```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt


sonar_data = pd.read_csv('/content/sonar data.csv', header=None) # header= None because there is no header in dataset


sonar_data.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	0.1609	0.1582	0.2238	0.0645	0.0660	0.2273	0.3100	0.
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	0.4918	0.6552	0.6919	0.7797	0.7464	0.9444	1.0000	0.
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	0.6333	0.7060	0.5544	0.5320	0.6479	0.6931	0.6759	0.
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	0.0881	0.1992	0.0184	0.2261	0.1729	0.2131	0.0693	0.
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	0.4152	0.3952	0.4256	0.4135	0.4528	0.5326	0.7306	0.

It is a Supervised Learning Problem because target column have labels R(Rock) or M(Mine)

In Un-Supervised Learning there is no labels in data or target colmn

It is a classification because there is discrete(separate) values in target column

```
# Check the unique values in the last column (target column)
unique_values = sonar_data[60].unique()
print("Unique target values:", unique_values)
```

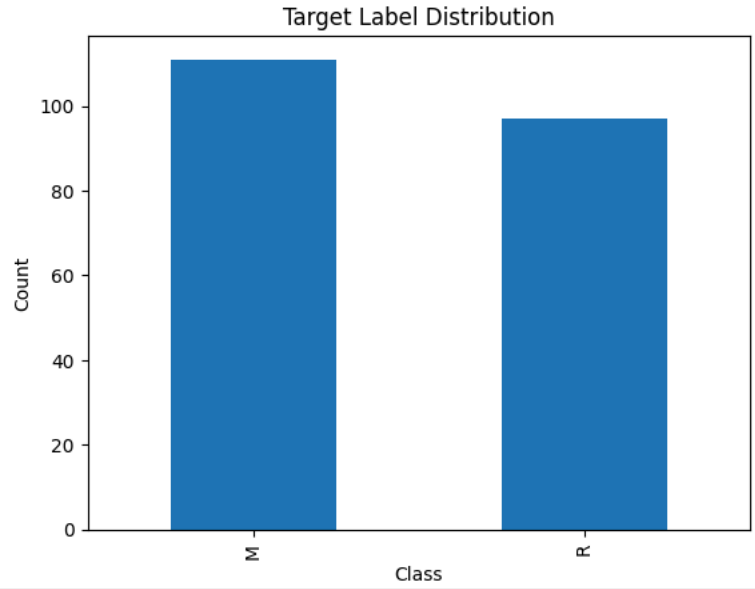
```
Unique target values: ['R' 'M']
```

```
# Check data type and number of unique values in the target column
target = sonar_data[60]
print("Data type of target:", target.dtype)
print("Number of unique values:", target.nunique())

# Determine if classification or regression
if target.nunique() < 20 and target.dtype == 'object':
    print("This is a classification problem.")
else:
    print("This is a regression problem.")
```

```
Data type of target: object
Number of unique values: 2
This is a classification problem.
```

```
# Plot the distribution of target labels
target.value_counts().plot(kind='bar')
plt.title("Target Label Distribution")
plt.xlabel("Class")
plt.ylabel("Count")
plt.show()
```




```
sonar_data.shape # tell the rows and columns
```

```
(208, 61)
```

```
sonar_data.describe() # give the statistical overview of the data
```

```
sonar_data[60].value_counts() # we use [60] because we want to count the values only 60th (target) column
# this count show that there is no much difference between both Mine or Rock value
```

60	count
M	111
R	97

dtype: int64

```
sonar_data.groupby(60).mean()
```

60	0	1	2	3	4	5	6	7	8	9	10	11	12	
M	0.034989	0.045544	0.050720	0.064768	0.086715	0.111864	0.128359	0.149832	0.213492	0.251022	0.289581	0.301459	0.314426	0.32
R	0.022498	0.030303	0.035951	0.041447	0.062028	0.096224	0.114180	0.117596	0.137392	0.159325	0.174713	0.191589	0.226249	0.26

Separating the features (X) and target labels (Y)

Which Method to Use?

Use .drop() and indexing (sonar_data[60]):

X = sonar_data.drop(columns=60, axis=1) # Features Y = sonar_data[60] # Target column

when working with datasets without column names (like the Sonar dataset).

Use .iloc:

X = sonar_data.iloc[:, :-1] # All rows, all columns except the last Y = sonar_data.iloc[:, -1] # All rows, only the last column

for numerical indexing if you want more flexibility and avoid specifying column names. ***Use .pop():***

if you want to modify the original dataset and remove the target column.

Y = sonar_data.pop(60) # Removes column 60 and stores it in Y X = sonar_data # Remaining columns are the features

Using Column Names (If Available):

If your dataset has column names, you can specify them directly instead of using column indices.

X = sonar_data.drop(columns=['Target']) # Assuming the target column is named 'Target' Y = sonar_data['Target']

```
X = sonar_data.drop(columns=60, axis=1)  # Features
Y = sonar_data[60]  # Target column
```

```
display(X)
display(Y)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	0.1609	0.1582	0.2238	0.0645	0.0660	0.2273	0.3100
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	0.4918	0.6552	0.6919	0.7797	0.7464	0.9444	1.0000
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	0.6333	0.7060	0.5544	0.5320	0.6479	0.6931	0.6759
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	0.0881	0.1992	0.0184	0.2261	0.1729	0.2131	0.0693
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	0.4152	0.3952	0.4256	0.4135	0.4528	0.5326	0.7306
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
203	0.0187	0.0346	0.0168	0.0177	0.0393	0.1630	0.2028	0.1694	0.2328	0.2684	0.3108	0.2933	0.2275	0.0994	0.1801	0.2200	0.2732
204	0.0323	0.0101	0.0298	0.0564	0.0760	0.0958	0.0990	0.1018	0.1030	0.2154	0.3085	0.3425	0.2990	0.1402	0.1235	0.1534	0.1901
205	0.0522	0.0437	0.0180	0.0292	0.0351	0.1171	0.1257	0.1178	0.1258	0.2529	0.2716	0.2374	0.1878	0.0983	0.0683	0.1503	0.1723
206	0.0303	0.0353	0.0490	0.0608	0.0167	0.1354	0.1465	0.1123	0.1945	0.2354	0.2898	0.2812	0.1578	0.0273	0.0673	0.1444	0.2070
207	0.0260	0.0363	0.0136	0.0272	0.0214	0.0338	0.0655	0.1400	0.1843	0.2354	0.2720	0.2442	0.1665	0.0336	0.1302	0.1708	0.2177

208 rows × 60 columns

	60
0	R
1	R
2	R
3	R
4	R
...	...
203	M
204	M
205	M
206	M
207	M

208 rows × 1 columns

dtype: object

Training and Testing Splits

```
X_train, X_test, Y_train, Y_Test = train_test_split(X, Y, test_size = 0.1, stratify=Y, random_state=1)
```

```
print("X= : ", X.shape, "\nX_train= : ",X_train.shape, "\nX_test= : ", X_test.shape)
```

```
X= :  (208, 60)
X_train= :  (187, 60)
X_test= :  (21, 60)
```

Model Training

```
model = LogisticRegression()
```

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

```
LogisticRegression()
```

Model Evaluation

```
# accuracy on training data
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
```

```
# accuracy on test data
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
```

Making a Predictive System

```
input_data = (0.0164,0.0173,0.0347,0.0070,0.0187,0.0671,0.1056,0.0697,0.0962,0.0251,0.0801,0.1056,0.1266,0.0890,0.0198,0.1133,0.2826,0.32

# convert data into numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the np array 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')
```

```
['R']
The Object is a Rock
```