import numpy as np

import matplotlib.pyplot as plt

# One step of gradient descent

def step\_gradient(b\_current, k\_current, train\_x, train\_y, learning\_rate):

b\_grad = 0

k\_grad = 0

n = len(train\_y)

for i in range(n):

x = train\_x[i]

y = train\_y[i]

b\_grad += (1/n) \* (((k\_current \* x) + b\_current) - y)

k\_grad += (1/n) \* x \* (((k\_current \* x) + b\_current) - y)

new\_b = b\_current - (learning\_rate \* b\_grad)

new\_k = k\_current - (learning\_rate \* k\_grad)

return [new\_b, new\_k]

def run\_descent(train\_x, train\_y, init\_b, init\_k, num\_iters, learning\_rate):

b = init\_b

k = init\_k

for i in range(num\_iters):

b, k = step\_gradient(b, k, train\_x, train\_y, learning\_rate)

return [b, k]

# Our main function for set up everything and run linear regression

def run():

file = 'data.csv'

points = np.array(np.genfromtxt(file, delimiter=',', skip\_header=1))

learning\_rate = 0.0000001 # I've tried .01 - .000001 but only this worked properly

train\_x = points[:,0] # sizes of the houses

train\_y = points[:,1] # prices

init\_b = 0

init\_k = 0

print('{} - number of training examples'.format(len(train\_y)))

print('k = 0, b = 0 | initial parameters')

num\_iters = 200

[b, k] = run\_descent(train\_x, train\_y, init\_b, init\_k, num\_iters, learning\_rate)

print('k = %.2f, b = %.2f | final parameters' % (k, b))

plt.plot(train\_x, train\_y, 'ro')

plt.plot([0, 7000], [0 + b, 7000 \* k + b], color='black', linestyle='-', linewidth=2) # changed the line style --

plt.xlabel('Size')

plt.ylabel('Price')

plt.tight\_layout()

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

run()