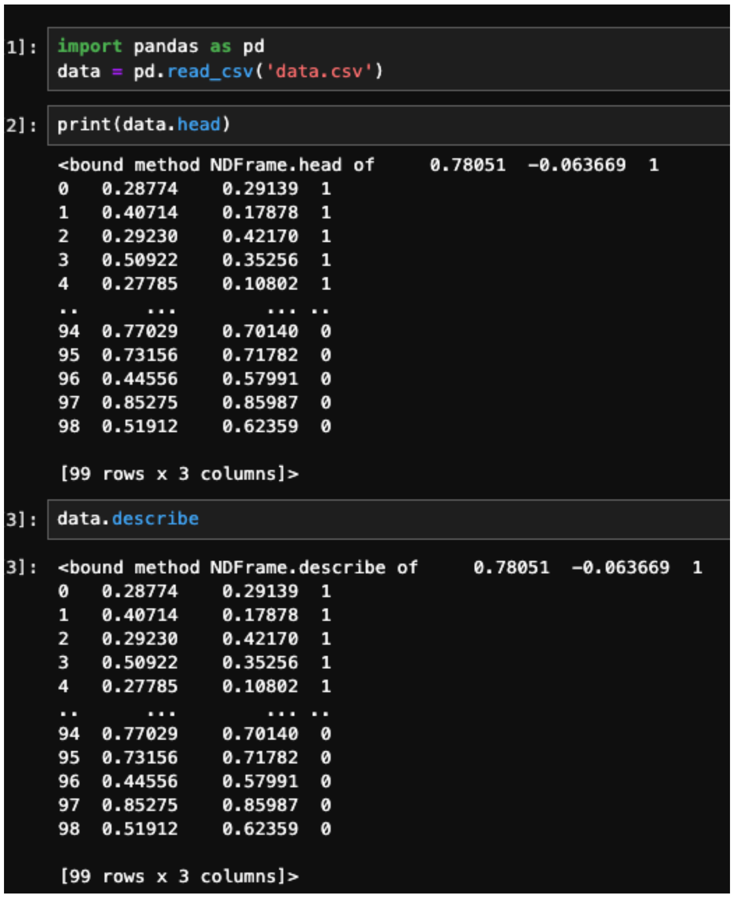
ASSIGNMENT 5: Perceptron

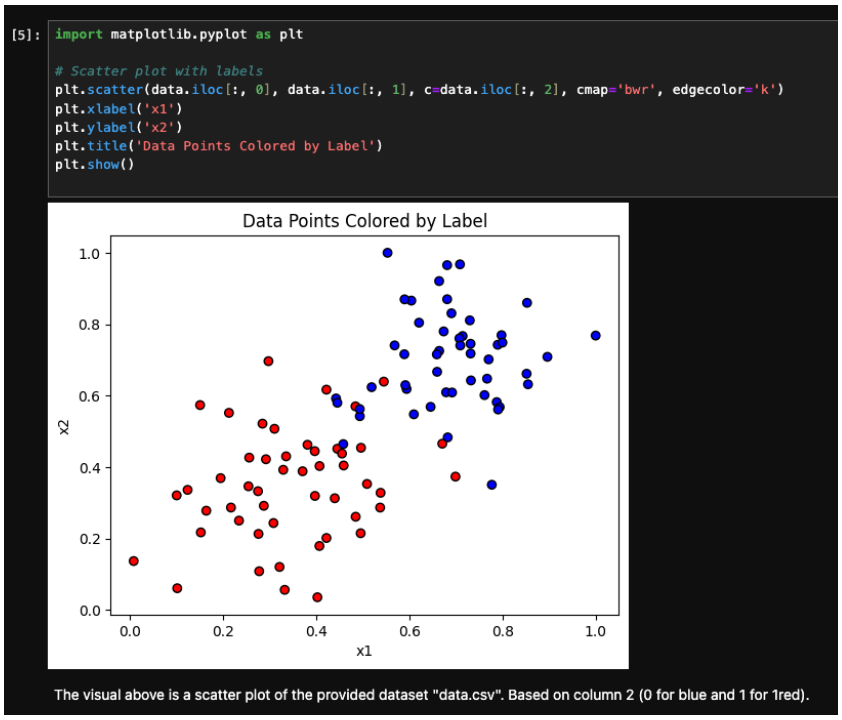
Author: Shambhavi Danayak
Professor: Bo Shen
Student ID: 012654513
Submission date: 04/20/2025

PART 1:

- **Plot data from data.csv**



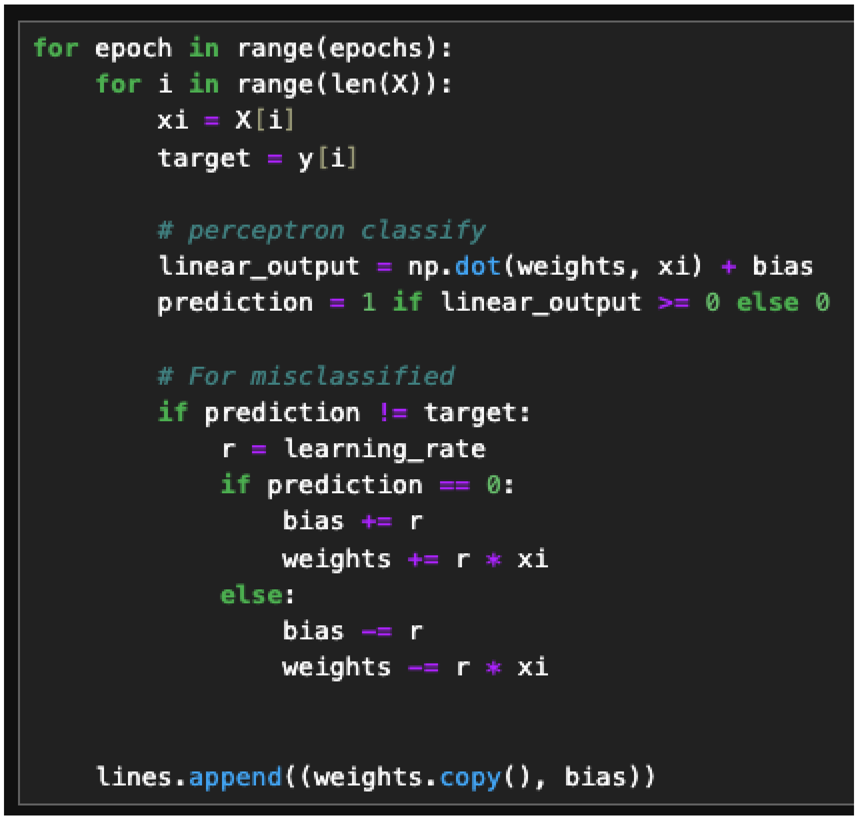
```
1]: import pandas as pd
    data = pd.read_csv('data.csv')

2]: print(data.head)

    <bound method NDFrame.head of      0.78051  -0.063669  1
    0   0.28774    0.29139  1
    1   0.40714    0.17878  1
    2   0.29230    0.42170  1
    3   0.50922    0.35256  1
    4   0.27785    0.10802  1
    ..      ...        ... ..
    94  0.77029    0.70140  0
    95  0.73156    0.71782  0
    96  0.44556    0.57991  0
    97  0.85275    0.85987  0
    98  0.51912    0.62359  0

    [99 rows x 3 columns]>

3]: data.describe

3]: <bound method NDFrame.describe of      0.78051  -0.063669  1
    0   0.28774    0.29139  1
    1   0.40714    0.17878  1
    2   0.29230    0.42170  1
    3   0.50922    0.35256  1
    4   0.27785    0.10802  1
    ..      ...        ... ..
    94  0.77029    0.70140  0
    95  0.73156    0.71782  0
    96  0.44556    0.57991  0
    97  0.85275    0.85987  0
    98  0.51912    0.62359  0

    [99 rows x 3 columns]>
```



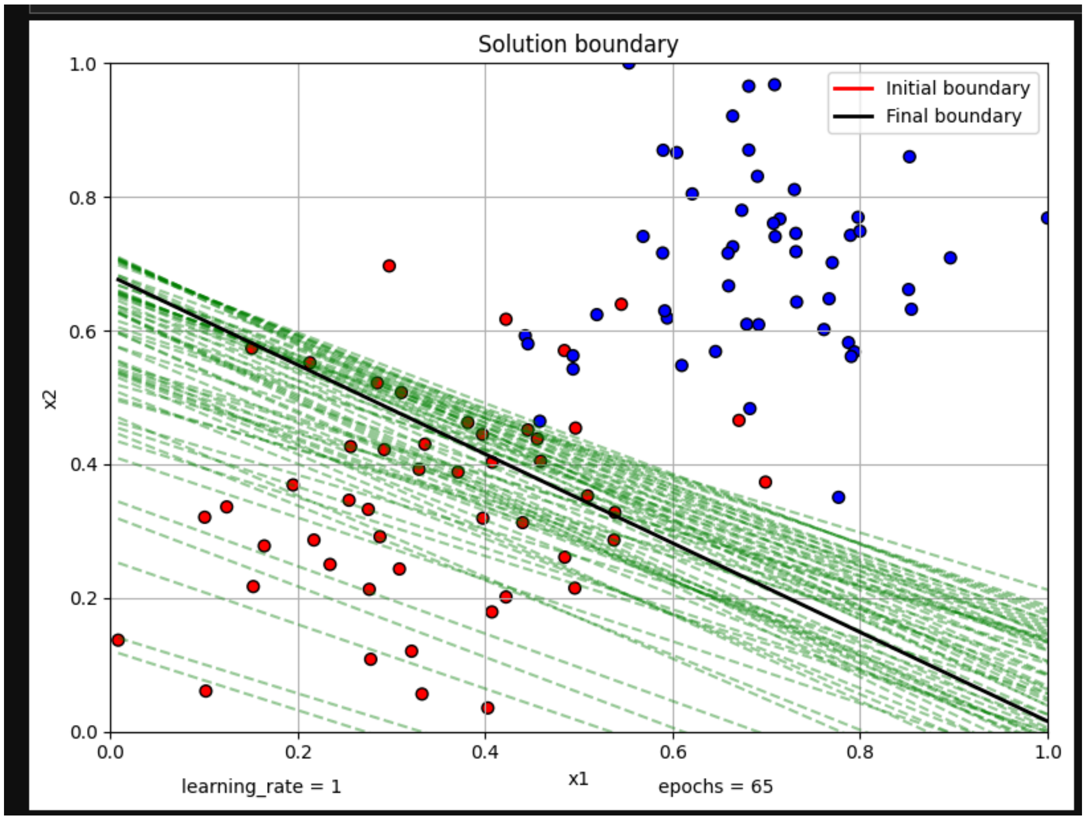
```
[5]: import matplotlib.pyplot as plt

     # Scatter plot with labels
     plt.scatter(data.iloc[:, 0], data.iloc[:, 1], c=data.iloc[:, 2], cmap='bwr', edgecolor='k')
     plt.xlabel('x1')
     plt.ylabel('x2')
     plt.title('Data Points Colored by Label')
     plt.show()
```

The visual above is a scatter plot of the provided dataset "data.csv". Based on column 2 (0 for blue and 1 for 1red).

- **Implement perceptron using the heuristic approach (left box next page)**
- **Plot the initial separation line as red, subsequent ones after each iteration in dashed green, and the last one in black**



```
for epoch in range(epochs):
    for i in range(len(X)):
        xi = X[i]
        target = y[i]

        # perceptron classify
        linear_output = np.dot(weights, xi) + bias
        prediction = 1 if linear_output >= 0 else 0

        # For misclassified
        if prediction != target:
            r = learning_rate
            if prediction == 0:
                bias += r
                weights += r * xi
            else:
                bias -= r
                weights -= r * xi


    lines.append((weights.copy(), bias))
```

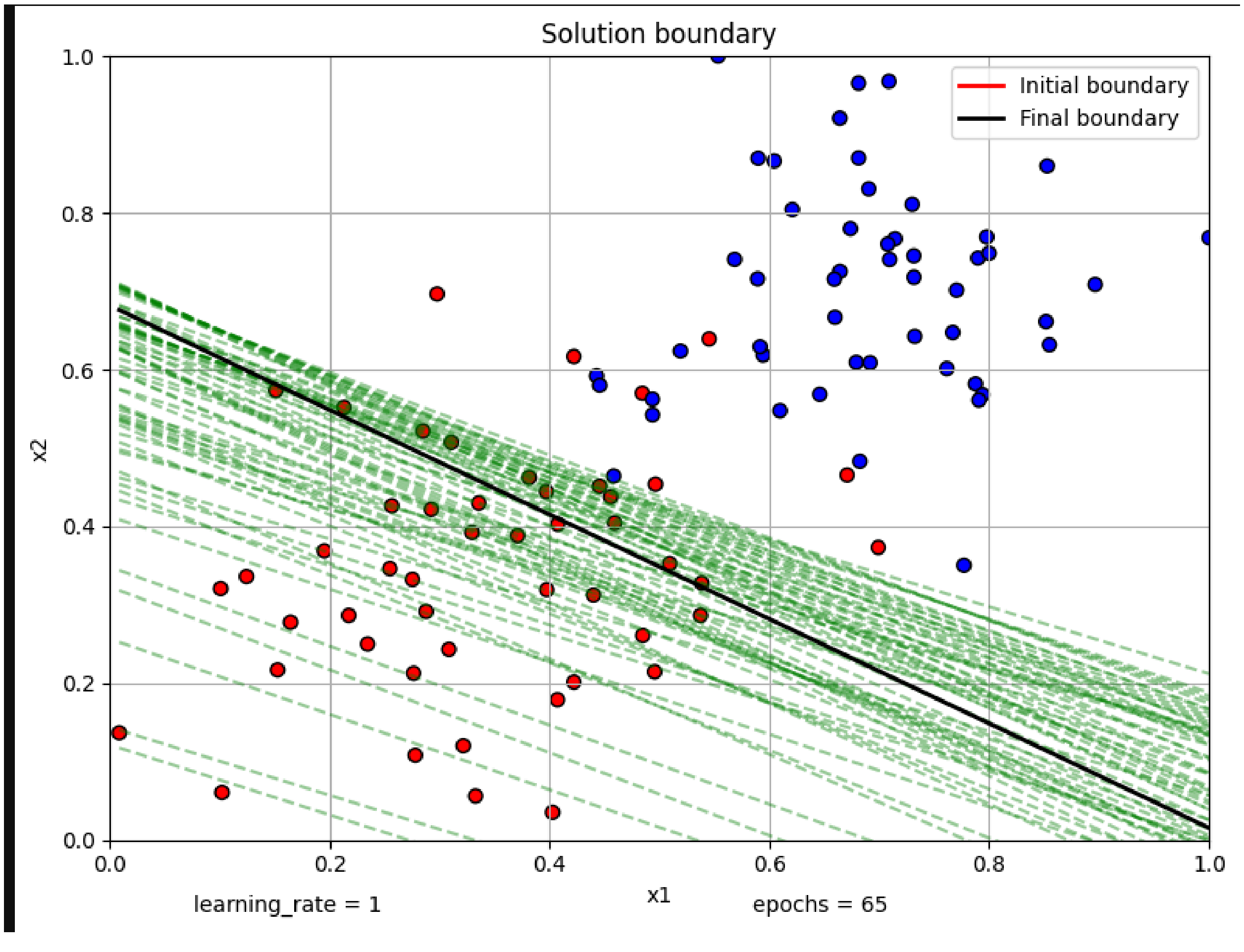


EXPLANATION: The above uses a Heuristic approach to implement a Perceptron. The algorithm starts with random weights and bias, and updates them iteratively based on classification error. If the prediction is incorrect, the weights and bias are adjusted — increased if the point should be classified as 1, and decreased if it should be 0. This update is repeated for a fixed number of epochs, allowing the decision boundary to shift and better separate the classes.

- Repeat #2 enough number of times

Below are the results of running the implemented perceptron using a heuristic approach with various learning rates and epochs.

Learning rate=1, epochs=65

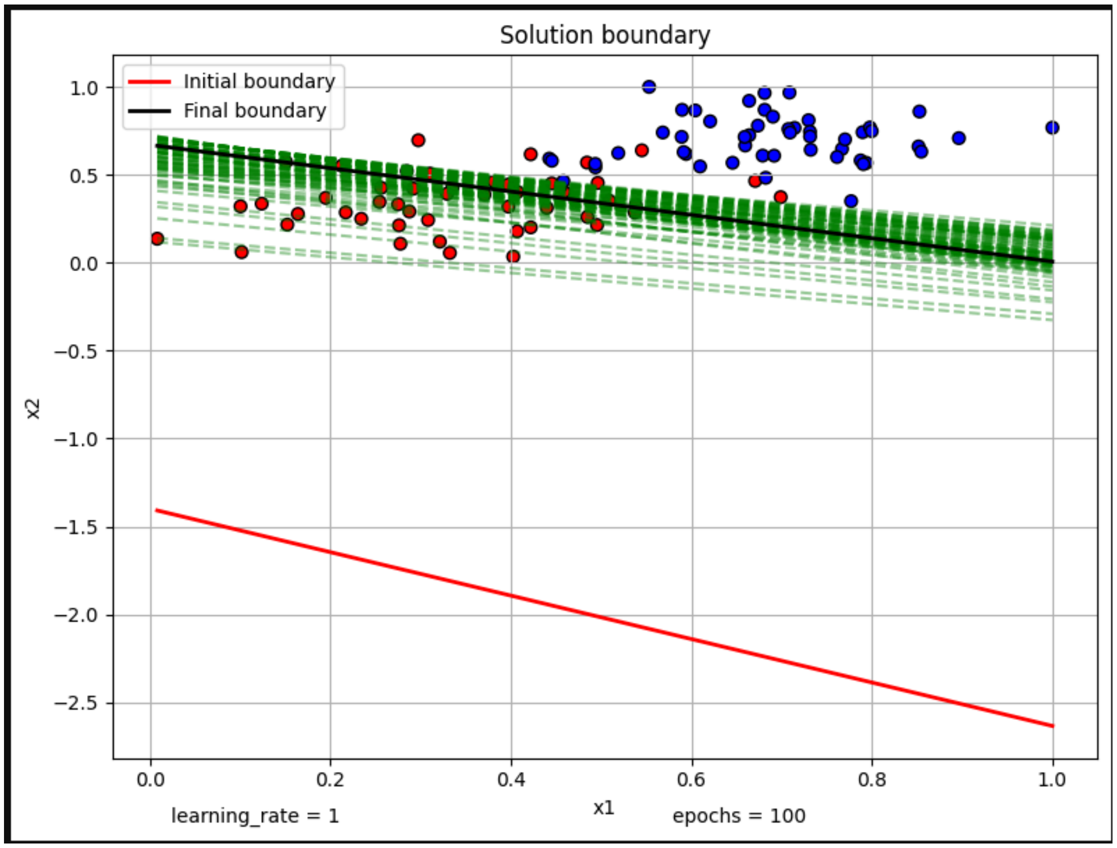


ANALYSIS:

- The dense green lines indicate many misclassified points across epochs
- The final decision boundary (black) successfully separates most of the red and blue points with many misclassified red points.

- Due to higher learning rate (1) causes the decision boundary to shift aggressively.

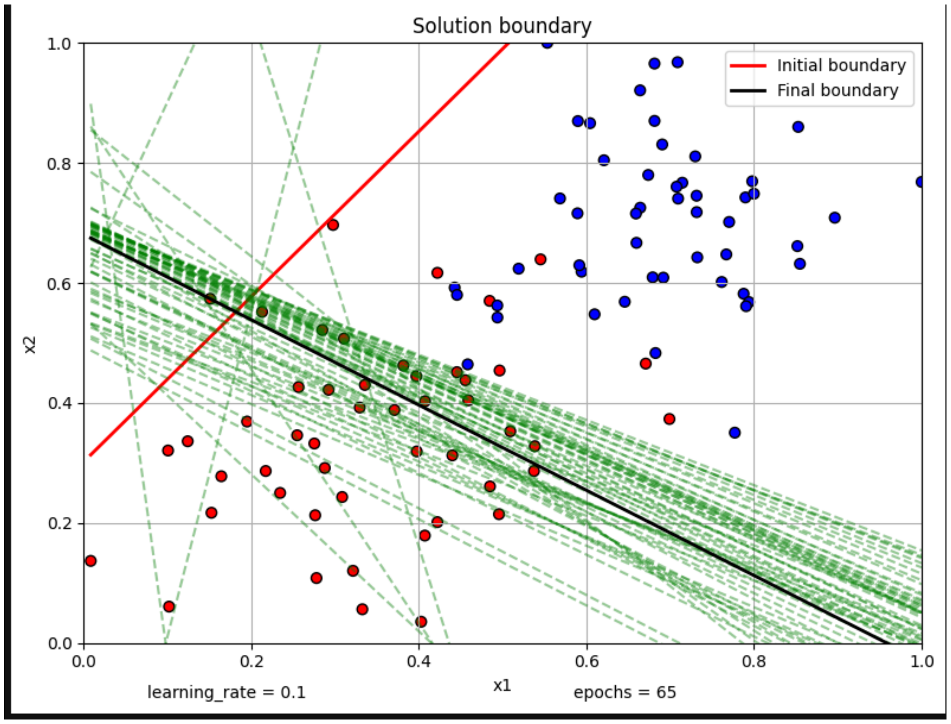
Learning rate=1, epochs=100



ANALYSIS:

- With a higher number of epochs the perceptron continues to adjust the decision boundary.
- Despite increasing the number of epochs from 65 to 100, the final decision boundary shows only marginal improvement, and similar misclassified points (impurities) remain.

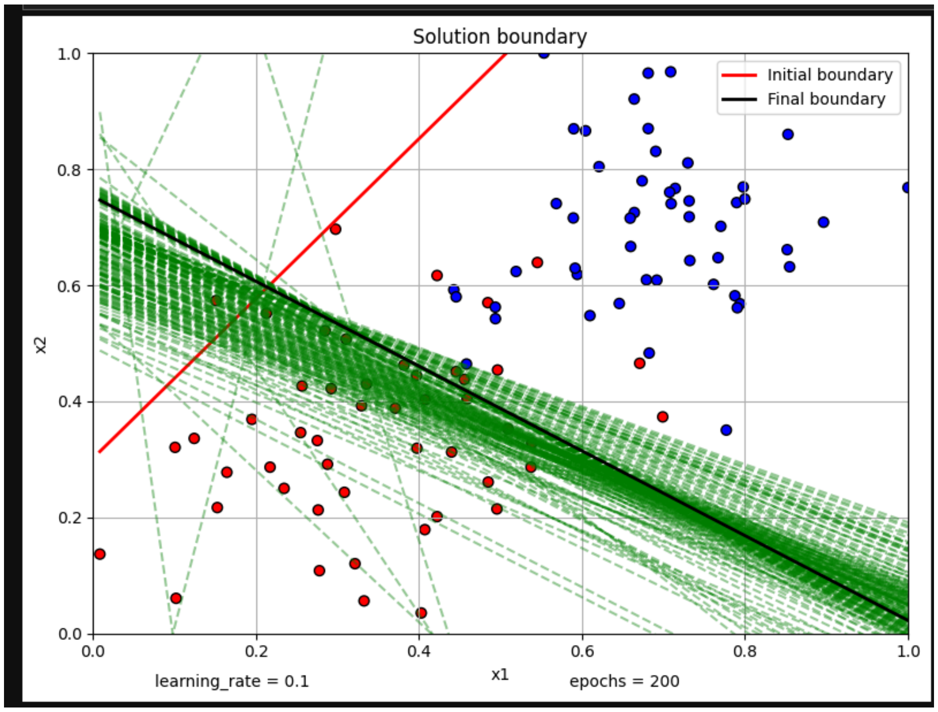
Learning rate=0.1, epochs=65



ANALYSIS:

- The spread out dense green lines indicates the perceptron makes smaller, more gradual updates.
- The decision boundary separates most data points, but the slower learning rate causes the model to take more steps
- As there are still impurities (misclassified red points) we can say that a low learning rate does not guarantee better accuracy.

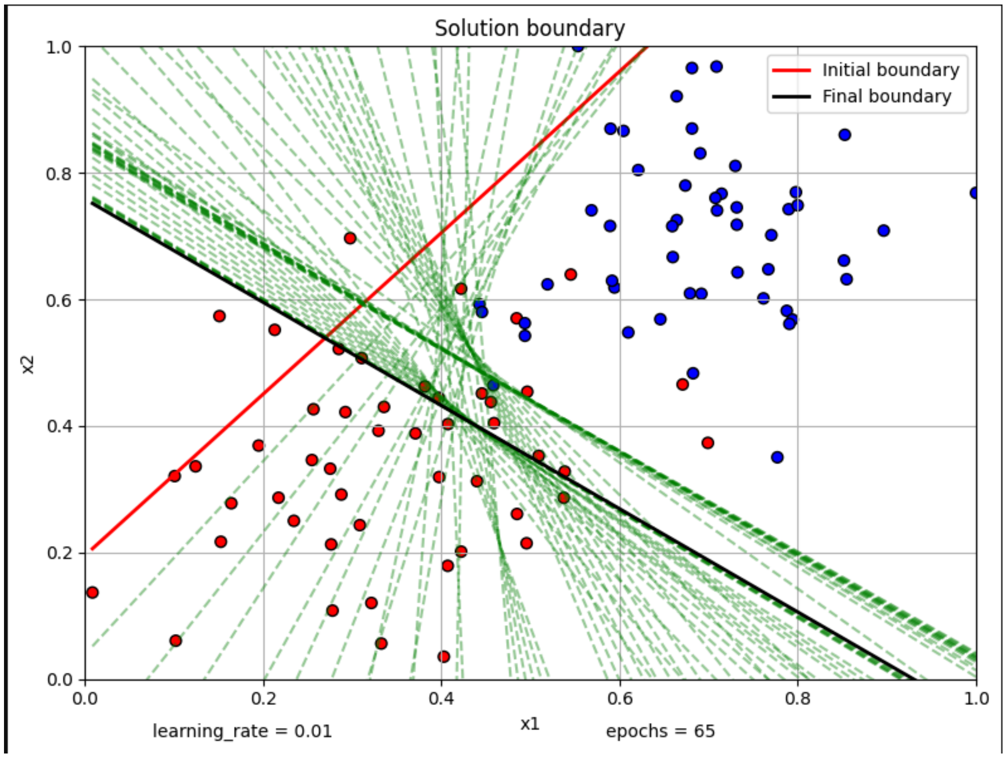
Learning rate=0.1, epochs=65



ANALYSIS:

- The dense green lines indicate that with lower learning rate and increased epochs the model performs a large number of small updates.
- The final decision boundary is well placed, showing improved class separation compared to earlier runs.

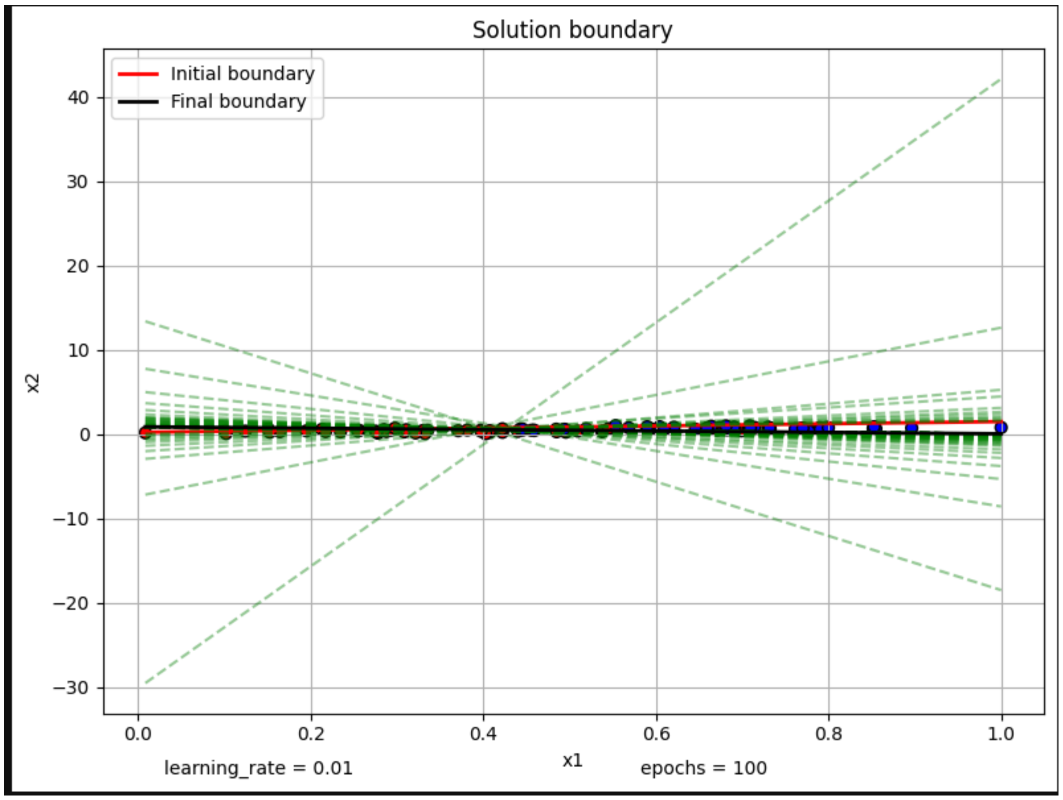
Learning rate=0.01, epochs=65



ANALYSIS:

- The spread out green lines indicate that very low learning rate makes the perceptron take small steps before reaching the decision boundary.
- Limited epochs prevent the model from stabilizing fully at this low learning rate, leading to visible residual misclassifications and unstable convergence behavior.

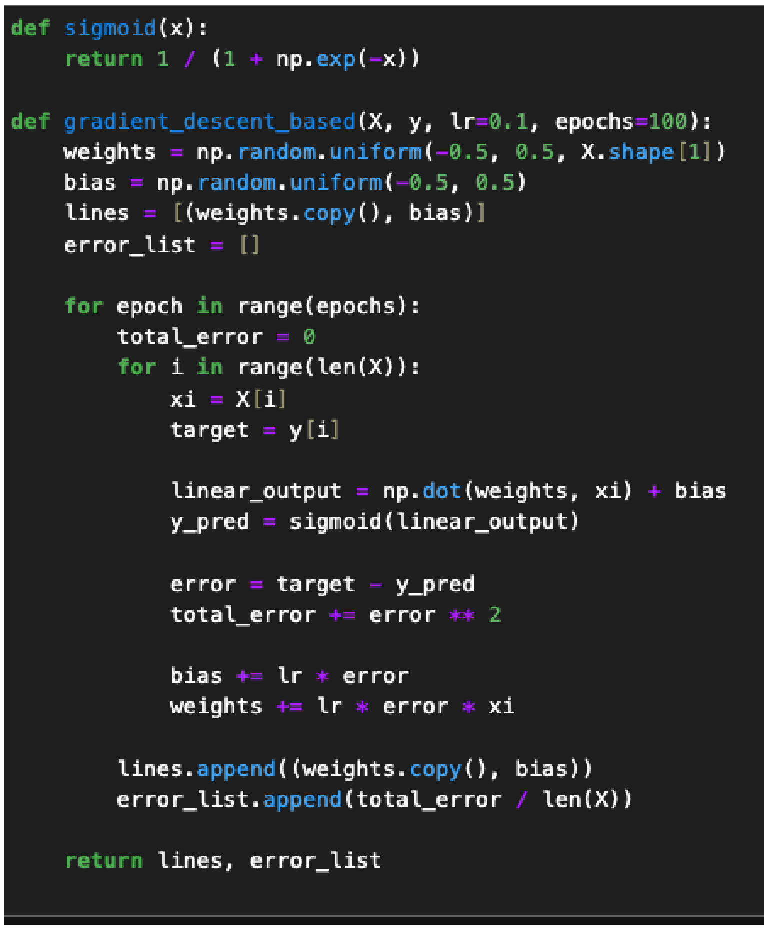
Learning rate=0.01, epochs=100



ANALYSIS:

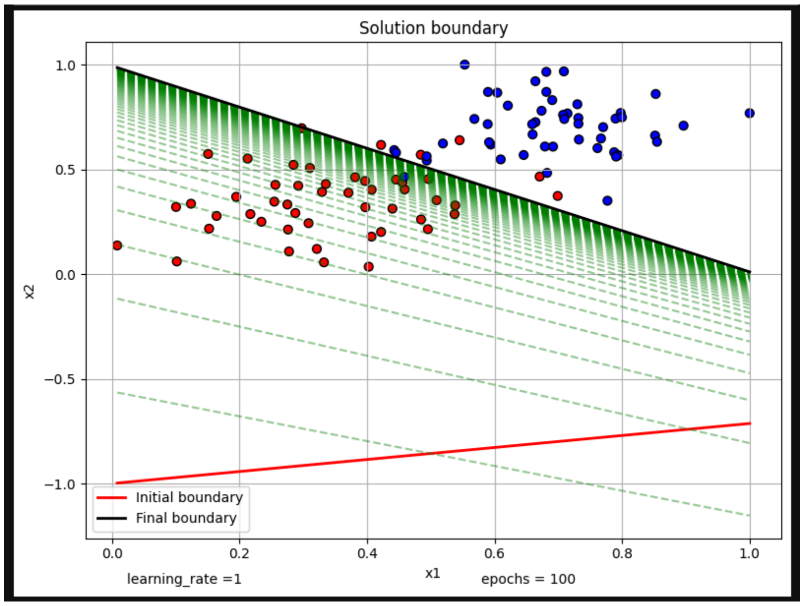
- The very low learning rate causes extremely small updates, resulting in a slow and shallow learning process, as reflected by the minimal shifts in decision boundaries.
- The plot suggests underfitting, highlighting the limitations of too small a learning rate without a proportional increase in training duration.

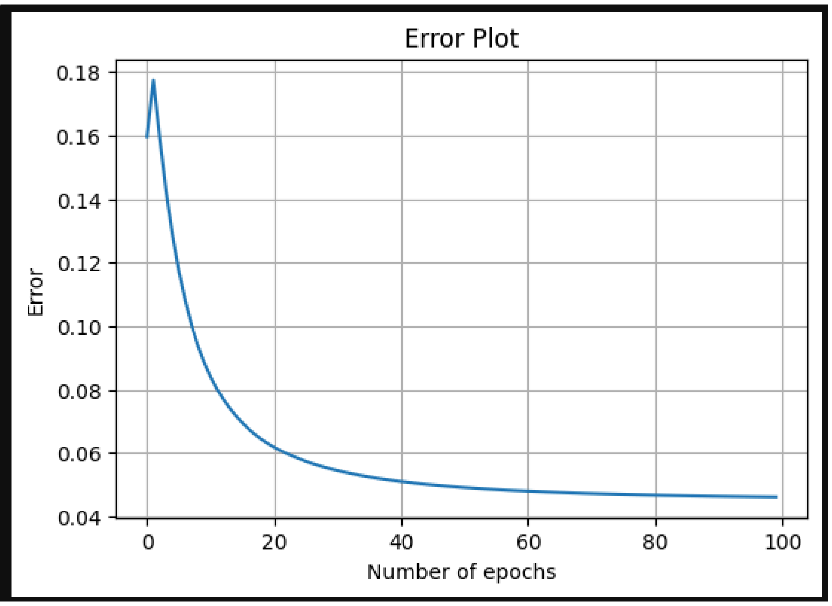
PART 2: Implement perceptron using the Gradient Descent approach



```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def gradient_descent_based(X, y, lr=0.1, epochs=100):
    weights = np.random.uniform(-0.5, 0.5, X.shape[1])
    bias = np.random.uniform(-0.5, 0.5)
    lines = [(weights.copy(), bias)]
    error_list = []

    for epoch in range(epochs):
        total_error = 0
        for i in range(len(X)):
            xi = X[i]
            target = y[i]

            linear_output = np.dot(weights, xi) + bias
            y_pred = sigmoid(linear_output)

            error = target - y_pred
            total_error += error ** 2

            bias += lr * error
            weights += lr * error * xi

        lines.append((weights.copy(), bias))
        error_list.append(total_error / len(X))

    return lines, error_list
```

Learning rate= 1 and epochs=100

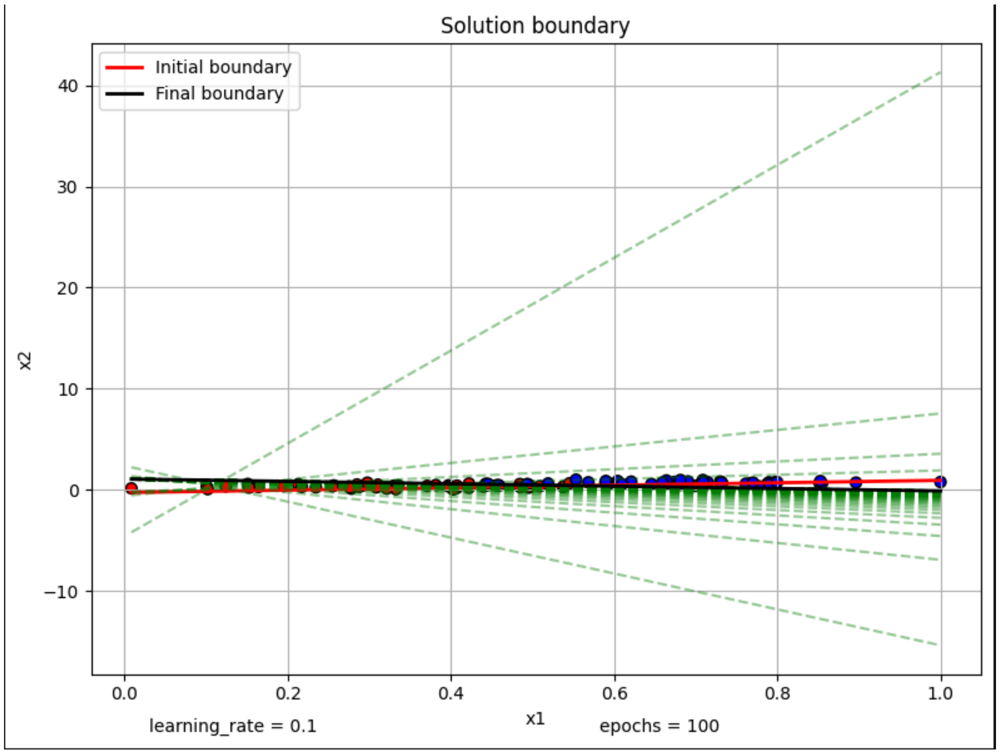


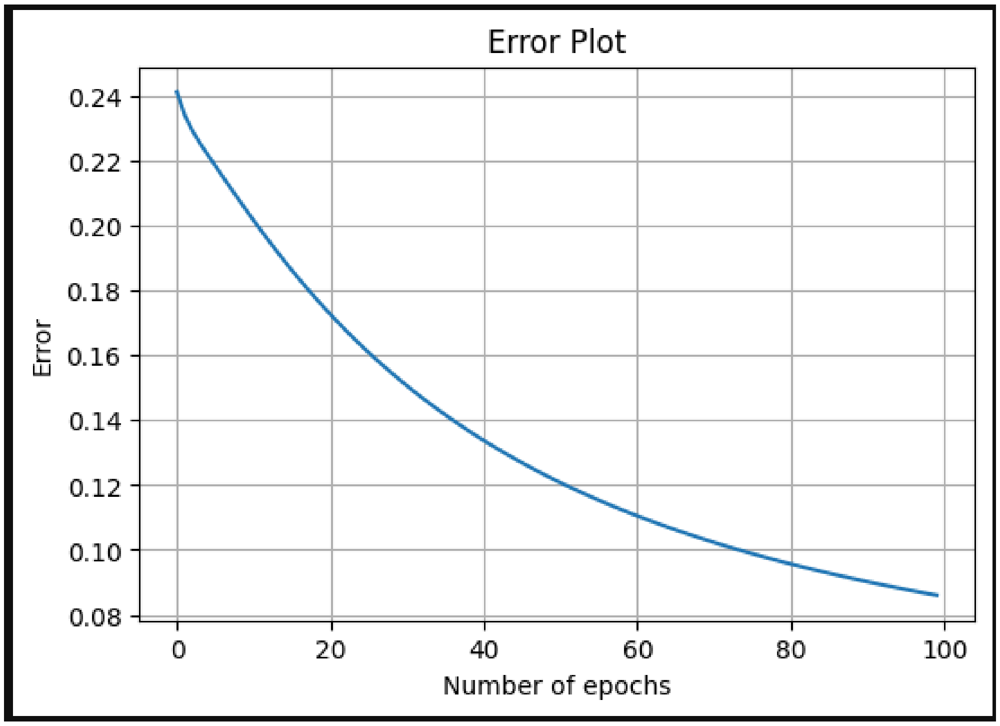


ANALYSIS:

- The error curve shows a steep drop during the first 20–30 epochs, indicating that the high learning rate enabled the model to quickly correct large initial errors.
- sigmoid-based updates produce a smooth, continuously decreasing error and a clean sequence of green boundary adjustments, reflecting stable gradient-driven learning without drastic jumps.
- The decision boundary is well placed with most of the points correctly classified. Very few misclassifications in comparison with the heuristic approach.

Learning rate= 0.1 and epochs=100

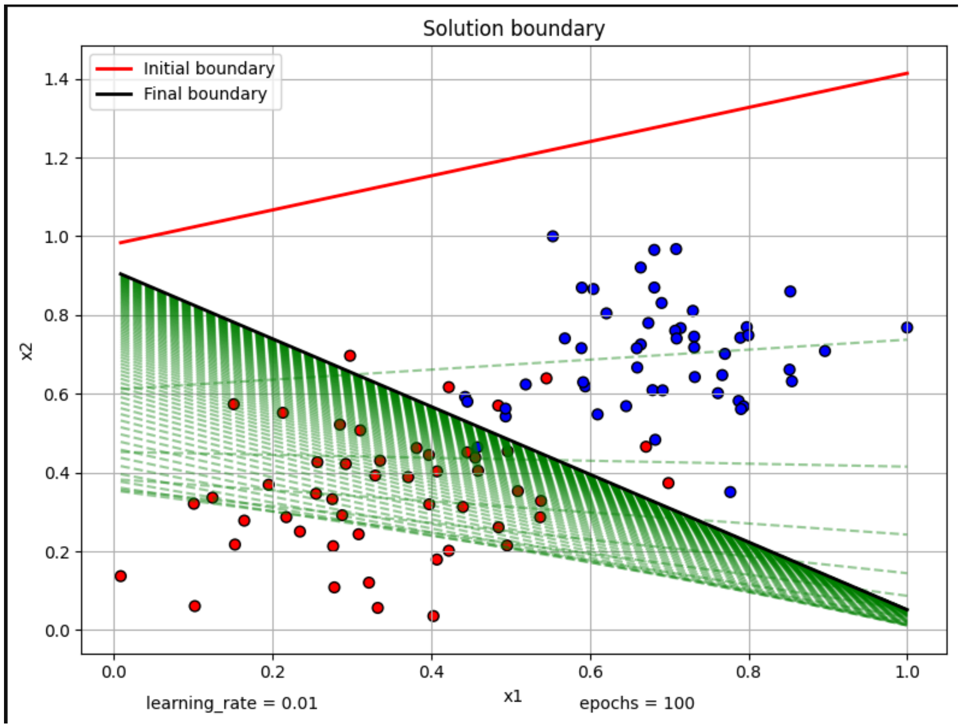


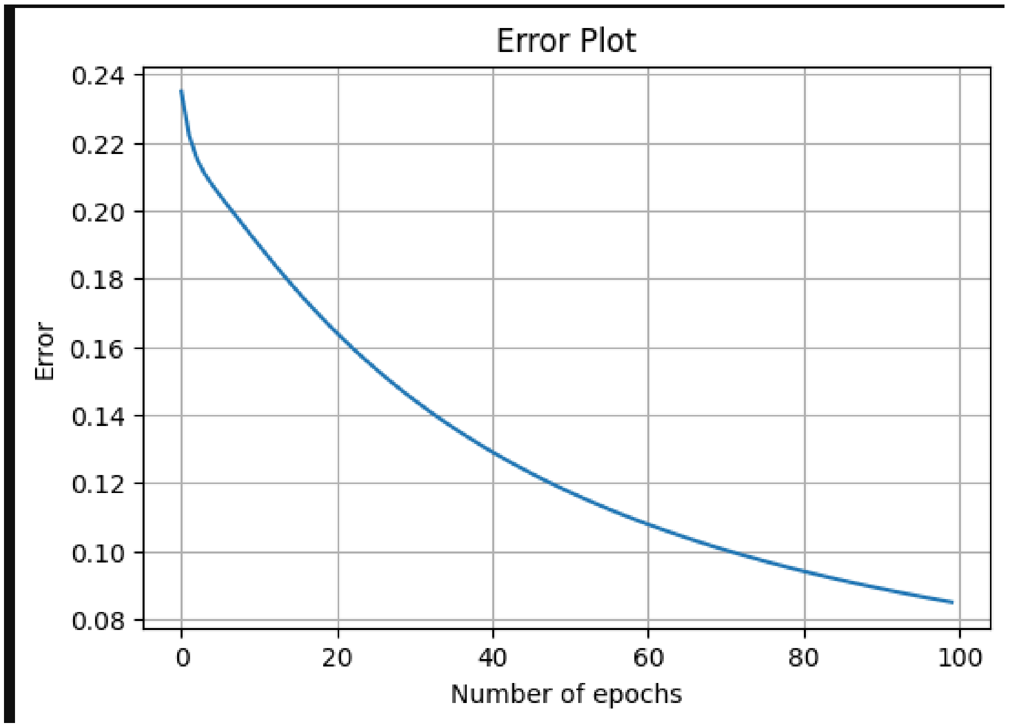


ANALYSIS:

- By looking at the produced solution boundary graph we can see that the model did not learn strong decision patterns. The final boundary remains close to the initial boundary and points are misclassified heavily leading to poor classification.
- The error plot tells us that the model is improving, but at a limited pace, and has not yet fully optimized its decision boundary

Learning rate= 0.01 and epochs=100





ANALYSIS:

- The error plot shows a slow but steady decline in error across 100 epochs, indicating that the model is improving, but the low learning rate is significantly slowing convergence.
- Dense green lines indicate that low learning rate converges towards decision boundary but results in an inefficient path towards optimization.
- The decision boundary is very placed with very few misclassified red points.

Thank you!