Machine Learning
Home Work #1

Name : Jae Wook Lee
School : Dankook Uni
Major : Mobile System Engineering
ID : 32153492
E-mail : danny95ljw@gmail.com / 32153492@dankook.ac.kr
Phone : +82 10-2538-5030

1. Introduction

This report will be introducing about brief concept of ‘linear regression’ & ‘gradient descent algorithm’, ‘normal equation’. Afterwards it will be introducing the codewords and self thoughts about this assignment.
(This code will be introducing ‘linear regression’ by using ‘gradient descent’, ‘normal equation’.)

2. Concept

- Linear Regression

$$\boxed{\begin{array}{l}\text{Hypothesis:}\\ \quad h_\theta(x) = \theta_0 + \theta_1 x\end{array}}$$

The goal is to search for perfect theta0, theta1 that matches that dataset. If we can find good parameters (theta0, theta1), this hypothesis will be able to predict the value for that dataset.

- **Gradient Descent**

$$\boxed{\begin{array}{l}\text{Cost Function:}\\ \qquad J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2\end{array}}$$

Cost function is a function that can make us check the difference with our hypothesis and actual dataset value. And our goal is to make that Error value as small as possible.

$$\boxed{\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1)}$$

From hypothesis we will find theta0 and theta1 by using ‘Gradient Descent’ algorithm and this is how it looks like.

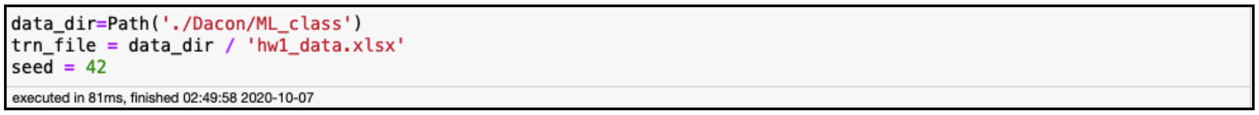
$$\boxed{\begin{array}{l}\theta_0 := \theta_0 - \alpha \frac{1}{m} \boxed{\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)}\\ \theta_1 := \theta_1 - \alpha \frac{1}{m} \boxed{\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)} \cdot x^{(i)}\end{array}}$$

Those partial derivative can be reshaped by using sigma.

2. Code Works

Coding environment
This code works at Jupiter Notebook and need to import certain libraries such as 'Matplotlib, numpy, path lib, seaborn, mpl_toolkits.mplot3d'. These are already shown at attached code file. **(My data set was from website so it is in 'xlsx'. Download the dataset from website directly)**

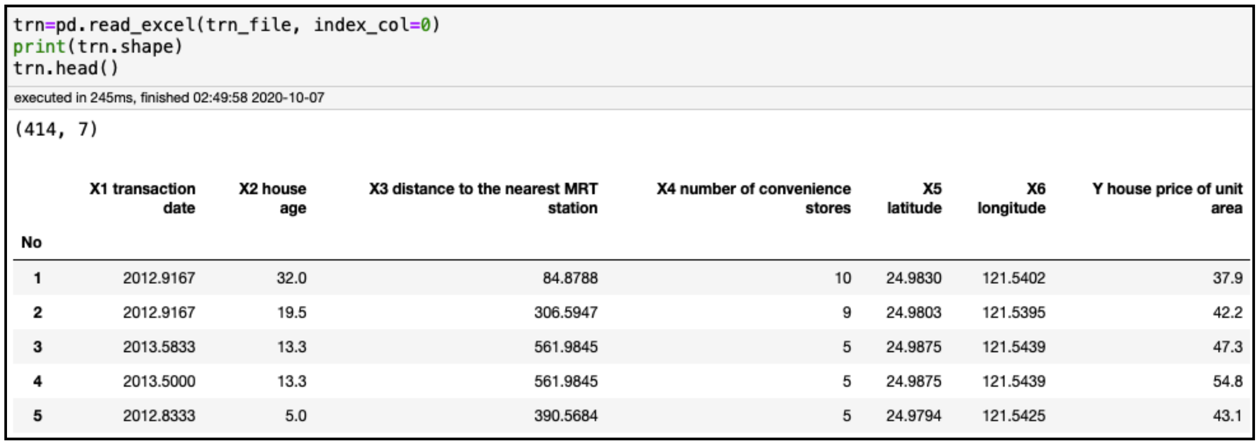
Task 1



```
data_dir=Path('./Dacon/ML_class')
trn_file = data_dir / 'hw1_data.xlsx'
seed = 42
```

```
executed in 81ms, finished 02:49:58 2020-10-07
```

Calling the files according to where the xlsx dataset file is located in my computer.



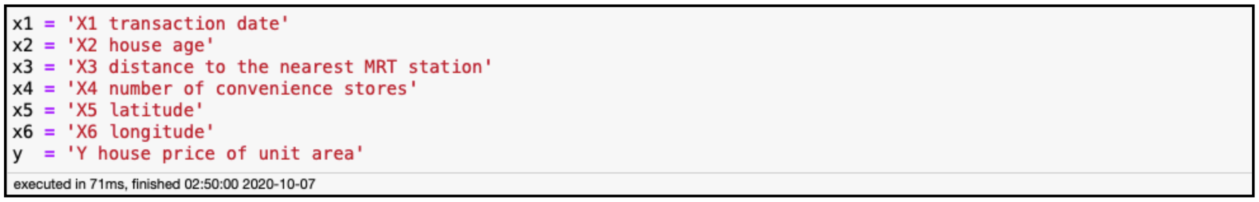
```
trn=pd.read_excel(trn_file, index_col=0)
print(trn.shape)
trn.head()
```

```
executed in 245ms, finished 02:49:58 2020-10-07
```

```
(414, 7)
```

	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
No							
1	2012.9167	32.0	84.8788	10	24.9830	121.5402	37.9
2	2012.9167	19.5	306.5947	9	24.9803	121.5395	42.2
3	2013.5833	13.3	561.9845	5	24.9875	121.5439	47.3
4	2013.5000	13.3	561.9845	5	24.9875	121.5439	54.8
5	2012.8333	5.0	390.5684	5	24.9794	121.5425	43.1

Checking how the dataset looks like.



```
x1 = 'X1 transaction date'
x2 = 'X2 house age'
x3 = 'X3 distance to the nearest MRT station'
x4 = 'X4 number of convenience stores'
x5 = 'X5 latitude'
x6 = 'X6 longitude'
y  = 'Y house price of unit area'
```

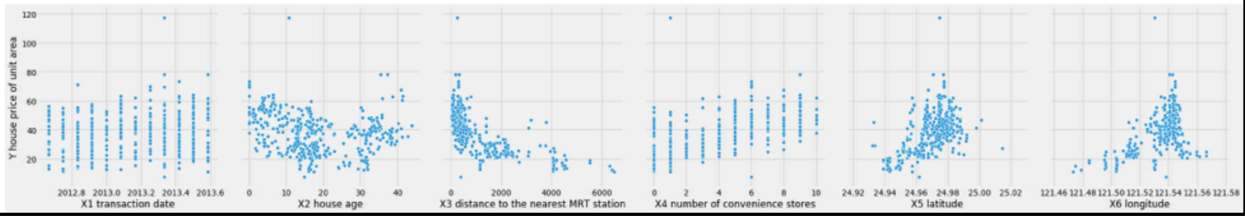
```
executed in 71ms, finished 02:50:00 2020-10-07
```

Made the name of the feature in easy way that I can code without a lot of effort.

```
sns.pairplot(x_vars= [x1,x2,x3,x4,x5,x6], y_vars=y, size=5,data = trn)
```

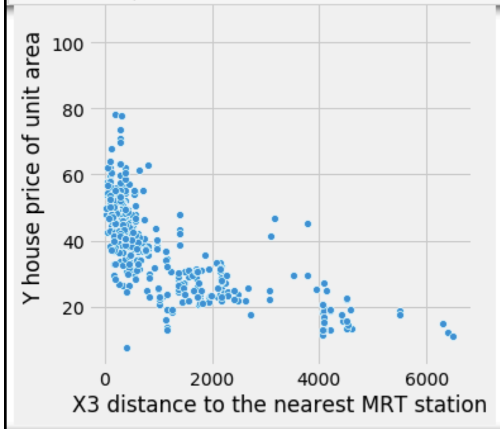
executed in 1.01s, finished 02:50:14 2020-10-07

```
<seaborn.axisgrid.PairGrid at 0x7f7fee102fd0>
```



```
sns.pairplot(x_vars = x3, y_vars = y,
             size = 5, data = trn)
```

executed in 225ms, finished 02:50:14 2020-10-07



EDA process by using seaborn library. I have draw the dataset and found out it have a shape of exponential function (1/n)^m. So I have come with some solution 1) make exponential into log form which make the dataset scattered than before. 2) normalizing by using ‘z-score’. But I think log is better because the idea came out more intuitively.
After applying log1p, the dataset looks more adequate to apply ‘linear regression’.

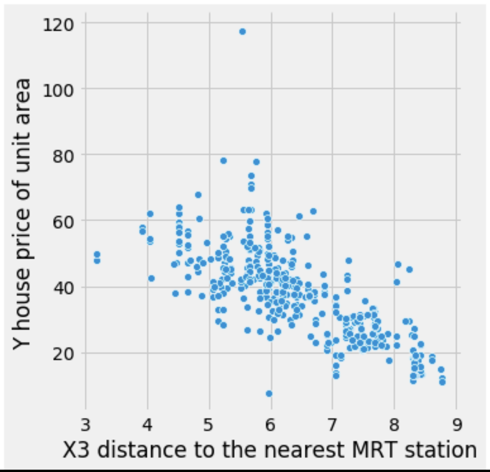
```
trn['X3 distance to the nearest MRT station']=trn['X3 distance to the nearest MRT station'].apply(np.log1p)
```

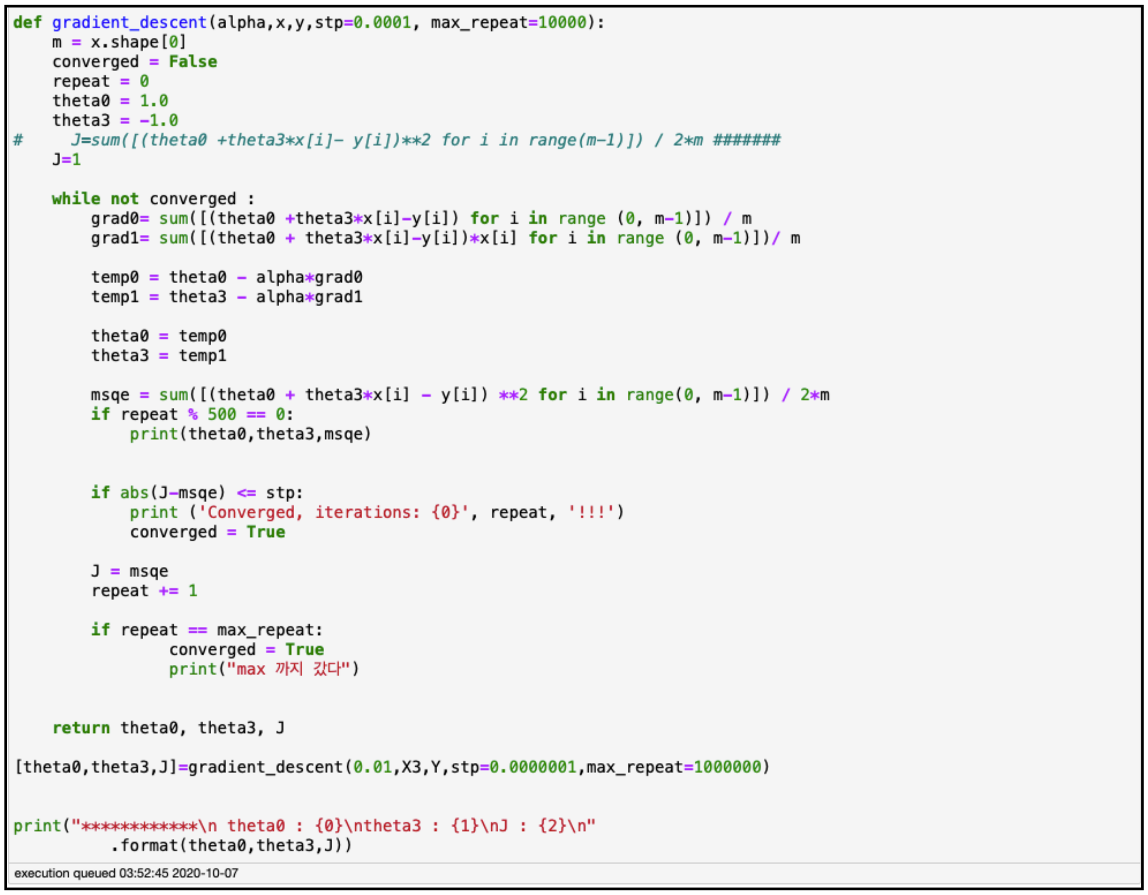
executed in 60ms, finished 02:50:14 2020-10-07

```
sns.pairplot(x_vars = x3, y_vars = y,
             size = 5, data = trn)
```

executed in 289ms, finished 02:50:15 2020-10-07

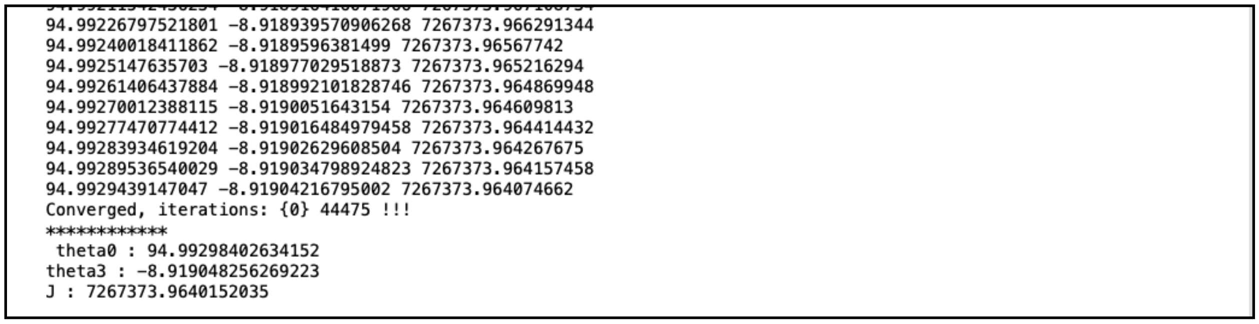
```
<seaborn.axisgrid.PairGrid at 0x7f7febf4f410>
```





```
def gradient_descent(alpha,x,y,stp=0.0001, max_repeat=10000):
    m = x.shape[0]
    converged = False
    repeat = 0
    theta0 = 1.0
    theta3 = -1.0
#     J=sum([(theta0 +theta3*x[i]- y[i])**2 for i in range(m-1)]) / 2*m #######
    J=1

    while not converged :
        grad0= sum([(theta0 +theta3*x[i]-y[i]) for i in range (0, m-1)]) / m
        grad1= sum([(theta0 + theta3*x[i]-y[i])*x[i] for i in range (0, m-1)])/ m

        temp0 = theta0 - alpha*grad0
        temp1 = theta3 - alpha*grad1

        theta0 = temp0
        theta3 = temp1

        msqe = sum([(theta0 + theta3*x[i] - y[i]) **2 for i in range(0, m-1)]) / 2*m
        if repeat % 500 == 0:
            print(theta0,theta3,msqe)


        if abs(J-msqe) <= stp:
            print ('Converged, iterations: {0}', repeat, '!!!')
            converged = True

        J = msqe
        repeat += 1

        if repeat == max_repeat:
                converged = True
                print("max 까지 갔다")


    return theta0, theta3, J

[theta0,theta3,J]=gradient_descent(0.01,X3,Y,stp=0.0000001,max_repeat=1000000)


print("************\n theta0 : {0}\ntheta3 : {1}\nJ : {2}\n"
          .format(theta0,theta3,J))
```

execution queued 03:52:45 2020-10-07

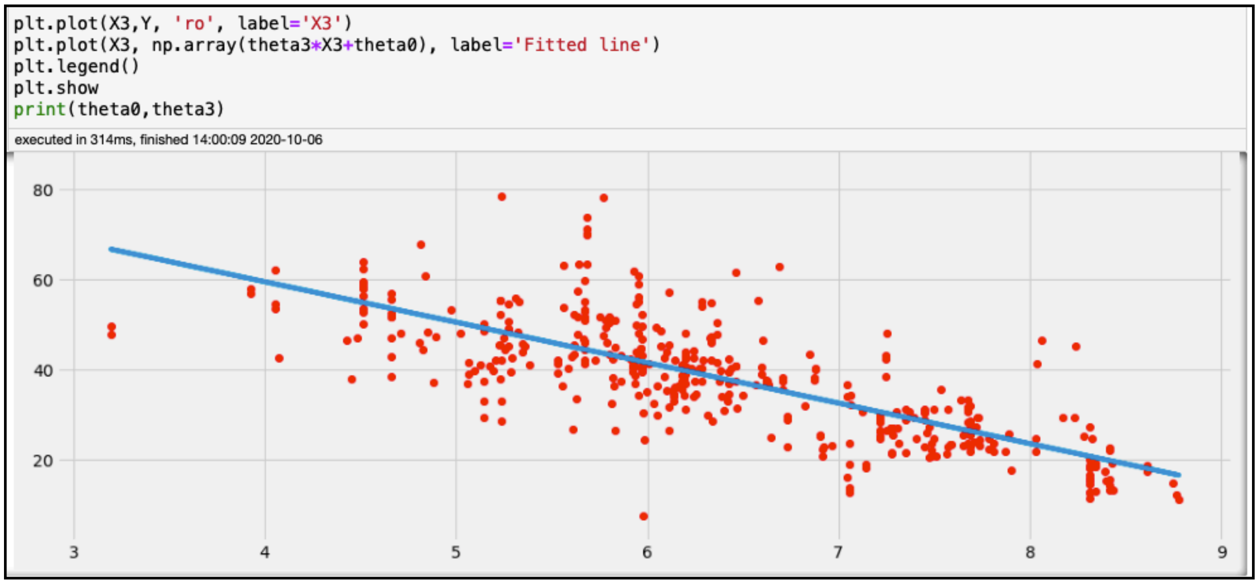


```
94.99226797521801 -8.918939570906268 7267373.966291344
94.99240018411862 -8.9189596381499 7267373.96567742
94.9925147635703 -8.918977029518873 7267373.965216294
94.99261406437884 -8.918992101828746 7267373.964869948
94.99270012388115 -8.9190051643154 7267373.964609813
94.99277470774412 -8.919016484979458 7267373.964414432
94.99283934619204 -8.91902629608504 7267373.964267675
94.99289536540029 -8.919034798924823 7267373.964157458
94.9929439147047 -8.91904216795002 7267373.964074662
Converged, iterations: {0} 44475 !!!
************
 theta0 : 94.99298402634152
theta3 : -8.919048256269223
J : 7267373.9640152035
```

Codes for ‘gradient descent’.

1) Parameters have been set randomly which will cause no problem because it will be rewritten.
2) Put the essential code at ‘while loop’ until value for cost function gets close to stop value.
3) The partial derivative can be written by using sigma after calculation.
4) We have to make sure that the parameters have to be updated simultaneously which I put the value at ‘temp0,1’ and updated later by ‘theta0,3=temp0,1’
5) When (J-msqe) gets converged to threshold (stp), break the while loop.
6) Even if the (J-msqe) don’t , it will stop by maximum iteration number (repeat).
7) Theta0 : 94.99298402634152
   Theta3 : -8.919048256269223
   J.   : 7267373.9640152035

(I have made to print out the result 500th values because otherwise the calculation is so slow)

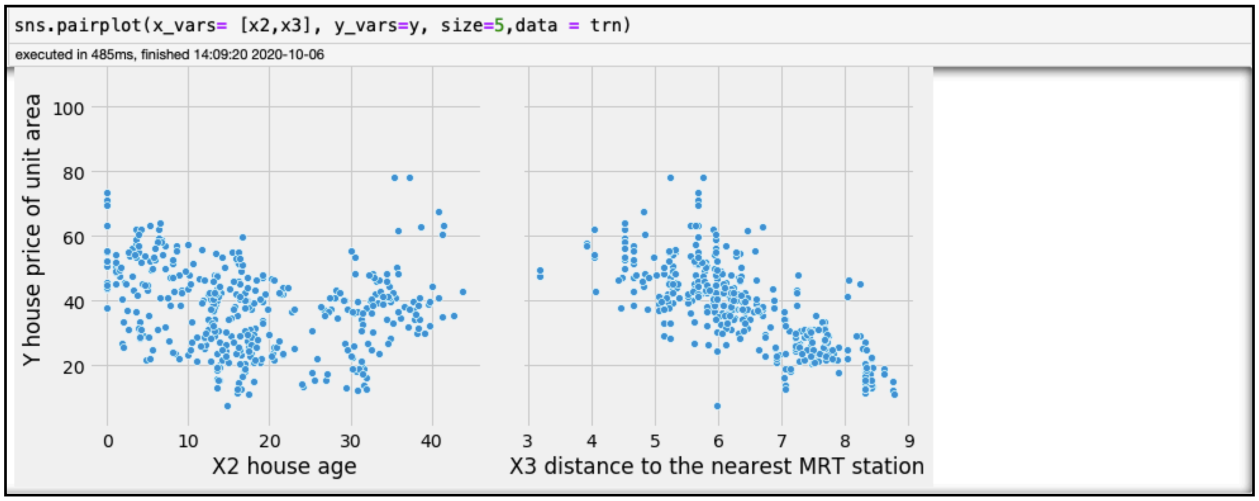


```
plt.plot(X3,Y, 'ro', label='X3')
plt.plot(X3, np.array(theta3*X3+theta0), label='Fitted line')
plt.legend()
plt.show
print(theta0,theta3)
```

executed in 314ms, finished 14:00:09 2020-10-06

Checking if value of parameter for hypothesis works out good in dataset.

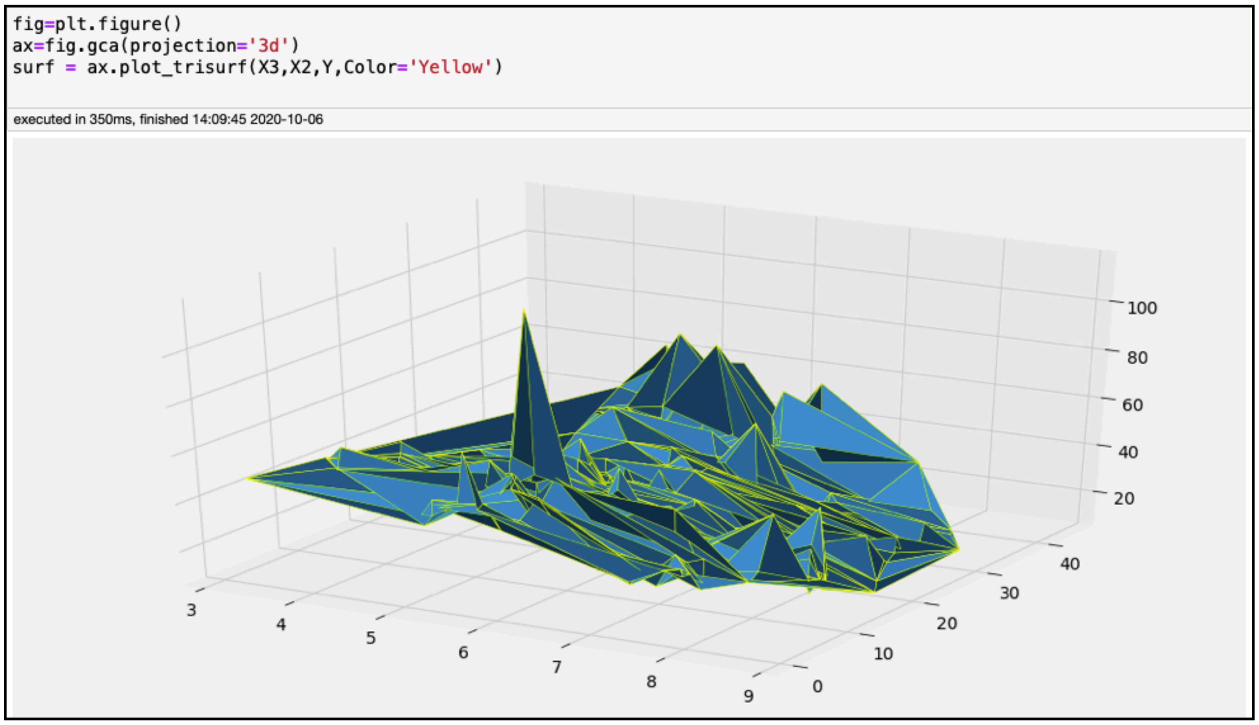
Task 2



```
sns.pairplot(x_vars= [x2,x3], y_vars=y, size=5,data = trn)
```

```
executed in 485ms, finished 14:09:20 2020-10-06
```

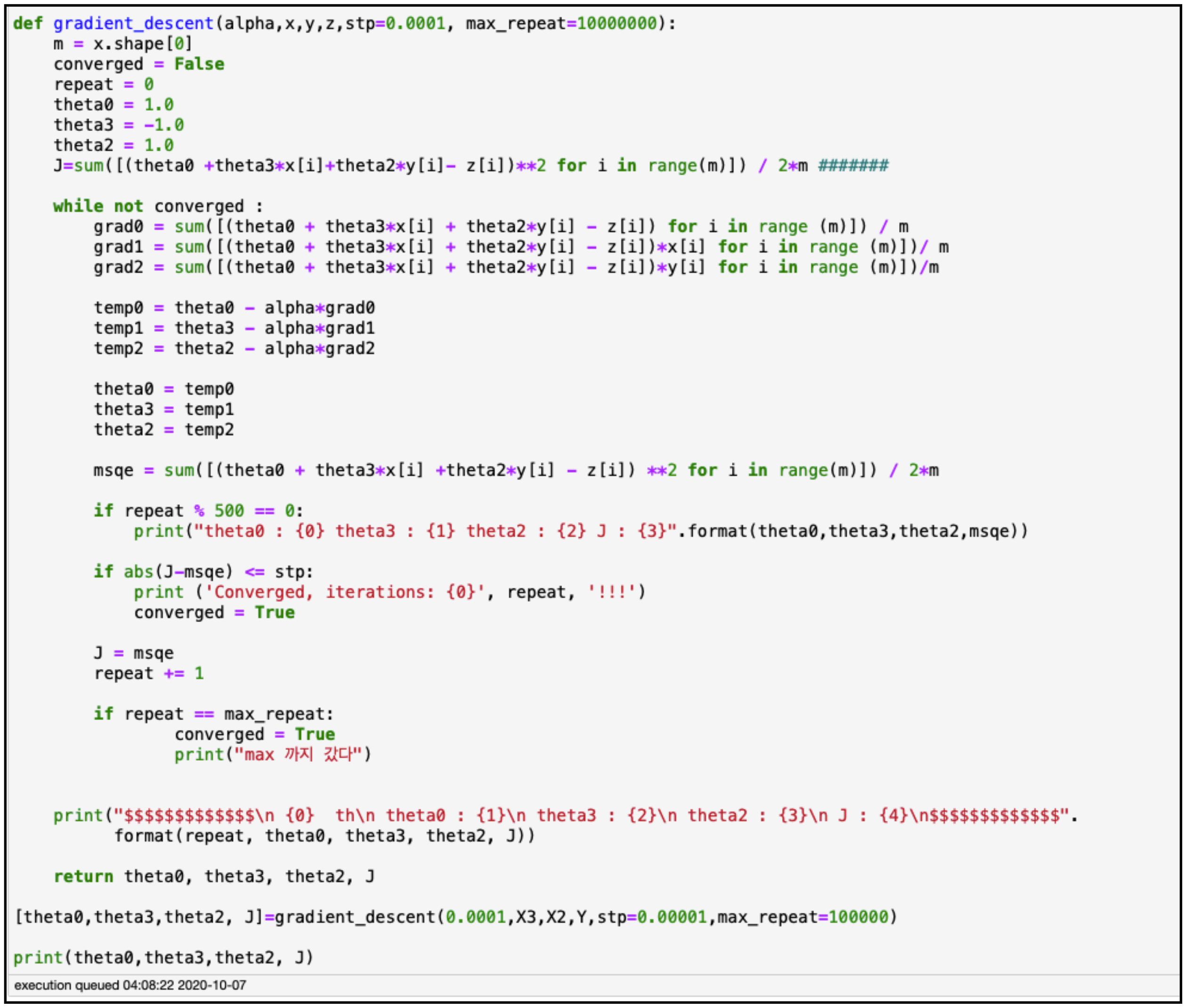
Y house price of unit area
100
80
60
40
20
0
10
20
30
40
X2 house age
3
4
5
6
7
8
9
X3 distance to the nearest MRT station



```
fig=plt.figure()
ax=fig.gca(projection='3d')
surf = ax.plot_trisurf(X3,X2,Y,Color='Yellow')
```

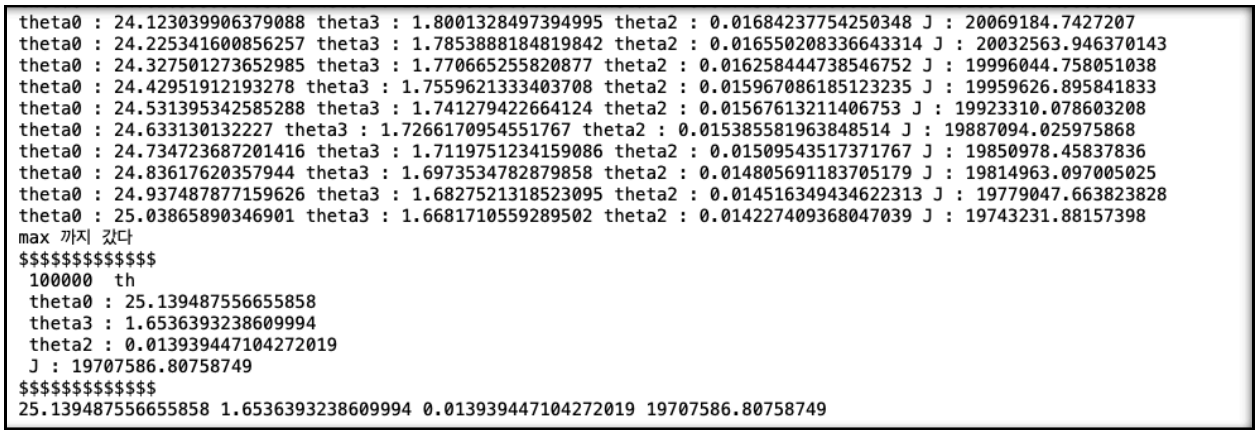
```
executed in 350ms, finished 14:09:45 2020-10-06
```

Since we will be having two independent value, I have plotted 3-D graph.



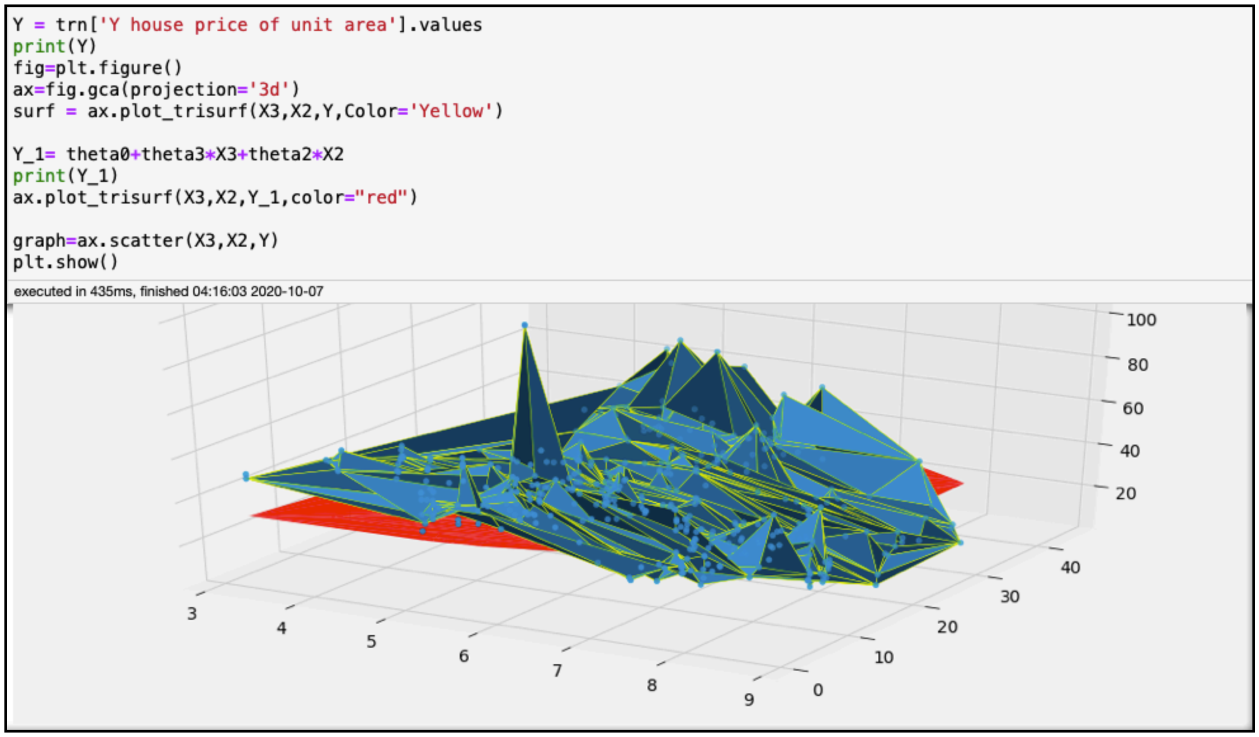
```python
def gradient_descent(alpha,x,y,z,stp=0.0001, max_repeat=10000000):
    m = x.shape[0]
    converged = False
    repeat = 0
    theta0 = 1.0
    theta3 = -1.0
    theta2 = 1.0
    J=sum([(theta0 +theta3*x[i]+theta2*y[i]- z[i])**2 for i in range(m)]) / 2*m #######

    while not converged :
        grad0 = sum([(theta0 + theta3*x[i] + theta2*y[i] - z[i]) for i in range (m)]) / m
        grad1 = sum([(theta0 + theta3*x[i] + theta2*y[i] - z[i])*x[i] for i in range (m)])/ m
        grad2 = sum([(theta0 + theta3*x[i] + theta2*y[i] - z[i])*y[i] for i in range (m)])/m

        temp0 = theta0 - alpha*grad0
        temp1 = theta3 - alpha*grad1
        temp2 = theta2 - alpha*grad2

        theta0 = temp0
        theta3 = temp1
        theta2 = temp2

        msqe = sum([(theta0 + theta3*x[i] +theta2*y[i] - z[i]) **2 for i in range(m)]) / 2*m

        if repeat % 500 == 0:
            print("theta0 : {0} theta3 : {1} theta2 : {2} J : {3}".format(theta0,theta3,theta2,msqe))

        if abs(J-msqe) <= stp:
            print ('Converged, iterations: {0}', repeat, '!!!')
            converged = True

        J = msqe
        repeat += 1

        if repeat == max_repeat:
                converged = True
                print("max 까지 갔다")


    print("$$$$$$$$$$$$$\n {0}  th\n theta0 : {1}\n theta3 : {2}\n theta2 : {3}\n J : {4}\n$$$$$$$$$$$$$".
         format(repeat, theta0, theta3, theta2, J))

    return theta0, theta3, theta2, J

[theta0,theta3,theta2, J]=gradient_descent(0.0001,X3,X2,Y,stp=0.00001,max_repeat=100000)

print(theta0,theta3,theta2, J)
```

execution queued 04:08:22 2020-10-07



```
theta0 : 24.123039906379088 theta3 : 1.8001328497394995 theta2 : 0.016842377754250348 J : 20069184.7427207
theta0 : 24.225341600856257 theta3 : 1.7853888184819842 theta2 : 0.016550208336643314 J : 20032563.946370143
theta0 : 24.327501273652985 theta3 : 1.770665255820877 theta2 : 0.016258444738546752 J : 19996044.758051038
theta0 : 24.42951912193278 theta3 : 1.7559621333403708 theta2 : 0.015967086185123235 J : 19959626.895841833
theta0 : 24.531395342585288 theta3 : 1.741279422664124 theta2 : 0.01567613211406753 J : 19923310.078603208
theta0 : 24.633130132227 theta3 : 1.7266170954551767 theta2 : 0.015385581963848514 J : 19887094.025975868
theta0 : 24.734723687201416 theta3 : 1.7119751234159086 theta2 : 0.015095435173717767 J : 19850978.45837836
theta0 : 24.83617620357944 theta3 : 1.6973534782879858 theta2 : 0.014805691183705179 J : 19814963.097005025
theta0 : 24.937487877159626 theta3 : 1.6827521318523095 theta2 : 0.014516349434622313 J : 19779047.663823828
theta0 : 25.03865890346901 theta3 : 1.6681710559289502 theta2 : 0.014227409368047039 J : 19743231.88157398
max 까지 갔다
$$$$$$$$$$$$$
 100000  th
 theta0 : 25.139487556655858
 theta3 : 1.6536393238609994
 theta2 : 0.013939447104272019
 J : 19707586.80758749
$$$$$$$$$$$$$
25.139487556655858 1.6536393238609994 0.013939447104272019 19707586.80758749
```

The code is nothing different compare with previous code. I just added X2 and theta2.
Theta0 : 25.139487556655858
Theta3 : 1.6536393238609994
Theta2 : 0.013939447104272019
J.   : 19707586.80758749 (we can see the cost function decrease correctly)
(I have made to print out the result 500th values because otherwise the calculation is so slow)



```
Y = trn['Y house price of unit area'].values
print(Y)
fig=plt.figure()
ax=fig.gca(projection='3d')
surf = ax.plot_trisurf(X3,X2,Y,Color='Yellow')

Y_1= theta0+theta3*X3+theta2*X2
print(Y_1)
ax.plot_trisurf(X3,X2,Y_1,color="red")

graph=ax.scatter(X3,X2,Y)
plt.show()
```

executed in 435ms, finished 04:16:03 2020-10-07

Plot the surface of the hypothesis and compare with the dataset, which makes my hypothesis is correct.

3. Problem & Solution

While implementing this code, there were a lot of troubles for me.
First of all, when I coded without scaling X3 by log1p, the parameter value was diverging. At that moment I didn't knew what was going on, but find out it can be handled by "reducing learning rate" or change the X3 by 'log1p' or 'normalization'.
But, according to my research, there are two kinds of normalization; 'min-max normalization' and 'z-score normalization'. But the thing is 'min-max normalization' can be bad when the data set have outlier. On the other hand, 'z-score normalization' can handle that outlier problem.
Second, I thought the value of cost function should be small, which made me very annoying. But finds out all we have to check is if the cost value is decreasing properly (J-msqe). This part was the hardest part for me and handled it by asking to colleagues.
Third, for task 2 I was very confused about how to plot the hypothesis (H=t0+t2*x2+t3*x3). But it was handled by making into 3-dimension graph.

4. Personal Thinks

It was very interesting to code linear regression by using gradient descent algorithm. I thought doing this assignment would take short time but got delayed a lot because of the cost function. I knew the concept that cost function is a value related (hypothesis - dataset value) but didn't knew how big the value was and it decrease as iteration goes on. Also, I think this assignment can be very difficult for those of who never code about linear regression in actual dataset.