Sonar Rock vs Mine Prediction

Sonar Data Classification Workflow

Step 1: Load the sonar data

Step 2: Preprocess the data (cleaning, normalization, encoding if needed)

Step 3: Split the data into training and testing sets

Step 4: Train a Logistic Regression model using the training data

Step 5: Evaluate the model using the test data

Step 6: Use the trained Logistic Regression model to predict new incoming sonar data

Step 7: Output prediction — classify as Rock (R) or Mine (M)

Importing the libraries

```
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 preprocessing

```
data = pd.read_csv("/content/sonar.csv",header=None)
```

```
data.head()
```

	4	5	6	7	8	9	...	51	52	53	54	55	
	54	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027	0.0065	0.0159	0.0072	0.0167	0
	33	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.0084	0.0089	0.0048	0.0094	0.0191	0
	74	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.0232	0.0166	0.0095	0.0180	0.0244	0
	05	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.0121	0.0036	0.0150	0.0085	0.0073	0
	90	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.0031	0.0054	0.0105	0.0110	0.0015	0

```
data.shape
```

```
(208, 61)
```

```
data.describe()
```

What can I help you build?

	0	1	2	3	4	5	6	7	8	9
count	208.000000	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	0.208259
std	0.022991	0.032960	0.038428	0.046528	0.055552	0.059105	0.061788	0.085152	0.118387	0.134416
min	0.001500	0.000600	0.001500	0.005800	0.006700	0.010200	0.003300	0.005500	0.007500	0.011300
25%	0.013350	0.016450	0.018950	0.024375	0.038050	0.067025	0.080900	0.080425	0.097025	0.111275
50%	0.022800	0.030800	0.034300	0.044050	0.062500	0.092150	0.106950	0.112100	0.152250	0.182400
75%	0.035550	0.047950	0.057950	0.064500	0.100275	0.134125	0.154000	0.169600	0.233425	0.268700
max	0.137100	0.233900	0.305900	0.426400	0.401000	0.382300	0.372900	0.459000	0.682800	0.710600

8 rows × 60 columns

```
data[60].value_counts() #as we have the rock and mined labeled at 60th column
```

60	count
M	111
R	97

dtype: int64

M = Mine R = Rock ,Data is now grouped between them

```
data.groupby(60).mean()
```

60	0	1	2	3	4	5	6	7	8	9	...	50	5
M	0.034989	0.045544	0.050720	0.064768	0.086715	0.111864	0.128359	0.149832	0.213492	0.251022	...	0.019352	0.01601
R	0.022498	0.030303	0.035951	0.041447	0.062028	0.096224	0.114180	0.117596	0.137392	0.159325	...	0.012311	0.01045

2 rows × 60 columns

separating data and lables

```
x= data.drop(columns=60,axis=1)
y= data[60]
```

```
print(x)
print(y)
```

```
         0       1       2       3       4       5       6       7       8   \
0    0.0200  0.0371  0.0428  0.0207  0.0954  0.0986  0.1539  0.1601  0.3109
1    0.0453  0.0523  0.0843  0.0689  0.1183  0.2583  0.2156  0.3481  0.3337
2    0.0262  0.0582  0.1099  0.1083  0.0974  0.2280  0.2431  0.3771  0.5598
3    0.0100  0.0171  0.0623  0.0205  0.0205  0.0368  0.1098  0.1276  0.0598
4    0.0762  0.0666  0.0481  0.0394  0.0590  0.0649  0.1209  0.2467  0.3564
..      ...     ...     ...     ...     ...     ...     ...     ...     ...
203  0.0187  0.0346  0.0168  0.0177  0.0393  0.1630  0.2028  0.1694  0.2328
204  0.0323  0.0101  0.0298  0.0564  0.0760  0.0958  0.0990  0.1018  0.1030
205  0.0522  0.0437  0.0180  0.0292  0.0351  0.1171  0.1257  0.1178  0.1258
206  0.0303  0.0353  0.0490  0.0608  0.0167  0.1354  0.1465  0.1123  0.1945
207  0.0260  0.0363  0.0136  0.0272  0.0214  0.0338  0.0655  0.1400  0.1843

         9   ...      50      51      52      53      54      55      56  \
0    0.2111  ...  0.0232  0.0027  0.0065  0.0159  0.0072  0.0167  0.0180
1    0.2872  ...  0.0125  0.0084  0.0089  0.0048  0.0094  0.0191  0.0140
2    0.6194  ...  0.0033  0.0232  0.0166  0.0095  0.0180  0.0244  0.0316
3    0.1264  ...  0.0241  0.0121  0.0036  0.0150  0.0085  0.0073  0.0050
4    0.4459  ...  0.0156  0.0031  0.0054  0.0105  0.0110  0.0015  0.0072
..      ...  ...     ...     ...     ...     ...     ...     ...     ...
203  0.2684  ...  0.0203  0.0116  0.0098  0.0199  0.0033  0.0101  0.0065
```

```
204  0.2154  ...  0.0051  0.0061  0.0093  0.0135  0.0063  0.0063  0.0034
205  0.2529  ...  0.0155  0.0160  0.0029  0.0051  0.0062  0.0089  0.0140
206  0.2354  ...  0.0042  0.0086  0.0046  0.0126  0.0036  0.0035  0.0034
207  0.2354  ...  0.0181  0.0146  0.0129  0.0047  0.0039  0.0061  0.0040

        57      58      59
0    0.0084  0.0090  0.0032
1    0.0049  0.0052  0.0044
2    0.0164  0.0095  0.0078
3    0.0044  0.0040  0.0117
4    0.0048  0.0107  0.0094
..      ...     ...     ...
203  0.0115  0.0193  0.0157
204  0.0032  0.0062  0.0067
205  0.0138  0.0077  0.0031
206  0.0079  0.0036  0.0048
207  0.0036  0.0061  0.0115

[208 rows x 60 columns]
0      R
1      R
2      R
3      R
4      R
      ..
203    M
204    M
205    M
206    M
207    M
Name: 60, Length: 208, dtype: object
```

Train and Test data split

```
X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size=0.1,stratify=y,random_state=1)
```

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

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

Model training

```
model = LogisticRegression()
```

training the logistic regrassion model

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

```
▾ LogisticRegression
LogisticRegression()
```

Model evaluation

```
x_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(x_train_prediction,Y_train)
```

```
print("Accuracy on training data : ",training_data_accuracy)
```

```
Accuracy on training data :  0.8342245989304813
```

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

```
print("Accuracy on test data : ",test_data_accuracy)
```

```
Accuracy on test data :  0.7619047619047619
```

Building a Predictive system

```
input_data =(0.043,0.0902,0.0833,0.0813,0.0165,0.0277,0.0569,0.2057,0.3887,0.7106,0.7342,0.5033,
input_data_as_numpy_array = np.asarray(input_data) #changing the input_data to a numpy array
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)
prediction = model.predict(input_data_reshaped)
print(prediction)
if(prediction[0]=='R'):
  print("The object is a Rock")
else:
  print("The object is a Mine")
```

```
['M']
The object is a Mine
```