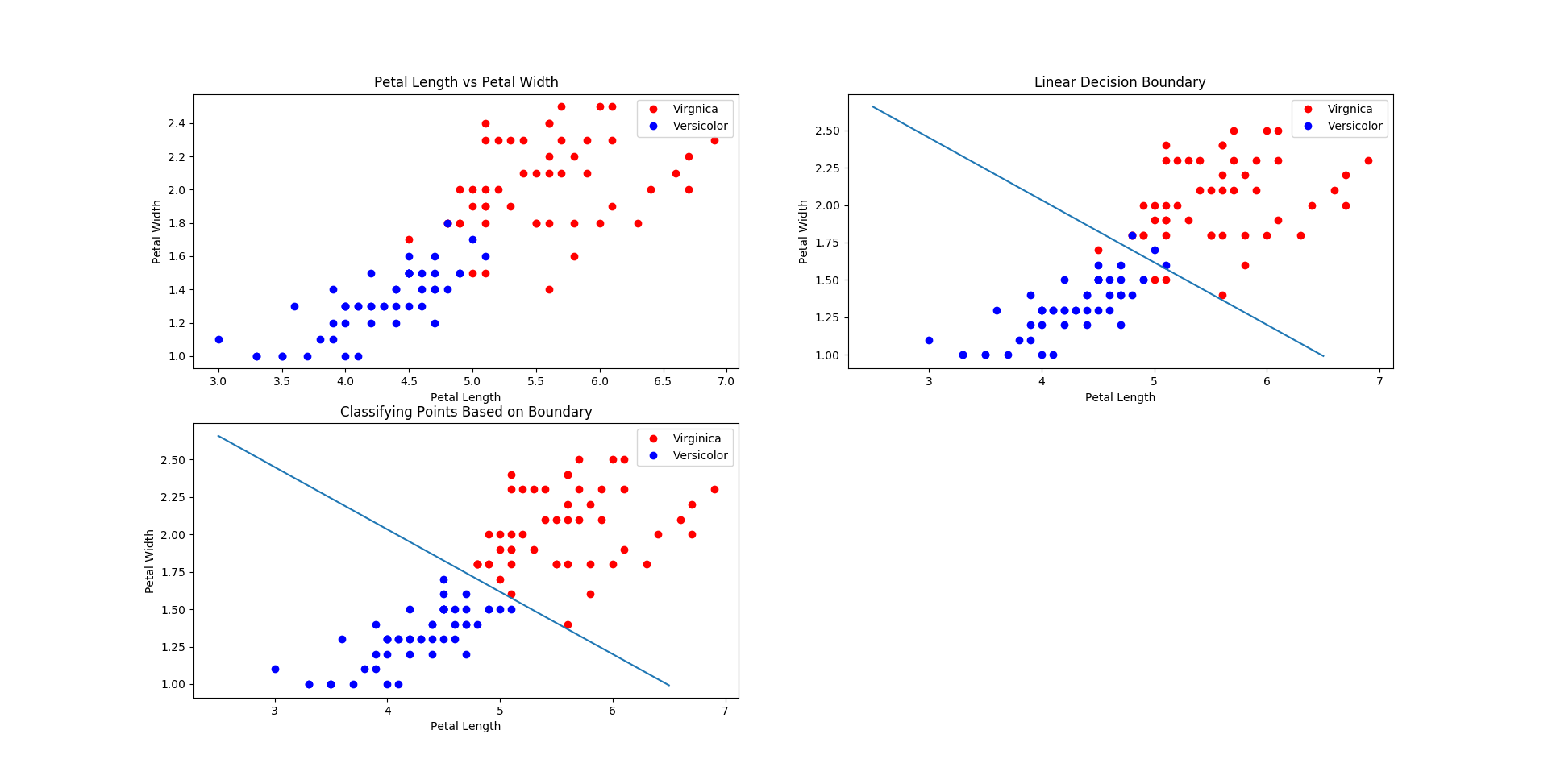
EECS 391 P2 Write Up Isaac Ng



To load the data, I used the below code.

data = openfile("irisdata.csv")  
vir\_length = getdata(data, 2, "virginica")  
vir\_width = getdata(data, 3, "virginica")  
ver\_length = getdata(data, 2, "versicolor")  
ver\_width = getdata(data, 3, "versicolor")  
set\_length = getdata(data, 2, "setosa")  
set\_width = getdata(data, 3, "setosa")

1a. This produced the left top graph.

plt.figure(1)  
plt.subplot(221)  
plt.xlabel("Petal Length")  
plt.ylabel("Petal Width")  
plt.plot(vir\_length, vir\_width, "ro", label="Virgnica")  
plt.plot(ver\_length, ver\_width, "bo", label="Versicolor")  
plt.title("Petal Length vs Petal Width")  
plt.legend()

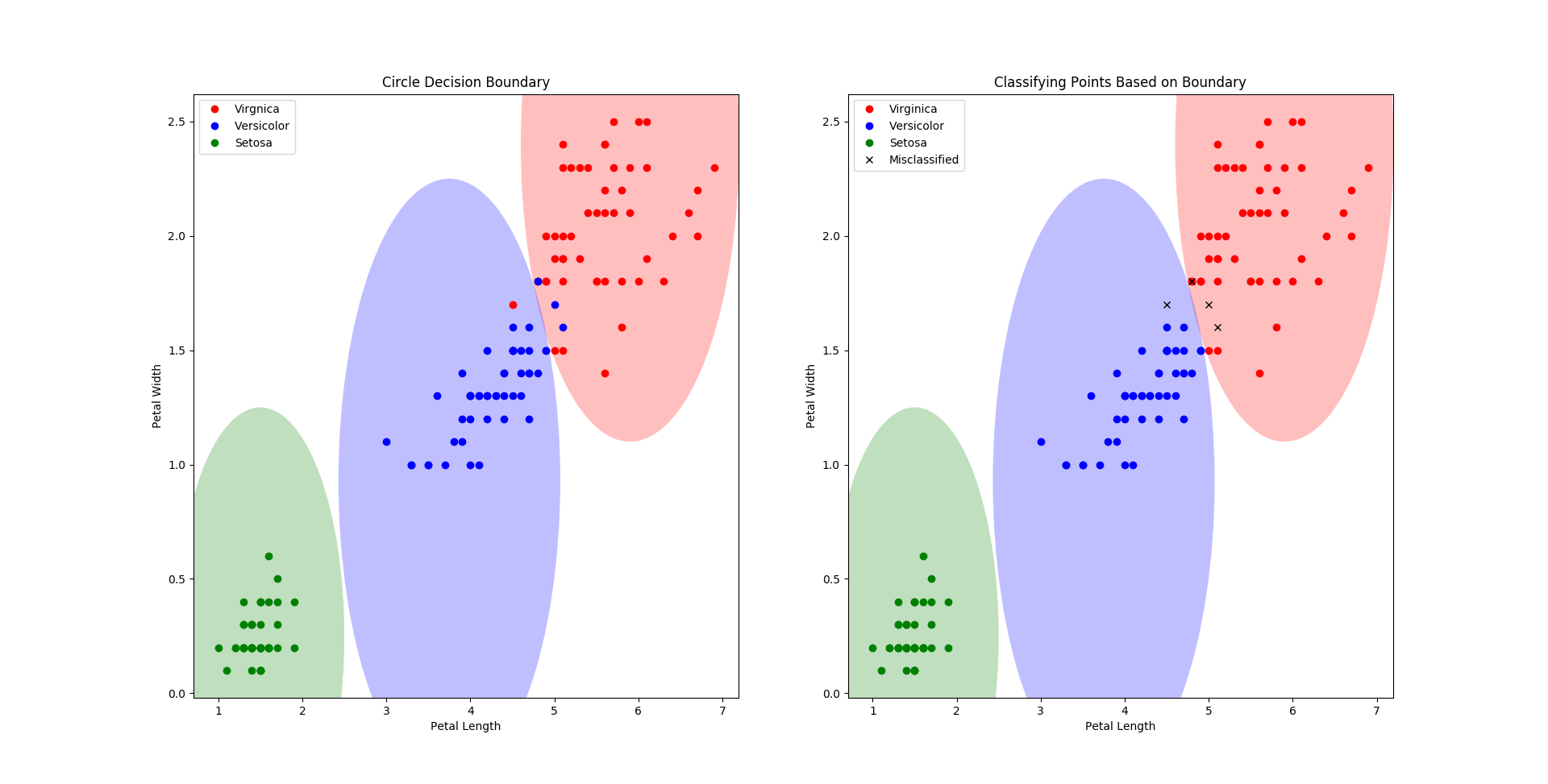
1b. This produces the top right graph. I looked at the graph, approximated where it’s y intercept was and then took the slope by putting the change in y over x. This gave us a line of -2.5/6 \* x + 3.7.

plt.figure(1)  
plt.subplot(222)  
plt.xlabel("Petal Length")  
plt.ylabel("Petal Width")  
plt.plot(vir\_length, vir\_width, "ro", label="Virgnica")  
plt.plot(ver\_length, ver\_width, "bo", label="Versicolor")  
x = np.arange(2.5, 7)  
y = -2.5 / 6 \* x + 3.7  
plt.plot(x, y)  
plt.title("Linear Decision Boundary")  
plt.legend()

1c. This produces the bottom left graph. Basically, if the points were above the line, it was classified as virginica and if it was below the line, it was versicolor. Refer to the code for the classify method.

virginica = splitdata(data, "virginica")  
versicolor = splitdata(data, "versicolor")  
plt.figure(1)  
plt.subplot(223)  
classify(virginica, versicolor, -2.5 / 6, 3.7)

1d. Refer to the code for circleclassify. I looked at the graph to determine the location of the centers and estimated the radius starting at 1 and moving up by .1 increments if it didn’t cover an appropriate amount of points. I classified the points as being within the respective circles. If the point‘s straight line distance was between the circles center and the outer limit of the radius, it was classified as being within the circle and then assigned the respective name of that circle. If the point was misclassified, I made an x where the point was, indicating that it was another type but got put into the other type based on the circle boundary.



plt.figure(2)  
plt.subplot(121)  
plt.xlabel("Petal Length")  
plt.ylabel("Petal Width")  
plt.title("Circle Decision Boundary")  
plt.plot(vir\_length, vir\_width, "ro", label="Virgnica")  
plt.plot(ver\_length, ver\_width, "bo", label="Versicolor")  
plt.plot(set\_length, set\_width, "go", label="Setosa")  
circle = plt.Circle((5.9, 2.4), radius=1.3, fc='r', alpha=0.25)  
plt.gca().add\_patch(circle)  
circle = plt.Circle((3.75, .93), radius=1.32, fc='b', alpha=0.25)  
plt.gca().add\_patch(circle)  
circle = plt.Circle((1.5, .25), radius=1, fc='g', alpha=0.25)  
plt.gca().add\_patch(circle)  
plt.legend()  
setosa = splitdata(data, "setosa")  
circleclassify(virginica, versicolor, setosa)

2a. Below is my mse function. Check my code for additional functionality.

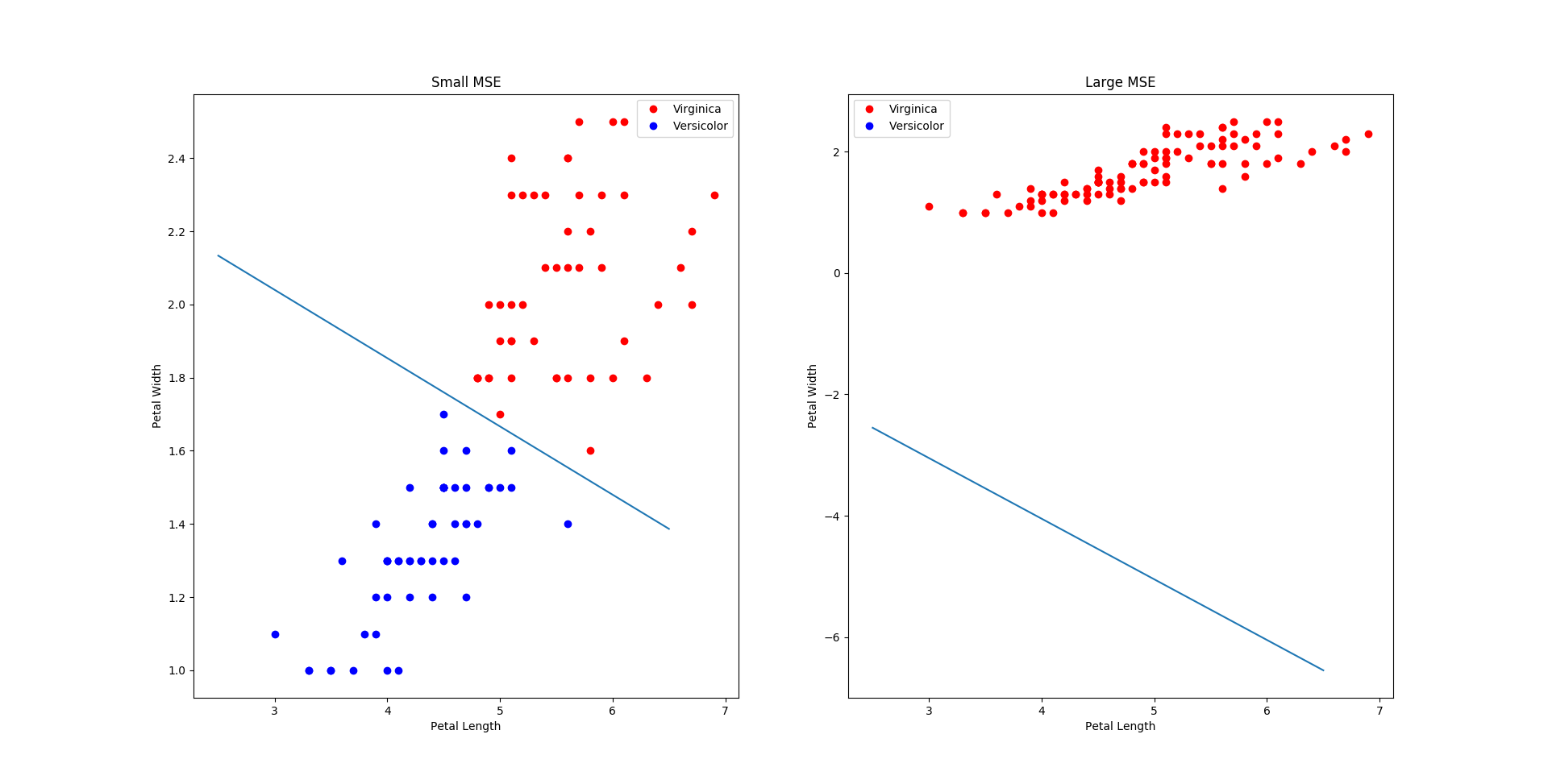
def mse(data, boundary, pattern):  
 virginica = np.array(splitdata(data, pattern[0]))[:, 2:4].astype(np.float32)  
 versicolor = np.array(splitdata(data, pattern[1]))[:, 2:4].astype(np.float32)  
 total = np.append(virginica, versicolor, 0)  
 labels = np.append(np.ones(50), -np.ones(50))  
 w = boundary[0]  
 b = boundary[1]  
 errors = np.dot(total, w) + b - labels  
 return (errors \*\* 2).mean()

2b. Below is the graph produced and the code used to produce it. As we can see the small mse is relatively accurate while the large mse misclassifies the points as all virginica due to them all being above the line. The below mse values are copied form the console window.

Small mse is 0.283306245243

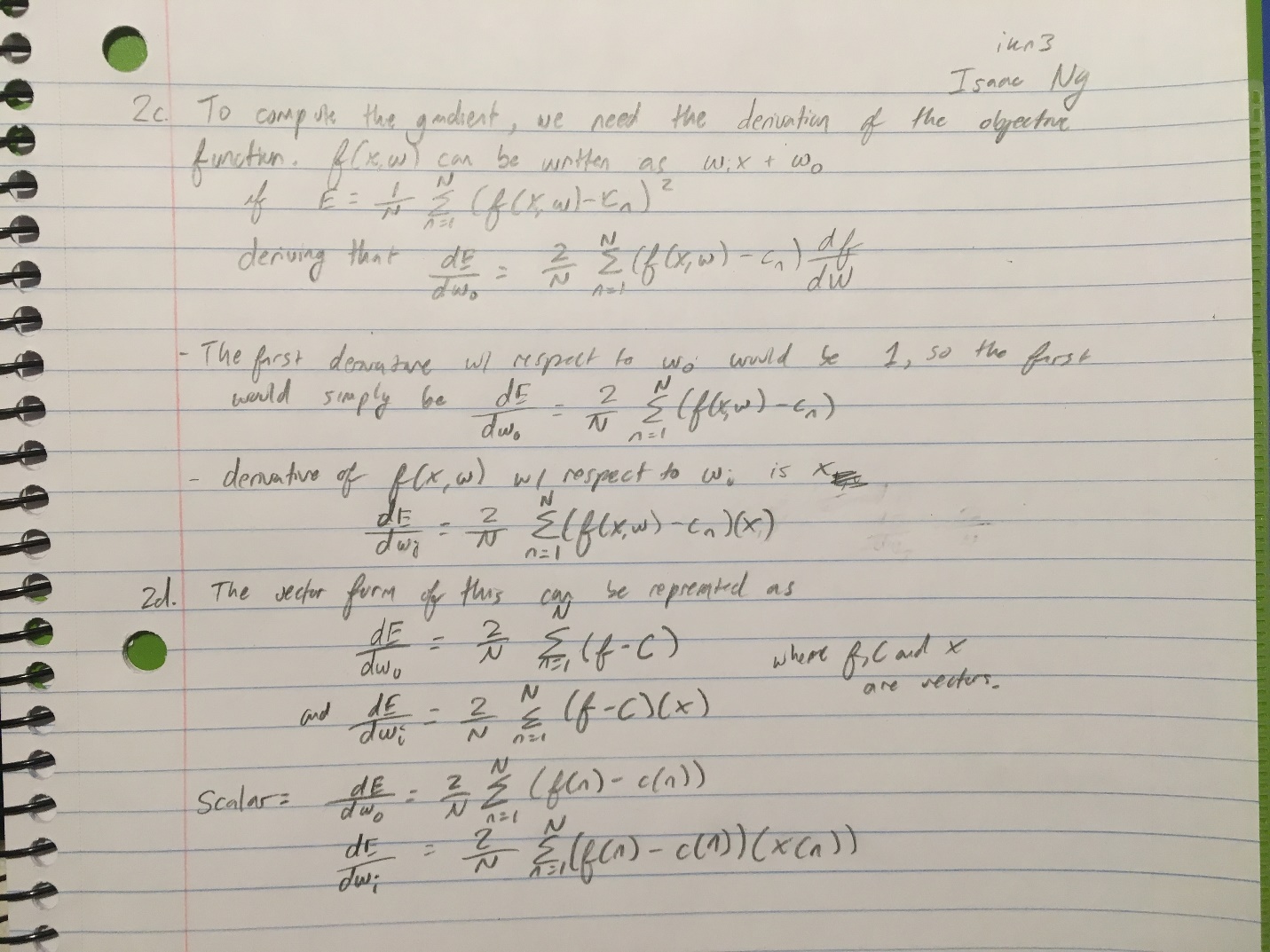
Large mse is 4521.92995719

w = np.array([.28, 1.5])  
b = -3.9  
smallerror = mse(data, (w, b), ("virginica", "versicolor"))  
print("Small mse is", smallerror)  
plt.figure(3)  
plt.subplot(121)  
classify(virginica, versicolor, (-w[0] / w[1]), -b / w[1])  
plt.title("Small MSE")  
w = np.array([10, 10])  
b = .5  
largeerror = mse(data, (w, .5), ("virginica", "versicolor"))  
print("Large mse is ", largeerror)  
virginica = splitdata(data, "virginica")  
versicolor = splitdata(data, "versicolor")  
plt.subplot(122)  
classify(virginica, versicolor, (-w[0] / w[1]), -b / w[1])  
plt.title("Large MSE")

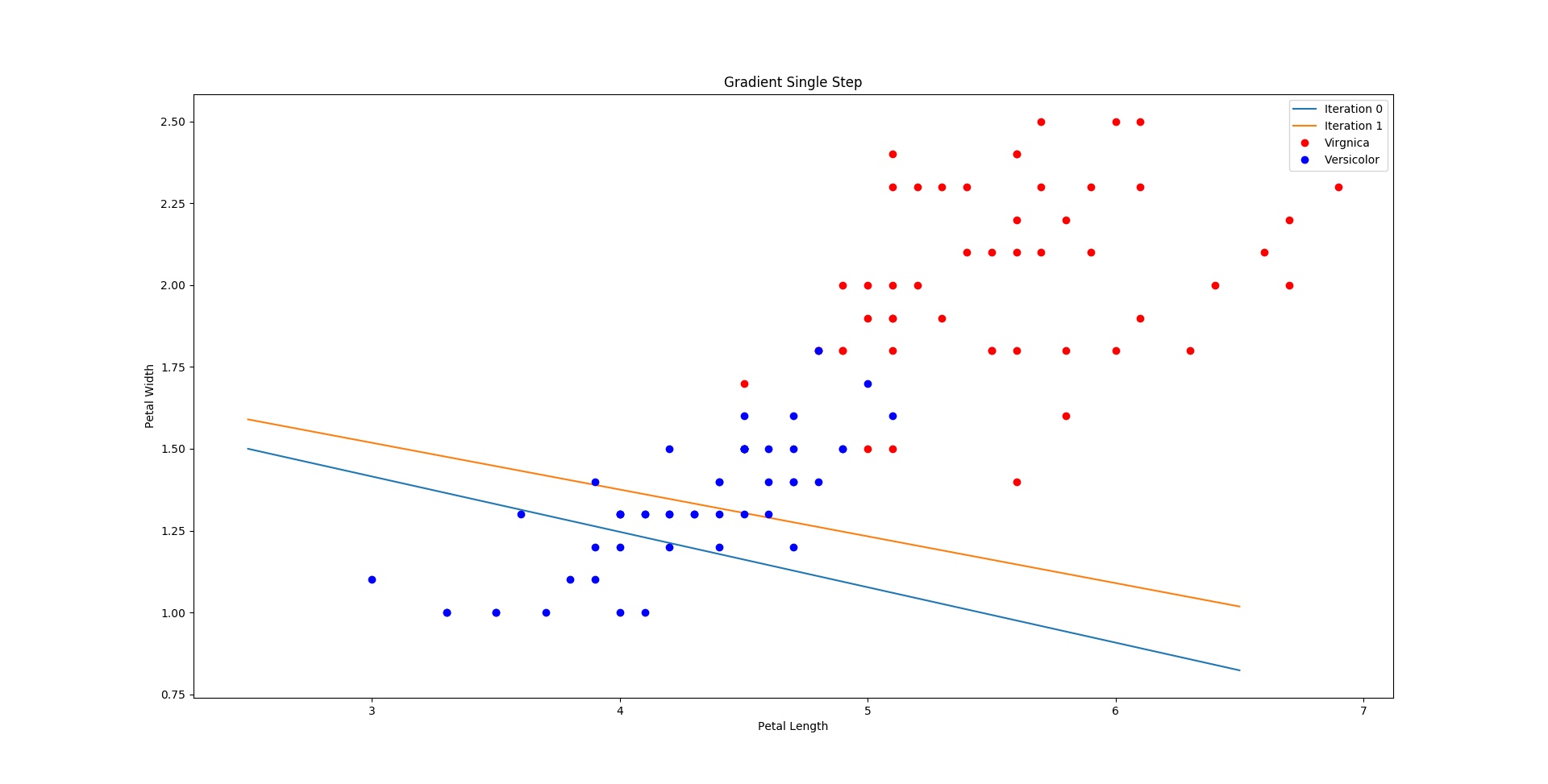


2c. Refer to pictures attached

2d. Refer to pictures attached

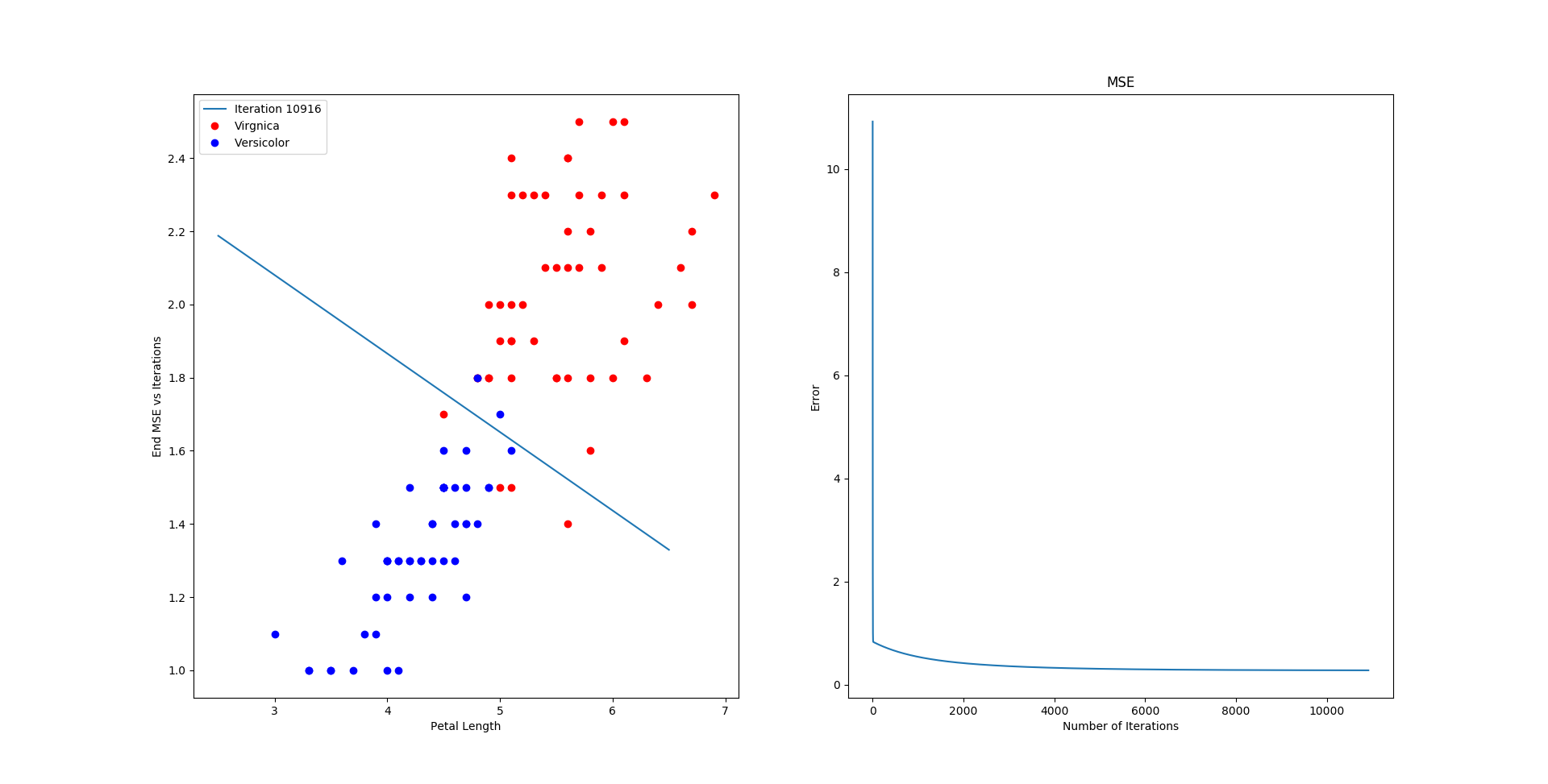
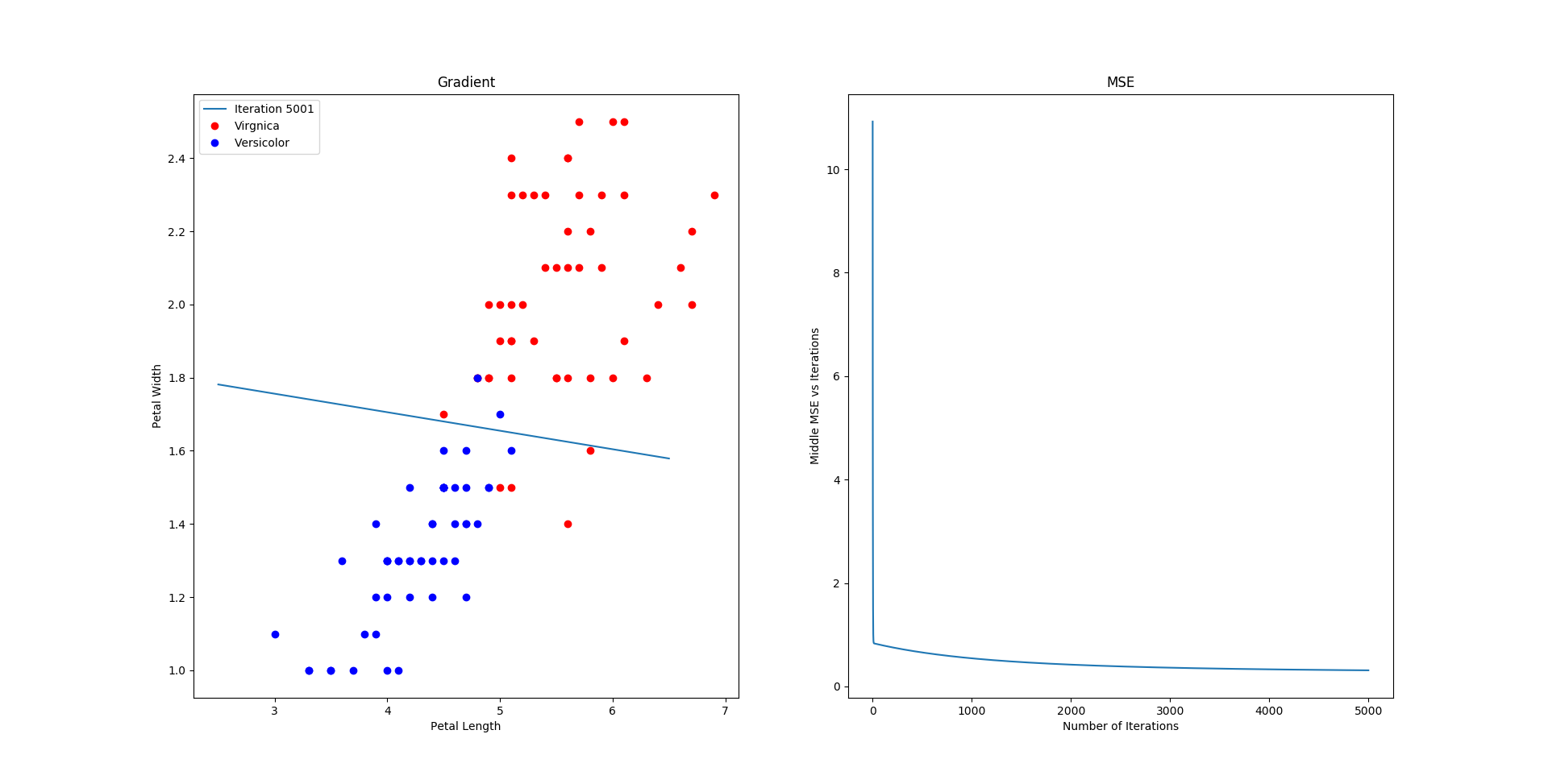
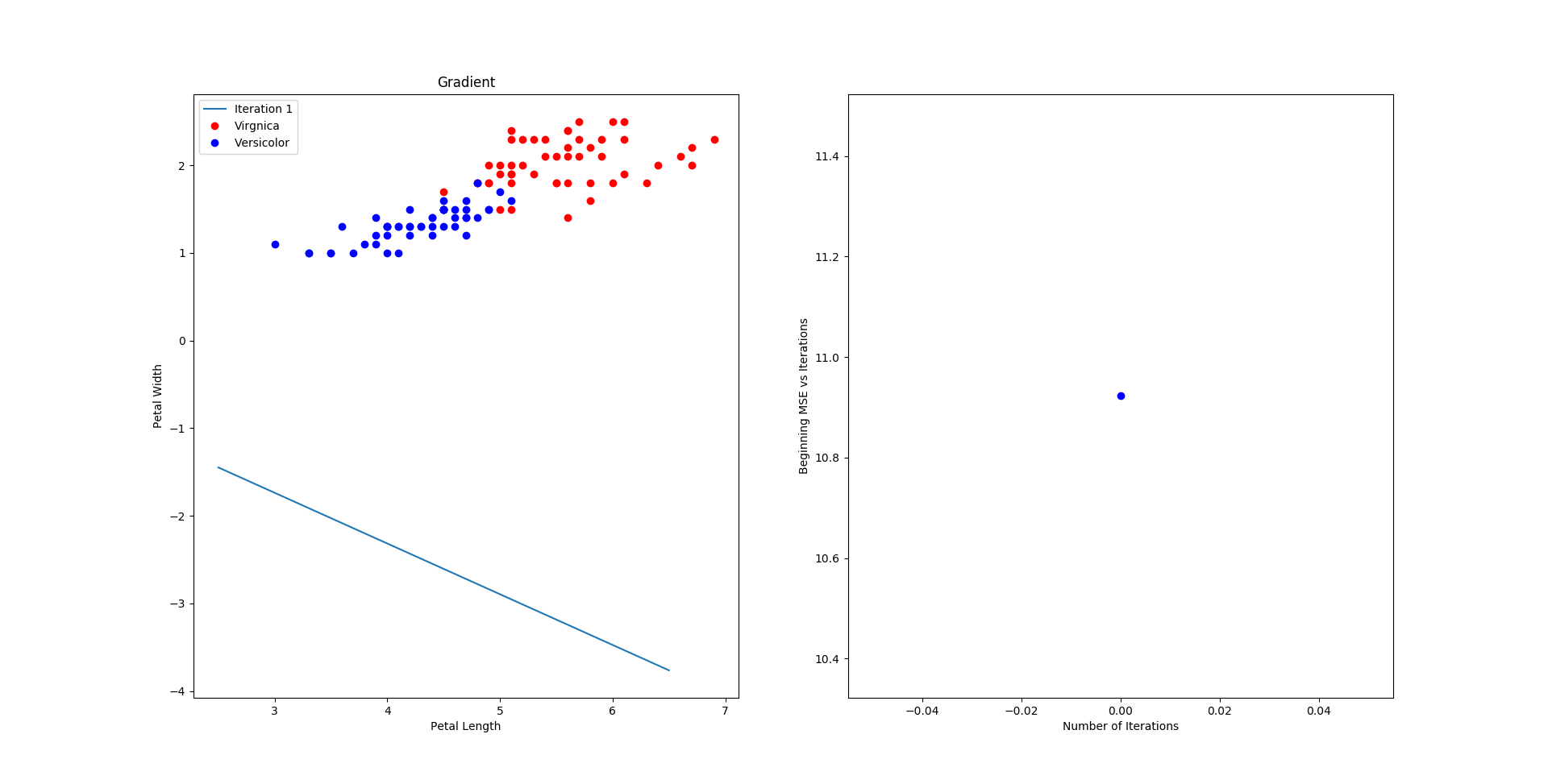


2e. My gradient function is shown below as well as my gradient graph. We see that as the iteration increases, the line begins to move closer to an optimal value.



def gradient(data, boundary, pattern):  
 virginica = np.array(splitdata(data, pattern[0]))[:, 2:4].astype(np.float32)  
 versicolor = np.array(splitdata(data, pattern[1]))[:, 2:4].astype(np.float32)  
 total = np.append(virginica, versicolor, 0)  
 labels = np.append(np.ones(50), -np.ones(50))  
 w = boundary[0]  
 b = boundary[1]  
 errors = np.dot(total, w) + b - labels  
 gradientw = .01 \* np.dot(total.T, errors)  
 gradientb = .01 \* errors.sum()  
 return gradientw, gradientb

3a. For 3a to 3c, see the code. It produces the graphs shown below.



3d. I experimented in order to get the gradient step size. The lecture slides recommended .1/100, but it diverged after a couple of steps. After only taking a couple steps, it was clear that this would diverge. I next chose .001 but it tended to overshoot the graph after a high number of iterations. In the end, I chose .01, which seemed to exhibit the behavior I desired and approximate the optimal decision boundary for the linear decision boundary.

3e. We want the stopping criteria to basically be when the MSE barely adjusts itself. For me, I chose a stopping criterion when the change becomes less than 1e-6. This is extremely small and runs about 10000 iterations before it detects this. We can also see that the MSE flattens and stays around the same value at 10000 indicating that our stopping criteria is fairly accurate.