Assignment 3 : Gradient Descent

Aryaman Gautam

J001

In [146…

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Univariate Linear Regression

In [147…

```
data = pd.read_csv("ex1data1.txt", header = None)
```

In [148…

```
data.head()
```

Out[148…

	0	1
0	6.1101	17.5920
1	5.5277	9.1302
2	8.5186	13.6620
3	7.0032	11.8540
4	5.8598	6.8233

In [149…

```
data.describe()
```

Out[149…

	0	1
count	97.000000	97.000000
mean	8.159800	5.839135
std	3.869884	5.510262
min	5.026900	-2.680700
25%	5.707700	1.986900
50%	6.589400	4.562300
75%	8.578100	7.046700
max	22.203000	24.147000

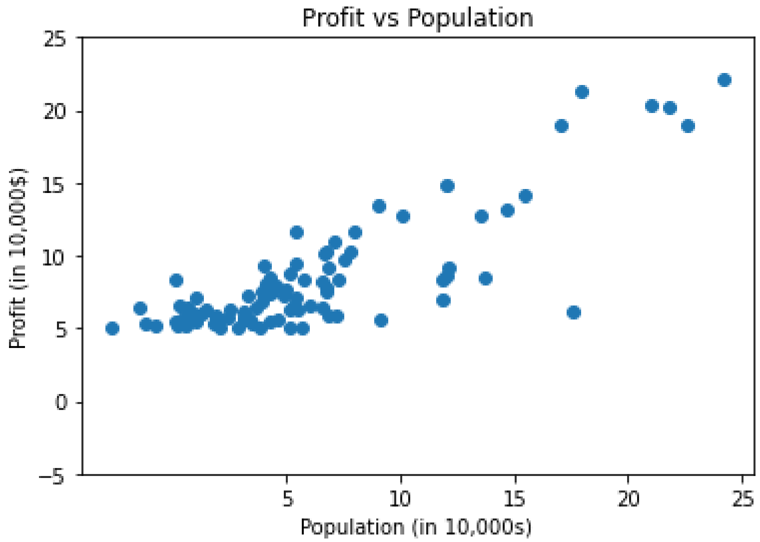
In [150…

```
data.columns = ['Profit','Population']
```

In [151…

```
plt.scatter(data['Population'], data['Profit'])
plt.xticks(np.arange(5,30,step=5))
plt.yticks(np.arange(-5,30,step=5))
plt.xlabel('Population (in 10,000s)')
plt.ylabel('Profit (in 10,000$)')
plt.title('Profit vs Population')
```

Out[151… `Text(0.5, 1.0, 'Profit vs Population')`



Cost Function J(θ)

In [152…
```
def computeCost(X,y,theta):
    """
    Take in a numpy arary X,y,theta and get cost function using theta as parameter in a linear regression model
    """
    m = len(y)
    predictions = X.dot(theta)
    square_err = (predictions - y)**2

    return 1/(m)*np.sum(square_err)
```

In [153…
```
data['x0'] = 1 #new column added
```

In [154…
```
data.values
```

Out[154…
```
array([[ 6.1101 , 17.592  ,  1.     ],
       [ 5.5277 ,  9.1302 ,  1.     ],
       [ 8.5186 , 13.662  ,  1.     ],
```

```
[ 7.0032 , 11.854  ,  1.      ],
[ 5.8598 ,  6.8233 ,  1.      ],
[ 8.3829 , 11.886  ,  1.      ],
[ 7.4764 ,  4.3483 ,  1.      ],
[ 8.5781 , 12.     ,  1.      ],
[ 6.4862 ,  6.5987 ,  1.      ],
[ 5.0546 ,  3.8166 ,  1.      ],
[ 5.7107 ,  3.2522 ,  1.      ],
[14.164  , 15.505  ,  1.      ],
[ 5.734  ,  3.1551 ,  1.      ],
[ 8.4084 ,  7.2258 ,  1.      ],
[ 5.6407 ,  0.71618,  1.      ],
[ 5.3794 ,  3.5129 ,  1.      ],
[ 6.3654 ,  5.3048 ,  1.      ],
[ 5.1301 ,  0.56077,  1.      ],
[ 6.4296 ,  3.6518 ,  1.      ],
[ 7.0708 ,  5.3893 ,  1.      ],
[ 6.1891 ,  3.1386 ,  1.      ],
[20.27   , 21.767  ,  1.      ],
[ 5.4901 ,  4.263  ,  1.      ],
[ 6.3261 ,  5.1875 ,  1.      ],
[ 5.5649 ,  3.0825 ,  1.      ],
[18.945  , 22.638  ,  1.      ],
[12.828  , 13.501  ,  1.      ],
[10.957  ,  7.0467 ,  1.      ],
[13.176  , 14.692  ,  1.      ],
[22.203  , 24.147  ,  1.      ],
[ 5.2524 , -1.22   ,  1.      ],
[ 6.5894 ,  5.9966 ,  1.      ],
[ 9.2482 , 12.134  ,  1.      ],
[ 5.8918 ,  1.8495 ,  1.      ],
[ 8.2111 ,  6.5426 ,  1.      ],
[ 7.9334 ,  4.5623 ,  1.      ],
[ 8.0959 ,  4.1164 ,  1.      ],
[ 5.6063 ,  3.3928 ,  1.      ],
[12.836  , 10.117  ,  1.      ],
[ 6.3534 ,  5.4974 ,  1.      ],
[ 5.4069 ,  0.55657,  1.      ],
[ 6.8825 ,  3.9115 ,  1.      ],
[11.708  ,  5.3854 ,  1.      ],
[ 5.7737 ,  2.4406 ,  1.      ],
[ 7.8247 ,  6.7318 ,  1.      ],
[ 7.0931 ,  1.0463 ,  1.      ],
[ 5.0702 ,  5.1337 ,  1.      ],
[ 5.8014 ,  1.844  ,  1.      ],
```

```
[11.7    ,  8.0043 ,  1.     ],
[ 5.5416 ,  1.0179 ,  1.     ],
[ 7.5402 ,  6.7504 ,  1.     ],
[ 5.3077 ,  1.8396 ,  1.     ],
[ 7.4239 ,  4.2885 ,  1.     ],
[ 7.6031 ,  4.9981 ,  1.     ],
[ 6.3328 ,  1.4233 ,  1.     ],
[ 6.3589 , -1.4211 ,  1.     ],
[ 6.2742 ,  2.4756 ,  1.     ],
[ 5.6397 ,  4.6042 ,  1.     ],
[ 9.3102 ,  3.9624 ,  1.     ],
[ 9.4536 ,  5.4141 ,  1.     ],
[ 8.8254 ,  5.1694 ,  1.     ],
[ 5.1793 , -0.74279,  1.     ],
[21.279  , 17.929  ,  1.     ],
[14.908  , 12.054  ,  1.     ],
[18.959  , 17.054  ,  1.     ],
[ 7.2182 ,  4.8852 ,  1.     ],
[ 8.2951 ,  5.7442 ,  1.     ],
[10.236  ,  7.7754 ,  1.     ],
[ 5.4994 ,  1.0173 ,  1.     ],
[20.341  , 20.992  ,  1.     ],
[10.136  ,  6.6799 ,  1.     ],
[ 7.3345 ,  4.0259 ,  1.     ],
[ 6.0062 ,  1.2784 ,  1.     ],
[ 7.2259 ,  3.3411 ,  1.     ],
[ 5.0269 , -2.6807 ,  1.     ],
[ 6.5479 ,  0.29678,  1.     ],
[ 7.5386 ,  3.8845 ,  1.     ],
[ 5.0365 ,  5.7014 ,  1.     ],
[10.274  ,  6.7526 ,  1.     ],
[ 5.1077 ,  2.0576 ,  1.     ],
[ 5.7292 ,  0.47953,  1.     ],
[ 5.1884 ,  0.20421,  1.     ],
[ 6.3557 ,  0.67861,  1.     ],
[ 9.7687 ,  7.5435 ,  1.     ],
[ 6.5159 ,  5.3436 ,  1.     ],
[ 8.5172 ,  4.2415 ,  1.     ],
[ 9.1802 ,  6.7981 ,  1.     ],
[ 6.002  ,  0.92695,  1.     ],
[ 5.5204 ,  0.152  ,  1.     ],
[ 5.0594 ,  2.8214 ,  1.     ],
[ 5.7077 ,  1.8451 ,  1.     ],
[ 7.6366 ,  4.2959 ,  1.     ],
[ 5.8707 ,  7.2029 ,  1.     ],
```

```
       [ 5.3054 ,  1.9869 ,  1.      ],
       [ 8.2934 ,  0.14454,  1.      ],
       [13.394  ,  9.0551 ,  1.      ],
       [ 5.4369 ,  0.61705,  1.      ]])
```

In [155…

```
data_val = data.values
m =  len(data_val[:-1])
X = data[['x0','Population']].iloc[:-1].values
y = data['Profit'][:-1].values.reshape(m,1)
theta = np.zeros((2,1))

m, X.shape, y.shape, theta.shape
```

Out[155…

```
(96, (96, 2), (96, 1), (2, 1))
```

In [156…

```
theta
```

Out[156…

```
array([[0.],
       [0.]])
```

In [157…

```
computeCost(X,y,theta)
```

Out[157…

```
81.94398512635416
```

Gradient Descent

In [158…

```
def gradientDescent(X,y,theta,alpha,num_iters):
    """
    Take numpy array for X,y,theta and update theta for every iteration of gradient steps
    Return theta and the list of cost of theta during each iteration
    """

    m = len(y)
    J_history = []
    for i in range(num_iters):
        predictions = X.dot(theta)
        error = np.dot(X.transpose(), (predictions-y))
```

```python
        descent = alpha * 1/m * error
        theta-=descent
        J_history.append(computeCost(X,y,theta))

    return theta, J_history
```

In [159…

```python
theta, J_history = gradientDescent(X,y,theta,0.001, 2000)
```

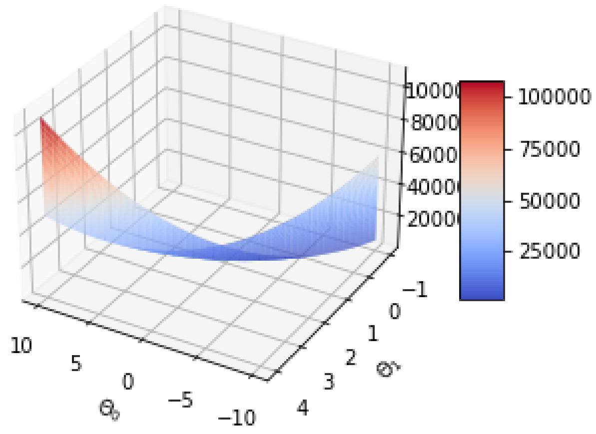
In [160…

```python
print(f"h(x) = {str(round(theta[0,0],2))} + {str(round(theta[1,0],2))}x1")
```

```
h(x) = 2.88 + 0.76x1
```

In [161…

```python
from mpl_toolkits.mplot3d import Axes3D
#Generating values for theta0, theta1 and the resulting cost value
theta0_vals=np.linspace(-10,10,100)
theta1_vals=np.linspace(-1,4,100)
J_vals=np.zeros((len(theta0_vals),len(theta1_vals)))
for i in range(len(theta0_vals)):
    for j in range(len(theta1_vals)):
        t=np.array([theta0_vals[i],theta1_vals[j]])
        J_vals[i,j]=computeCost(X,y,t)
#Generating the surface plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
surf=ax.plot_surface(theta0_vals,theta1_vals,J_vals,cmap="coolwarm")
fig.colorbar(surf, shrink=0.5, aspect=5)
ax.set_xlabel("$\Theta_0$")
ax.set_ylabel("$\Theta_1$")
ax.set_zlabel("$J(\Theta)$")
#rotate for better angle
ax.view_init(30,120)
```

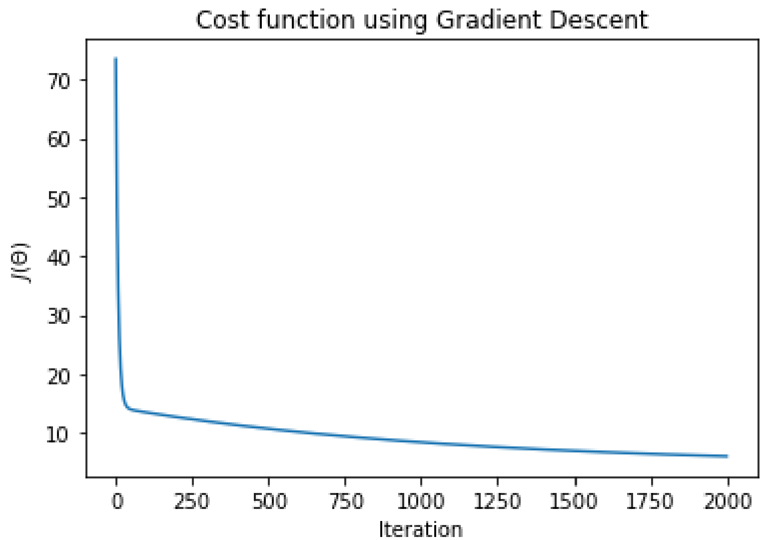


In [162…

```
plt.plot(J_history)
plt.xlabel("Iteration")
plt.ylabel("$J(\Theta)$")
plt.title("Cost function using Gradient Descent")
```

Out[162…

```
Text(0.5, 1.0, 'Cost function using Gradient Descent')
```

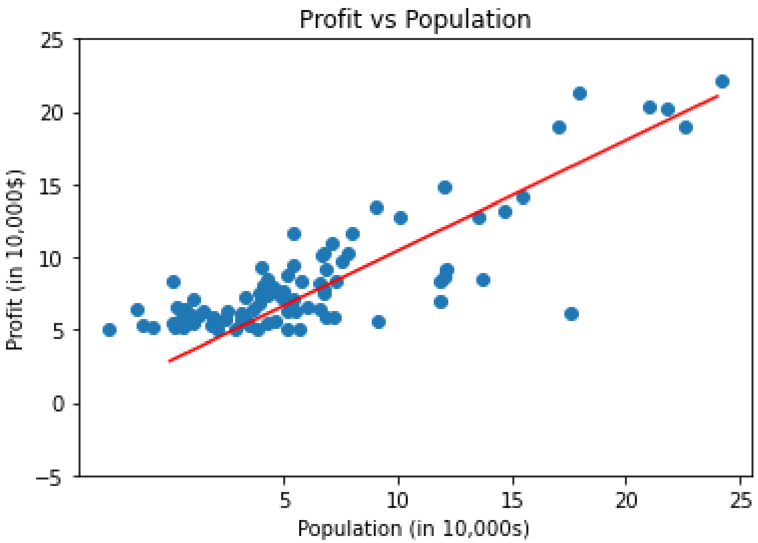


In [163…

```
plt.scatter(data['Population'], data['Profit'])
x_value = [x for x in range(25)]
y_value = [x*theta[1] + theta[0] for x in x_value]
plt.plot(x_value, y_value, color = 'r')
plt.xticks(np.arange(5,30,step=5))
plt.yticks(np.arange(-5,30,step=5))
plt.xlabel('Population (in 10,000s)')
plt.ylabel('Profit (in 10,000$)')
plt.title('Profit vs Population')
```

Out[163…

```
Text(0.5, 1.0, 'Profit vs Population')
```



In [164…

```
def predict(x,theta):
    """
    Takes in numpy array x and theta and returns predicted value of y
    """
    predictions = np.dot(theta.transpose(),x)
    return predictions[0]
```

In [165…

```
data.tail(1)
```

Out[165…

	Profit	Population	x0
96	5.4369	0.61705	1

In [166…

```
predict1 = predict(data[['x0','Population']].iloc[-1].values, theta)*10000
print(f'For a population of 6170 the predicted profit is ${predict1}')
```

```
For a population of 6170 the predicted profit is $33425.787565695806
```

Multivariate Regression

In [167…

```
data2 = pd.read_csv('ex1data2.txt',header=None)
data2.head()
```

Out[167…

	0	1	2
0	2104	3	399900
1	1600	3	329900
2	2400	3	369000
3	1416	2	232000
4	3000	4	539900

In [168…

```
data2.describe()
```

Out[168…

	0	1	2
count	47.000000	47.000000	47.000000
mean	2000.680851	3.170213	340412.659574
std	794.702354	0.760982	125039.899586
min	852.000000	1.000000	169900.000000
25%	1432.000000	3.000000	249900.000000

	0	1	2
50%	1888.000000	3.000000	299900.000000
75%	2269.000000	4.000000	384450.000000
max	4478.000000	5.000000	699900.000000

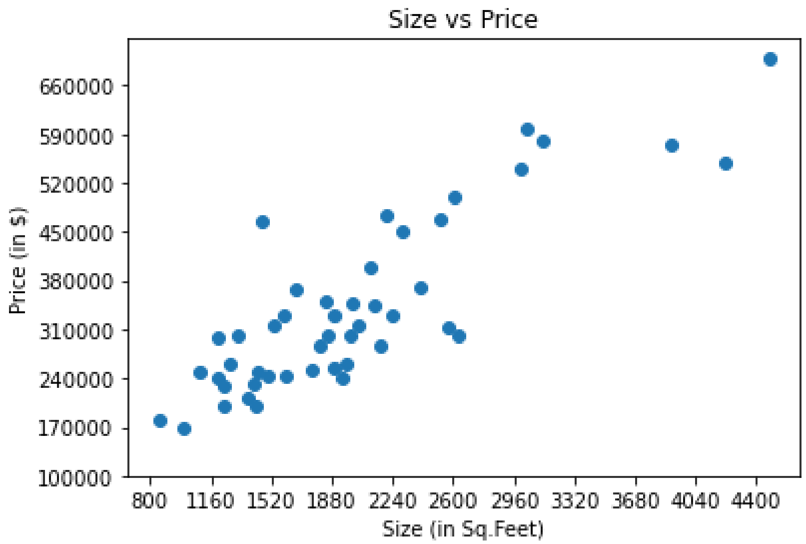
In [169…

```
data2.columns = ['Size of Hause(Sq. feet)','No. of Rooms',"Price"]
```

In [170…

```
plt.scatter(data2['Size of Hause(Sq. feet)'], data2['Price'])
plt.xticks(np.arange(800,4478,step=360))
plt.yticks(np.arange(100000,699900,step=70000))
plt.xlabel('Size (in Sq.Feet)')
plt.ylabel('Price (in $)')
plt.title('Size vs Price')
```

Out[170… Text(0.5, 1.0, 'Size vs Price')



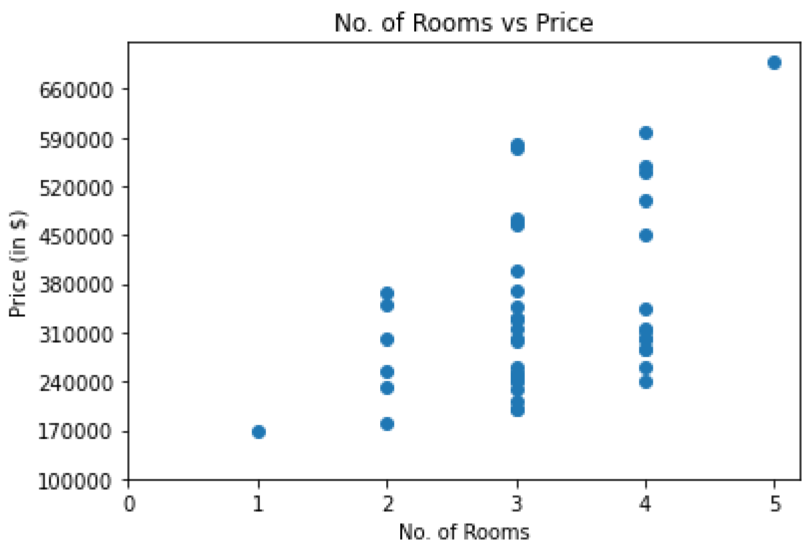
In [171…

```
plt.scatter(data2['No. of Rooms'], data2['Price'])
plt.xticks(np.arange(0,6,step=1))
```

```
plt.yticks(np.arange(100000,699900,step=70000))
plt.xlabel('No. of Rooms')
plt.ylabel('Price (in $)')
plt.title('No. of Rooms vs Price')
```

Out[171…

```
Text(0.5, 1.0, 'No. of Rooms vs Price')
```



In [172…

```
data2.columns = ['Size of the house (in square feet)','Number of bedrooms','Price of the house']
data2.head()
```

Out[172…

	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

In [173…

```
def normalize(dt):
```

```
    df = dt.copy()
    for col in df.columns:
        df[col] = (df[col]-df[col].mean())/df[col].std()
    return df

#normalized
```

influence the result more due to its larger value. But this doesn't necessarily mean it is more important as a predictor. So we normalize the data to bring all the variables to the same range

In [174…

```
normalized_data2 = normalize(data2)
```

In [175…

```
normalized_data2['x0'] = 1
```

In [176…

```
data_val = normalized_data2.values
m =  len(data_val[:-1])
X = normalized_data2[['x0','Size of the house (in square feet)','Number of bedrooms']].iloc[:-1].values
y = normalized_data2['Price of the house'][:-1].values.reshape(m,1)
n = X.shape[1]
theta = np.zeros((n,1))

m, X.shape, y.shape, theta.shape
```

Out[176…

```
(46, (46, 3), (46, 1), (3, 1))
```

In [177…

```
computeCost(X,y,theta)
```

Out[177…

```
0.9858408807221255
```

In [178…

```
thet, Jhist = gradientDescent(X,y,theta,0.01,1000)
```

In [179…

```
print(f"h(x) = {str(round(thet[0,0],4))} + {str(round(thet[1,0],4))}x1 + {str(round(thet[2,0],4))}x2")
```

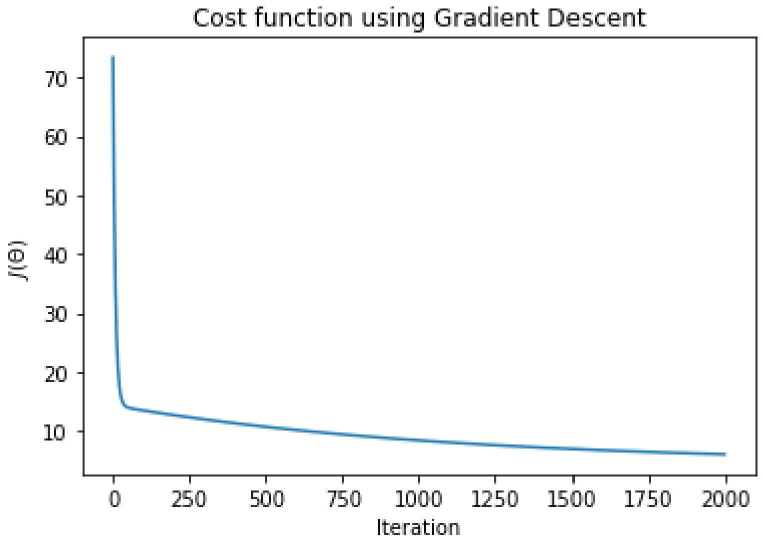
```
h(x) = -0.0014 + 0.8805x1 + -0.0478x2
```

In [180…

```
plt.plot(J_history)
plt.xlabel("Iteration")
plt.ylabel("$J(\Theta)$")
plt.title("Cost function using Gradient Descent")
```

Out[180…

```
Text(0.5, 1.0, 'Cost function using Gradient Descent')
```



In [181…

```
X_req = normalized_data2[['x0','Size of the house (in square feet)','Number of bedrooms']].iloc[-1].values
X_req
```

Out[181…

```
array([ 1.        , -1.00374794, -0.22367519])
```

In [182…

```
thet
```

Out[182…

```
array([[-0.00135533],
       [ 0.88054216],
       [-0.04783476]])
```

In [183…

```
data2.tail(1)
```

Out[183…

	Size of the house (in square feet)	Number of bedrooms	Price of the house
46	1203	3	239500

In [184…

```
normalized_data2.tail(1)
```

Out[184…

	Size of the house (in square feet)	Number of bedrooms	Price of the house	x0
46	-1.003748	-0.223675	-0.807044	1

In [185…

```
def find_val(pred):
    mul = data2["Price of the house"].std()
    ad = data2["Price of the house"].mean()
    return pred*mul+ad
```

In [186…

```
X_req = normalized_data2[['x0','Size of the house (in square feet)','Number of bedrooms']].iloc[-1].values
predict1 = predict(X_req, thet)
print(find_val(predict1))
```

```
231065.48439645045
```