


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/credit_data.csv')

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



	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13
0	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	-0.99131
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.48901
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.71721
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.50771
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.34581

credit\_card\_data.tail()

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	-1.593105	2.71194	
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	-0.150189	0.91580	
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	0.411614	0.06311	
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	-1.933849	-0.96288	
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	-1.040458	-0.03151	

```
# dataset informations
credit_card_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
#   Column      Non-Null Count  Dtype
---  -
0    Time      284807 non-null  float64
1    V1         284807 non-null  float64
2    V2         284807 non-null  float64
3    V3         284807 non-null  float64
4    V4         284807 non-null  float64
5    V5         284807 non-null  float64
6    V6         284807 non-null  float64
7    V7         284807 non-null  float64
8    V8         284807 non-null  float64
9    V9         284807 non-null  float64
10   V10        284807 non-null  float64
11   V11        284807 non-null  float64
12   V12        284807 non-null  float64
13   V13        284807 non-null  float64
14   V14        284807 non-null  float64
15   V15        284807 non-null  float64
16   V16        284807 non-null  float64
17   V17        284807 non-null  float64
18   V18        284807 non-null  float64
19   V19        284807 non-null  float64
20   V20        284807 non-null  float64
21   V21        284807 non-null  float64
22   V22        284807 non-null  float64
23   V23        284807 non-null  float64
24   V24        284807 non-null  float64
25   V25        284807 non-null  float64
26   V26        284807 non-null  float64
```

```

27 V27      284807 non-null float64
28 V28      284807 non-null float64
29 Amount   284807 non-null float64
30 Class    284807 non-null int64
dtypes: float64(30), int64(1)
memory usage: 67.4 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       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64

```

```

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

```

```

0      284315
1         492
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)

```

```

(284315, 31)
(492, 31)

```

```

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

```

```

count    284315.000000
mean       88.291022
std       250.105092
min         0.000000
25%        5.650000
50%       22.000000
75%       77.050000

```

```
max      25691.160000
Name: Amount, dtype: float64

fraud.Amount.describe()

count      492.000000
mean       122.211321
std        256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%        105.890000
max       2125.870000
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	V10	V11	
Class													
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	0.009824	-0.006576	0.011
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	-5.676883	3.800173	-6.25

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
203131	134666.0	-1.220220	-1.729458	-1.118957	-0.266099	0.823338	-0.098556	-0.407751
95383	65279.0	-1.295124	0.157326	1.544771	-2.468209	-1.683113	-0.623764	-0.371798
99706	67246.0	-1.481168	1.226490	1.857550	2.980777	-0.672645	0.581449	-0.143172
153895	100541.0	-0.181013	1.395877	1.204669	4.349279	1.330126	1.277520	1.568221
249976	154664.0	0.475977	-0.573662	0.480520	-2.524647	-0.616284	-0.361317	-0.347861

```
new_dataset.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050

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

1      492
0      492
```

```
Name: Class, dtype: int64
```

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

	Time	V1	V2	V3	V4	V5	V6	V
Class								
0	96783.638211	-0.053037	0.055150	-0.036786	-0.046439	0.077614	-0.023218	-0.00070
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.56873

### Splitting the data into Features & Targets

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

```
print(X)
```

	Time	V1	V2	...	V27	V28	Amount
203131	134666.0	-1.220220	-1.729458	...	0.173995	-0.023852	155.00
95383	65279.0	-1.295124	0.157326	...	0.317321	0.105345	70.00
99706	67246.0	-1.481168	1.226490	...	-0.546577	0.076538	40.14
153895	100541.0	-0.181013	1.395877	...	-0.229857	-0.329608	137.04
249976	154664.0	0.475977	-0.573662	...	0.058961	0.012816	19.60
...	...	...	...	...	...	...	...
279863	169142.0	-1.927883	1.125653	...	0.292680	0.147968	390.00
280143	169347.0	1.378559	1.289381	...	0.389152	0.186637	0.76
280149	169351.0	-0.676143	1.126366	...	0.385107	0.194361	77.89
281144	169966.0	-3.113832	0.585864	...	0.884876	-0.253700	245.00
281674	170348.0	1.991976	0.158476	...	0.002988	-0.015309	42.53

```
[984 rows x 30 columns]
```

```
print(Y)
```

```
203131    0
95383     0
99706     0
153895     0
249976     0
...
279863     1
280143     1
280149     1
281144     1
281674     1
Name: Class, Length: 984, dtype: int64
```

### 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)
```

```
(984, 30) (787, 30) (197, 30)
```

### Model Training

#### Logistic Regression

```
model = LogisticRegression()
```

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

```
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

### 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 : 0.9415501905972046
```

```
# 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 : 0.9390862944162437
```