Gradient Crescent Anatomy

⇒ import math, copy
MatPlotLib, a popular Library for plotting data
⇒ import numpy as np
Numpy, a popular Library for Scientific Computing
⇒ import matplotlib.pyplot as plt
Matplotlib, a library for plotting graph
⇒ plt.style.use('./deeplearning.mplstyle')
Use plt.style.use() function from matplotlib Library
⇒ from Lab_utils_uni import plt_house_x, plt_contour_wgrad, plt_divergence, plt_gradients
plotting routines in the Lab_utils.py file in the Local directory
⇒ x_train = np.array([1.0, 2.0])
Create the x_train array with np.array([1.0, 2.0])
⇒ y_train = np.array([300.0, 500.0])
Create the y_train array with np.array([300.0, 500.0])
⇒ def compute_cost(x, y, w, b):
Function to calculate the cost
⇒ m = x.shape[0]
x.shape has a shape attribute and assign its value to m
⇒ cost = 0
cost assign zero value
⇒ for i in range(m):
for loop range from i to m
⇒ f_wb = w * x[i] + b
Linear equation where f_wb is o/p, w is weight
x[i] is I/p at index [i]

→ Cost = cost + (f_wb − y[i])**2
Calculate the squared difference between "f_wb" and y[i] and assign it to variable cost
→ Total cost = 1/(2*m) * Cost
Computed total cost providing by this formula
→ Return total_cost
Return total cost to return variable
→ def Compute_gradient (x, y, w, b):
Function to compute gradient descent
→ m = X.Shape[0]
Shape is numpy dimension array assign to m.
→ dj_dw = 0
Derivative with respect to w
→ dj_db = 0
Derivative with respect to b
→ for i in range (m):
for loop range from i to m
→ f_wb = w * x[i] + b
Linear equation where f_wb is o/p. w is weight x[i] is I/p at index [i]
→ dj_dw_i = (f_wb − y[i]) * x[i]
Derivative with respect to w
→ dj_db_i = f_wb − y[i]
Compute derivative with respect to b
→ dj_db += dj_db_i
Assign the dj_db value to dj_db_i
→ dj_dw += dj_dw_i
Assign the dj_dw value to dj_dw_i

=> dj_dw = dj_dw/m
Divide the derivative to total number of data set.
=> dj_db = dj_db/m
Divide the derivative of b to total number of data set
=> Return dj_dw, dj_db
show the derivative of w and b to return variable
=> plt_gradient(X_train, Y_train, Compute_cost, Compute_gradient)
Plot the graph b/w X and Y AXIS with respect to minimum Cost
=> plt.show()
show the graph on Screen.