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')
```

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

	Time	V1	V2	V3	V4	V5	V6	V7	V
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.09869
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08510
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24767
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37743
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27053

```
#print last 5 rows of the dataset
credit_card_data.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	
19893	30631	-0.377215	0.973528	1.647077	0.732439	0.024728	-0.541379	0.828488	-0.
19894	30631	1.209281	0.078793	0.061820	0.593730	-0.235772	-0.448524	-0.141196	0.
19895	30632	1.286596	-1.450336	0.814530	-1.308949	-2.055209	-0.592064	-1.317286	0.
19896	30633	-0.488090	1.018448	0.670593	-0.245462	0.828347	-0.233102	0.662586	-0.
19897	30633	-2.609841	2.479357	0.763844	0.044509	-0.645716	0.762867	-1.626415	-7.

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19898 entries, 0 to 19897
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Time    19898 non-null  int64  
 1   V1      19898 non-null  float64
 2   V2      19898 non-null  float64
 3   V3      19898 non-null  float64
 4   V4      19898 non-null  float64
 5   V5      19898 non-null  float64
 6   V6      19898 non-null  float64
 7   V7      19898 non-null  float64
 8   V8      19898 non-null  float64
 9   V9      19898 non-null  float64
 10  V10     19898 non-null  float64
 11  V11     19897 non-null  float64
 12  V12     19897 non-null  float64
 13  V13     19897 non-null  float64
 14  V14     19897 non-null  float64
 15  V15     19897 non-null  float64
 16  V16     19897 non-null  float64
 17  V17     19897 non-null  float64
 18  V18     19897 non-null  float64
 19  V19     19897 non-null  float64
 20  V20     19897 non-null  float64
 21  V21     19897 non-null  float64
 22  V22     19897 non-null  float64
 23  V23     19897 non-null  float64
 24  V24     19897 non-null  float64
 25  V25     19897 non-null  float64
 26  V26     19897 non-null  float64
 27  V27     19897 non-null  float64
 28  V28     19897 non-null  float64
 29  Amount  19897 non-null  float64
 30  Class   19897 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 4.7 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        0
V7        0
V8        0
V9        0
V10       0
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 and fraudulent transaction
credit_card_data['Class'].value_counts()
```

```
0.0    19812
1.0       85
Name: Class, dtype: int64
```

the 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)
```

```
(19812, 31)
(85, 31)
```

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

```
count    19812.000000
mean        70.169855
std        205.091118
min          0.000000
25%          5.900000
50%         16.070000
75%         59.950000
max       7879.420000
Name: Amount, dtype: float64
```

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

```
count      85.000000
mean       93.869647
std       261.736641
min         0.000000
25%         1.000000
50%         1.000000
75%        99.990000
max      1809.680000
Name: Amount, dtype: float64
```

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

Class	Time	V1	V2	V3	V4	V5	V6	
0.0	15483.312841	-0.207233	0.215159	0.803126	0.250998	-0.137392	0.104262	-0.108
1.0	17436.164706	-8.862174	6.570214	-12.622087	6.343007	-6.187820	-2.567646	-8.600

Under_ Sampling

Build a sample dataset conatining similar distribution of normal transaction and fraudulent transaction

Number of fraudulent transactione->85

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

Concatenating two data frames

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

```
new_dataset.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	
4304	3758	-0.693825	-0.158285	1.621470	-1.644706	-0.647281	-0.398048	0.548695	-0
6086	6942	1.475676	-0.082584	-0.176180	-0.551063	-0.255257	-1.112522	-0.028757	-0
4246	3754	1.239582	-0.421236	0.792134	-0.469097	-0.762769	-0.213225	-0.667352	-0
13550	24036	1.017670	-0.719840	0.285711	0.886092	1.162706	4.830132	-1.608480	1
16748	28109	-0.496303	0.948806	1.625647	0.382160	0.143376	-0.099795	0.539903	0

```
new_dataset.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7
17480	28755	-30.552380	16.713389	-31.103685	6.534984	-22.105532	-4.977692	-20.371514
18466	29526	1.102804	2.829168	-3.932870	4.707691	2.937967	-1.800904	1.672734
18472	29531	-1.060676	2.608579	-2.971679	4.360089	3.738853	-2.728395	1.987616
18773	29753	0.269614	3.549755	-5.810353	5.809370	1.538808	-2.269219	-0.824203
18809	29785	0.923764	0.344048	-2.880004	1.721680	-3.019565	-0.639736	-3.801325

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

```
0.0    492
1.0     85
Name: Class, dtype: int64
```

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

Class	Time	V1	V2	V3	V4	V5	V6	
0.0	15501.339431	-0.085908	0.154170	0.802138	0.288643	-0.098097	0.117210	-0.057
1.0	17436.164706	-8.862174	6.570214	-12.622087	6.343007	-6.187820	-2.567646	-8.600

Splitting the data into features and targets(0/1)

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

```
print(X)
```

```
        Time         V1         V2  ...       V27       V28  Amount
4304    3758  -0.693825  -0.158285  ...  0.028479  0.128220  151.94
6086    6942   1.475676  -0.082584  ... -0.046937 -0.004705   15.00
4246    3754   1.239582  -0.421236  ...  0.018817  0.020705   40.07
13550  24036   1.017670  -0.719840  ...  0.071766  0.037366   76.80
16748  28109  -0.496303   0.948806  ... -0.071135  0.021213    9.99
...      ...        ...        ...  ...       ...       ...     ...
17480  28755 -30.552380  16.713389  ...  1.232636  0.356660   99.99
18466  29526   1.102804   2.829168  ...  0.009146  0.153318    0.68
18472  29531  -1.060676   2.608579  ... -0.130918  0.192177    0.68
18773  29753   0.269614   3.549755  ...  0.553255  0.402400    0.68
18809  29785   0.923764   0.344048  ...  0.489035 -0.049729   30.30

[577 rows x 30 columns]
```

```
print(Y)
```

```
4304     0.0
6086     0.0
4246     0.0
13550    0.0
16748    0.0
        ...
17480    1.0
18466    1.0
18472    1.0
18773    1.0
18809    1.0
Name: Class, Length: 577, dtype: float64
```

Split the data into training data and test data

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

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

```
(577, 30) (461, 30) (116, 30)
```

Model Training

Logistic Regression

```
model = LogisticRegression()
```

```
#training the logistic regression model with training data
```

```
model.fit(X_train, Y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Convergenc
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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Model evaluation

Based on 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: 0.9891540130151844
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
#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: 0.9827586206896551
```