Problem Statement

SONAR uses sound signals from submarine needs to detect that object underwater is rock or mine

Work Flow

1. Sonar Data:- labortary data is used here which is obtained from rock and metal cylinder 2. Data Preprocessing:- need to do analyse the and make data fit for modelling 3. Split Data:- splitting of data into test-train 4. Model Selection:- choosing the best model 5. Model Evaluation:- check for the model evaluation

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv('sonar_data.csv',header=None)
```

In [3]:
```
df
```

Out[3]:

	0	1	2	3	4	5	6	7	8	9	...	51	52	53	54	55	
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	0.0167	0.0
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	0.0191	0.0
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	0.0244	0.0
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	0.0073	0.0
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	0.0015	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
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	0.0199	0.0033	0.0101	0.0
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	0.0135	0.0063	0.0063	0.0
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	0.0051	0.0062	0.0089	0.0
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	0.0126	0.0036	0.0035	0.0
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	0.0047	0.0039	0.0061	0.0

208 rows × 61 columns

In [4]:
```
df.isnull().sum()
```

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

observation:- no null values in the data

In [5]:
```
df.duplicated().sum()
```

Out[5]:
```
0
```

observation:- no duplicate value in the dataset

In [6]:
```
df[60].value_counts()
```

Out[6]:
```
60
M    111
R     97
Name: count, dtype: int64
```

no need to perform under sampling

observation:- signal has detected 111 mines and 97 rocks in labortary experiment

In [7]:
```
#checking if their is need for standardization or not
df.drop(60,axis=1).std()
```

Out[7]:
```
0     0.022991
1     0.032960
2     0.038428
3     0.046528
4     0.055552
5     0.059105
6     0.061788
7     0.085152
8     0.118387
9     0.134416
10    0.132705
11    0.140072
12    0.140962
13    0.164474
14    0.205427
15    0.232650
16    0.263677
17    0.261529
18    0.257988
19    0.262653
20    0.257818
21    0.255883
22    0.250175
23    0.239116
24    0.244926
25    0.237228
26    0.245657
27    0.237189
28    0.240250
29    0.220749
30    0.213992
31    0.213237
32    0.206513
33    0.231242
34    0.259132
35    0.264121
36    0.239912
37    0.212973
38    0.199075
39    0.178662
40    0.171111
41    0.168728
42    0.138993
43    0.133291
44    0.151628
45    0.133938
46    0.086953
47    0.062417
48    0.035954
49    0.013665
50    0.012008
51    0.009634
52    0.007060
53    0.007301
54    0.007088
55    0.005736
56    0.005785
57    0.006470
58    0.006181
59    0.005031
dtype: float64
```

no need to perform standartization as all the data is in common format and range(standard deviation is around 1)

In [8]:
```
#statistical analysis
df.describe()
```

Out[8]:

	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

In [9]:
```
df.groupby(60).mean()
```

Out[9]:

60	0	1	2	3	4	5	6	7	8	9	...	50	51	
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.(
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.(

2 rows × 60 columns

In [10]:
```
#separating features and target variable
x=df.drop(60,axis=1)
```

In [11]:
```
y=df[60]
```

In [12]:
```
x.shape
```

Out[12]: (208, 60)

In [13]:
```
y.shape
```

Out[13]: (208,)

In [14]:
```
#splitting the data into train and test
from sklearn.model_selection import train_test_split
```

In [15]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=2,stratify=y)
#stratify-> data will be split acc. to no. of rock and mine
```

model used for evaluation 1.logistic 2.decision tree 3.random forest 4.naive bayes 5.knn

In [16]:
```
#logistic
from sklearn.linear_model import LogisticRegression
```

In [17]:
```
reg=LogisticRegression()
```

In [19]:
```
reg.fit(x_train,y_train)
```

Out[19]:
```
▾ LogisticRegression
LogisticRegression()
```

In [20]:
```
y_predict=reg.predict(x_test)
```

In [21]:
```
#model evaluation
from sklearn.metrics import accuracy_score
```

In [24]:
```
logistic_accuracy=round(accuracy_score(y_predict,y_test),3)
logistic_accuracy
```

Out[24]: 0.833

In [25]:
```
#decision tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

In [26]:
```
decision_model=DecisionTreeClassifier()
```

In [27]:
```python
decision_model.fit(x_train,y_train)
```

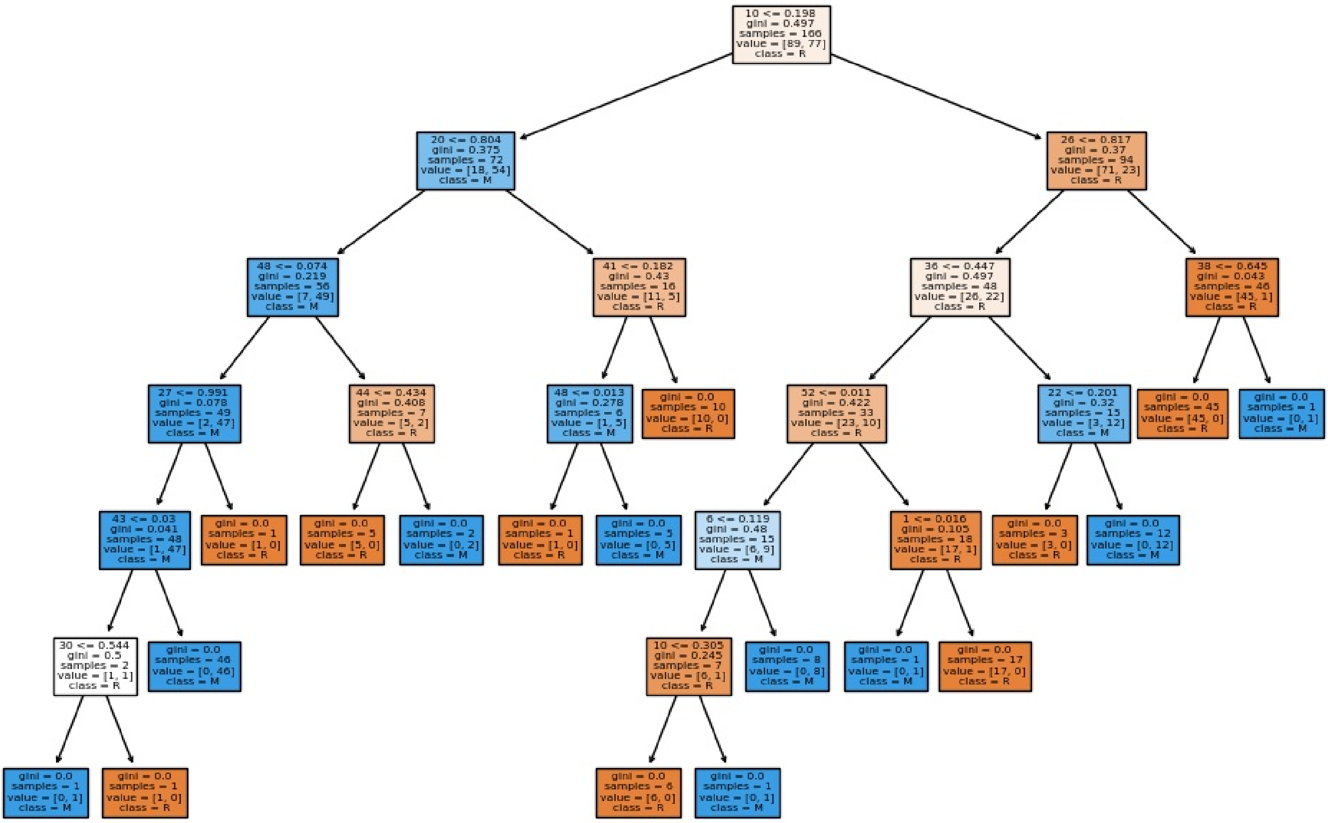
Out[27]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [28]:
```python
y_predict=decision_model.predict(x_test)
```

In [30]:
```python
#model evaluation
decision_accuracy=round(accuracy_score(y_predict,y_test),3)
decision_accuracy
```

Out[30]:
```
0.738
```

In [39]:
```python
#visualize decision tree
plt.figure(figsize=(12,8))
tree.plot_tree(decision_model,filled=True,feature_names=x.columns,class_names=['R','M'],label='all')
plt.show()
```



In [40]:
```python
#random forest
from sklearn.ensemble import RandomForestClassifier
```

In [42]:
```python
random_model=RandomForestClassifier(n_estimators=100,random_state=2)
```

In [43]:
```python
random_model.fit(x_train,y_train)
```

Out[43]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=2)
```

In [44]:
```python
y_predict=random_model.predict(x_test)
```

In [46]:
```python
#model evaluation
random_accuracy=round(accuracy_score(y_predict,y_test),3)
random_accuracy
```

Out[46]:
```
0.952
```

In [50]:
```python
#feature importance
importance=random_model.feature_importances_
feature_name=x.columns
feature_imp_df=pd.DataFrame({'feature':feature_name,
                            'importance':importance})
print(feature_imp_df.sort_values(by='importance',ascending=False))
```

```
    feature  importance
10       10    0.062564
11       11    0.050623
8         8    0.045347
48       48    0.040870
47       47    0.037796
36       36    0.031665
50       50    0.027847
20       20    0.026632
46       46    0.024030
35       35    0.023441
22       22    0.023441
9         9    0.022913
45       45    0.019324
44       44    0.019167
34       34    0.018222
26       26    0.017482
30       30    0.017344
19       19    0.016905
38       38    0.016594
42       42    0.016578
27       27    0.016358
51       51    0.015585
18       18    0.014425
21       21    0.014414
12       12    0.014297
17       17    0.014212
39       39    0.014094
43       43    0.013720
14       14    0.013641
0         0    0.013421
23       23    0.013275
49       49    0.013040
16       16    0.013012
7         7    0.012779
54       54    0.011897
15       15    0.011760
31       31    0.011423
4         4    0.011127
33       33    0.011031
3         3    0.010441
5         5    0.010428
41       41    0.010375
6         6    0.010343
28       28    0.010343
25       25    0.010049
56       56    0.009748
24       24    0.009724
1         1    0.009583
59       59    0.009431
58       58    0.009372
57       57    0.009342
40       40    0.009191
52       52    0.009127
29       29    0.008620
55       55    0.007865
2         2    0.007401
13       13    0.007109
53       53    0.006803
37       37    0.006205
32       32    0.006203
```

In [51]:
```
#naive bayes
from sklearn.naive_bayes import BernoulliNB
```

In [52]:
```
naive_model=BernoulliNB()
```

In [53]:
```
naive_model.fit(x_train,y_train)
```

Out[53]:
```
▾ BernoulliNB
BernoulliNB()
```

In [54]:
```
y_predict=naive_model.predict(x_test)
```

In [55]:
```
#model evaluation
naive_accuracy=round(accuracy_score(y_predict,y_test),3)
naive_accuracy
```

Out[55]:
```
0.524
```

In [57]:
```
#k-nearest neighbour
from sklearn.neighbors import KNeighborsClassifier
```

In [58]:
```
k=7
k_model=KNeighborsClassifier(n_neighbors=k)
```

In [60]:
```
k_model.fit(x_train,y_train)
```

Out[60]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

In [61]:
```
y_predict=k_model.predict(x_test)
```

In [62]:
```
#model evaluation
k_accuracy=round(accuracy_score(y_predict,y_test),3)
k_accuracy
```

Out[62]: 0.738

In [63]:
```
model_evaluation=pd.DataFrame({'accuracy score':[logistic_accuracy,decision_accuracy,
                                                  random_accuracy,naive_accuracy,
                                                  k_accuracy]},index=['logistic reg','decision tree','random fore
                                                                      'naive bayes','knn'])
```

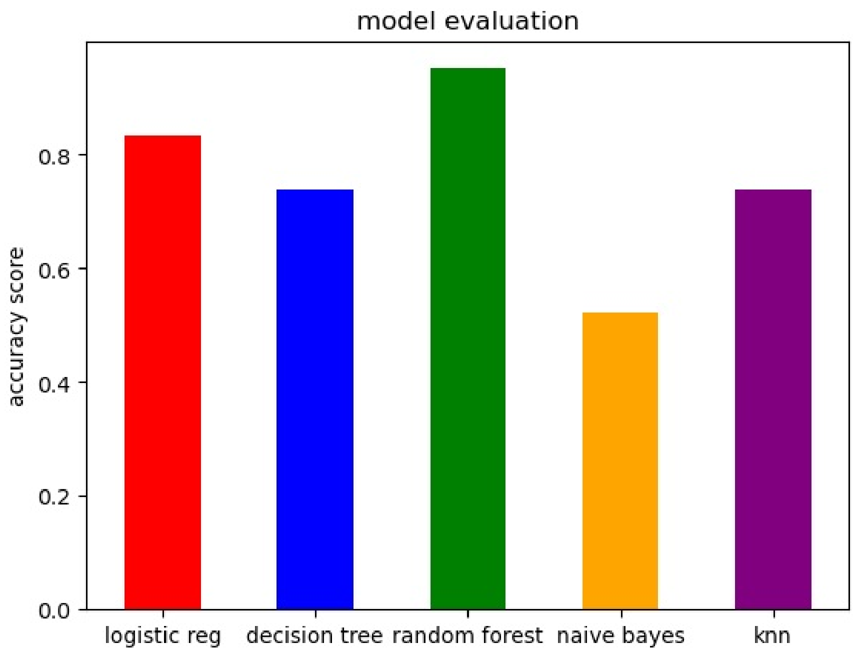
In [64]:
```
model_evaluation
```

Out[64]:

	accuracy score
logistic reg	0.833
decision tree	0.738
random forest	0.952
naive bayes	0.524
knn	0.738

In [94]:
```
colors=['red','blue','green','orange','purple']
model_evaluation.plot(kind='bar',y='accuracy score',legend=False,color=colors)
plt.ylabel('accuracy score')
plt.title('model evaluation')
plt.xticks(rotation=360)
```

Out[94]:
```
(array([0, 1, 2, 3, 4]),
 [Text(0, 0, 'logistic reg'),
  Text(1, 0, 'decision tree'),
  Text(2, 0, 'random forest'),
  Text(3, 0, 'naive bayes'),
  Text(4, 0, 'knn')])
```



observation:- random forest can predict more accurately

as from the above chart it is clear that random forest is more acurate for pedicting the roc/mine underwater, make a predictive system

Making Predictive System

In [97]:
```
input_data=(0.0162,0.0041,0.0239,0.0441,0.0630,0.0921,0.1368,0.1078,0.1552,0.1779,0.2164,
            0.2568,0.3089,0.3829,0.4393,0.5335,0.5996,0.6728,0.7309,0.8092,0.8941,0.9668,
            1.0000,0.9893,0.9376,0.8991,0.9184,0.9128,0.7811,0.6018,0.3765,0.3300,0.2280,
            0.0212,0.1117,0.1788,0.2373,0.2843,0.2241,0.2715,0.3363,0.2546,0.1867,0.2160,
            0.1278,0.0768,0.1070,0.0946,0.0636,0.0227,0.0128,0.0173,0.0135,0.0114,0.0062,0.0157,0.0088,0.0036,0
#coping some values and using them to see if prediction is done correctly
#this should give-> M

#changing input data to numpy array for easy processing
data_array=np.asarray(input_data)

#reshape the data, as we are predicting for one instance and so model doesn't get confused with no. of data poi
data=data_array.reshape(1,-1)

#making prediction
prediction=random_model.predict(data)
prediction

if(prediction[0]=='R'):
    print('object is a rock')
else:
    print('object is a mine')
```

```
object is a mine
```

prediction is done correctly

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js