SONAR : Rock vs Mine detection using logestic regration

Importind some important dependies

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
```

Data collection and Data Processing

In [2]:

```
#loading the data set to a pandas dataset
sonar_data = pd.read_csv('Copy of sonar data.csv', header = None)
```

In [3]:

```
sonar_data.head()
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	...	51	52	5
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027	0.0065	0.01
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.0084	0.0089	0.004
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.0232	0.0166	0.00
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.0121	0.0036	0.01
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.0031	0.0054	0.01

5 rows × 61 columns

In [4]:

```
sonar_data.tail()
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	...	51	52	
203	0.0187	0.0346	0.0168	0.0177	0.0393	0.1630	0.2028	0.1694	0.2328	0.2684	...	0.0116	0.0098	0.
204	0.0323	0.0101	0.0298	0.0564	0.0760	0.0958	0.0990	0.1018	0.1030	0.2154	...	0.0061	0.0093	0.
205	0.0522	0.0437	0.0180	0.0292	0.0351	0.1171	0.1257	0.1178	0.1258	0.2529	...	0.0160	0.0029	0.
206	0.0303	0.0353	0.0490	0.0608	0.0167	0.1354	0.1465	0.1123	0.1945	0.2354	...	0.0086	0.0046	0.
207	0.0260	0.0363	0.0136	0.0272	0.0214	0.0338	0.0655	0.1400	0.1843	0.2354	...	0.0146	0.0129	0.

5 rows × 61 columns

In [5]:

```
#number of rows and columns
sonar_data.shape
```

Out[5]:
```
(208, 61)
```

In [6]:
```
sonar_data.describe() #describe() --> statistical data
```

Out[6]:

	0	1	2	3	4	5	6	7
count	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000
mean	0.029164	0.038437	0.043832	0.053892	0.075202	0.104570	0.121747	0.134799
std	0.022991	0.032960	0.038428	0.046528	0.055552	0.059105	0.061788	0.085152
min	0.001500	0.000600	0.001500	0.005800	0.006700	0.010200	0.003300	0.005500
25%	0.013350	0.016450	0.018950	0.024375	0.038050	0.067025	0.080900	0.080425
50%	0.022800	0.030800	0.034300	0.044050	0.062500	0.092150	0.106950	0.112100
75%	0.035550	0.047950	0.057950	0.064500	0.100275	0.134125	0.154000	0.169600
max	0.137100	0.233900	0.305900	0.426400	0.401000	0.382300	0.372900	0.459000

8 rows × 60 columns

In [7]:
```
sonar_data[60].value_counts()
```

Out[7]:
```
M    111
R     97
Name: 60, dtype: int64
```

In [8]:
```
sonar_data.groupby(60).mean()
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9
60										
M	0.034989	0.045544	0.050720	0.064768	0.086715	0.111864	0.128359	0.149832	0.213492	0.251022
R	0.022498	0.030303	0.035951	0.041447	0.062028	0.096224	0.114180	0.117596	0.137392	0.159325

2 rows × 60 columns

In [9]:
```
#sapearte data and the lables
X = sonar_data.drop(columns = 60, axis = 1)
Y = sonar_data[60]
```

In [11]:
```
X_train , X_test , Y_train , Y_test = train_test_split(X , Y ,test_size = 0.1 , stratif
```

In [25]:
```
print(X.shape , X_train.shape , X_test.shape)
```

```
(208, 60) (187, 60) (21, 60)
```

```
In [26]: print(X_train,Y_train)
```

```
          0       1       2       3       4       5       6       7       8   \
115  0.0414  0.0436  0.0447  0.0844  0.0419  0.1215  0.2002  0.1516  0.0818
38   0.0123  0.0022  0.0196  0.0206  0.0180  0.0492  0.0033  0.0398  0.0791
56   0.0152  0.0102  0.0113  0.0263  0.0097  0.0391  0.0857  0.0915  0.0949
123  0.0270  0.0163  0.0341  0.0247  0.0822  0.1256  0.1323  0.1584  0.2017
18   0.0270  0.0092  0.0145  0.0278  0.0412  0.0757  0.1026  0.1138  0.0794
..      ...     ...     ...     ...     ...     ...     ...     ...     ...
140  0.0412  0.1135  0.0518  0.0232  0.0646  0.1124  0.1787  0.2407  0.2682
5    0.0286  0.0453  0.0277  0.0174  0.0384  0.0990  0.1201  0.1833  0.2105
154  0.0117  0.0069  0.0279  0.0583  0.0915  0.1267  0.1577  0.1927  0.2361
131  0.1150  0.1163  0.0866  0.0358  0.0232  0.1267  0.2417  0.2661  0.4346
203  0.0187  0.0346  0.0168  0.0177  0.0393  0.1630  0.2028  0.1694  0.2328

          9   ...      50      51      52      53      54      55      56  \
115  0.1975  ...  0.0222  0.0045  0.0136  0.0113  0.0053  0.0165  0.0141
38   0.0475  ...  0.0149  0.0125  0.0134  0.0026  0.0038  0.0018  0.0113
56   0.1504  ...  0.0048  0.0049  0.0041  0.0036  0.0013  0.0046  0.0037
123  0.2122  ...  0.0197  0.0189  0.0204  0.0085  0.0043  0.0092  0.0138
18   0.1520  ...  0.0045  0.0084  0.0010  0.0018  0.0068  0.0039  0.0120
..      ...  ...     ...     ...     ...     ...     ...     ...     ...
140  0.2058  ...  0.0798  0.0376  0.0143  0.0272  0.0127  0.0166  0.0095
5    0.3039  ...  0.0104  0.0045  0.0014  0.0038  0.0013  0.0089  0.0057
154  0.2169  ...  0.0039  0.0053  0.0029  0.0020  0.0013  0.0029  0.0020
131  0.5378  ...  0.0228  0.0099  0.0065  0.0085  0.0166  0.0110  0.0190
203  0.2684  ...  0.0203  0.0116  0.0098  0.0199  0.0033  0.0101  0.0065

          57      58      59
115  0.0077  0.0246  0.0198
38   0.0058  0.0047  0.0071
56   0.0011  0.0034  0.0033
123  0.0094  0.0105  0.0093
18   0.0132  0.0070  0.0088
..      ...     ...     ...
140  0.0225  0.0098  0.0085
5    0.0027  0.0051  0.0062
154  0.0062  0.0026  0.0052
131  0.0141  0.0068  0.0086
203  0.0115  0.0193  0.0157

[187 rows x 60 columns] 115    M
38     R
56     R
123    M
18     R
      ..
140    M
5      R
154    M
131    M
203    M
Name: 60, Length: 187, dtype: object
```

Training Model using Logestic Regration

```
In [27]: model = LogisticRegression()
```

Training Model with training data

In [28]:

```
model.fit(X_train, Y_train)
```

Out[28]:

```
LogisticRegression()
```

Model Evaluation

In [32]:

```
#accuracy on trainig data
X_train_predict = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_predict , Y_train)
```

In [33]:

```
print("Accuracy score of trainiag data " , training_data_accuracy)
```

```
Accuracy score of trainiag data  0.8342245989304813
```

accuracy on test data

In [34]:

```
X_test_predict = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_predict , Y_test)
```

In [35]:

```
print("Accuracy of test data " , test_data_accuracy)
```

```
Accuracy of test data  0.7619047619047619
```

making a predictive system

In [50]:

```
input_data = (0.0363,0.0478,0.0298,0.0210,0.1409,0.1916,0.1349,0.1613,0.1703,0.1444,0.1

#changing input data to a numpy array
input_data_as_array = np.asarray(input_data)
#reshape the numpy array as we prediting for one instance
input_data_reshape = input_data_as_array.reshape(1,-1)
predition = model.predict(input_data_reshape)
print(predition)

if(predition[0]=='R'):
    print("THE OBJECT IS ROCK")
else:
    print("THE OBJECT IS A MINE")
```

```
['M']
THE OBJECT IS A MINE
```

In [51]:

```
input_data = (0.0453,0.0523,0.0843,0.0689,0.1183,0.2583,0.2156,0.3481,0.3337,0.2872,0.4

#changing input data to a numpy array
input_data_as_array = np.asarray(input_data)
#reshape the numpy array as we prediting for one instance
input_data_reshape = input_data_as_array.reshape(1,-1)
predition = model.predict(input_data_reshape)
print(predition)

if(predition[0]=='R'):
    print("THE OBJECT IS ROCK")
else:
    print("THE OBJECT IS A MINE")
```

```
['R']
THE OBJECT IS ROCK
```