20152410 배형준 머신러닝 과제3

In [1]:

```
# library import
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
```

1. Input points

load csv file using read_csv in pandas and make plot of data points

In [2]:

```
# set my local working directory

import os

directory = 'C:₩₩Users₩₩golds₩₩Desktop₩₩중앙대학교₩₩2020-1 4학년 1학기₩₩머신러닝'
os.chdir(directory)
```

In [3]:

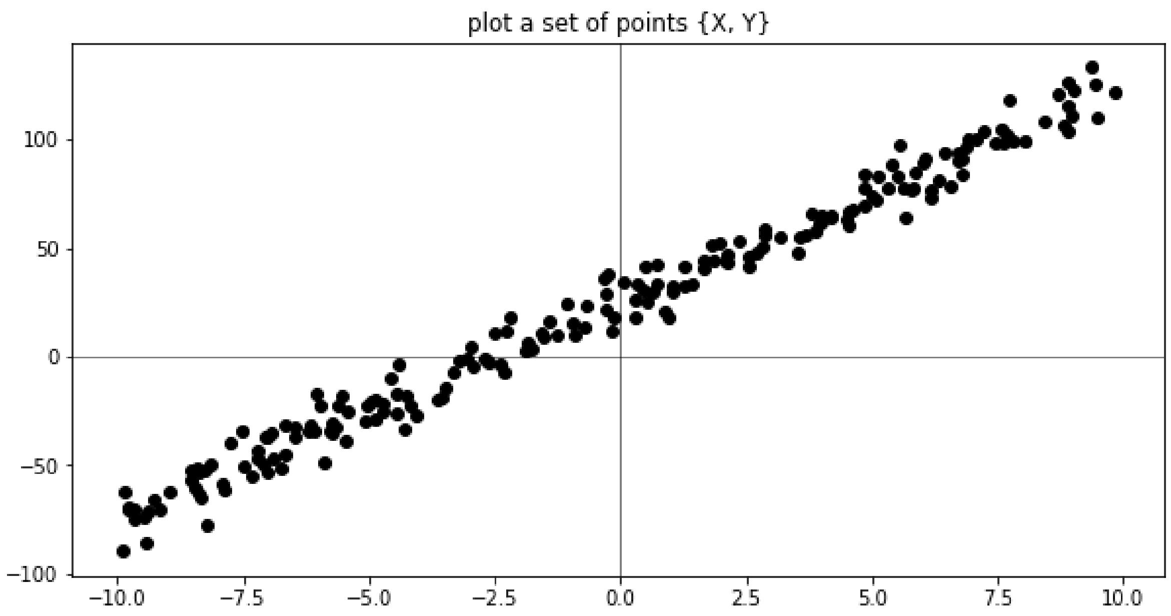
```
file_directory = './과제3/data.csv'
data = pd.read_csv(file_directory, header=None)
data.columns = ['X', 'Y']
X = data.iloc[:, 0]
Y = data.iloc[:, 1]
data.head()
```

Out[3]:

	X	Y
0	-5.518410	-18.426116
1	7.063695	100.441003
2	-8.515615	-57.190307
3	-4.962353	-20.424181
4	-4.435724	-26.448666

In [4]:

```
plt.figure(figsize=(10, 5))
plt.plot(X, Y, 'ko') # scatter plot
plt.axhline(y=0, color='k', linewidth=0.5) # x axis
plt.axvline(x=0, color='k', linewidth=0.5) # y axis
plt.title('plot a set of points {X, Y}')
plt.show()
```



2. Linear regression result

use my user definition function make_regression which was made at assignment 2

In [5]:

```
def make_regression(X, Y):

    # set random initial condition of parameters

    m = len(Y)

    initial_theta_0 = np.random.randn(1)
    initial_theta_1 = np.random.randn(1)
    list_theta_0 = [initial_theta_0]
    list_theta_1 = [initial_theta_1]

    initial_loss = np.sum((initial_theta_0 + initial_theta_1 * X - Y)**2) / (2*m)
    list_loss = [initial_loss]

    temp_theta_0 = initial_theta_0
    temp_theta_1 = initial_theta_1
    temp_loss = initial_loss

    learning_rate = 0.01
    error_bound = 0.00001

    # model learning

    while True:
        # calculate gradient
        gradient_theta_0 = np.sum(temp_theta_0 + temp_theta_1 * X - Y) / m
        gradient_theta_1 = np.sum((temp_theta_0 + temp_theta_1 * X - Y) * X) / m

        # renew the parameters
        next_theta_0 = temp_theta_0 - learning_rate * gradient_theta_0
        next_theta_1 = temp_theta_1 - learning_rate * gradient_theta_1

        temp_theta_0 = next_theta_0
        temp_theta_1 = next_theta_1

        # calculate loss to evaluate the parameters
        next_loss = np.sum((next_theta_0 + next_theta_1 * X - Y)**2) / (2*m)

        # store results
        list_theta_0.append(next_theta_0)
        list_theta_1.append(next_theta_1)
        list_loss.append(next_loss)

        # stopping rule
        if temp_loss - next_loss < error_bound:
            break

        temp_loss = next_loss

    theta_0_hat = list_theta_0[-1]
    theta_1_hat = list_theta_1[-1]

    return theta_0_hat, theta_1_hat, list_theta_0, list_theta_1, list_loss
```

In [6]:

```
theta_0_hat, theta_1_hat, list_theta_0, list_theta_1, list_loss = make_regression(X, Y)
```

In [7]:

```
print(' My linear regression model estimates the paramaters as follows. ₩n theta_0 : {}, theta_1 : {
      .format(theta_0_hat, theta_1_hat))
```

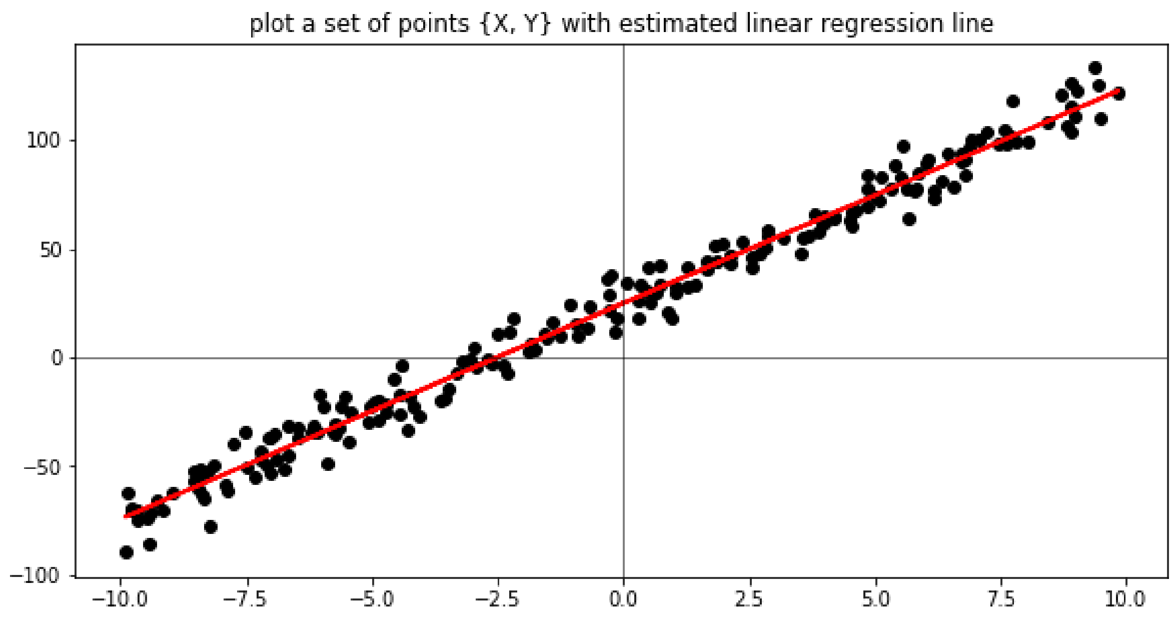
```
 My linear regression model estimates the paramaters as follows.
 theta_0 : [24.87599216], theta_1 : [9.93415201]
```

In [8]:

```
# get fitted values for making plot

Y_hat = theta_0_hat + theta_1_hat * X
```

In [9]:

```
plt.figure(figsize=(10, 5))
plt.plot(X, Y, 'ko') # scatter plot
plt.plot(X, Y_hat, 'r', linestyle='-', linewidth=2) # estimated linear regression line
plt.axhline(y=0, color='k', linewidth=0.5) # x axis
plt.axvline(x=0, color='k', linewidth=0.5) # y axis
plt.title('plot a set of points {X, Y} with estimated linear regression line')
plt.show()
```



3. Plot the energy surface

In [10]:

```
# import library for 3d plot

from mpl_toolkits import mplot3d
```

In [11]:

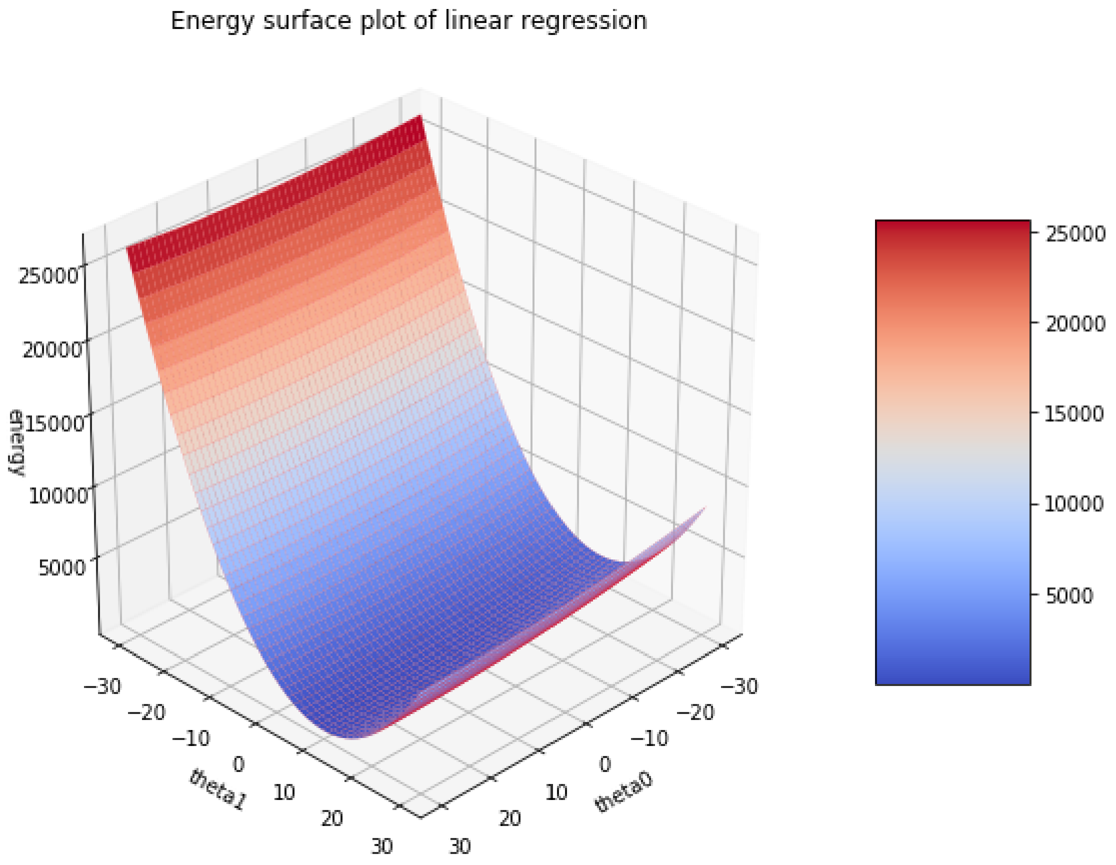
```
def make_energy_surface(theta0_arange, theta1_arange, X, Y):
    m = len(Y)
    theta0_axis, theta1_axis = np.meshgrid(theta0_arange, theta1_arange)
    shape = theta0_axis.shape
    energy = np.zeros(shape)

    for i in range(shape[0]):
        for j in range(shape[1]):
            energy[i, j] = np.sum((theta0_axis[i, j] + theta1_axis[i, j] * X - Y)**2) / (2*m)

    return theta0_axis, theta1_axis, energy
```

In [12]:

```
# make 2 axis and calculate energy

theta0_arange = np.arange(-30, 30, 0.1)
theta1_arange = np.arange(-30, 30, 0.1)

theta0_ax, theta1_ax, energy = make_energy_surface(theta0_arange, theta1_arange, X, Y)
```

In [13]:

```
# 태어나서 처음으로 3d plot 출력하는 중

fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(theta0_ax, theta1_ax, energy, cmap='coolwarm', linewidth=0.5, antialiased=Tr
# antialiased=True : 투시를 좀 더 잘 되게 해서 그래프를 좀 더 부드럽게 만들어주는 역할인듯
wire = ax.plot_wireframe(theta0_ax, theta1_ax, energy ,color='r', linewidth=0.1) # energy 그래프의
fig.colorbar(surf, shrink=1, aspect=3) # shrink, aspect : colorbar 크기
fig.tight_layout() # 명시된 여백에 관련된 서브플랏 파라미터를 자동으로 조정

ax.view_init(azim=45) # 그래프의 카메라 각도 조절

ax.set_title('Energy surface plot of linear regression')
ax.set_xlabel('theta0')
ax.set_ylabel('theta1')
ax.set_zlabel('energy')

plt.show()
```



4. Plot the gradient descent path on the energy surface

energy surface plot with gradient descent learning curve

with following initial condition : $\theta_0^{(0)} = -30, \theta_1^{(0)} = -30$

In [14]:

```
def make_regression(X, Y):

    # set random initial condition of parameters

    m = len(Y)

    initial_theta_0 = -30 # adjust initial condition of theta_0
    initial_theta_1 = -30 # adjust initial condition of theta_1
    list_theta_0 = [initial_theta_0]
    list_theta_1 = [initial_theta_1]

    initial_loss = np.sum((initial_theta_0 + initial_theta_1 * X - Y)**2) / (2*m)
    list_loss = [initial_loss]

    temp_theta_0 = initial_theta_0
    temp_theta_1 = initial_theta_1
    temp_loss = initial_loss

    learning_rate = 0.01
    error_bound = 0.00001

    # model learning

    while True:
        # calculate gradient
        gradient_theta_0 = np.sum(temp_theta_0 + temp_theta_1 * X - Y) / m
        gradient_theta_1 = np.sum((temp_theta_0 + temp_theta_1 * X - Y) * X) / m

        # renew the parameters
        next_theta_0 = temp_theta_0 - learning_rate * gradient_theta_0
        next_theta_1 = temp_theta_1 - learning_rate * gradient_theta_1

        temp_theta_0 = next_theta_0
        temp_theta_1 = next_theta_1

        # calculate loss to evaluate the parameters
        next_loss = np.sum((next_theta_0 + next_theta_1 * X - Y)**2) / (2*m)

        # store results
        list_theta_0.append(next_theta_0)
        list_theta_1.append(next_theta_1)
        list_loss.append(next_loss)

        # stopping rule
        if temp_loss - next_loss < error_bound:
            break

        temp_loss = next_loss

    theta_0_hat = list_theta_0[-1]
    theta_1_hat = list_theta_1[-1]

    return theta_0_hat, theta_1_hat, list_theta_0, list_theta_1, list_loss
```

In [15]:

```
theta_0_hat, theta_1_hat, list_theta_0, list_theta_1, list_loss = make_regression(X, Y)
```

In [16]:

```
print(' My linear regression model estimates the paramaters as follows. ₩n theta_0 : {}, theta_1 : {
      .format(theta_0_hat, theta_1_hat))
```

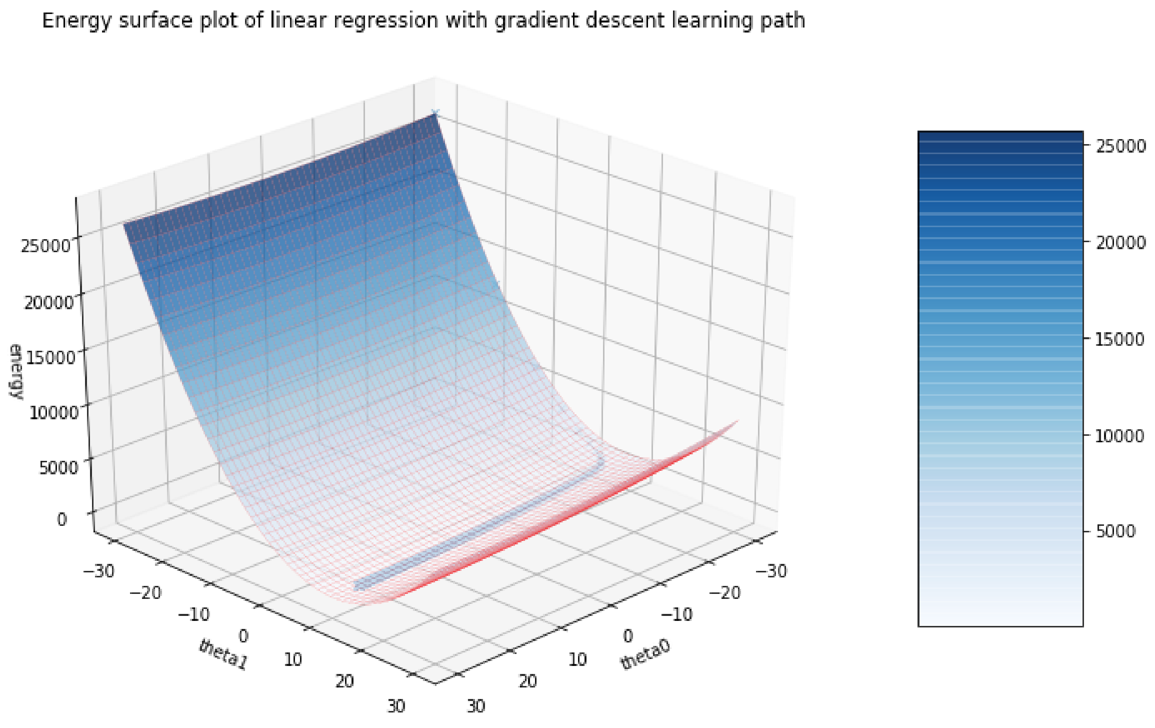
```
 My linear regression model estimates the paramaters as follows.
 theta_0 : 24.875786274544552, theta_1 : 9.934148838555593
```

In [17]:

```
array_theta_0 = np.array(list_theta_0).reshape(-1, 1)
array_theta_1 = np.array(list_theta_1).reshape(-1, 1)
array_loss = np.array(list_loss).reshape(-1, 1)
```

In [18]:

```
# 3d surface plot of energy with gradient descent learning path

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(theta0_ax, theta1_ax, energy, cmap='Blues', linewidth=0, alpha=0.75, antiali
    # alpha로 투명도 조절
wire = ax.plot_wireframe(theta0_ax, theta1_ax, energy ,color='r', linewidth=0.1)
fig.colorbar(surf, shrink=1, aspect=3)
fig.tight_layout()

ax.view_init(azim=45)

ax.set_title('Energy surface plot of linear regression with gradient descent learning path')
ax.set_xlabel('theta0')
ax.set_ylabel('theta1')
ax.set_zlabel('energy')

# 학습곡선의 path
ax.scatter(array_theta_0, array_theta_1, array_loss, '-k', marker='x', linewidth=1)

plt.show()
```


Energy surface plot of linear regression with gradient descent learning path
energy
theta1
theta0