1. Input data

- Configuration variable들을 정의하고 Input data를 정규분포를 따르는 랜덤값으로 생성했습니다.

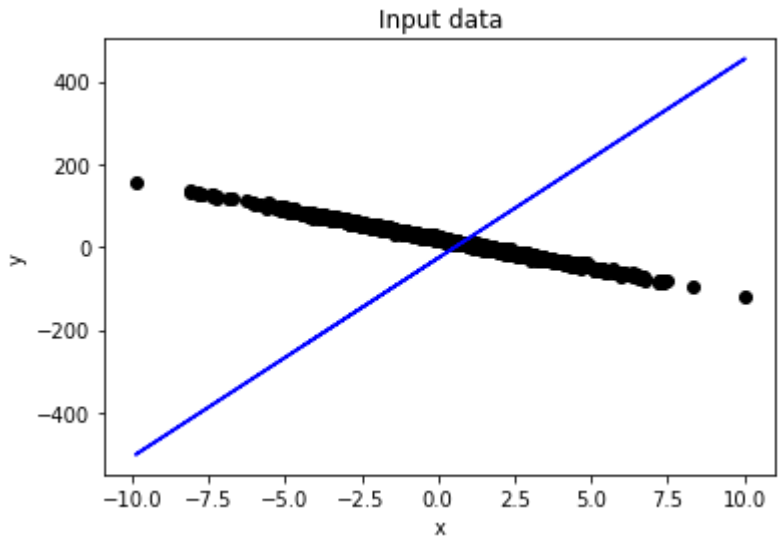
dataset의 크기는 1000, 찾고자하는 함수는 $f(x) = -14x + 20$ 이다. 또한 learning rate은 0.001로 뒀습 random.normal을 통해 정규분포가 $N(0, 3)$ 의 분포를 따르도록 x값과 n을 정의했습니다.

- 학습시킬 linear model을 초기화하고 input data와 linear model을 시각화했습니다.

linear model은 $(theta0, theta1) = (48, -27)$을 만족하는 $f(x) = 48x - 27$ 로 초기화하고 linear model과 랜덤값으로 생성한 x,y값을 시각화했습니다.

```
import numpy as np
import matplotlib.pyplot as plt
'''
1. Data
 - Config
 - Input Data

 2. Linear Model
 - Initialize linear model
 - Visualize data and linear model
'''

# Config
m=1000
b=20
a=-14
learning_rate=0.001
theta0, theta1=-27, 48

# Input data
x=np.random.normal(0, 3, size=m)
temp_y=a*x+b
n=np.random.normal(0, 3, size=m)
y=temp_y+n

# Initialize linear model
h=theta1*x+theta0

# Visualize data and linear model
plt.scatter(x, y, color='black')
plt.plot(x, h, color='blue')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Input data')
```

```
34 plt.title( 'Input data' )
35 plt.show()
```



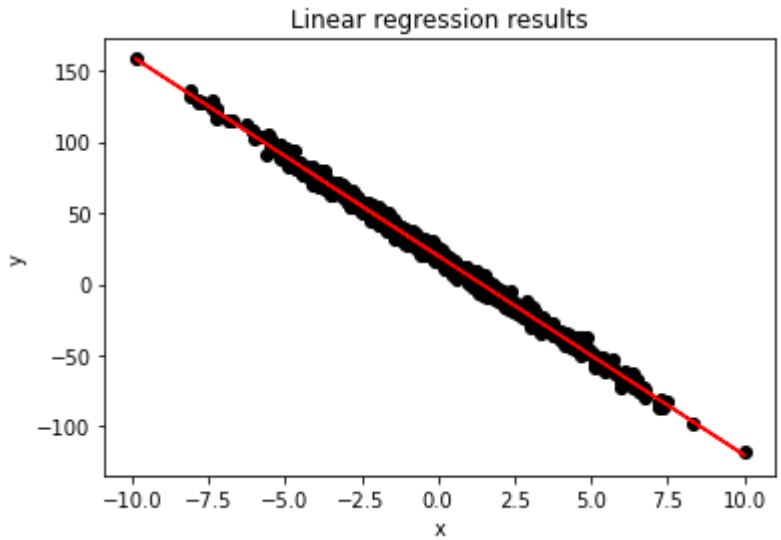
2. Output results

- 먼저 Objective function, gradient descent, derivation 들을 정의했습니다.
- Gradient descent방법으로 theta0와 theta1을 update시키면서 학습을 이어갑니다.
  이때 학습이 끝나는 조건은 두 값이 수렴하여 이전 theta값들과 upate후의 theta값들이 동일할때입니다.

```
'''
3. Objective function
 - Define objective function

4. Gradient Descent
 - Define derivations
 - Update theta0 and theta1
'''

# Define objective function
def objective_function() :
    J = np.sum((h-y)**2) / (2*m)
    return J

# Define derivations
def dev() :
    dev0=np.sum(h-y) / m
    dev1=np.sum((h-y)*x) / m
    return dev0, dev1

# Update theta0 and theta1
def gradient_descent() :
    return theta0-(learning_rate*dev0), theta1-(learning_rate*dev1)

J_list=[]
```

```
theta0_list=[]
theta1_list=[]
# Train
count=0
while(1) :
    count+=1
    J=objective_function()
    J_list.append(J)
    theta0_list.append(theta0)
    theta1_list.append(theta1)
    dev0, dev1 = dev()
    temp0, temp1 = theta0, theta1
    theta0, theta1 = gradient_descent()
    h=theta0+theta1*x
    if(temp0==theta0) and (temp1==theta1) :
        break
print("theta0 : ", theta0)
print("theta1 : ", theta1)
print("loss : ", J)
# Visualize data and linear model
plt.scatter(x, y, color='black')
plt.plot(x, h, color='red')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear regression results')
plt.show()
```

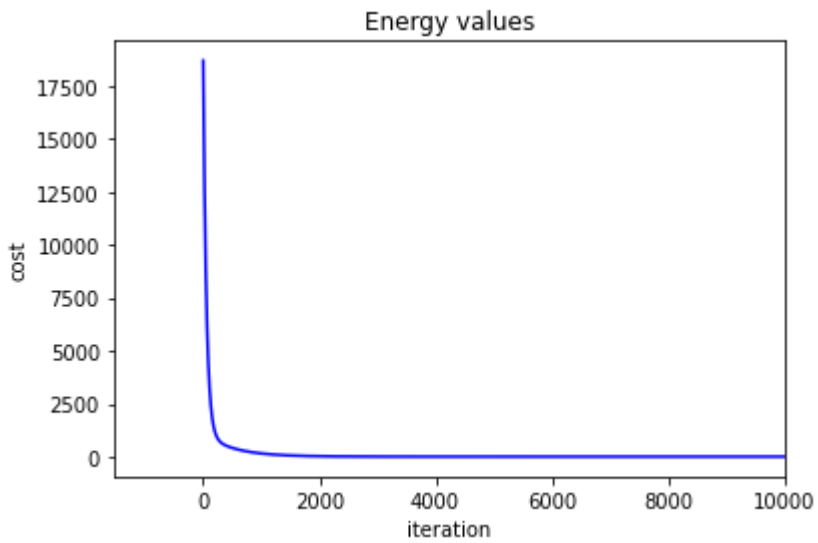
```
theta0 :  20.01254840580673
theta1 :  -14.001667149441015
loss :  4.300044723130642
```



3. Energy Values

- update를 할때마다 J_list에 그동안의 J(objective function의 value) 값들을 넣었습니다.
- 각 iteration마다 J값이 어떻게 변하는지 시각화했습니다.

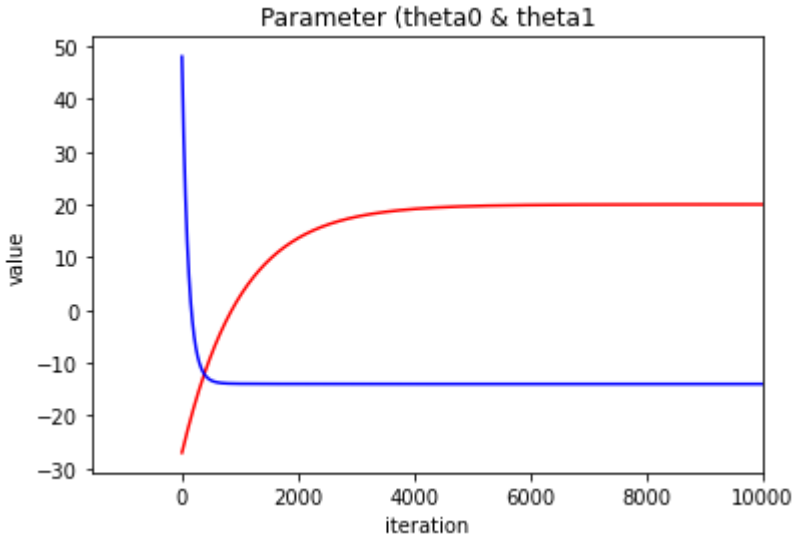
```
plt.plot(J_list, color='blue')
plt.xlabel("iteration")
plt.xlim(right=10000)
plt.ylabel("cost")
plt.title("Energy values")
plt.show()
```



4. Model parameters

- update를 할때마다 변하는 theta0과 theta1의 값들을 따로 리스트해 저장해뒀었습니다.
- 각 iteration마다 theta0과 theta1이 어떻게 변하는지 시각화했습니다.

```
plt.plot(theta0_list, color='red')
plt.plot(theta1_list, color='blue')
plt.xlabel("iteration")
plt.xlim(right=10000)
plt.ylabel("value")
plt.title("Parameter (theta0 & theta1")
plt.show()
```


Parameter (theta0 & theta1
50
40
30
20
10
0
−10
−20
−30
value
0
2000
4000
6000
8000
10000
iteration