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

Data Processing

```python
sonar_data = pd.read_csv('/content/Sonar_data.csv', header=None)
```

```python
sonar_data.head()
```

	0	1	2	3	4	5	6	7	8	9	...	
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.

5 rows × 61 columns

```python
sonar_data.shape
```

```
(208, 61)
```

```python
sonar_data[60].value_counts()

# here we got almost similar numbers for both the categories so we don't have to stratify the data compulsarly
# M    111
# R     97
# Name: 60, dtype: int64
```

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

Separating label and features

```python
X = sonar_data.drop(60, axis=1)
# (shape= 208, 60)
# (if we didn't specified axis above, shape would have (207, 61) by dropping 60th row)
X
```

	0	1	2	3	4	5	6	7	8	9	...
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...
207	0.0260	0.0363	0.0136	0.0272	0.0214	0.0338	0.0655	0.1400	0.1843	0.2354	...

```
y = sonar_data[60]
y.head()
# (m is mine and r is rock)
```

```
0    R
1    R
2    R
3    R
4    R
Name: 60, dtype: object
```

```
# a simple explanation of how groupby works:

# Splitting: The DataFrame is split into groups based on one or more criteria.
# Applying: A function is applied to each group independently.
# Combining: The results of the function applications are combined back into a new DataFrame.

# data = {'Category': ['A', 'B', 'A', 'B', 'A', 'B'],
#         'Value': [10, 20, 30, 40, 50, 60]}

# df = pd.DataFrame(data)

# # Grouping by 'Category'
# grouped = df.groupby('Category')

# # Calculating the mean for each group
# mean_values = grouped.mean()

# print(mean_values)

```

```
sonar_data.groupby(60).mean()
# taking mean of each grp based on grouping(categrising) clmn 60
```

	0	1	2	3	4	5	6	7	
60									
M	0.034989	0.045544	0.050720	0.064768	0.086715	0.111864	0.128359	0.149832	0.213
R	0.022498	0.030303	0.035951	0.041447	0.062028	0.096224	0.114180	0.117596	0.137

2 rows × 60 columns

```
sonar_data.groupby(60).sum()
```

	0	1	2	3	4	5	6	7	8	9
60										
M	3.8838	5.0554	5.6299	7.1892	9.6254	12.4169	14.2478	16.6314	23.6976	27.8634
R	2.1823	2.9394	3.4872	4.0204	6.0167	9.3337	11.0755	11.4068	13.3270	15.4545

2 rows × 60 columns

Splitting training and testing data

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
```

```
len(x_test)
```

```
42
```

```
len(x_train)
```

```
166
```

Model training

```
model = LogisticRegression()
```

Traing our loogistic regression model

```
model.fit(x_train, y_train)

# The .fit() method takes the feature matrix X_train and target variable y_train as arguments and fits
# the logistic regression model to the training data. During this process, the model learns the coefficients
# and intercept that define the decision boundary.

# After the model is trained using .fit(), we can use the trained model to make predictions on new data using
# the .predict() method.
```

Model Evaluation

```
# let's see the accuracy on training data first
x_train_prediction = model.predict(x_train)
```

```
x_train_accuracy = accuracy_score(x_train_prediction, y_train)
```

```
print('Accuracy on training data :', x_train_accuracy)
# Accuracy on training data : 0.8373493975903614
```

```
Accuracy on training data : 0.8373493975903614
```

```
# Accuracy on test data
x_test_prediction = model.predict(x_test)
x_test_accuracy = accuracy_score(x_test_prediction, y_test)
print('Accuracy on testing data :', x_test_accuracy)
# Accuracy on testing data : 0.8571428571428571
# Don't know how i'm getting better prediction in my test data than train data 🤷‍♀️
```

```
Accuracy on testing data : 0.8571428571428571
```

Maling a predictive system

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

```

```