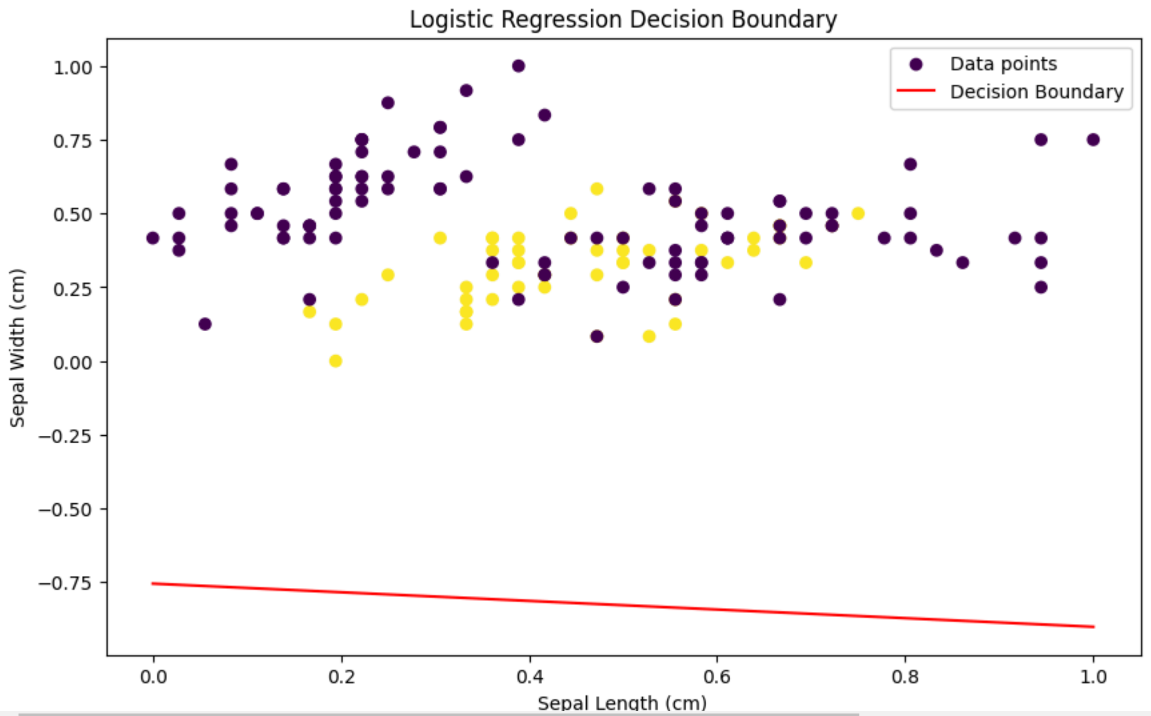
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('p_iris.csv')

# Extract features and target variable
# Assuming you want to predict Species_1 or Species_2
# Let's use 'Species_1' for binary classification
X = data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']].values
y = data['Species_1'].astype(int).values  # Convert boolean to integer (0 or 1)

# Define the sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Define the cost function (binary cross-entropy)
def cost_function(y, y_pred):
    return -np.mean(y * np.log(y_pred + 1e-10) + (1 - y) * np.log(1 - y_pred + 1e-10))

# Implementing Logistic Regression using Gradient Descent
def logistic_regression(X, y, learning_rate=0.01, iterations=1000):
    m, n = X.shape
    weights = np.zeros(n)  # Initialize weights
    bias = 0  # Initialize bias

    for _ in range(iterations):
        # Linear combination
        linear_model = np.dot(X, weights) + bias
        # Apply sigmoid function
        y_pred = sigmoid(linear_model)

        # Compute gradients
        dw = (1/m) * np.dot(X.T, (y_pred - y))  # Gradient with respect to weights
        db = (1/m) * np.sum(y_pred - y)  # Gradient with respect to bias

        # Update weights and bias
        weights -= learning_rate * dw
        bias -= learning_rate * db

        # Optionally print the cost every 100 iterations
        if _ % 100 == 0:
            cost = cost_function(y, y_pred)
            print(f"Cost after iteration {_}: {cost}")

    return weights, bias

# Train the model
weights, bias = logistic_regression(X, y)

print(f"Trained weights: {weights}")
print(f"Trained bias: {bias}")

# Make predictions
def predict(X, weights, bias):
    linear_model = np.dot(X, weights) + bias
    y_pred = sigmoid(linear_model)
    return [1 if i >= 0.5 else 0 for i in y_pred]

# Get predictions
y_pred = predict(X, weights, bias)

# Plotting the results (only if you are using 2 features)
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', label='Data points')
x_values = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
y_values = -(weights[0] * x_values + bias) / weights[1]  # Decision boundary
plt.plot(x_values, y_values, color='red', label='Decision Boundary')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.title('Logistic Regression Decision Boundary')
plt.legend()
plt.show()
```

```
Cost after iteration 0: 0.6931471803599453
Cost after iteration 100: 0.65922578838218
Cost after iteration 200: 0.6438176806245366
Cost after iteration 300: 0.6357462632668022
Cost after iteration 400: 0.6307088229397281
Cost after iteration 500: 0.627005773448819
Cost after iteration 600: 0.6239441545051513
Cost after iteration 700: 0.6212320977419205
Cost after iteration 800: 0.6187417998431392
Cost after iteration 900: 0.6164140176717633
Trained weights: [-0.08170087 -0.55821476  0.09061479 -0.00896527]
Trained bias: -0.4212070132659844
```


Logistic Regression Decision Boundary
Data points
Decision Boundary
Sepal Width (cm)
Sepal Length (cm)