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

```
## 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

```
## Importing the dataset

data = pd.read_csv(r"C:\Users\ASUS\Downloads\Credit Card.csv")
data.head()

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

         V8        V9  ...       V21       V22       V23       V24
V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928
0.128539
```

```
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846
0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -
0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575
0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -
0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

Getting some additional information about the DataSet

```
## 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
```

```python
## Checking missing values in each column
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
```

```python
## Checking distribution of class column

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

```
0    1173
1       2
Name: Class, dtype: int64
```

This is highly imbalanced data set :-

- 0 --> Legit Transaction

- 1 --> Fraudulent Transaction

```python
## Separating the data for analysis

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

## Checking the shape of the DataSet
print(legit.shape)
print(fraud.shape)
```

```
(1173, 31)
(2, 31)
```

```python
## Statistical measure of data
## Checking the Description  of the DataSet for Legit Transactions

legit.Amount.describe()
```

```
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
```

```python
## Statistical measure of data
## Checking the Description  of the DataSet for Fraudalent 
Transactions

fraud.Amount.describe()
```

```
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
```

```
## Compare the values for both transaction on basic of mean

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

```
             Time        V1        V2        V3        V4        V5
V6  \
Class

0      440.514919 -0.191290  0.240299  0.879805  0.245752 -0.028989
0.127834
1      439.000000 -2.677884 -0.602658 -0.260694  3.143275  0.418809 -
1.245684

             V7        V8        V9  ...       V20       V21       V22
\
Class                                ...

0      0.105288 -0.070509  0.005240  ...  0.063227 -0.005717 -0.120199

1     -1.105907  0.661932 -1.520521  ...  1.114625  0.589464  0.200214


            V23       V24       V25       V26       V27       V28
Amount
Class

0     -0.046657  0.004919  0.116965  0.029516  0.014069 -0.015882
65.06451
1      0.455377  0.013198  0.162159  0.016239  0.004186 -0.053756
264.50000

[2 rows x 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

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

Concanating 2 DataFrames (Joining)

```
## Joining two dataframe and checking it
new_data = pd.concat([legit_sample, fraud], axis = 0)
new_data.head()
```

```
      Time        V1        V2        V3        V4        V5        V6  \
1048   792 -0.735386  0.647026  1.730371  0.536997  0.256815 -0.231825   
358    265 -0.293839 -0.044369  1.093146 -1.576473 -0.107492 -0.791217   
361    265  0.073631  1.051207 -0.281223  0.853749  1.065966  1.219197   
724    548 -1.233426 -0.212441  1.839632 -1.802986 -0.493195  0.350424   
791    602 -1.108292 -0.770162  2.759309  0.089810 -0.171879  0.093366   

            V7        V8        V9  ...       V21       V22       V23  \
1048  0.540271  0.011373 -0.433969  ... -0.088287 -0.169437  0.294318   
358   0.291465 -0.093164 -1.406366  ... -0.235571 -0.286207  0.069303   
361  -1.225597 -2.262214 -0.584441  ... -1.150128  0.870673 -0.266733   
724  -0.905316  0.844863 -1.523517  ...  0.668410  1.574750 -0.176482   
791   0.569612 -0.756861  1.559096  ... -0.387886 -0.010135 -0.292322   

           V24       V25       V26       V27       V28  Amount  Class  
1048  0.051337 -0.112973 -0.637161  0.000513  0.001356   25.54      0  
358   0.107632 -0.385142  0.866055 -0.017603  0.039893   24.84      0  
361  -1.048732  0.232705 -0.262463  0.187976  0.231428    1.00      0  
724  -0.221561 -0.058504 -0.163971  0.014670 -0.033210   24.99      0  
791   0.491111 -0.771185  0.873655 -0.839738 -0.665924  120.02      0  

[5 rows x 31 columns]
```

```
## Checking total values of data

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

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

## Checking whether we got a good sample or bad sample, in case if we got a bad sample then the mean values will be very different

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

             Time        V1        V2        V3        V4        V5
V6  \
Class

0      449.422764 -0.243323  0.241207  0.879793  0.239239  0.001162
0.107653
1      439.000000 -2.677884 -0.602658 -0.260694  3.143275  0.418809 -
1.245684

             V7        V8        V9  ...       V20       V21       V22
\
Class                                ...

0      0.108289 -0.099954  0.042197  ...  0.073888 -0.025809 -0.093615

1     -1.105907  0.661932 -1.520521  ...  1.114625  0.589464  0.200214


            V23       V24       V25       V26       V27       V28
Amount
Class

0     -0.045121  0.007348  0.105741  0.056640  0.009067 -0.028192
63.301768
1      0.455377  0.013198  0.162159  0.016239  0.004186 -0.053756
264.500000

[2 rows x 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

```
## Assigning the values to x and y variable for model building
x = new_data.drop(columns = 'Class', axis =1)
y = new_data['Class']
```

```
print(x)
```

```
      Time        V1        V2        V3        V4        V5        V6  \
1048   792 -0.735386  0.647026  1.730371  0.536997  0.256815 -0.231825
358    265 -0.293839 -0.044369  1.093146 -1.576473 -0.107492 -0.791217
361    265  0.073631  1.051207 -0.281223  0.853749  1.065966  1.219197
724    548 -1.233426 -0.212441  1.839632 -1.802986 -0.493195  0.350424
791    602 -1.108292 -0.770162  2.759309  0.089810 -0.171879  0.093366
...    ...       ...       ...       ...       ...       ...       ...
383    282 -0.356466  0.725418  1.971749  0.831343  0.369681 -0.107776
471    346  1.077079  0.284980  0.007731  1.657073  0.052020  0.446389
877    665  1.270835 -0.839493  0.407857 -0.388279 -1.279813 -0.829868
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  \
1048  0.540271  0.011373 -0.433969  ... -0.105872 -0.088287 -0.169437
358   0.291465 -0.093164 -1.406366  ... -0.451588 -0.235571 -0.286207
361  -1.225597 -2.262214 -0.584441  ...  0.420519 -1.150128  0.870673
724  -0.905316  0.844863 -1.523517  ...  0.108815  0.668410  1.574750
791   0.569612 -0.756861  1.559096  ... -0.143261 -0.387886 -0.010135
...        ...       ...       ...  ...       ...       ...       ...
383   0.751610 -0.120166 -0.420675  ... -0.133602  0.020804  0.424312
471  -0.407036  0.355704  0.626039  ... -0.142799 -0.174337 -0.174161
877  -0.549670 -0.116525 -0.264824  ...  0.095461  0.002550 -0.102115
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
1048  0.294318  0.051337 -0.112973 -0.637161  0.000513  0.001356
25.54
358   0.069303  0.107632 -0.385142  0.866055 -0.017603  0.039893
24.84
361  -0.266733 -1.048732  0.232705 -0.262463  0.187976  0.231428
1.00
724  -0.176482 -0.221561 -0.058504 -0.163971  0.014670 -0.033210
24.99
791  -0.292322  0.491111 -0.771185  0.873655 -0.839738 -0.665924
120.02
...        ...       ...       ...       ...       ...       ...      .
..
383  -0.015989  0.466754 -0.809962  0.657334 -0.043150 -0.046401
0.00
471  -0.153375 -0.466331  0.611001 -0.252871  0.090375  0.054820
10.99
877  -0.074715  0.376595  0.493224 -0.246441 -0.005995  0.021517
73.00
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]
```

```
print(y)
```

```
1048    0
358     0
361     0
724     0
791     0
       ..
383     0
471     0
877     0
541     1
623     1
Name: Class, Length: 494, dtype: int64
```

Splitting the Data into Train & Test

```
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)
```

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

Model Training

- Logistic Regression

```
## Using the Logistic Regression Model
model = LogisticRegression()

## Training the logistic regression model with train data

model.fit(x_train, y_train)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\
_logistic.py:814: 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()
```

Model Evaluation :-

```
## 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

## 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 : 0.98989898989899
```

CONCLUSION :-

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

Predictive System :-

```
input_data = (1,-0.966271711572087,-
0.185226008082898,1.79299333957872,-0.863291275036453,-
0.0103088796030823,1.24720316752486,0.23760893977178,0.377435874652262
,-1.38702406270197,-0.0549519224713749,-
0.226487263835401,0.178228225877303,0.507756869957169,-
0.28792374549456,-0.631418117709045,-1.0596472454325,-
0.684092786345479,1.96577500349538,-1.2326219700892,-
0.208037781160366,-0.108300452035545,0.00527359678253453,-
0.190320518742841,-1.17557533186321,0.647376034602038,-
0.221928844458407,0.0627228487293033,0.0614576285006353,123.5)

# 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:\ProgramData\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 - - - - -
- - -
```