

CASE :-

We are going to predict whether a credit card transaction is Legit (Legal) or Fraud.

WORK FLOW :-

- Credit card data
- Data Pre Processing
- Data Analysis
- Train Test Split
- Logistic Regression Model
- Model Evaluation
- Predictive System

Importing necessary libraries

```
In [1]: ## Importing the necessary Libraries

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

Importing the dataset and viewing it

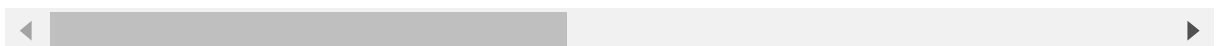
```
In [2]: ## Importing the dataset

data = pd.read_csv(r"C:\Users\lenovo\Desktop\Credit Card.csv")
data.head()
```

Out[2]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	C
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-C
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	C

5 rows × 31 columns



Getting some additional information about the DataSetIn [3]: *## Data information*

data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1175 entries, 0 to 1174

Data columns (total 31 columns):

#	Column	Non-Null Count	Dtype
0	Time	1175 non-null	int64
1	V1	1175 non-null	float64
2	V2	1175 non-null	float64
3	V3	1175 non-null	float64
4	V4	1175 non-null	float64
5	V5	1175 non-null	float64
6	V6	1175 non-null	float64
7	V7	1175 non-null	float64
8	V8	1175 non-null	float64
9	V9	1175 non-null	float64
10	V10	1175 non-null	float64
11	V11	1175 non-null	float64
12	V12	1175 non-null	float64
13	V13	1175 non-null	float64
14	V14	1175 non-null	float64
15	V15	1175 non-null	float64
16	V16	1175 non-null	float64
17	V17	1175 non-null	float64
18	V18	1175 non-null	float64
19	V19	1175 non-null	float64
20	V20	1175 non-null	float64
21	V21	1175 non-null	float64
22	V22	1175 non-null	float64
23	V23	1175 non-null	float64
24	V24	1175 non-null	float64
25	V25	1175 non-null	float64
26	V26	1175 non-null	float64
27	V27	1175 non-null	float64
28	V28	1175 non-null	float64
29	Amount	1175 non-null	float64
30	Class	1175 non-null	int64

dtypes: float64(29), int64(2)

memory usage: 284.7 KB

```
In [4]: ## Checking missing values in each column  
data.isnull().sum()
```

```
Out[4]: 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
```

```
In [5]: ## Checking distribution of class column  
data['Class'].value_counts()
```

```
Out[5]: 0    1173  
1         2  
Name: Class, dtype: int64
```

This is highly imbalanced data set :-

- 0 --> Legit Transaction
- 1 --> Fraudulent Transaction

In [6]: *## Separating the data for analysis*

```
legit = data[data.Class == 0]
fraud = data[data.Class == 1]
```

In [7]: *## Checking the shape of the DataSet*

```
print(legit.shape)
print(fraud.shape)
```

(1173, 31)

(2, 31)

In [8]: *## Statistical measure of data*

Checking the Description of the DataSet for Legit Transactions

```
legit.Amount.describe()
```

Out[8]:

count	1173.000000
mean	65.064510
std	181.271328
min	0.000000
25%	5.310000
50%	15.380000
75%	55.450000
max	3828.040000

Name: Amount, dtype: float64

In [9]: *## Statistical measure of data*

Checking the Description of the DataSet for Fraudulent Transactions

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

Out[9]:

count	2.000000
mean	264.500000
std	374.059487
min	0.000000
25%	132.250000
50%	264.500000
75%	396.750000
max	529.000000

Name: Amount, dtype: float64

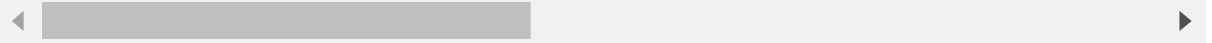
In [10]: *## Compare the values for both transaction on basic of mean*

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

Out[10]:

	Time	V1	V2	V3	V4	V5	V6	V7	
Class									
0	440.514919	-0.191290	0.240299	0.879805	0.245752	-0.028989	0.127834	0.105288	-0.
1	439.000000	-2.677884	-0.602658	-0.260694	3.143275	0.418809	-1.245684	-1.105907	0.

2 rows × 30 columns



Under - Sampling : (For imbalanced data)

- Build a sample dataset containing similar distribution of Legit Transaction & Fraud Transaction. We are going to take 492 Random Transactions from Legit Transactions then we are going to join them with Fraud Transactions. We will have 492 Legit Transaction & 492 Fraud Transaction. It will have uniform distribution & give better predictions.
- Fraud Transactions --> 492

In [11]: `legit_sample = legit.sample(n = 492)`

Concanating 2 DataFrames (Joining)

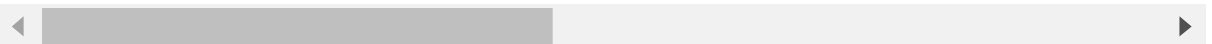
In [12]: *## Joining two dataframe and checking it*

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

Out[12]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
948	718	0.325401	-2.509865	0.614651	-0.174645	-1.747268	1.067329	-0.857323	0.432395
559	417	-2.680348	1.872052	1.144712	-0.693664	0.155172	0.601325	0.904201	-0.520079
273	194	-1.131517	1.016399	0.735810	1.166614	0.790236	-1.187196	0.736469	-0.327992
1003	758	1.195191	0.164982	0.608915	0.653734	-0.511610	-0.725919	-0.055123	-0.049900
111	73	1.148187	0.085837	0.120702	1.126665	0.214711	0.537381	-0.049989	0.186175

5 rows × 31 columns



In [13]: *## Checking total values of data*

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

Out[13]:

```
0    492
1     2
Name: Class, dtype: int64
```

In [14]: *## Checking whether we got a good sample or bad sample, in case if we got a bad sample*

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

Out[14]:

	Time	V1	V2	V3	V4	V5	V6	V7
Class								
0	454.817073	-0.204733	0.274865	0.906965	0.294621	-0.016292	0.120708	0.104506
1	439.000000	-2.677884	-0.602658	-0.260694	3.143275	0.418809	-1.245684	-1.105907

2 rows × 30 columns

From above distribution of class values by comparing with previous values we can say that nature of dataset have not changed & the difference is still there & our model will predict with good accuracy

Model Building :-

Splitting the Data into Features & Target

In [15]: *## Assigning the values to x and y variable for model building*

```
x = new_data.drop(columns = 'Class', axis =1)
y = new_data['Class']
```

```
In [16]: print(x)
```

	Time	V1	V2	V3	V4	V5	V6	\
948	718	0.325401	-2.509865	0.614651	-0.174645	-1.747268	1.067329	
559	417	-2.680348	1.872052	1.144712	-0.693664	0.155172	0.601325	
273	194	-1.131517	1.016399	0.735810	1.166614	0.790236	-1.187196	
1003	758	1.195191	0.164982	0.608915	0.653734	-0.511610	-0.725919	
111	73	1.148187	0.085837	0.120702	1.126665	0.214711	0.537381	
...	
502	369	0.953918	-0.760595	1.091611	0.147115	-0.729796	1.430148	
970	735	-2.647397	2.245069	2.369673	2.293448	-0.844090	0.291524	
136	84	-0.792329	-0.840664	2.610465	-2.196338	-0.396962	-0.707363	
541	406	-2.312227	1.951992	-1.609851	3.997906	-0.522188	-1.426545	
623	472	-3.043541	-3.157307	1.088463	2.288644	1.359805	-1.064823	
	V7	V8	V9	...	V20	V21	V22	\
948	-0.857323	0.432395	0.010094	...	0.844358	0.372382	0.178948	
559	0.904201	-0.520079	3.013065	...	1.483877	-0.459592	0.485421	
273	0.736469	-0.327992	-0.555549	...	-0.265971	0.009613	0.315739	
1003	-0.055123	-0.049900	0.119843	...	-0.179902	-0.216482	-0.623974	
111	-0.049989	0.186175	0.111781	...	-0.148395	-0.091112	-0.095475	
...	
502	-1.070470	0.727965	1.432734	...	-0.290093	-0.098744	-0.015598	
970	0.008385	-0.000207	1.321319	...	1.355069	-0.214398	0.643305	
136	-0.057934	-0.548676	-1.870127	...	-0.046317	-0.525187	-0.776454	
541	-2.537387	1.391657	-2.770089	...	0.126911	0.517232	-0.035049	
623	0.325574	-0.067794	-0.270953	...	2.102339	0.661696	0.435477	
	V23	V24	V25	V26	V27	V28	Amount	
948	-0.332993	-0.256952	-0.006418	-0.247253	-0.011687	0.085306	467.74	
559	-0.365437	-0.744118	0.328655	0.457695	0.566152	0.168241	29.99	
273	0.054210	0.294232	0.003877	-0.314159	-0.099512	0.122697	1.00	
1003	0.215072	0.376696	0.078802	0.106358	-0.016346	0.017350	0.99	
111	-0.166750	-0.653433	0.713020	-0.288035	0.031507	0.000372	19.77	
...	
502	0.248461	-0.596979	-0.291046	1.053935	0.021906	0.003337	32.63	
970	-0.142705	0.336224	0.431444	0.411999	1.012232	0.374644	10.38	
136	-0.132811	0.376664	0.221800	-0.574396	-0.419351	-0.277906	21.97	
541	-0.465211	0.320198	0.044519	0.177840	0.261145	-0.143276	0.00	
623	1.375966	-0.293803	0.279798	-0.145362	-0.252773	0.035764	529.00	

[494 rows x 30 columns]

In [17]: `print(y)`

```
948      0
559      0
273      0
1003     0
111      0
..
502      0
970      0
136      0
541      1
623      1
Name: Class, Length: 494, dtype: int64
```

Splitting the Data into Train & Test

In [18]: `x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, str`

In [19]: `print(x.shape, x_train.shape, x_test.shape)`

```
(494, 30) (395, 30) (99, 30)
```

Model Training

- Logistic Regression

In [20]: `## Using the Logistic Regression Model`
`model = LogisticRegression()`

In [21]: `## Training the logistic regression model with train data`

```
model.fit(x_train, y_train)
```

```
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:444: 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> (<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 (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
`n_iter_i = _check_optimize_result(`

Out[21]:

```
▼ LogisticRegression
LogisticRegression()
```


Model Evaluation :-

In [22]: *## Checking the Accuracy score on training data*

```
x_train_predict = model.predict(x_train)
train_data_accuracy = accuracy_score(x_train_predict, y_train)
print('Accuracy Score on Training Data :', train_data_accuracy)
```

Accuracy Score on Training Data : 1.0

In [23]: *## Checking the Accuracy score on testing data*

```
x_test_predict = model.predict(x_test)
test_data_accuracy = accuracy_score(x_test_predict, y_test)
print('Accuracy Score on Testing Data :', test_data_accuracy)
```

Accuracy Score on Testing Data : 1.0

CONCLUSION :-

Accuracy score of our model is very good & our model is not underfitted/overfitted. We can use this model for Prediction.

Predictive System :-

In [24]: `input_data = (1,-0.966271711572087,-0.185226008082898,1.79299333957872,-0.8632`

```
# Changing the input data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshaping the array for one sample
input_data_reshape = input_data_as_numpy_array.reshape(1,-1)

prediction = model.predict(input_data_reshape)
print(prediction)

if (prediction[0] == 0):
    print('The Transaction is Legit')
else:
    print('The Transaction is Fraud')
```

[0]
The Transaction is Legit

C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
warnings.warn(

In this way we can conclude that the Transaction is Legit on the basic of our predictive Model.

- - - - - X X X X X X X X - - - -
- - - -