

Certainly! Here’s a line-by-line explanation of the code:

python

Copy code

import numpy as np

import pandas as pd

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

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

1. **Import libraries**:
   * numpy: Used for numerical computations and array handling.
   * pandas: For handling data as DataFrames, which is useful for data analysis.
   * train\_test\_split: From sklearn, it splits data into training and test sets.
   * LogisticRegression: A machine learning model for binary classification.
   * accuracy\_score: Metric for measuring the model's accuracy.

python

Copy code

# loading the dataset to a pandas DataFrame

sonar\_data = pd.read\_csv('/content/sonar data.csv', header=None)

1. **Load dataset**: Reads a CSV file (sonar data) into a DataFrame, sonar\_data. header=None means there’s no header row.

python

Copy code

sonar\_data.head()

1. **Preview data**: Displays the first five rows of the dataset.

python

Copy code

# number of rows and columns

sonar\_data.shape

1. **Dataset shape**: Returns the dimensions (number of rows and columns) of the dataset.

python

Copy code

sonar\_data.describe() # describe --> statistical measures of the data

1. **Statistical summary**: Displays summary statistics for each column in the dataset (e.g., mean, standard deviation, min, max).

python

Copy code

sonar\_data[60].value\_counts()

1. **Class distribution**: Counts the occurrences of each label in column 60, which is the target variable ('R' for rock or 'M' for mine).

python

Copy code

sonar\_data.groupby(60).mean()

1. **Mean feature values per class**: Groups the data by label and calculates the mean for each feature in each class.

python

Copy code

# separating data and labels

X = sonar\_data.drop(columns=60, axis=1)

Y = sonar\_data[60]

1. **Separate features and labels**:
   * X: Contains the features (all columns except 60).
   * Y: Contains the labels (column 60).

python

Copy code

print(X)

print(Y)

1. **Print features and labels**: Displays the feature set X and labels Y.

python

Copy code

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.1, stratify=Y, random\_state=1)

1. **Split data into training and test sets**:
   * test\_size=0.1: 10% of the data goes to the test set.
   * stratify=Y: Ensures both sets have the same proportion of labels.
   * random\_state=1: Ensures reproducibility.

python

Copy code

print(X.shape, X\_train.shape, X\_test.shape)

1. **Print shapes of datasets**: Shows the dimensions of X, X\_train, and X\_test.

python

Copy code

print(X\_train)

print(Y\_train)

1. **Print training data**: Displays the training feature set and labels.

python

Copy code

model = LogisticRegression()

# training the Logistic Regression model with training data

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

1. **Initialize and train model**:
   * LogisticRegression(): Creates an instance of the model.
   * model.fit(X\_train, Y\_train): Trains the model on the training data.

python

Copy code

# 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)

1. **Training accuracy**:
   * model.predict(X\_train): Predicts labels for the training set.
   * accuracy\_score: Calculates the accuracy by comparing predictions to Y\_train.

python

Copy code

# 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)

1. **Test accuracy**:
   * model.predict(X\_test): Predicts labels for the test set.
   * accuracy\_score: Calculates accuracy by comparing predictions to Y\_test.

python

Copy code

input\_data = (0.0307, 0.0523, ..., 0.0055) # shortened for readability

1. **New input data**: Tuple containing sensor readings for a new observation.

python

Copy code

# changing the input\_data to a numpy array

input\_data\_as\_numpy\_array = np.asarray(input\_data)

1. **Convert to numpy array**: Converts input\_data to a NumPy array, making it easier to reshape.

python

Copy code

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

input\_data\_reshaped = input\_data\_as\_numpy\_array.reshape(1, -1)

1. **Reshape array**: Changes the array shape to (1, -1), required for prediction with one instance.

python

Copy code

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

print(prediction)

1. **Prediction**: Uses the model to predict the class of input\_data\_reshaped. Outputs ['R'] or ['M'].

python

Copy code

if (prediction[0] == 'R'):

print('The object is a Rock')

else:

print('The object is a mine')

1. **Interpret prediction**: Checks the model’s prediction and prints a message:
   * 'R' means “Rock”
   * 'M' means “Mine”