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 [3]:

```
import os
os.getcwd()
credit_card_data = pd.read_csv('creditcard.csv')
credit_card_data
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592
...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577

284807 rows × 31 columns

In [4]:

```
credit_card_data.head()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.0986
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0851
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2476
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3774
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2705

5 rows × 31 columns

In [5]:

```
# information about data set
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]:

```
#another way to check missing value in data set
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]:

```
# check the distribution of legit and fraudulent function
credit_card_data['Class'].value_counts()
```

Out[7]:

```
0    284315
```

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

In [9]:
```
#seperating data for analysis
legit = credit_card_data[credit_card_data.Class == 0]
fraud = credit_card_data[credit_card_data.Class == 1]
```

In [10]:
```
print(legit.shape)
print(fraud.shape)
```

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

In [11]:
```
#statistical measure of data
legit.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]:
```
fraud.Amount.describe()
```

Out[12]:
```
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 [13]:
```
#comparing the mean value of each transaction column wise
credit_card_data.groupby('Class').mean()
```

Out[13]:

	Time	V1	V2	V3	V4	V5	V6	
Class								
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.0096
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568

2 rows × 30 columns

In [14]:
```
#sampling from population
legit_sample = legit.sample(n=492)
```

In [16]:
```
#now concating two random sample of fraud and legit
new_dataset = pd.concat([legit_sample, fraud], axis=0)
new_dataset
```

Out[16]:

	Time	V1	V2	V3	V4	V5	V6	
179127	123933.0	-1.226099	1.263841	1.592882	-0.732320	-0.009359	0.618837	-0.1503
277998	167982.0	0.017972	0.066563	-1.474137	-2.069225	0.987035	-1.342736	0.8827
162320	115013.0	-0.673372	0.884540	-0.039556	-1.128214	1.689072	1.605549	0.4835

	Time	V1	V2	V3	V4	V5	V6	
256276	157617.0	-1.746942	0.268513	-0.819984	0.899060	2.062543	-0.477698	1.3732
80635	58580.0	-0.925514	1.023498	1.485087	-0.787077	0.508082	-0.016270	0.6986
...	...	...	...	...	...	...	...	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.8828
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.4131
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.2347
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.2080
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.2230

984 rows × 31 columns

In [17]:
```
new_dataset.head()
```

Out[17]:

	Time	V1	V2	V3	V4	V5	V6	V
179127	123933.0	-1.226099	1.263841	1.592882	-0.732320	-0.009359	0.618837	-0.15030
277998	167982.0	0.017972	0.066563	-1.474137	-2.069225	0.987035	-1.342736	0.88271
162320	115013.0	-0.673372	0.884540	-0.039556	-1.128214	1.689072	1.605549	0.48353
256276	157617.0	-1.746942	0.268513	-0.819984	0.899060	2.062543	-0.477698	1.37320
80635	58580.0	-0.925514	1.023498	1.485087	-0.787077	0.508082	-0.016270	0.69864

5 rows × 31 columns

In [19]:
```
new_dataset['Class'].value_counts()
```

Out[19]:
```
1    492
0    492
Name: Class, dtype: int64
```

In [20]:
```
#again comparing mean of each column wise
new_dataset.groupby('Class').mean()
```

Out[20]:

	Time	V1	V2	V3	V4	V5	V6	
Class								
0	93238.636179	-0.012739	-0.051224	0.081062	0.067771	-0.034524	0.046348	-0.0486
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.5687

2 rows × 30 columns

In [22]:
```
#spliting the data into feature and targets
x = new_dataset.drop(columns = 'Class', axis=1)
y = new_dataset['Class']
```

In [23]:
```
print(x)
print(y)
```

```
             Time        V1        V2        V3        V4        V5        V6
\
179127  123933.0 -1.226099  1.263841  1.592882 -0.732320 -0.009359  0.618837
277998  167982.0  0.017972  0.066563 -1.474137 -2.069225  0.987035 -1.342736
```

```
162320  115013.0 -0.673372  0.884540 -0.039556 -1.128214  1.689072  1.605549
256276  157617.0 -1.746942  0.268513 -0.819984  0.899060  2.062543 -0.477698
80635    58580.0 -0.925514  1.023498  1.485087 -0.787077  0.508082 -0.016270
...          ...       ...       ...       ...       ...       ...       ...
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  \
179127 -0.150303 -1.385052  0.152878  ... -0.293957  1.464472 -0.628600
277998  0.882777 -0.151395 -1.270524  ...  0.127353  0.390618  1.001797
162320  0.483538 -0.487899  0.218933  ... -0.219425  1.100461  1.031925
256276  1.373207  0.194511 -2.035495  ...  0.563382  0.554844  0.944406
80635   0.698643  0.186929 -0.136151  ... -0.069692 -0.285632 -0.906396
...          ...       ...       ...  ...       ...       ...       ...
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
179127 -0.231586  0.525292  0.270160  0.465720  0.152674  0.097235   60.00
277998 -0.070283  0.667252 -0.117175 -0.171167  0.278003  0.170028   47.90
162320 -0.302046 -0.975122  0.000803 -0.308771  0.182194 -0.025066    1.00
256276 -0.539891  0.297941  1.643287  0.073394 -0.146000 -0.198981  131.76
80635  -0.234127 -0.832464  0.250941  0.106850  0.101328  0.104278    4.99
...          ...       ...       ...       ...       ...       ...     ...
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]
179127    0
277998    0
162320    0
256276    0
80635     0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

In [25]:

```python
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, stra
print(x.shape, x_train.shape, x_test.shape)
```

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

In [26]:

```python
#model training
model = LogisticRegression()
```

In [27]:

```python
#training logistic regression model with training data
model.fit(x_train, y_train)
```

```
/Applications/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logi
stic.py:762: 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-regress
ion
  n_iter_i = _check_optimize_result(
```

Out[27]:
```
LogisticRegression()
```

In [31]:
```python
#accuracy on training data
# accuracy on training data
x_train_prediction = model.predict(x_train)
training_data_accuracy = accuracy_score(x_train_prediction, y_train)
```

In [32]:
```python
print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.9491740787801779
```

In [33]:
```python
#accuracy of test data
# accuracy on test data
x_test_prediction = model.predict(x_test)
test_data_accuracy = accuracy_score(x_test_prediction, y_test)
```

In [34]:
```python
print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.934010152284264
```

In [ ]:
```python
# so test data accuracy and trainning data accuracy is close to each other so
```