

NAME: VANDHITHA R S

EMAIL: vandithachinnu77777@gmail.com
(<mailto:vandithachinnu77777@gmail.com>)

In [1]:

```
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
```

In [2]:

```
# Loading the dataset to a Pandas DataFrame
credit_card_data = pd.read_csv('creditcard.csv')
```

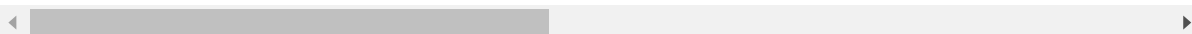
In [3]:

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

Out[3]:

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

5 rows × 31 columns



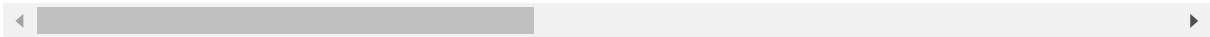
In [4]:

```
credit_card_data.tail()
```

Out[4]:

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

5 rows × 31 columns



In [5]:

```
# dataset information
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
```

In [6]:

```
# checking the number of missing values in each column
credit_card_data.isnull().sum()
```

Out[6]:

```
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 [7]:

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

Out[7]:

```
0    284315
1      492
Name: Class, dtype: int64
```

In [8]:

```
# separating the data for analysis
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

In [9]:

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

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

In [10]:

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

Out[10]:

```
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 [11]:

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

Out[11]:

```
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
```

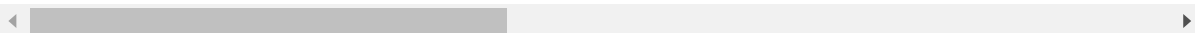
In [12]:

```
# compare the values for both transactions
credit_card_data.groupby('Class').mean()
```

Out[12]:

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

2 rows × 30 columns



In [13]:

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

In [14]:

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

In [15]:

```
new_dataset.head()
```

Out[15]:

	Time	V1	V2	V3	V4	V5	V6	V7	
237032	149065.0	-3.869184	4.119667	-3.275938	-2.319375	0.790778	-0.963375	1.152998	0.
248956	154172.0	0.439804	1.343752	-1.364688	1.309440	0.340152	-0.412269	0.200711	0.
192447	129696.0	1.657460	-0.793798	-1.815345	0.190633	0.274968	-0.152264	0.325799	-0.
160701	113561.0	-0.781717	0.931177	1.868558	0.633489	0.744240	1.285919	0.954177	-0.
180208	124431.0	-0.641807	0.950912	-1.093181	-1.313436	0.148106	1.010542	2.916784	-0.

5 rows × 31 columns

In [16]:

```
new_dataset.tail()
```

Out[16]:

	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	0.
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.

5 rows × 31 columns

In [17]:

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

Out[17]:

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

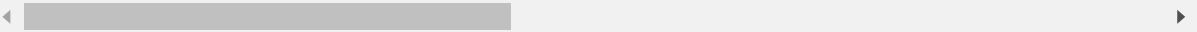
In [18]:

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

Out[18]:

	Time	V1	V2	V3	V4	V5	V6	V7
Class								
0	92014.520325	-0.058161	0.136871	0.033326	0.011810	-0.046360	-0.024287	-0.006837
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731

2 rows × 30 columns



In [19]:

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

In [20]:

print(X)

	Time	V1	V2	V3	V4	V5	V6
237032	149065.0	-3.869184	4.119667	-3.275938	-2.319375	0.790778	-0.963375
248956	154172.0	0.439804	1.343752	-1.364688	1.309440	0.340152	-0.412269
192447	129696.0	1.657460	-0.793798	-1.815345	0.190633	0.274968	-0.152264
160701	113561.0	-0.781717	0.931177	1.868558	0.633489	0.744240	1.285919
180208	124431.0	-0.641807	0.950912	-1.093181	-1.313436	0.148106	1.010542
...
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	\
237032	1.152998	0.278303	2.183822	...	1.986920	-0.214113	0.490138	
248956	0.200711	0.159247	-0.200281	...	0.049756	-0.106938	-0.260220	
192447	0.325799	-0.080752	0.520302	...	0.185542	-0.122988	-0.841470	
160701	0.954177	-0.010412	0.280441	...	0.454631	-0.683470	-1.263217	
180208	2.916784	-0.502763	-0.460707	...	-0.186647	-0.019281	0.504615	
...	
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
237032	-0.198128	-1.376379	0.720342	0.224324	1.923757	1.208351	3.85
248956	0.356932	0.504631	-1.333203	-0.386510	-0.186872	0.049028	9.99
192447	0.077184	0.034016	-0.182242	-0.637951	-0.070437	-0.023619	214.94
160701	-0.244564	0.145956	0.429790	-0.854676	0.147297	-0.255731	42.58
180208	-0.330715	-0.106055	0.054687	-0.025296	-0.393897	-0.477849	341.10
...
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 [21]:

```
print(Y)
```

```
237032    0
248956    0
192447    0
160701    0
180208    0
..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

In [24]:

```
X_train, X_test, Y_train, Y_test = train_test_split (X, Y, test_size = 0.2, stratify=Y, ran
```

In [25]:

```
print(X.shape, X_train.shape, X_test.shape)
```

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

In [26]:

```
model = LogisticRegression()
```

In [30]:

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

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:763: 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[30]:

```
LogisticRegression()
```

In [28]:

```
# accuracy on training data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

In [29]:

```
print('Accuracy score on Test Data : ', test_data_accuracy)
```

Accuracy score on Test Data : 0.934010152284264