```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


# x = np.random.rand(10, 1)
# y = 2 * x + np.random.randn(10, 1)
# print(x)
# print("\n")
# print(y)


x = np.array([0.80581147, 0.38348503, 0.6652413, 0.64155897, 0.24070017, 0.35429554, 0.70827991, 0.32378987, 0.8708774 , 0.22902348])
y = np.array([ 2.27515993, -0.15308204, 1.43590601, 0.79686399, -0.45275524, 2.03862963,  1.16148089, 2.68814558, 0.70110376, 0.04881045])









def h(t0, t1):
     return (t0 + t1*x)







def J(t0, t1):
    m = len(y)
    prediction_y = h(t0, t1)
    error = np.square(prediction_y - y)
    cost = (1/(2*m)) * np.sum(error)
    return cost
```

Extra work -> code for gradient descent algorithm

```
theta_0 = 0
theta_1 = 0

def hypothesis (theta_0,theta_1,X):
    return theta_1*X + theta_0

def cost_function (X,y,theta_0,theta_1):
    m = len(X)
    summation = 0.0
    for i in range (m):
        summation += ((theta_1 * X[i] + theta_0) - y[i])**2
    return summation /(2.0*m)

def gradient_descent(X,y,theta_0,theta_1,learning_rate):
    t0_deriv = 0
    t1_deriv = 0
    m = len(X)

    for i in range (m):
        t0_deriv += (theta_1 * X[i] + theta_0) - y[i]
        t1_deriv += ((theta_1 * X[i] + theta_0) - y[i])* X[i]

    t0_deriv *= (1/m);
    t1_deriv *= (1/m);
    theta_0 -= learning_rate * t0_deriv
    theta_1 -= learning_rate * t1_deriv
    return theta_0,theta_1
```

```
def training (X, y, theta_0, theta_1, learning_rate, iters):
    cost_history = [0]
    t0_history = [0]
    t1_history = [0]

    for i in range(iters):
        theta_0,theta_1 = gradient_descent(X, y, theta_0, theta_1, learning_rate)
        t0_history.append(theta_0)
        t1_history.append(theta_1)
        cost = cost_function(X, y, theta_0, theta_1)
        cost_history.append(cost)
        print ("iter={}, theta_0={}, theta_1={}, cost= {}".format(i, theta_0, theta_1, cost))
```

Just so that i can check my values

```
training(x, y, 0, 0, 0.7, 10)
```

```
iter=0, theta_0=0.7378184071999999, theta_1=0.43184439813684977, cost= 0.49116521897867294
iter=1, theta_0=0.8012753902913154, theta_1=0.4956454849183549, cost= 0.4843873173188012
iter=2, theta_0=0.7969858878924806, theta_1=0.5217248812327018, cost= 0.4834507500789906
iter=3, theta_0=0.7861640338206259, theta_1=0.5434370028231807, cost= 0.48262246468916015
iter=4, theta_0=0.7749792128611396, theta_1=0.5641641479863776, cost= 0.4818413211379499
iter=5, theta_0=0.7640456234203411, theta_1=0.5842631998124999, cost= 0.4811041078401937
iter=6, theta_0=0.7534170434161637, theta_1=0.6037852562646958, cost= 0.4804083488419879
iter=7, theta_0=0.7430909240718381, theta_1=0.6227501236075909, cost= 0.4797517130684434
iter=8, theta_0=0.7330592592983839, theta_1=0.6411740321393511, cost= 0.4791320005809309
iter=9, theta_0=0.7233137132380598, theta_1=0.6590724453410794, cost= 0.4785471351461565
```

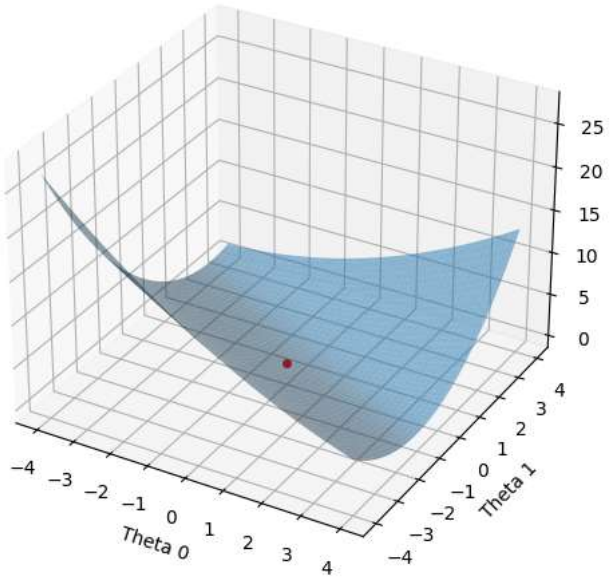
```
def plot_cost_surface(J, theta0_vals, theta1_vals, t0, t1):
    theta0_grid, theta1_grid = np.meshgrid(theta0_vals, theta1_vals)

    J_vals = np.zeros((len(theta0_vals), len(theta1_vals)))
    for i in range(len(theta0_vals)):
        for j in range(len(theta1_vals)):
            J_vals[i,j] = J(theta0_vals[i], theta1_vals[j])

    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(theta0_grid, theta1_grid, J_vals, alpha=0.5)
    ax.scatter(t0, t1, J(t0, t1), c='r', marker='o', s=15)
    ax.set_xlabel('Theta 0')
    ax.set_ylabel('Theta 1')
    ax.set_zlabel('J')
    plt.show()
```

a) Plot of J(θ0, θ1)on python with a point specifying value of J at initially taken θ0 and θ1.

```
plot_cost_surface(J, np.linspace(-4, 4, 100), np.linspace(-4, 4, 100), 0, 0)
```



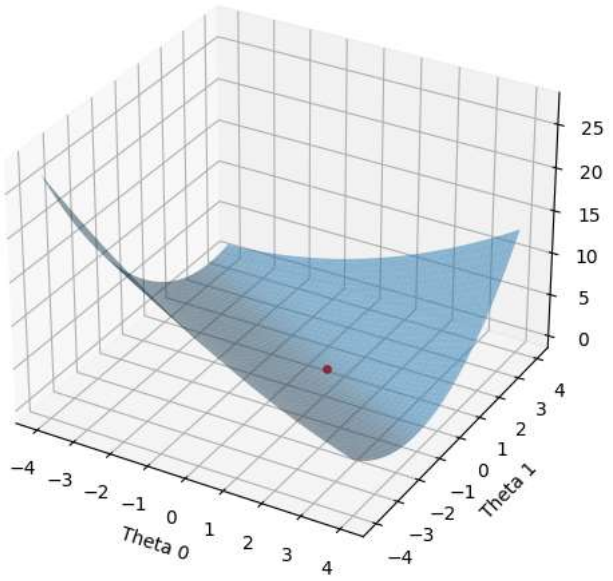
b) A table with columns J, θ0 and θ1 and at each iteration make a plot of J(θ0, θ1) on which value of J should be marked at updated(θ0, θ1) to see how far are you from your objective. Working of each iteration should also be submitted.

1st iteration: 0.737818 + 0.431844x

```
plot_cost_surface(J, np.linspace(-4, 4, 100), np.linspace(-4, 4, 100),  0.737818, 0.431844)
```

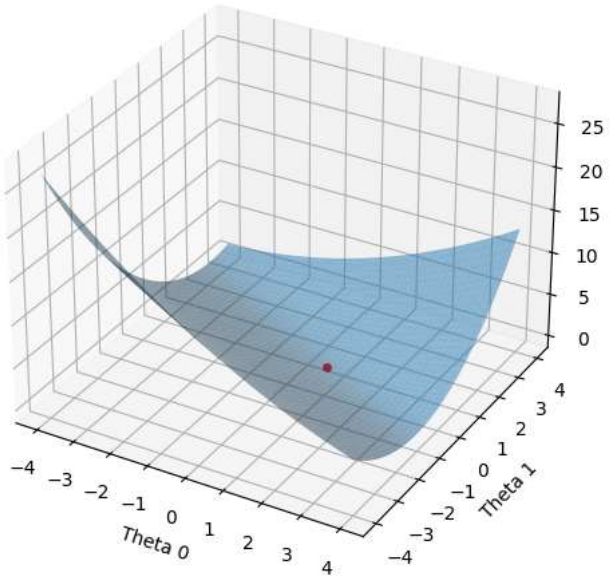
2nd iteration: 0.801275 + 0.495645x

```
plot_cost_surface(J, np.linspace(-4, 4, 100), np.linspace(-4, 4, 100),  0.801275, 0.495645)
```



3rd iteration: 0.796986 + 0.521724x

```
plot_cost_surface(J, np.linspace(-4, 4, 100), np.linspace(-4, 4, 100),  0.796986, 0.521724)
```

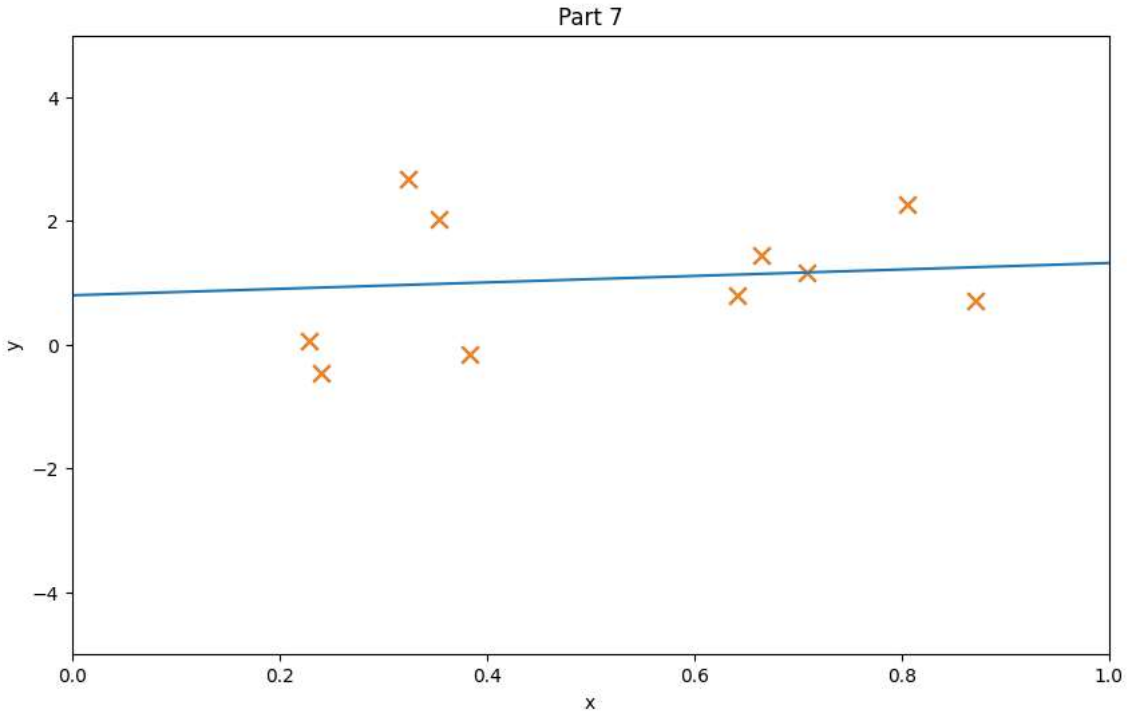


7- Draw a scatter plot of x and y with the plot of line y = θ0 + θ1x, where θ0 and θ1 are your final values obtained after the last iteration of gradient descent method.

last iteration: y = 0.796986 + 0.521724x

```
theta0_final = 0.796986
theta1_final = 0.521724

plt.figure(figsize=(10,6))
plt.axis([0, 1,-5,5])
plt.scatter(x, y, marker='x', s=80)
best_fit_x = np.linspace(0, 1, 100)
best_fit_y = [theta0_final + theta1_final*xx for xx in best_fit_x]
plt.plot(best_fit_x, best_fit_y, '-')
plt.scatter(x, y, marker='x', s=80)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Part 7')
plt.show()
```



✓ 0s completed at 6:27 PM