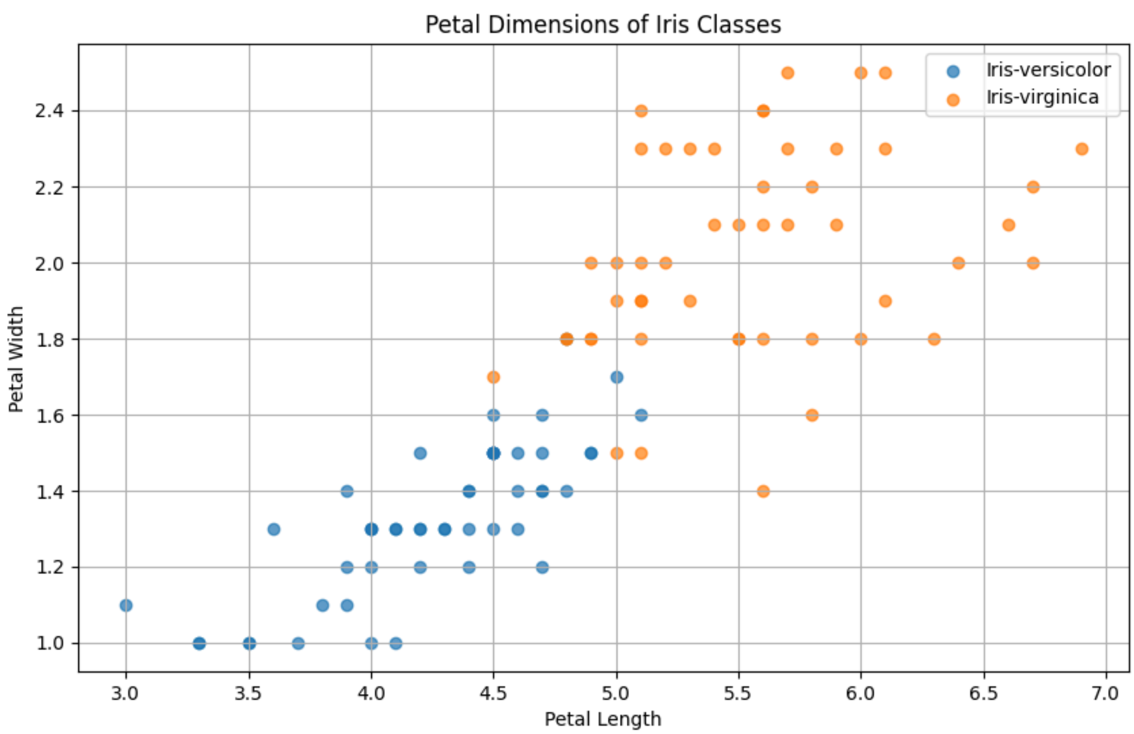
Preliminaries

I just want to mention that I used the following imports in the python code.

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

Exercise 1

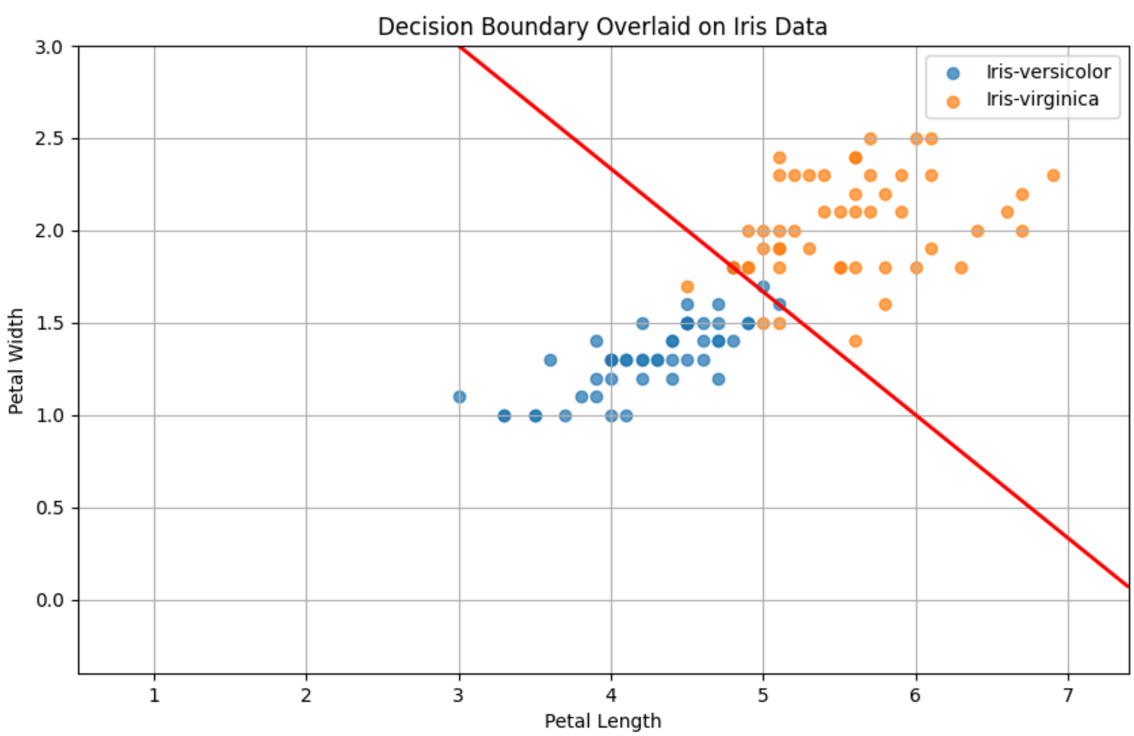
a)




```python
def plotClasses(iris_data):
    # class_1 = iris_data[iris_data['species'] == 'setosa']
    class_2 = iris_data[iris_data['species'] == 'versicolor']
    class_3 = iris_data[iris_data['species'] == 'virginica']

    plt.figure(figsize=(10, 6))

    plt.scatter(class_2['petal_length'], class_2['petal_width'],
                label="Iris-versicolor", alpha=0.7)

```

```python
    # plt.scatter(class_1['petal_length'], class_1['petal_width'],
    #             label="Iris-versicolor", alpha=0.7)

    plt.scatter(class_3['petal_length'], class_3['petal_width'],
                label="Iris-virginica", alpha=0.7)

    plt.xlabel('Petal Length')
    plt.ylabel('Petal Width')
    plt.title('Petal Dimensions of Iris Classes')
    plt.legend()
    plt.grid(True)

    plt.show()
```

B & c)



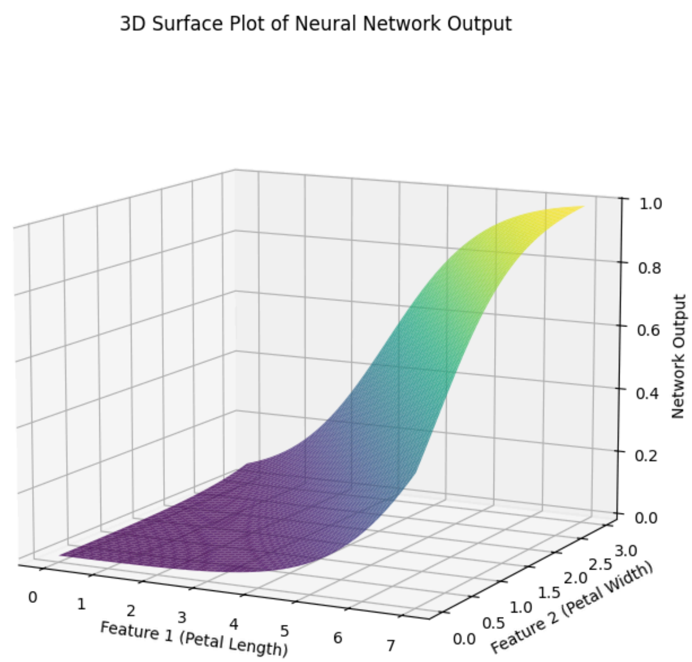

```python
def sigmoid(x):
    return 1/(1+np.exp(-x))


def oneLayerNN(data, weights, bias):
    return sigmoid(np.dot(data, weights) + bias)


def decisionBoundaryPlot(data, weights, bias):
```

```
    class_2 = data[data['species'] == 'versicolor']
    class_3 = data[data['species'] == 'virginica']

    plt.figure(figsize=(10, 6))

    plt.scatter(class_2['petal_length'], class_2['petal_width'],
                label="Iris-versicolor", alpha=0.7)

    # plt.scatter(class_1['petal_length'], class_1['petal_width'],
    #             label="Iris-versicolor", alpha=0.7)

    plt.scatter(class_3['petal_length'], class_3['petal_width'],
                label="Iris-virginica", alpha=0.7)

    x_min, x_max = data['petal_length'].min() - 0.5, data['petal_length'].max() + 0.5

    y_min, y_max = data['petal_width'].min() - 0.5, data['petal_width'].max() + 0.5

    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200),
                         np.linspace(y_min, y_max, 200))

    grid_points = np.c_[xx.ravel(), yy.ravel()]
    outputs = oneLayerNN(grid_points, weights, bias)

    outputs = outputs.reshape(xx.shape)
    plt.contour(xx, yy, outputs, levels=[0.5], colors='red', linewidths=2)

    # Add labels and title
    plt.xlabel('Petal Length')
    plt.ylabel('Petal Width')
    plt.title('Decision Boundary Overlaid on Iris Data')
    plt.legend()
    plt.grid(True)
    plt.show()
```

The particular weights and bias for the image instance were:

Weights: `[1.0,1.5]` and Bias: `-7.5`

d)




```python
def plot_surface(weights, bias, x_range, y_range):
        # Generate a grid of points over the feature space
    xx, yy = np.meshgrid(np.linspace(x_range[0], x_range[1], 100),
                         np.linspace(y_range[0], y_range[1], 100))

    # Flatten the grid and compute model outputs
    grid_points = np.c_[xx.ravel(), yy.ravel()]
    outputs = oneLayerNN(grid_points, weights, bias)

    # Reshape the output to match the grid shape
    zz = outputs.reshape(xx.shape)

    # Create a 3D surface plot
    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(xx, yy, zz, cmap='viridis', alpha=0.8)

    # Add labels and title
    ax.set_xlabel('Feature 1 (Petal Length)')
    ax.set_ylabel('Feature 2 (Petal Width)')
    ax.set_zlabel('Network Output')
    ax.set_title('3D Surface Plot of Neural Network Output')

```

```
    # Show the plot
    plt.show()
```

This was generated with the same weights and bias as above.

e)

```
def sampleOutput(data, weights, bias):
    data = data[(data['species'] == 'versicolor') | (data['species'] == 'virginica')]

    features = data[['petal_length', 'petal_width']].values
    outputs = oneLayerNN(features, weights, bias)

    data['output'] = outputs

    data['class'] = data['output'] > 0.5

    data['class'] = data['class'].map({True: 'virginica', False: 'versicolor'})

    unambiguous_examples = data[(outputs < 0.2) | (outputs > 0.8)]
    near_boundary_examples = data[(outputs >= 0.4) & (outputs <= 0.6)]

    # Display results
    print("Unambiguous Examples:")
    print(unambiguous_examples
          [['petal_length', 'petal_width', 'output', 'class', 'species']])

    print("\nNear-Boundary Examples:")
    print(near_boundary_examples
          [['petal_length', 'petal_width', 'output', 'class', 'species']])
```

Output

```
Unambiguous Examples:
     petal_length  petal_width    output       class     species
53            4.0          1.3  0.175086  versicolor  versicolor
57            3.3          1.0  0.062973  versicolor  versicolor
59            3.9          1.4  0.182426  versicolor  versicolor
60            3.5          1.0  0.075858  versicolor  versicolor
62            4.0          1.0  0.119203  versicolor  versicolor
64            3.6          1.3  0.124553  versicolor  versicolor
67            4.1          1.0  0.130108  versicolor  versicolor
69            3.9          1.1  0.124553  versicolor  versicolor
```

```
71              4.0          1.3  0.175086  versicolor  versicolor
79              3.5          1.0  0.075858  versicolor  versicolor
80              3.8          1.1  0.114052  versicolor  versicolor
81              3.7          1.0  0.091123  versicolor  versicolor
82              3.9          1.2  0.141851  versicolor  versicolor
88              4.1          1.3  0.190002  versicolor  versicolor
89              4.0          1.3  0.175086  versicolor  versicolor
92              4.0          1.2  0.154465  versicolor  versicolor
93              3.3          1.0  0.062973  versicolor  versicolor
95              4.2          1.2  0.182426  versicolor  versicolor
98              3.0          1.1  0.054681  versicolor  versicolor
99              4.1          1.3  0.190002  versicolor  versicolor
100             6.0          2.5  0.904651   virginica   virginica
102             5.9          2.1  0.824914   virginica   virginica
104             5.8          2.2  0.832018   virginica   virginica
105             6.6          2.1  0.904651   virginica   virginica
107             6.3          1.8  0.817574   virginica   virginica
109             6.1          2.5  0.912934   virginica   virginica
117             6.7          2.2  0.924142   virginica   virginica
118             6.9          2.3  0.945319   virginica   virginica
120             5.7          2.3  0.838891   virginica   virginica
122             6.7          2.0  0.900250   virginica   virginica
130             6.1          1.9  0.809998   virginica   virginica
131             6.4          2.0  0.869892   virginica   virginica
132             5.6          2.2  0.802184   virginica   virginica
135             6.1          2.3  0.885948   virginica   virginica
136             5.6          2.4  0.845535   virginica   virginica
140             5.6          2.4  0.845535   virginica   virginica
143             5.9          2.3  0.864127   virginica   virginica
144             5.7          2.5  0.875447   virginica   virginica

Near-Boundary Examples:
     petal_length  petal_width    output       class     species
52              4.9          1.5  0.413382  versicolor  versicolor
56              4.7          1.6  0.401312  versicolor  versicolor
70              4.8          1.8  0.500000  versicolor  versicolor
72              4.9          1.5  0.413382  versicolor  versicolor
77              5.0          1.7  0.512497   virginica  versicolor
83              5.1          1.6  0.500000  versicolor  versicolor
119             5.0          1.5  0.437823  versicolor   virginica
121             4.9          2.0  0.598688   virginica   virginica
123             4.9          1.8  0.524979   virginica   virginica
126             4.8          1.8  0.500000  versicolor   virginica
127             4.9          1.8  0.524979   virginica   virginica
133             5.1          1.5  0.462570  versicolor   virginica
134             5.6          1.4  0.549834   virginica   virginica
```

```
138            4.8          1.8  0.500000  versicolor   virginica
146            5.0          1.9  0.586618   virginica   virginica
149            5.1          1.8  0.574443   virginica   virginica

```

Exercise 2

a)

```python
def mse(data, weights, bias, labels):
    data = data[(data['species'] == 'versicolor') | (data['species'] == 'virginica')]

    features = data[['petal_length', 'petal_width']]
    outputs = oneLayerNN(features, weights, bias)

    mse = np.mean((labels-outputs)**2)
    print(mse)
    return mse
```

b)

```python
mse(iris_data, weights, bias, data['species'].map({'versicolor': 0, 'virginica': 1}))

mse(iris_data, np.array([-0.5, 2.5]), -5, data['species'].map({'versicolor': 0, 'virginica': 1}))
```

Output

```
Weights: [1.  1.5], Bias: -7.5, MSE: 0.08529593279618802
Weights: [-0.5  2.5], Bias: -5, MSE: 0.4308316119255908
```

c)

We define the error function as:

$$E = \frac{1}{2} \sum_{n=0}^{N} \left(\sigma \left(\vec{w} \cdot \vec{x}_n \right) - c_n \right)^2$$

Then we take the derivative as:

$$\frac{\partial E}{\partial \vec{w}} = \frac{1}{2} \frac{\partial}{\partial \vec{w}} \sum_{n=0}^{N} \left(\sigma\left(\vec{w} \cdot \vec{x}_n\right) - c_n\right)^2$$

By the linearity of the partial derivative operator over finite sums we can take the partial derivative inside the sum. For brevity let $e_n = \sigma\left(\vec{w} \cdot \vec{x}_n\right) - c_n$. Hence:

$$\frac{\partial E}{\partial \vec{w}} = \frac{1}{2} \sum_{n=0}^{N} \frac{\partial}{\partial \vec{w}} (e_n)^2 = \sum_{n=0}^{N} e_n \frac{\partial e_n}{\partial \vec{w}}$$

Examining the inside term we can take the relevant partial derivatives with respect to $\vec{w}$.

$$\frac{\partial e_n}{\partial \vec{w}} = \frac{\partial}{\partial \vec{w}} \left(\sigma\left(\vec{w} \cdot \vec{x}_n\right) - c_n\right) = \sigma\left(\vec{w} \cdot \vec{x}_n\right)\left(1 - \sigma\left(\vec{w} \cdot \vec{x}_n\right)\right)\vec{x}_n$$

Returning to our sum we find:

$$\frac{\partial E}{\partial \vec{w}} = \sum_{n=0}^{N} \left(\sigma\left(\vec{w} \cdot \vec{x}_n\right) - c_n\right)\left(\sigma\left(\vec{w} \cdot \vec{x}_n\right)\left(1 - \sigma\left(\vec{w} \cdot \vec{x}_n\right)\right)\vec{x}_n\right)$$

Thus concluding the derivation.

d)

```
gradient(data, np.array([1.16663732, 6.19051848, -15.84598601]), data['species'].map({'versicolor': 0, 'virginica': 1}))

```

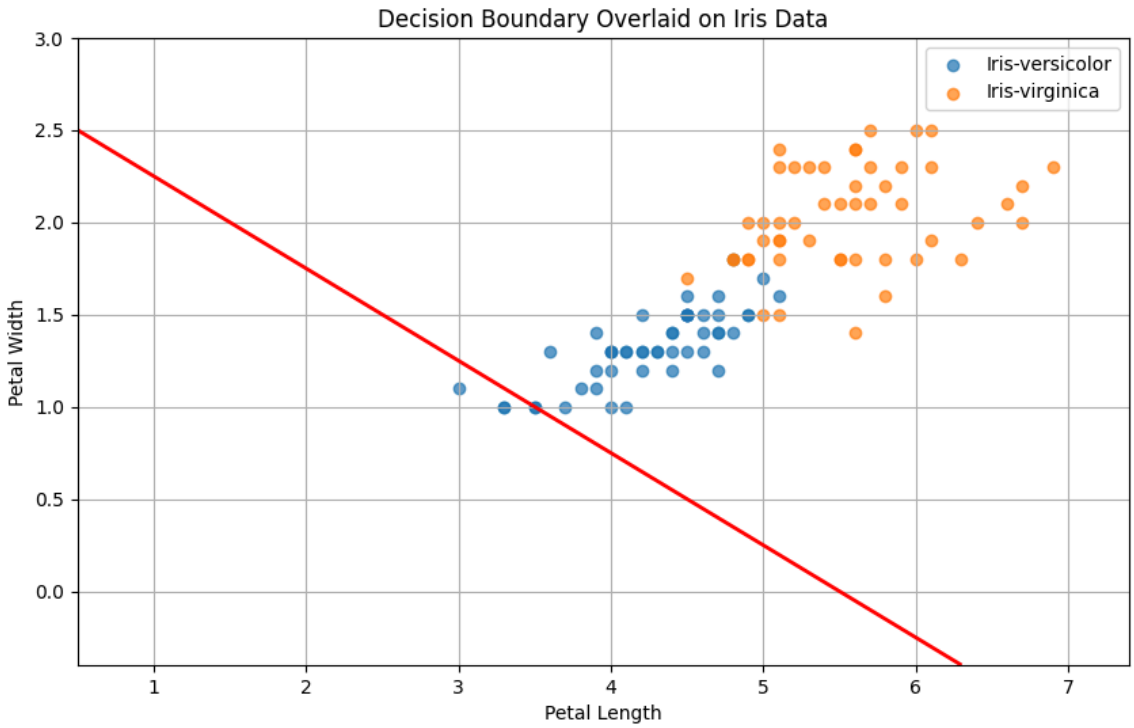
I defined a gradient function and called it as such. The weights are passed in as

$$\langle w_{\text{petal width}}, w_{\text{petal length}}, \text{bias} \rangle$$

This bias term is later included in the dot product by a dummy column in the input data which is a series of ones.

Hence the following weights yielded a decision boundary:

$$w_1 = \langle 2, 4, -11 \rangle$$



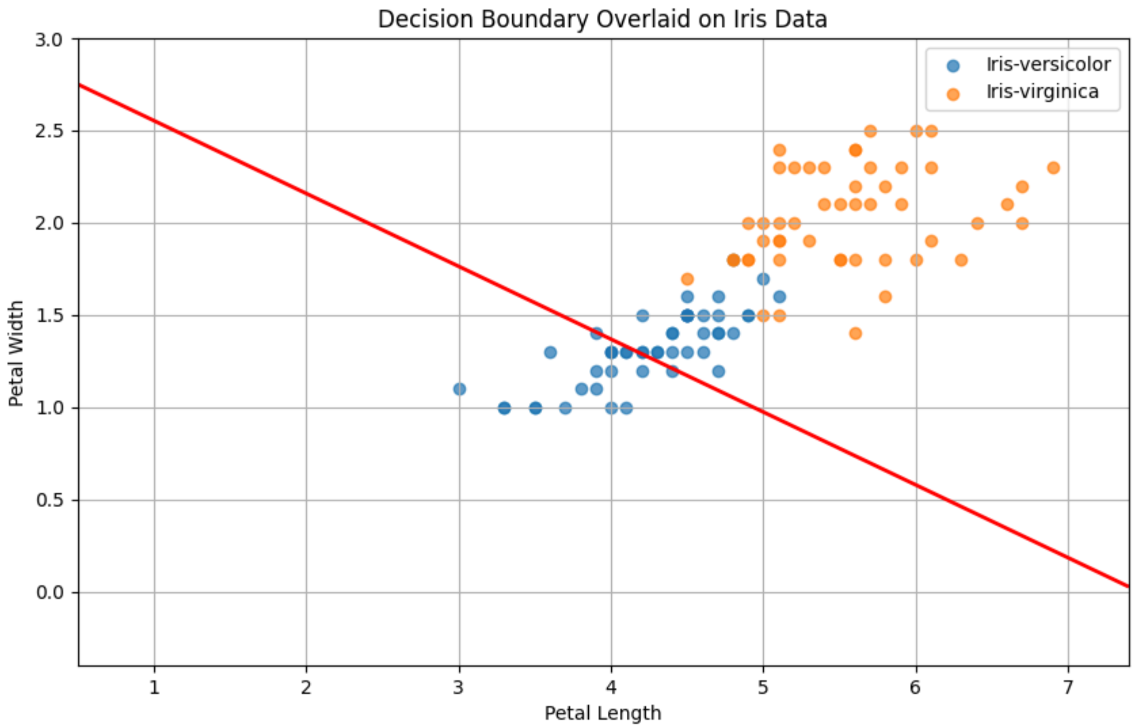
With a gradient: `[189.76129556 59.28079357 43.73572109]`

Which indicates we need to decrease all weights.

Following this 'recommendation' I changed the weights to:

$$w_2 = \langle 1.5, 3.8, -11.2 \rangle$$

Which yields the following decision boundary:



Which is clearly better, i.e. misclassifies less points.

Code Snippet

```
def gradient(data, weights, labels):
    # Filter and preprocess data
    data = data[(data['species'] == 'versicolor') | (data['species'] ==
'virginica')]
    data = data[['petal_length', 'petal_width']]
    data['bias'] = 1  # Add a bias term
    labels = labels[(data.index)]

    # Convert to numpy arrays
    X = data.to_numpy()
    y = labels.to_numpy()

    # Compute z = X @ weights
    z = np.dot(X, weights)

    # Apply sigmoid function
    sigma_z = 1 / (1 + np.exp(-z))

    # Compute the gradient: (sigma_z - y) * X
    errors = sigma_z - y
    gradient = np.dot(errors, X)
```

```

    print(gradient)

    return gradient
```

Exercise 3

a)

Code Snippet

```
def optimize(data, weights, labels, epsilon=0.01, maxIters=999, minDelta=1e-6):
    mse_history = []  # Store MSE for each iteration
    last_loss = float('inf')
    convergedIters = -1
    for i in range(maxIters):
        grad = gradient(data, weights, labels)
        weights -= epsilon * grad

        mse_value = mse(data, weights, labels)
        mse_history.append(mse_value)
        """
        if i % 500 == 0:
            print(i)
            decisionBoundaryPlot(data, weights)
        """
        if abs(mse_value - last_loss) < minDelta:
            print(f"Converged after {i} iterations")
            convergedIters = i
            break

        last_loss = mse_value
    print("Final Weights:", weights)
    return weights, mse_history, convergedIters
```

b)

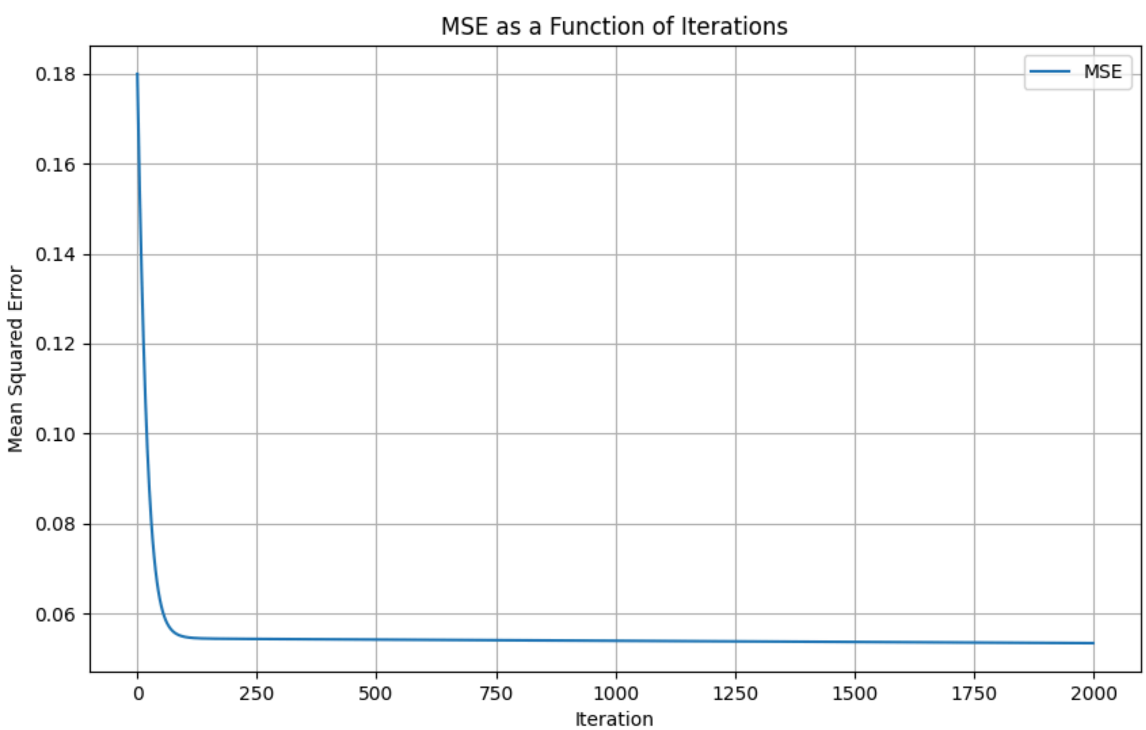
Code Snippet

```
final_weights, mse_history = optimize(data, np.array([1.5, 3.8, -11.2]),

labels, maxIters=2000, epsilon=0.0001)
decisionBoundaryPlot(data, final_weights)
```

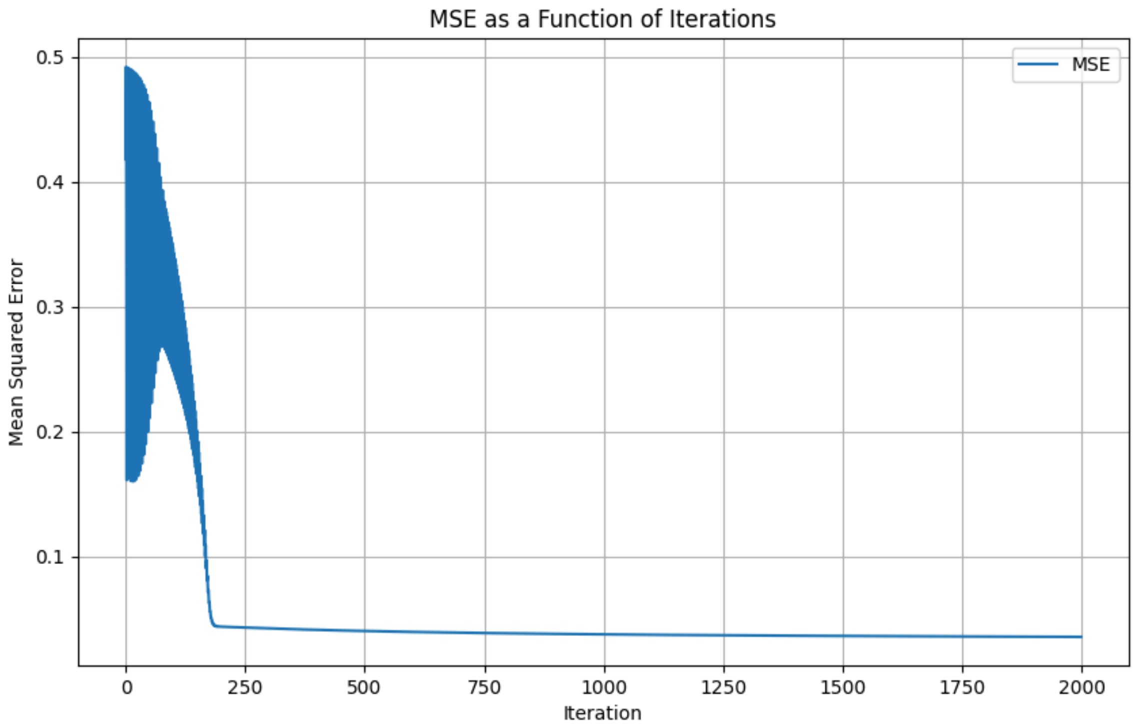
```

# Plot MSE over iterations
plt.figure(figsize=(10, 6))
plt.plot(range(len(mse_history)), mse_history, label="MSE")
plt.xlabel("Iteration")
plt.ylabel("Mean Squared Error")
plt.title("MSE as a Function of Iterations")
plt.grid(True)
plt.legend()
plt.show()
```

Objective Function Plot

This plot was with an $\epsilon = 0.0001$



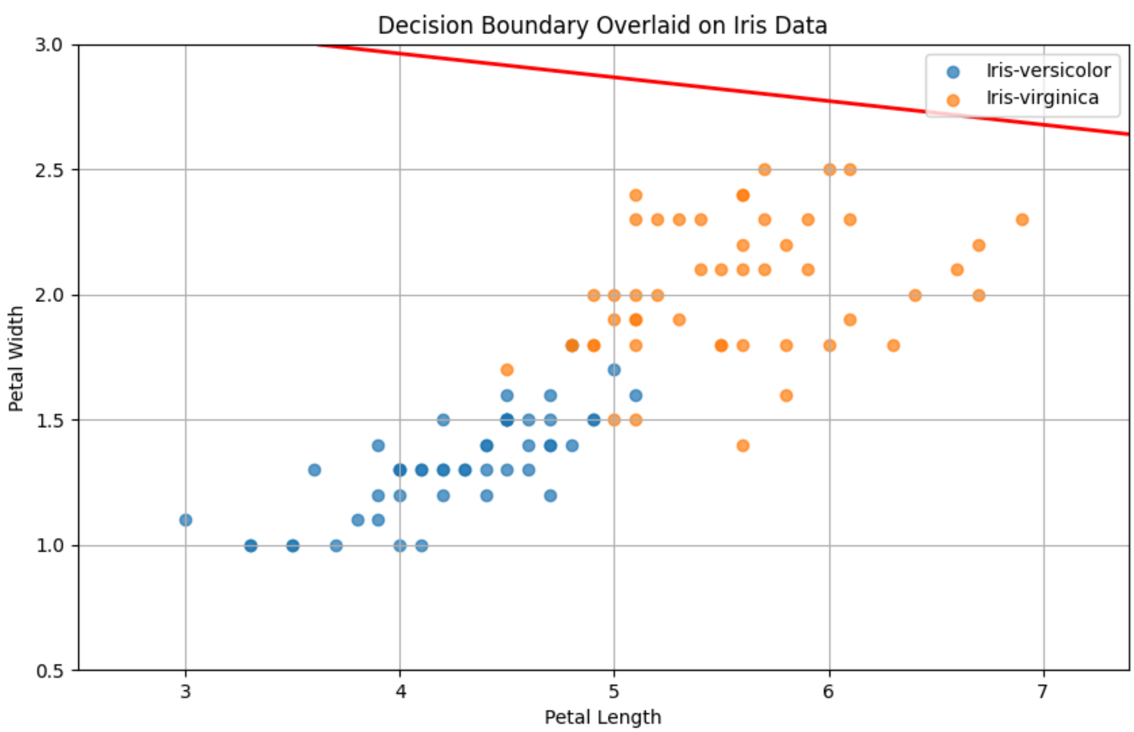
This plot was with an $\epsilon = 0.01$



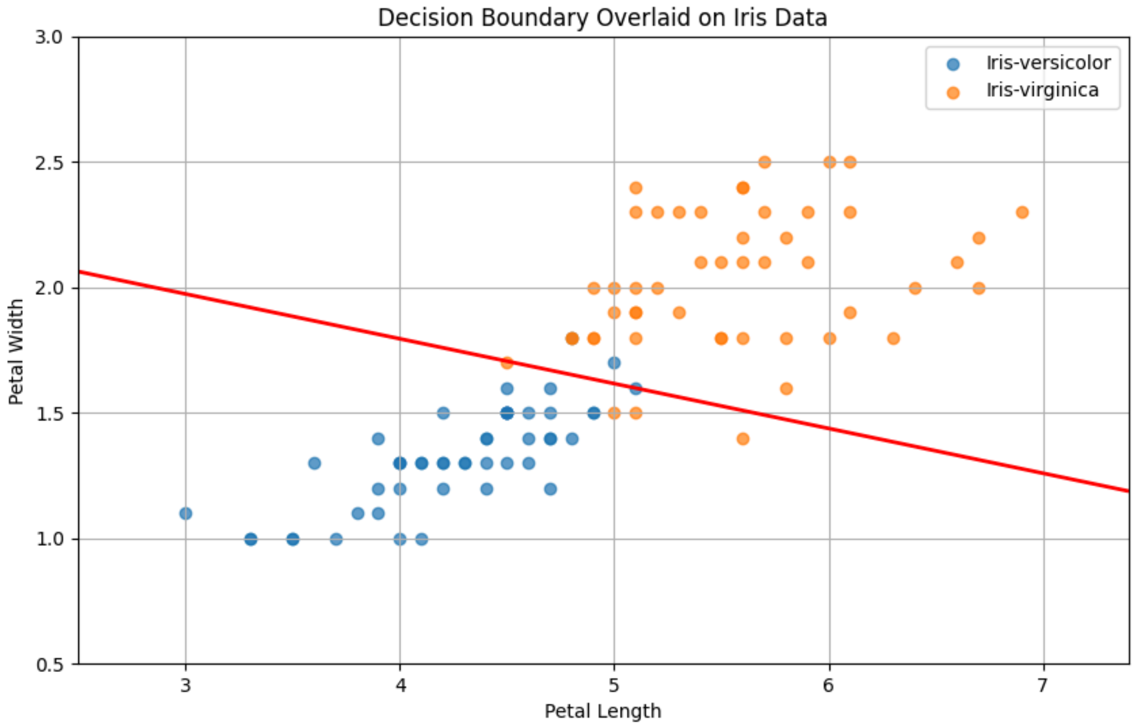
Decision Boundary Plot

With $\epsilon = 0.01$

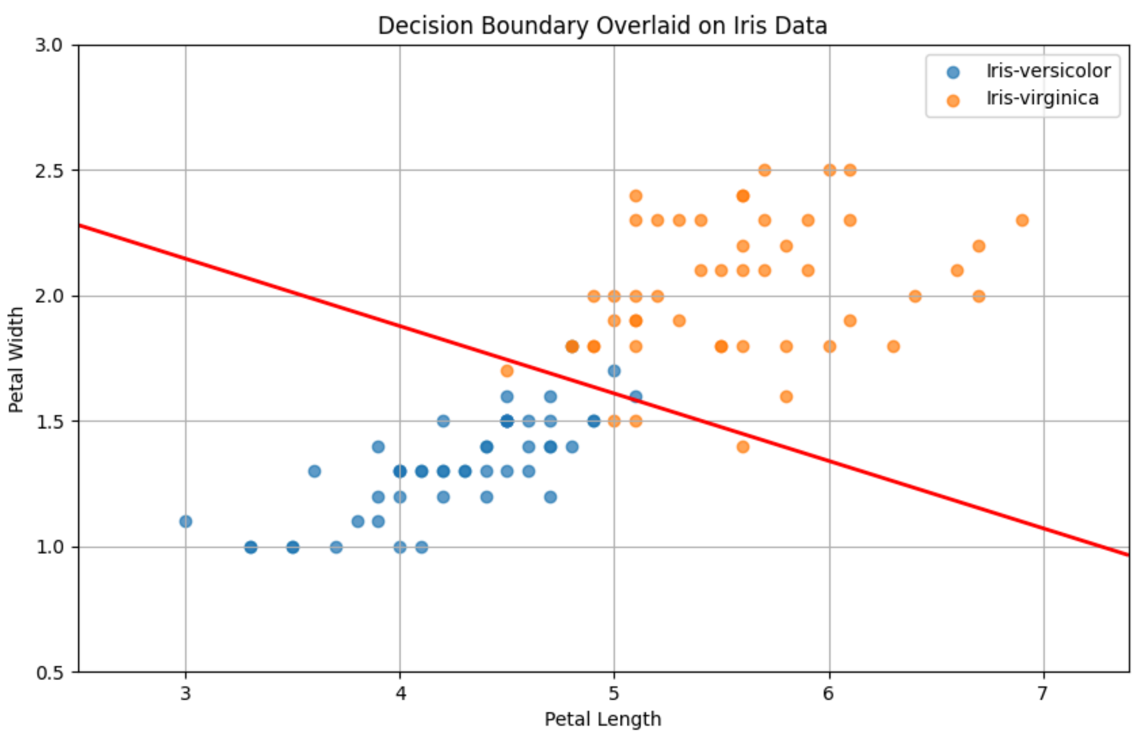
Initial



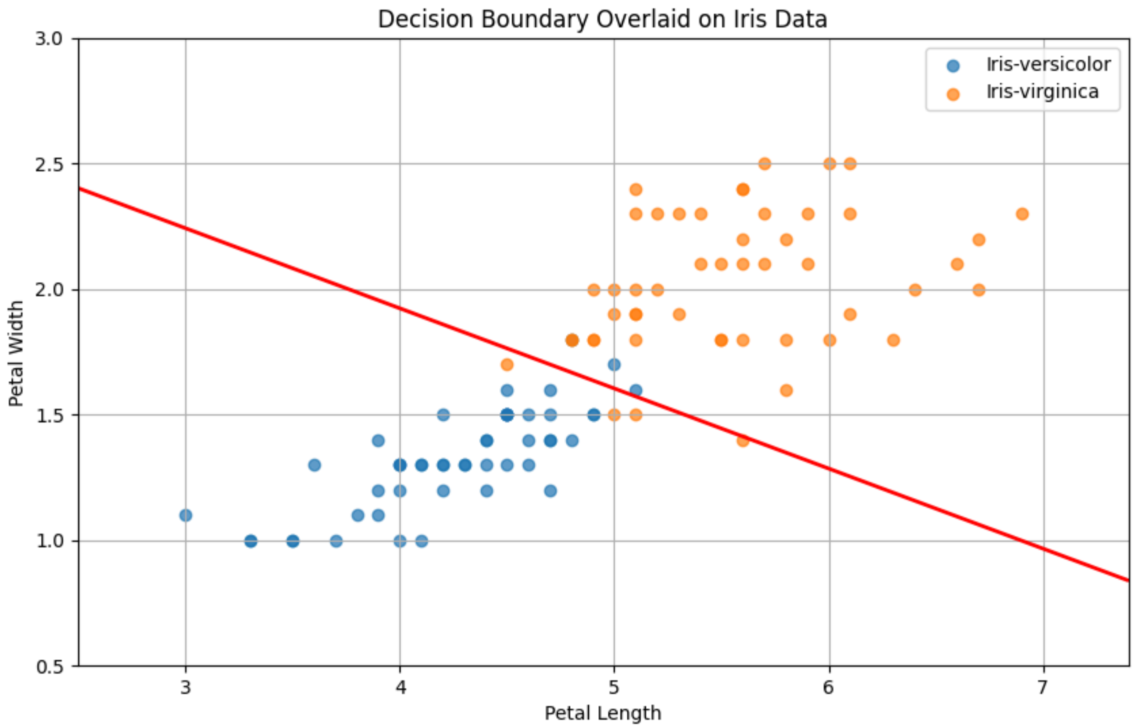
500



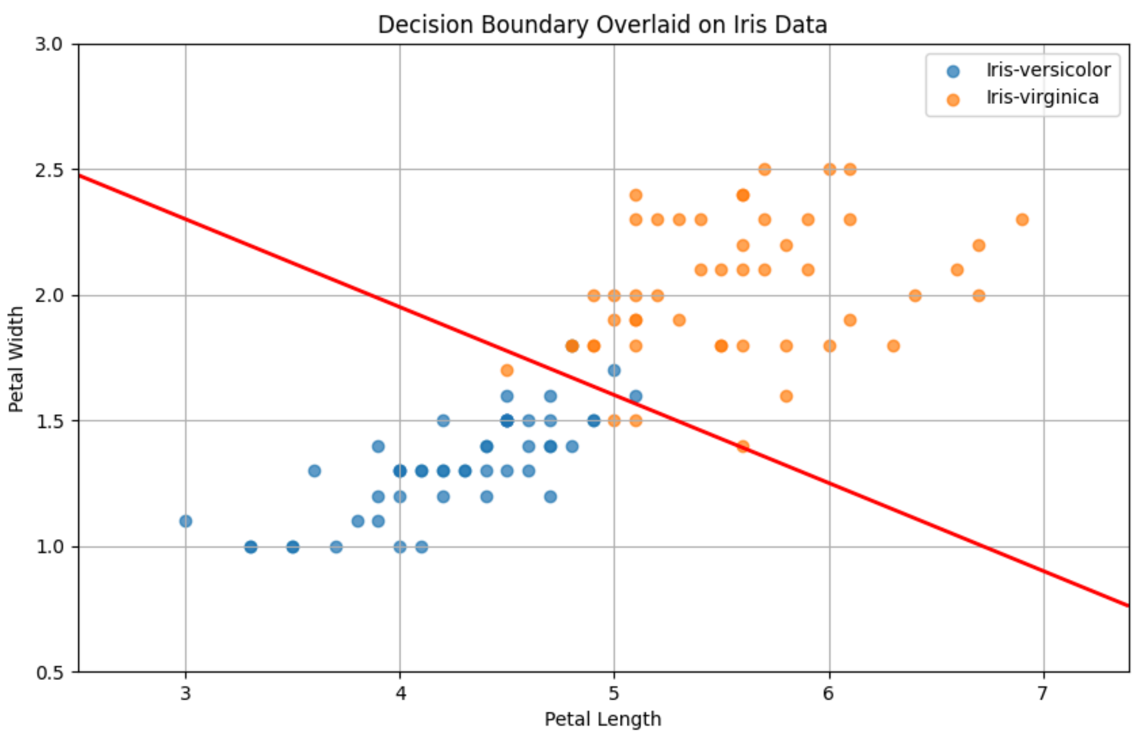
1000



1500

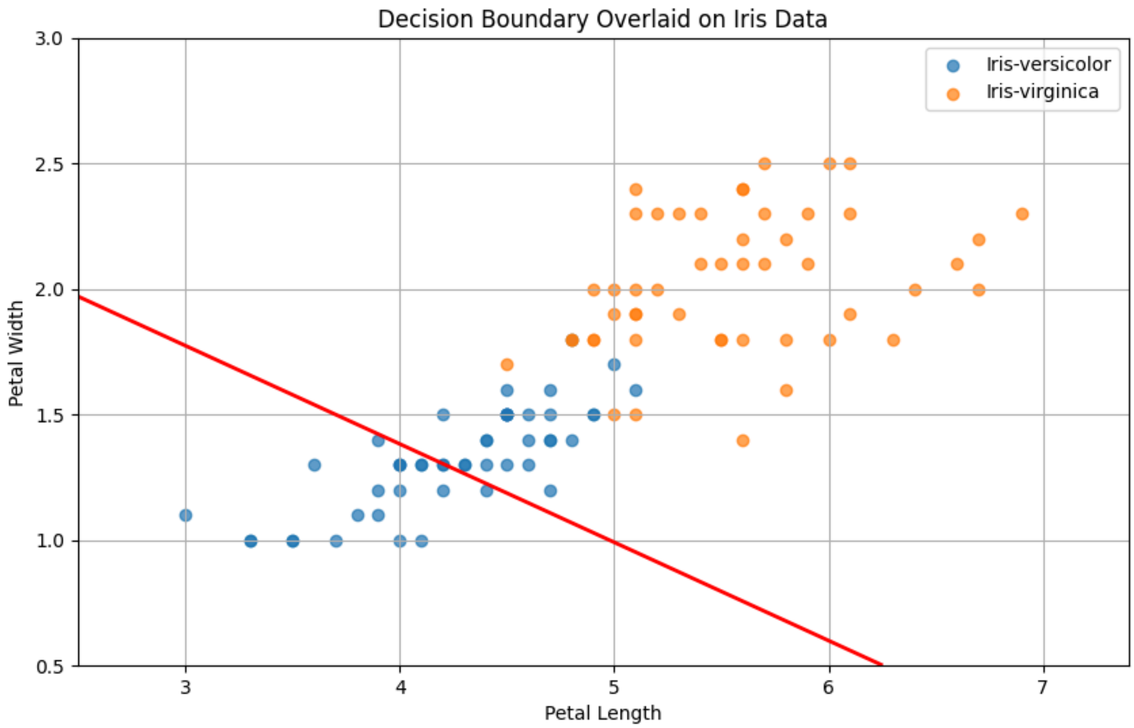


2000

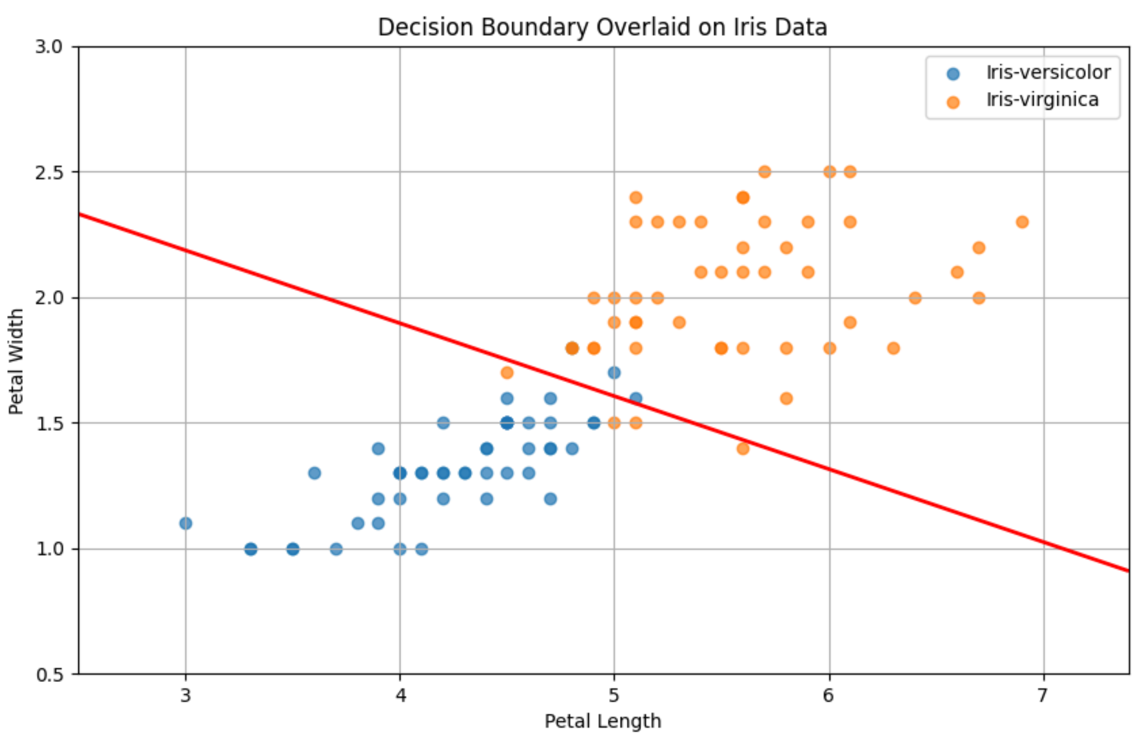


With $\epsilon = 0.0001$, plotted at 500 iteration intervals

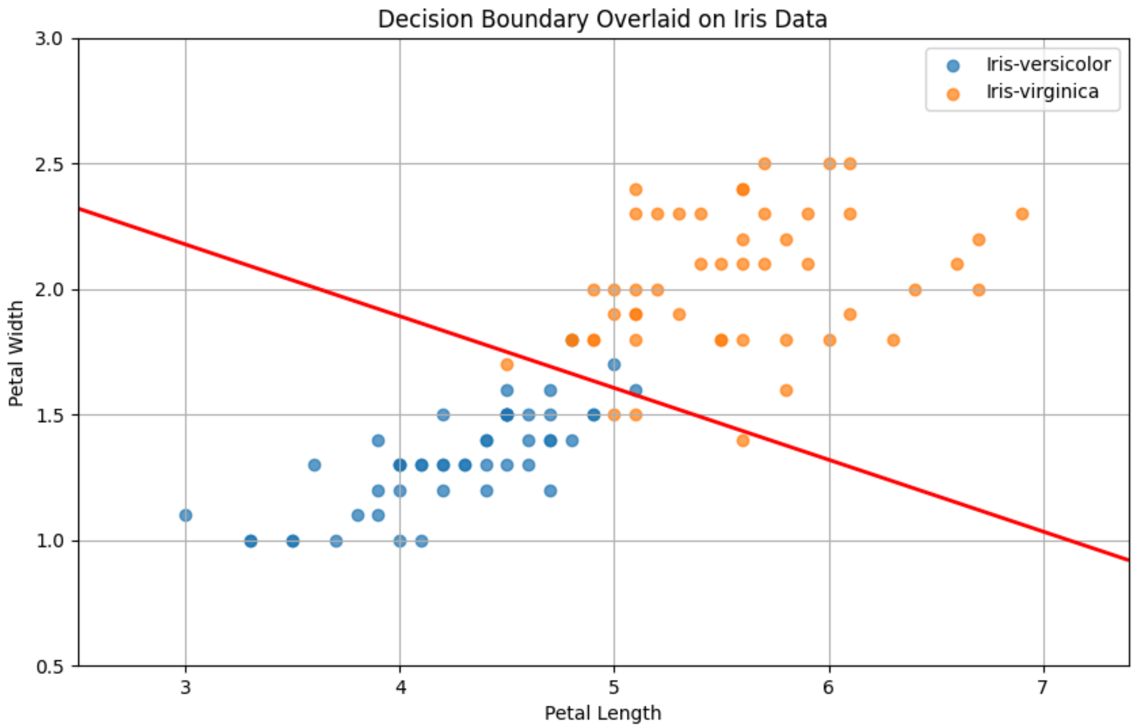
Initial


Decision Boundary Overlaid on Iris Data
Iris-versicolor
Iris-virginica
Petal Width
Petal Length
3.0
2.5
2.0
1.5
1.0
0.5
3
4
5
6
7

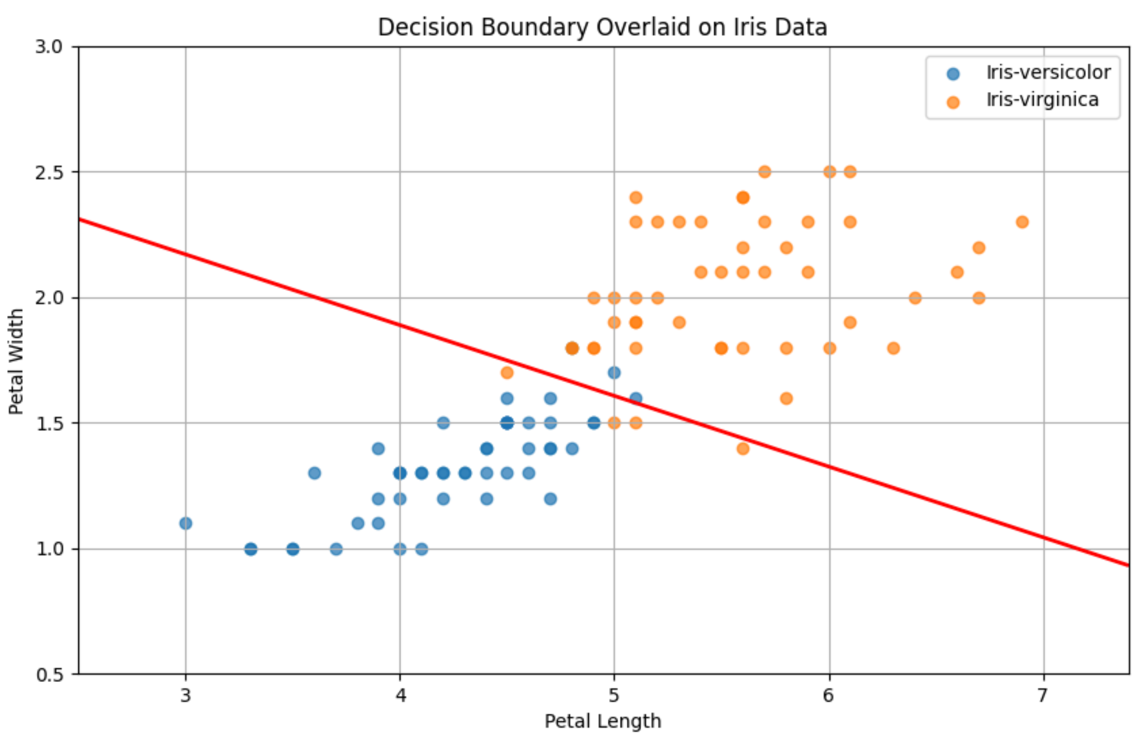
500


Decision Boundary Overlaid on Iris Data
Iris-versicolor
Iris-virginica
Petal Width
Petal Length
3.0
2.5
2.0
1.5
1.0
0.5
3
4
5
6
7

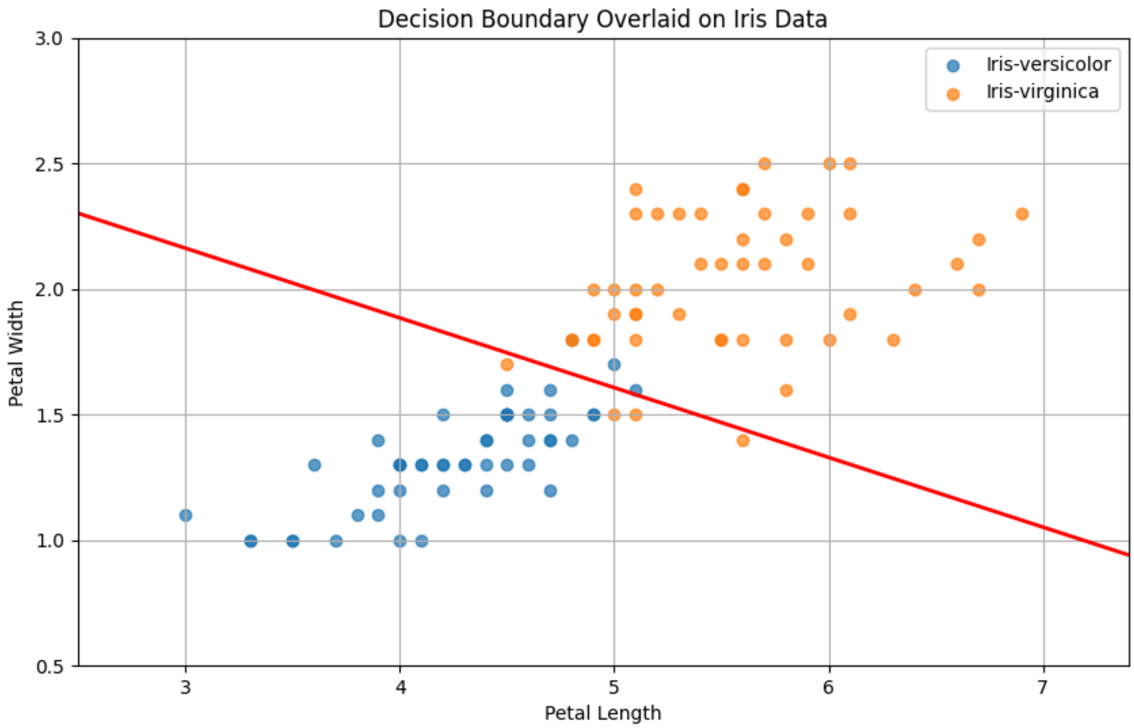
1000


Decision Boundary Overlaid on Iris Data
Iris-versicolor
Iris-virginica
Petal Width
Petal Length
3.0
2.5
2.0
1.5
1.0
0.5
3
4
5
6
7

1500


Decision Boundary Overlaid on Iris Data
Iris-versicolor
Iris-virginica
Petal Width
Petal Length
3.0
2.5
2.0
1.5
1.0
0.5
3
4
5
6
7

2000



c)

Code Snippet

```
np.random.seed(42)
for _ in range(4):
        initial_weights = np.random.uniform(-0.25, 0.25, size=3)
        print(initial_weights)
        decisionBoundaryPlot(data, initial_weights)
        w, mse, i = optimize(data, initial_weights, labels, maxIters=9999,
epsilon=0.01, minDelta=1e-5)
```

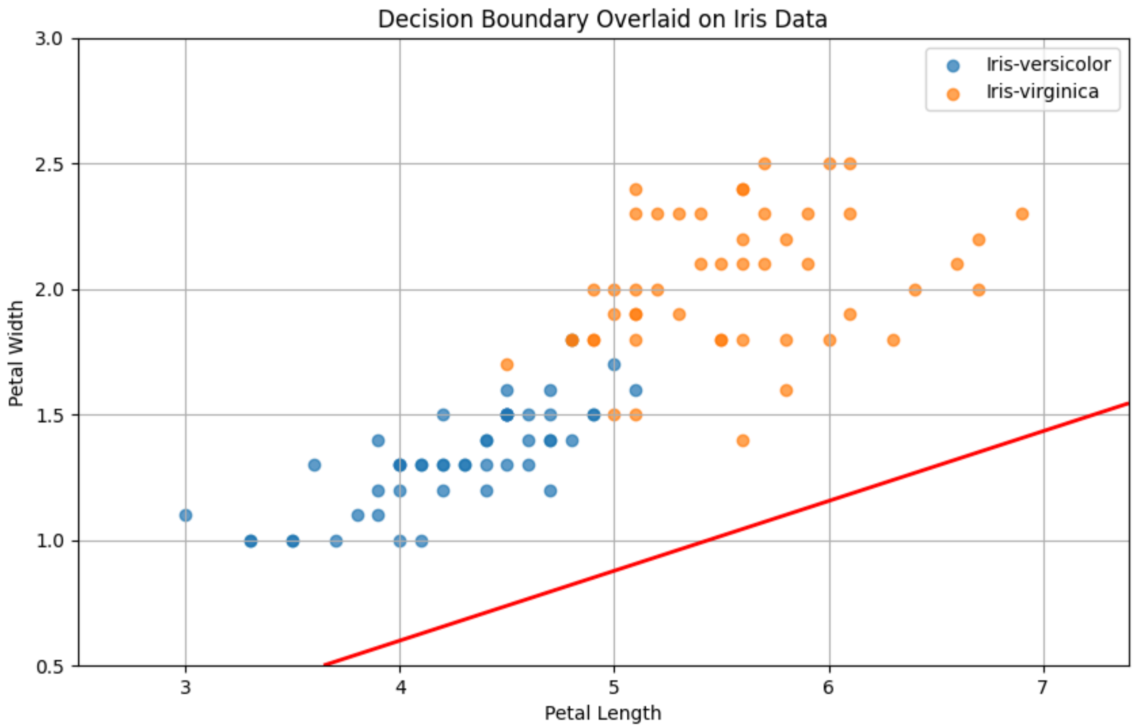
I chose the stopping condition to be some value δ such that $|\text{MSE}_t - \text{MSE}_{t-1}| < \delta$ terminates the loop. In other words, when the improvement according the objective function becomes negligible.

First I ran this with a learning rate $\epsilon = 0.01$

Output

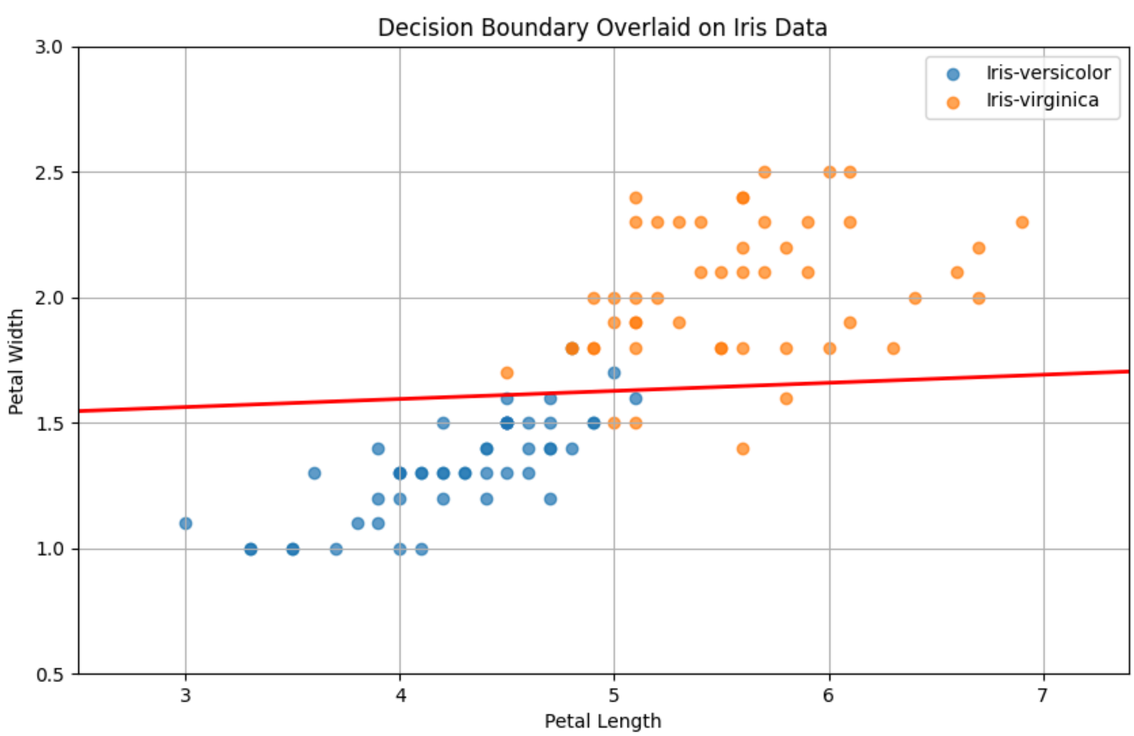
1st Trial

$w = [-0.06272994, 0.22535715, 0.11599697]$



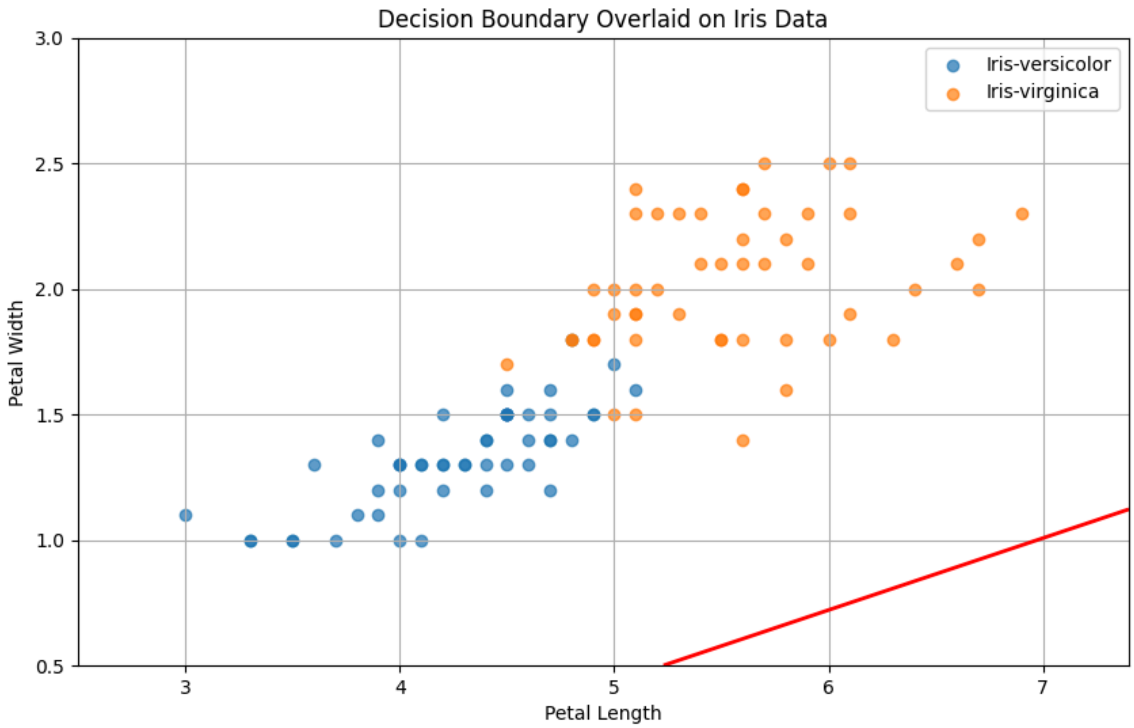
$w_{\text{converged}} = [-0.34854547, 10.8105952, -15.84675652]$

After 352 iterations we had convergence at:



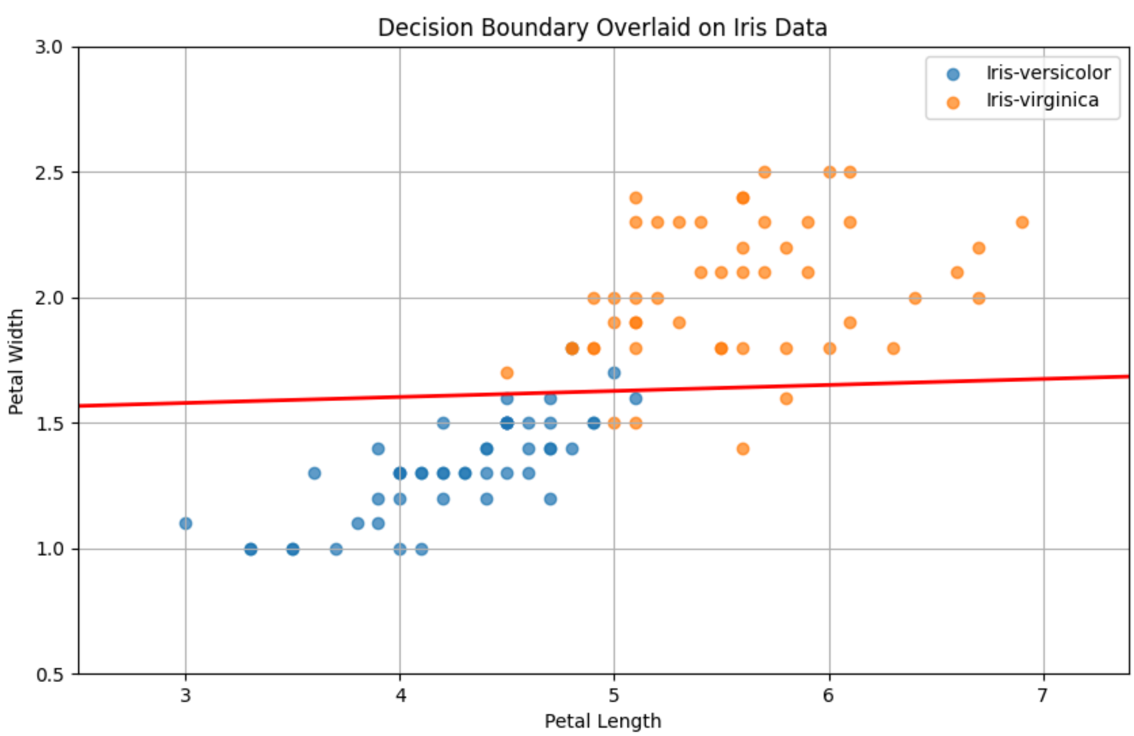
2nd Trial

$w = [0.04932924, -0.17199068, -0.17200274]$



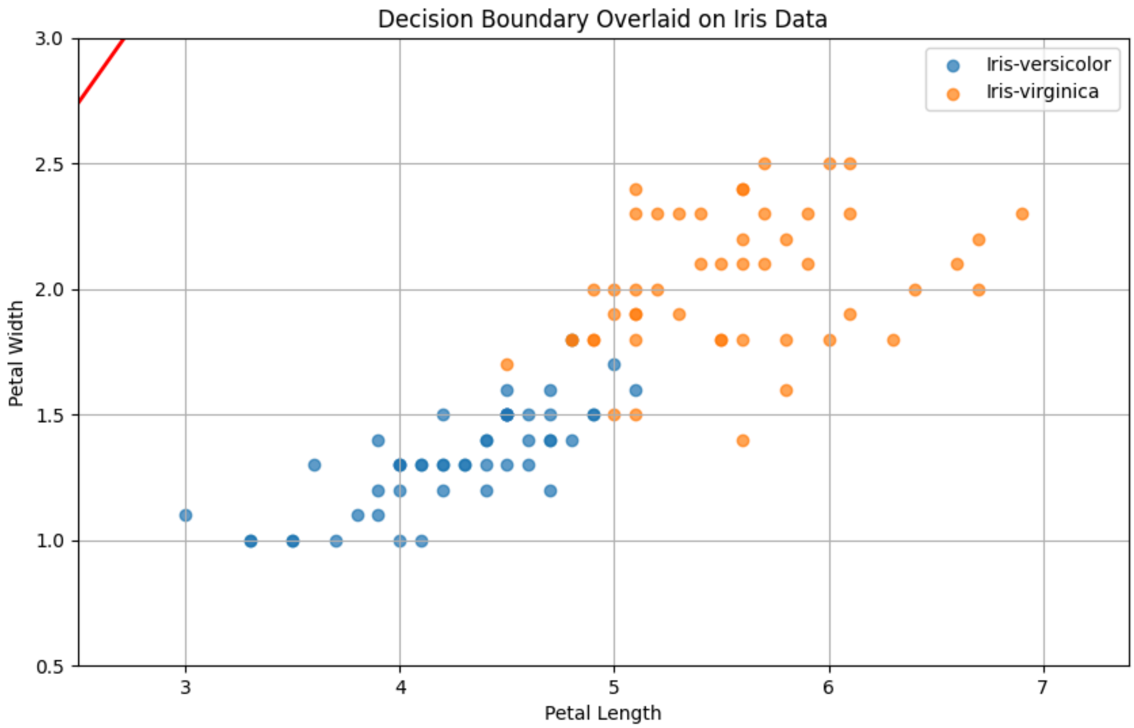
$w_{\text{converged}} = [-0.25486567, 10.63529813, -16.02755514]$

After 350 iterations we had convergence at:



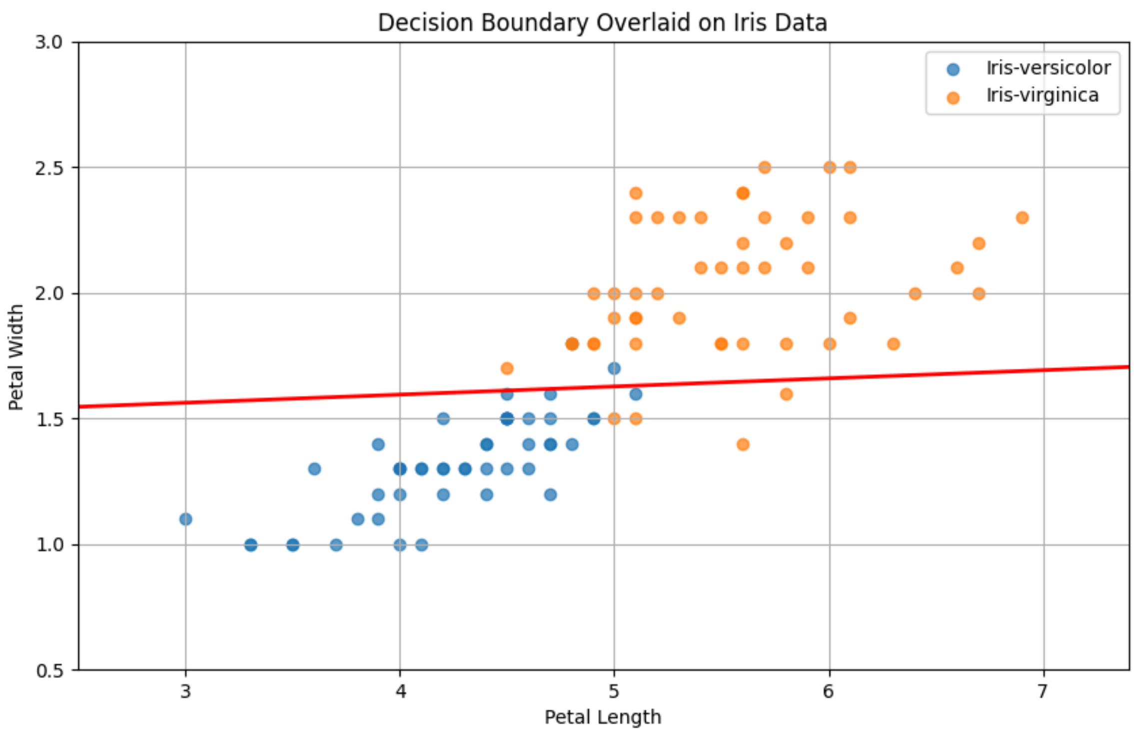
3rd Trial

$w = [-0.22095819, 0.18308807, 0.05055751]$



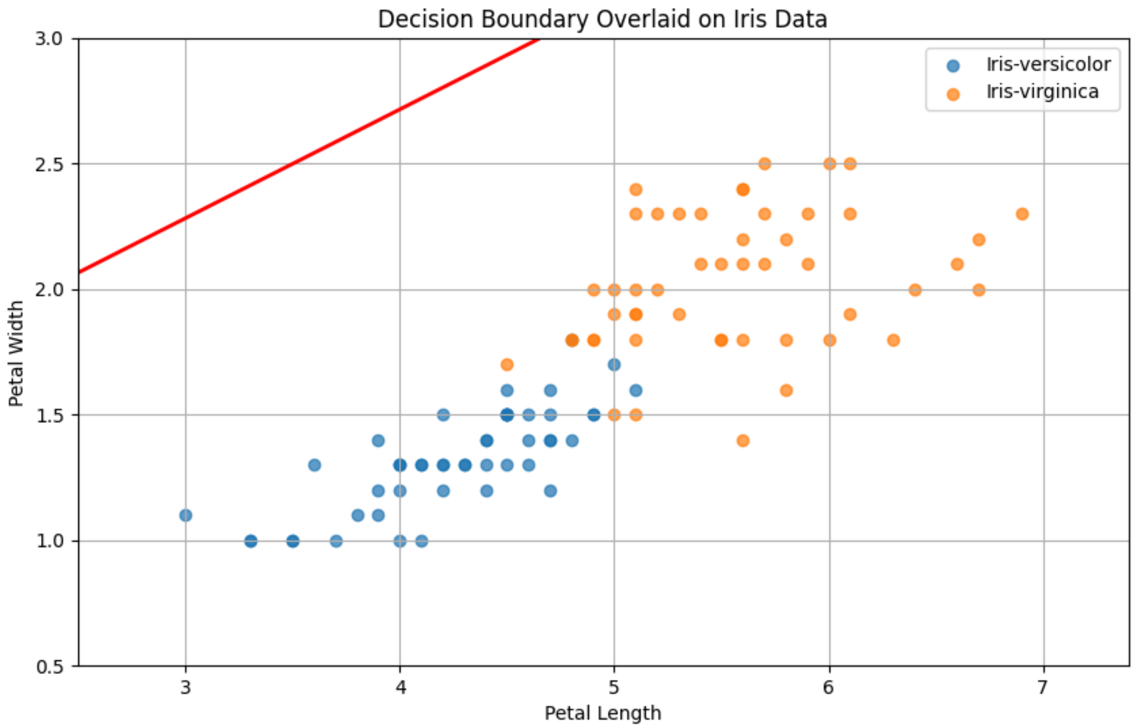
$w_{\text{converged}} = [-0.35071788, 10.81513701, -15.84226559]$

After 350 iterations we had convergence at:

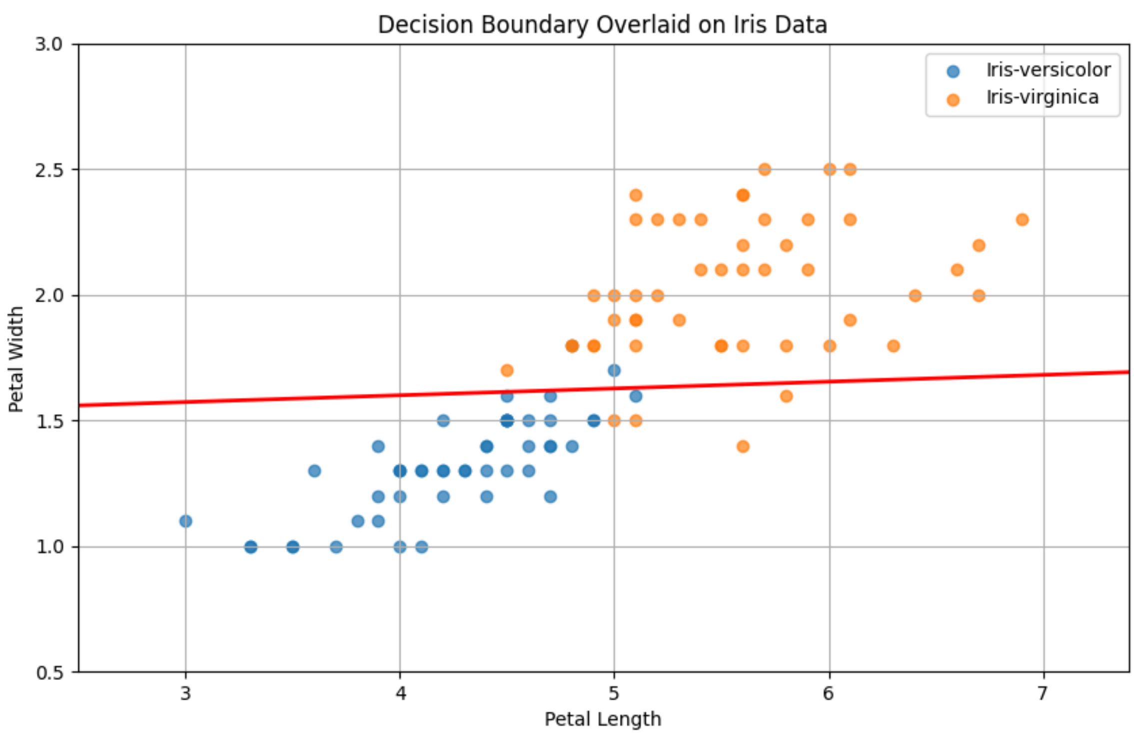


4th Trial

$w = [0.10403629, -0.23970775, 0.23495493]$



$w_{\text{converged}} = [-0.28950702, 10.70004222, -15.96086814]$
After 355 iterations we had convergence at:

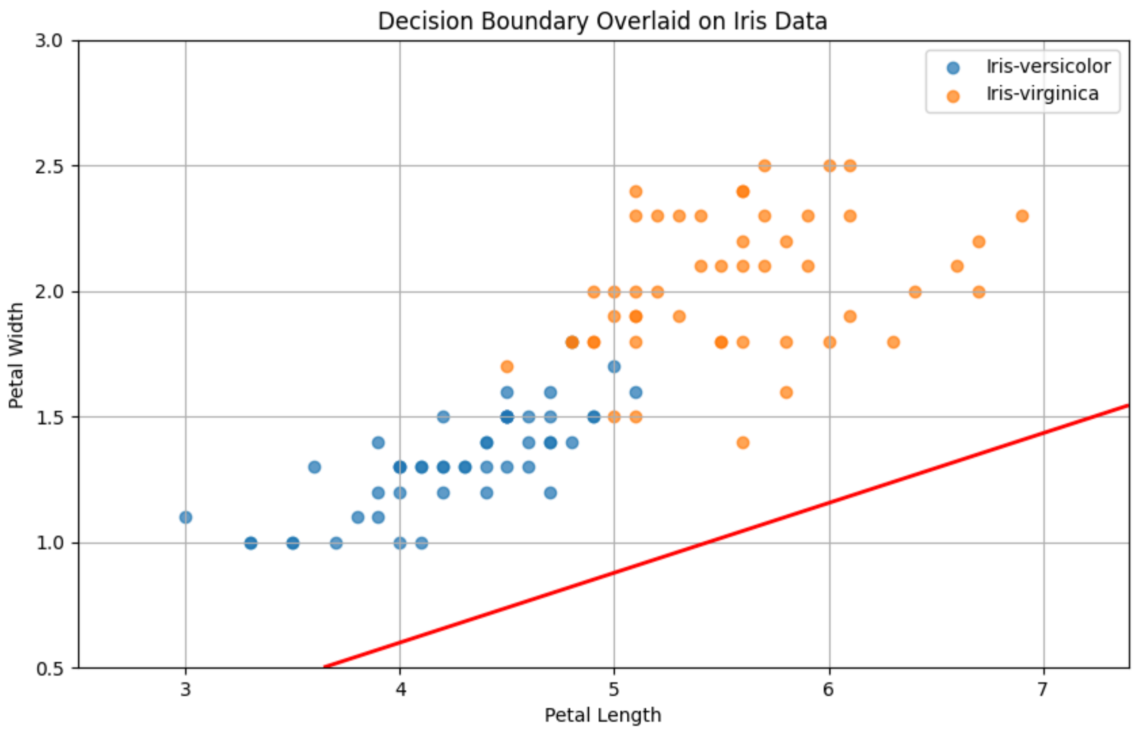


Next I ran it with a considerably smaller $\epsilon = 0.0001$.

Output

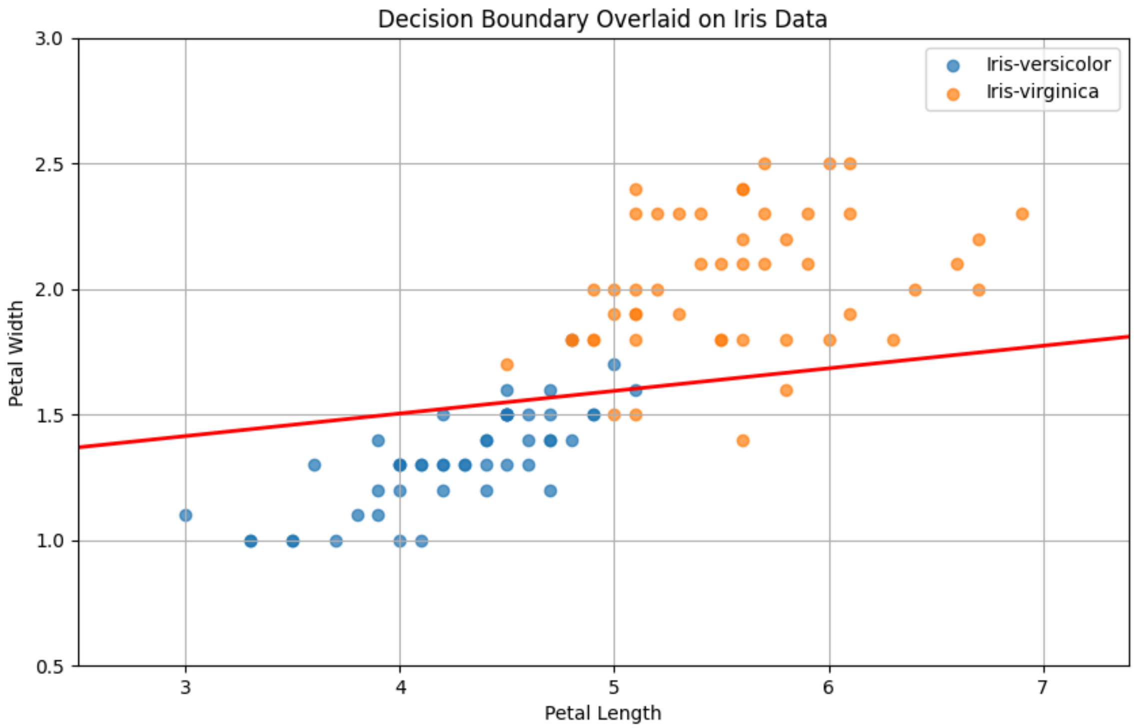
1st Trial

$w = [-0.06272994, 0.22535715, 0.11599697]$



$w_{\text{converged}} = [-0.21101993, 2.34299673, -2.68027061]$

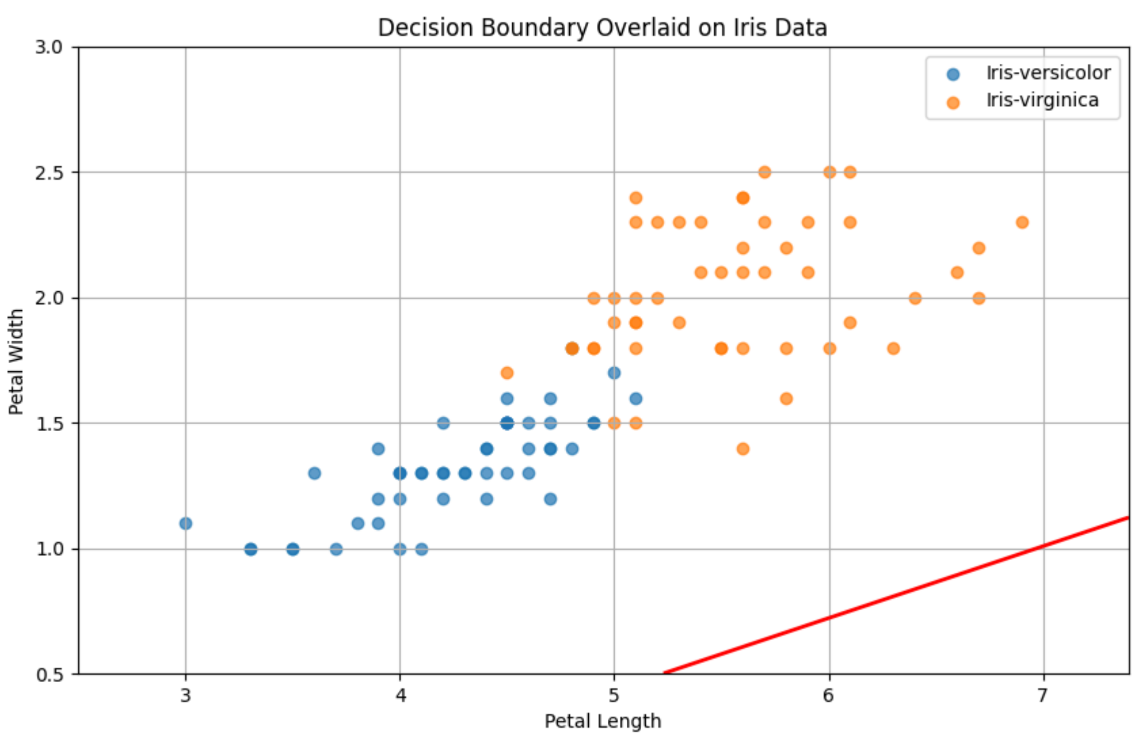
After 5370 iterations we had convergence at:



With a final $\mathrm{MSE} = 0.13307768698447137$

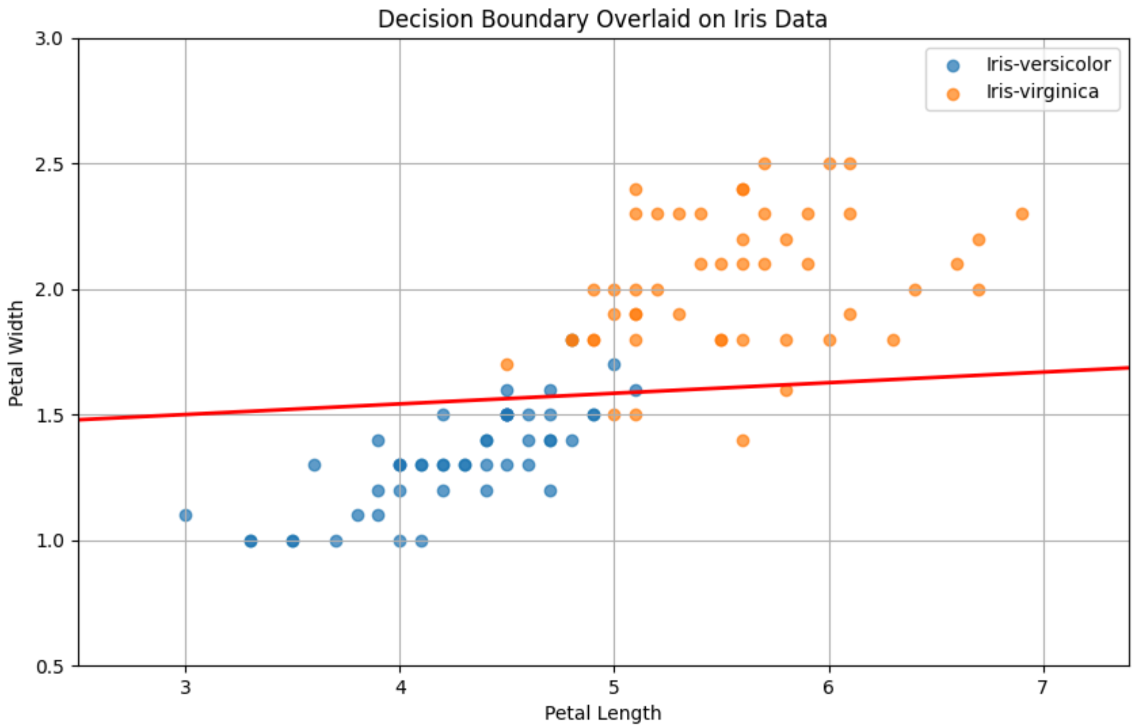
2nd Trial

$w = [0.04932924, -0.17199068, -0.17200274]$



$w_{\text{converged}} = [-0.09012191, 2.13293405, -2.92941176]$

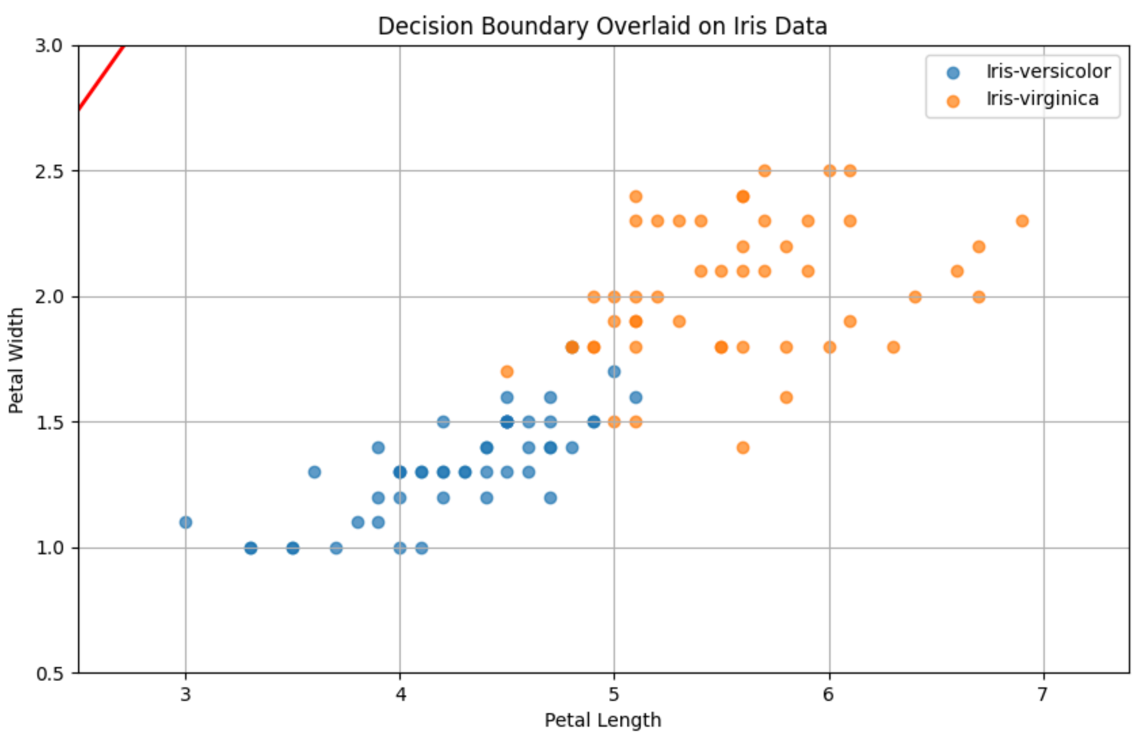
After 5405 iterations we had convergence at:



With a final $\mathrm{MSE} = 0.13063885216705365$

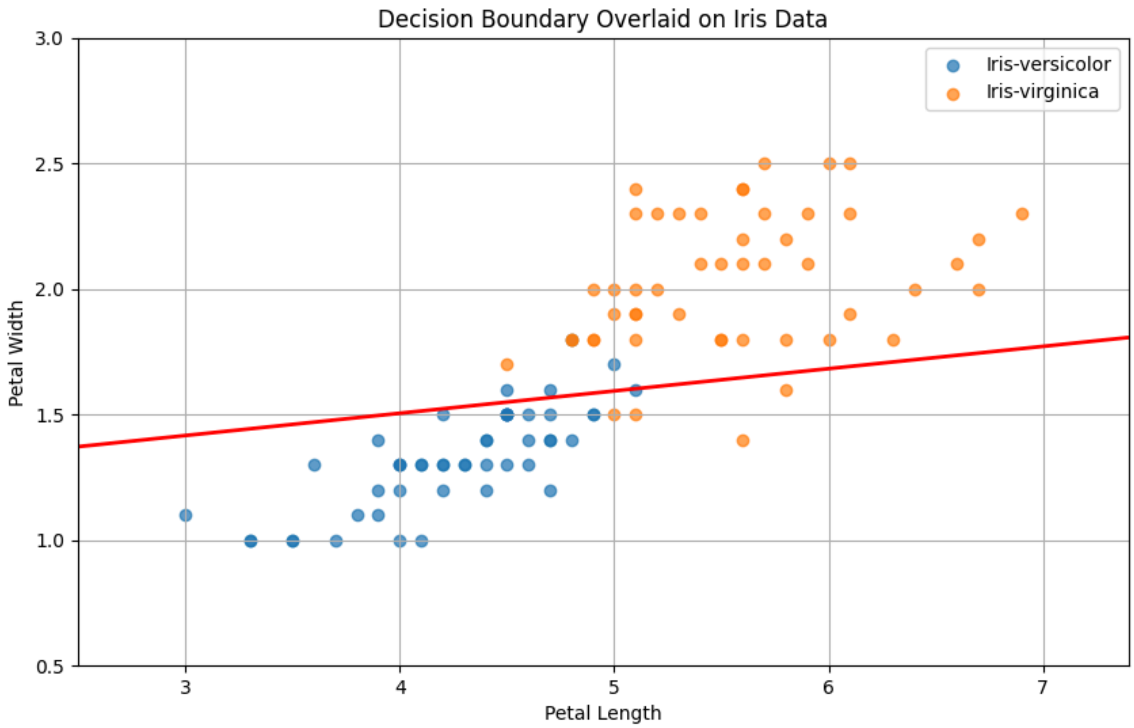
3rd Trial

$w = [-0.22095819, 0.18308807, 0.05055751]$



$w_{\text{converged}} = [-0.20769341, 2.33715552 - 2.68703228]$

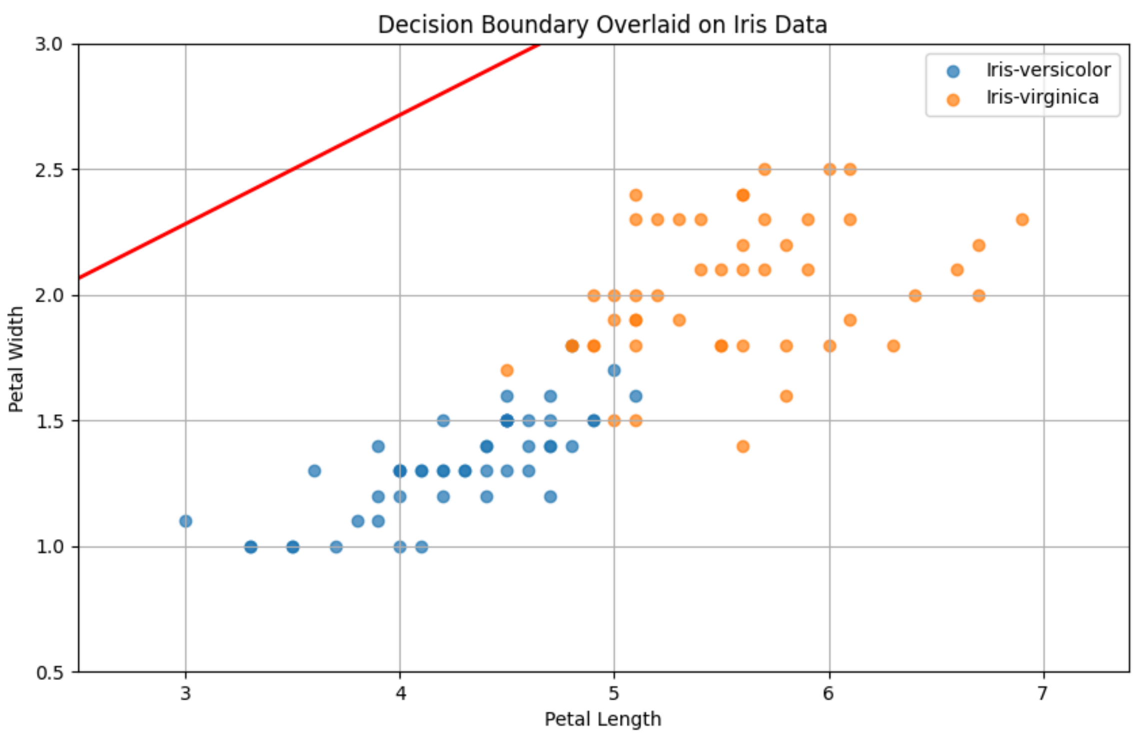
After 5331 iterations we had convergence at:



With a final $\text{MSE} = 0.133007819921006666$

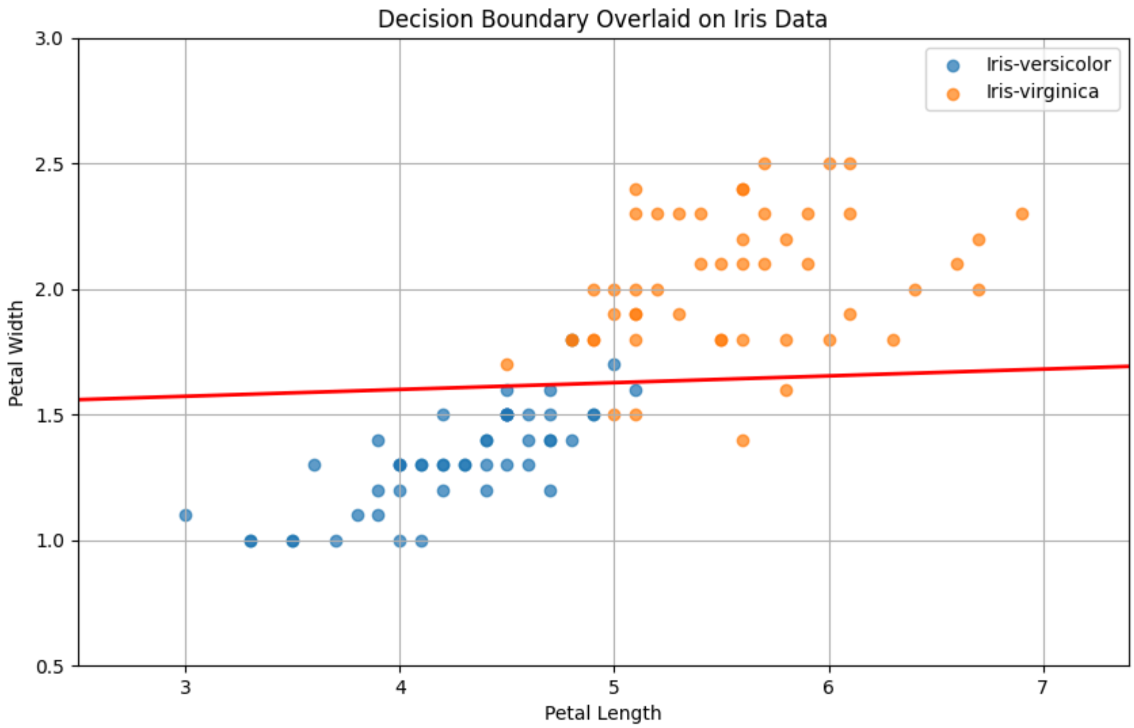
4th Trial

$w = [0.10403629, -0.23970775, 0.23495493]$



$w_{\text{converged}} = [-0.14074069, 2.21954725, -2.82293308]$

After 5795 iterations we had convergence at:



With a final $\mathrm{MSE} = 0.13166561957875447$

d)

Code Snippet

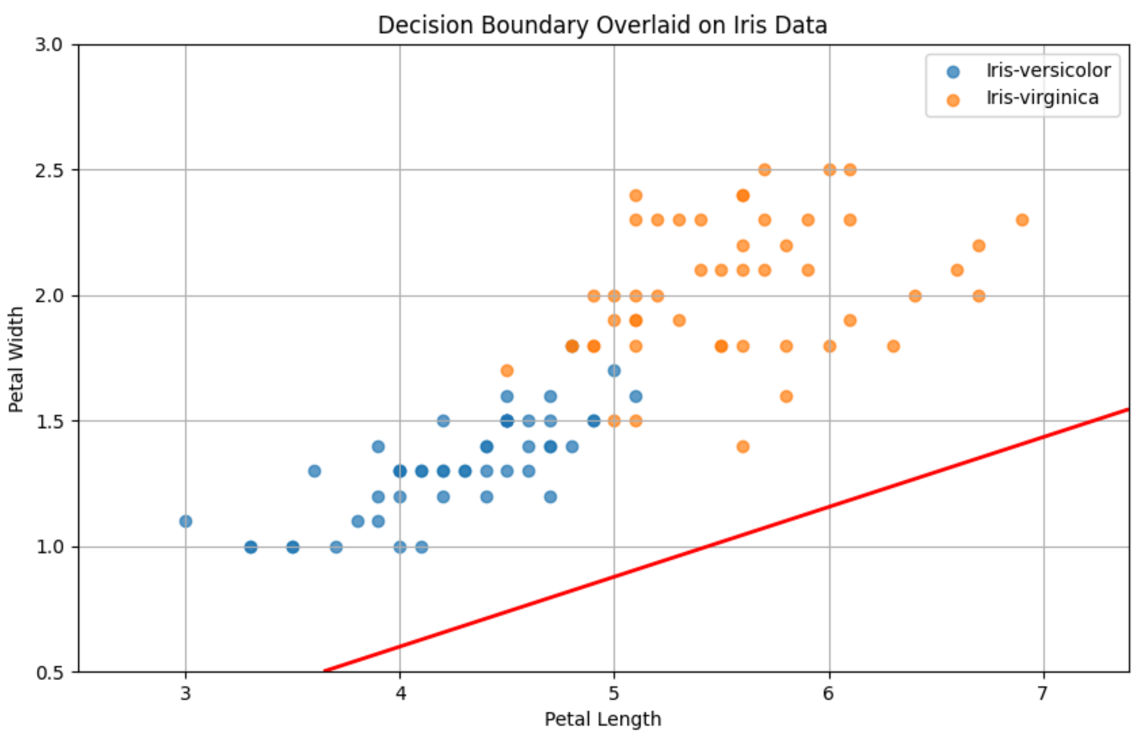
```
def optimize_tracked(data, weights, labels, epsilon=0.01, maxIters=999,
minDelta=1e-6):
    mse_history = []  # Store MSE for each iteration
    last_loss = None
    convergedIters = -1
    initial_mse = None
    half_mse_reached = False
    half_mse_iteration = -1
    half_mse_weights = None

    for i in range(maxIters):
        grad = gradient(data, weights, labels)
        weights -= epsilon * grad

        mse_value = mse(data, weights, labels)
        mse_history.append(mse_value)

        if i == 0:
            initial_mse = mse_value
            last_loss = None  # Initialize properly
```

```

        if not half_mse_reached and mse_value < (initial_mse / 2):
            half_mse_reached = True
            half_mse_iteration = i
            half_mse_weights = weights.copy()

        if last_loss is not None and abs(mse_value - last_loss) < minDelta:
            print(f"Converged after {i} iterations")
            convergedIters = i
            break

        last_loss = mse_value  # Update after the stopping criterion check

    return weights, mse_history, half_mse_iteration, half_mse_weights, convergedIters

# Main
iris_data = pd.read_csv("irisdata.csv")
data = iris_data[(iris_data['species'] == 'versicolor') | (iris_data['species'] == 'virginica')]
labels = data['species'].map({'versicolor': 0, 'virginica': 1})
data['bias'] = 1

# Initialize random weights and parameters
np.random.seed(42)
initial_weights = np.random.uniform(-1, 1, size=3)
learning_rate = 0.001
max_iters = 9000
epsilon_loss = 1e-6

print("Initial Weights:", initial_weights)

# Plot initial decision boundary
print("Initial Decision Boundary:")
decisionBoundaryPlot(data, initial_weights)

# Optimize weights and track MSE
final_weights, mse_history, half_mse_iteration, half_mse_weights, convergedIters = optimize_tracked(
        data, initial_weights, labels, minDelta=epsilon_loss, maxIters=max_iters, epsilon=learning_rate
)

# Plot decision boundary when error is reduced by half
if half_mse_iteration > -1:
```

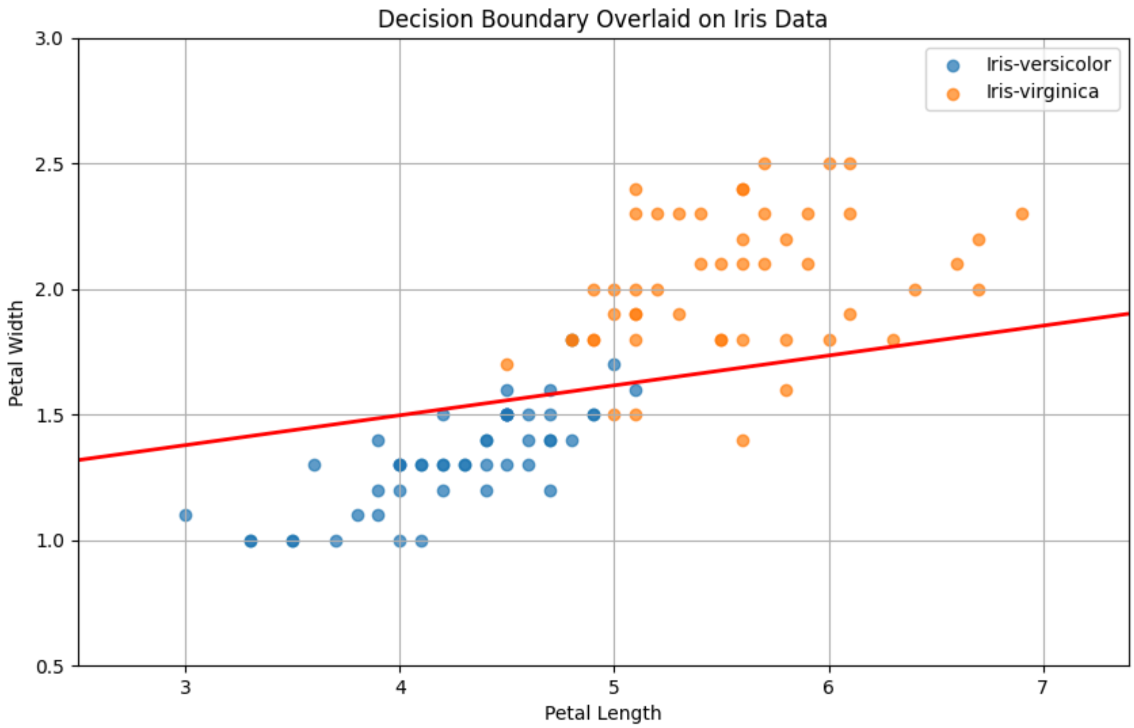
```
        print(f"Decision Boundary after MSE reduced by half (Iteration {half_mse_iteration}):")
        decisionBoundaryPlot(data, half_mse_weights)

# Plot final decision boundary
print(f"Final Decision Boundary after Convergence (Iteration {convergedIters}):")
decisionBoundaryPlot(data, final_weights)

# Plot MSE over iterations
plt.figure(figsize=(10, 6))
plt.plot(range(len(mse_history)), mse_history, label="MSE")
plt.axvline(x=half_mse_iteration, color='orange', linestyle='--', label="MSE Reduced by Half")
plt.xlabel("Iteration")
plt.ylabel("Mean Squared Error")
plt.title("MSE as a Function of Iterations")
plt.legend()
plt.grid(True)
plt.show()
```

Initial

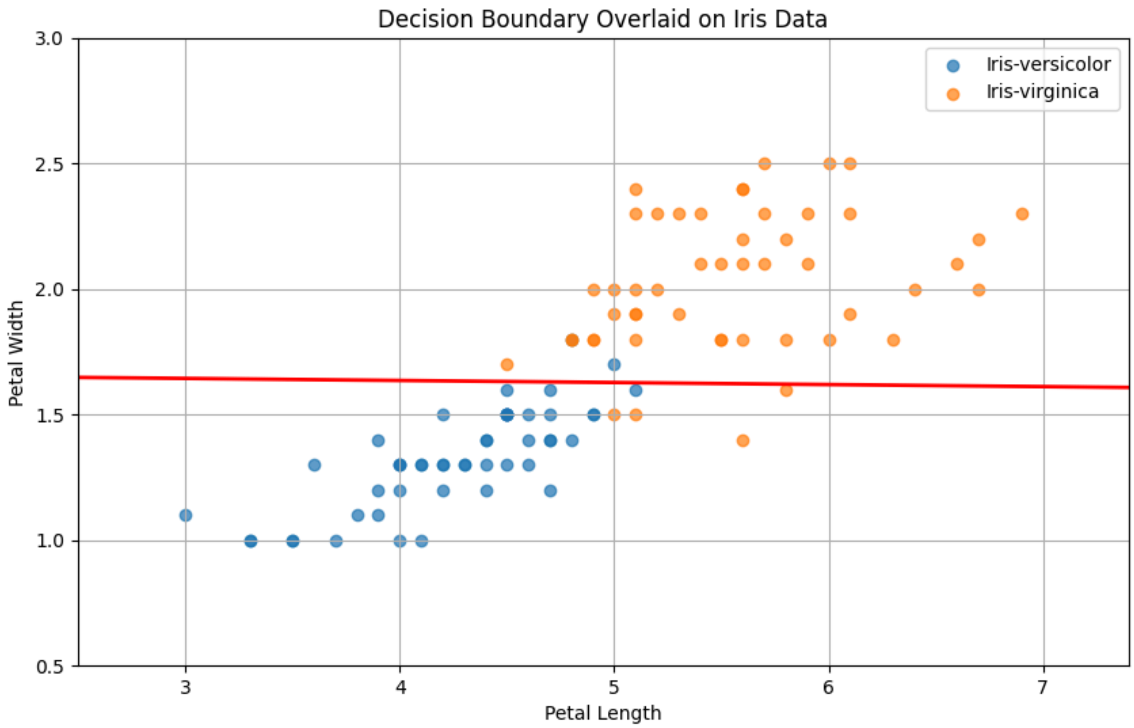
$w = [-0.25091976, 0.90142861, 0.46398788]$

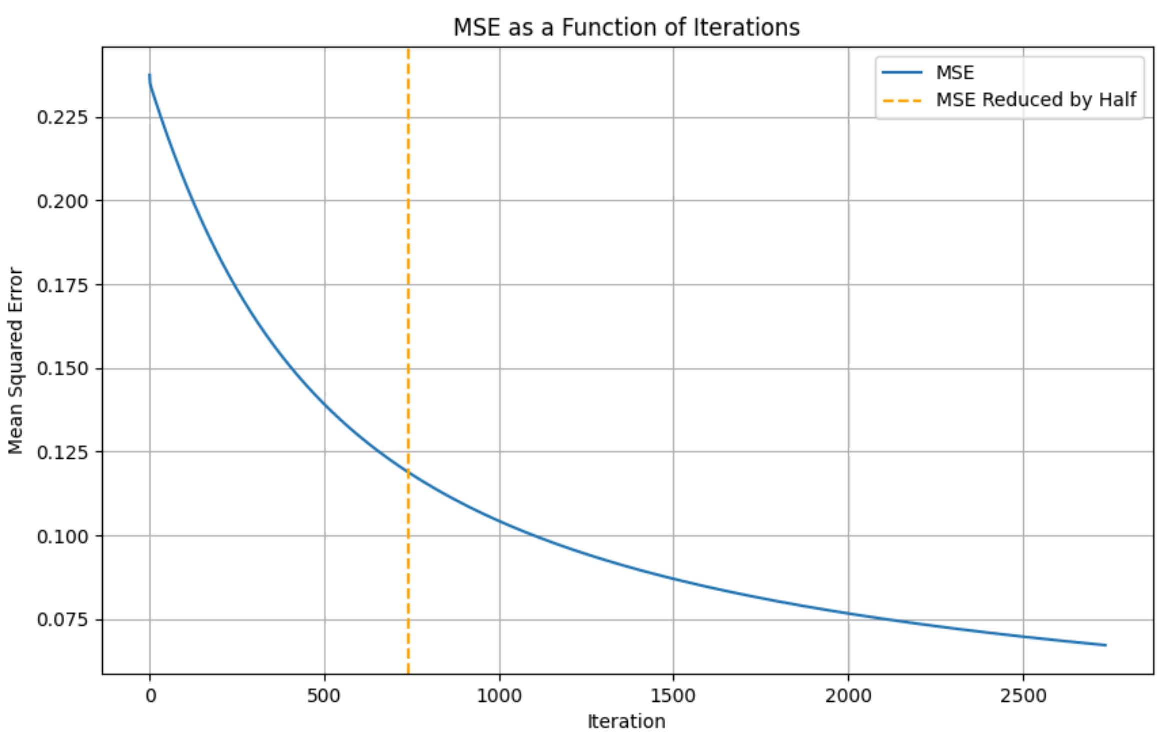


Half Way (742 Iterations)



Final (2736 Iterations)


Decision Boundary Overlaid on Iris Data
Iris-versicolor
Iris-virginica
Petal Width
Petal Length
3.0
2.5
2.0
1.5
1.0
0.5
3
4
5
6
7


MSE as a Function of Iterations
MSE
MSE Reduced by Half
Mean Squared Error
Iteration
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0
500
1000
1500
2000
2500