In [1]:
```python
#LogisticRegression
```

In [2]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.model_selection import train_test_split
```

In [3]:
```python
iris = datasets.load_iris()
```

In [4]:
```python
X = iris.data[:100, :2]  # We only take the first two features.
y = iris.target[:100]  # We take the first 100 instances, which only belong
```

In [15]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran
len(X_train) , len(X_test)
```

Out[15]:
```
(80, 20)
```

In [6]:
```python
model = LogisticRegression()
```

In [7]:
```python
model.fit(X_train, y_train)
```

Out[7]:
```
▾ LogisticRegression ⓘ ?

▸ Parameters
```

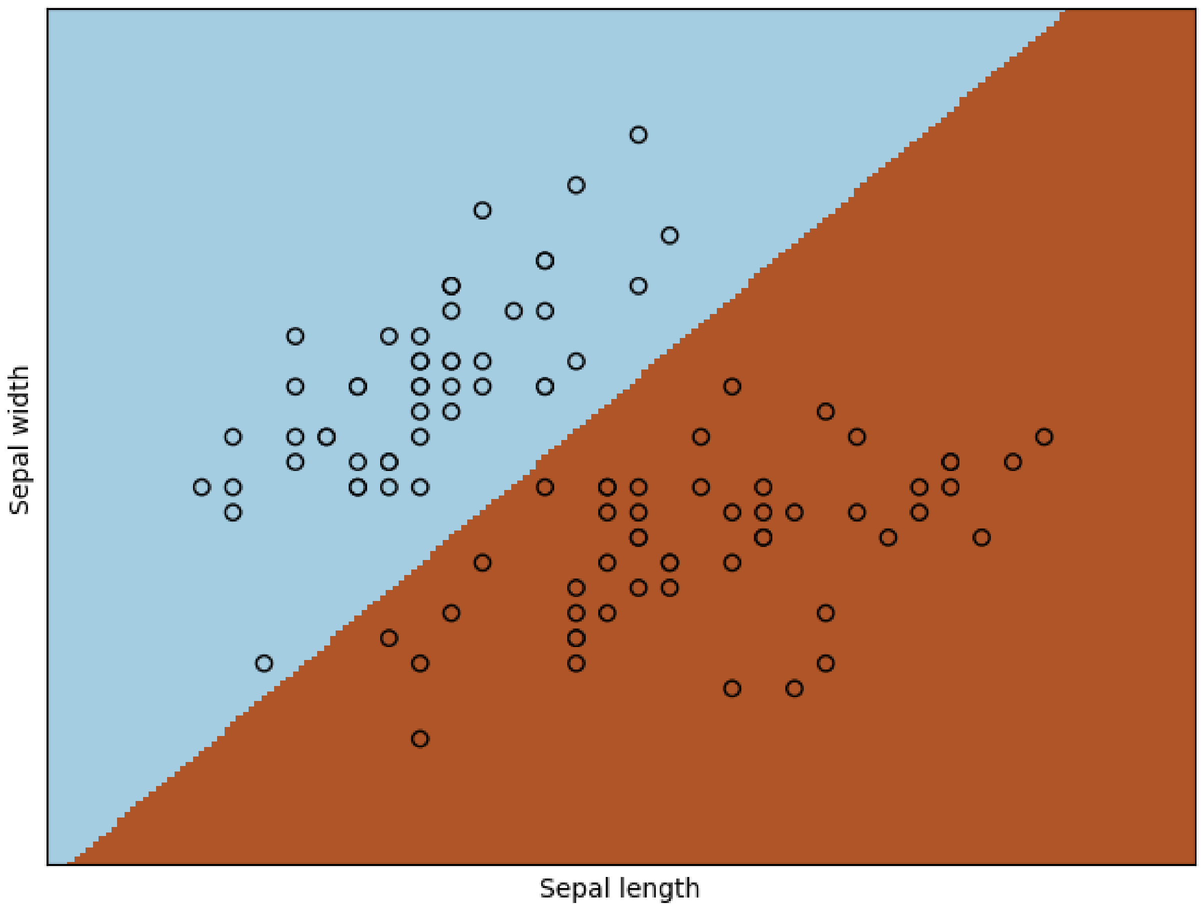
In [8]:
```python
print("Coefficients: ", model.coef_)
print("Intercept: ", model.intercept_)
```

```
Coefficients:  [[ 2.88998626 -2.72779317]]
Intercept:  [-7.09121494]
```

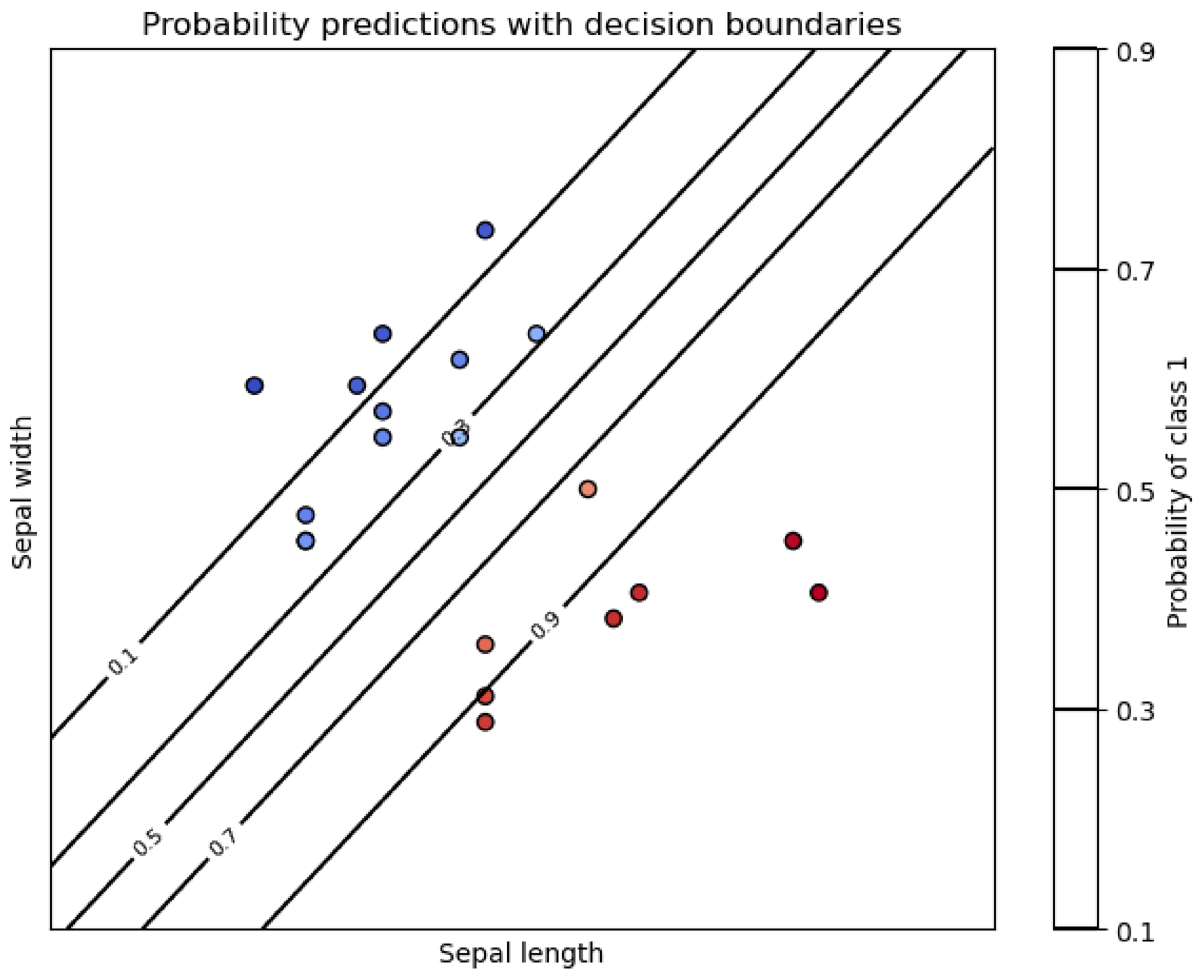
In [18]:
```python
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)
y_pred
y_prob #If the predicted probability for class 0 is greater than 0.5, the mo
```

```
Out[18]: array([[0.05296719, 0.94703281],
                [0.07379633, 0.92620367],
                [0.22604741, 0.77395259],
                [0.80260571, 0.19739429],
                [0.9380746 , 0.0619254 ],
                [0.8357283 , 0.1642717 ],
                [0.9738481 , 0.0261519 ],
                [0.09474686, 0.90525314],
                [0.82893864, 0.17106136],
                [0.86984184, 0.13015816],
                [0.72788282, 0.27211718],
                [0.84229973, 0.15770027],
                [0.05215948, 0.94784052],
                [0.93419699, 0.06580301],
                [0.15297609, 0.84702391],
                [0.92138671, 0.07861329],
                [0.00722562, 0.99277438],
                [0.01649098, 0.98350902],
                [0.80260571, 0.19739429],
                [0.68130725, 0.31869275]])
```

```
In [10]: x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
h = .02  # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(8, 6))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```



```
In [11]: # Step 11 - Plotting the probabilities with contour lines
         plt.figure(2, figsize=(8, 6))

         # Probabilities of class 1 for each point in the grid
         Z_prob = model.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]
         Z_prob = Z_prob.reshape(xx.shape)

         # Scatter plot of the test data, colored by the probability of class 1
         plt.scatter(X_test[:, 0], X_test[:, 1], c=y_prob[:, 1], edgecolors='k', cmap

         # Contour plot for levels of probabilities
         contour = plt.contour(xx, yy, Z_prob, levels=[0.1, 0.3, 0.5, 0.7, 0.9], colo
         plt.clabel(contour, inline=True, fontsize=8)

         plt.xlabel('Sepal length')
         plt.ylabel('Sepal width')
         plt.colorbar(label='Probability of class 1')
         plt.title('Probability predictions with decision boundaries')

         plt.xlim(xx.min(), xx.max())
         plt.ylim(yy.min(), yy.max())
         plt.xticks(())
         plt.yticks(())

Out[11]: ([], [])
```



```
In [12]: from sklearn.metrics import accuracy_score, confusion_matrix

         accuracy = accuracy_score(y_test, y_pred)
         print('Accuracy:', accuracy)
```

```
Accuracy: 1.0
```

```
In [13]: cm = confusion_matrix(y_test, y_pred)
         print("Confusion Matrix: \n", cm)
```

```
Confusion Matrix:
 [[12  0]
 [ 0  8]]
```