```
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
```

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

```
Mounted at /content/drive
```

```
# 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
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431

5 rows × 31 columns

```
credit_card_data.tail()
```

Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21
25807	-0.769852	2.704375	-2.083145	1.018899	1.083598	-1.255315	1.242032	-0.525902	1.466585	...	-0.448296
25808	-0.897475	0.963371	0.997351	0.329928	0.998766	-1.287190	0.713085	0.019353	-0.859152	...	0.118559
25809	-0.377066	0.984515	0.988848	-0.261443	0.563332	0.197124	0.489867	0.281753	-0.543286	...	-0.234987
25810	-0.353184	0.311241	1.586426	-1.515835	-0.636334	-0.836015	0.441214	-0.188933	1.218595	...	0.049963
25810	0.827638	-0.539202	1.108173	1.532278	-0.950308	0.344304	-0.467828	0.217786	0.858742	...	NaN

× 31 columns

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14595 entries, 0 to 14594
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Time    14595 non-null  int64
 1   V1      14595 non-null  float64
 2   V2      14595 non-null  float64
 3   V3      14595 non-null  float64
 4   V4      14595 non-null  float64
```

```
 5   V5      14595 non-null  float64
 6   V6      14595 non-null  float64
 7   V7      14595 non-null  float64
 8   V8      14595 non-null  float64
 9   V9      14595 non-null  float64
 10  V10     14595 non-null  float64
 11  V11     14595 non-null  float64
 12  V12     14595 non-null  float64
 13  V13     14595 non-null  float64
 14  V14     14595 non-null  float64
 15  V15     14595 non-null  float64
 16  V16     14595 non-null  float64
 17  V17     14595 non-null  float64
 18  V18     14595 non-null  float64
 19  V19     14595 non-null  float64
 20  V20     14595 non-null  float64
 21  V21     14594 non-null  float64
 22  V22     14594 non-null  float64
 23  V23     14594 non-null  float64
 24  V24     14594 non-null  float64
 25  V25     14594 non-null  float64
 26  V26     14594 non-null  float64
 27  V27     14594 non-null  float64
 28  V28     14594 non-null  float64
 29  Amount  14594 non-null  float64
 30  Class   14594 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 3.5 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       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
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 & fraudulent transactions
credit_card_data['Class'].value_counts()
```

```
0.0    14533
1.0       61
Name: Class, 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)
```

```
(14533, 31)
(61, 31)
```

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

```
count    14533.000000
mean        64.065668
std        176.589083
min          0.000000
25%          5.550000
50%         15.950000
75%         52.990000
max       7712.430000
Name: Amount, dtype: float64
```

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

```
count      61.000000
mean       88.402295
std       297.522823
min         0.000000
25%         1.000000
50%         1.000000
75%         3.790000
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	..
Class											
0.0	10777.717402	-0.215448	0.262816	0.879788	0.274916	-0.106953	0.136946	-0.128751	-0.019781	0.974505	
1.0	13325.934426	-5.118440	4.839489	-9.563219	6.621692	-3.266514	-2.089348	-6.722938	1.489551	-2.929718	

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	...	
5111	4784	1.374105	-0.253948	0.113605	0.069158	-0.052759	0.404988	-0.408880	-0.164304	0.450830	...	-0.6
7770	10823	0.625403	-1.140300	0.322045	0.473485	-0.836196	0.153164	-0.164060	0.058232	1.930849	...	-0.0
6116	7001	0.771466	-0.598544	2.246886	3.139235	-1.127466	2.269853	-1.620516	0.885450	2.449515	...	0.0
13185	23170	-3.743846	-3.265379	0.995384	3.041958	0.965919	-0.587442	1.862605	-0.117721	-0.399114	...	0.0
10090	15324	-3.690755	3.414470	0.714492	-1.985861	-0.543478	-1.366884	1.123768	-0.849273	5.116339	...	-0.9

5 rows × 31 columns

```
new_dataset.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
14104	25095	1.192396	1.338974	-0.678876	3.123672	0.643245	-1.184323	0.397586	-0.253499	0.411135	...
14170	25198	-15.903635	10.393917	-19.133602	6.185969	-12.538021	-4.027030	-13.897827	10.662252	-2.844954	...
14197	25231	-16.598665	10.541751	-19.818982	6.017295	-13.025901	-4.128779	-14.118865	11.161144	-4.099551	...
14211	25254	-17.275191	10.819665	-20.363886	6.046612	-13.465033	-4.166647	-14.409448	11.580797	-4.073856	...
14338	25426	1.125336	1.130146	-0.962975	2.675688	0.990075	-0.243318	0.316192	0.122960	-1.143343	...

5 rows × 31 columns

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

```
0.0    492
1.0     61
Name: Class, dtype: int64
```

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

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	..
0.0	10255.290650	-0.228924	0.404879	0.835122	0.248904	-0.041809	0.170144	-0.116762	-0.053026	0.923775	
1.0	13325.934426	-5.118440	4.839489	-9.563219	6.621692	-3.266514	-2.089348	-6.722938	1.489551	-2.929718	

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  \
5111    4784   1.374105  -0.253948   0.113605  0.069158  -0.052759  0.404988
7770   10823   0.625403  -1.140300   0.322045  0.473485  -0.836196  0.153164
6116    7001   0.771466  -0.598544   2.246886  3.139235  -1.127466  2.269853
13185  23170  -3.743846  -3.265379   0.995384  3.041958   0.965919 -0.587442
10090  15324  -3.690755   3.414470   0.714492 -1.985861  -0.543478 -1.366884
...      ...        ...        ...        ...       ...        ...       ...
14104  25095   1.192396   1.338974  -0.678876  3.123672   0.643245 -1.184323
14170  25198 -15.903635  10.393917 -19.133602  6.185969 -12.538021 -4.027030
14197  25231 -16.598665  10.541751 -19.818982  6.017295 -13.025901 -4.128779
14211  25254 -17.275191  10.819665 -20.363886  6.046612 -13.465033 -4.166647
14338  25426   1.125336   1.130146  -0.962975  2.675688   0.990075 -0.243318

              V7         V8        V9  ...       V20       V21       V22  \
5111   -0.408880  -0.164304  0.450830  ... -0.455944 -0.688367 -0.964160
7770   -0.164060   0.058232  1.930849  ...  0.360946 -0.025609 -0.280835
6116   -1.620516   0.885450  2.449515  ... -0.242193  0.043859  0.629459
13185   1.862605  -0.117721 -0.399114  ...  2.299492  0.055303 -1.963346
10090   1.123768  -0.849273  5.116339  ...  2.370351 -0.965167 -0.590039
...          ...        ...       ...  ...       ...       ...       ...
14104   0.397586  -0.253499  0.411135  ... -0.185455 -0.377503 -0.889597
14170 -13.897827  10.662252 -2.844954  ...  1.501565  1.577548 -1.280137
14197 -14.118865  11.161144 -4.099551  ...  1.534920  1.725853 -1.151606
14211 -14.409448  11.580797 -4.073856  ...  1.544970  1.729804 -1.208096
14338   0.316192   0.122960 -1.143343  ... -0.138814 -0.166737 -0.521934

            V23       V24       V25       V26       V27       V28  Amount
5111  -0.087617 -0.971928  0.565894  0.418575 -0.006715 -0.002836   14.00
7770  -0.269747  0.043434  0.176311  1.058848 -0.141906  0.027320  291.88
6116  -0.097964 -0.327014  0.179940  0.251557  0.070432  0.025485   78.95
13185  2.270957 -0.387028  0.510184 -0.487578 -0.377596  0.173759  788.77
10090  0.046301  0.623774  0.368253  0.566924  0.708040 -0.355509    2.31
...         ...       ...       ...       ...       ...       ...     ...
14104 -0.074208  0.035446  0.550578 -0.027171 -0.024921  0.073605    3.12
14170 -0.601295  0.040404  0.995502 -0.273743  1.688136  0.527831   99.99
14197 -0.680052  0.108176  1.066878 -0.233720  1.707521  0.511423   99.99
14211 -0.726839  0.112540  1.119193 -0.233189  1.684063  0.503740   99.99
14338 -0.112376 -0.592077  0.520791  0.043354  0.015159  0.063612    3.76

[553 rows x 30 columns]
```

```
print(Y)
```

```
5111     0.0
7770     0.0
6116     0.0
13185    0.0
10090    0.0
        ...
14104    1.0
14170    1.0
14197    1.0
14211    1.0
14338    1.0
Name: Class, Length: 553, 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)
```

```
(553, 30) (442, 30) (111, 30)
```

Model Training

Logistic Regression

```
model = LogisticRegression()
```

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

```
l/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converg
AL NO. of ITERATIONS REACHED LIMIT.

the number of iterations (max_iter) or scale the data as shown in:
://scikit-learn.org/stable/modules/preprocessing.html
so refer to the documentation for alternative solver options:
://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
i = _check_optimize_result(
cRegression
Regression()
```

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 :  0.9909502262443439
```

```
# 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 :  0.972972972972973
```