**Part a:**

# Dr Jianhua Chen

# Programming assignment part a

# Student name: Seyedeh Shaghayegh Rabbanian 899645944 (srabba2@lsu.edu)

# CSC 7333

#Importing packages which are needed for programming

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Part a (Reading the dataset KCSmall2.csv)

mydataset= pd.read\_csv('KCSmall2.csv',header=None)

x = mydataset.iloc[:,0]

y = mydataset.iloc[:,1]

# General code: Part a (Performing gradient descent to learn parameter vector (theta0 = b ,theta1 = a)

# First we initialize values for a and be to be 0. We specify values for number of iterations and learning rate and m which is the number of

# training examples equals to length of x. After that, by using for loop we calculate our predicted value for y, loss (error) function and our

# derivatives. Then, by using update rule, we update the values for our parameters. This for loop continues until we complete our number of iterations

def gradient\_descent(x,y):

b\_curr = a\_curr = 0

iterations = 1000

m = len(x)

learning\_rate = 0.1

for i in range(iterations):

y\_predicted = a\_curr \* x + b\_curr

loss = (1/(2\*m)) \* sum([val\*\*2 for val in (y-y\_predicted)])

ad = -(1/m)\*sum(x\*(y-y\_predicted))

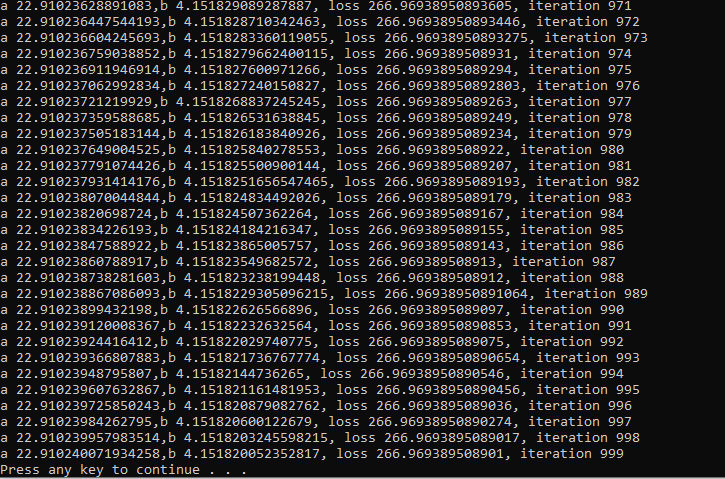
bd = -(1/m)\*sum(y-y\_predicted)

a\_curr = a\_curr - learning\_rate \* ad

b\_curr = b\_curr - learning\_rate \* bd

print ("a {},b {}, loss {}, iteration {}".format(a\_curr,b\_curr,loss,i))

gradient\_descent(x,y)



# Desired output of the program

# (1) Plot the data

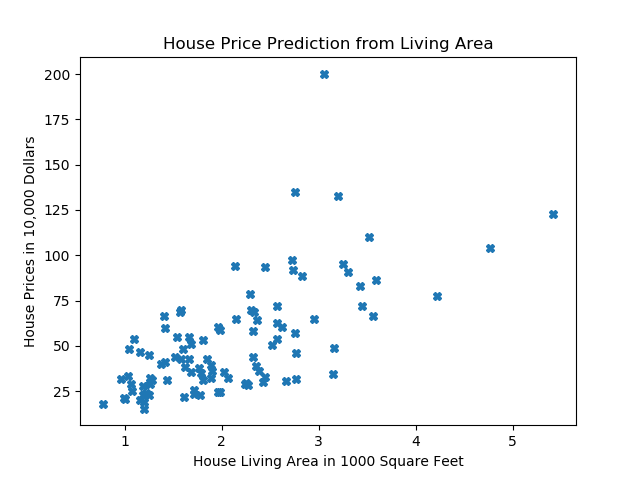
plt.plot(x,y,'X')

plt.title('House Price Prediction from Living Area')

plt.xlabel('House Living Area in 1000 Square Feet')

plt.ylabel('House Prices in 10,000 Dollars')

plt.show()



# (2) Print the loss function for (0,0) and (-1,20)

m = len(x)

y\_predicted2 = (0) \* x + 0

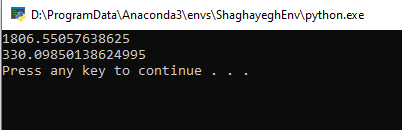
loss2 = (1/(2\*m)) \* sum([val\*\*2 for val in (y-y\_predicted2)])

print(loss2)

y\_predicted3 = (20) \* x -1

loss3 = (1/(2\*m)) \* sum([val\*\*2 for val in (y-y\_predicted3)])

print(loss3)



# (3) Plot the loss function for n=15 iterations (alpha= 0.01,0.1,0.2,0.4) and print out parameters (I will get learning rate by using input)

def gradient\_descent(x,y):

b\_curr = a\_curr = 0

iterations = 15

m = len(x)

learning\_rate = float(input('Enter learning rate value: '))

loss\_list = []

for i in range(iterations):

y\_predicted = a\_curr \* x + b\_curr

loss = (1/(2\*m)) \* sum([val\*\*2 for val in (y-y\_predicted)])

loss\_list.append(loss)

ad = -(1/m)\*sum(x\*(y-y\_predicted))

bd = -(1/m)\*sum(y-y\_predicted)

a\_curr = a\_curr - learning\_rate \* ad

b\_curr = b\_curr - learning\_rate \* bd

print ("a {},b {}, loss {}, iteration {}".format(a\_curr,b\_curr,loss,i))

plt.plot(list(range(iterations)),loss\_list,'X')

plt.title('Loss Function Curve for 15 iterations (alpha = '+ str(learning\_rate) + ')')

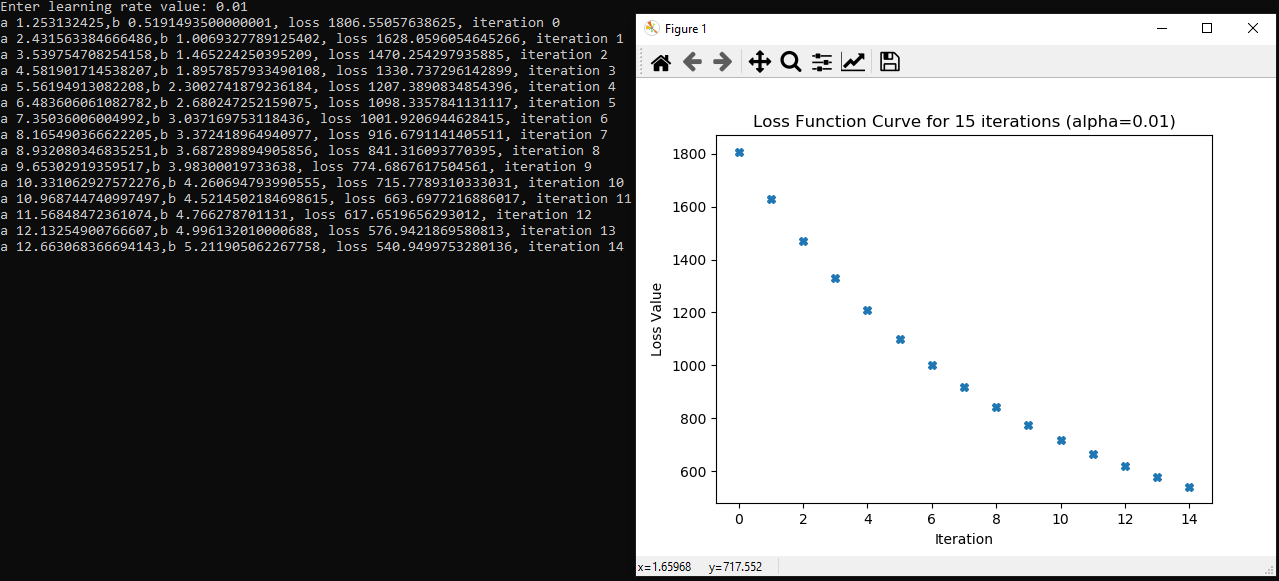
plt.xlabel('Iteration')

plt.ylabel('Loss Value')

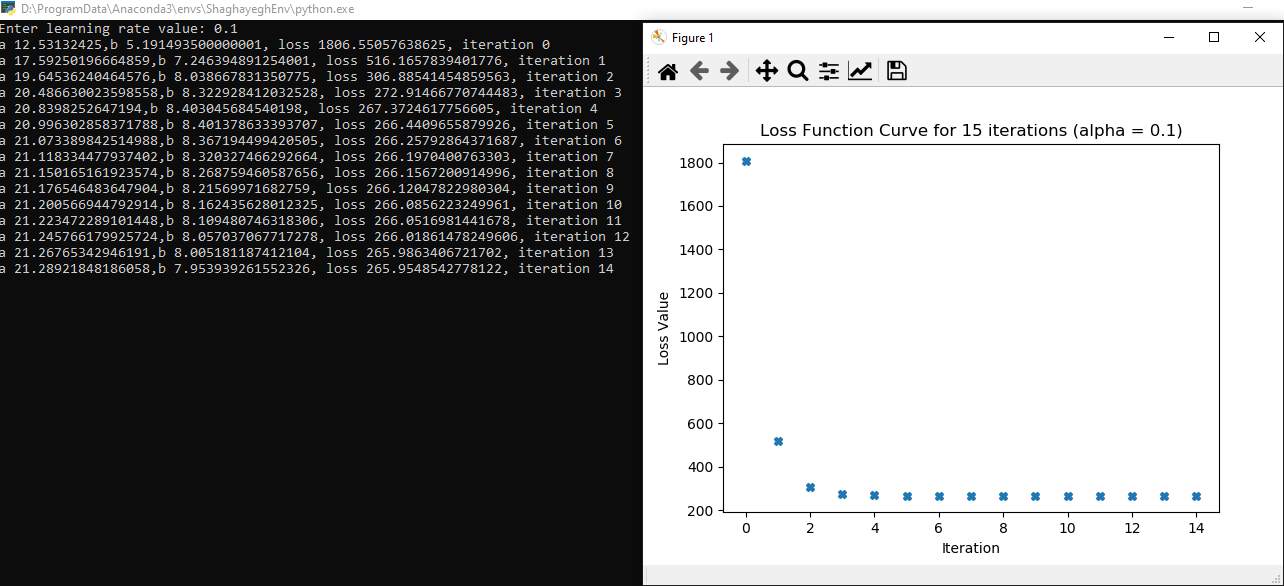
plt.show()

gradient\_descent(x,y)

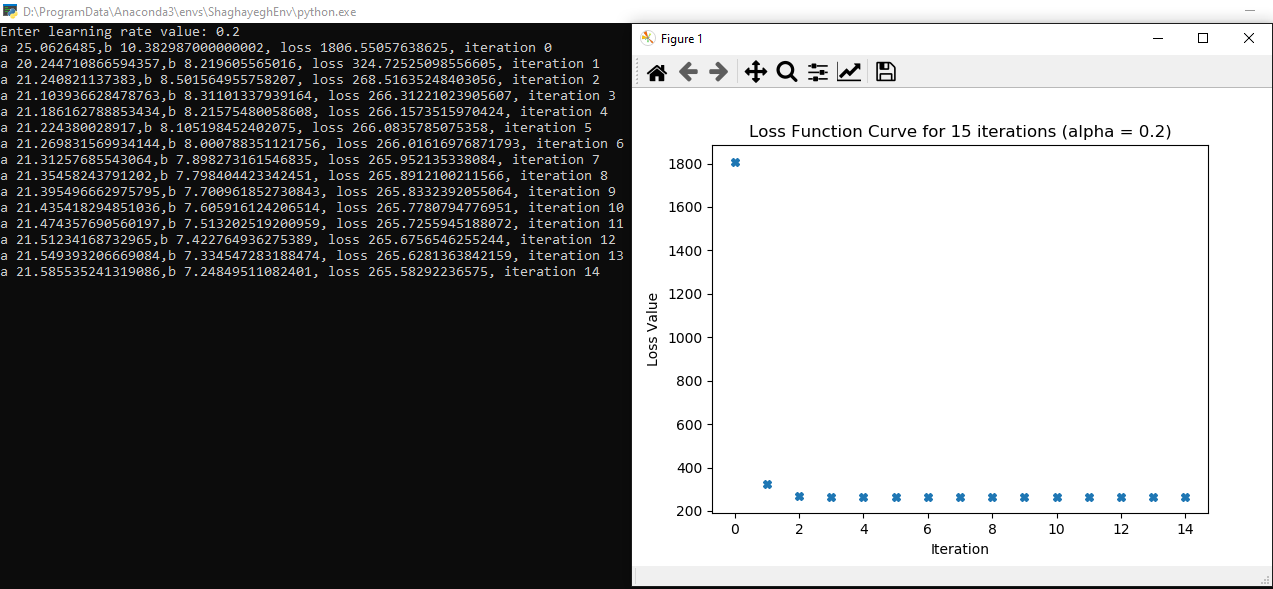
Learning rate = 0.01



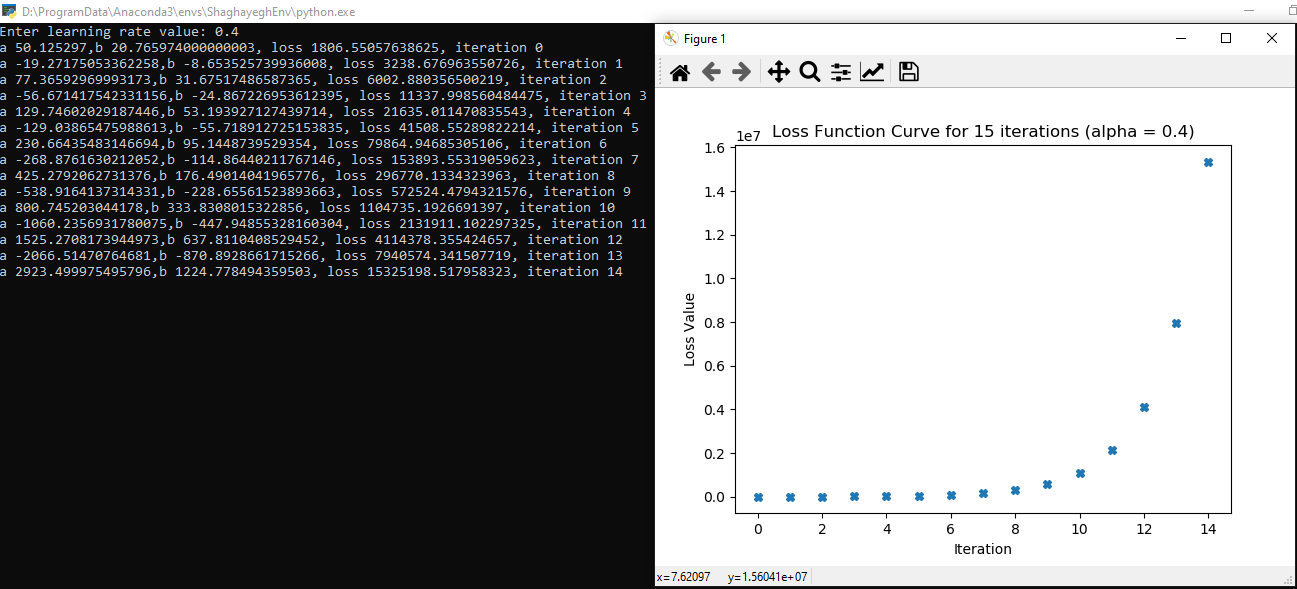
Learning rate = 0.1



Learning rate = 0.2



Learning rate = 0.4



Loss function with learning\_rate=0.2 reaches its least value after 15 iterations which is 265.5829.

**Final parameters value:**

# (4) Predict y\_predicted for x=3.5 and x=7

print('(4) Predict y\_predicted for x=3.5 and x=7')

New\_x=[3.5,7]

for l in New\_x:

price=a\_curr\*l+b\_curr

print("Living area: {}, Price: {}".format(l,price))



**Part b:**

# Dr Jianhua Chen

# Programming assignment part b

# Student name: Seyedeh Shaghayegh Rabbanian 899645944 (srabba2@lsu.edu)

# CSC 7333

# Importing packages

import numpy as np

import pandas as pd

from sklearn import preprocessing

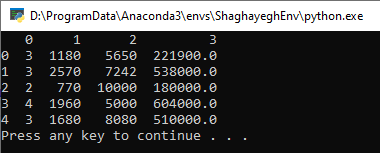
import matplotlib.pyplot as plt

# Loading dataset

mydataset= pd.read\_csv('KCSmall\_NS2.csv',header=None)

# (1) Print out the first 5 rows of the raw input data

print(mydataset[0:5])



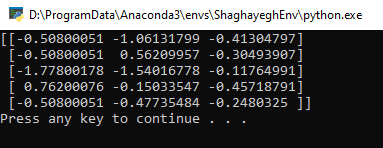
x = mydataset.iloc[:,0:3]

y = mydataset.iloc[:,3]

# (2) Standardizing input data

x\_scaled = preprocessing.scale(x)

print(x\_scaled)



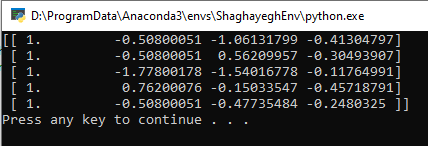
# Defining a dummy column ( Column of all 1)

dummy\_column = np.ones((100,1),dtype='int32')

# Adding the dummy column to the scaled matrix

Final\_x = np.append(dummy\_column,x\_scaled,axis = 1)

print(Final\_x[0:5])



# (3) Print out the cost J value for theta = (0,0,0,0)

n= Final\_x.shape[1]

theta = np.zeros(n)

print(theta)

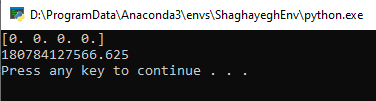
def cost\_function(Final\_x,y):

m=len(y)

loss=np.sum((Final\_x.dot(theta)-y)\*\*2)/(2\*m)

print(loss)

cost\_function(Final\_x,y)



# (4) Run gradient descent for n=50 iterations with alpha=(0.01,0.1,0.5,1,1.5)

def gradient\_descent(x,y):

theta = np.zeros(n)

iterations = 50

m = len(y)

learning\_rate = float(input('Enter learning rate value: '))

loss\_list = []

for i in range(iterations):

y\_predicted = Final\_x.dot(theta)

loss=np.sum((Final\_x.dot(theta)-y)\*\*2)/(2\*m)

loss\_list.append(loss)

thetad = (1/m)\*((Final\_x.T).dot(y\_predicted-y))

theta = theta - learning\_rate \* thetad

print ("Theta{}, loss {}, iteration {}".format(theta,loss,i))

plt.plot(list(range(iterations)),loss\_list,'X')

plt.title('Loss Function Curve for 50 iterations (alpha = '+ str(learning\_rate) + ')')

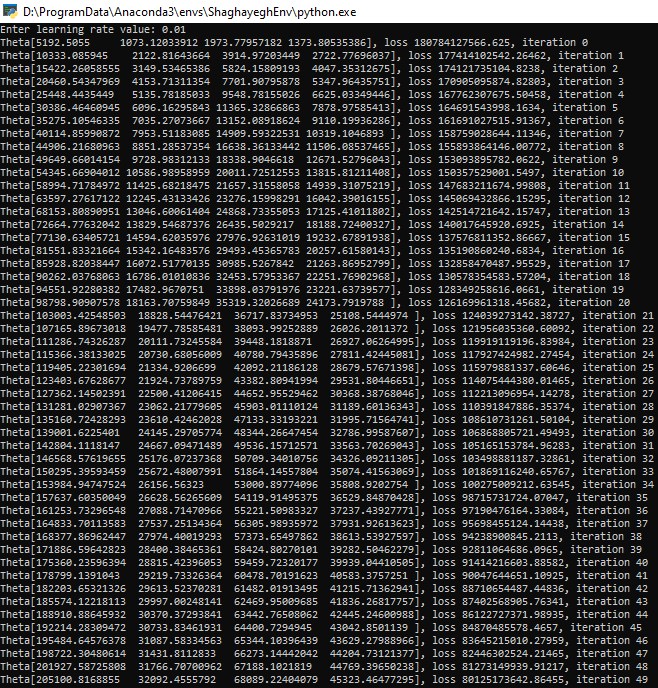
plt.xlabel('Iteration')

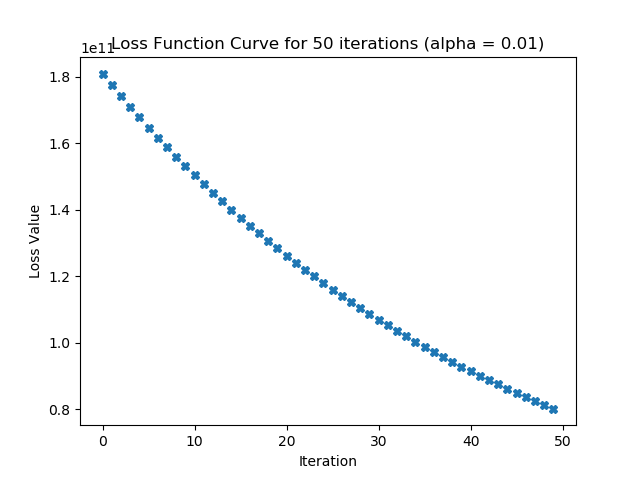
plt.ylabel('Loss Value')

plt.show()

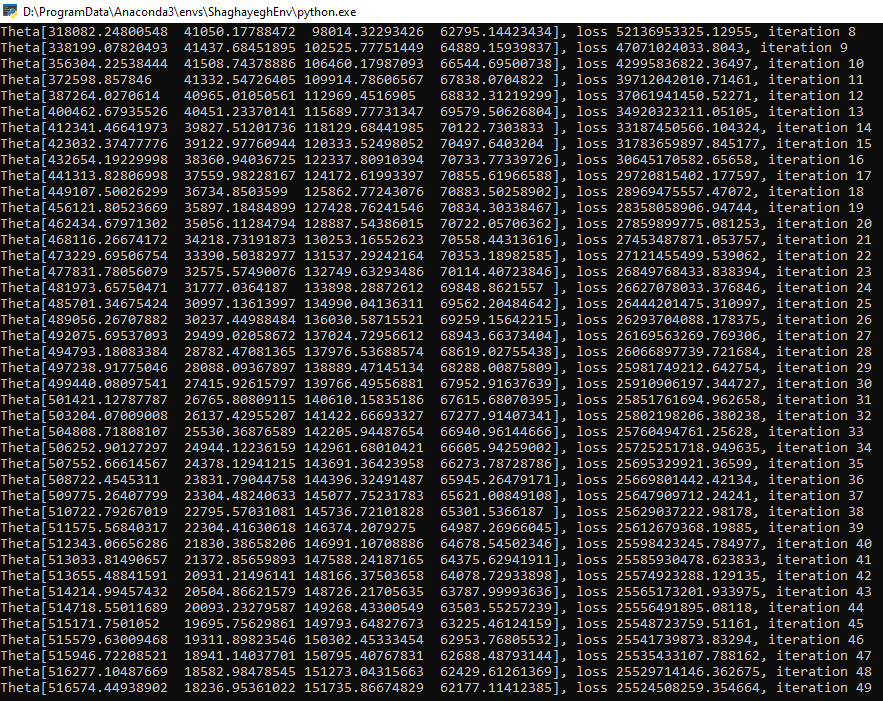
gradient\_descent(x,y)

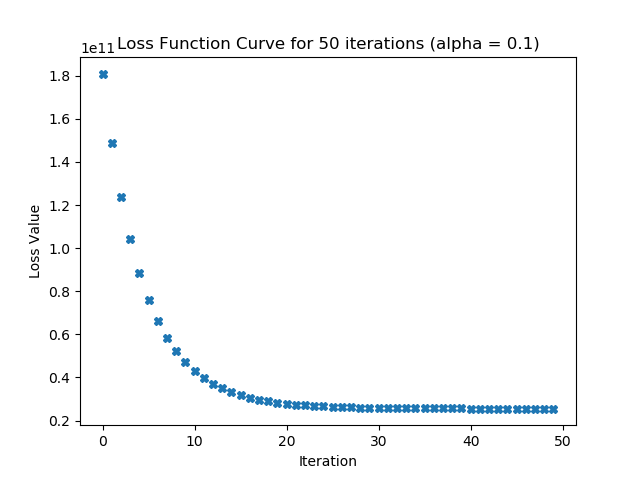
Learning rate=0.01



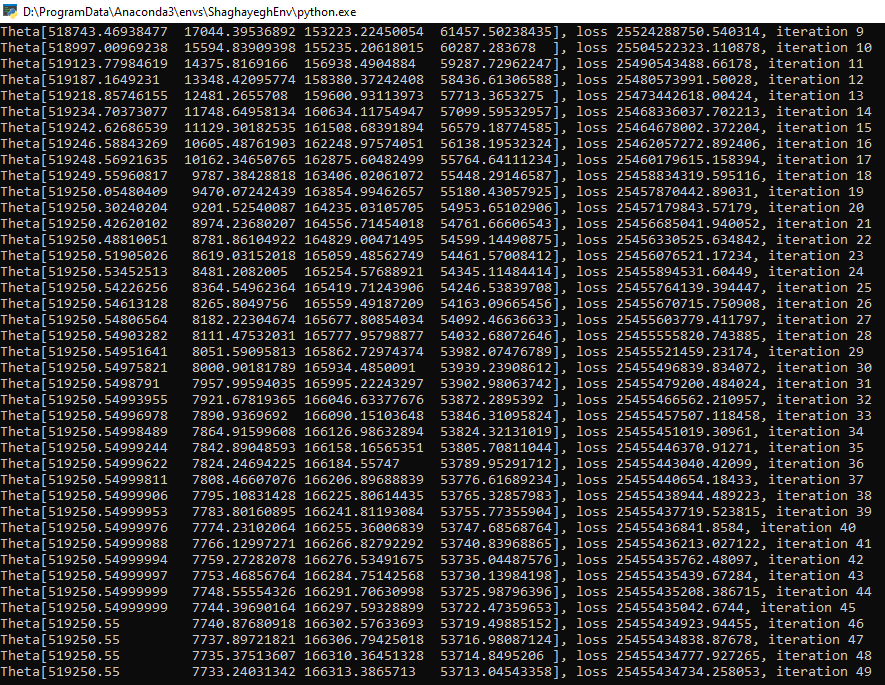


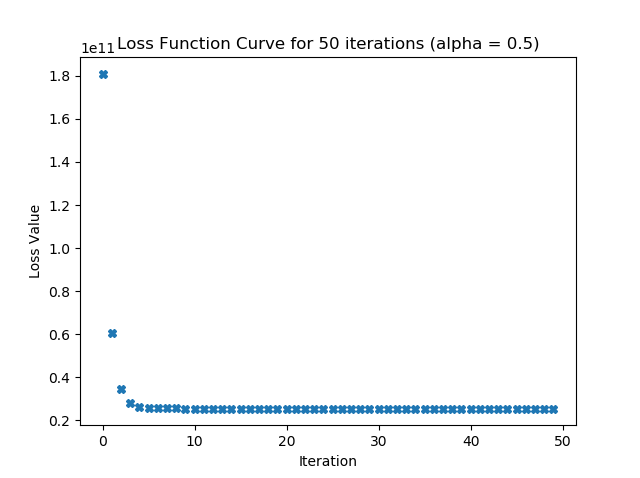
Learning rate=0.1



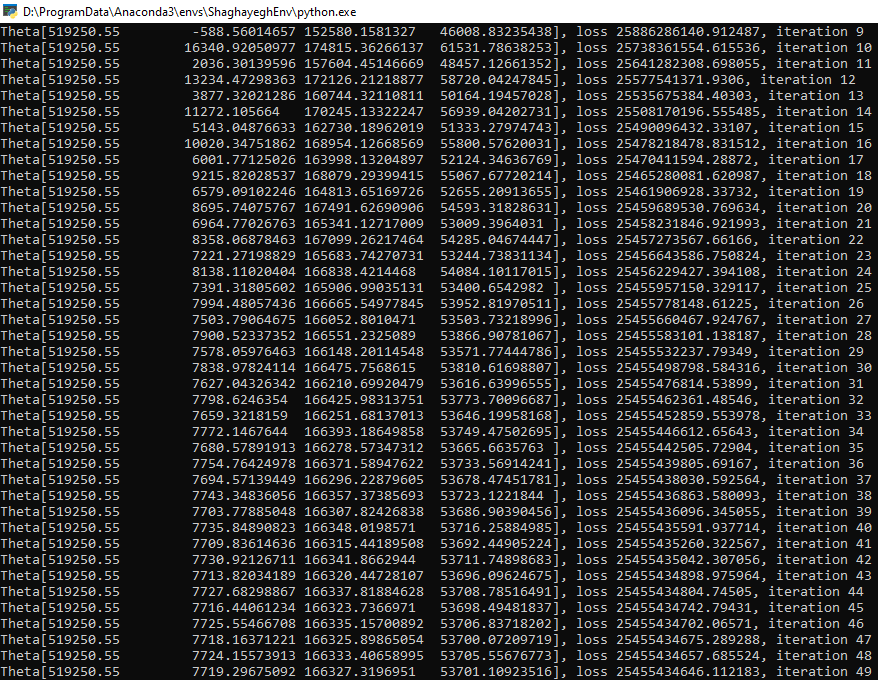


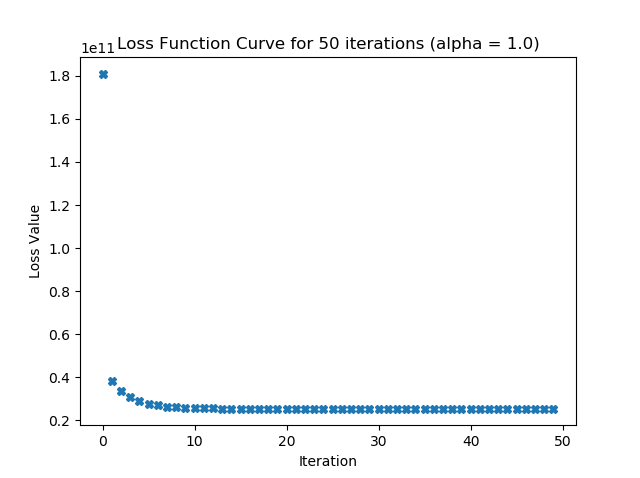
Learning rate=0.5



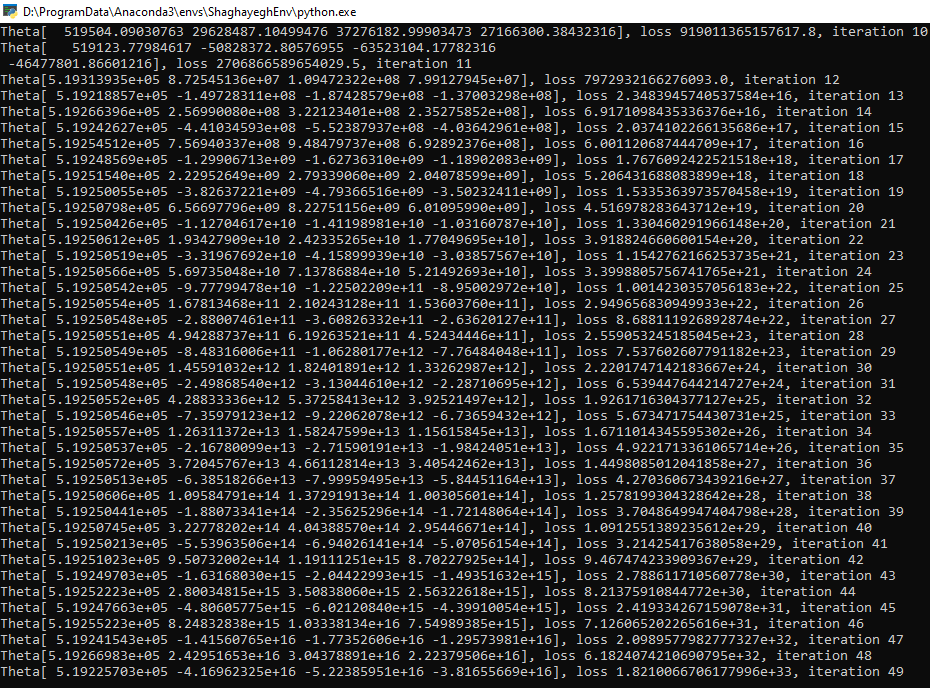


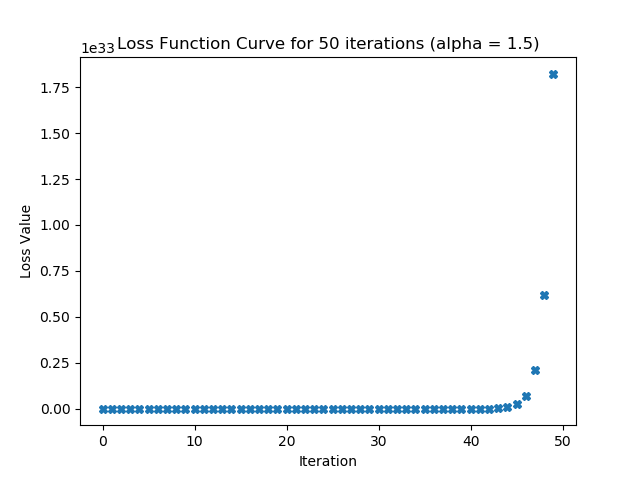
Learning rate=1.0





Learning rate=1.5





Loss function with learning\_rate=1 reaches its least value after 50 iterations which is 25455434646.112.

**Final parameters value:**

# (5) Print out the predicted y value for the input n\_bed=3, liv\_area=2000, lot\_area=8550.

# First we should standardize data point.

x\_test = np.array([3, 2000, 8550])

colaverage = x.mean(axis = 0)

colstd = x.std(axis = 0)

standard = (x\_test-colaverage)/colstd

#print(standard)

#Adding dummy column to standardized matix

dummy\_column2 = np.array([1])

Final\_xtest = np.append(dummy\_column2,standard,axis = 0)

print(Final\_xtest)

New\_theta = np.array([519250.55 ,7719.29675092,166327.3196951,53701.10923516])

print(New\_theta)

Predictedvalue = Final\_xtest.dot(New\_theta.T)

print(Predictedvalue)

