IMPORT ALL DEPENDENCIES

In [1]:
```
import numpy as np #Useful for making Arrays
import pandas as pd # Useful for CSV file and it works with dataframe
from sklearn.model_selection import train_test_split #Useful for training and testing data
from sklearn.linear_model import LogisticRegression #Useful for checking accuracy of the model
from sklearn.metrics import accuracy_score # Useful for checking the performance of the model
```

Loading dataset of CSV file using pandas function

In [2]:
```
credit_card_data=pd.read_csv(r"G:\Project 4 Python\Credit Card Fraud detection\creditcard.csv'
```

first 5 rows of the dataset

In [3]:
```
credit_card_data.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-(
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-(
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	(
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-(
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-(

5 rows × 31 columns

Last 5 rows of the dataset

In [4]:
```
credit_card_data.tail()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486

5 rows × 31 columns

dataset informations

In [5]:
```
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

In [11]:
```
credit_card_data.isnull().sum()
```

Out[11]:
```
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
```

Distribuation of legit and fradulant transaction

In [14]:
```
credit_card_data['Class'].value_counts()
```

Out[14]:
```
0    284315
1       492
Name: Class, dtype: int64
```

This Dataset is highly unblanced

0 --> Normal Transaction

1 --> fraudulent transaction

separating the data for analysis

In [15]:
```
legit=credit_card_data[credit_card_data.Class==0]
fraud=credit_card_data[credit_card_data.Class==1]
```

In [16]:
```
print(legit.shape)
print(fraud.shape)
```

```
(284315, 31)
(492, 31)
```

statistical measures of the data

In [17]:
```
credit_card_data.describe()
```

Out[17]:

	Time	V1	V2	V3	V4	V5	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15	2.040130e-
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+

8 rows × 31 columns

In [19]:
```
legit.Amount.describe()
```

Out[19]:
```
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
```

In [18]:
```
fraud.Amount.describe()
```

Out[18]:
```
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

In [20]:
```
credit_card_data.groupby('Class').mean()
```

Out[20]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
Class										
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.0(
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.58

2 rows × 30 columns

Under-Sampling

Build a sample dataset containing similar distribution of normal transactions and Fraudulent Transactions

Number of Fraudulent Transactions --> 492

In [35]:
```
legit_sample=legit.sample(n=600)
```

Concatenating two DataFrames

In [36]:
```
new_dataset = pd.concat([legit_sample,fraud],axis=0)
```

In [37]:
```
new_dataset.head()
```

Out[37]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V
4437	3769.0	1.381243	-0.717063	-0.926771	-1.607880	1.448006	3.278493	-1.156568	0.707654	0.3588
245645	152843.0	0.254035	-0.599243	0.097654	-2.872652	-0.195793	0.093498	-0.234849	0.077571	-2.0705
230619	146410.0	1.758375	-0.830913	-0.100479	0.493096	-0.907640	0.135050	-0.858581	0.218170	1.1352
227055	144918.0	1.582516	-0.824635	-0.004023	1.524511	-1.034557	-0.182077	-0.547553	0.057638	1.2233
182479	125415.0	1.986014	0.045096	-1.679146	0.449706	0.083987	-1.192513	0.081583	-0.158578	0.5150

5 rows × 31 columns

In [38]:
```
new_dataset.tail()
```

Out[38]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.06494
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.12739
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.65225
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.63233
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.57782

5 rows × 31 columns

In [39]:
```
new_dataset['Class'].value_counts()
```

Out[39]:
```
0    600
1    492
Name: Class, dtype: int64
```

In [40]:
```
new_dataset.groupby('Class').mean()
```

Out[40]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
Class										
0	95831.173333	0.062597	0.028971	0.040787	0.034962	-0.000162	0.062980	0.099546	0.029312	-0.0525
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.5811

2 rows × 30 columns

Splitting the data into Features & Targets

In [41]:
```
X= new_dataset.drop(columns='Class',axis=1)
Y = new_dataset['Class']
```

In [29]:
```
print(X)
```

```
                Time        V1        V2        V3        V4        V5        V6  \
32158    36664.0 -1.716916 -0.685866  0.547017  0.248903 -0.501067  1.196848
41352    40646.0 -1.176898  1.238675  2.256913  0.662897 -0.862182  0.450617
284297  172312.0  1.944069 -0.241760 -1.441286  0.155229  0.802930  0.811877
191806  129415.0  0.618606  0.786401 -2.672958  0.020681  3.075941  3.629842
106015   69782.0  1.218326  0.050756  0.079325 -0.017878 -0.345180 -0.997185
...          ...       ...       ...       ...       ...       ...       ...
279863  169142.0 -1.927883  1.125653 -4.518331  1.749293 -1.566487 -2.010494
280143  169347.0  1.378559  1.289381 -5.004247  1.411850  0.442581 -1.326536
280149  169351.0 -0.676143  1.126366 -2.213700  0.468308 -1.120541 -0.003346
281144  169966.0 -3.113832  0.585864 -5.399730  1.817092 -0.840618 -2.943548
281674  170348.0  1.991976  0.158476 -2.583441  0.408670  1.151147 -0.096695

              V7        V8        V9  ...       V20       V21       V22  \
32158   2.556734  0.167478 -0.681876  ...  0.881448  0.332741 -0.058212
41352   0.066160  0.296521 -0.022682  ...  0.196463  0.244123  0.911383
284297 -0.106648  0.218693  0.589631  ... -0.311713 -0.176625 -0.263084
191806  0.062101 -1.024825 -0.704039  ... -0.388292  1.727217  0.053016
106015  0.205235 -0.143924 -0.247635  ... -0.056691 -0.386430 -1.275435
...          ...       ...       ...  ...       ...       ...       ...
279863 -0.882850  0.697211 -2.064945  ...  1.252967  0.778584 -0.319189
280143 -1.413170  0.248525 -1.127396  ...  0.226138  0.370612  0.028234
280149 -2.234739  1.210158 -0.652250  ...  0.247968  0.751826  0.834108
281144 -2.208002  1.058733 -1.632333  ...  0.306271  0.583276 -0.269209
281674  0.223050 -0.068384  0.577829  ... -0.017652 -0.164350 -0.295135

             V23       V24       V25       V26       V27       V28  Amount
32158   0.733897 -0.930729  0.996298 -0.290633 -0.165209  0.068558  530.00
41352  -0.402904 -0.096735  0.340112 -0.123942 -0.132138 -0.158732   70.00
284297  0.403805 -0.337542 -0.399494  0.344410 -0.025605 -0.069121    1.18
191806 -0.090547  0.707946  0.247061 -0.389554  0.397034  0.341352   59.70
106015  0.185119  0.336071  0.044891  0.604388 -0.112236 -0.004058   27.50
...          ...       ...       ...       ...       ...       ...     ...
279863  0.639419 -0.294885  0.537503  0.788395  0.292680  0.147968  390.00
280143 -0.145640 -0.081049  0.521875  0.739467  0.389152  0.186637    0.76
280149  0.190944  0.032070 -0.739695  0.471111  0.385107  0.194361   77.89
281144 -0.456108 -0.183659 -0.328168  0.606116  0.884876 -0.253700  245.00
281674 -0.072173 -0.450261  0.313267 -0.289617  0.002988 -0.015309   42.53

[984 rows x 30 columns]
```

In [42]:
```python
print(Y)
```

```
4437      0
245645    0
230619    0
227055    0
182479    0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 1092, dtype: int64
```

Split the data into Training data & Testing Data

In [43]:
```python
X_train,X_test,Y_train,Y_test = train_test_split(X,Y, test_size=0.2, stratify=Y, random_state=
```

In [44]:
```python
print(X.shape,X_train.shape,X_test.shape)
```

```
(1092, 30) (873, 30) (219, 30)
```

In [45]:
```python
print(Y.shape,Y_train.shape,Y_test.shape)
```

```
(1092,) (873,) (219,)
```

Model Training || Logistic Regression

In [46]:
```
model= LogisticRegression()
```

Training the logistic regression model with training data

In [49]:
```
model.fit(X_train, Y_train)
```

Out[49]:
```
LogisticRegression()
```

In [50]:
```
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

In [51]:
```
print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.9186712485681557
```

In [52]:
```
# accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

In [53]:
```
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.9269406392694064
```

In [ ]:
```

```