# Importing Libraries

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 = (8,6))

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()



