In [7]:

```
import numpy as np
```

In [8]:

```
class Perceptron:
    def __init__(self, input_size):
        self.weights = np.zeros(input_size + 1)

    def predict(self, inputs):
        summation = np.dot(inputs, self.weights[1:]) + self.weights[0]
        return 1 if summation > 0 else 0

    def train(self, training_inputs, labels, learning_rate, epochs):
        for _ in range(epochs):
            for inputs, label in zip(training_inputs, labels):
                prediction = self.predict(inputs)
                self.weights[1:] += learning_rate * (label - prediction) * inputs
                self.weights[0] += learning_rate * (label - prediction)
```

In [9]:

```
# Training the Perceptron for AND function
and_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
and_labels = np.array([0, 0, 0, 1])
and_perceptron = Perceptron(input_size=2)
and_perceptron.train(and_inputs, and_labels, learning_rate=0.1, epochs=10)

# Testing the AND Perceptron
print("AND Function:")
for inputs in and_inputs:
    print(f"Inputs: {inputs}, Output: {and_perceptron.predict(inputs)}")
```

```
AND Function:
Inputs: [0 0], Output: 0
Inputs: [0 1], Output: 0
Inputs: [1 0], Output: 0
Inputs: [1 1], Output: 1
```

In [10]:

```
# Training the Perceptron for OR function
or_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
or_labels = np.array([0, 1, 1, 1])
or_perceptron = Perceptron(input_size=2)
or_perceptron.train(or_inputs, or_labels, learning_rate=0.1, epochs=10)

# Testing the OR Perceptron
print("OR Function:")
for inputs in or_inputs:
    print(f"Inputs: {inputs}, Output: {or_perceptron.predict(inputs)}")
```

```
OR Function:
Inputs: [0 0], Output: 0
Inputs: [0 1], Output: 1
Inputs: [1 0], Output: 1
Inputs: [1 1], Output: 1
```

In [11]:

```
# Testing the OR Perceptron
print("OR Function:")
for inputs in or_inputs:
    print(f"Inputs: {inputs}, Output: {or_perceptron.predict(inputs)}")
```

```
OR Function:
Inputs: [0 0], Output: 0
Inputs: [0 1], Output: 1
Inputs: [1 0], Output: 1
Inputs: [1 1], Output: 1
```

In [12]:

```
# Testing the NAND Perceptron
print("NAND Function:")
for inputs in nand_inputs:
    print(f"Inputs: {inputs}, Output: {nand_perceptron.predict(inputs)}")
```

```
NAND Function:
Inputs: [0 0], Output: 1
Inputs: [0 1], Output: 1
Inputs: [1 0], Output: 1
Inputs: [1 1], Output: 0
```

In [13]:

```
# Training the Perceptron for XOR function
xor_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
xor_labels = np.array([0, 1, 1, 0])
xor_perceptron = Perceptron(input_size=2)
xor_perceptron.train(xor_inputs, xor_labels, learning_rate=0.1, epochs=10)

# Testing the XOR Perceptron
print("XOR Function:")
for inputs in xor_inputs:
    print(f"Inputs: {inputs}, Output: {xor_perceptron.predict(inputs)}")
```

```
XOR Function:
Inputs: [0 0], Output: 1
Inputs: [0 1], Output: 1
Inputs: [1 0], Output: 0
Inputs: [1 1], Output: 0
```

In [14]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
```

In [15]:

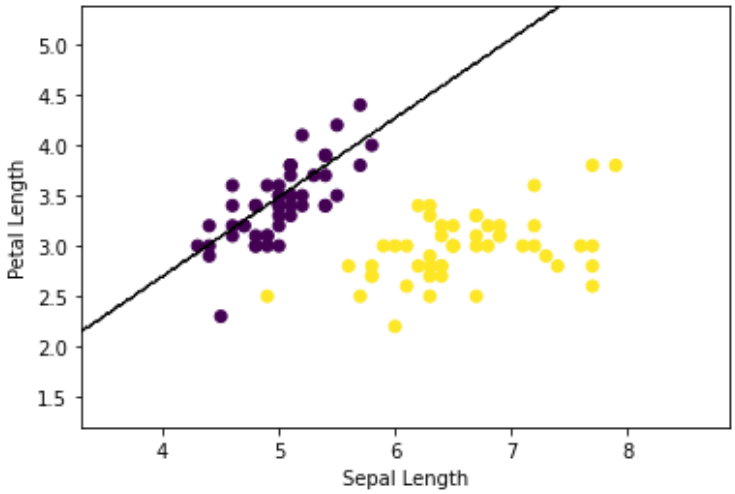
```
# Step 1: Download the Iris dataset
iris = datasets.load_iris()
X = iris.data[:, :2]  # Selecting only Sepal Length and Petal Length features
y = iris.target

# Preparing a separate dataset for Iris-setosa and Iris-virginica classes only
X_subset = X[(y == 0) | (y == 2)]
y_subset = y[(y == 0) | (y == 2)]
y_subset = np.where(y_subset == 0, -1, 1)  # Labeling Iris-setosa as -1 and Iris-virginica as
```

In [16]:

```
# Step 2: Training a Perceptron model
class Perceptron:
    def __init__(self, input_size):
        self.weights = np.zeros(input_size + 1)

    def predict(self, inputs):
        summation = np.dot(inputs, self.weights[1:]) + self.weights[0]
        return np.where(summation > 0, 1, -1)

    def train(self, training_inputs, labels, learning_rate, epochs):
        for _ in range(epochs):
            for inputs, label in zip(training_inputs, labels):
                prediction = self.predict(inputs)
                self.weights[1:] += learning_rate * (label - prediction) * inputs
                self.weights[0] += learning_rate * (label - prediction)

perceptron = Perceptron(input_size=2)
perceptron.train(X_subset, y_subset, learning_rate=0.1, epochs=100)
```

In [17]:

```
# Step 3: Plotting the scatter plot and decision boundary
plt.scatter(X_subset[:, 0], X_subset[:, 1], c=y_subset)
plt.xlabel('Sepal Length')
plt.ylabel('Petal Length')

x_min, x_max = X_subset[:, 0].min() - 1, X_subset[:, 0].max() + 1
y_min, y_max = X_subset[:, 1].min() - 1, X_subset[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))
Z = perceptron.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, colors='k', linewidths=0.5)
plt.show()
```



In [ ]: