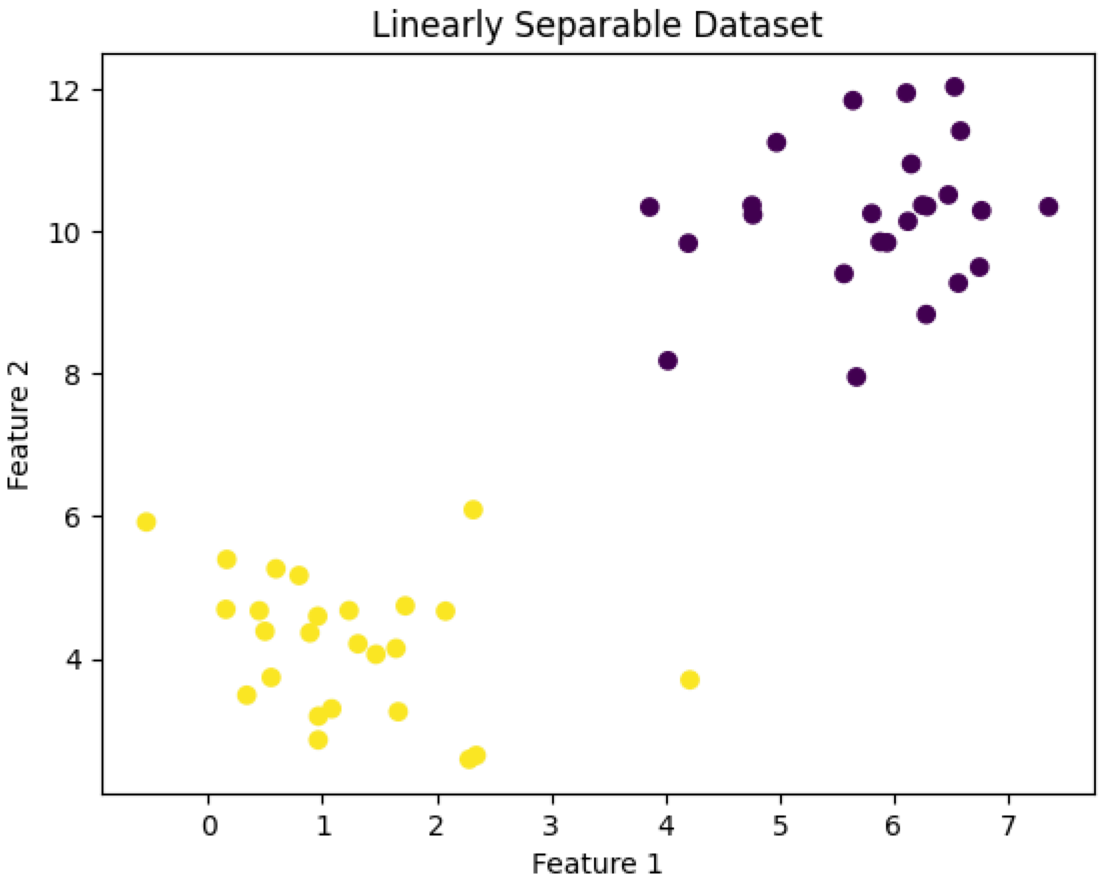
Problem 1

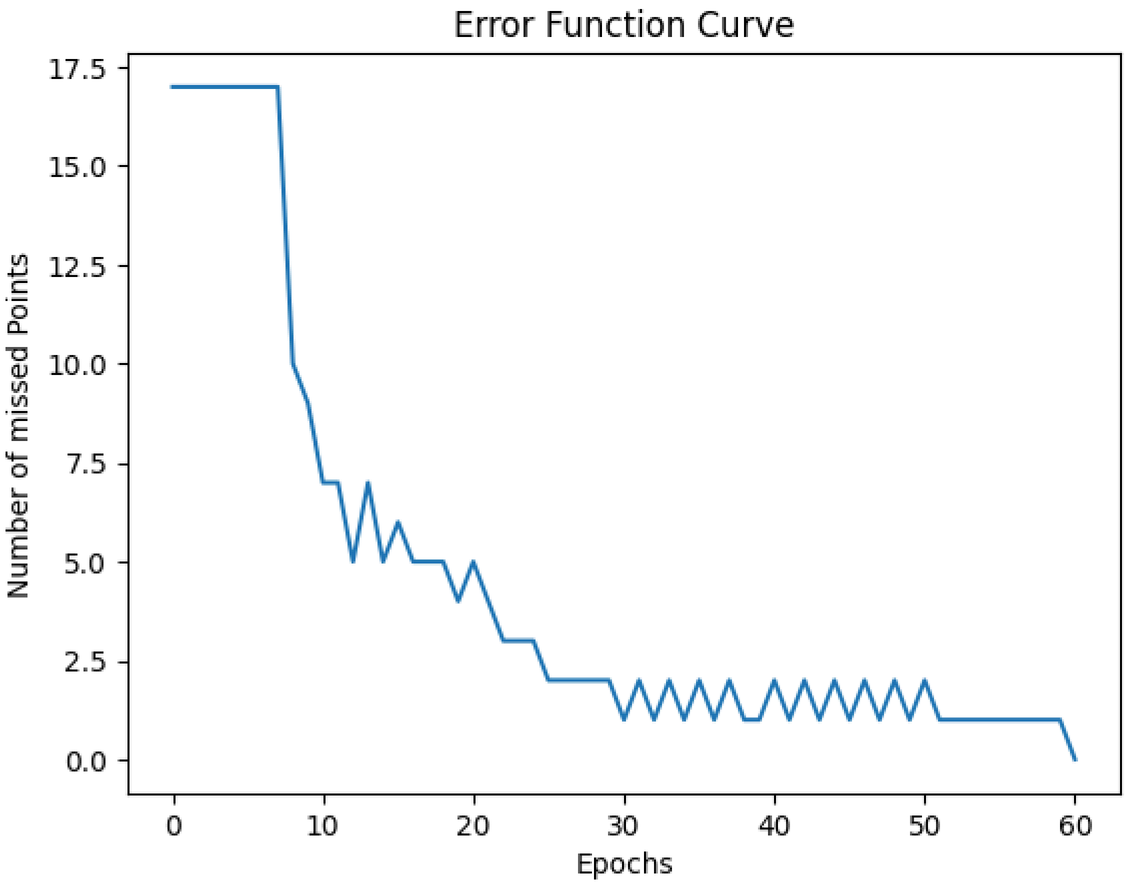
(1)

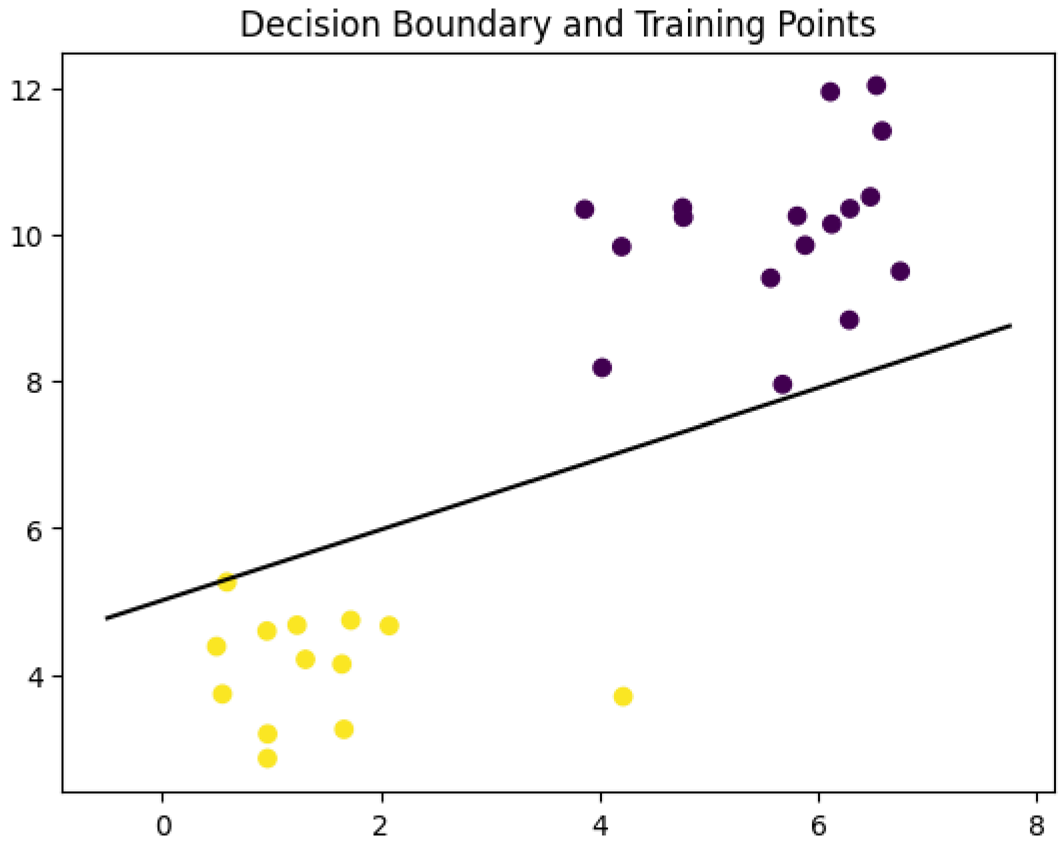
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split

np.random.seed(42)

x, y = datasets.make_blobs(n_samples = 50, centers = 2, n_features = 2,
                           center_box = (0, 10), cluster_std = 1, random_
y = np.where(y == 0, -1, 1)

plt.scatter(x[:, 0], x[:, 1], c = y)
plt.title('Linearly Separable Dataset')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.4
```



(2)

In [2]:
```
def batch_perceptron(x_train, y_train, learning_rate=0.001,
                     epochs=100, batch_size=x_train.shape[0]):
    w = np.array([1, 1])
    b = 1

    errors = []
    b_history = []
    w_history = []

    for _ in range(epochs):
        missed = []
        num_error = 0

        for i in range(batch_size):
            if y_train[i] * (np.dot(x_train[i], w) + b) <= 0:
                missed.append((x_train[i], y_train[i]))
                num_error += 1

        for x_i, y_i in missed:
            w = w + learning_rate * y_i * x_i
            b = b + learning_rate * y_i

        errors.append(num_error)
        w_history.append(w.copy())
        b_history.append(b)

        if num_error == 0:
            break

    return w, b, errors, w_history, b_history

w, b, errors, w_history, b_history = batch_perceptron(x_train, y_train)

plt.figure("Figure 1")
plt.plot(errors)
plt.xlabel('Epochs')
plt.ylabel('Number of missed Points')
plt.title('Error Function Curve')
plt.show()

x_min, x_max = x_train[:, 0].min() - 1, x_train[:, 0].max() + 1
y_min_line = -(b + w[0] * x_min) / w[1]
y_max_line = -(b + w[0] * x_max) / w[1]

plt.figure("Figure 2")
plt.plot([x_min, x_max], [y_min_line, y_max_line], 'k-', label='Decision
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
plt.title('Decision Boundary and Training Points')
plt.show()
```

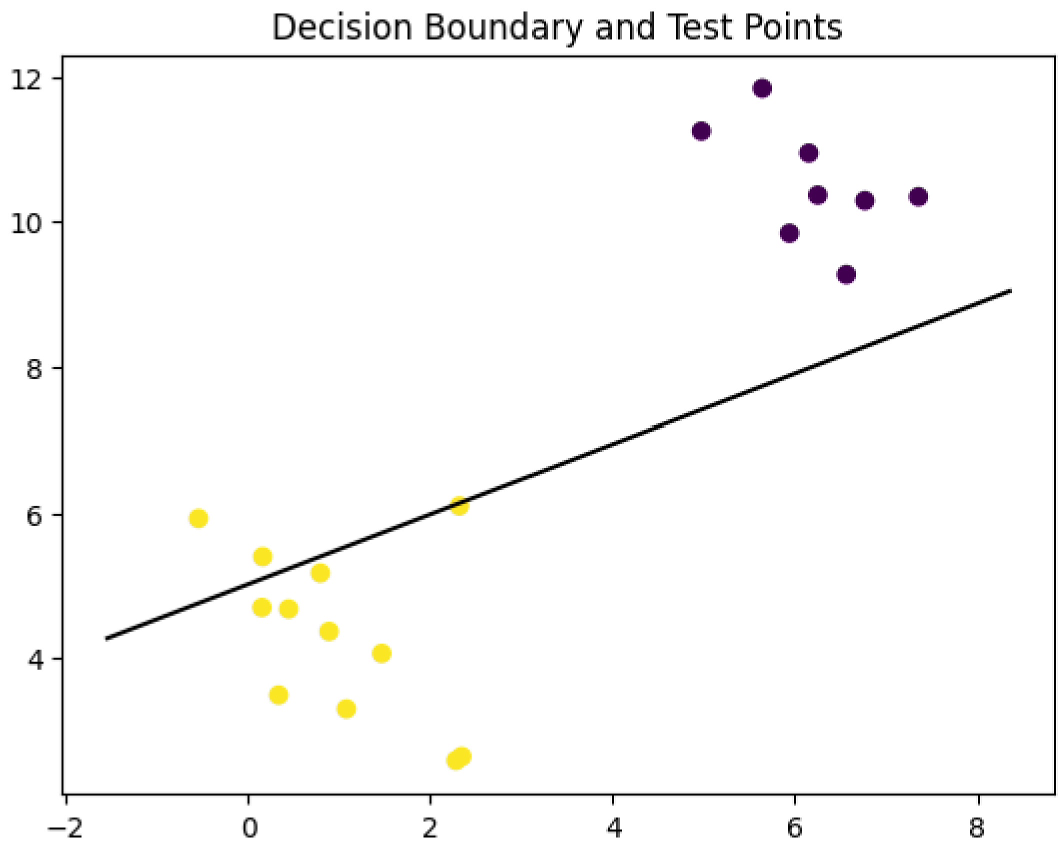

Error Function Curve
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Number of missed Points
0
10
20
30
40
50
60
Epochs


Decision Boundary and Training Points
12
10
8
6
4
0
2
4
6
8

(3)

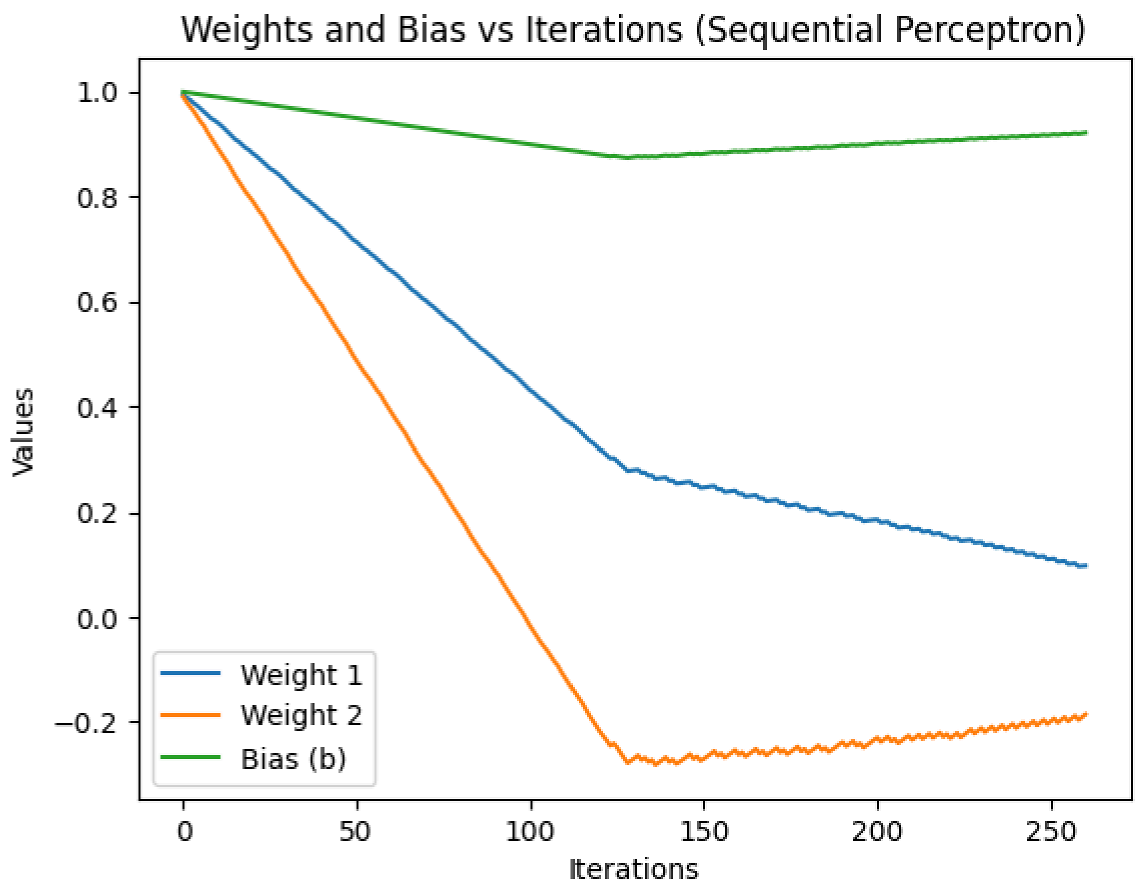
In [3]:
```
def run_perceptron(x, w, b):
    return np.where(np.dot(x, w) + b > 0, 1, -1)

y_pred = run_perceptron(x_test, w, b)

accuracy = np.mean(y_pred == y_test)
error_rate = 1 - accuracy

print(f"Batch Perceptron Accuracy: {accuracy * 100:.2f}%")
print(f"Batch Perceptron Error rate: {error_rate * 100:.2f}%")

x_min, x_max = x_test[:, 0].min() - 1, x_test[:, 0].max() + 1
y_min_line = -(b + w[0] * x_min) / w[1]
y_max_line = -(b + w[0] * x_max) / w[1]

plt.plot([x_min, x_max], [y_min_line, y_max_line], 'k-', label='Decision
plt.scatter(x_test[:, 0], x_test[:, 1], c = y_test)
plt.title('Decision Boundary and Test Points')
plt.show()
```

```
Batch Perceptron Accuracy: 90.00%
Batch Perceptron Error rate: 10.00%
```

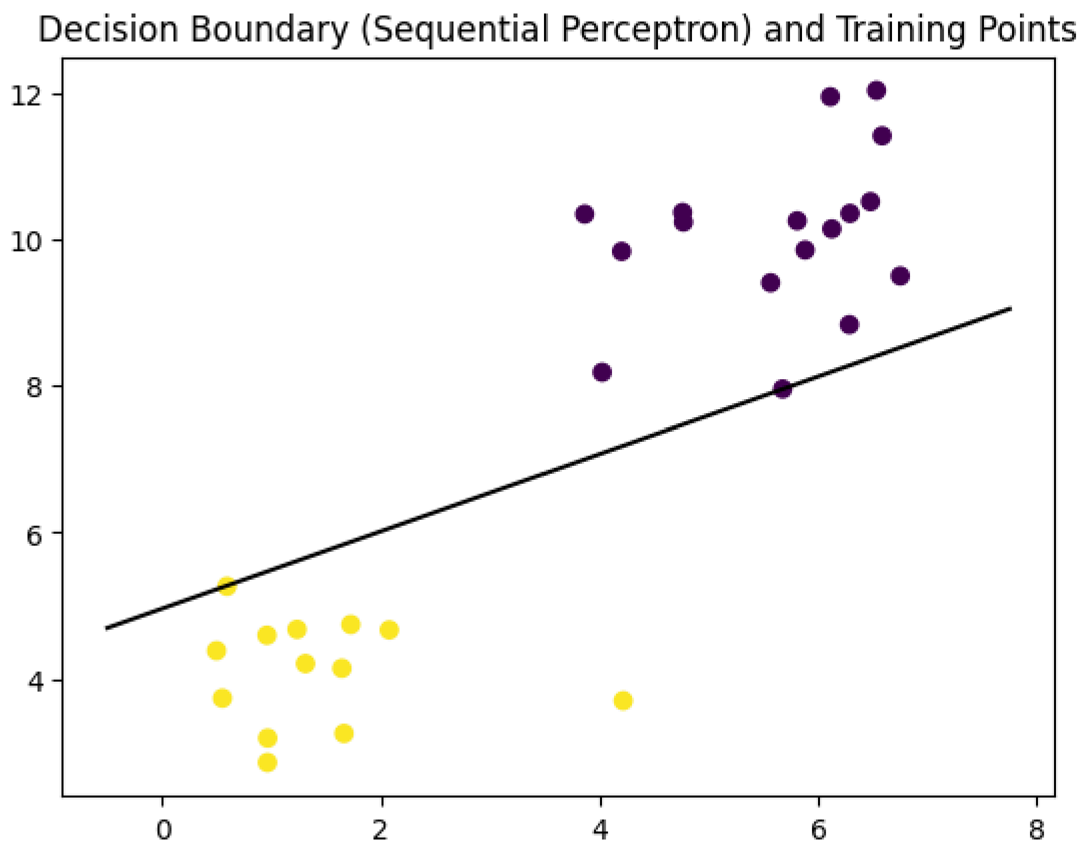


(4)

In [4]:
```
def sequential_perceptron(x_train, y_train, learning_rate=0.001, epochs=1
    w = np.array([1, 1])
    b = 1
    weights = []

    for _ in range(epochs):
        for i in range(x_train.shape[0]):
            if y_train[i] * (np.dot(x_train[i], w) + b) <= 0:
                w = w + learning_rate * y_train[i] * x_train[i]
                b = b + learning_rate * y_train[i]
                weights.append((w.copy(), b))

        predictions = np.sign(np.dot(x_train, w) + b)
        if np.all(predictions == y_train):
            break

    return w, b, weights

w, b, w_h = sequential_perceptron(x_train, y_train)

weights_array = np.array([np.append(w, b) for w, b in w_h])

plt.figure("Figure 1")
plt.plot(weights_array[:, 0], label="Weight 1")
plt.plot(weights_array[:, 1], label="Weight 2")
plt.plot(weights_array[:, 2], label="Bias (b)")
plt.xlabel('Iterations')
plt.ylabel('Values')
plt.title('Weights and Bias vs Iterations (Sequential Perceptron)')
plt.legend()
plt.show()

x_min, x_max = x_train[:, 0].min() - 1, x_train[:, 0].max() + 1
y_min_line_seq = -(b + w[0] * x_min) / w[1]
y_max_line_seq = -(b + w[0] * x_max) / w[1]

plt.figure("Figure 2")
plt.plot([x_min, x_max], [y_min_line_seq, y_max_line_seq],
         'k-', label='Decision Boundary')
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
plt.title('Decision Boundary (Sequential Perceptron) and Training Points'
plt.show()
```



Weights and Bias vs Iterations (Sequential Perceptron)
Values
Iterations
Weight 1
Weight 2
Bias (b)
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
50
100
150
200
250

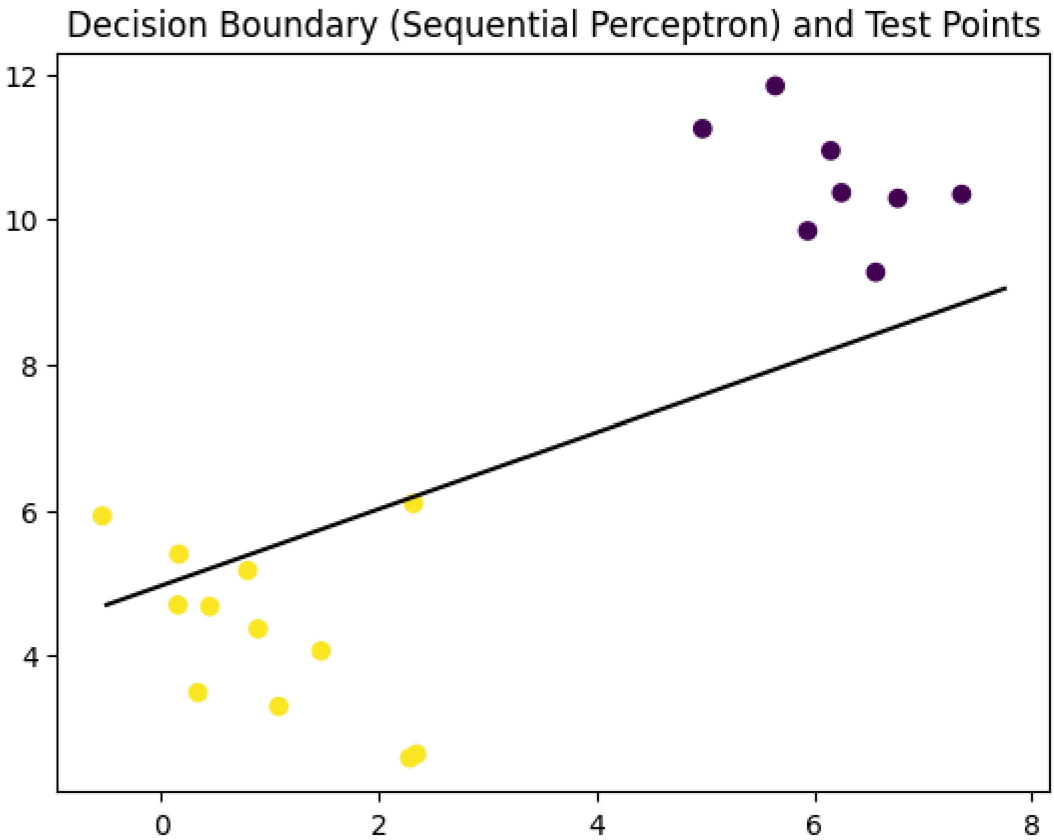


Decision Boundary (Sequential Perceptron) and Training Points
12
10
8
6
4
0
2
4
6
8

(5)

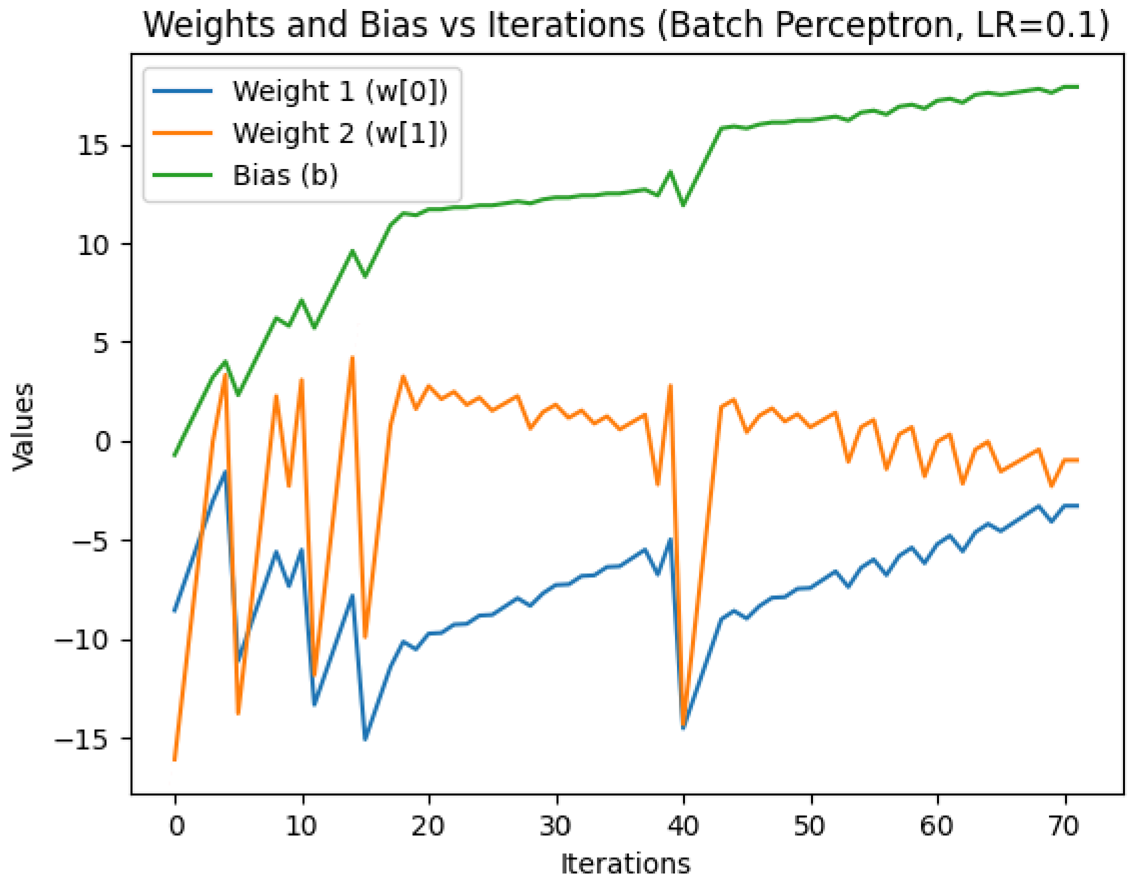
In [5]:
```
y_pred_seq = run_perceptron(x_test, w, b)

accuracy_seq = np.mean(y_pred_seq == y_test)
error_rate_seq = 1 - accuracy_seq

print(f"Sequential Perceptron Accuracy: {accuracy_seq * 100:.2f}%")
print(f"Sequential Perceptron Error Rate: {error_rate_seq * 100:.2f}%")

x_min, x_max = x_train[:, 0].min() - 1, x_train[:, 0].max() + 1
y_min_line = -(b + w[0] * x_min) / w[1]
y_max_line = -(b + w[0] * x_max) / w[1]

plt.plot([x_min, x_max], [y_min_line, y_max_line], 'k-', label='Decision
plt.scatter(x_test[:, 0], x_test[:, 1], c = y_test)
plt.title('Decision Boundary (Sequential Perceptron) and Test Points')
plt.show()
```

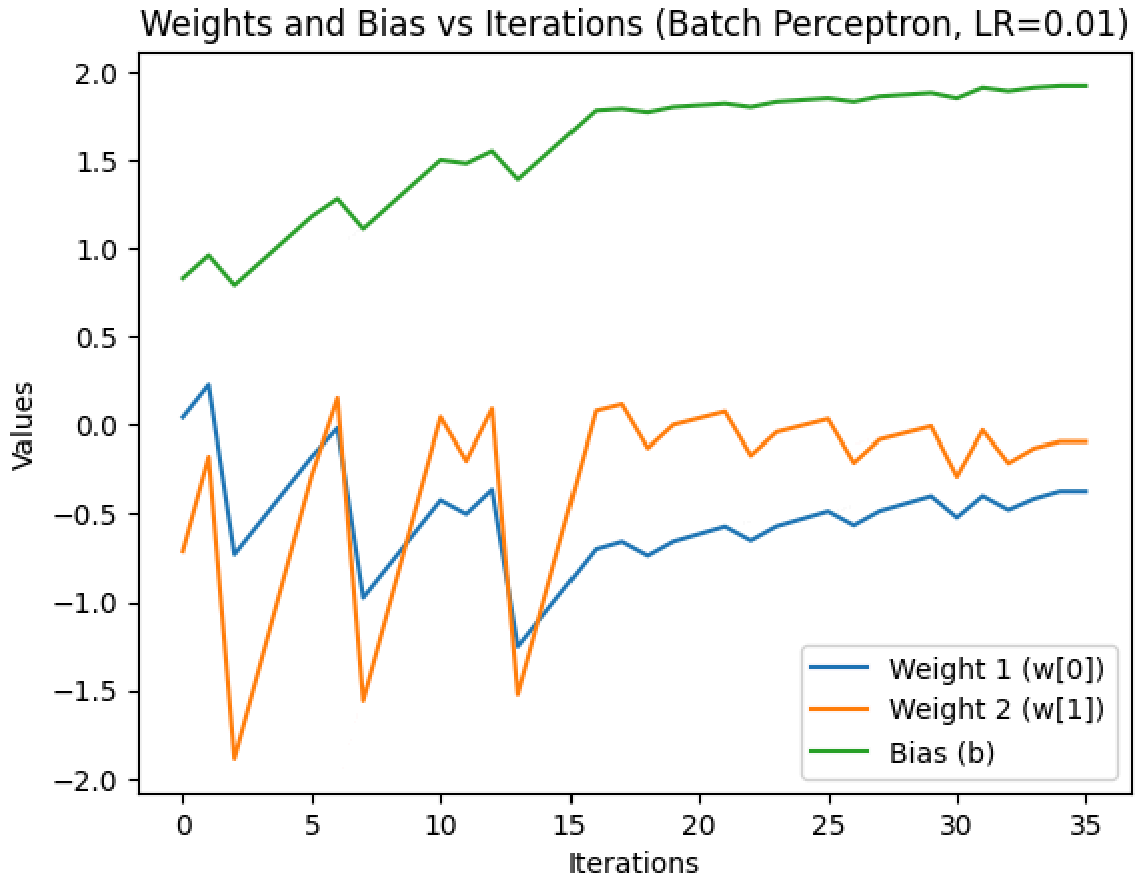
```
Sequential Perceptron Accuracy: 90.00%
Sequential Perceptron Error Rate: 10.00%
```

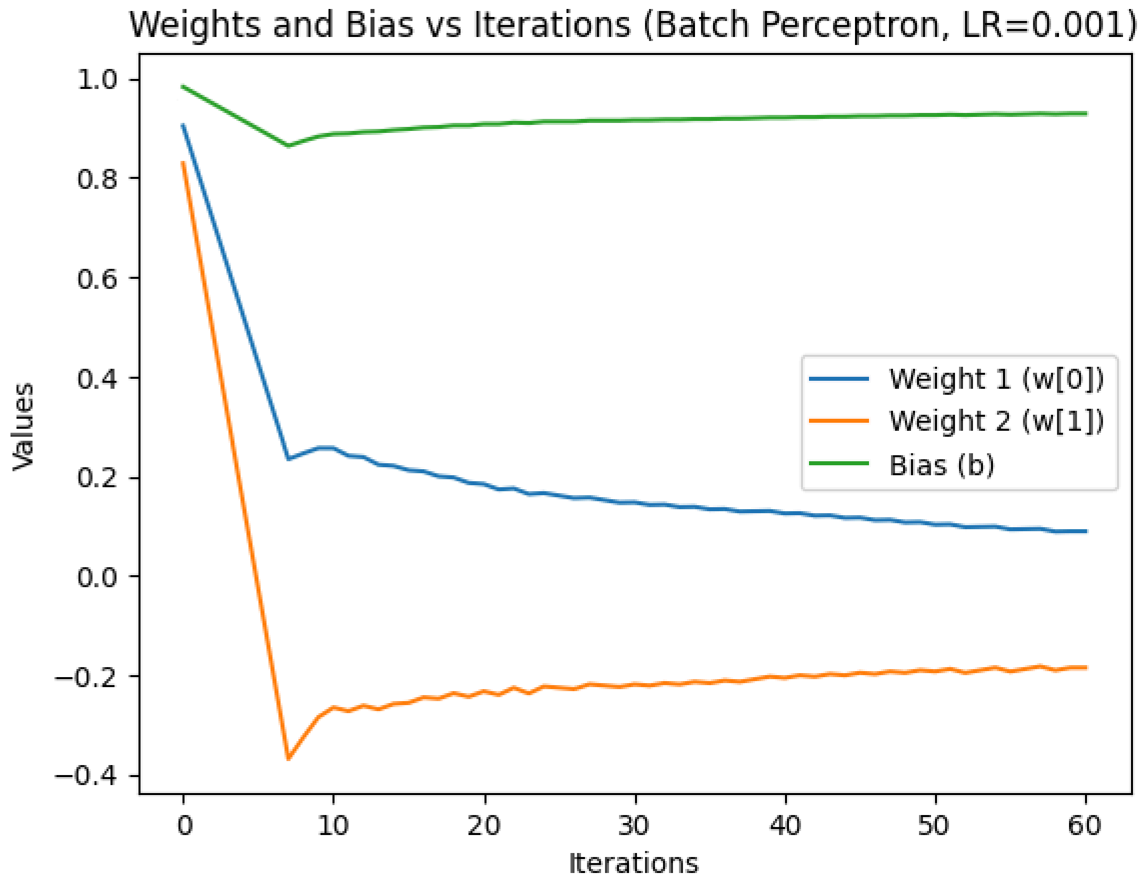


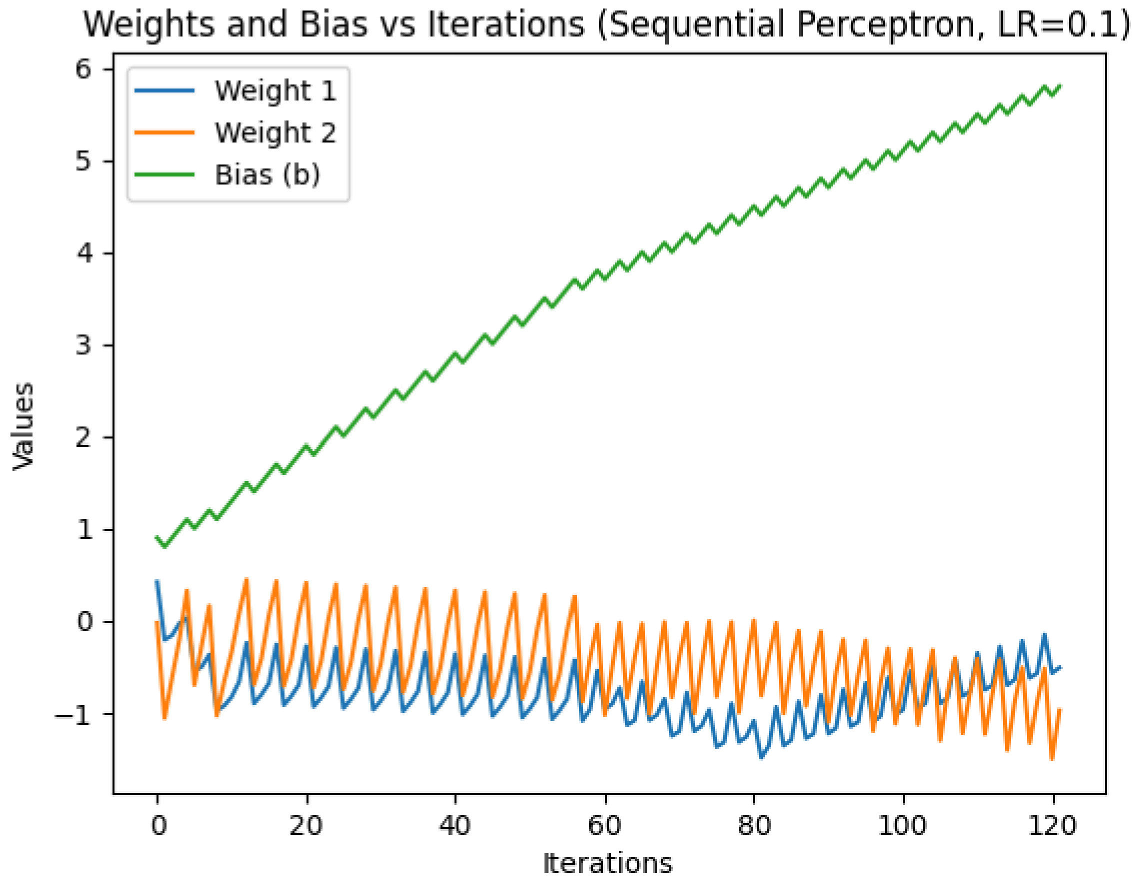
(6):

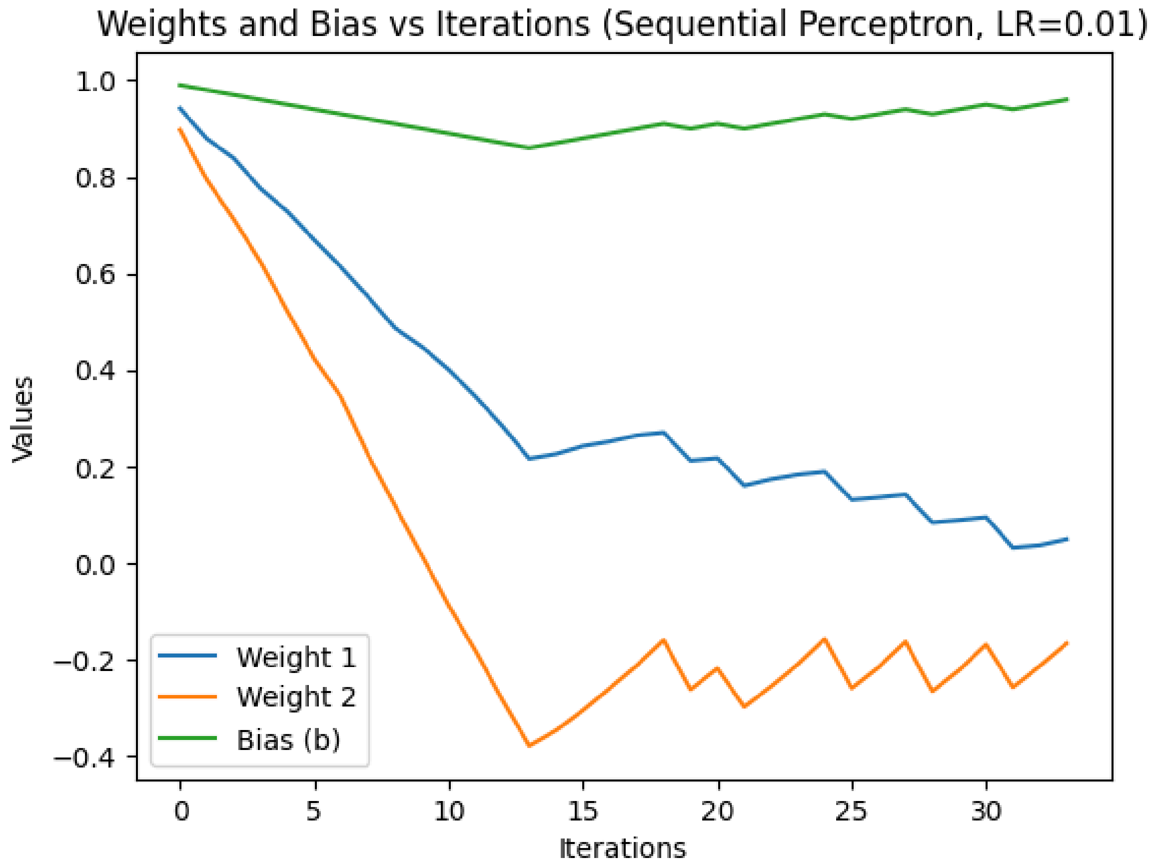
```
In [6]: learning_rates = [0.1, 0.01, 0.001]
methods = ['Batch', 'Sequential']


for method in methods:
    for lr in learning_rates:
        if method == 'Sequential':
            w, b, w_h = sequential_perceptron(x_train, y_train, learning_

            weights_array = np.array([np.append(w, b) for w, b in w_h])

            plt.plot(weights_array[:, 0], label="Weight 1")
            plt.plot(weights_array[:, 1], label="Weight 2")
            plt.plot(weights_array[:, 2], label="Bias (b)")
            plt.xlabel('Iterations')
            plt.ylabel('Values')
            plt.title(f'Weights and Bias vs Iterations ({method} Perceptr
            plt.legend()
            plt.show()

        else:
            w, b, errors, w_history, b_history = batch_perceptron(x_train
                                                                  y_train

            w_history = np.array(w_history)
            b_history = np.array(b_history)


            plt.plot(w_history[:, 0], label='Weight 1 (w[0])')
            plt.plot(w_history[:, 1], label='Weight 2 (w[1])')
            plt.plot(b_history, label='Bias (b)')

            plt.xlabel('Iterations')
            plt.ylabel('Values')
            plt.title(f'Weights and Bias vs Iterations ({method} Perceptr
            plt.legend()
            plt.show()
```

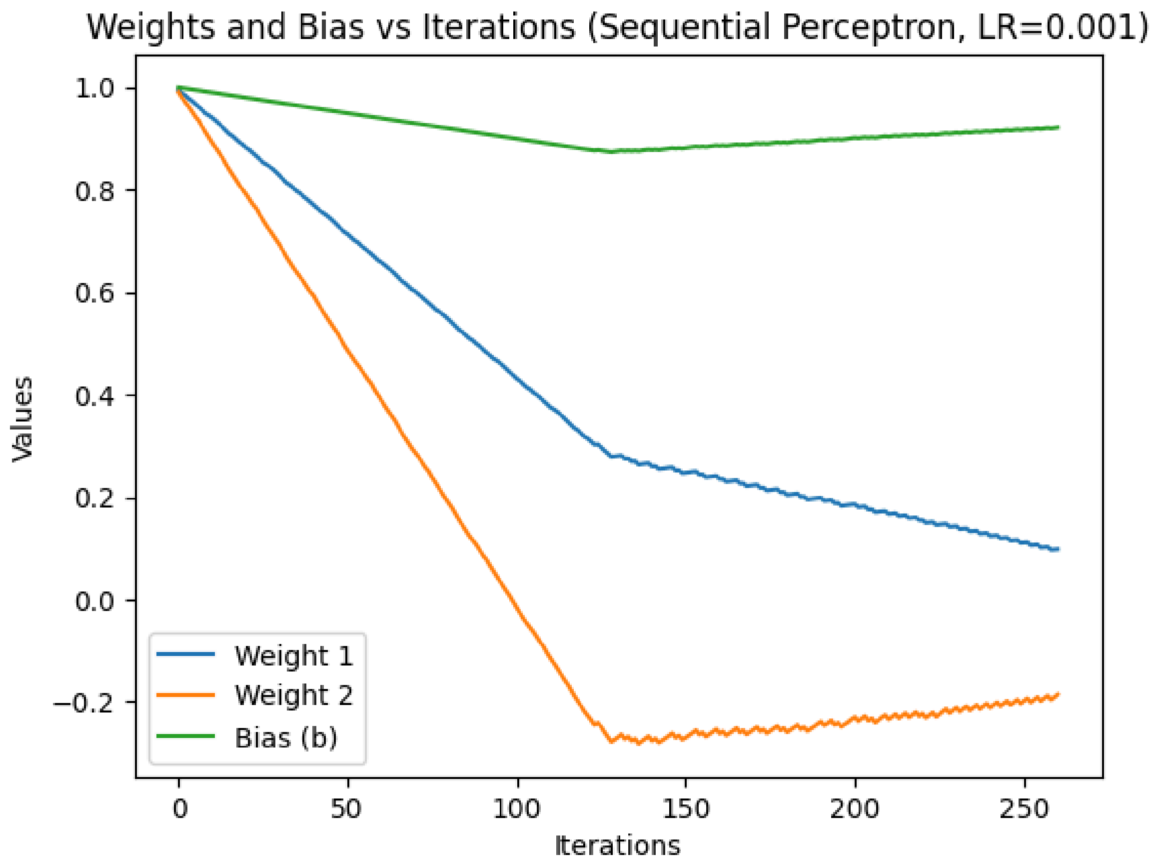

Weights and Bias vs Iterations (Batch Perceptron, LR=0.1)
Weight 1 (w[0])
Weight 2 (w[1])
Bias (b)
Values
Iterations


Weights and Bias vs Iterations (Batch Perceptron, LR=0.01)
Weight 1 (w[0])
Weight 2 (w[1])
Bias (b)
Values
Iterations


Weights and Bias vs Iterations (Batch Perceptron, LR=0.001)
Weight 1 (w[0])
Weight 2 (w[1])
Bias (b)
Values
Iterations
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0
10
20
30
40
50
60


Weights and Bias vs Iterations (Sequential Perceptron, LR=0.1)
Weight 1
Weight 2
Bias (b)
Values
Iterations
6
5
4
3
2
1
0
−1
0
20
40
60
80
100
120



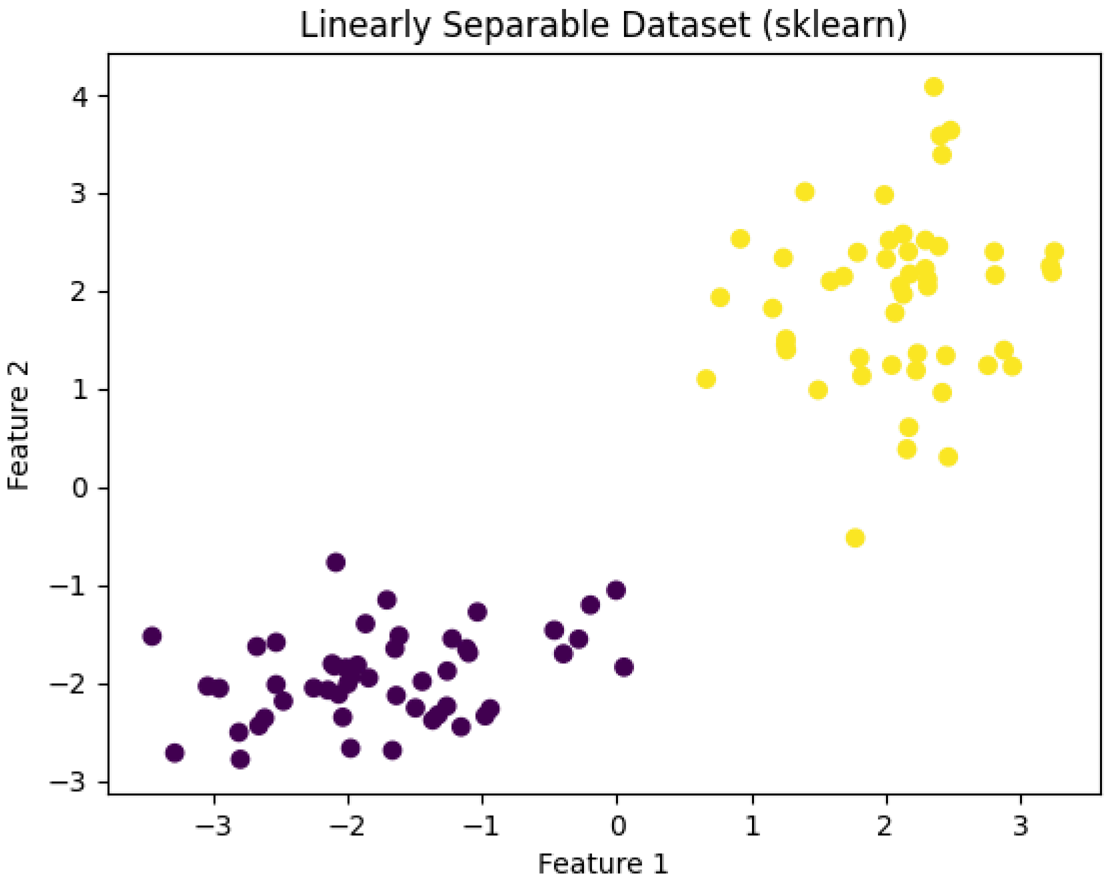


For both model I choosed my learning rate to be 0.001. From the plots provided above, it is clear to that if choosing learning rate to be 0.1 or 0.01, the weight vectors are hard to converge because the curve shows the zigzag pattern, which is caused by learning rate being to large. The 0.001 learning rate set has the proper converge behavior. Although the converging speed might be slower, it;s important to maintain stability and make sure that weight vector converge.

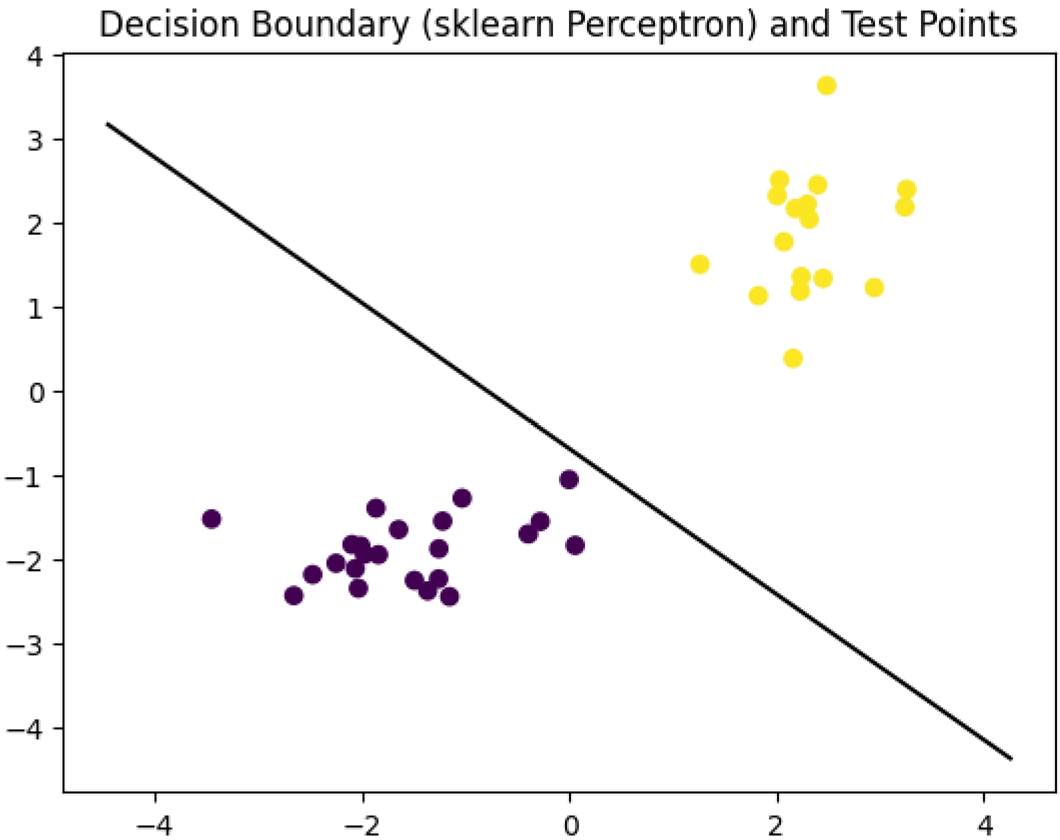
Problem 2

Part a:

In [7]:
```
from sklearn.metrics import (classification_report,
                             confusion_matrix,
                             precision_score,
                             recall_score,
                             f1_score)
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score
from sklearn.linear_model import Perceptron
import numpy as np
import matplotlib.pyplot as plt


# part a

x, y = make_classification(n_samples = 100, n_features = 2,
                           n_informative = 2, n_redundant = 0,
                           n_clusters_per_class = 1, class_sep = 2, rando

plt.scatter(x[:, 0], x[:, 1], c = y)
plt.title('Linearly Separable Dataset (sklearn)')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.4

model = Perceptron()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy {accuracy * 100:.2f}%")

w = model.coef_[0]
b = model.intercept_[0]

x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
y_min_line = -(b + w[0] * x_min) / w[1]
y_max_line = -(b + w[0] * x_max) / w[1]

plt.plot([x_min, x_max], [y_min_line, y_max_line], 'k-', label='Decision
plt.scatter(x_test[:, 0], x_test[:, 1], c = y_test)
plt.title('Decision Boundary (sklearn Perceptron) and Test Points')
plt.show()
```

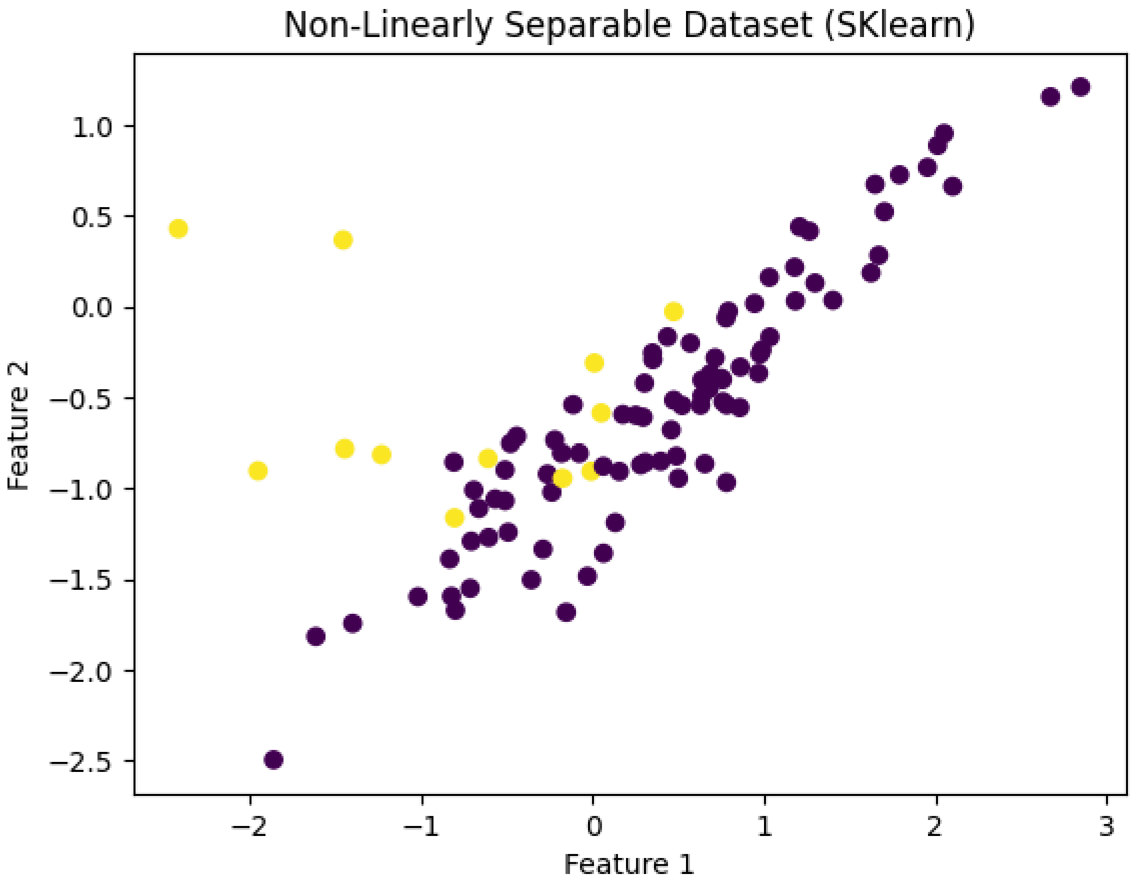



```
Accuracy 100.00%
```



Part b:

In [8]:
```
x, y = make_classification(n_samples = 100, n_features = 2,
                           n_informative = 2, n_redundant = 0,
                           n_clusters_per_class = 1, class_sep = 0.5,
                           flip_y = 0.1, weights = [0.9, 0.1])

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.4

plt.scatter(x[:, 0], x[:, 1], c = y)
plt.title('Non-Linearly Separable Dataset (SKlearn)')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()


model = Perceptron()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')


print("Classification Report:\n")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:\n")
print(confusion_matrix(y_test, y_pred))

w = model.coef_[0]
b = model.intercept_[0]

x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
y_min_line = -(b + w[0] * x_min) / w[1]
y_max_line = -(b + w[0] * x_max) / w[1]

plt.plot([x_min, x_max], [y_min_line, y_max_line], 'k-', label='Decision
plt.scatter(x_test[:, 0], x_test[:, 1], c = y_test)
plt.title('Decision Boundary (sklearn Perceptron) and Test Points')
plt.show()
```




```
Accuracy: 0.88
Precision: 0.50
Recall: 0.40
F1 Score: 0.44
Classification Report:

              precision    recall  f1-score   support

           0       0.92      0.94      0.93        35
           1       0.50      0.40      0.44         5

    accuracy                           0.88        40
   macro avg       0.71      0.67      0.69        40
weighted avg       0.86      0.88      0.87        40

Confusion Matrix:

[[33  2]
 [ 3  2]]
```