**import** os  
**import** numpy **as** np  
**from** matplotlib **import** pyplot  
**import** pandas **as** pd  
  
completeData = pd.read\_csv(**r"C:\Users\user\Downloads\house\_data\_complete.csv"**).dropna()  
**for** i **in** range(3):  
 training, validation, testing = np.split(completeData.sample(frac=1), [int(.6\*len(completeData)),int(.8\*len(completeData))])  
  
 pyplot.plot(training.values[:,3], training.values[:,2], **'ro'**, ms=10, mec=**'k'**)  
 pyplot.ylabel(**'Prices'**)  
 pyplot.xlabel(**'Bedrooms'**)  
 *#pyplot.show()* lamArr = [0, 0.001, 0.002, 0.004]  
 m= training.values[:,2].size  
 mT = testing.values[:,2].size  
 mV = validation.values[:,2].size  
 **def** featureNormalize(X):  
 mu = np.mean(X, axis=0)  
 sigma = np.std(X, axis=0)  
 X\_norm = (X - mu) / sigma  
 **return** X\_norm, mu, sigma  
  
 testing = testing.drop(columns=[**'price'**, **'date'**]).values  
 testing= testing[:,1:5]  
 t\_norm, mu, sigma = featureNormalize(testing)  
 testing = np.concatenate([np.ones((mT, 1)), t\_norm], axis=1)  
  
 validation = validation.drop(columns=[**'price'**, **'date'**]).values  
 validation= validation[:,1:5]  
 v\_norm, mu, sigma = featureNormalize(validation)  
 validation = np.concatenate([np.ones((mV, 1)), v\_norm], axis=1)  
  
 X = training.drop(columns=[**'price'**, **'date'**]).values  
 X= X[:,1:5]  
 X\_norm, mu, sigma = featureNormalize(X)  
 print(**'Computed mean:'**, mu)  
 print(**'Computed standard deviation:'**, sigma)  
 X = np.concatenate([np.ones((m, 1)), X\_norm], axis=1)  
  
 initial\_theta = np.zeros(X.shape[1])  
 h1 = np.dot(X, initial\_theta)  
 h2 = np.dot(np.power(X,2) , initial\_theta)  
 ktr = X.copy()  
 ktr[:, 2] = np.power(ktr[:, 2], 2)  
 h3 = np.dot(ktr , initial\_theta)  
  
 h1t = np.dot(testing, initial\_theta)  
 h2t = np.dot(np.power(testing,2) , initial\_theta)  
 kte = testing.copy()  
 kte[:, 2] = np.power(kte[:, 2], 2)  
 h3t = np.dot(kte , initial\_theta)  
  
 h1v = np.dot(validation, initial\_theta)  
 h2v = np.dot(np.power(validation,2) , initial\_theta)  
 kv = validation.copy()  
 kv[:, 2] = np.power(kv[:, 2], 2)  
 h3v = np.dot(kv , initial\_theta)  
  
 y = training.values[:,2]  
 yT = testing[:,2]  
 yV = validation[:,2]  
  
 **def** costFunctionReg(theta, X, y, h, lambda\_, m):  
 J= np.dot((h - y), (h - y)) / (2 \* m) + ((lambda\_/(2 \* m))\* np.sum(np.dot(theta, theta)))  
 **return** J  
 **def** costFunction(y, h, m):  
 J= np.dot((h - y), (h - y)) / (2 \* m)  
 **return** J  
 aH1 =[]  
 aH2 =[]  
 aH3 =[]  
 t1= []  
 t2= []  
 t3 = []  
  
 *#for i in lamArr:  
 # cost = costFunctionReg(initial\_theta, validation, yV, h1v, i, mV)  
 # cost2 = costFunctionReg(initial\_theta, validation, yV, h2v, i, mV)  
 # cost3 = costFunctionReg(initial\_theta, validation, yV, h3v, i, mV)  
 # aH1.append(cost)  
 # aH2.append(cost2)  
 # aH3.append(cost3)  
 #lambda\_1= lamArr[min(range(len(aH1)), key=aH1.\_\_getitem\_\_)]  
 #lambda\_2= lamArr[min(range(len(aH2)), key=aH2.\_\_getitem\_\_)]  
 #lambda\_3= lamArr[min(range(len(aH3)), key=aH3.\_\_getitem\_\_)]* **for** i **in** lamArr:  
 lambda\_ = i  
 cost = costFunctionReg(initial\_theta, X, y, h1, lambda\_, m)  
 cost2 = costFunctionReg(initial\_theta, X, y, h2, lambda\_, m)  
 cost3 = costFunctionReg(initial\_theta, ktr, y, h3, lambda\_, m)  
 print(**'Cost at initial theta (zeros): {:.3f}'**.format(cost3))  
 **def** gradientDescent(X, y, theta, alpha, num\_iters, lambda\_):  
 m = y.shape[0] *# number of training examples* theta = theta.copy()  
 J\_history = [] *# Use a python list to save cost in every iteration* **for** i **in** range(num\_iters):  
 alphabym = alpha / m  
 h = np.dot(X, theta)  
 theta = theta\*(1 - (alpha\*lambda\_)/m) - ((alpha / m) \* (np.dot(X.T, h - y)))  
 J\_history.append(costFunctionReg(theta, X, y, h, lambda\_, m))  
  
 **return** theta, J\_history  
  
 **def** gradientDescent2(X, y, theta, alpha, num\_iters, lambda\_):  
 m = y.shape[0] *# number of training examples* theta = theta.copy()  
 J\_history = [] *# Use a python list to save cost in every iteration* **for** i **in** range(num\_iters):  
 alphabym = alpha / m  
 h = np.dot(np.power(X,2), theta)  
 theta = theta\*(1 - (alpha\*lambda\_)/m) - ((alpha / m) \* (np.dot(X.T, h - y)))  
 J\_history.append(costFunctionReg(theta, X, y, h, lambda\_, m))  
  
 **return** theta, J\_history  
  
 **def** gradientDescent3(X, y, theta, alpha, num\_iters, lambda\_):  
 m = y.shape[0] *# number of training examples* theta = theta.copy()  
 J\_history = [] *# Use a python list to save cost in every iteration* **for** i **in** range(num\_iters):  
 alphabym = alpha / m  
 h = np.dot(ktr, theta)  
 theta = theta\*(1 - (alpha\*lambda\_)/m) - ((alpha / m) \* (np.dot(X.T, h - y)))  
 J\_history.append(costFunctionReg(theta, X, y, h, lambda\_, m))  
  
 **return** theta, J\_history  
  
 iterations = 150  
 alpha = 0.01  
 alpha2 = 0.003  
 theta, J\_history = gradientDescent(X,y, initial\_theta, alpha, iterations, i)  
 theta2, J\_history2 = gradientDescent2(X,y, initial\_theta, alpha2, iterations, i)  
 theta3, J\_history3 = gradientDescent3(ktr,y, initial\_theta, alpha, iterations, i)  
 h1r = np.dot(validation, theta)  
 h2r = np.dot(np.power(validation, 2), theta2)  
 kr = validation.copy()  
 kr[:, 2] = np.power(kr[:, 2], 2)  
 h3r = np.dot(kr, theta3)  
 aH1.append(costFunction(yV, h1r, mV))  
 aH2.append(costFunction(yV, h2r, mV))  
 aH3.append(costFunction(yV, h3r, mV))  
  
 lambda\_1= lamArr[min(range(len(aH1)), key=aH1.\_\_getitem\_\_)]  
 lambda\_2= lamArr[min(range(len(aH2)), key=aH2.\_\_getitem\_\_)]  
 lambda\_3= lamArr[min(range(len(aH3)), key=aH3.\_\_getitem\_\_)]  
 print(lambda\_1, **'|'**,lambda\_2, **'|'**,lambda\_3)  
 theta, J\_history = gradientDescent(X, y, initial\_theta, alpha, iterations, lambda\_1)  
 theta2, J\_history2 = gradientDescent2(X, y, initial\_theta, alpha2, iterations, lambda\_2)  
 theta3, J\_history3 = gradientDescent3(ktr, y, initial\_theta, alpha, iterations, lambda\_3)  
  
 pyplot.figure()  
 pyplot.plot(np.arange(len(J\_history)), J\_history, lw=2, label=**'h1'**)  
 pyplot.plot(np.arange(len(J\_history2)), J\_history2, lw=2, label=**'h2'**)  
 pyplot.plot(np.arange(len(J\_history3)), J\_history3, lw=2, label=**'h3'**)  
 pyplot.legend()  
 pyplot.xlabel(**'Number of iterations'**)  
 pyplot.ylabel(**'Cost Function'**)  
 pyplot.show()  
 jTestH1 = costFunctionReg(theta, testing, yT, h1t, lambda\_1, mT)  
 jTestH2 = costFunctionReg(theta2, testing, yT, h2t, lambda\_2, mT)  
 jTestH3 = costFunctionReg(theta3, testing, yT, h3t, lambda\_3, mT)  
 t1.append(jTestH1)  
 t2.append(jTestH2)  
 t3.append(jTestH3)  
t1Avg = np.mean(t1)  
t2Avg = np.mean(t2)  
t3Avg = np.mean(t3)  
print(t1Avg)  
print(t2Avg)  
print(t3Avg)