Importing Libraries and Ignoring Warnings

In this section, we import the essential libraries required for data analysis and building a logistic regression model. Specifically, we use:

Pandas and NumPy for handling data. Scikit-learn for splitting data, applying logistic regression, and evaluating model accuracy. Additionally, some unnecessary warnings are suppressed to improve readability and focus on significant results.

In [20]:

```
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 warnings
warnings.filterwarnings("ignore", message="X does not have valid feature names")
```

Loading and Displaying the Sonar Dataset

In this section, the sonar dataset is loaded from a CSV file using Pandas. The first few rows of the dataset are displayed using head(), allowing a quick overview of the structure and content of the data. This step is essential for initial data exploration and understanding.

In [21]:

```
sonar_data = pd.read_csv("sonar.mine.csv")
sonar_data.head()
```

Out[21]:

	Freq_1	Freq_2	Freq_3	Freq_4	Freq_5	Freq_6	Freq_7	Freq_8	Freq_9	Freq_10	...	Freq_52	Freq_53	Freq_54	Freq_55	Freq_56	Fre
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.
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.
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.
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.
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.

5 rows × 61 columns

Dataset Dimensions and Statistical Summary

This section retrieves the shape of the sonar dataset using shape, which provides the number of rows and columns. Additionally, describe() is called to generate a statistical summary of the numerical features, including measures such as count, mean, standard deviation, minimum, and maximum values. These insights are crucial for assessing the dataset's size and understanding its distribution.

In [22]:

```
sonar_data.shape
```

Out[22]:

```
(208, 61)
```

In [23]:

```
sonar_data.describe()
```

Out[23]:

	Freq_1	Freq_2	Freq_3	Freq_4	Freq_5	Freq_6	Freq_7	Freq_8	Freq_9	Freq_10	...	Freq_
count	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	...	208.0000
mean	0.029164	0.038437	0.043832	0.053892	0.075202	0.104570	0.121747	0.134799	0.178003	0.208259	...	0.0160
std	0.022991	0.032960	0.038428	0.046528	0.055552	0.059105	0.061788	0.085152	0.118387	0.134416	...	0.0120
min	0.001500	0.000600	0.001500	0.005800	0.006700	0.010200	0.003300	0.005500	0.007500	0.011300	...	0.0000
25%	0.013350	0.016450	0.018950	0.024375	0.038050	0.067025	0.080900	0.080425	0.097025	0.111275	...	0.0084
50%	0.022800	0.030800	0.034300	0.044050	0.062500	0.092150	0.106950	0.112100	0.152250	0.182400	...	0.0139
75%	0.035550	0.047950	0.057950	0.064500	0.100275	0.134125	0.154000	0.169600	0.233425	0.268700	...	0.0208
max	0.137100	0.233900	0.305900	0.426400	0.401000	0.382300	0.372900	0.459000	0.682800	0.710600	...	0.1004

8 rows × 60 columns

Counting Instances of Each Label

In this section, the value_counts() function is used to count the number of occurrences of each unique value in the "Label" column of the sonar dataset. This analysis helps in understanding the distribution of classes within the dataset, indicating whether the data is balanced or imbalanced, which is essential for model training.

In [24]:

```
sonar_data["Label"].value_counts()
```

Out[24]:
```
Label
M    111
R     97
Name: count, dtype: int64
```

M ----> Mine

R ----> Rock

Analyzing Mean Values Grouped by Label

This line groups the dataset by the "Label" column and calculates the mean for each feature within each label group. This helps in understanding the differences between the groups based on their average feature values, which can provide insights into the dataset's structure.

In [25]:
```
sonar_data.groupby("Label").mean()
```

Out[25]:

Label	Freq_1	Freq_2	Freq_3	Freq_4	Freq_5	Freq_6	Freq_7	Freq_8	Freq_9	Freq_10	...	Freq_51	Freq_52	Freq_53
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
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

2 rows × 60 columns

Separating Features and Labels

In this section, the dataset is divided into features and labels. The features are extracted by dropping the "Label" column from the sonar_data, resulting in the variable x. The labels, representing the target variable, are stored in the variable y. This separation is a crucial step in preparing the data for model training and evaluation.

In [26]:
```
# separating data and Labels
x =sonar_data.drop(columns="Label", axis =1)
y = sonar_data["Label"]
```

Training and test data

Splitting the Data into Training and Test Sets

This section uses train_test_split to divide the feature set (x) and the labels (y) into training and testing datasets. The test_size is set to 10%, and the stratify parameter ensures that the distribution of labels is preserved in both sets. The shapes of the original feature set and the resulting training and testing sets are printed to verify the split. This step is essential for evaluating the model's performance on unseen data.

In [27]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size= 0.1, stratify= y, random_state=1)
```

In [28]:
```
print(x.shape, x_train.shape, x_test.shape)
```

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

Training the Logistic Regression Model

In this section, a logistic regression model is instantiated and then trained using the training data (x_train and y_train). The fit() method is employed to optimize the model parameters based on the provided training dataset. This step is crucial for enabling the model to learn the underlying patterns in the data, preparing it for making predictions on new, unseen data.

In [29]:
```
model= LogisticRegression()
```

In [30]:
```
#training the logistic regression model with training data
model.fit(x_train, y_train)
```

Out[30]:
```
▾ LogisticRegression
LogisticRegression()
```

Model Evaluation

Evaluating Model Accuracy on Training and Test Data

In this section, the accuracy of the logistic regression model is assessed on both the training and test datasets. The model makes predictions on the training data (x_train), and the accuracy is calculated using accuracy_score, comparing the predictions to the actual labels (y_train). This process is repeated for the test dataset (x_test) to evaluate how well the model performs on unseen data. The results are printed, providing insight into the model's performance and generalization capabilities.

In [31]:
```python
# accuracy on training data
x_train_pred = model.predict(x_train)
training_data_accu = accuracy_score(x_train_pred, y_train)
```

In [32]:
```python
print('accuracy on training data :',training_data_accu)
```

```
accuracy on training data : 0.8342245989304813
```

In [33]:
```python
x_test_pred = model.predict(x_test)
test_data_accu = accuracy_score(x_test_pred, y_test)
```

In [34]:
```python
print('accuracy on training data :',test_data_accu)
```

```
accuracy on training data : 0.7619047619047619
```

Making Predictions with New Input Data

In this section, a new input data instance is defined as a tuple of feature values. The data is then converted into a NumPy array and reshaped to match the expected input format for the model (1 sample with multiple features). The trained logistic regression model is used to predict whether the input data represents a "Rock" or a "Mine." The prediction result is printed to provide insight into the classification made by the model based on the provided features.

In [35]:
```python
input_data =(0.0286,0.0453, 0.0277,0.0174,0.0384,0.099,0.1201,0.1833,0.2105,0.3039,0.2988,0.425,0.6343,0.8198,1
)

#changing the input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)


#reshape the np array as we are predecting for one instance

input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)
prediction = model.predict(input_data_reshaped)
if(prediction[0]=='R'):
    print('the object is Rock')
else:
    print('the object is a Mine')
```

```
the object is Rock
```

In [ ]: