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 dataset to 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	V10	V11	V12	
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.

```
credit_card_data.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V1
29794	35633	0.786689	-0.691214	-0.329291	0.149435	0.714779	1.949061	-0.136906	0.474172	0.206173	-0.367534	-0.534179	0.17163
29795	35633	0.800996	-2.159993	0.008378	-1.081828	-1.768799	-0.445016	-0.571165	-0.162429	-1.785636	1.254357	-0.369288	-1.14765
29796	35633	1.115726	-0.472602	0.983034	0.294673	-1.218768	-0.341755	-0.667340	0.171155	0.805427	-0.106976	0.989409	0.54558
29797	35634	1.239103	-1.000617	0.843324	-0.560021	-1.400343	-0.151696	-1.026058	-0.001637	-0.131138	0.458943	-0.971230	-0.42194
29798	35634	1.374193	-0.720679	0.891375	-0.541402	-1.700000	NaN	NaN	NaN	NaN	NaN	NaN	Nal

```
#number of rows and columns
credit_card_data.shape
```

```
(29799, 31)
```

```
#dataset information
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      29799 non-null  float64
 7   V7      29799 non-null  float64
 8   V8      29799 non-null  float64
 9   V9      29799 non-null  float64
 10  V10     29799 non-null  float64
 11  V11     29799 non-null  float64
 12  V12     29799 non-null  float64
 13  V13     29799 non-null  float64
 14  V14     29799 non-null  float64
 15  V15     29799 non-null  float64
 16  V16     29799 non-null  float64
 17  V17     29799 non-null  float64
 18  V18     29799 non-null  float64
 19  V19     29799 non-null  float64
 20  V20     29799 non-null  float64
 21  V21     29799 non-null  float64
 22  V22     29799 non-null  float64
 23  V23     29799 non-null  float64
 24  V24     29799 non-null  float64
 25  V25     29799 non-null  float64
 26  V26     29799 non-null  float64
```

```
 27  V27     29799 non-null  float64
 28  V28     29799 non-null  float64
 29  Amount  29799 non-null  float64
 30  Class   29799 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 7.0 MB
```

```
#check missing values in dataset
credit_card_data.isnull().sum()
```

	0
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

```
credit_card_data_cleaned = credit_card_data.dropna()
```

```
print (credit_card_data_cleaned)
```

```
        Time        V1        V2        V3        V4  ...       V26       V27       V28  Amount  Class
0          0 -1.359807 -0.072781  2.536347  1.378155  ... -0.189115  0.133558 -0.021053  149.62    0.0
1          0  1.191857  0.266151  0.166480  0.448154  ...  0.125895 -0.008983  0.014724    2.69    0.0
2          1 -1.358354 -1.340163  1.773209  0.379780  ... -0.139097 -0.055353 -0.059752  378.66    0.0
3          1 -0.966272 -0.185226  1.792993 -0.863291  ... -0.221929  0.062723  0.061458  123.50    0.0
4          2 -1.158233  0.877737  1.548718  0.403034  ...  0.502292  0.219422  0.215153   69.99    0.0
...      ...       ...       ...       ...       ...  ...       ...       ...       ...     ...    ...
35736  38239 -3.120568 -0.121316  1.307875  1.582101  ... -0.070668 -0.870421  0.624730    2.28    0.0
35737  38240  1.232390  0.122010  0.157352  0.261906  ...  0.158446 -0.018118 -0.003168    1.79    0.0
35738  38240  1.114040  0.571203  0.427035  2.442135  ... -0.025773 -0.035775  0.011219   24.99    0.0
35739  38241  1.057020  0.007895  0.239256  1.236048  ... -0.314989  0.027467  0.010613   53.96    0.0
35740  38241 -1.546226  0.693338  1.002815 -1.528992  ...  0.739989  0.043625 -0.140629    0.76    0.0
```

```
[35741 rows x 31 columns]
```

```
#distribution of legit transaction and fradulent transaction
credit_card_data['Class'].value_counts()
```

Class	count
0.0	35638
1.0	103

This Dataset is highly unbalanced

0 = Normal Transaction

1 = Fradulent Transaction

```
#seperating 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)
```

```
(35638, 31)
(103, 31)
```

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

	Amount
count	35638.000000
mean	84.185241
std	227.223359
min	0.000000
25%	7.000000
50%	22.000000
75%	76.000000
max	7879.420000

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

	Amount
count	103.000000
mean	90.471165
std	247.173335
min	0.000000
25%	1.000000
50%	3.760000
75%	99.990000
max	1809.680000

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

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	
Class													
0.0	24007.013890	-0.186593	0.055909	0.752026	0.179227	-0.201556	0.102516	-0.094806	0.021671	0.268769	-0.067683	0.441763	-0
1.0	20498.291262	-7.762676	5.838647	-10.957218	5.911555	-5.453798	-2.306085	-7.743100	3.867920	-2.961800	-6.700936	5.501599	-8

Under Sampling

Build sample datset containing similar distribution of normal and fradulent transaction

```
legit_sample = legit.sample(n=103)
```

Concatenation two dataframes

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

```
new_dataset.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12
19656	30442	0.960089	-0.833866	0.849283	0.633604	-1.385595	-0.318221	-0.578834	0.027567	-1.033440	0.946116	1.172593	0.510031
33533	37268	1.149505	0.396102	0.615531	2.310495	-0.180587	-0.071877	-0.104923	0.173158	-0.685428	0.893060	0.597537	-0.479812
5015	4596	1.116750	-0.470737	0.228822	-1.806058	-0.133105	0.472825	-0.437491	0.206482	3.326446	-2.211929	1.892923	-0.755930
7505	10247	1.062773	-0.093531	1.485816	1.439849	-0.880143	0.484425	-0.923960	0.357869	2.100503	-0.406822	2.034377	-1.887537
7480	10180	-0.467669	1.075407	1.725491	0.062699	0.068057	-0.543345	0.483334	0.011039	0.655667	-0.506955	2.218326	-1.950564

```
new_dataset.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12
30442	35926	-3.896583	4.518355	-4.454027	5.547453	-4.121459	-1.163407	-6.805053	2.928356	-4.917130	-6.600461	3.367846	-7.888978
30473	35942	-4.194074	4.382897	-5.118363	4.455230	-4.812621	-1.224645	-7.281328	3.332250	-3.679659	-7.524368	2.954344	-7.099825
30496	35953	-4.844372	5.649439	-6.730396	5.252842	-4.409566	-1.740767	-6.311699	3.449167	-5.416284	-7.833556	3.657350	-7.781448
31002	36170	-5.685013	5.776516	-7.064977	5.902715	-4.715564	-1.755633	-6.958679	3.877795	-5.541529	-7.502112	3.676703	-7.642983
33276	37167	-7.923891	-5.198360	-3.000024	4.420666	2.272194	-3.394483	-5.283435	0.131619	0.658176	-0.794994	3.266066	-2.719185

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

	count
Class	
0.0	103
1.0	103

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

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	
Class													
0.0	21641.097087	0.117184	0.202749	0.905756	0.477824	0.018123	0.191926	-0.01695	0.093605	0.36955	-0.163345	0.479711	-0.7
1.0	20498.291262	-7.762676	5.838647	-10.957218	5.911555	-5.453798	-2.306085	-7.74310	3.867920	-2.96180	-6.700936	5.501599	-8.4

Spliting the data in features and targets

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

```
print(X)
```

```
         Time        V1        V2        V3  ...       V26       V27       V28  Amount
19656   30442  0.960089 -0.833866  0.849283  ... -0.344320  0.036178  0.055045  158.00
33533   37268  1.149505  0.396102  0.615531  ... -0.257171 -0.024791  0.012328   10.51
5015     4596  1.116750 -0.470737  0.228822  ... -0.768814  0.117068  0.001664    2.18
7505    10247  1.062773 -0.093531  1.485816  ... -0.442725  0.050931  0.016670    4.99
7480    10180 -0.467669  1.075407  1.725491  ...  0.015819  0.232992  0.104472    2.67
...       ...       ...       ...       ...  ...       ...       ...       ...     ...
30442   35926 -3.896583  4.518355 -4.454027  ...  0.412191  0.635789  0.501050    4.56
30473   35942 -4.194074  4.382897 -5.118363  ... -0.131687  0.473934  0.473757   14.46
30496   35953 -4.844372  5.649439 -6.730396  ... -0.270328  0.210214  0.391855  111.70
31002   36170 -5.685013  5.776516 -7.064977  ... -0.049447  0.303445  0.219380  111.70
33276   37167 -7.923891 -5.198360 -3.000024  ...  0.520508  1.937421 -1.552593   12.31

[206 rows x 30 columns]
```

```
print(Y)
```

```
19656    0.0
33533    0.0
5015     0.0
7505     0.0
7480     0.0
        ...
30442    1.0
30473    1.0
30496    1.0
31002    1.0
33276    1.0
Name: Class, Length: 206, dtype: float64
```

split data in training and test data

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.3,stratify=Y,random_state=3)
```

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

```
(206, 30) (144, 30) (62, 30)
```

Model Training

Logistic Regression

```
model = LogisticRegression()
```

```
#training 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: ConvergenceWarning: lbfgs failed to converge (status=
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 ⓘ ?

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 on test data : ',test_data_accuracy)
```

```
Accuracy on test data :  0.9838709677419355
```