```
#Importing neccessary modules
import numpy as np
import pandas as pd
```

```
#reading the csv data into a pandas dataframe
data = pd.read_csv('A1_HousePrice.csv')
```

```
#Checking the data
data.head()
```

	Size of the house (in square feet)	Number of bedrooms	Price of the house
0	2104	3	399900
1	1600	3	329900
2	2400	3	369000
3	1416	2	232000
4	3000	4	539900

```
#checking the data dimensions
data.shape
```

```
(47, 3)
```

```
# assigning data values in X1 and Y1 variables and converting to numpy array
X1 = np.array(data.iloc[:,:-1])
Y1 = np.array(data.iloc[:,-1])
```

```
#Making sure dimensions are as required so reshaping the variables
X1 = X1.reshape(X1.shape[0],2)
Y1 = Y1.reshape(Y1.shape[0],1)
print(X1.shape)
print(Y1.shape)
```

```
(47, 2)
(47, 1)
```

1. Normalising the Data

```
norm_X1 = (X1-X1.mean())/X1.std()
norm_X1
```

```
array([[ 9.64150080e-01, -8.73910212e-01],
       [ 5.23225574e-01, -8.73910212e-01],
       [ 1.22310574e+00, -8.73910212e-01],
       [ 3.62253135e-01, -8.74785062e-01],
```

```
       [ 1.74801587e+00, -8.73035361e-01],
       [ 8.60042905e-01, -8.73035361e-01],
       [ 4.65485460e-01, -8.73910212e-01],
       [ 3.71876488e-01, -8.73910212e-01],
       [ 3.30758528e-01, -8.73910212e-01],
       [ 4.30491452e-01, -8.73910212e-01],
       [ 8.20674646e-01, -8.73035361e-01],
       [ 8.73165658e-01, -8.73910212e-01],
       [ 7.76932135e-01, -8.73910212e-01],
       [ 3.04104448e+00, -8.72160511e-01],
       [ 2.32775304e-01, -8.73910212e-01],
       [ 1.13562072e+00, -8.73035361e-01],
       [ 2.78267515e-01, -8.74785062e-01],
       [ 2.04780098e-01, -8.73910212e-01],
       [ 1.40594944e+00, -8.73035361e-01],
       [ 1.77513622e+00, -8.73035361e-01],
       [ 6.69325559e-01, -8.73910212e-01],
       [ 7.75182435e-01, -8.74785062e-01],
       [ 5.26724975e-01, -8.73910212e-01],
       [ 8.39921350e-01, -8.73035361e-01],
       [ 2.52663256e+00, -8.73910212e-01],
       [ 8.58004690e-02, -8.73910212e-01],
       [ 3.98996844e-01, -8.73910212e-01],
       [ 1.33333687e+00, -8.73910212e-01],
       [ 1.04813570e+00, -8.73910212e-01],
       [ 1.43044524e+00, -8.73910212e-01],
       [ 7.32314774e-01, -8.74785062e-01],
       [-1.68455200e-03, -8.75659912e-01],
       [ 9.08159667e-01, -8.73035361e-01],
       [ 1.86787035e+00, -8.73910212e-01],
       [ 7.07818968e-01, -8.73035361e-01],
       [ 3.80624990e-01, -8.73910212e-01],
       [ 2.07404648e-01, -8.73910212e-01],
       [ 9.88645886e-01, -8.73035361e-01],
       [ 2.81095887e+00, -8.73035361e-01],
       [ 1.01489139e+00, -8.73035361e-01],
       [ 5.79215988e-01, -8.74785062e-01],
       [ 1.08138001e+00, -8.73910212e-01],
       [ 1.36920573e+00, -8.73035361e-01],
       [ 1.73285490e-01, -8.73910212e-01],
       [-1.31162383e-01, -8.74785062e-01],
       [ 7.43687827e-01, -8.73035361e-01],
       [ 1.75910041e-01, -8.73910212e-01]])
```

```
norm_Y1 = (Y1-Y1.mean())/Y1.std()
norm_Y1
```

```
array([[ 0.48089023],
       [-0.08498338],
       [ 0.23109745],
       [-0.87639804],
       [ 1.61263744],
       [-0.32750064],
       [-0.20624201],
       [-1.1431751 ],
       [-1.03807621],
       [-0.791517  ],
       [-0.81173485],
       [ 0.05325146],
       [-0.08418307],
```

```
       [ 2.90606282],
       [-0.65085698],
       [ 0.88508566],
       [-0.32750064],
       [-1.1358915 ],
       [ 1.29007331],
       [ 2.09039644],
       [-0.70744435],
       [-0.69046814],
       [-0.78828343],
       [-0.65085698],
       [ 1.88749033],
       [-0.73169607],
       [ 1.00311072],
       [ 1.03948831],
       [ 1.08799176],
       [-0.32750064],
       [ 0.07669479],
       [-1.37840876],
       [-0.20624201],
       [ 1.93599378],
       [-0.44067536],
       [-0.73169607],
       [-0.89337424],
       [ 0.03708364],
       [ 1.686201  ],
       [-0.43178306],
       [ 0.22705549],
       [-0.08498338],
       [-0.21351753],
       [-0.33477616],
       [-1.29756968],
       [-0.32750064],
       [-0.81576872]])
```

```
print(norm_X1.shape , norm_Y1.shape)
```

```
(47, 2) (47, 1)
```

```
# adding a column of ones for further dot products
X = np.c_[np.ones(norm_X1.shape[0]),norm_X1]
```

```
X
```

```
array([[ 1.00000000e+00,  9.64150080e-01, -8.73910212e-01],
       [ 1.00000000e+00,  5.23225574e-01, -8.73910212e-01],
       [ 1.00000000e+00,  1.22310574e+00, -8.73910212e-01],
       [ 1.00000000e+00,  3.62253135e-01, -8.74785062e-01],
       [ 1.00000000e+00,  1.74801587e+00, -8.73035361e-01],
       [ 1.00000000e+00,  8.60042905e-01, -8.73035361e-01],
       [ 1.00000000e+00,  4.65485460e-01, -8.73910212e-01],
       [ 1.00000000e+00,  3.71876488e-01, -8.73910212e-01],
       [ 1.00000000e+00,  3.30758528e-01, -8.73910212e-01],
       [ 1.00000000e+00,  4.30491452e-01, -8.73910212e-01],
       [ 1.00000000e+00,  8.20674646e-01, -8.73035361e-01],
       [ 1.00000000e+00,  8.73165658e-01, -8.73910212e-01],
       [ 1.00000000e+00,  7.76932135e-01, -8.73910212e-01],
       [ 1.00000000e+00,  3.04104448e+00, -8.72160511e-01],
```

```
       [ 1.00000000e+00,  2.32775304e-01, -8.73910212e-01],
       [ 1.00000000e+00,  1.13562072e+00, -8.73035361e-01],
       [ 1.00000000e+00,  2.78267515e-01, -8.74785062e-01],
       [ 1.00000000e+00,  2.04780098e-01, -8.73910212e-01],
       [ 1.00000000e+00,  1.40594944e+00, -8.73035361e-01],
       [ 1.00000000e+00,  1.77513622e+00, -8.73035361e-01],
       [ 1.00000000e+00,  6.69325559e-01, -8.73910212e-01],
       [ 1.00000000e+00,  7.75182435e-01, -8.74785062e-01],
       [ 1.00000000e+00,  5.26724975e-01, -8.73910212e-01],
       [ 1.00000000e+00,  8.39921350e-01, -8.73035361e-01],
       [ 1.00000000e+00,  2.52663256e+00, -8.73910212e-01],
       [ 1.00000000e+00,  8.58004690e-02, -8.73910212e-01],
       [ 1.00000000e+00,  3.98996844e-01, -8.73910212e-01],
       [ 1.00000000e+00,  1.33333687e+00, -8.73910212e-01],
       [ 1.00000000e+00,  1.04813570e+00, -8.73910212e-01],
       [ 1.00000000e+00,  1.43044524e+00, -8.73910212e-01],
       [ 1.00000000e+00,  7.32314774e-01, -8.74785062e-01],
       [ 1.00000000e+00, -1.68455200e-03, -8.75659912e-01],
       [ 1.00000000e+00,  9.08159667e-01, -8.73035361e-01],
       [ 1.00000000e+00,  1.86787035e+00, -8.73910212e-01],
       [ 1.00000000e+00,  7.07818968e-01, -8.73035361e-01],
       [ 1.00000000e+00,  3.80624990e-01, -8.73910212e-01],
       [ 1.00000000e+00,  2.07404648e-01, -8.73910212e-01],
       [ 1.00000000e+00,  9.88645886e-01, -8.73035361e-01],
       [ 1.00000000e+00,  2.81095887e+00, -8.73035361e-01],
       [ 1.00000000e+00,  1.01489139e+00, -8.73035361e-01],
       [ 1.00000000e+00,  5.79215988e-01, -8.74785062e-01],
       [ 1.00000000e+00,  1.08138001e+00, -8.73910212e-01],
       [ 1.00000000e+00,  1.36920573e+00, -8.73035361e-01],
       [ 1.00000000e+00,  1.73285490e-01, -8.73910212e-01],
       [ 1.00000000e+00, -1.31162383e-01, -8.74785062e-01],
       [ 1.00000000e+00,  7.43687827e-01, -8.73035361e-01],
       [ 1.00000000e+00,  1.75910041e-01, -8.73910212e-01]])
```

```
#defining theta values
np.random.seed(123)
theta = np.random.rand(3)
theta = theta.reshape((3,1))

alpha = 0.01  #alpha values for calculating new thetas


## FWD Propagation
m = norm_Y1.size

h_theta = np.dot(X,theta)  #calculating h theta
h_theta
error = h_theta - norm_Y1  #calculating error
error
cost = 1/(2*m)*np.dot(X.T,error)  #finding cost
cost

theta = theta - alpha*(1/m)*np.dot(X.T,error)  #calculating new theta
theta
```

```
array([[0.68898646],
       [0.28412766],
       [0.23339176]])
```

```
## Backward Propagation
alpha = 0.01
epoch = 10000
m = norm_Y1.size
np.random.seed(123)
theta = np.random.rand(3)
theta = theta.reshape((3,1))

def GD(X1,Y1,theta,epoch,alpha): # gradient descent function  to find best theta

  past_cost = []
  past_theta = [theta]
  for i in range(epoch):
    h_theta = np.dot(X1,theta)
    error = h_theta -Y1
    cost = 1/(2*m)*np.dot(error.T,error)
    past_cost.append(cost)
    theta = theta - alpha*1/m*np.dot(X1.T,error)
    past_theta.append(theta)
    if (np.round(past_theta[i],6) == np.round(past_theta[i+1],6)).all():
          break
  return past_theta,past_cost , i+1


past_theta , past_cost,best_epoch = GD(X,norm_Y1,theta,epoch,alpha)
```

```
best_theta = past_theta[-1] #finding best value of theta
print(best_theta)
```

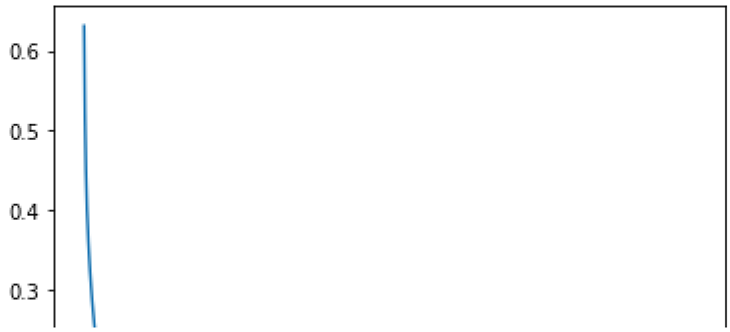
```
[[-0.20244439]
 [ 1.24249438]
 [ 1.01078756]]
```

```
best_epoch
```

```
2672
```

```
cost1= np.asarray((past_cost)) #having costs to plot in future
cost1.shape
cost1 = cost1.reshape((10000,1))
```

```
import matplotlib.pyplot as plt
plt.plot(cost1)
plt.show()
```




```
import statsmodels.api as sm
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarnin
  import pandas.util.testing as tm
```

```
regressor = sm.OLS(norm_Y1,sm.add_constant(norm_X1)).fit() #compare with linear regression
print(regressor.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.733
Model:                            OLS   Adj. R-squared:                  0.721
Method:                 Least Squares   F-statistic:                     60.38
Date:                Mon, 02 Aug 2021   Prob (F-statistic):           2.43e-13
Time:                        10:27:04   Log-Likelihood:                -35.663
No. Observations:                  47   AIC:                             77.33
Df Residuals:                      44   BIC:                             82.88
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -71.6734    124.814     -0.574      0.569    -323.218     179.872
x1             1.2864      0.137      9.409      0.000       1.011       1.562
x2           -80.7422    142.770     -0.566      0.575    -368.475     206.991
==============================================================================
Omnibus:                        4.083   Durbin-Watson:                   1.826
Prob(Omnibus):                  0.130   Jarque-Bera (JB):                2.977
Skew:                           0.567   Prob(JB):                        0.226
Kurtosis:                       3.484   Cond. No.                     3.99e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
[2] The condition number is large, 3.99e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```