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	V9	...	V21	V22	V23	
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.(
1	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.:
2	1	-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.(
3	1	-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.:
4	2	-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.:

5 rows × 31 columns

```
credit_card_data.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V2
29794	35633	0.786689	-0.691214	-0.329291	0.149435	0.714779	1.949061	-0.136906	0.474172	0.206173	...	-0.165285	-0.793473	-0.03011
29795	35633	0.800996	-2.159993	0.008378	-1.081828	-1.768799	-0.445016	-0.571165	-0.162429	-1.785636	...	0.016930	-0.350492	-0.23488
29796	35633	1.115726	-0.472602	0.983034	0.294673	-1.218768	-0.341755	-0.667340	0.171155	0.805427	...	0.104463	0.366801	-0.07321
29797	35634	1.239103	-1.000617	0.843324	-0.560021	-1.400343	-0.151696	-1.026058	-0.001637	-0.131138	...	0.325954	0.855203	-0.24568
29798	35634	1.374193	-0.720679	0.891375	-0.541402	-1.700000	NaN	NaN	NaN	NaN	...	NaN	NaN	Na

5 rows × 31 columns

```
#dataset informations
credit_card_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29799 entries, 0 to 29798
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Time    29799 non-null  int64  
 1   V1      29799 non-null  float64
 2   V2      29799 non-null  float64
 3   V3      29799 non-null  float64
 4   V4      29799 non-null  float64
 5   V5      29799 non-null  float64
 6   V6      29798 non-null  float64
 7   V7      29798 non-null  float64
 8   V8      29798 non-null  float64
 9   V9      29798 non-null  float64
 10  V10     29798 non-null  float64
 11  V11     29798 non-null  float64
 12  V12     29798 non-null  float64
 13  V13     29798 non-null  float64
 14  V14     29798 non-null  float64
 15  V15     29798 non-null  float64
 16  V16     29798 non-null  float64
 17  V17     29798 non-null  float64
 18  V18     29798 non-null  float64
 19  V19     29798 non-null  float64
 20  V20     29798 non-null  float64
```

```
 21  V21     29798 non-null  float64
 22  V22     29798 non-null  float64
 23  V23     29798 non-null  float64
 24  V24     29798 non-null  float64
 25  V25     29798 non-null  float64
 26  V26     29798 non-null  float64
 27  V27     29798 non-null  float64
 28  V28     29798 non-null  float64
 29  Amount  29798 non-null  float64
 30  Class   29798 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 7.0 MB
```

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

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        1
V7        1
V8        1
V9        1
V10       1
V11       1
V12       1
V13       1
V14       1
V15       1
V16       1
V17       1
V18       1
V19       1
V20       1
V21       1
V22       1
V23       1
V24       1
V25       1
V26       1
V27       1
V28       1
Amount    1
Class     1
dtype: int64
```

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

```
Class
0.0    29704
1.0       94
Name: count, dtype: int64
```

This Dataset is highly unbalanced

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)
```

```
(29704, 31)
(94, 31)
```

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

```
count    29704.000000
mean        79.570030
std        221.991154
min          0.000000
25%          6.637500
50%         20.000000
75%         70.652500
max       7879.420000
Name: Amount, dtype: float64
```

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

```
count      94.000000
mean       95.590000
std       257.920621
min         0.000000
25%         1.000000
50%         1.050000
75%        99.990000
max      1809.680000
Name: Amount, dtype: float64
```

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

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	
Class														
0.0	21422.566422	-0.184395	0.106639	0.759413	0.194690	-0.186422	0.096790	-0.096841	0.018203	0.361273	...	0.044000	-0.035795	-(
1.0	19007.702128	-8.099702	6.084984	-11.565958	6.014185	-5.681925	-2.370349	-7.912202	4.043743	-2.891421	...	0.679513	0.573983	-(

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	V8	V9	...	V21	V22	
7755	10799	1.266542	0.015942	0.623743	-0.050315	-0.513566	-0.469132	-0.423446	-0.067397	1.470987	...	-0.256529	-0.522403	0.05
22921	32503	-1.950713	1.976018	0.718646	-0.084854	-0.389506	-0.873640	0.166806	-1.740505	0.081659	...	1.628822	-0.023158	0.17
16292	27689	0.808134	-0.364113	0.106022	1.335512	-0.045837	0.410419	0.231907	0.072522	-0.060674	...	0.041210	-0.045832	-0.27
12943	22744	-11.050688	6.768340	-11.068786	2.497231	-6.750103	-2.797250	-4.525065	7.175295	1.199926	...	-0.171703	-1.230148	-0.37
7864	10945	1.261789	-0.058362	0.552553	0.015400	-0.358434	-0.039203	-0.544068	-0.014109	1.506255	...	-0.067001	0.044749	-0.13

5 rows × 31 columns

```
new_dataset.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V2
27362	34521	1.081234	0.416414	0.862919	2.520863	-0.005021	0.563341	-0.123372	0.223122	-0.673598	...	-0.159387	-0.305154	0.05362
27627	34634	0.333499	1.699873	-2.596561	3.643945	-0.585068	-0.654659	-2.275789	0.675229	-2.042416	...	0.469212	-0.144363	-0.31798
27738	34684	-2.439237	2.591458	-2.840126	1.286244	-1.777016	-1.436139	-2.206056	-2.282725	-0.292885	...	1.774460	-0.771390	0.06572
27749	34687	-0.860827	3.131790	-5.052968	5.420941	-2.494141	-1.811287	-5.479117	1.189472	-3.908206	...	1.192694	0.090356	-0.34188
29687	35585	-2.019001	1.491270	0.005222	0.817253	0.973252	-0.639268	-0.974073	-3.146929	-0.003159	...	2.839596	-1.185443	-0.14281

5 rows × 31 columns

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

```
Class
0.0    492
1.0     94
Name: count, dtype: int64
```

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

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	
Class														
0.0	21659.467480	-0.126652	0.159711	0.648066	0.132789	-0.223356	0.054391	-0.116954	0.051761	0.285043	...	0.079924	-0.040564	-(
1.0	19007.702128	-8.099702	6.084984	-11.565958	6.014185	-5.681925	-2.370349	-7.912202	4.043743	-2.891421	...	0.679513	0.573983	-(

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  \
7755   10799   1.266542  0.015942   0.623743 -0.050315 -0.513566 -0.469132
22921  32503  -1.950713  1.976018   0.718646 -0.084854 -0.389506 -0.873640
16292  27689   0.808134 -0.364113   0.106022  1.335512 -0.045837  0.410419
12943  22744 -11.050688  6.768340 -11.068786  2.497231 -6.750103 -2.797250
7864   10945   1.261789 -0.058362   0.552553  0.015400 -0.358434 -0.039203
...      ...        ...       ...        ...       ...       ...       ...
27362  34521   1.081234  0.416414   0.862919  2.520863 -0.005021  0.563341
27627  34634   0.333499  1.699873  -2.596561  3.643945 -0.585068 -0.654659
27738  34684  -2.439237  2.591458  -2.840126  1.286244 -1.777016 -1.436139
27749  34687  -0.860827  3.131790  -5.052968  5.420941 -2.494141 -1.811287
29687  35585  -2.019001  1.491270   0.005222  0.817253  0.973252 -0.639268

             V7        V8        V9  ...       V20       V21       V22  \
7755  -0.423446 -0.067397  1.470987  ... -0.098823 -0.256529 -0.522403
22921  0.166806 -1.740505  0.081659  ... -0.358938  1.628822 -0.023158
16292  0.231907  0.072522 -0.060674  ...  0.226764  0.041210 -0.045832
12943 -4.525065  7.175295  1.199926  ...  0.067608 -0.171703 -1.230148
7864  -0.544068 -0.014109  1.506255  ... -0.031828 -0.067001  0.044749
...         ...       ...       ...  ...       ...       ...       ...
27362 -0.123372  0.223122 -0.673598  ... -0.165249 -0.159387 -0.305154
27627 -2.275789  0.675229 -2.042416  ...  0.329342  0.469212 -0.144363
27738 -2.206056 -2.282725 -0.292885  ...  0.513530  1.774460 -0.771390
27749 -5.479117  1.189472 -3.908206  ...  1.085760  1.192694  0.090356
29687 -0.974073 -3.146929 -0.003159  ... -1.029965  2.839596 -1.185443

            V23       V24       V25       V26       V27       V28  Amount
7755   0.058309 -0.000197  0.101932  0.869285 -0.102920 -0.010869    2.14
22921  0.179898  0.720869 -0.340300  0.156747 -1.005075 -0.552337    3.87
16292 -0.276343 -0.261509  0.633807 -0.330598  0.006688  0.034265  183.00
12943 -0.373350  0.369727  0.427764 -0.425988 -0.140189 -0.318567   89.99
7864  -0.132443 -0.475118  0.293059  1.064506 -0.093105 -0.012373   15.95
...         ...       ...       ...       ...       ...       ...     ...
27362  0.053620  0.011761  0.375146 -0.106299  0.021008  0.010559    1.52
27627 -0.317981 -0.769644  0.807855  0.228164  0.551002  0.305473   18.96
27738  0.065727  0.103916 -0.057578  0.242652 -0.268649 -0.743713  125.30
27749 -0.341881 -0.215924  1.053032  0.271139  1.373300  0.691195   19.02
```

```
29687 -0.142812 -0.086103 -0.329113  0.523601  0.626283  0.152440    0.76

[586 rows x 30 columns]
```

```
print(Y)
```

```
7755     0.0
22921    0.0
16292    0.0
12943    0.0
7864     0.0
        ...
27362    1.0
27627    1.0
27738    1.0
27749    1.0
29687    1.0
Name: Class, Length: 586, dtype: float64
```

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, random_state=2)
```

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

```
(586, 30) (468, 30) (118, 30)
```

Model Training

Logistic Regression

```
model = LogisticRegression()
```

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

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:460: 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(
▸ LogisticRegression
```

Model Evaluation

Accuracy Score

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

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

```
Accuracy on Training data :  0.9722222222222222
```

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

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

```
Accuracy score on Test Data :  0.9322033898305084
```