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()
# printing the last five rows of the dataset
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	\
107661	70549	1.198279	-0.851404	-0.172063	-0.760588	-0.732620	-0.390914	-0.308317	-0.108760	-1.116133	...	-0.317584	-0.724378	-0.145:
107662	70549	-4.608890	-2.417203	-1.684811	0.206299	1.240726	3.368097	-1.802464	1.883831	-1.202476	...	-0.379691	-0.212864	-0.474!
107663	70550	-1.125806	0.163657	1.873904	1.407388	0.400394	0.167562	0.488385	0.250106	-0.798662	...	0.117333	0.190644	0.003!
107664	70550	-0.423253	0.752399	1.577846	0.280206	-0.094335	-0.263924	0.474498	0.088235	0.119206	...	0.051250	0.557695	-0.004(
107665	70550	0.907958	-0.923497	0.730426	0.211849	-0.948813	0.409957	-0.555812	0.199092	1.260090	...	-0.149574	-0.492247	-0.083!

5 rows × 31 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 107666 entries, 0 to 107665
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    107666 non-null  int64
 1   V1      107666 non-null  float64
 2   V2      107666 non-null  float64
 3   V3      107666 non-null  float64
 4   V4      107666 non-null  float64
 5   V5      107666 non-null  float64
 6   V6      107666 non-null  float64
 7   V7      107666 non-null  float64
 8   V8      107666 non-null  float64
 9   V9      107666 non-null  float64
 10  V10     107666 non-null  float64
 11  V11     107666 non-null  float64
 12  V12     107666 non-null  float64
 13  V13     107666 non-null  float64
 14  V14     107666 non-null  float64
 15  V15     107666 non-null  float64
 16  V16     107666 non-null  float64
 17  V17     107666 non-null  float64
 18  V18     107666 non-null  float64
 19  V19     107666 non-null  float64
 20  V20     107666 non-null  float64
 21  V21     107666 non-null  float64
```

```
 22  V22     107666 non-null  float64
 23  V23     107666 non-null  float64
 24  V24     107666 non-null  float64
 25  V25     107666 non-null  float64
 26  V26     107666 non-null  float64
 27  V27     107666 non-null  float64
 28  V28     107665 non-null  float64
 29  Amount  107665 non-null  float64
 30  Class   107665 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 25.5 MB
```

```
# checking the number of misssing value in each column
credit_card_data.isnull().sum()
```

```
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       1
Amount    1
Class     1
dtype: int64
```

```
# dropping all the missing values
credit_card_data=credit_card_data.dropna()
```

```
credit_card_data.isnull().sum()
```

```
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
```

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

```
0.0    107428
1.0       237
Name: Class, dtype: int64
```

This data is highly unbalaced

0----->Normal transaction 1----->Fradulent 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)
```

```
(107428, 31)
(237, 31)
```

```
# stastical measure of the data
legit.Amount.describe()
```

```
count    107428.000000
mean         96.131142
std         260.844741
min           0.000000
25%           7.100000
50%          25.170000
75%          86.970000
max       19656.530000
Name: Amount, dtype: float64
```

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

```
count     237.000000
mean      118.885148
std       255.864211
min         0.000000
25%         1.000000
50%         7.610000
75%        99.990000
max      1809.680000
Name: Amount, dtype: float64
```

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

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	
0.0	44326.693739	-0.244646	-0.035961	0.700676	0.146649	-0.275017	0.101216	-0.102301	0.055409	-0.049402	...	0.043082	-0.033876	-0.
1.0	38578.008439	-6.020609	4.209147	-7.728592	4.759216	-4.298181	-1.547669	-6.396113	1.618176	-2.755420	...	0.252535	1.368462	-0.

2 rows × 30 columns

Under-Sampling

Build a sample dataset containing similar distribution of normal transaction and fraudulent transactions

Number of fradulent transaction is 237

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

Concatinating two dataframe

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

```
new_dataset.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23
62549	50313	-0.933480	0.370584	2.380690	0.806094	-1.256009	1.119959	-0.767293	0.912573	1.284643	...	0.271730	1.149106	-0.111641
42168	40978	-0.675169	1.383807	0.542348	0.948633	-0.363193	-0.856115	0.365628	0.233094	-0.403687	...	0.138380	0.391091	0.018021
74654	55687	1.253583	0.107155	-0.007781	0.332624	-0.365354	-1.251624	0.287560	[illegible]	0.079325	...	-0.059514	-0.215625	-0.064891
11911	20525	1.145316	0.071595	0.827779	1.230429	-0.371442	0.355653	-0.668756	0.278263	1.628965	...	0.005147	0.214609	-0.056842
46095	42638	-1.160903	1.151422	-0.918765	0.872705	-0.085144	-0.277674	0.202334	0.899959	-0.761588	...	0.221992	0.617456	0.039441

5 rows × 31 columns

```
new_dataset.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V2
105178	69394	1.140431	1.134243	-1.429455	2.012226	0.622800	-1.152923	0.221159	0.037372	0.034486	...	-0.367136	-0.891627	-0.16057
106679	70071	-0.440095	1.137239	-3.227080	3.242293	-2.033998	-1.618415	-3.028013	0.764555	-1.801937	...	0.764187	-0.275578	-0.34357
106998	70229	0.315642	1.636778	-1.519650	4.028571	-1.186794	-0.789813	-2.279807	0.472988	-1.657635	...	0.345921	-0.108002	-0.16544
107067	70270	-1.512516	1.133139	-1.601052	2.813401	-2.664503	-0.310371	-1.520895	0.852996	-1.496495	...	0.729828	0.485286	0.5670(
107637	70536	-2.271755	-0.457655	-2.589055	2.230778	-4.278983	0.388610	0.102485	0.813128	-1.092921	...	1.096342	0.658399	1.71167

5 rows × 31 columns

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

```
0.0    492
1.0    237
Name: Class, dtype: int64
```

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

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	
Class														
0.0	41195.428862	-0.250206	-0.036615	0.613645	0.097205	-0.258968	0.170005	-0.212102	0.057319	0.014821	...	0.062665	-0.003914	-0
1.0	38578.008439	-6.020609	4.209147	-7.728592	4.759216	-4.298181	-1.547669	-6.396113	1.618176	-2.755420	...	0.252535	1.368462	-0.

2 rows × 30 columns

Splitting the data into features and target

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

```
print(X)
```

```
         Time        V1        V2        V3        V4        V5        V6  \
62549   50313 -0.933480  0.370584  2.380690  0.806094 -1.256009  1.119959
42168   40978 -0.675169  1.383807  0.542348  0.948633 -0.363193 -0.856115
74654   55687  1.253583  0.107155 -0.007781  0.332624 -0.365354 -1.251624
11911   20525  1.145316  0.071595  0.827779  1.230429 -0.371442  0.355653
46095   42638 -1.160903  1.151422 -0.918765  0.872705 -0.085144 -0.277674
...       ...       ...       ...       ...       ...       ...       ...
105178  69394  1.140431  1.134243 -1.429455  2.012226  0.622800 -1.152923
106679  70071 -0.440095  1.137239 -3.227080  3.242293 -2.033998 -1.618415
```

```
106998  70229  0.315642  1.636778 -1.519650  4.028571 -1.186794 -0.789813
107067  70270 -1.512516  1.133139 -1.601052  2.813401 -2.664503 -0.310371
107637  70536 -2.271755 -0.457655 -2.589055  2.230778 -4.278983  0.388610

              V7        V8        V9  ...       V20       V21       V22  \
62549  -0.767293  0.912573  1.284643  ... -0.330535  0.271730  1.149106
42168   0.365628  0.233094 -0.403687  ... -0.092365  0.138380  0.391091
74654   0.287560 -0.296029  0.079325  ... -0.085239 -0.059514 -0.215625
11911  -0.668756  0.278263  1.628965  ... -0.320584  0.005147  0.214609
46095   0.202334  0.899959 -0.761588  ... -0.116223  0.221992  0.617456
...          ...       ...       ...  ...       ...       ...       ...
105178  0.221159  0.037372  0.034486  ... -0.099712 -0.367136 -0.891627
106679 -3.028013  0.764555 -1.801937  ...  0.895841  0.764187 -0.275578
106998 -2.279807  0.472988 -1.657635  ...  0.388885  0.345921 -0.108002
107067 -1.520895  0.852996 -1.496495  ...  1.249586  0.729828  0.485286
107637  0.102485  0.813128 -1.092921  ...  2.285758  1.096342  0.658399

             V23       V24       V25       V26       V27       V28  Amount
62549  -0.111641  0.151708 -0.116496 -0.300782 -0.060142 -0.018001   53.00
42168   0.018021  0.381528 -0.083928 -0.330524 -0.043626  0.019685   18.90
74654  -0.064891  0.429353  0.495357  0.599883 -0.077577  0.004861   27.83
11911  -0.056842 -0.413966  0.361166 -0.289520  0.015357  0.003739    3.00
46095   0.039441 -0.309553 -0.723957 -0.370495  0.412846  0.013050   89.99
...          ...       ...       ...       ...       ...       ...     ...
105178 -0.160578 -0.108326  0.668374 -0.352393  0.071993  0.113684    1.00
106679 -0.343572  0.233085  0.606434 -0.315433  0.768291  0.459623  227.30
106998 -0.165442  0.279895  0.808783  0.117363  0.589595  0.309064    3.79
107067  0.567005  0.323586  0.040871  0.825814  0.414482  0.267265  318.11
107637  1.711676  0.333540  0.538591 -0.193529  0.258194  0.247269  824.83

[729 rows x 30 columns]
```

```
print(Y)
```

```
62549     0.0
42168     0.0
74654     0.0
11911     0.0
46095     0.0
         ...
105178    1.0
106679    1.0
106998    1.0
107067    1.0
107637    1.0
Name: Class, Length: 729, dtype: float64
```

Splitting the data into training and 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)
```

```
(729, 30) (583, 30) (146, 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:458: 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
LogisticRegression()
```

Model Evalution

+ Code + Text

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: 0.9554030874785592
```

```
# 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: 0.9041095890410958
```