

Assignment 1

Student ID: 15412372

Problem 1: Loading and exploring a dataset in Python

1. X.shape, y.shape

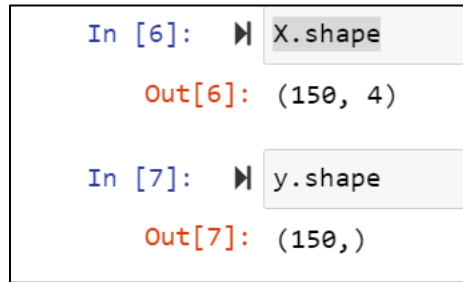
A screenshot of a Jupyter Notebook cell showing two input-output pairs. The first input is 'In [6]: X.shape' and the output is 'Out[6]: (150, 4)'. The second input is 'In [7]: y.shape' and the output is 'Out[7]: (150,)'.

Fig. result of X.shape and y.shape

X.shape has dimensions of 150x4 numpy.ndarray. Rows(150) being samples and columns(4) being the 4 features of the samples (Sepal Length, Sepal Width, Petal Length and Petal Width).

y.shape has dimensions of 150 1d-array with all the rows being samples in the data set.

2. `X[10:20, 1:3]`

```
In [17]: x1 = X[10:20, 1:3]

In [18]: x1

Out[18]: array([[3.7, 1.5],
                [3.4, 1.6],
                [3. , 1.4],
                [3. , 1.1],
                [4. , 1.2],
                [4.4, 1.5],
                [3.9, 1.3],
                [3.5, 1.4],
                [3.8, 1.7],
                [3.8, 1.5]])
```

Fig. result of `X[10:20, 1:3]`

Dimension of the above output is (10,2). Slicing a portion of x from rows 10 to 19 and columns 1 to 2. So the obtained array is subset of X.

`X[:40, 1:]` : Dimension of this is (40,3). That means rows from 0 to 39 and columns from 1 to 3.

`X[110:., :]` : Dimension of this is (40,4). That means rows from 110 to 150 and columns from 0 to 3.

3. Mean, median and standard deviation

```
In [68]: feature1 = X[:,0]
feature1

Out[68]: array([5.1, 4.9, 4.7, 4.6, 5. , 5.4, 4.6, 5. , 4.4, 4.9, 5.4, 4.8, 4.8,
 4.3, 5.8, 5.7, 5.4, 5.1, 5.7, 5.1, 5.4, 5.1, 4.6, 5.1, 4.8, 5. ,
 5. , 5.2, 5.2, 4.7, 4.8, 5.4, 5.2, 5.5, 4.9, 5. , 5.5, 4.9, 4.4,
 5.1, 5. , 4.5, 4.4, 5. , 5.1, 4.8, 5.1, 4.6, 5.3, 5. , 7. , 6.4,
 6.9, 5.5, 6.5, 5.7, 6.3, 4.9, 6.6, 5.2, 5. , 5.9, 6. , 6.1, 5.6,
 6.7, 5.6, 5.8, 6.2, 5.6, 5.9, 6.1, 6.3, 6.1, 6.4, 6.6, 6.8, 6.7,
 6. , 5.7, 5.5, 5.5, 5.8, 6. , 5.4, 6. , 6.7, 6.3, 5.6, 5.5, 5.5,
 6.1, 5.8, 5. , 5.6, 5.7, 5.7, 6.2, 5.1, 5.7, 6.3, 5.8, 7.1, 6.3,
 6.5, 7.6, 4.9, 7.3, 6.7, 7.2, 6.5, 6.4, 6.8, 5.7, 5.8, 6.4, 6.5,
 7.7, 7.7, 6. , 6.9, 5.6, 7.7, 6.3, 6.7, 7.2, 6.2, 6.1, 6.4, 7.2,
 7.4, 7.9, 6.4, 6.3, 6.1, 7.7, 6.3, 6.4, 6. , 6.9, 6.7, 6.9, 5.8,
 6.8, 6.7, 6.7, 6.3, 6.5, 6.2, 5.9])

In [69]: np.mean(feature1, axis=0)

Out[69]: 5.843333333333334

In [70]: np.median(feature1,axis=0)

Out[70]: 5.8

In [71]: np.std(feature1,axis=0)

Out[71]: 0.8253012917851409
```

Fig. Mean, median and standard deviation for feature 1 in X

```
In [73]: feature2 = X[:,1]
feature2

Out[73]: array([3.5, 3. , 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.4, 3. ,
 3. , 4. , 4.4, 3.9, 3.5, 3.8, 3.8, 3.4, 3.7, 3.6, 3.3, 3.4, 3. ,
 3.4, 3.5, 3.4, 3.2, 3.1, 3.4, 4.1, 4.2, 3.1, 3.2, 3.5, 3.6, 3. ,
 3.4, 3.5, 2.3, 3.2, 3.5, 3.8, 3. , 3.8, 3.2, 3.7, 3.3, 3.2, 3.2,
 3.1, 2.3, 2.8, 2.8, 3.3, 2.4, 2.9, 2.7, 2. , 3. , 2.2, 2.9, 2.9,
 3.1, 3. , 2.7, 2.2, 2.5, 3.2, 2.8, 2.5, 2.8, 2.9, 3. , 2.8, 3. ,
 2.9, 2.6, 2.4, 2.4, 2.7, 2.7, 3. , 3.4, 3.1, 2.3, 3. , 2.5, 2.6,
 3. , 2.6, 2.3, 2.7, 3. , 2.9, 2.9, 2.5, 2.8, 3.3, 2.7, 3. , 2.9,
 3. , 3. , 2.5, 2.9, 2.5, 3.6, 3.2, 2.7, 3. , 2.5, 2.8, 3.2, 3. ,
 3.8, 2.6, 2.2, 3.2, 2.8, 2.8, 2.7, 3.3, 3.2, 2.8, 3. , 2.8, 3. ,
 2.8, 3.8, 2.8, 2.8, 2.6, 3. , 3.4, 3.1, 3. , 3.1, 3.1, 3.1, 2.7,
 3.2, 3.3, 3. , 2.5, 3. , 3.4, 3. ])

In [74]: np.mean(feature2, axis=0)

Out[74]: 3.0573333333333337

In [75]: np.median(feature2,axis=0)

Out[75]: 3.0

In [76]: np.std(feature2,axis=0)

Out[76]: 0.4344109677354946
```

Fig. Mean, median and standard deviation for feature 2 in X

```

In [77]: feature3 = X[:,2]
         feature3

Out[77]: array([1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.6, 1.4,
                1.1, 1.2, 1.5, 1.3, 1.4, 1.7, 1.5, 1.7, 1.5, 1. , 1.7, 1.9, 1.6,
                1.6, 1.5, 1.4, 1.6, 1.6, 1.5, 1.5, 1.4, 1.5, 1.2, 1.3, 1.4, 1.3,
                1.5, 1.3, 1.3, 1.3, 1.6, 1.9, 1.4, 1.6, 1.4, 1.5, 1.4, 4.7, 4.5,
                4.9, 4. , 4.6, 4.5, 4.7, 3.3, 4.6, 3.9, 3.5, 4.2, 4. , 4.7, 3.6,
                4.4, 4.5, 4.1, 4.5, 3.9, 4.8, 4. , 4.9, 4.7, 4.3, 4.4, 4.8, 5. ,
                4.5, 3.5, 3.8, 3.7, 3.9, 5.1, 4.5, 4.5, 4.7, 4.4, 4.1, 4. , 4.4,
                4.6, 4. , 3.3, 4.2, 4.2, 4.2, 4.3, 3. , 4.1, 6. , 5.1, 5.9, 5.6,
                5.8, 6.6, 4.5, 6.3, 5.8, 6.1, 5.1, 5.3, 5.5, 5. , 5.1, 5.3, 5.5,
                6.7, 6.9, 5. , 5.7, 4.9, 6.7, 4.9, 5.7, 6. , 4.8, 4.9, 5.6, 5.8,
                6.1, 6.4, 5.6, 5.1, 5.6, 6.1, 5.6, 5.5, 4.8, 5.4, 5.6, 5.1, 5.1,
                5.9, 5.7, 5.2, 5. , 5.2, 5.4, 5.1])

In [78]: np.mean(feature3)

Out[78]: 3.7580000000000005

In [79]: np.median(feature3)

Out[79]: 4.35

In [80]: np.std(feature3)

Out[80]: 1.759404065775303

```

Fig. Mean, median and standard deviation for feature 3 in X

```

In [81]: feature4 = X[:,3]
         feature4

Out[81]: array([0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.2, 0.1,
                0.1, 0.2, 0.4, 0.4, 0.3, 0.3, 0.3, 0.2, 0.4, 0.2, 0.5, 0.2, 0.2,
                0.4, 0.2, 0.2, 0.2, 0.2, 0.4, 0.1, 0.2, 0.2, 0.2, 0.2, 0.1, 0.2,
                0.2, 0.3, 0.3, 0.2, 0.6, 0.4, 0.3, 0.2, 0.2, 0.2, 0.2, 1.4, 1.5,
                1.5, 1.3, 1.5, 1.3, 1.6, 1. , 1.3, 1.4, 1. , 1.5, 1. , 1.4, 1.3,
                1.4, 1.5, 1. , 1.5, 1.1, 1.8, 1.3, 1.5, 1.2, 1.3, 1.4, 1.4, 1.7,
                1.5, 1. , 1.1, 1. , 1.2, 1.6, 1.5, 1.6, 1.5, 1.3, 1.3, 1.3, 1.2,
                1.4, 1.2, 1. , 1.3, 1.2, 1.3, 1.3, 1.1, 1.3, 2.5, 1.9, 2.1, 1.8,
                2.2, 2.1, 1.7, 1.8, 1.8, 2.5, 2. , 1.9, 2.1, 2. , 2.4, 2.3, 1.8,
                2.2, 2.3, 1.5, 2.3, 2. , 2. , 1.8, 2.1, 1.8, 1.8, 1.8, 2.1, 1.6,
                1.9, 2. , 2.2, 1.5, 1.4, 2.3, 2.4, 1.8, 1.8, 2.1, 2.4, 2.3, 1.9,
                2.3, 2.5, 2.3, 1.9, 2. , 2.3, 1.8])

In [82]: np.mean(feature4,axis=0)

Out[82]: 1.1993333333333336

In [83]: np.median(feature4,axis=0)

Out[83]: 1.3

In [84]: np.std(feature4,axis=0)

Out[84]: 0.7596926279021594

```

Fig. Mean, median and standard deviation for feature 4 in X

4. Plot the histograms

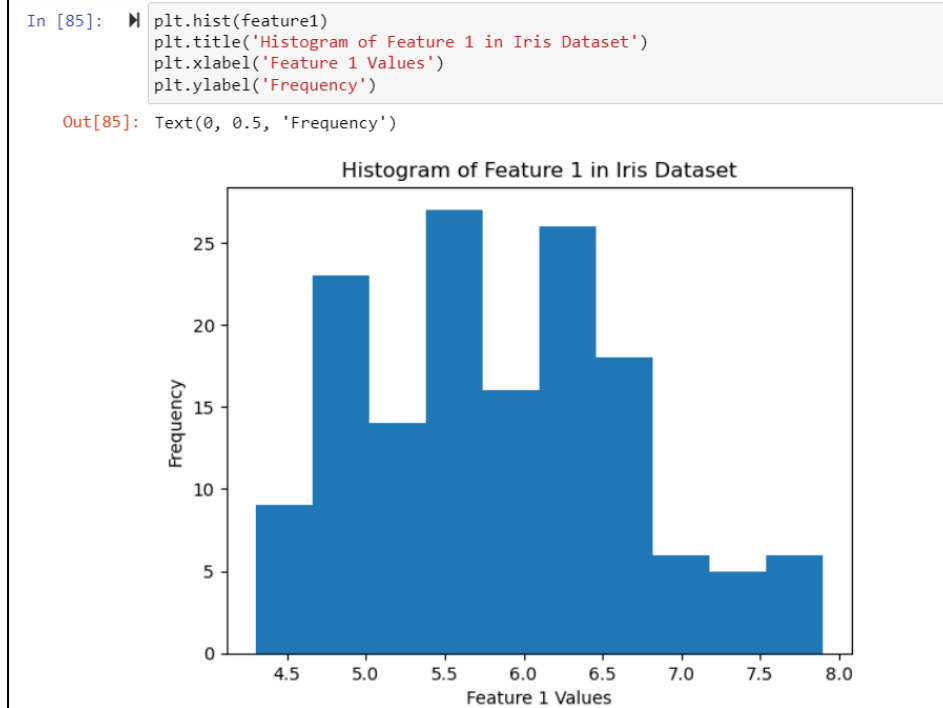


Fig. Histogram plot of feature 1 in X

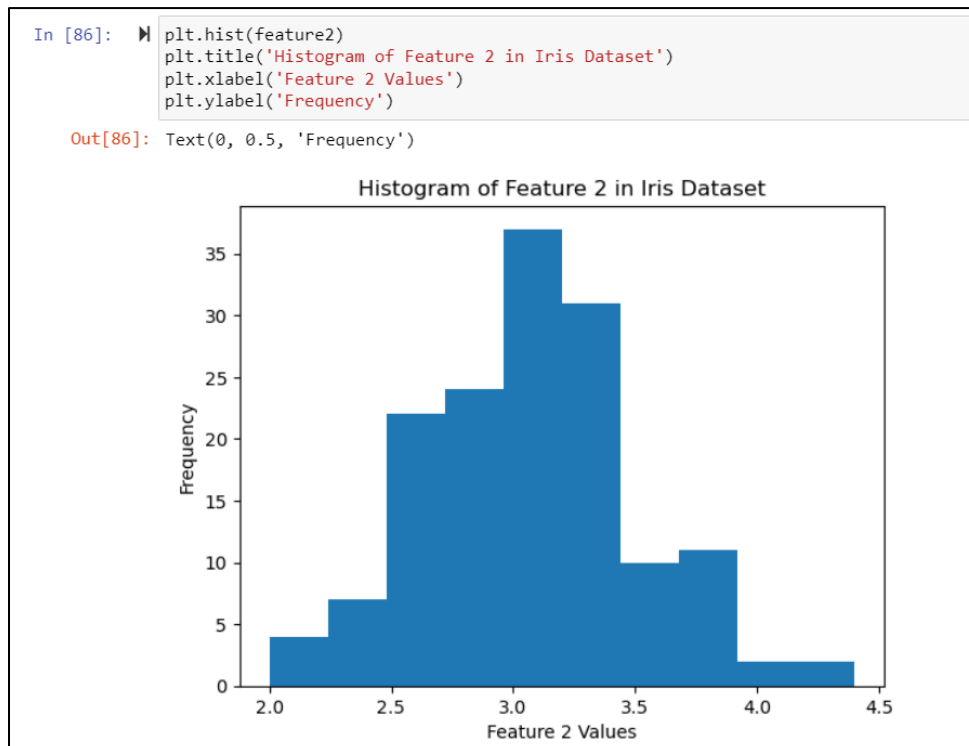


Fig. Histogram plot of feature 2 in X

```
In [87]: plt.hist(feature3)
plt.title('Histogram of Feature 3 in Iris Dataset')
plt.xlabel('Feature 3 Values')
plt.ylabel('Frequency')
```

```
Out[87]: Text(0, 0.5, 'Frequency')
```

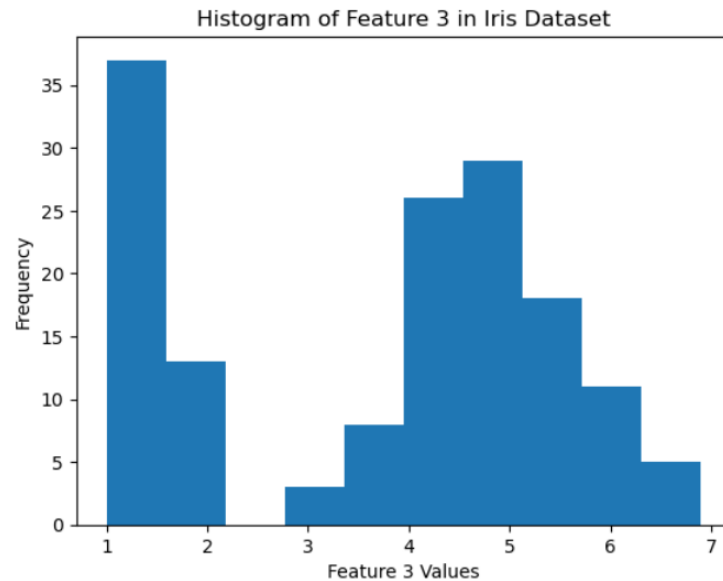


Fig: Histogram plot of feature 3 in X

```
In [88]: plt.hist(feature4)
plt.title('Histogram of Feature 4 in Iris Dataset')
plt.xlabel('Feature 4 Values')
plt.ylabel('Frequency')
```

```
Out[88]: Text(0, 0.5, 'Frequency')
```

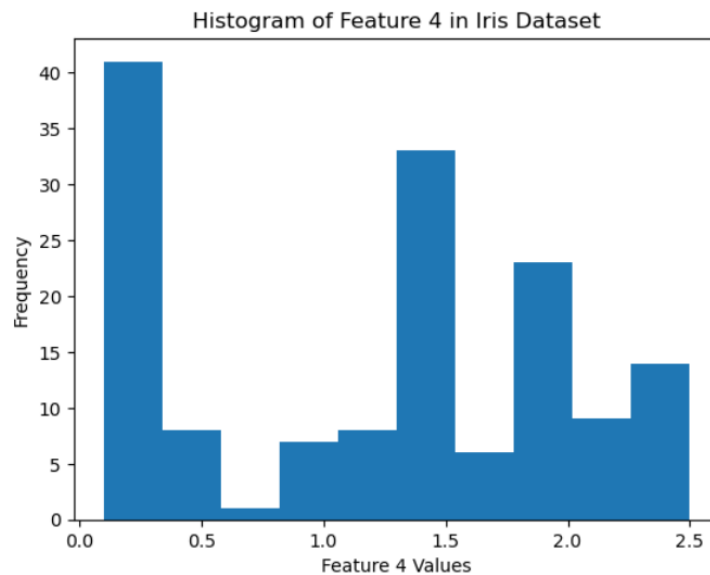


Fig: Histogram plot of feature 4 in X

5. Scatter plot of features

```
In [89]: plt.scatter(feature1, feature3, c=y, cmap='viridis', edgecolors='k')
plt.xlabel('Feature 1')
plt.ylabel('Feature 3')
plt.title('Scatter Plot of Features 1 and 3 with Class Colors')

Out[89]: Text(0.5, 1.0, 'Scatter Plot of Features 1 and 3 with Class Colors')
```

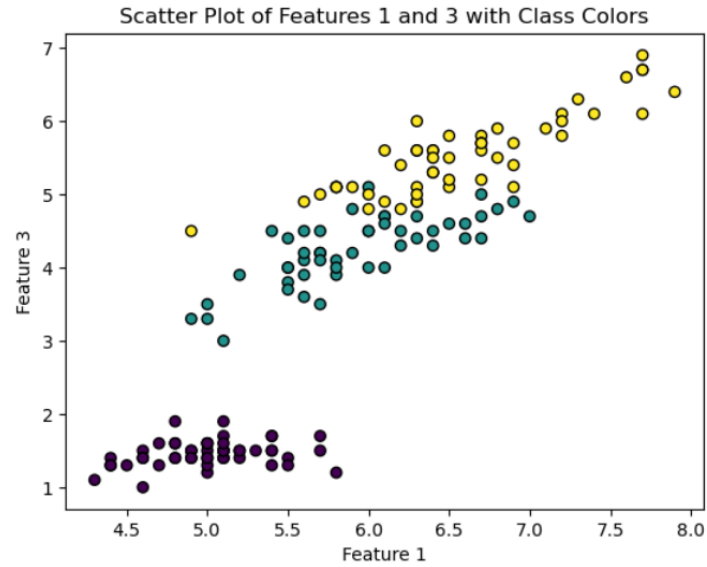


Fig. Scatter plot of feature 1 and feature 3

```
In [90]: plt.scatter(feature2, feature4, c=y, cmap='viridis', edgecolors='k')
plt.xlabel('Feature 2')
plt.ylabel('Feature 3')
plt.title('Scatter Plot of Features 2 and 4 with Class Colors')

Out[90]: Text(0.5, 1.0, 'Scatter Plot of Features 2 and 4 with Class Colors')
```

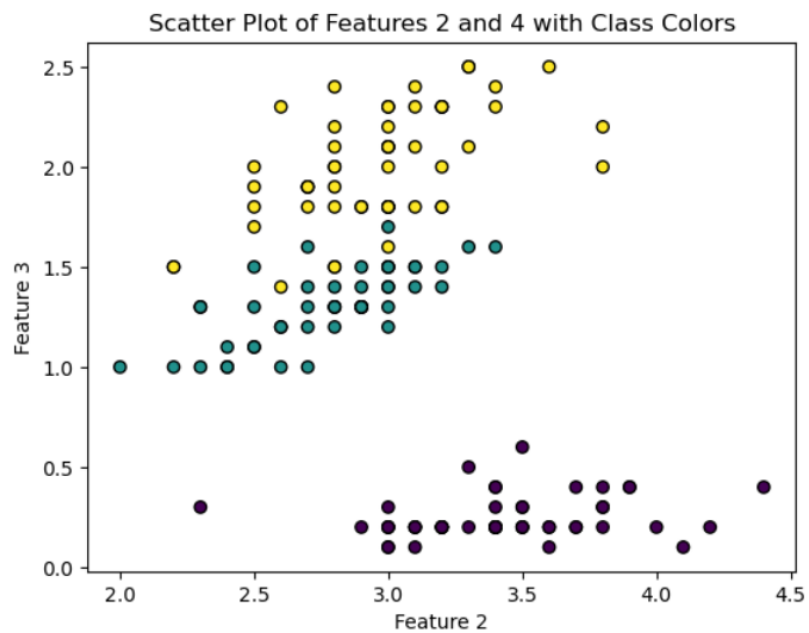


Fig. Scatter plot of feature 2 and feature 4

Problem 2: Logistic Regression

1. Selecting part of the data

```
In [181]: X = iris.data[iris.target < 2]

In [182]: y = iris.target[iris.target < 2]

In [183]: num_samples = len(X)
           print(f"Number of samples for classes 0 and 1: {num_samples}")
           Number of samples for classes 0 and 1: 100
```

Fig. Selecting class 0 and 1

Number of samples for class 0 and 1 is 100

2. Scaling

```
In [157]: scaler = StandardScaler()
In [158]: X_scaled = scaler.fit_transform(X)
```

Fig. Scaling X vectors

The `StandardScaler` class from the preprocessing module in scikit-learn is used for standardizing features by removing the mean and scaling to unit variance. It is part of the preprocessing techniques applied to the input data before feeding it into a machine learning model.

1. **Mean Removal:** The `StandardScaler` centers the data by removing the mean. For each feature, it subtracts the mean of that feature from all the values. This ensures that the transformed data has a mean of zero.
2. **Scaling to Unit Variance:** After centering the data, the scaler scales each feature to have unit variance. It divides each feature by its standard deviation. This standardizes the range of the transformed values and ensures that all features contribute equally to the model training.

The scaling formula applied to each feature x is given by:

$$\text{Scaled value} = (x - \text{mean}) / \text{standard deviation}$$

3. Train test data set

Splitting the data set with 80%-20% as training and test data

```
In [186]: X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=15412372)

In [188]: print("Size of X Training Set:", X_train.shape)
           print("Size of X Test Set:", X_test.shape)
           print("Size of y Training Set:", y_train.shape)
           print("Size of y Test Set:", y_test.shape)

           Size of X Training Set: (80, 4)
           Size of X Test Set: (20, 4)
           Size of y Training Set: (80,)
           Size of y Test Set: (20,)
```

Fig. Test, Train data set

Size of X training data set = (80,4)

Size of X test data set = (20,4)

Size of y training data set = (80,)

Size of y test data set = (20,)

4. Logistic regression with the training data

```
In [161]: LR_Classifier = LogisticRegression()

In [162]: LR_Classifier.fit(X_train, y_train)

Out[162]: LogisticRegression
LogisticRegression()

In [163]: coefficients = LR_Classifier.coef_
print("Coefficients:")
print(coefficients)

Coefficients:
[[ 0.75430386 -1.19214376  1.42432095  1.46709018]]
```

Fig. Training data

Coefficients:

```
[[ 0.75430386 -1.19214376  1.42432095  1.46709018]]
```

Explanation:

Each coefficient corresponds to a feature in your dataset. They represent the change in the log-odds of the response variable for a one-unit change in the predictor variable. Positive coefficients indicate a positive relationship with the response (increases the odds), and negative coefficients indicate a negative relationship (decreases the odds).

5. Trained model on test data

```
In [164]: ▶ y_predicted = LR_Classifier.predict(X_test)
```

```
In [165]: ▶ y_train_predicted = LR_Classifier.predict(X_train)
```

```
In [166]: ▶ print( classification_report(y_test, y_predicted))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	11
1	1.00	1.00	1.00	9
accuracy			1.00	20
macro avg	1.00	1.00	1.00	20
weighted avg	1.00	1.00	1.00	20

```
In [167]: ▶ print( accuracy_score(y_test, y_predicted))
```

1.0

Fig. Test data and performance computation

6. Repeat steps 1-5 with all the classes of flowers

```

In [168]: X1 = iris.data

In [169]: y1 = iris.target

In [170]: #scaler = StandardScaler()
X1_scaled = scaler.fit_transform(X1)

In [171]: X1_train, X1_test, y1_train, y1_test = train_test_split(X1_scaled, y1, test_size=0.2, random_state=15412372)

In [172]: print("Size of Training Set:", X1_train.shape)
print("Size of Test Set:", X1_test.shape)
Size of Training Set: (120, 4)
Size of Test Set: (30, 4)

In [173]: LR_Classifier.fit(X1_train, y1_train)

Out[173]: LogisticRegression
LogisticRegression()

In [174]: coefficients = LR_Classifier.coef_
print("Coefficients:")
print(coefficients)
Coefficients:
[[-1.01486925  1.1084658 -1.84383687 -1.72411295]
 [ 0.51131276 -0.31123113 -0.28950265 -0.72033397]
 [ 0.50355649 -0.79723467  2.13333952  2.44444692]]

In [175]: y1_predicted = LR_Classifier.predict(X1_test)

In [176]: y1_train_predicted = LR_Classifier.predict(X1_train)

In [177]: print(classification_report(y1_test, y1_predicted))

              precision    recall  f1-score   support

     0             1.00      1.00      1.00        10
     1             1.00      0.88      0.93         8
     2             0.92      1.00      0.96        12

   accuracy              0.97        30
  macro avg              0.97      0.96      0.96        30
 weighted avg              0.97      0.97      0.97        30

In [178]: print(accuracy_score(y1_test, y1_predicted))

0.9666666666666667

```

Fig. Logistic regression on all the classes

Coefficients:

```

[[-1.01486925  1.1084658 -1.84383687 -1.72411295]
 [ 0.51131276 -0.31123113 -0.28950265 -0.72033397]
 [ 0.50355649 -0.79723467  2.13333952  2.44444692]]

```

The matrix where each row represents a class, and each column represents a feature. The values in this matrix indicate the importance of each feature for predicting each class. Positive coefficients indicate a positive relationship with the response (increases the odds), and negative coefficients indicate a negative relationship (decreases the odds).

The accuracy obtained for all the 3 classes is 0.9667 (96.67%).