rock-vs-mine-prediction

0.0.1 Importing the Dependencies

[70]:
```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

0.0.2 Data Collection and Data Processing

[71]:
```
# loading the dataframe
df=pd.read_csv('sonar data.csv',header=None)
```

[11]:
```
df.head()
```

[11]:

	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	

	9	…	51	52	53	54	55	56	57	\
0	0.2111	…	0.0027	0.0065	0.0159	0.0072	0.0167	0.0180	0.0084	
1	0.2872	…	0.0084	0.0089	0.0048	0.0094	0.0191	0.0140	0.0049	
2	0.6194	…	0.0232	0.0166	0.0095	0.0180	0.0244	0.0316	0.0164	
3	0.1264	…	0.0121	0.0036	0.0150	0.0085	0.0073	0.0050	0.0044	
4	0.4459	…	0.0031	0.0054	0.0105	0.0110	0.0015	0.0072	0.0048	

	58	59	60
0	0.0090	0.0032	R
1	0.0052	0.0044	R
2	0.0095	0.0078	R
3	0.0040	0.0117	R

```
4  0.0107  0.0094   R

[5 rows x 61 columns]
```

[12]:
```
# number of rows and columns
df.shape
```

[12]:
```
(208, 61)
```

[13]:
```
# info
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: 99.3+ KB
```

[14]:
```
# describe // statistical measures of the data
df.describe()
```

[14]:

	0	1	2	3	4	5	\
count	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	
std	0.022991	0.032960	0.038428	0.046528	0.055552	0.059105	
min	0.001500	0.000600	0.001500	0.005800	0.006700	0.010200	
25%	0.013350	0.016450	0.018950	0.024375	0.038050	0.067025	
50%	0.022800	0.030800	0.034300	0.044050	0.062500	0.092150	
75%	0.035550	0.047950	0.057950	0.064500	0.100275	0.134125	
max	0.137100	0.233900	0.305900	0.426400	0.401000	0.382300	

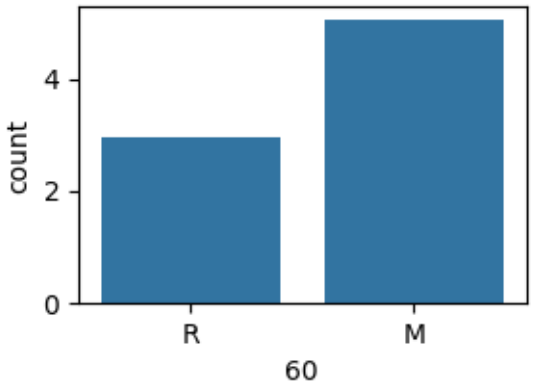
```
                6           7           8           9   …          50  \
count  208.000000  208.000000  208.000000  208.000000  …  208.000000
mean     0.121747    0.134799    0.178003    0.208259  …    0.016069
std      0.061788    0.085152    0.118387    0.134416  …    0.012008
min      0.003300    0.005500    0.007500    0.011300  …    0.000000
25%      0.080900    0.080425    0.097025    0.111275  …    0.008425
50%      0.106950    0.112100    0.152250    0.182400  …    0.013900
75%      0.154000    0.169600    0.233425    0.268700  …    0.020825
max      0.372900    0.459000    0.682800    0.710600  …    0.100400

               51          52          53          54          55          56  \
count  208.000000  208.000000  208.000000  208.000000  208.000000  208.000000
mean     0.013420    0.010709    0.010941    0.009290    0.008222    0.007820
std      0.009634    0.007060    0.007301    0.007088    0.005736    0.005785
min      0.000800    0.000500    0.001000    0.000600    0.000400    0.000300
25%      0.007275    0.005075    0.005375    0.004150    0.004400    0.003700
50%      0.011400    0.009550    0.009300    0.007500    0.006850    0.005950
75%      0.016725    0.014900    0.014500    0.012100    0.010575    0.010425
max      0.070900    0.039000    0.035200    0.044700    0.039400    0.035500

               57          58          59
count  208.000000  208.000000  208.000000
mean     0.007949    0.007941    0.006507
std      0.006470    0.006181    0.005031
min      0.000300    0.000100    0.000600
25%      0.003600    0.003675    0.003100
50%      0.005800    0.006400    0.005300
75%      0.010350    0.010325    0.008525
max      0.044000    0.036400    0.043900

[8 rows x 60 columns]
```

[17]:
```
# Count of Mine (M) and Rock (R)
df[60].value_counts()
```

[17]:
```
60
M    111
R     97
Name: count, dtype: int64
```

[79]:
```
# count plot
plt.figure(figsize=(3,2))
sns.countplot(x=df[60], data=df)
```

[79]:
```
<Axes: xlabel='60', ylabel='count'>
```



[80]:
```
# check null vlaues
df.isnull().sum()
```

[80]:
```
0     0
1     0
2     0
3     0
4     0
     ..
56    0
57    0
58    0
59    0
60    0
Length: 61, dtype: int64
```

[25]:
```
# get mean of both Mine and Rock which is M and R
df.groupby(60).mean()
```

[25]:

60	0	1	2	3	4	5	6	7	8	9	…	50	51	52	53	54	55	56	57	58	59
M	0.034989	0.045544	0.050720	0.064768	0.086715	0.111864	0.128359	0.149832	0.213492	0.251022	…	0.019352	0.016014	0.011643	0.012185	0.009923	0.008914	0.007825	0.009060	0.008695	0.006930
R	0.022498	0.030303	0.035951	0.041447	0.062028	0.096224	0.114180	0.117596	0.137392	0.159325	…	0.012311	0.010453	0.009640	0.009518						

```
R   0.008567  0.007430  0.007814  0.006677  0.007078  0.006024

[2 rows x 60 columns]
```

0.0.3 Separating data and labels

[27]:
```
X= df.drop(columns=60, axis=1)
Y= df[60]
```

[28]:
```
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
```

0.0.4 Train and Test data

[37]:
```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.1, stratify
↪= Y, random_state = 1)
```

[39]:
```
print(X.shape, X_train.shape, X_test.shape)
```

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

[39]:
```
print(X.shape, X_train.shape, X_test.shape)
```

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

1 Model Training >> Logistic Regression

[44]:
```
model = LogisticRegression()
```

[45]:
```
# training the logistic Regrssion model with training data
model.fit(X_train, Y_train)
```

[45]:
```
LogisticRegression()
```

1.0.1 Model Evaluation

[68]:
```
# prediction
X_train_prediction = model.predict(X_train)
```

[50]:
```
# accuarcy of training data
training_data_accuarcy= accuracy_score(X_train_prediction, Y_train)
```

```
print(training_data_accuarcy)
```

```
0.8342245989304813
```

[66]:
```
# compare the prediction value with Actual value
result_df=pd.DataFrame({'Actual': Y_train, 'Prediction':X_train_prediction})
```

[69]:
```
# get output and check accuracy
print(result_df.tail(10))
print(f"\n Accuracy on the test set: {training_data_accuarcy}")
```

```
    Actual Prediction
177      M          M
171      M          M
94       R          R
185      M          M
67       R          R
140      M          M
5        R          R
154      M          M
131      M          M
203      M          M

 Accuracy on the test set: 0.8342245989304813
```

2 Making Predicting data

[65]:
```
input_data = (0.0526,0.0563,0.1219,0.1206,0.0246,0.1022,0.0539,0.0439,0.2291,0.
 ↪1632,0.2544,0.2807,0.3011,0.3361,0.3024,0.2285,0.2910,0.1316,0.1151,0.3404,0.
 ↪5562,0.6379,0.6553,0.7384,0.6534,0.5423,0.6877,0.7325,0.7726,0.8229,0.8787,0.
 ↪9108,0.6705,0.6092,0.7505,0.4775,0.1666,0.3749,0.3776,0.2106,0.5886,0.5628,0.
 ↪2577,0.5245,0.6149,0.5123,0.3385,0.1499,0.0546,0.0270,0.0380,0.0339,0.0149,0.
 ↪0335,0.0376,0.0174,0.0132,0.0103,0.0364,0.0208)

# Changing the input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the np arrray as we are predicting for one instance
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
```