SONAR ROCK MINE PREDICTION PROJECT

Importing required Libraries

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing data

In [2]:

```
df = pd.read_csv("Sonar Data.csv", header = None)
```

In [3]:

```
df.head()
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	...	51	52	53	54
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	0.0159	0.0072
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	0.0048	0.0094
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	0.0095	0.0180
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	0.0150	0.0085
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	0.0105	0.0110

5 rows × 61 columns

performing EDA

In [4]:

```
df.shape
```

Out[4]:

```
(208, 61)
```

In [62]:

```
df.isnull().sum()
```

Out[62]:

```
0      0
1      0
2      0
3      0
4      0
      ..
56     0
57     0
58     0
59     0
60     0
Length: 61, dtype: int64
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 208 entries, 0 to 207
```

```
Data columns (total 61 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       208 non-null    float64
 1   1       208 non-null    float64
 2   2       208 non-null    float64
 3   3       208 non-null    float64
 4   4       208 non-null    float64
 5   5       208 non-null    float64
 6   6       208 non-null    float64
 7   7       208 non-null    float64
 8   8       208 non-null    float64
 9   9       208 non-null    float64
 10  10      208 non-null    float64
 11  11      208 non-null    float64
 12  12      208 non-null    float64
 13  13      208 non-null    float64
 14  14      208 non-null    float64
 15  15      208 non-null    float64
 16  16      208 non-null    float64
 17  17      208 non-null    float64
 18  18      208 non-null    float64
 19  19      208 non-null    float64
 20  20      208 non-null    float64
 21  21      208 non-null    float64
 22  22      208 non-null    float64
 23  23      208 non-null    float64
 24  24      208 non-null    float64
 25  25      208 non-null    float64
 26  26      208 non-null    float64
 27  27      208 non-null    float64
 28  28      208 non-null    float64
 29  29      208 non-null    float64
 30  30      208 non-null    float64
 31  31      208 non-null    float64
 32  32      208 non-null    float64
 33  33      208 non-null    float64
 34  34      208 non-null    float64
 35  35      208 non-null    float64
 36  36      208 non-null    float64
 37  37      208 non-null    float64
 38  38      208 non-null    float64
 39  39      208 non-null    float64
 40  40      208 non-null    float64
 41  41      208 non-null    float64
 42  42      208 non-null    float64
 43  43      208 non-null    float64
 44  44      208 non-null    float64
 45  45      208 non-null    float64
 46  46      208 non-null    float64
 47  47      208 non-null    float64
 48  48      208 non-null    float64
 49  49      208 non-null    float64
 50  50      208 non-null    float64
 51  51      208 non-null    float64
 52  52      208 non-null    float64
 53  53      208 non-null    float64
 54  54      208 non-null    float64
 55  55      208 non-null    float64
 56  56      208 non-null    float64
 57  57      208 non-null    float64
 58  58      208 non-null    float64
 59  59      208 non-null    float64
 60  60      208 non-null    object 
dtypes: float64(60), object(1)
memory usage: 99.2+ KB
```

In [19]:

```
df.describe()
```

Out[19]:

	0	1	2	3	4	5	6	7	8
count	208.000000	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	0.178003
std	0.022991	0.032960	0.038428	0.046528	0.055552	0.059105	0.061788	0.085152	0.118387
min	0.001500	0.000600	0.001500	0.005800	0.006700	0.010200	0.003300	0.005500	0.007500
25%	0.013350	0.016450	0.018950	0.024375	0.038050	0.067025	0.080900	0.080425	0.097025
50%	0.022800	0.030800	0.034300	0.044050	0.062500	0.092150	0.106950	0.112100	0.152250
75%	0.035550	0.047950	0.057950	0.064500	0.100275	0.134125	0.154000	0.169600	0.233425
max	0.137100	0.233900	0.305900	0.426400	0.401000	0.382300	0.372900	0.459000	0.682800

8 rows × 60 columns

In [20]:

```
#Checking for the balance in the dataset
df[60].value_counts()
```

Out[20]:

```
M    111
R     97
Name: 60, dtype: int64
```

In [21]:

```
#Checkiing for outliers misbalancing data
df.groupby(60).mean()
```

Out[21]:

	0	1	2	3	4	5	6	7	8	9	...	50
60												
M	0.034989	0.045544	0.050720	0.064768	0.086715	0.111864	0.128359	0.149832	0.213492	0.251022	...	0.019352
R	0.022498	0.030303	0.035951	0.041447	0.062028	0.096224	0.114180	0.117596	0.137392	0.159325	...	0.012311

2 rows × 60 columns

Feature Selection

In [22]:

```
#Splitting up the data into attributes and target
X = df.drop(columns= 60, axis=1)
y= df[60]
```

In [23]:

```
X.shape
```

Out[23]:

```
(208, 60)
```

In [24]:

```
y.shape
```

Out[24]:

```
(208,)
```

Feature Splitting

In [25]:

```
from sklearn.model_selection import train_test_split
```

In [26]:

```
#to perform modelling we need to split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
```

In [27]:

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(166, 60)
(166,)
(42, 60)
(42,)
```

Model Building

In [28]:

```
from sklearn.linear_model import LogisticRegression
```

In [30]:

```
lr = LogisticRegression()
```

In [31]:

```
lr
```

Out[31]:

```
LogisticRegression()
```

In [33]:

```
#to check the model working we need to fit the data into the model we have created
lr.fit(X_train, y_train)
```

Out[33]:

```
LogisticRegression()
```

In [34]:

```
#For Chekcing the working of the model we built we need to import accuracy score from skle
from sklearn.metrics import accuracy_score
```

Prediction

In [35]:

```
#predicting the target for training dataset
X_train_prediction = lr.predict(X_train)
```

In [37]:

```
training_data_accuracy = accuracy_score(X_train_prediction, y_train)
```

In [42]:

```
print("The Accuracy score for the Training data is :" , round(training_data_accuracy * 10
```

```
The Accuracy score for the Training data is : 83.13 %
```

In [44]:

```
#predicting the accuracy for test dataset
X_test_prediction = lr.predict(X_test)
```

In [47]:

```
test_data_accuracy = accuracy_score(X_test_prediction, y_test)
```

In [48]:

```
print("The Accuracy score for the Test data is :" , round(test_data_accuracy * 100,2), "%"
```

```
The Accuracy score for the Test data is : 83.33 %
```

To check the model is running correct or not

In [61]:

```
input_data = (0.0200,0.0371,0.0428,0.0207,0.0954,0.0986,0.1539,0.1601,0.3109,0.2111,0.160

input_data_as_numpy_array= np.asarray(input_data)

input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction= lr.predict(input_data_reshaped)
print(prediction)

if (prediction == "R"):
    print("It is a Rock")
else:
    print("It is a Mine")
```

```
['R']
It is a Rock
```

Thus We can check out the Sonar Waves Prediction using this Model.