Importing the required Dependencies

#Import the basic dependencies and keep on importing the dependencies as per need

import numpy as  np

import pandas as pd

Data collection and preprocessing

# converting the csv file into pandas data frame

sonardata = '/content/sonar\_data.csv.csv'

dataset = pd.read\_csv(sonardata, header=None)

dataset.head()

dataset.shape

dataset.tail()

dataset.describe()

#The data seems pretty standardize so no need of standardizing of data

#So now lets check the missing values

missing\_values = dataset.isnull().sum()

print(missing\_values)

Training Test Slpit

from sklearn.model\_selection import train\_test\_split

#first we need to split the data

X = dataset.drop(60, axis = 1)

Y = dataset[60]

Label Encoding Y data

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(Y)

Y = y\_encoded

# Rock ---> 1

# Mine ---> 0

print(X)

print(Y)

# use train test split

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y, stratify=Y, random\_state=3)

X\_train.shape, X\_test.shape, Y\_train.shape, Y\_test.shape

TRAINING THE MODEL

#importing the logistic regression

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

model.fit(X\_train, Y\_train)

Evaluating the model 1 Accuracy Score 2 Confusion Matrix

# 1) Accuracy Score

from sklearn.metrics import accuracy\_score

Y\_predict = model.predict(X\_test)

accuracy = accuracy\_score(Y\_test, Y\_predict)

print("Accuracy:", accuracy)

#2 Confusion Matrix

from sklearn.metrics import confusion\_matrix

conf\_matrix = confusion\_matrix(Y\_test, Y\_predict)

print("Confusion Matrix:")

print(conf\_matrix)

Confusion matrix using Matplotlib

import matplotlib.pyplot as plt

# Create a confusion matrix plot

plt.figure(figsize=(8, 6))

plt.imshow(conf\_matrix, cmap=plt.cm.Blues, interpolation='nearest')

plt.title('Confusion Matrix')

plt.colorbar()

classes = ['Rock', 'Sonar']

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

for i in range(len(classes)):

    for j in range(len(classes)):

        plt.text(j, i, format(conf\_matrix[i, j], 'd'),

                 horizontalalignment="center",

                 color="white" if conf\_matrix[i, j] > conf\_matrix.max() / 2 else "black")

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.tight\_layout()

plt.show()

Make a predictive

input\_data = (0.0260,0.0363,0.0136,0.0272,0.0214,0.0338,0.0655,0.1400,0.1843,0.2354,0.2720,0.2442,0.1665,0.0336,0.1302,0.1708,0.2177,0.3175,0.3714,0.4552,0.5700,0.7397,0.8062,0.8837,0.9432,1.0000,0.9375,0.7603,0.7123,0.8358,0.7622,0.4567,0.1715,0.1549,0.1641,0.1869,0.2655,0.1713,0.0959,0.0768,0.0847,0.2076,0.2505,0.1862,0.1439,0.1470,0.0991,0.0041,0.0154,0.0116,0.0181,0.0146,0.0129,0.0047,0.0039,0.0061,0.0040,0.0036,0.0061,0.0115)

# Changing the input data to a 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)

# Make prediction

prediction = model.predict(input\_data\_reshaped)

# Decode prediction

decoded\_prediction = label\_encoder.inverse\_transform(prediction)

if(prediction[0] == 1):

  print('The object is a Rock')

else:

  print('The object is a Mine')