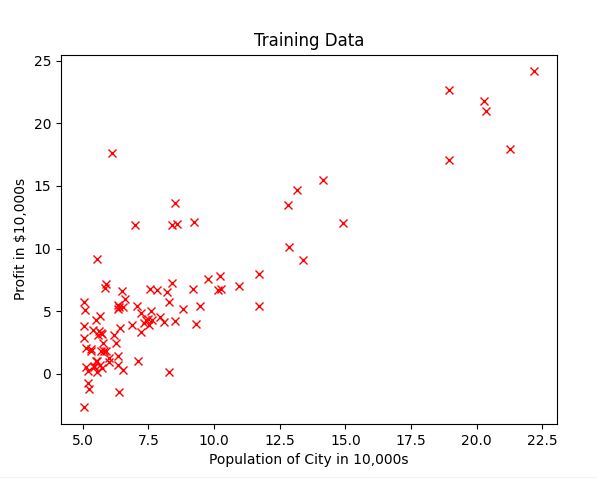
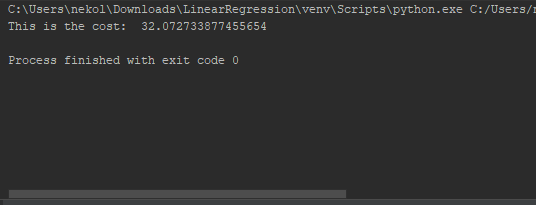
Tonya Shulkey

Linear Regression



Compute Cost:

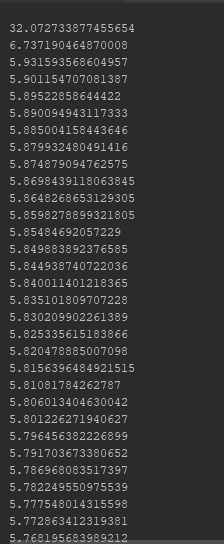
def cost(x, y, theta):  
 m = len(x[0]) #number of training examples  
 sumation = 0.0 #The values are floats so sumation should be float  
  
 for i in range(m):  
 # h(x) = theta0 + theta1x  
  
 hypothesis = theta[0] \* x[0][i] + theta[1] \* x[1][i]  
 sumation += (hypothesis - y[i]) \*\* 2  
 cost = sumation / (2.0 \* m)  
  
 return cost



Gradient Descent:

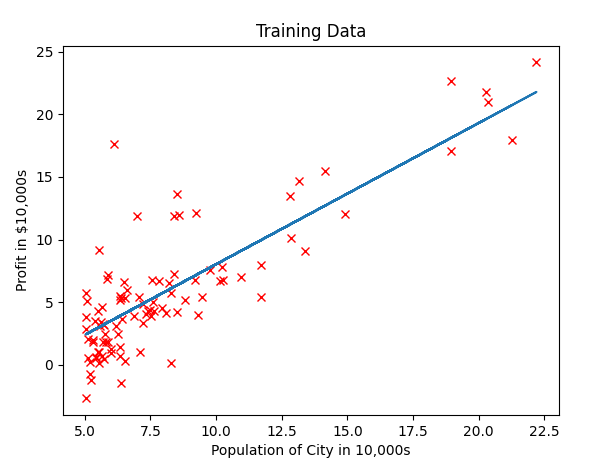
# Batch gradient descent  
def batch\_gradient\_descent(x, y, theta, alpha, iterations):  
 m = len(x[0]) #number of training examples  
  
 J = []  
 J.append(cost(x, y, theta))  
  
 cost\_it = 0  
  
 theta0 = []  
 theta1 = []  
 theta0.append(theta[0])  
 theta1.append(theta[1])  
  
 it = 0 #how many iterations occured out of the total iterations  
 while it < iterations:  
  
 sumation1 = 0.0  
 sumation2 = 0.0  
  
 for j in range(m):  
 hypothesis = theta[0] \* x[0][j] + theta[1] \* x[1][j]  
 sumation1 += (hypothesis - y[j])  
 sumation2 += (hypothesis - y[j]) \* x[1][j]  
  
 # simulatanious update  
 theta[0] = theta[0] - (alpha \* (sumation1 / m))  
 theta[1] = theta[1] - (alpha \* (sumation2 / m))  
  
 theta0.append(theta[0])  
 theta1.append(theta[1])  
  
 J.append(cost(x, y, theta))  
 cost\_it += 1  
  
 it += 1  
  
 return theta, J, theta0, theta1

Some of the cost values using gradient descent (it only decreases) using 1000 iterations and alpha = 0.01

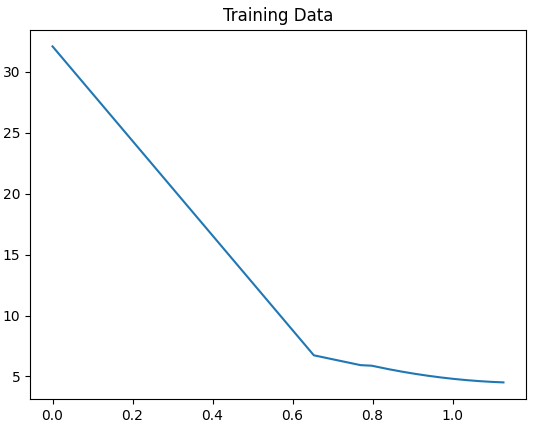


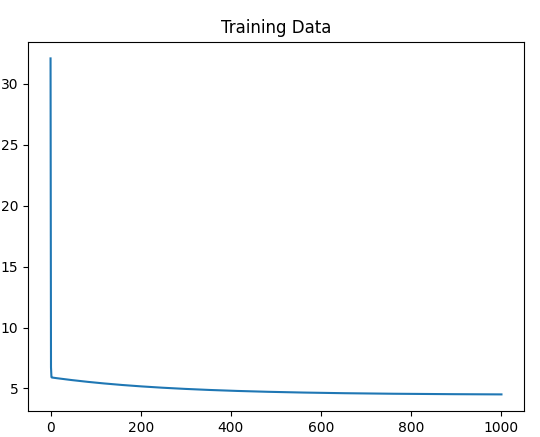
Theta0 and Theta1





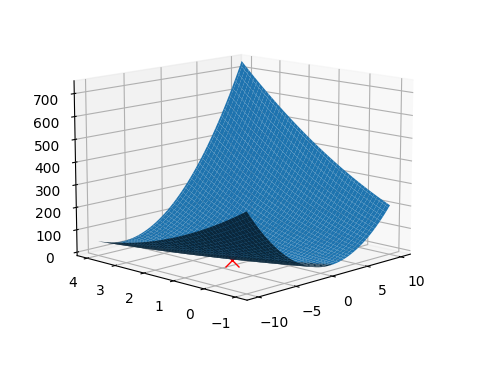
J(theta) in terms of theta1





J(theta) with the code I have already we know that J(theta) is only decreasing and it starts to flatten out as it reaches the global minimum. In the code you would store the parameters for each iteration. And plot those points on a contour graph. Where the last two thetas found are where the cost flattens.

3d Graph



Theta0 and theta1 contour graph

