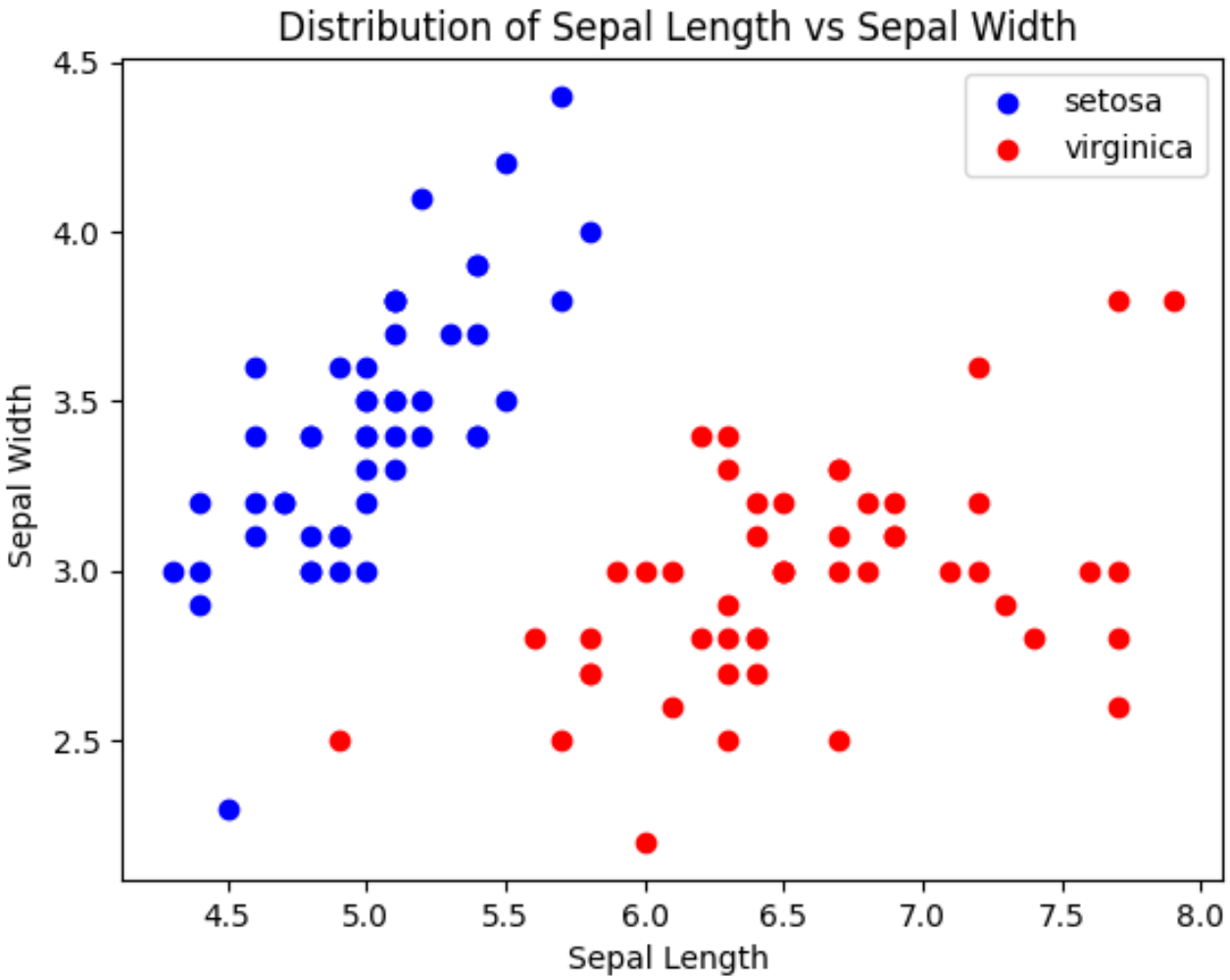
Assignment-2 Report

1.Segregate the data of my choice and plot its distribution.

- I choose iris-setosa and iris-virginica as the label
- Sepal length and sepal width as the input features



2. 80:20 train test split

- I split the data %80 train and %20 as split using sklearn train_test_spilt function both separated 50:50 randomly

```
Training dataset:
Num of setosa samples: 40
Num of virginica samples: 40
Total data in training set: 80
Test Set:
Num of setosa samples: 10
Num of virginica samples: 10
Total data in test set: 20
```

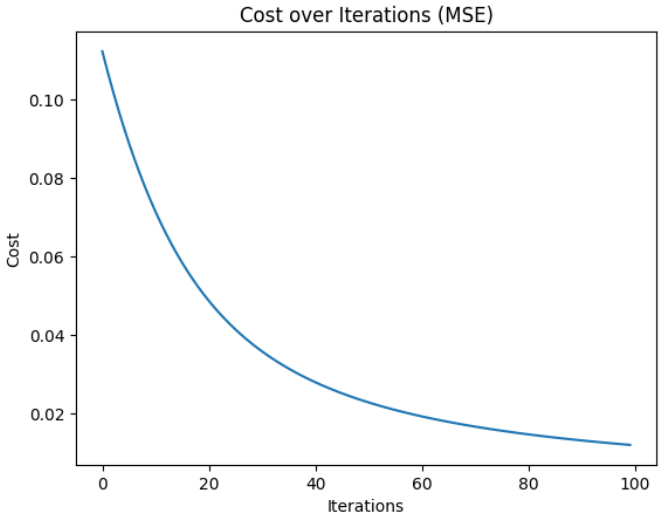
3. Logistic regression model accuracy and cost of the models through iterations for my models

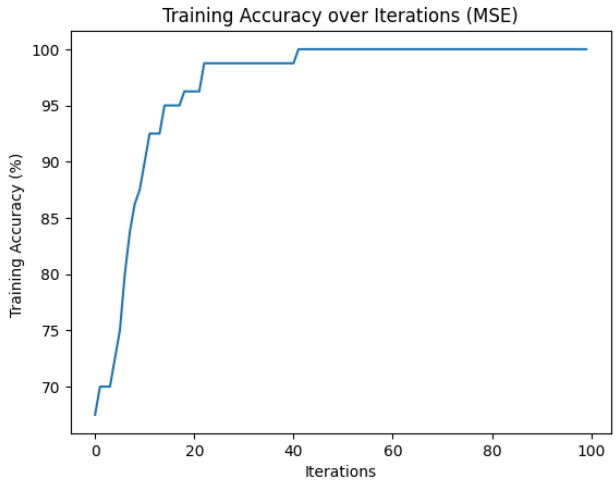
- I used two different cost functions one is log-loss the other is MSE
- For the log losses logistic regression the weights are: final weights: [2.07841092 -0.73141164]

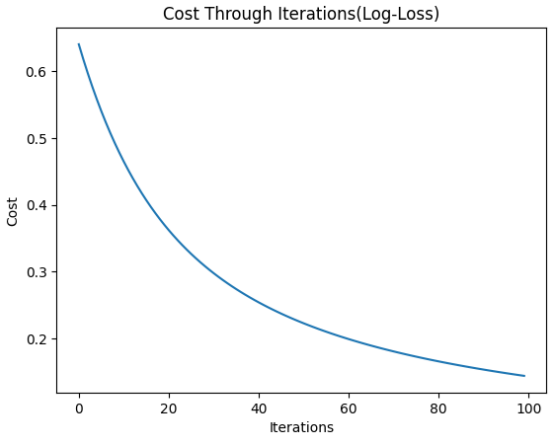
final bias: 0.07620664959869698
final accuracy: % 100.0
final cost: 0.14395353918143203

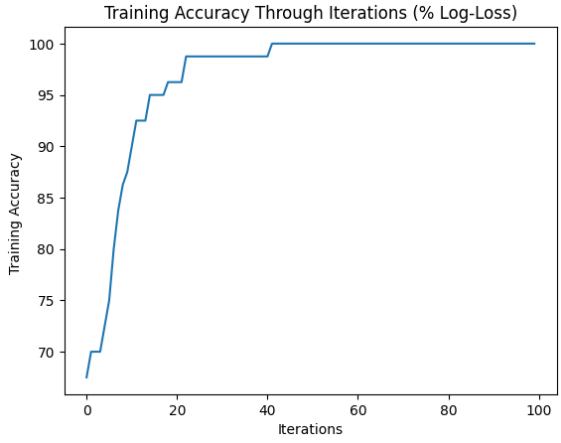
- For the MSE my weight are: Final weights: [2.07841092 -0.73141164]

Final bias: 0.07620664959869698
final accuracy: % 100.0
final cost: 0.011958838540524909









4.Model testing and accuracy for test data

- After training logistic regression model for both MSE cost and Log-loss and both had %100 training accuracy but both performed %95 accuracy in the unseen data
- The test data wasn't as good as training data but it still performs really well and didn't overfit or underfoot

```
test accuracy (Log-Loss) is: % 95.0
test accuracy using OldLogisticRegression (MSE) is: % 95.0
```

5. Summary

In my model i applied the selction of the input features and classes for prediciton as sepal length and sepal width and for lables as virginica and setosa. I looked at the distirbution of the data and it look like linearly seperetable but two points are close to each other so it could effect the training or test accuracy it went to the test set and thats why it couldnt seperated well thus it make the test accuracy as % 95.

I used normalization even though the data is not that big for better model result. For linear regression I applied sigmoid function and applied inside the logistic regression to classify the data. I plotted the cost through iterations to see if the model is working as it should be and plotted accuracy to see if the models performance is good.

For cost function and update the parameters I used log loss for LogisticRegression model and OldLogisticRegression model I used MSE and see their performance. They both performed well in this case but for performance log loss was better computationally as O(N). Also to compute the gradient of the log loss, I differentiated the log loss function with respect to each weight, focusing on how changes in each weight impact the prediction error. By transposing the feature matrix, I aligned it with the error term, enabling efficient calculation of the gradient for all weights at once, which allowed for simultaneous updates during each iteration.

for the log lossed logistic regression the cost weights bias and accuracy are:
final weights: [3.73976568 -1.44467846]
final bias: 0.33752877424124866
final accuracy: % 100.0
final cost: 0.04657999054823778

for the MSE logistic regression the cost weights bias and accuracy are:
Final weights: [2.07841092 -0.73141164]
Final bias: 0.07620664959869698
final accuracy: % 100.0
final cost: 0.011958838540524909

In [3]:
```
#import neccesseray libraries
import numpy as  np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from io import StringIO
```

In [4]:
```
data = '''
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
5.8,4.0,1.2,0.2,Iris-setosa
5.7,4.4,1.5,0.4,Iris-setosa
5.4,3.9,1.3,0.4,Iris-setosa
5.1,3.5,1.4,0.3,Iris-setosa
5.7,3.8,1.7,0.3,Iris-setosa
5.1,3.8,1.5,0.3,Iris-setosa
5.4,3.4,1.7,0.2,Iris-setosa
5.1,3.7,1.5,0.4,Iris-setosa
4.6,3.6,1.0,0.2,Iris-setosa
5.1,3.3,1.7,0.5,Iris-setosa
4.8,3.4,1.9,0.2,Iris-setosa
5.0,3.0,1.6,0.2,Iris-setosa
5.0,3.4,1.6,0.4,Iris-setosa
5.2,3.5,1.5,0.2,Iris-setosa
5.2,3.4,1.4,0.2,Iris-setosa
4.7,3.2,1.6,0.2,Iris-setosa
4.8,3.1,1.6,0.2,Iris-setosa
5.4,3.4,1.5,0.4,Iris-setosa
5.2,4.1,1.5,0.1,Iris-setosa
5.5,4.2,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.2,Iris-setosa
5.0,3.2,1.2,0.2,Iris-setosa
5.5,3.5,1.3,0.2,Iris-setosa
4.9,3.6,1.4,0.1,Iris-setosa
4.4,3.0,1.3,0.2,Iris-setosa
5.1,3.4,1.5,0.2,Iris-setosa
5.0,3.5,1.3,0.3,Iris-setosa
4.5,2.3,1.3,0.3,Iris-setosa
4.4,3.2,1.3,0.2,Iris-setosa
5.0,3.5,1.6,0.6,Iris-setosa
5.1,3.8,1.9,0.4,Iris-setosa
4.8,3.0,1.4,0.3,Iris-setosa
5.1,3.8,1.6,0.2,Iris-setosa
4.6,3.2,1.4,0.2,Iris-setosa
5.3,3.7,1.5,0.2,Iris-setosa
5.0,3.3,1.4,0.2,Iris-setosa
```

```
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
5.5,2.3,4.0,1.3,Iris-versicolor
6.5,2.8,4.6,1.5,Iris-versicolor
5.7,2.8,4.5,1.3,Iris-versicolor
6.3,3.3,4.7,1.6,Iris-versicolor
4.9,2.4,3.3,1.0,Iris-versicolor
6.6,2.9,4.6,1.3,Iris-versicolor
5.2,2.7,3.9,1.4,Iris-versicolor
5.0,2.0,3.5,1.0,Iris-versicolor
5.9,3.0,4.2,1.5,Iris-versicolor
6.0,2.2,4.0,1.0,Iris-versicolor
6.1,2.9,4.7,1.4,Iris-versicolor
5.6,2.9,3.6,1.3,Iris-versicolor
6.7,3.1,4.4,1.4,Iris-versicolor
5.6,3.0,4.5,1.5,Iris-versicolor
5.8,2.7,4.1,1.0,Iris-versicolor
6.2,2.2,4.5,1.5,Iris-versicolor
5.6,2.5,3.9,1.1,Iris-versicolor
5.9,3.2,4.8,1.8,Iris-versicolor
6.1,2.8,4.0,1.3,Iris-versicolor
6.3,2.5,4.9,1.5,Iris-versicolor
6.1,2.8,4.7,1.2,Iris-versicolor
6.4,2.9,4.3,1.3,Iris-versicolor
6.6,3.0,4.4,1.4,Iris-versicolor
6.8,2.8,4.8,1.4,Iris-versicolor
6.7,3.0,5.0,1.7,Iris-versicolor
6.0,2.9,4.5,1.5,Iris-versicolor
5.7,2.6,3.5,1.0,Iris-versicolor
5.5,2.4,3.8,1.1,Iris-versicolor
5.5,2.4,3.7,1.0,Iris-versicolor
5.8,2.7,3.9,1.2,Iris-versicolor
6.0,2.7,5.1,1.6,Iris-versicolor
5.4,3.0,4.5,1.5,Iris-versicolor
6.0,3.4,4.5,1.6,Iris-versicolor
6.7,3.1,4.7,1.5,Iris-versicolor
6.3,2.3,4.4,1.3,Iris-versicolor
5.6,3.0,4.1,1.3,Iris-versicolor
5.5,2.5,4.0,1.3,Iris-versicolor
5.5,2.6,4.4,1.2,Iris-versicolor
6.1,3.0,4.6,1.4,Iris-versicolor
5.8,2.6,4.0,1.2,Iris-versicolor
5.0,2.3,3.3,1.0,Iris-versicolor
5.6,2.7,4.2,1.3,Iris-versicolor
5.7,3.0,4.2,1.2,Iris-versicolor
5.7,2.9,4.2,1.3,Iris-versicolor
6.2,2.9,4.3,1.3,Iris-versicolor
5.1,2.5,3.0,1.1,Iris-versicolor
5.7,2.8,4.1,1.3,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
7.1,3.0,5.9,2.1,Iris-virginica
6.3,2.9,5.6,1.8,Iris-virginica
6.5,3.0,5.8,2.2,Iris-virginica
7.6,3.0,6.6,2.1,Iris-virginica
4.9,2.5,4.5,1.7,Iris-virginica
7.3,2.9,6.3,1.8,Iris-virginica
6.7,2.5,5.8,1.8,Iris-virginica
7.2,3.6,6.1,2.5,Iris-virginica
```

```
6.5,3.2,5.1,2.0,Iris-virginica
6.4,2.7,5.3,1.9,Iris-virginica
6.8,3.0,5.5,2.1,Iris-virginica
5.7,2.5,5.0,2.0,Iris-virginica
5.8,2.8,5.1,2.4,Iris-virginica
6.4,3.2,5.3,2.3,Iris-virginica
6.5,3.0,5.5,1.8,Iris-virginica
7.7,3.8,6.7,2.2,Iris-virginica
7.7,2.6,6.9,2.3,Iris-virginica
6.0,2.2,5.0,1.5,Iris-virginica
6.9,3.2,5.7,2.3,Iris-virginica
5.6,2.8,4.9,2.0,Iris-virginica
7.7,2.8,6.7,2.0,Iris-virginica
6.3,2.7,4.9,1.8,Iris-virginica
6.7,3.3,5.7,2.1,Iris-virginica
7.2,3.2,6.0,1.8,Iris-virginica
6.2,2.8,4.8,1.8,Iris-virginica
6.1,3.0,4.9,1.8,Iris-virginica
6.4,2.8,5.6,2.1,Iris-virginica
7.2,3.0,5.8,1.6,Iris-virginica
7.4,2.8,6.1,1.9,Iris-virginica
7.9,3.8,6.4,2.0,Iris-virginica
6.4,2.8,5.6,2.2,Iris-virginica
6.3,2.8,5.1,1.5,Iris-virginica
6.1,2.6,5.6,1.4,Iris-virginica
7.7,3.0,6.1,2.3,Iris-virginica
6.3,3.4,5.6,2.4,Iris-virginica
6.4,3.1,5.5,1.8,Iris-virginica
6.0,3.0,4.8,1.8,Iris-virginica
6.9,3.1,5.4,2.1,Iris-virginica
6.7,3.1,5.6,2.4,Iris-virginica
6.9,3.1,5.1,2.3,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
6.8,3.2,5.9,2.3,Iris-virginica
6.7,3.3,5.7,2.5,Iris-virginica
6.7,3.0,5.2,2.3,Iris-virginica
6.3,2.5,5.0,1.9,Iris-virginica
6.5,3.0,5.2,2.0,Iris-virginica
6.2,3.4,5.4,2.3,Iris-virginica
5.9,3.0,5.1,1.8,Iris-virginica

'''
```

CODE DATA SHOULD BE HERE AND CODE SHOULD BE SELF SUFFICIENT

In [5]:
```
cols = ["sepal length", "sepal width", "petal length", "petal width", "fl


df = pd.read_csv(StringIO(data), names=cols, skiprows=1)
#df = pd.read_csv("./data/X.csv", names=cols) #read from the csv file wit

df.head() #make sure its working with displayin first 5 rows
```

Out[5]:

	sepal length	sepal width	petal length	petal width	flower
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [6]:

```
# select the two input features and the label here


# from df select irist setosa and iris virginica
df = df[(df['flower'] == 'Iris-setosa') | (df['flower'] == 'Iris-virginic

# select two features as sepal length sepal width and label as flower
df = df[['sepal length', 'sepal width', 'flower']]

# for binary classification map to the flower iris setosa 0 iris virginic
df['flower'] = df['flower'].map({'Iris-setosa': 0, 'Iris-virginica': 1})

# making sure the modified df is working printed 60 rows becausw wanted t
df.head(60)
```

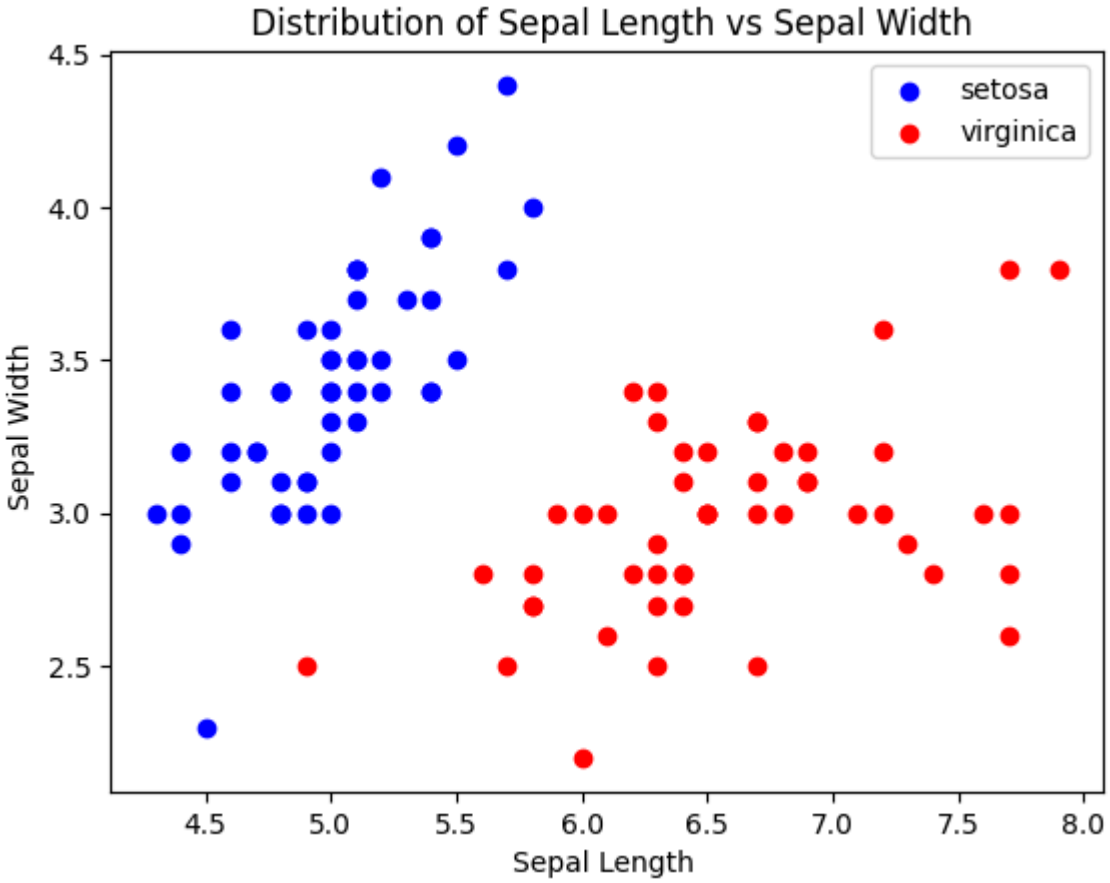
Out[6]:

	sepal length	sepal width	flower
0	5.1	3.5	0
1	4.9	3.0	0
2	4.7	3.2	0
3	4.6	3.1	0
4	5.0	3.6	0
5	5.4	3.9	0
6	4.6	3.4	0
7	5.0	3.4	0
8	4.4	2.9	0
9	4.9	3.1	0
10	5.4	3.7	0
11	4.8	3.4	0
12	4.8	3.0	0
13	4.3	3.0	0
14	5.8	4.0	0
15	5.7	4.4	0
16	5.4	3.9	0
17	5.1	3.5	0
18	5.7	3.8	0
19	5.1	3.8	0
20	5.4	3.4	0
21	5.1	3.7	0
22	4.6	3.6	0
23	5.1	3.3	0
24	4.8	3.4	0
25	5.0	3.0	0
26	5.0	3.4	0
27	5.2	3.5	0
28	5.2	3.4	0
29	4.7	3.2	0
30	4.8	3.1	0
31	5.4	3.4	0
32	5.2	4.1	0
33	5.5	4.2	0

	sepal length	sepal width	flower
34	4.9	3.1	0
35	5.0	3.2	0
36	5.5	3.5	0
37	4.9	3.6	0
38	4.4	3.0	0
39	5.1	3.4	0
40	5.0	3.5	0
41	4.5	2.3	0
42	4.4	3.2	0
43	5.0	3.5	0
44	5.1	3.8	0
45	4.8	3.0	0
46	5.1	3.8	0
47	4.6	3.2	0
48	5.3	3.7	0
49	5.0	3.3	0
100	6.3	3.3	1
101	5.8	2.7	1
102	7.1	3.0	1
103	6.3	2.9	1
104	6.5	3.0	1
105	7.6	3.0	1
106	4.9	2.5	1
107	7.3	2.9	1
108	6.7	2.5	1
109	7.2	3.6	1

In [7]:
```
#segregate the data and plot distribution

setosa = df[df['flower'] == 0]
virginica =df[df['flower'] == 1]


#plot the new data

plt.scatter(setosa['sepal length'], setosa['sepal width'], color='blue',
plt.scatter(virginica['sepal length'], virginica['sepal width'], color='r
plt.xlabel('Sepal Length ')
plt.ylabel('Sepal Width ')
```

```
plt.title(' Distribution of Sepal Length vs Sepal Width')
plt.legend()
plt.show()
```



features looks like it can be well classified i can see the decision boundary well so the features i selected are correct

In [8]:
```
#seperate the data to train and test as %80 and %20 respectively


#X1 = df['sepal length']
#X2 = df['sepal width']
X = df[['sepal length', 'sepal width']]
y =  df['flower']

# Split the data randomly in 80:20, that is train and test data both shou
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# make sure the distribution by printing for train and test as well
print("Training dataset:")
print("Num of setosa samples:", (y_train == 0).sum())
print("Num of virginica samples:", (y_train == 1).sum())
print("Total data in training set:", len(y_train))

print("Test Set:")
print("Num of setosa samples:", (y_test == 0).sum())
print("Num of virginica samples:", (y_test == 1).sum())
print("Total data in test set:", len(y_test))
```

```
Training dataset:
Num of setosa samples: 40
Num of virginica samples: 40
Total data in training set: 80
Test Set:
Num of setosa samples: 10
Num of virginica samples: 10
Total data in test set: 20
```

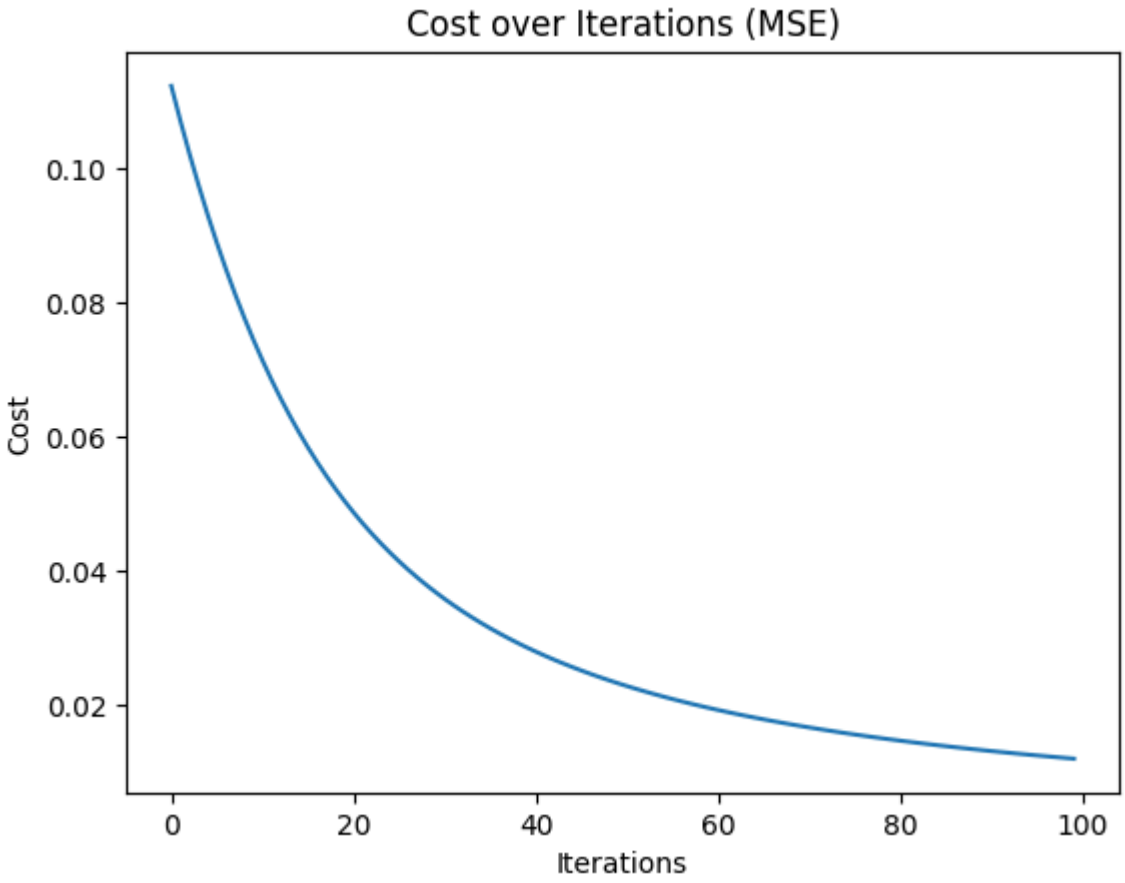
In [9]:
```python
#sigmoid function to use it for logistic regression
def sigmoid(z):

    g = 1/(1+np.exp(-z))

    return g
```

In [10]:
```python
def OldLogisticRegression(X, y, alpha, iterations, w, b):
    m, n = X.shape  # Get the number of samples and features

    # Initialize weights and bias
    weights = np.array(w)
    bias = b

    # Normalize the input features
    X_norm = (X - X.mean()) / X.std()

    cost_history = []  # Store the cost through iterations
    accuracy_history = []  # Store the accuracy through iterations

    for it in range(iterations):
        z = np.dot(X_norm, weights) + bias  # Apply the model correspondi
        y_pred = sigmoid(z)  # Apply sigmoid function to get the logistic


        # Calculate the mse and append to the cost history list
        cost = (1 / (2 * m)) * np.sum((y - y_pred) ** 2)
        cost_history.append(cost)


        # Initialize the gradients of weights (2 features) and the bias
        w_gradient = (1 / m) * np.dot(X_norm.T, (y_pred - y))  # Gradient
        b_gradient = (1 / m) * np.sum(y_pred - y)  # Gradient for bias

        # Apply the update rule here for weights and bias
        weights -= alpha * w_gradient
        bias -= alpha * b_gradient


        # Calculate the accuracy
        pred_class = [1 if i > 0.5 else 0 for i in y_pred]  # Apply the t
        accuracy = np.mean(pred_class == y) * 100  # Get it as percentage
        accuracy_history.append(accuracy)  # Append the prediction into a

    return weights, bias, cost_history, accuracy_history
```

In [15]:
```python
alpha = 0.1
iterations = 100
w = [0.5, 0.5]  # initial weights
b = 0  # initial bias
```

```
# train the model
weights_old, bias_old, cost_history_old, accuracy_history_old = OldLogist

# print final weights and bias
print("Final weights:", weights_old)
print("Final bias:", bias_old)
print("final accuracy: %",accuracy_history_old[-1])
print("final cost: ", cost_history_old[-1])

# plot the cost function over iterations

plt.plot(range(iterations), cost_history_old)
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.title("Cost over Iterations (MSE)")
plt.show()

# plot accuracy over iterations
plt.plot(range(iterations), accuracy_history_old)
plt.xlabel("Iterations")
plt.ylabel("Training Accuracy (%) ")
plt.title("Training Accuracy over Iterations (MSE)")
plt.show()
```

```
Final weights: [ 2.07841092 -0.73141164]
Final bias: 0.07620664959869698
final accuracy: % 100.0
final cost:  0.011958838540524909
```

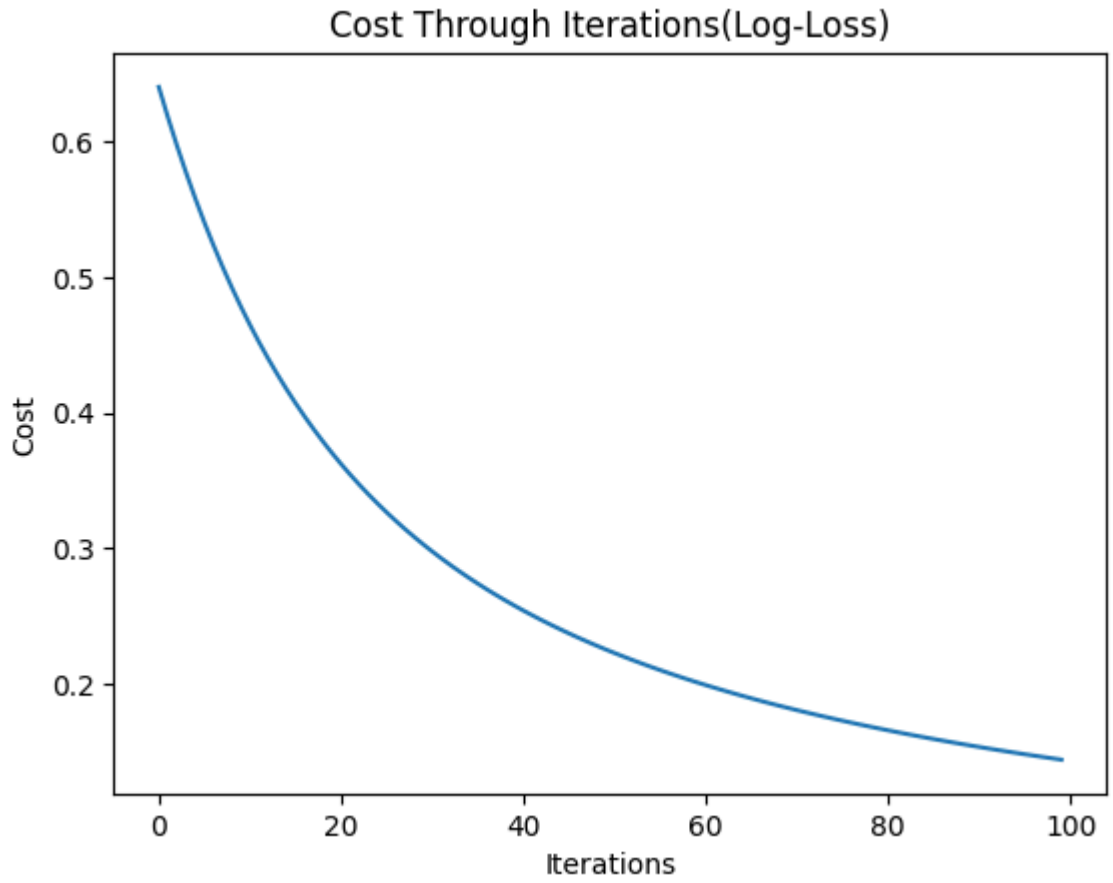


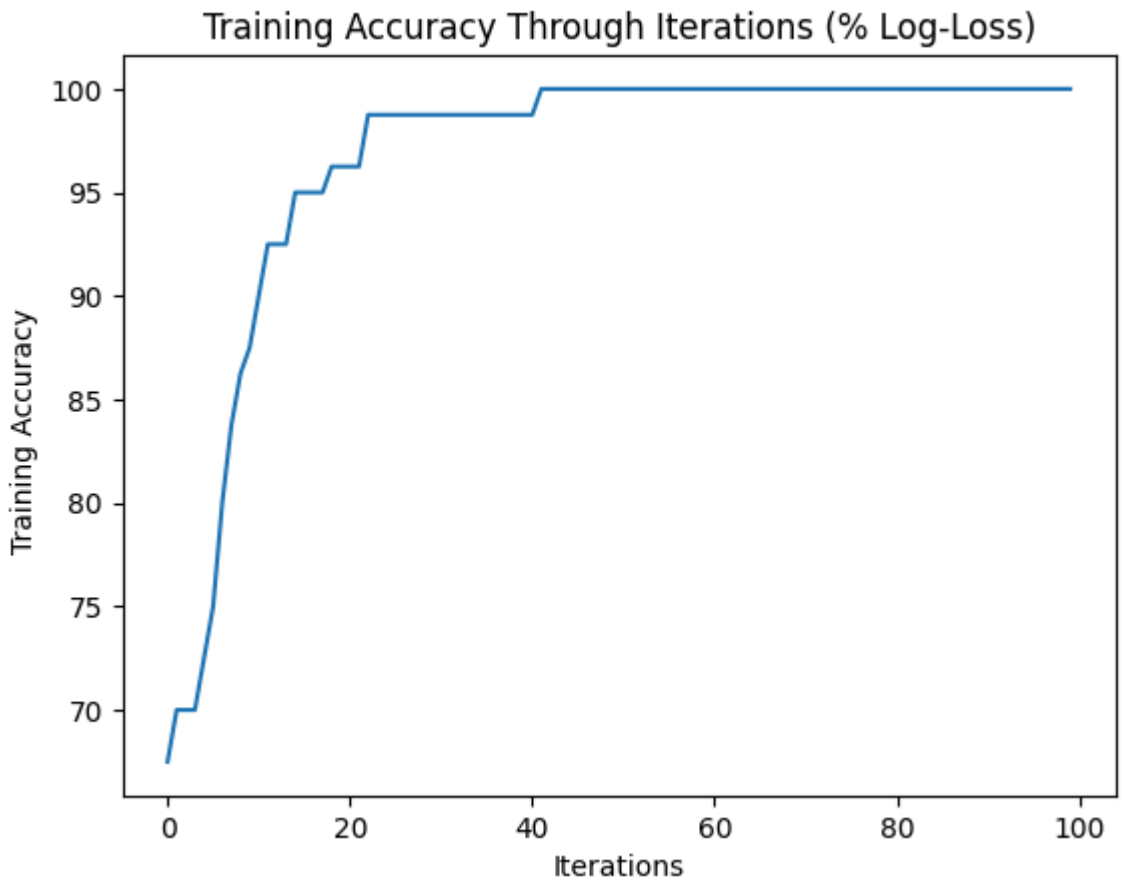
In [12]:
```python
def LogisticRegression(X, y, alpha, iteratioons, w, b):

    m,n = X.shape #get the shape of the

    #initialize the first weights and the bias here
    weights = np.array(w)
    bias = b

    X_norm = (X - X.mean()) /  X.std() #normalize the input features

    cost_history = [] # storet the cost to iterations
    accuracy_list = [] #for accuracy to store it

    for i in range(iteratioons):

        z = np.dot(X_norm, weights) + bias #model setup
        y_pred = sigmoid(z) ##putting the model into the sigmoid function

        #calculate cost and append to cost_history list

        cost = -(1 / m) * np.sum(y * np.log(y_pred) + (1 - y) * np.log(1
        cost_history.append(cost)


        #calculate gradients for weight and bias

        w_gradient = (1/m) * np.dot(X_norm.T, (y_pred - y)) #calculate th
        #corresponds to the error*X and turn to transpose for dimension e
        b_gradient = (1/m) * np.sum(y_pred - y) #calculate the bias gradi

        #update the weights at the same time simulteanosly
```

```
        weights -= alpha * w_gradient
        bias -= alpha * b_gradient

        #calculate accuracy on train data

        pred_class = [1 if i> 0.5 else 0 for i in y_pred]

        accuracy = np.mean(pred_class ==y) * 100 #multpily by 100 for per
        accuracy_list.append(accuracy) #add to the list for accuracy

    return weights, bias, cost_history , accuracy_list
```

In [16]:
```
#DRIVER CODE FOR LOGISTIC REGRESSION

#alpha, iterations, initial weight and bias setup
alpha = 0.1
iterations = 100
w = [0.5, 0.5]
b = 0

#train the model
weights, bias, cost_history, accuracy_history = LogisticRegression(X_trai

#final weight and biases cost and accuracy print to see
print("final weights: ", weights)
print("final bias: ", bias)
print("final accuracy: %",accuracy_history[-1])
print("final cost: ", cost_history[-1])

#plot the cost function through iterations

plt.plot(range(iterations), cost_history)
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.title("Cost Through Iterations(Log-Loss)")
plt.show()

#plot the accuracy through iterations

plt.plot(range(iterations), accuracy_history)
plt.xlabel("Iterations")
plt.ylabel("Training Accuracy ")
plt.title("Training Accuracy Through Iterations (% Log-Loss)")
plt.show()
```

```
final weights:  [ 2.07841092 -0.73141164]
final bias:  0.07620664959869698
final accuracy: % 100.0
final cost:  0.14395353918143203
```





Change the features and try to get to the 100 accuracy

In [17]:
```
X_norm_test = (X_test - X_train.mean()) / X_train.std() #normalize the te

z_test = np.dot(X_norm_test,  weights) + bias # model for the test data
```

```
y_test_pred = sigmoid(z_test) # get it into sigmoid function for logistic

y_pred_class_test = [1 if pred > 0.5 else 0 for pred in y_test_pred] # ap

test_accuracy = np.mean(y_pred_class_test == y_test) * 100 #get the proba

print("test accuracy (Log-Loss) is: %", test_accuracy)


#testing the old logistic rregression fucntion as well


X_norm_test_old = (X_test - X_train.mean()) / X_train.std()  #normalize t
z_test_old = np.dot(X_norm_test_old, weights_old) + bias_old # model for

y_test_pred_old = sigmoid(z_test_old)  # get it into sigmoid function for

y_pred_class_test_old = [1 if pred > 0.5 else 0 for pred in y_test_pred_o

test_accuracy_old = np.mean(y_pred_class_test_old == y_test) * 100  #get

print("test accuracy using OldLogisticRegression (MSE) is: %", test_accur
```

```
test accuracy (Log-Loss) is: % 95.0
test accuracy using OldLogisticRegression (MSE) is: % 95.0
```

Summary

In my model i applied the selction of the input features and classes for prediciton as sepal length and sepal width and for lables as virginica and setosa. I looked at the distirbution of the data and it look like linearly seperetable but two points are close to each other so it could effect the training or test accuracy it went to the test set and thats why it couldnt seperated well thus it make the test accuracy as % 95.

I used normalization even though the data is not that big for better model result. For linear regression I applied sigmoid function and applied inside the logistic regression to classify the data. I plotted the cost through iterations to see if the model is working as it should be and plotted accuracy to see if the models performance is good.

For cost function and update the parameters I used log loss for LogisticRegression model and OldLogisticRegression model I used MSE and see their performance. They both performed well in this case but for performance log loss was better computationally as O(N). Also to compute the gradient of the log loss, I differentiated the log loss function with respect to each weight, focusing on how changes in each weight impact the prediction error. By transposing the feature matrix, I aligned it with the error term, enabling efficient calculation of the gradient for all weights at once, which allowed for simultaneous updates during each iteration.

for the log lossed logistic regression the cost weights bias and accuracy are: final weights: [3.73976568 -1.44467846] final bias: 0.33752877424124866 final accuracy: % 100.0 final cost: 0.04657999054823778

for the MSE logistic regression the cost weights bias and accuracy are: Final weights: [2.07841092 -0.73141164] Final bias: 0.07620664959869698 final accuracy: % 100.0 final cost: 0.011958838540524909