

## ✖ Importing Libraries

- **numpy (np)**: A library for numerical operations in Python.
- **pandas (pd)**: A data manipulation and analysis library for Python.
- **train\_test\_split**: A function from scikit-learn used to split a dataset into training and testing sets.
- **LogisticRegression**: A class from scikit-learn for implementing logistic regression models.
- **accuracy\_score**: A function from scikit-learn for computing the accuracy of a classification model.

```
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
```

## ✖ Reading Sonar Dataset from CSV

- **pd.read\_csv()**: The `read_csv` function from the pandas library is used to read data from a CSV file.
  - **'/content/Copy of sonar data.csv'**: The file path specifies the location of the CSV file containing the sonar data.
  - **header=None**: Indicates that the CSV file doesn't have column headers. This is important when the data file doesn't include a row specifying the names of each column.

```
sonar_data = pd.read_csv('/content/Copy of sonar data.csv', header=None)
```

## ✖ Displaying the First Few Rows of the Sonar Dataset

The `sonar_data.head()` command is used to display the initial rows of the loaded dataset. This allows a quick overview of the structure and content of the data. Each row represents a data entry, and each column contains different features or attributes. The absence of specific column names, indicated by `header=None` during data loading, results in default numerical column indices.

```
sonar_data.head()
```

	0	1	2	3	4	5	6	7	8	9	...	51	52	53	54	55	56	5
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	0.0180	0.008
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	0.0140	0.004
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	0.0316	0.016
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	0.0050	0.004
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	0.0072	0.004

5 rows × 61 columns

The `sonar_data.shape` command is used to retrieve the dimensions of the loaded dataset, specifically the number of rows and columns. The output would be a tuple representing the dataset's shape in the form `(number_of_rows, number_of_columns)`. This information is valuable for understanding the size and structure of the dataset without revealing the actual code.

```
sonar_data.shape
```

```
(208, 61)
```

The `sonar_data.describe()` method provides descriptive statistics of the numerical columns in the dataset. These statistics include count, mean, standard deviation, minimum, 25th percentile (Q1), median (50th percentile or Q2), 75th percentile (Q3), and maximum values. This summary helps in understanding the distribution and central tendencies of the numerical features in the dataset. Without revealing the code, you can use this method in your Python environment to explore the descriptive statistics of the loaded sonar dataset.

```
sonar_data.describe()
```

1/9/24, 11:26 PMRock Vs Mine Prediction.ipynb - Colaboratory

	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	...	208.000000
mean	0.029164	0.038437	0.043832	0.053892	0.075202	0.104570	0.121747	0.134799	0.178003	0.208259	...	0.016100
std	0.022991	0.032960	0.038428	0.046528	0.055552	0.059105	0.061788	0.085152	0.118387	0.134416	...	0.012100
min	0.001500	0.000600	0.001500	0.005800	0.006700	0.010200	0.003300	0.005500	0.007500	0.011300	...	0.000000
25%	0.013350	0.016450	0.018950	0.024375	0.038050	0.067025	0.080900	0.080425	0.097025	0.111275	...	0.008000
50%	0.022800	0.030800	0.034300	0.044050	0.062500	0.092150	0.106950	0.112100	0.152250	0.182400	...	0.013000
75%	0.035550	0.047950	0.057950	0.064500	0.100275	0.134125	0.154000	0.169600	0.233425	0.268700	...	0.020000
max	0.137100	0.233900	0.305900	0.426400	0.401000	0.382300	0.372900	0.459000	0.682800	0.710600	...	0.100000

rows × 60 columns

The `sonar_data[60].value_counts()` command is used to obtain a count of unique values in the specified column of the dataset. In this case, it's assumed that the dataset contains a column with index 60. The result will display the frequency of each unique value in that particular column, which can be valuable for understanding the distribution of categorical or discrete data without revealing the actual code.

```
sonar_data[60].value_counts()

M    111
R     97
Name: 60, dtype: int64
```

The `sonar_data.groupby(60).mean()` command is likely used to group the dataset by the unique values in the column with index 60 and calculate the mean of the numerical features for each group. This is a common operation in pandas to explore the average values for different categories within a dataset. Without revealing the code, you can use this method in your Python environment to group the data by the unique values in column 60 and calculate the mean for each group.

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

	0	1	2	3	4	5	6	7	8	9	...	50	51	52
60														
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

The code below is used to create two separate datasets, `x` and `y`, from the original `sonar_data`. Here's an explanation without revealing the code:

1. Feature Matrix (X):
- `X = sonar_data.drop(columns=60, axis=1)`
  - This line creates a new dataset `x` by removing the column with index 60 from the original `sonar_data`. The resulting dataset `x` contains all the columns from the original dataset except the one with index 60. It represents the features or independent variables for further analysis or model training.
2. Target Variable (Y):
- `Y = sonar_data[60]`
  - This line creates a new dataset `y` by selecting the column with index 60 from the original `sonar_data`. The resulting dataset `y` represents the target variable or the dependent variable that you aim to predict or analyze in a machine learning context.

These two datasets, `x` and `y`, are often used in machine learning workflows where `x` contains the input features, and `y` contains the corresponding labels or target values.

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

The `print(X)` command outputs the contents of the feature matrix `x`, which is the dataset obtained by removing the column with index 60 from the original `sonar_data`.

The `print(Y)` command outputs the contents of the target variable `y`, which is the column with index 60 from the original `sonar_data`.

```
print(X)
print(Y)
```

```

      0      1      2      3      4      5      6      7      8  \
0  0.0200 0.0371 0.0428 0.0207 0.0954 0.0986 0.1539 0.1601 0.3109
1  0.0453 0.0523 0.0843 0.0689 0.1183 0.2583 0.2156 0.3481 0.3337
2  0.0262 0.0582 0.1099 0.1083 0.0974 0.2280 0.2431 0.3771 0.5598
3  0.0100 0.0171 0.0623 0.0205 0.0205 0.0368 0.1098 0.1276 0.0598
4  0.0762 0.0666 0.0481 0.0394 0.0590 0.0649 0.1209 0.2467 0.3564
..      ...      ...      ...      ...      ...      ...      ...      ...
203 0.0187 0.0346 0.0168 0.0177 0.0393 0.1630 0.2028 0.1694 0.2328
204 0.0323 0.0101 0.0298 0.0564 0.0760 0.0958 0.0990 0.1018 0.1030
205 0.0522 0.0437 0.0180 0.0292 0.0351 0.1171 0.1257 0.1178 0.1258
206 0.0303 0.0353 0.0490 0.0608 0.0167 0.1354 0.1465 0.1123 0.1945
207 0.0260 0.0363 0.0136 0.0272 0.0214 0.0338 0.0655 0.1400 0.1843

      9      ...     50     51     52     53     54     55     56  \
0  0.2111 ...  0.0232 0.0027 0.0065 0.0159 0.0072 0.0167 0.0180
1  0.2872 ...  0.0125 0.0084 0.0089 0.0048 0.0094 0.0191 0.0140
2  0.6194 ...  0.0033 0.0232 0.0166 0.0095 0.0180 0.0244 0.0316
3  0.1264 ...  0.0241 0.0121 0.0036 0.0150 0.0085 0.0073 0.0050
4  0.4459 ...  0.0156 0.0031 0.0054 0.0105 0.0110 0.0015 0.0072
..      ...      ...      ...      ...      ...      ...      ...
203 0.2684 ...  0.0203 0.0116 0.0098 0.0199 0.0033 0.0101 0.0065
204 0.2154 ...  0.0051 0.0061 0.0093 0.0135 0.0063 0.0063 0.0034
205 0.2529 ...  0.0155 0.0160 0.0029 0.0051 0.0062 0.0089 0.0140
206 0.2354 ...  0.0042 0.0086 0.0046 0.0126 0.0036 0.0035 0.0034
207 0.2354 ...  0.0181 0.0146 0.0129 0.0047 0.0039 0.0061 0.0040

      57      58      59
0  0.0084 0.0090 0.0032
1  0.0049 0.0052 0.0044
2  0.0164 0.0095 0.0078
3  0.0044 0.0040 0.0117
4  0.0048 0.0107 0.0094
..      ...      ...      ...
203 0.0115 0.0193 0.0157
204 0.0032 0.0062 0.0067
205 0.0138 0.0077 0.0031
206 0.0079 0.0036 0.0048
207 0.0036 0.0061 0.0115

[208 rows x 60 columns]
0      R
1      R
2      R
3      R
4      R
..
203    M
204    M
205    M
206    M
207    M
Name: 60, Length: 208, dtype: object
```

The code below is using the `train_test_split` function from `scikit-learn` to split the feature matrix (`x`) and the target variable (`y`) into training and testing sets. Here's an explanation without revealing the code:

- **Input:**

- `X`: Feature matrix, representing the independent variables.
- `Y`: Target variable, representing the dependent variable.
- `test_size`: The proportion of the dataset to include in the test split (in this case, 10%).
- `stratify`: Ensures that the distribution of classes in the target variable is preserved in the train-test split.
- `random_state`: Seed for reproducibility.

- **Output:**

- `X_train`: Training set of features.
- `X_test`: Testing set of features.
- `Y_train`: Training set of target values.
- `Y_test`: Testing set of target values.

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

The `print(X.shape, X_train.shape, X_test.shape)` command is used to display the shapes of the feature matrix `x`, the training set `x_train`, and the testing set `x_test`. Here's what each shape represents:

- `x.shape`: Shape of the original feature matrix `x`.
- `x_train.shape`: Shape of the training set features.

- `X_test.shape`: Shape of the testing set features.

```
print(X.shape, X_train.shape, X_test.shape)
```

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

The `print(X_train)` command outputs the contents of the training set features (`X_train`), allowing you to inspect the values and structure of the data used for training a machine learning model.

The `print(Y_train)` command outputs the contents of the training set target variable (`Y_train`), which represents the corresponding labels or values that the model aims to predict during training.

```
print(X_train)
print(Y_train)
```

	0	1	2	3	4	5	6	7	8	\
115	0.0414	0.0436	0.0447	0.0844	0.0419	0.1215	0.2002	0.1516	0.0818	
38	0.0123	0.0022	0.0196	0.0206	0.0180	0.0492	0.0033	0.0398	0.0791	
56	0.0152	0.0102	0.0113	0.0263	0.0097	0.0391	0.0857	0.0915	0.0949	
123	0.0270	0.0163	0.0341	0.0247	0.0822	0.1256	0.1323	0.1584	0.2017	
18	0.0270	0.0092	0.0145	0.0278	0.0412	0.0757	0.1026	0.1138	0.0794	
..	...	...	...	...	...	...	...	...	...	
140	0.0412	0.1135	0.0518	0.0232	0.0646	0.1124	0.1787	0.2407	0.2682	
5	0.0286	0.0453	0.0277	0.0174	0.0384	0.0990	0.1201	0.1833	0.2105	
154	0.0117	0.0069	0.0279	0.0583	0.0915	0.1267	0.1577	0.1927	0.2361	
131	0.1150	0.1163	0.0866	0.0358	0.0232	0.1267	0.2417	0.2661	0.4346	
203	0.0187	0.0346	0.0168	0.0177	0.0393	0.1630	0.2028	0.1694	0.2328	

  

	9	...	50	51	52	53	54	55	56	\
115	0.1975	...	0.0222	0.0045	0.0136	0.0113	0.0053	0.0165	0.0141	
38	0.0475	...	0.0149	0.0125	0.0134	0.0026	0.0038	0.0018	0.0113	
56	0.1504	...	0.0048	0.0049	0.0041	0.0036	0.0013	0.0046	0.0037	
123	0.2122	...	0.0197	0.0189	0.0204	0.0085	0.0043	0.0092	0.0138	
18	0.1520	...	0.0045	0.0084	0.0010	0.0018	0.0068	0.0039	0.0120	
..	...	...	...	...	...	...	...	...	...	
140	0.2058	...	0.0798	0.0376	0.0143	0.0272	0.0127	0.0166	0.0095	
5	0.3039	...	0.0104	0.0045	0.0014	0.0038	0.0013	0.0089	0.0057	
154	0.2169	...	0.0039	0.0053	0.0029	0.0020	0.0013	0.0029	0.0020	
131	0.5378	...	0.0228	0.0099	0.0065	0.0085	0.0166	0.0110	0.0190	
203	0.2684	...	0.0203	0.0116	0.0098	0.0199	0.0033	0.0101	0.0065	

  

	57	58	59
115	0.0077	0.0246	0.0198
38	0.0058	0.0047	0.0071
56	0.0011	0.0034	0.0033
123	0.0094	0.0105	0.0093
18	0.0132	0.0070	0.0088
..	...	...	...
140	0.0225	0.0098	0.0085
5	0.0027	0.0051	0.0062
154	0.0062	0.0026	0.0052
131	0.0141	0.0068	0.0086
203	0.0115	0.0193	0.0157

```
[187 rows x 60 columns]
```

```
115    M
38     R
56     R
123    M
18     R
..
140    M
5      R
154    M
131    M
203    M
```

```
Name: 60, Length: 187, dtype: object
```

The `model = LogisticRegression()` command initializes a logistic regression model using the default parameters. In this case, a logistic regression model object named `model` is created, and it is ready to be trained on the provided data.

Logistic regression is a commonly used algorithm for binary classification problems, where the target variable has two possible outcomes. This model is part of the scikit-learn library, and the `LogisticRegression()` function sets up the logistic regression model for subsequent training and predictions.

```
model = LogisticRegression()
```

The `model.fit(X_train, Y_train)` command is used to train the logistic regression model (`model`) on the provided training data. Here's an explanation without revealing the code:

- **Inputs:**

- `X_train`: The feature matrix of the training set, representing the independent variables.
- `Y_train`: The target variable of the training set, representing the dependent variable.

- **Operation:**

- The `fit` method is called on the logistic regression model (`model`), and it learns the relationships between the features in `X_train` and the corresponding target values in `Y_train`.

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

```
LogisticRegression
LogisticRegression()
```

The below code segment is used to make predictions on the training set (`X_train`) using the trained logistic regression model and then calculate the accuracy of the predictions compared to the actual target values (`Y_train`). Here's an explanation without revealing the code:

- **Prediction on Training Set:**

- `X_train_prediction = model.predict(X_train)`: The logistic regression model (`model`) is used to predict the target variable values on the training set features (`X_train`).

- **Accuracy Calculation:**

- `training_data_accuracy = accuracy_score(X_train_prediction, Y_train)`: The accuracy of the predictions on the training set is calculated by comparing the predicted values (`X_train_prediction`) with the actual target values (`Y_train`) using the `accuracy_score` function.

After executing this code, `training_data_accuracy` will contain the accuracy of the logistic regression model on the training data, providing an indication of how well the model performs on the data it was trained on.

```
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

The `print('Accuracy on training data : ', training_data_accuracy)` command is used to display the accuracy of the logistic regression model on the training data. The value of `training_data_accuracy` (calculated in the previous steps) is printed to the console.

```
print('Accuracy on training data : ', training_data_accuracy)

Accuracy on training data : 0.8342245989304813
```

The below code segment is used to make predictions on the testing set (`X_test`) using the trained logistic regression model and then calculate the accuracy of the predictions compared to the actual target values (`Y_test`). Here's an explanation without revealing the code:

- **Prediction on Testing Set:**

- `X_test_prediction = model.predict(X_test)`: The logistic regression model (`model`) is used to predict the target variable values on the testing set features (`X_test`).

- **Accuracy Calculation:**

- `test_data_accuracy = accuracy_score(X_test_prediction, Y_test)`: The accuracy of the predictions on the testing set is calculated by comparing the predicted values (`X_test_prediction`) with the actual target values (`Y_test`) using the `accuracy_score` function.

After executing this code, `test_data_accuracy` will contain the accuracy of the logistic regression model on the testing data, providing an indication of how well the model generalizes to new, unseen data.

```
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

The `print('Accuracy on test data : ', test_data_accuracy)` command is used to display the accuracy of the logistic regression model on the testing data. The value of `test_data_accuracy` (calculated in the previous steps) is printed to the console.

```
print('Accuracy on test data : ', test_data_accuracy)

Accuracy on test data : 0.7619047619047619
```

The below code segment is used to make predictions on new input data using the trained logistic regression model. Here's an explanation without revealing the code:

- **Input Data Preparation:**

- `input_data`: A set of features representing a sample input.
- `input_data_as_numpy_array`: The input data is converted to a NumPy array.
- `input_data_reshaped`: The shape of the input data is adjusted to match the expected input format of the model (reshaped to have one row and multiple columns).

- **Prediction:**

- `prediction = model.predict(input_data_reshaped)`: The logistic regression model is used to predict the target variable value for the input data.

- **Output:**

- `print(prediction)`: The predicted value (either 'R' or 'M') is printed to the console.

- **Interpretation of Prediction:**

- The code checks if the predicted value is 'R' (Rock) or 'M' (Mine) and prints a corresponding message to identify the object.

```
input_data = (0.0307,0.0523,0.0653,0.0521,0.0611,0.0577,0.0665,0.0664,0.1460,0.2792,0.3877,0.4992,0.4981,0.4972,0.5607,0.7339,0.8230,0.9
```

```
input_data_as_numpy_array = np.asarray(input_data)
```

```
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')
```

```
['M']
The object is a mine
```