18) To verify Gradiant Descent

import numpy as np

import matplotlib.pyplot as plt

def mean\_squared\_error(y\_true, y\_predicted):

    # Calculating the loss or cost

    cost = np.sum((y\_true-y\_predicted)\*\*2) / len(y\_true)

    return cost

# Gradient Descent Function

# Here iterations, learning\_rate, stopping\_threshold

# are hyperparameters that can be tuned

def gradient\_descent(x, y, iterations = 1000, learning\_rate = 0.0001,

                     stopping\_threshold = 1e-6):

    # Initializing weight, bias, learning rate and iterations

    current\_weight = 0.1

    current\_bias = 0.01

    iterations = iterations

    learning\_rate = learning\_rate

    n = float(len(x))

    costs = []

    weights = []

    previous\_cost = None

    # Estimation of optimal parameters

    for i in range(iterations):

        # Making predictions

        y\_predicted = (current\_weight \* x) + current\_bias

        # Calculating the current cost

        current\_cost = mean\_squared\_error(y, y\_predicted)

        # If the change in cost is less than or equal to

        # stopping\_threshold we stop the gradient descent

        if previous\_cost and abs(previous\_cost-current\_cost)<=stopping\_threshold:

            break

        previous\_cost = current\_cost

        costs.append(current\_cost)

        weights.append(current\_weight)

        # Calculating the gradients

        weight\_derivative = -(2/n) \* sum(x \* (y-y\_predicted))

        bias\_derivative = -(2/n) \* sum(y-y\_predicted)

        # Updating weights and bias

        current\_weight = current\_weight - (learning\_rate \* weight\_derivative)

        current\_bias = current\_bias - (learning\_rate \* bias\_derivative)

        # Printing the parameters for each 1000th iteration

        print(f"Iteration {i+1}: Cost {current\_cost}, Weight \

        {current\_weight}, Bias {current\_bias}")

    # Visualizing the weights and cost at for all iterations

    plt.figure(figsize = (8,8))

    plt.plot(weights, costs)

    plt.scatter(weights, costs, marker='o', color='red')

    plt.title("Cost vs Weights")

    plt.ylabel("Cost")

    plt.xlabel("Weight")

    plt.show()

    return current\_weight, current\_bias

def main():

    # Data

    X = np.array([32.50234527, 53.42680403, 61.53035803, 47.47563963, 59.81320787,

           55.14218841, 52.21179669, 39.29956669, 48.10504169, 52.55001444,

           45.41973014, 54.35163488, 44.1640495 , 58.16847072, 56.72720806,

           48.95588857, 44.68719623, 60.29732685, 45.61864377, 38.81681754])

    Y = np.array([31.70700585, 68.77759598, 62.5623823 , 71.54663223, 87.23092513,

           78.21151827, 79.64197305, 59.17148932, 75.3312423 , 71.30087989,

           55.16567715, 82.47884676, 62.00892325, 75.39287043, 81.43619216,

           60.72360244, 82.89250373, 97.37989686, 48.84715332, 56.87721319])

    # Estimating weight and bias using gradient descent

    estimated\_weight, estimated\_bias = gradient\_descent(X, Y, iterations=2000)

    print(f"Estimated Weight: {estimated\_weight}\nEstimated Bias: {estimated\_bias}")

    # Making predictions using estimated parameters

    Y\_pred = estimated\_weight\*X + estimated\_bias

    # Plotting the regression line

    plt.figure(figsize = (8,6))

    plt.scatter(X, Y, marker='o', color='red')

    plt.plot([min(X), max(X)], [min(Y\_pred), max(Y\_pred)], color='blue',markerfacecolor='red',

             markersize=10,linestyle='dashed')

    plt.xlabel("X")

    plt.ylabel("Y")

    plt.show()

if \_\_name\_\_=="\_\_main\_\_":

    main()

Result:

Iteration 1: Cost 4352.088931274409, Weight 0.7593291142562117, Bias 0.02288558130709

Iteration 2: Cost 1114.8561474350017, Weight 1.081602958862324, Bias 0.02918014748569513

Iteration 3: Cost 341.42912086804455, Weight 1.2391274084945083, Bias 0.03225308846928192

Iteration 4: Cost 156.64495290904443, Weight 1.3161239281746984, Bias 0.03375132986012604

Iteration 5: Cost 112.49704004742098, Weight 1.3537591652024805, Bias 0.034479873154934775

Iteration 6: Cost 101.9493925395456, Weight 1.3721549833978113, Bias 0.034832195392868505

Iteration 7: Cost 99.4293893333546, Weight 1.3811467575154601, Bias 0.03500062439068245

Iteration 8: Cost 98.82731958262897, Weight 1.3855419247507244, Bias 0.03507916814736111

Iteration 9: Cost 98.68347500997261, Weight 1.3876903144657764, Bias 0.035113776874486774

Iteration 10: Cost 98.64910780902792, Weight 1.3887405007983562, Bias 0.035126910596389935

Iteration 11: Cost 98.64089651459352, Weight 1.389253895811451, Bias 0.03512954755833985

Iteration 12: Cost 98.63893428729509, Weight 1.38950491235671, Bias 0.035127053821718185

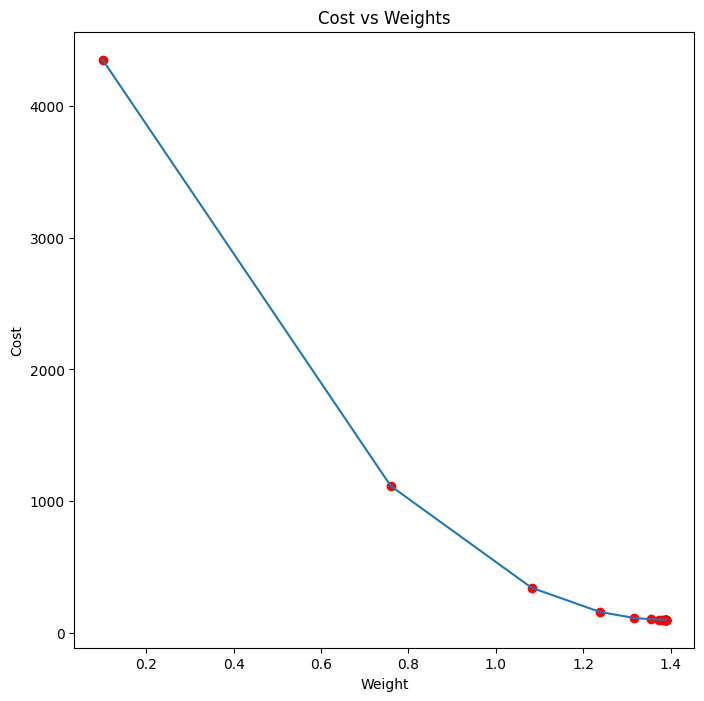
Iteration 13: Cost 98.63846506273883, Weight 1.3896276808137857, Bias 0.035122052266051224

Iteration 14: Cost 98.63835254057648, Weight 1.38968776283053, Bias 0.03511582492978764

Iteration 15: Cost 98.63832524036214, Weight 1.3897172043139192, Bias 0.03510899846107016

Iteration 16: Cost 98.63831830104695, Weight 1.389731668997059, Bias 0.035101879159522745

Iteration 17: Cost 98.63831622628217, Weight 1.389738813163012, Bias 0.03509461674147458



Estimated Weight: 1.389738813163012

Estimated Bias: 0.03509461674147458

