**ML EXPERIMENT 6**

import numpy as np

import matplotlib.pyplot as plt

# Define the sigmoid function

def sigmoid(z):

    return 1 / (1 + np.exp(-z))

# Define the logistic regression model

def logistic\_regression(X, w):

    z = np.dot(X, w)

    return sigmoid(z)

# Define the loss function (cross-entropy loss)

def loss(X, y, w):

    y\_pred = logistic\_regression(X, w)

    return -np.mean(y\*np.log(y\_pred) + (1-y)\*np.log(1-y\_pred))

# Define the gradient of the loss function with respect to the weights

def gradient(X, y, w):

    y\_pred = logistic\_regression(X, w)

    return np.dot(X.T, (y\_pred - y)) / len(y)

# Load the data

X = np.array([[1, 2], [3, 4], [5, 6]])

y = np.array([0, 1, 1])

# Initialize the weights

w = np.random.rand(2)

# Train the model

for i in range(1000):

    w -= 0.1 \* gradient(X, y, w)

# Make predictions

y\_pred = logistic\_regression(X, w)

# Print the predicted values

print(y\_pred)

# Plot the data and the decision boundary

x1 = np.linspace(0, 6, 100)

x2 = np.linspace(0, 6, 100)

X1, X2 = np.meshgrid(x1, x2)

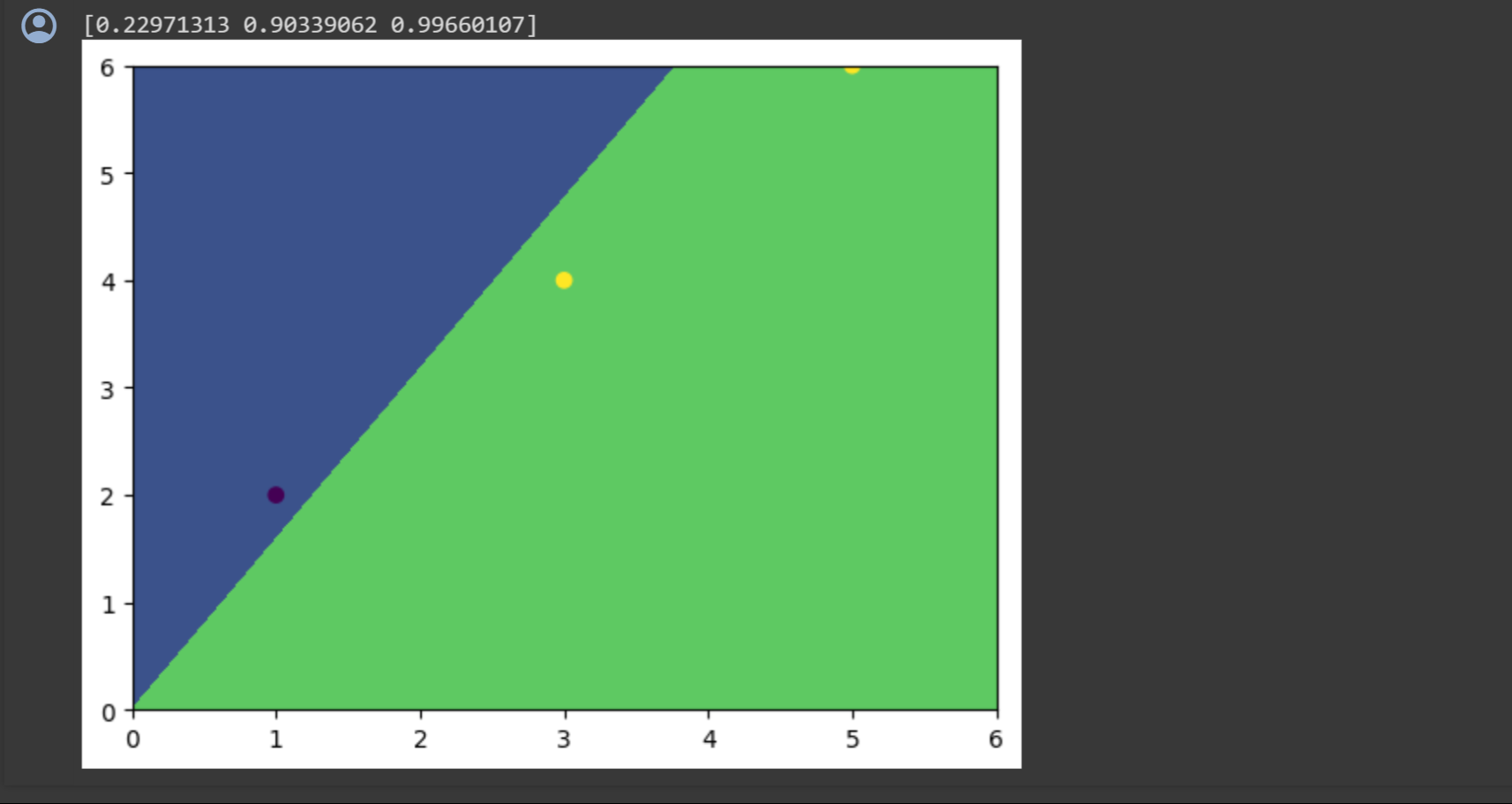
Z = logistic\_regression(np.c\_[X1.ravel(), X2.ravel()], w)

Z = Z.reshape(X1.shape)

plt.contourf(X1, X2, Z, levels=[0, 0.5, 1])

plt.scatter(X[:, 0], X[:, 1], c=y)

plt.show()



***Conclusion:*** We have successfully Implemented logistic regression using the sigmoid function.