min** | 0.000000 |
| **25%** | 1.000000 |
| **50%** | 9.250000 |
| **75%** | 105.890000 |
| **max** | 2125.870000 |

**dtype:** float64

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

| Class | Time | V1 | V2 | V3 | V4 | V5 | V6 | |
|---|---|---|---|---|---|---|---|---|
| 0 | 94838.202258 | 0.008258 | -0.006271 | 0.012171 | -0.007860 | 0.005453 | 0.002419 | 0.0096 |
| 1 | 80746.806911 | -4.771948 | 3.623778 | -7.033281 | 4.542029 | -3.151225 | -1.397737 | -5.5687 |

2 rows × 30 columns

Under-Sampling

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

Number of Fraudulent Transactions --> 492

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

Concatenating two DataFrames

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

```
new_dataset.head()
```

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | |
|---|---|---|---|---|---|---|---|---|---|
| 126587 | 78035.0 | 1.160426 | -0.502319 | 0.886433 | -0.884251 | -1.031212 | -0.046391 | -0.804541 | |
| 97839 | 66406.0 | -1.909825 | 1.256160 | 0.433707 | -0.810739 | 0.170353 | -0.370122 | 0.435935 | |
| 7407 | 9996.0 | 1.107242 | 0.667646 | 0.314573 | 2.653926 | 0.153394 | -0.585697 | 0.381893 | - |
| 56270 | 47345.0 | -1.219131 | 0.893169 | 1.854603 | 1.652850 | 1.457932 | -0.928450 | 0.826119 | - |
| 131375 | 79577.0 | 1.156505 | -0.773010 | -0.138779 | -1.923648 | -0.614735 | -0.508777 | -0.138414 | - |

5 rows × 31 columns

```
new_dataset.tail()
```

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 |
|---|---|---|---|---|---|---|---|---|
| **279863** | 169142.0 | -1.927883 | 1.125653 | -4.518331 | 1.749293 | -1.566487 | -2.010494 | -0.882850 |
| **280143** | 169347.0 | 1.378559 | 1.289381 | -5.004247 | 1.411850 | 0.442581 | -1.326536 | -1.413170 |
| **280149** | 169351.0 | -0.676143 | 1.126366 | -2.213700 | 0.468308 | -1.120541 | -0.003346 | -2.234739 |
| **281144** | 169966.0 | -3.113832 | 0.585864 | -5.399730 | 1.817092 | -0.840618 | -2.943548 | -2.208002 |
| **281674** | 170348.0 | 1.991976 | 0.158476 | -2.583441 | 0.408670 | 1.151147 | -0.096695 | 0.223050 |

5 rows × 31 columns

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

| Class | count |
|---|---|
| **0** | 492 |
| **1** | 492 |

**dtype:** int64

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

| Class | Time | V1 | V2 | V3 | V4 | V5 | V6 | V |
|---|---|---|---|---|---|---|---|---|
| **0** | 93421.455285 | 0.152470 | 0.086492 | 0.003421 | 0.029189 | 0.007979 | 0.042318 | -0.00927 |
| **1** | 80746.806911 | -4.771948 | 3.623778 | -7.033281 | 4.542029 | -3.151225 | -1.397737 | -5.56873 |

2 rows × 30 columns

Splitting the data into Features & Targets

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

```
print(X)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
126587   78035.0  1.160426 -0.502319  0.886433 -0.884251 -1.031212 -0.046391
97839    66406.0 -1.909825  1.256160  0.433707 -0.810739  0.170353 -0.370122
```

```
7407      9996.0  1.107242  0.667646  0.314573  2.653926  0.153394 -0.585697
56270    47345.0 -1.219131  0.893169  1.854603  1.652850  1.457932 -0.928450
131375   79577.0  1.156505 -0.773010 -0.138779 -1.923648 -0.614735 -0.508777
...          ...       ...       ...       ...       ...       ...       ...
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  \
126587 -0.804541  0.295474  1.672296  ... -0.194626  0.058137  0.378931
97839   0.435935  0.196471  0.457619  ... -0.266419 -0.294524 -0.553131
7407    0.381893 -0.204886  0.039885  ... -0.170259 -0.086024 -0.165620
56270   0.826119 -0.090274 -1.625399  ...  0.166811 -0.126169 -0.582343
131375 -0.138414 -0.095846  2.275762  ...  0.028865 -0.015037  0.244756
...          ...       ...       ...  ...       ...       ...       ...
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
126587 -0.009886  0.021266  0.372120 -0.671643  0.101383  0.020059    1.00
97839   0.212798 -0.571305 -0.588600 -0.209698 -0.885707 -0.558383    8.92
7407   -0.103467  0.471978  0.617368 -0.002281 -0.080451  0.000473   38.03
56270  -0.155451  0.360258  0.333332 -0.292738 -0.172752 -0.121043   11.37
131375 -0.273189 -0.359524  0.853840 -0.644211  0.081853  0.023791   79.15
...          ...       ...       ...       ...       ...       ...     ...
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]
```

```
print(Y)
```

```
126587    0
97839     0
7407      0
56270     0
131375    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

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y,
```

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

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

Model Training

Logistic Regression

```
model = LogisticRegression()
```

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

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: C
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-regressio
  n_iter_i = _check_optimize_result(
```

```
▾ LogisticRegression
LogisticRegression()
```

Model Evaluation

Accuracy Score