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

parameters = {"axes.labelsize": 20, "axes.titlesize": 30, 'xtick.labelsize': 12, "ytick.labelsize": 12, "legend.fontsize": 12}
plt.rcParams.update(parameters)

'''파일 받아오기'''
M = pd.read_csv('C:₩₩Users₩₩kim07₩₩Downloads₩₩lin_regression_data_01.csv', header=None)   # 데이터파일 불러오기
M = M.to_numpy(dtype=float)
o_x_data = M[:, 0]
o_y_data = M[:, 1]
o_x_data = o_x_data.reshape(-1,1)
o_y_data = o_y_data.reshape(-1,1)



'''noise생성해 데이터 20배 더 생성해주는 함수'''
def make_some_noise(noise, M):     # M([:,2]) size
    n_data = []        #증강된 데이터 받을 array
    for i in range(len(M)):
        for j in range(10):                                                            #+방향으로 10개
```

```
            Nx = np.random.rand() * (noise) + M[i, 0]                          #x, y의 i
번째 데이터 기준 +방향으로 noise범위 안에 랜덤값 만들기
            Ny = np.random.rand() * (noise) + M[i, 1]
            n_data = np.append(n_data, Nx)
            n_data = np.append(n_data, Ny)
            n_data = np.reshape(n_data, [-1, 2])                               #size (:, 2)
로 정렬
        for k in range(10):                                                     #-방향으
로 10개
            Nx = np.random.rand() * (-noise) + M[i, 0]                         #x, y의 i
번째 데이터 기준 -방향으로 noise범위 안에 랜덤값 만들기
            Ny = np.random.rand() * (-noise) + M[i, 1]
            n_data = np.append(n_data, Nx)
            n_data = np.append(n_data, Ny)
            n_data = np.reshape(n_data, [-1, 2])                               #size (:, 2)
로 정렬
    return n_data




'''데이터 분할 함수(비율의 합은 10)'''
def data_division(n_data, Tr_rate, V_rate, Te_rate):
    np.random.shuffle(n_data)                                                   #데이터
섞기


    tr_index = int(len(n_data) * Tr_rate / 10)                                 #Tr_set 비율
만큼 데이터 index 양 확인
```

```
    v_index = int(len(n_data) * V_rate / 10)                                    #V_set 비
율만큼 데이터 index 양 확인
    te_index = int(len(n_data) * Te_rate / 10)                                  #Te_set 비
율만큼 데이터 index 양 확인

    #비율대로 data 나누기
    tr_set = n_data[0:tr_index]
    v_set = n_data[tr_index : tr_index + v_index]
    te_set = n_data[tr_index + v_index : tr_index + v_index + te_index]
    return tr_set, v_set, te_set


def PBSF_ANALYSTIC_SOLUTION(K, x, y):
    x_pbsf_matrix = x                                                          #
x_pbsf_matrix 1행 설정
    for i in range(2, K + 1):
        x_pbsf_matrix = np.column_stack([x_pbsf_matrix, x**i])               # Low방향으
로 k제곱 한것 쌓기

    x_dummy = np.ones([len(x_pbsf_matrix), 1])
    x_pbsf_matrix = np.column_stack([x_pbsf_matrix, x_dummy])                       #
dummy data 추가
    x_pbsf_matrix_T = np.transpose(x_pbsf_matrix)                                   #
transpose

    w = np.dot(np.dot(np.linalg.inv(np.dot(x_pbsf_matrix_T,
x_pbsf_matrix)),x_pbsf_matrix_T), y)     # analystic solution
```

```
    return w, x_pbsf_matrix


''' by gaussian BSF analystic solution 구하는 함수'''
def GBSF_ANALYSTIC_SOLUTION(K, x, y):
    x_gbsf_matrix = []


    k = np.arange(K)
#평균과 곱할 k 생성
    mu = np.min(x) + ((np.max(x) - np.min(x)) / (K - 1)) * k                    #평균 값 생성
    mu = np.reshape(mu, [len(mu), 1])
#평균 array size 조정
    sigma = (np.max(x) - np.min(x)) / (K - 1)                                    #
분산값


    #기저함수 구하기
    basis = np.zeros([len(x), K])
    for i in range(K):
        basis[:, i] = np.exp(-0.5 * ((x - mu[i]) / sigma) ** 2)




    x_gbsf_matrix = basis
    x_dummy = np.ones([len(x_gbsf_matrix), 1])
    x_gbsf_matrix = np.column_stack([x_gbsf_matrix, x_dummy])
# dummy data 추가


    x_gbsf_matrix_T = np.transpose(x_gbsf_matrix)
```

```
# transpose

    w = np.dot(np.dot(np.linalg.inv(np.dot(x_gbsf_matrix_T, x_gbsf_matrix)),x_gbsf_matrix_T), y)     # analystic solution
    return w, x_gbsf_matrix



'''MSE 구하는 함수'''
def get_MSE(y_hat, y):
    error = y_hat - y
    error = np.reshape(error, [-1, 1])

    MSE = np.mean(error ** 2)
    return MSE

noise = 1.2                                                                  #noise 범위 설정

n_data = make_some_noise(noise, M)
n_x_data = n_data[:, 0]
n_y_data = n_data[:, 1]
n_x_data = n_x_data.reshape(-1,1)
n_y_data = n_y_data.reshape(-1,1)


#과제 1
```

```
fig = plt.figure()
plt.scatter(n_x_data, n_y_data)
plt.scatter(o_x_data, o_y_data)
plt.legend(['n_data', 'o_data'])
plt.xlabel("weight(g)")
plt.ylabel("length(cm)")

#training , validation, test set 비율
Tr_rate = 8
V_rate = 0
Te_rate = 2


tr_set, v_set, te_set = data_division(n_data, Tr_rate, V_rate, Te_rate)          #데이터 나누기 함수 호출

tr_x = tr_set[:, 0]
tr_y = tr_set[:, 1]
v_x = v_set[:, 0]
v_y = v_set[:, 1]
te_x = te_set[:, 0]
te_y = te_set[:, 1]

#과제 2
fig = plt.figure()
```

```
plt.scatter(tr_x, tr_y)
plt.scatter(v_x, v_y)
plt.scatter(te_x, te_y)
plt.legend(['tr_set', 'v_set', 'te_set'])
plt.xlabel("weight(g)")
plt.ylabel("length(cm)")

#과제 3
K_list = np.arange(2, 52, 1)                                                        # basis 개수 설정

#K개 basis마다 저장할 MSE tr, te list 생성
MSE_gbsf_tr_list = []
MSE_gbsf_te_list = []
MSE_pbsf_tr_list = []
MSE_pbsf_te_list = []

for i in K_list:
    '''Gaussian basis function으로 구한 w와 MSE'''
    w_gbsf_tr_set, x_gbsf_tr_set = GBSF_ANALYSTIC_SOLUTION(i, tr_set[:, 0], tr_set[:, 1]) #basis 개수 k개일 때 tr_set의 weight, x값
    w_gbsf_te_set, x_gbsf_te_set = GBSF_ANALYSTIC_SOLUTION(i, te_set[:, 0], te_set[:, 1]) #te_set의 x데이터들얻기 위함


    y_gbsf_hat_tr_set = np.dot(x_gbsf_tr_set, w_gbsf_tr_set) #tr_set의 y예측값
    y_gbsf_hat_te_set = np.dot(x_gbsf_te_set, w_gbsf_tr_set)
#te_set의 y예측값


    MSE_gbsf_tr = get_MSE(y_gbsf_hat_tr_set, tr_set[:, 1])

    MSE_gbsf_te = get_MSE(y_gbsf_hat_te_set, te_set[:, 1])


    MSE_gbsf_tr_list = np.append(MSE_gbsf_tr_list, MSE_gbsf_tr)

    MSE_gbsf_te_list = np.append(MSE_gbsf_te_list, MSE_gbsf_te)


    '''Polynominal basis funcion으로 구한 w와 MSE'''

    w_pbsf_tr_set, x_pbsf_tr_set = PBSF_ANALYSTIC_SOLUTION(i, tr_set[:, 0], tr_set[:, 1])
#basis 개수 k개일 때 tr_set의 weight, x값

    w_pbsf_te_set, x_pbsf_te_set = PBSF_ANALYSTIC_SOLUTION(i, te_set[:, 0], te_set[:, 1])
#te_set의 x데이터들얻기 위함


    y_pbsf_hat_tr_set = np.dot(x_pbsf_tr_set, w_pbsf_tr_set)
#tr_set의 y예측값

    y_pbsf_hat_te_set = np.dot(x_pbsf_te_set, w_pbsf_tr_set)
#te_set의 y예측값


    MSE_pbsf_tr = get_MSE(y_pbsf_hat_tr_set, tr_set[:, 1])

    MSE_pbsf_te = get_MSE(y_pbsf_hat_te_set, te_set[:, 1])


    MSE_pbsf_tr_list = np.append(MSE_pbsf_tr_list, MSE_pbsf_tr)

    MSE_pbsf_te_list = np.append(MSE_pbsf_te_list, MSE_pbsf_te)


fig = plt.figure()
```

```
plt.plot(K_list, MSE_gbsf_tr_list, 'r-o', markevery = 2)
plt.plot(K_list, MSE_gbsf_te_list, 'b-^', markevery = 2)
plt.legend(['tr_set', 'te_set'])
plt.xlabel("complexity")
plt.ylabel("error")
plt.title("Gaussian")

fig = plt.figure()
plt.plot(K_list, MSE_pbsf_tr_list, 'r-o', markevery = 2)
plt.plot(K_list, MSE_pbsf_te_list, 'b-^', markevery = 2)
plt.legend(['tr_set', 'te_set'])
plt.xlabel("complexity")
plt.ylabel("error")
plt.title("Polynominal")
```