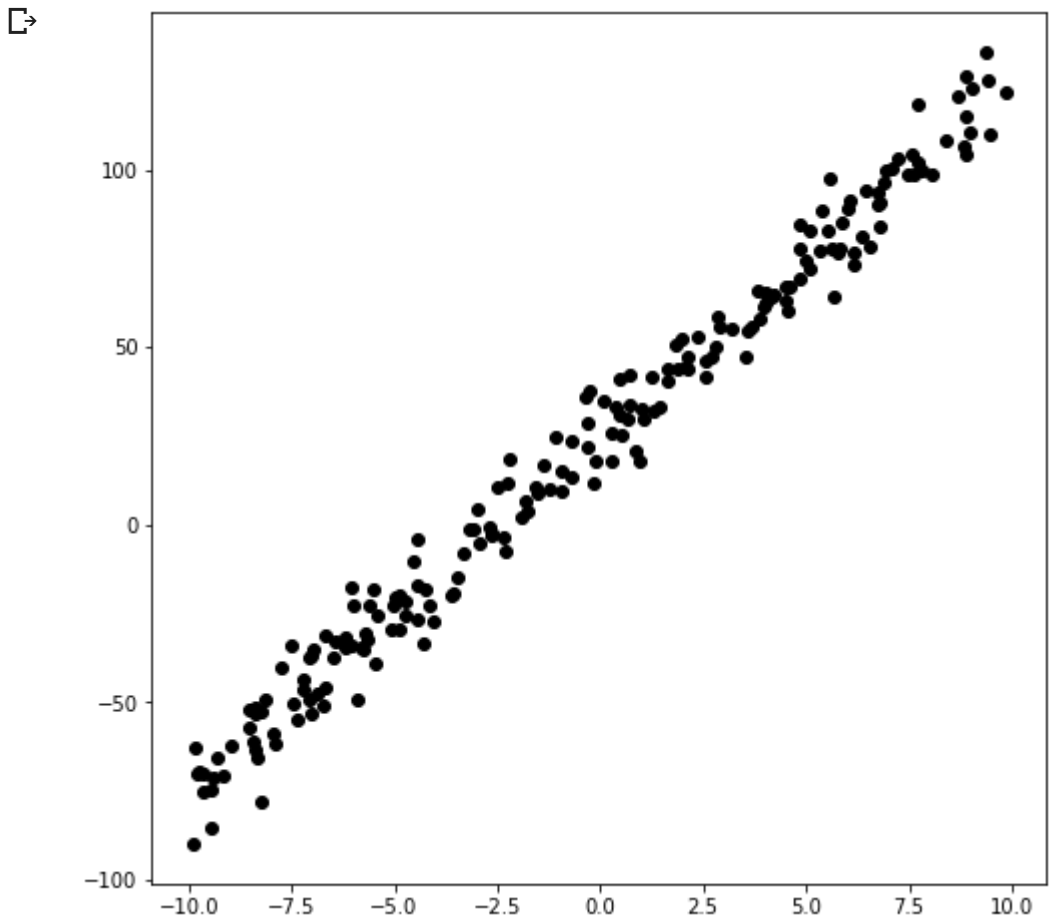
1. Load data

- data.csv 파일을 동일 디렉터리에 위치시키고 데이터를 읽었습니다.
- data.csv에서 읽은 data를 점의 형태로 시각화했습니다.

```
import numpy as np
import matplotlib.pyplot as plt

'''
After read the data from data.csv file, visualize data
'''

path = "drive/My Drive/Classroom/Machine Learning (2) 2020-1\
/class-MachineLearning/assignment03/data.csv"
data = np.genfromtxt(path, delimiter=',')

x_data = data[:, 0]
y_data = data[:, 1]

plt.figure(figsize=(8, 8))
plt.scatter(x_data, y_data, color='black')
plt.show()
```



2. Optimize linear function

- 주어진 x data와 y data를 표현할 수 있는 linear function을 regression 시킵니다.

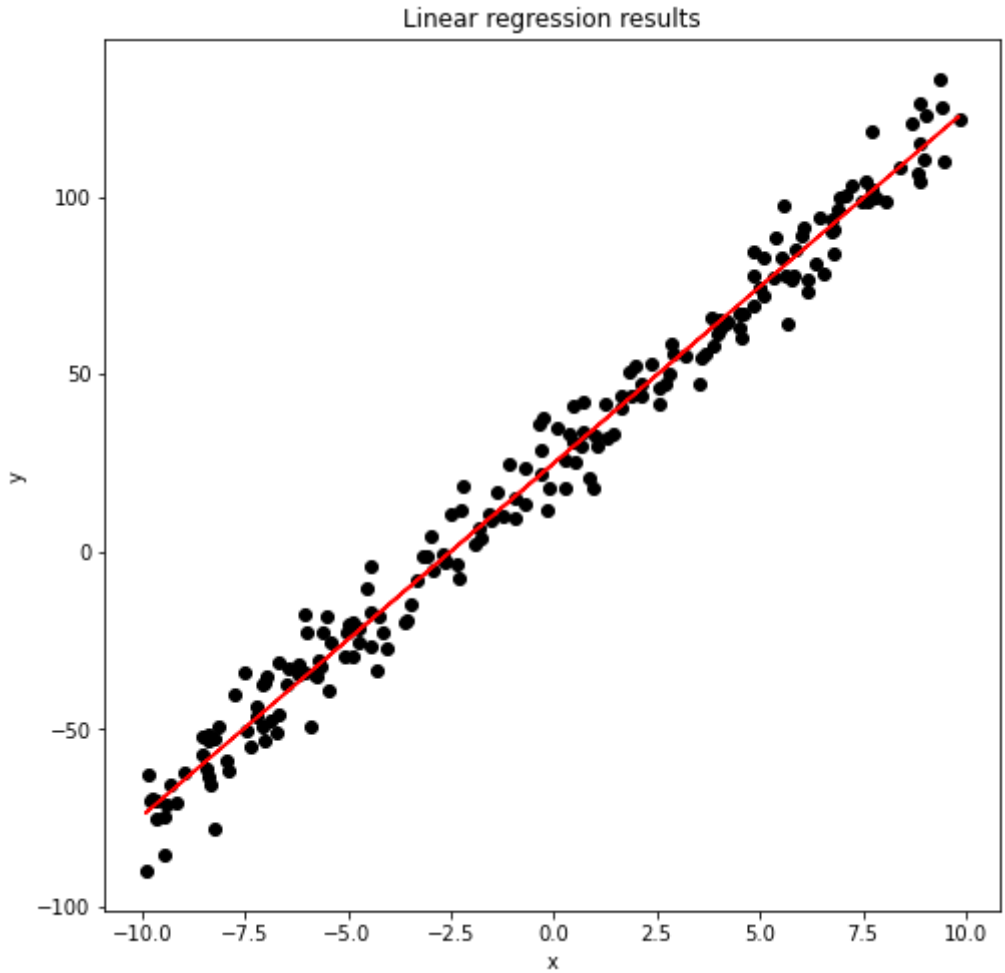
learning rate와 data size를 포함한 configuration들을 정의하고
objective function, gradient descent 등.. 의 함수를 train을 위해 정의했습니다.
optimize를 통해 얻어진 theta0와 theta1은 25와 10에 근사합니다.
즉, 찾고자했던 linear function은 $f(x) = 10x + 25$ 입니다.

- Optimize된 linear function과 input data를 시각화했습니다.

```
'''
Define
 - Config
 - objective function
 - derivation of objective function for each theta0 and theta1
 - gradient descent

Get Optimized theta0 and theta1 using gradient descent.
Visualize optimized linear function.
'''

#Config
learning_rate=0.001
m=len(x_data)
theta0, theta1=-30, -30

# Initialize linear model
h=theta1*x_data+theta0

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

# Define derivations
def dev() :
    dev0=np.sum(h-y_data) / m
    dev1=np.sum((h-y_data)*x_data) / 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_data
    if(temp0==theta0) and (temp1==theta1) :
        break
print("theta0 : ", theta0)
print("theta1 : ", theta1)
print("loss : ", J)
# Visualize data and linear model
plt.figure(figsize=(8, 8))
plt.scatter(x_data, y_data, color='black')
plt.plot(x_data, h, color='red')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear regression results')
plt.show()
```

```
theta0 :  24.90739329394621
theta1 :  9.934635539221306
loss :  27.467506725904556
```

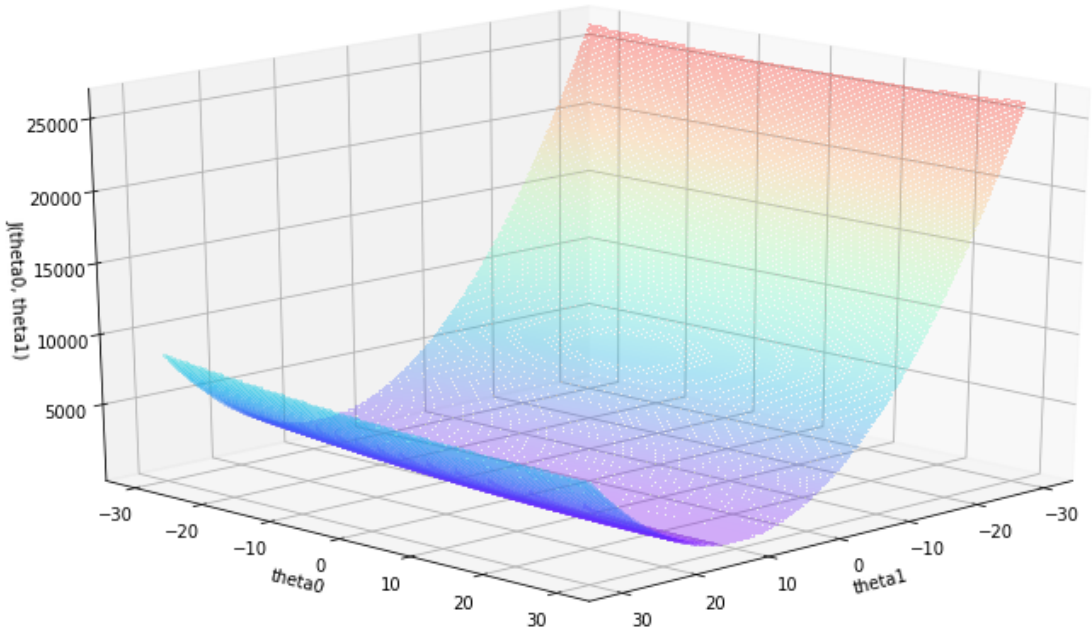


3. Visualize energy surface

- 각각의 theta0, theta1에 대한 J(loss)를 시각화했습니다.

그래프에서 theta0와 theta1에 대한 범위는 -30~30으로 설정하고
해당 범위 내에서 0.1씩 증가하면서 각각 601개의 값을 갖게됩니다.
meshgrid를 통해서 theta0와 theta1값들을 좌표평면에 대응시킬 수 있는 grid matrix로 만듭니다.
각 theta0, theta1 좌표에 따른 J(loss)값을 구하는 함수를 정의하고
matplotlib에서 제공하는 contourf3D함수를 통해서 energy surface를 시각화했습니다.

```
'''
Construct meshgrid field
Visualize loss at each theta0 and theta1 in 3-dimension
'''

import mpl_toolkits.mplot3d.axes3d as p3

def J(t0, t1) :
    h=x_data*t1 + t0
    J = np.sum((h-y_data)**2) / (2*m)
    return J
grid_theta0=np.linspace(start=-30, stop=30, num=601)
grid_theta1=np.linspace(start=-30, stop=30, num=601)
# Construct meshgrid between x and y
grid_theta0, grid_theta1=np.meshgrid(grid_theta0, grid_theta1)


fig = plt.figure(figsize=(13, 8))
ax = plt.axes(projection='3d')

ax.set_xlabel("theta1")
ax.set_ylabel("theta0")
ax.set_zlabel("J(theta0, theta1)")

grid_J=np.array([J(t1, t0) for (t1, t0) in \
                 zip(np.ravel(grid_theta1), np.ravel(grid_theta0))])
grid_J = grid_J.reshape(grid_theta0.shape)

#ax.plot_surface(grid_theta0, grid_theta1, grid_J, cmap=plt.cm.hot, alpha=0.6)
ax.contourf3D(grid_theta0, grid_theta1, grid_J, 200, cmap=plt.cm.rainbow, alpha=
#ax.plot(theta1_list,theta0_list, J_list,color = 'r', marker = '>', markerfacecc

#ax.plot([t for t in theta0_list], [t2 for t2 in theta1_list], cost , markerface
ax.view_init(20, 45)
plt.show()

```



4. Visualize gradient descent path

- 위에서 그린 energy surface위에 gradient descent path를 함께 시각화했습니다.

theta0와 theta1을 -30, -30으로 초기화하고 train과정에서 update되는 과정을 눈으로 확인하기 위해 update될때마다 각각 list안에 값을 저장하였습니다.
초기화한 값부터 최종 수렴한 값까지 각각의 (theta0, theta1, J) 좌표를 energy surface위에 plot하고 이를 화살표로 나타냄으로써 gradient descent되는 path를 확인할 수 있습니다.

```
'''
Visualize gradient descent path on energy surface
'''

fig = plt.figure(figsize=(13, 8))
ax = plt.axes(projection='3d')

ax.set_xlabel("theta1")
ax.set_ylabel("theta0")
ax.set_zlabel("J(theta0, theta1)")

grid_J=np.array([J(t1, t0) for (t1, t0) in \
                 zip(np.ravel(grid_theta1), np.ravel(grid_theta0))])
grid_J = grid_J.reshape(grid_theta0.shape)
```

```
15
16 #ax.plot_surface(grid_theta0, grid_theta1, grid_J, cmap=plt.cm.hot, alpha=0.6)
17 ax.contourf3D(grid_theta0, grid_theta1, grid_J, 200, cmap=plt.cm.rainbow, alpha=
18 ax.plot(theta1_list,theta0_list, J_list,color = 'black', marker = '>', \
19         markerfacecolor='r', markeredgecolor='r', markersize=5, alpha = .9)
20
21 #ax.plot([t for t in theta0_list], [t2 for t2 in theta1_list], cost , markerface
22 ax.view_init(20, 45)
23 plt.show()
```