Logistic Regression Model for Iris Dataset

In [21]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# For model training and evaluation
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f
from sklearn.metrics import confusion_matrix, classification_report
```

Load and Explore the Iris Dataset

Load the Iris dataset using sklearn's datasets module and explore its structure, including feature names and target classes.

In [22]:
```
# Load the Iris dataset
iris = load_iris()

# Create a DataFrame for better visualization
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target

# Map target names for better understanding
target_names = {i: name for i, name in enumerate(iris.target_names)}
iris_df['target_name'] = iris_df['target'].map(target_names)

# Display the first few rows of the dataset
print("First 5 rows of the dataset:")
print(iris_df.head())
```

```
First 5 rows of the dataset:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target target_name
0       0      setosa
1       0      setosa
2       0      setosa
3       0      setosa
4       0      setosa
```

Split the Dataset into Training and Testing Sets

Use train_test_split from sklearn to divide the dataset into training and testing sets.

In [23]:

```
# Define features X and target y
X = iris.data
y = iris.target

# Split the dataset into training (70%) and testing (30%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rar

# Check the shape of training and testing sets
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")

# Confirm that the class distribution is maintained in both sets
print("\nClass distribution in y:")
print(np.bincount(y))

print("\nClass distribution in y_train:")
print(np.bincount(y_train))

print("\nClass distribution in y_test:")
print(np.bincount(y_test))
```

```
X_train shape: (105, 4)
X_test shape: (45, 4)
y_train shape: (105,)
y_test shape: (45,)

Class distribution in y:
[50 50 50]

Class distribution in y_train:
[35 35 35]

Class distribution in y_test:
[15 15 15]
```

Train a Logistic Regression Model

Use sklearn's LogisticRegression to train a model on the training data.

In [24]:

```
# Initialize the logistic regression model
# Using 'multinomial' solver because we have more than 2 classes
logistic_model = LogisticRegression(multi_class='multinomial', solver='lbfgs

# Train the model on the training data
logistic_model.fit(X_train, y_train)

# Print the model coefficients
print("Model coefficients:")
for i, feature_name in enumerate(iris.feature_names):
    print(f"{feature_name}: {logistic_model.coef_[:, i]}")

print("\nIntercept values:")
print(logistic_model.intercept_)

# Get the probability estimates for training data
y_train_proba = logistic_model.predict_proba(X_train)

# Print probability estimates for first 5 samples
print("\nProbability estimates for first 5 training samples:")
for i in range(5):
    print(f"Sample {i+1}: {y_train_proba[i]} (Actual class: {y_train[i]})")
```

```
Model coefficients:
sepal length (cm): [-0.54508414  0.42116822  0.12391592]
sepal width (cm): [ 0.76445112 -0.42711735 -0.33733377]
petal length (cm): [-2.22894098 -0.10046531  2.32940629]
petal width (cm): [-0.97457748 -0.83878104  1.81335852]

Intercept values:
[  9.92774954   2.42287865 -12.35062819]

Probability estimates for first 5 training samples:
Sample 1: [0.27254453 0.72538322 0.00207225] (Actual class: 1)
Sample 2: [0.00288155 0.81615596 0.18096249] (Actual class: 1)
Sample 3: [9.79597444e-01 2.04023506e-02 2.05058225e-07] (Actual class: 0)
Sample 4: [5.72254040e-06 1.92906759e-02 9.80703602e-01] (Actual class: 2)
Sample 5: [0.02587304 0.9186112  0.05551576] (Actual class: 1)
```

```
c:\Users\admi\AppData\Local\Programs\Python\Python311\Lib\site-packages\skle
arn\linear_model\_logistic.py:1247: FutureWarning: 'multi_class' was depreca
ted in version 1.5 and will be removed in 1.7. From then on, it will always
use 'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
```

Make Predictions

Use the trained model to make predictions on the testing data.

In [25]:
```python
# Make predictions on the test set
y_pred = logistic_model.predict(X_test)

# Get probability estimates for test data
y_test_proba = logistic_model.predict_proba(X_test)

# Create a DataFrame to compare actual vs predicted values
results_df = pd.DataFrame({
    'Actual': [iris.target_names[i] for i in y_test],
    'Predicted': [iris.target_names[i] for i in y_pred]
})

# Display the first 10 predictions
print("First 10 predictions:")
print(results_df.head(10))

# Count the number of correct and incorrect predictions
correct = (y_test == y_pred).sum()
incorrect = (y_test != y_pred).sum()
total = len(y_test)

print(f"\nCorrect predictions: {correct} ({correct/total*100:.2f}%)")
print(f"Incorrect predictions: {incorrect} ({incorrect/total*100:.2f}%)")
```

```
First 10 predictions:
       Actual   Predicted
0   virginica   virginica
1  versicolor  versicolor
2   virginica  versicolor
3  versicolor  versicolor
4   virginica   virginica
5   virginica   virginica
6  versicolor  versicolor
7  versicolor  versicolor
8      setosa      setosa
9   virginica   virginica

Correct predictions: 42 (93.33%)
Incorrect predictions: 3 (6.67%)
```

Evaluate the Model

Calculate and display metrics such as accuracy, precision_score, recall_score, and f1_score using sklearn's metrics module.

In [26]:
```
# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Display the metrics
print("Model Performance Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")

# Generate a classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

```
Model Performance Metrics:
Accuracy: 0.9333
Precision: 0.9345
Recall: 0.9333
F1 Score: 0.9333

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.88      0.93      0.90        15
   virginica       0.93      0.87      0.90        15

    accuracy                           0.93        45
   macro avg       0.93      0.93      0.93        45
weighted avg       0.93      0.93      0.93        45
```