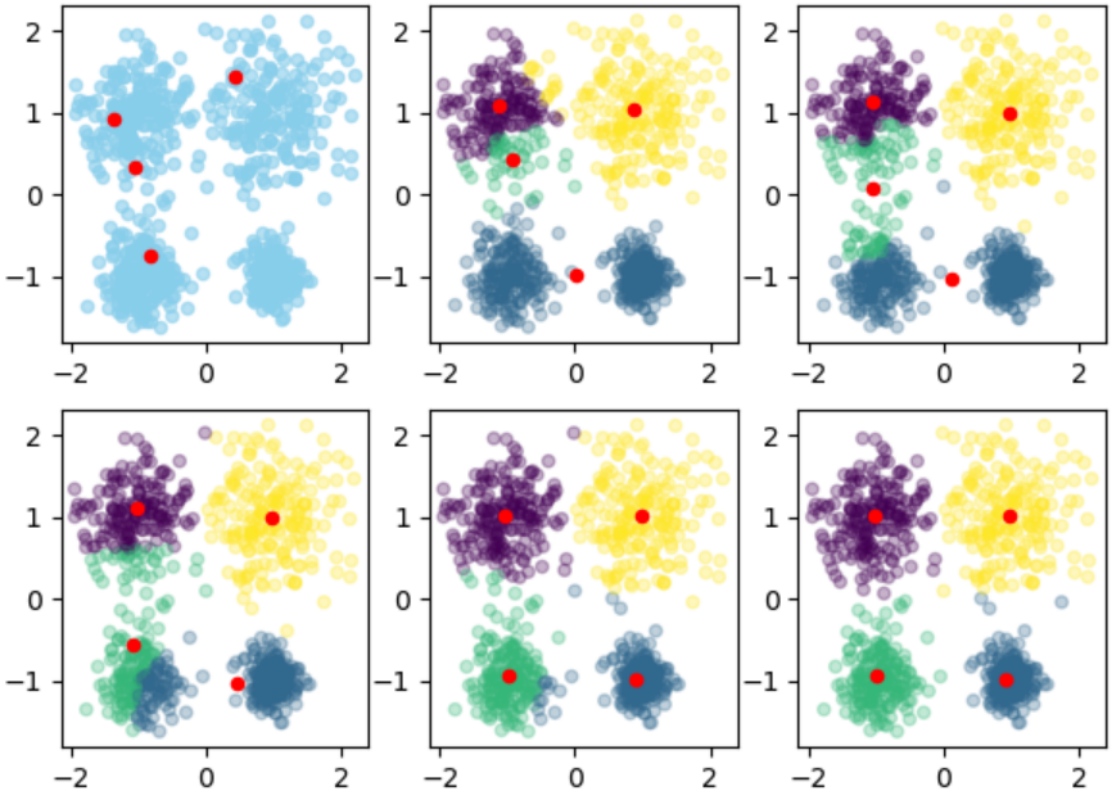
ML Day22 (Matplotlib)(K-means clustering)

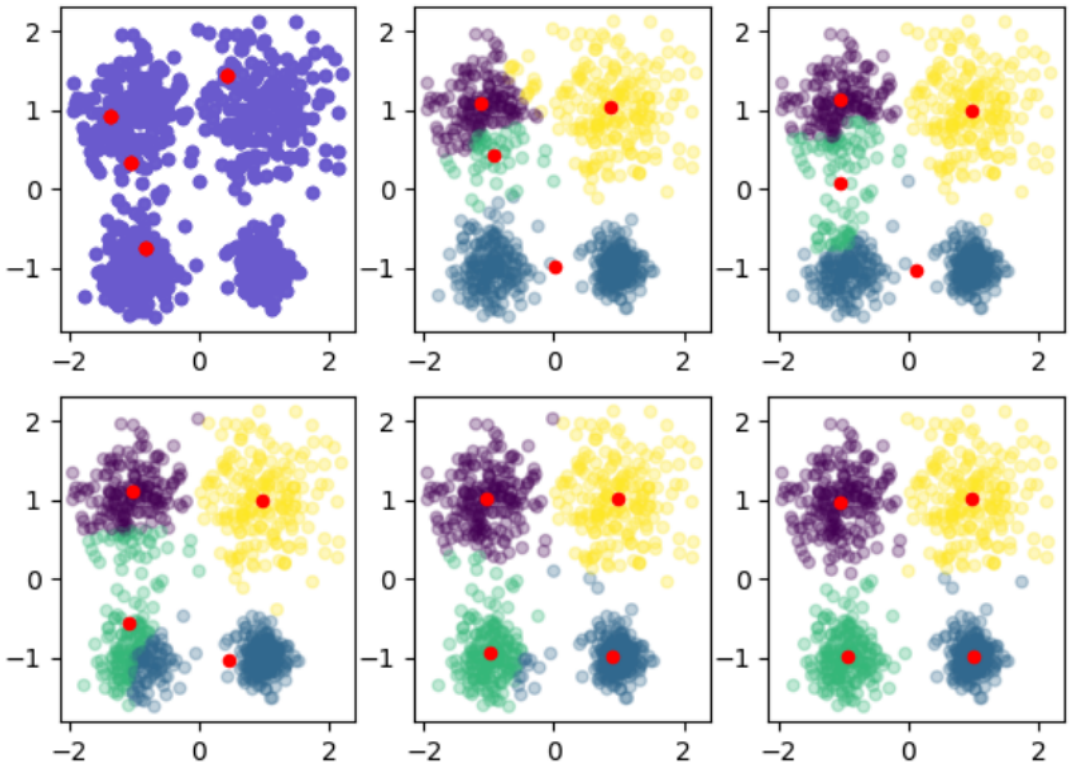
▼ **step (7) -** ax.scatter를 일일이 구현

```
np.random.seed(0)

n_samples = 200
stds = [0.5, 0.4, 0.3, 0.2]
means = [[1, 1], [-1, 1], [-1, -1], [1, -1]]
len_means = len(means)

fig, ax = plt.subplots(2, 3, figsize=(7, 5))


# 초기 4개의 군집 그리기, for문으로  len_means 만큼 돌아간 결과 나온 data vstack으로 쌓기
data_result = []
for i in range(len(means)):
    data = np.random.normal(loc=means[i], scale=stds[i], size=(n_samples, 2))
    ax[0][0].scatter(data[:, 0], data[:, 1], c='skyblue', s=20, alpha=0.6)
    data_result.append(data)
total_data = np.vstack([data_result[0], data_result[1], data_result[2], data_result[3]])
centeroid = shuffle(total_data, random_state=0)[:len_means]                  # shuffle함수로 뒤섞어준 후 [:len_means]까지 4개의 표본 추출
ax[0][0].scatter(centeroid[:, 0], centeroid[:, 1], color='r', s=20)


# Expectation step (1) / Euclidean distance를 구한 후 색 지정을 위해 nearest_clus_li에 append 해줌
nearest_clus_li = []
for tdata in total_data:
    dis = np.sum(((centeroid - tdata)**2)**0.5, axis=1)
    dist_argsort = np.argsort(dis)
    close_nearest = dist_argsort[0]
    nearest_clus_li.append(close_nearest)
ax[0][1].scatter(total_data[:, 0], total_data[:, 1], c=nearest_clus_li, s=20, alpha=0.3)   # c=nearest_clus_li => nearest_clus_li가


# Maximization step (1) / centeroid를 중심으로 색상이 바뀐 군집들의 평균을 구하고, 그 값을 다시 centeroid로 지정
arr_nearest = np.array(nearest_clus_li)

xy_data_li = []
for xy in range(len_means):
    x_data = total_data[arr_nearest == xy][:, 0].mean()
    y_data = total_data[arr_nearest == xy][:, 1].mean()
    xy_data_li.append([x_data, y_data])
    xy_arr = np.array(xy_data_li)
ax[0][1].scatter(xy_arr[:, 0], xy_arr[:, 1], color='r', s=20, zorder=1)

# Expectation step (2)
nearest_clus_li1 = []
for tdata1 in total_data:
    dis1 = np.sum(((xy_arr - tdata1)**2)**0.5, axis=1)
    dist_argsort1 = np.argsort(dis1)
    close_nearest1 = dist_argsort1[0]
    nearest_clus_li1.append(close_nearest1)
ax[0][2].scatter(total_data[:, 0], total_data[:, 1], c=nearest_clus_li1, s=20, alpha=0.3, zorder=1)

# # Maximization step (2)
arr_nearest1 = np.array(nearest_clus_li1)
```

```
xy_data_li1 = []
for xy1 in range(len_means):
    x_data1 = total_data[arr_nearest1 == xy1][:, 0].mean()
    y_data1 = total_data[arr_nearest1 == xy1][:, 1].mean()
    xy_data_li1.append([x_data1, y_data1])
    xy_arr1 = np.array(xy_data_li1)
ax[0][2].scatter(xy_arr1[:, 0], xy_arr1[:, 1], color='r', s=20, zorder=1)

# Expectation step (3)
nearest_clus_li2 = []
for tdata2 in total_data:
    dis2 = np.sum(((xy_arr1 - tdata2)**2)**0.5, axis=1)
    dist_argsort2 = np.argsort(dis2)
    close_nearest2 = dist_argsort2[0]
    nearest_clus_li2.append(close_nearest2)
ax[1][0].scatter(total_data[:, 0], total_data[:, 1], c=nearest_clus_li2, s=20, alpha=0.3, zorder=1)

# Maximization step (3)
arr_nearest2 = np.array(nearest_clus_li2)

xy_data_li2 = []
for xy2 in range(len_means):
    x_data2 = total_data[arr_nearest2 == xy2][:, 0].mean()
    y_data2 = total_data[arr_nearest2 == xy2][:, 1].mean()
    xy_data_li2.append([x_data2, y_data2])
    xy_arr2 = np.array(xy_data_li2)
ax[1][0].scatter(xy_arr2[:, 0], xy_arr2[:, 1], color='r', s=20, zorder=1)

# Expectation step (4)
nearest_clus_li3 = []
for tdata3 in total_data:
    dis3 = np.sum(((xy_arr2 - tdata3)**2)**0.5, axis=1)
    dist_argsort3 = np.argsort(dis3)
    close_nearest3 = dist_argsort3[0]
    nearest_clus_li3.append(close_nearest3)
ax[1][1].scatter(total_data[:, 0], total_data[:, 1], c=nearest_clus_li3, s=20, alpha=0.3, zorder=5)

# # Maximization step (4)
arr_nearest3 = np.array(nearest_clus_li3)

xy_data_li3 = []
for xy3 in range(len_means):
    x_data3 = total_data[arr_nearest3 == xy3][:, 0].mean()
    y_data3 = total_data[arr_nearest3 == xy3][:, 1].mean()
    xy_data_li3.append([x_data3, y_data3])
    xy_arr3 = np.array(xy_data_li3)
ax[1][1].scatter(xy_arr3[:, 0], xy_arr3[:, 1], color='r', s=20, zorder=5)

# Expectation step (5)
nearest_clus_li4 = []
for tdata4 in total_data:
    dis4 = np.sum(((xy_arr3 - tdata4)**2)**0.5, axis=1)
    dist_argsort4 = np.argsort(dis4)
    close_nearest4 = dist_argsort4[0]
    nearest_clus_li4.append(close_nearest4)
ax[1][2].scatter(total_data[:, 0], total_data[:, 1], c=nearest_clus_li4, s=20, alpha=0.3, zorder=5)

# # Maximization step (5)
arr_nearest4 = np.array(nearest_clus_li4)

xy_data_li4 = []
for xy4 in range(len_means):
    x_data4 = total_data[arr_nearest3 == xy4][:, 0].mean()
    y_data4 = total_data[arr_nearest3 == xy4][:, 1].mean()
    xy_data_li4.append([x_data4, y_data4])
    xy_arr4 = np.array(xy_data_li4)
ax[1][2].scatter(xy_arr4[:, 0], xy_arr4[:, 1], color='r', s=20, zorder=5)

plt.show()
```



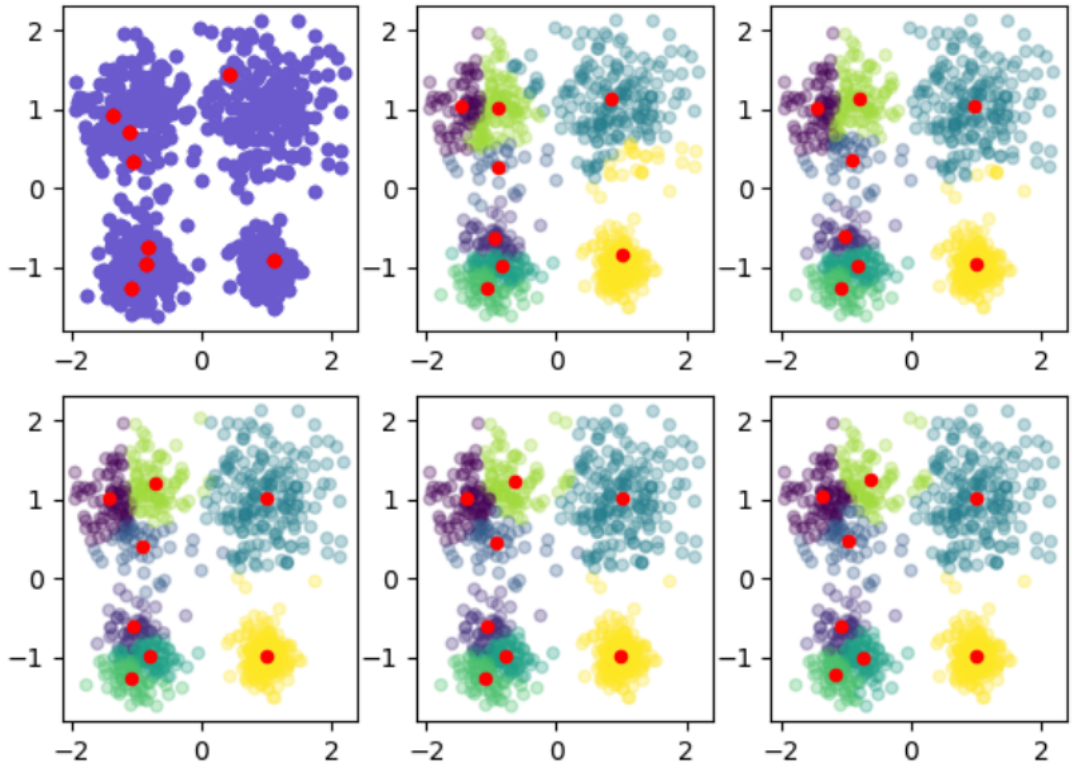
```
# for문과 enumerate를 사용하여 code를 줄이고 scatter 표현
np.random.seed(0)

n_samples = 200
stds = [0.5, 0.4, 0.3, 0.2]
means = [[1, 1], [-1, 1], [-1, -1], [1, -1]]
len_means = len(means)

fig, axes = plt.subplots(2, 3, figsize=(7, 5))

data_result = []
for i in range(len(means)):                                                    # 평균 means[i], 표준편차 stds[i]를 가지는
    data = np.random.normal(loc=means[i], scale=stds[i], size=(n_samples, 2))
    data_result.append(data)
total_data = np.vstack([data_result[0], data_result[1], data_result[2], data_result[3]])     # append한 list의 index를 이용하여 vstac

for i in range(len(means)):
    centeroid = shuffle(total_data, random_state=0)[:len_means]                # total_data를 shuffle 섞은 후 len_mean
    axes[0][0].scatter(centeroid[:, 0], centeroid[:, 1], color='r', s=20, zorder=3)          # centeroid와 total_data를 가지고 ax[0]
    axes[0][0].scatter(total_data[:, 0], total_data[:, 1], c='slateblue', s=20, alpha=0.6)

for ax_idx, ax in enumerate(axes.flatten()[1:]):                               # enumerate 함수를 사용하여 axes를 평평하게
    nearest_clus_li = []
    xy_data_li = []
    for tdata in total_data:                                                   # total_data를 for문을 돌며
        dis = np.sum(((centeroid - tdata) ** 2) ** 0.5, axis=1)                # 각각의 dot들과 centeroid간의 Euclidean
        dist_argsort = np.argsort(dis)                                         # np.argsort를 사용하여 dis를 오름차순으로
        close_nearest = dist_argsort[0]                                        # 정렬된 dist_argsort의 [0]index에 위치하
        nearest_clus_li.append(close_nearest)                                  # 비어있는 nearest_clus_li에 append
        arr_nearest = np.array(nearest_clus_li)                                # nearest_clus_li를 numpy array형태로 변
    for xy in range(len_means):
        x_data = total_data[arr_nearest == xy][:, 0].mean()                    # boolean 인덱싱으로 total_data의 위에서
        y_data = total_data[arr_nearest == xy][:, 1].mean()                    # 1번 column의 평균을 y_data로 대입
        xy_data_li.append([x_data, y_data])                                    # [x_data, y_data] 형식으로 비어있는 xy_d
        xy_arr = np.array(xy_data_li)
        centeroid = xy_arr                                                     # update된 centroid(xy_arr)를 다시 cent
    ax.scatter(total_data[:, 0], total_data[:, 1], c=nearest_clus_li, s=20, alpha=0.3)
    ax.scatter(centeroid[:, 0], centeroid[:, 1], color='r', s=20, zorder=1)

plt.show()
```



▼ **step (8) -** final (1) K = n 적용하여 graph가 변화할 수 있도록 구현

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.utils import shuffle

np.random.seed(0)

n_samples = 200
stds = [0.5, 0.4, 0.3, 0.2]
means = [[1, 1], [-1, 1], [-1, -1], [1, -1]]

K = 3

fig, axes = plt.subplots(2, 3, figsize=(7, 5))

# total_data 생성
data_result = []
for i in range(len(means)):                                                    # 평균 means[i], 표준편차 stds[i]를 가지는 (
    data = np.random.normal(loc=means[i], scale=stds[i], size=(n_samples, 2))
    data_result.append(data)                                                   # for문으로 생성된 data(200, 2)를 len(mean
total_data = np.vstack([data_result[0], data_result[1], data_result[2], data_result[3]])   # append한 list의 index를 이용하여 vstack을

# initial centroid, total_data scatterplot
for i in range(len(means)):                                                    # shuffle함수로 뒤섞어준 후 4개의 표본 추출
    centeroid = shuffle(total_data, random_state=0)[:K]
    axes[0][0].scatter(centeroid[:, 0], centeroid[:, 1], color='r', s=20, zorder=3)
    axes[0][0].scatter(total_data[:, 0], total_data[:, 1], c='slateblue', s=20, alpha=0.6)

# Expectation step / centroid와 total_data scatter의 각각의 dot 간의 euclidean distance를 구한 후 가장 근접한 centroid와 색상 일치
for ax_idx, ax in enumerate(axes.flatten()[1:]):                               # enumerate(axes.flatten()[1:]) -> 그래
    nearest_clus_li = []
    xy_data_li = []
    for tdata in total_data:
        dis = np.sum(((centeroid - tdata) ** 2) ** 0.5, axis=1)
        dist_argsort = np.argsort(dis)
        close_nearest = dist_argsort[0]
        nearest_clus_li.append(close_nearest)
        arr_nearest = np.array(nearest_clus_li)                                # nearest_clus_li : 입력 받은 K의 수가 nea
```

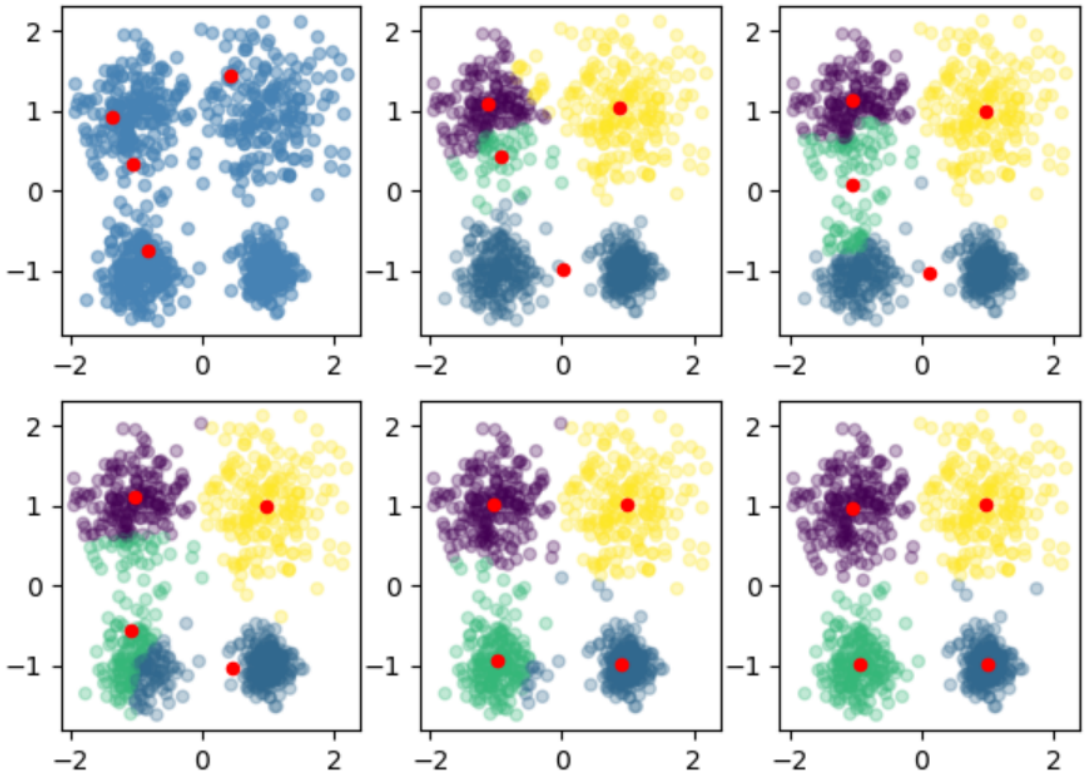
```
# Maximization step  / centeroid를 중심으로 색상이 바뀐 군집들의 평균을 구하고, 그 값을 다시 centeroid로 지정
    for xy in range(K):
        x_data = total_data[arr_nearest == xy][:, 0].mean()
        y_data = total_data[arr_nearest == xy][:, 1].mean()
        xy_data_li.append([x_data, y_data])
        xy_arr = np.array(xy_data_li)
        centeroid = xy_arr
    ax.scatter(total_data[:, 0], total_data[:, 1], c=nearest_clus_li, s=20, alpha=0.3)
    ax.scatter(centeroid[:, 0], centeroid[:, 1], color='r', s=20, zorder=1)

plt.show()
```



▼ **step (9) -** final (2) 함수를 만들어 구현

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.utils import shuffle

np.random.seed(0)

n_samples = 200
stds = [0.5, 0.4, 0.3, 0.2]
means = [[1, 1], [-1, 1], [-1, -1], [1, -1]]
len_means = len(means)

fig, axes = plt.subplots(2, 3, figsize=(7, 5))

def get_dataset():                                                              # 기본 data를 만드는 get_dataset 함수
    data_result = []
    for i in range(len(means)):
        data = np.random.normal(loc=means[i], scale=stds[i], size=(n_samples, 2))
        data_result.append(data)
    total_data = np.vstack([data_result[0], data_result[1], data_result[2], data_result[3]])
    return total_data                                                           # total_data를 return

def get_initial_centeroids(total_data, K):                                      # 초기 centeroid를 추출하는 get_initia
    for i in range(len(means)):
        centeroid = shuffle(total_data, random_state=0)[:K]                     # 기본 data에서 shuffle로 섞은 후 K 인덱
    return centeroid                                                            # (초기)centeroid를 return
```

```
def clustering(total_data, centeroid):                                  # total_data와 centeroid를 매개변수로
    for ax_idx, ax in enumerate(axes.flatten()[1:]):                    # axes를 flatten하여 [1:]부터 enumera
        nearest_clus_li = []
        for tdata in total_data:                                        # total_data를 for문을 이용하여 euclid
            dis = np.sum(((centeroid - tdata) ** 2) ** 0.5, axis=1)
            dist_argsort = np.argsort(dis)
            close_nearest = dist_argsort[0]
            nearest_clus_li.append(close_nearest)
            arr_nearest = np.array(nearest_clus_li)                     # 색상을 정해주기 위해 [0]번째 index값을
        centeroid = update_centeroids(total_data, arr_nearest, K)       # 함수 안 함수 (중첩함수, update_center
        ax.scatter(total_data[:, 0], total_data[:, 1], c=arr_nearest, s=20, alpha=0.3)
        ax.scatter(centeroid[:, 0], centeroid[:, 1], color='r', s=20, zorder=1)
    return arr_nearest, centeroid                                       # clustering 함수는 arr_nearest와 upd

def update_centeroids(total_data, arr_nearest, K):                      # total_data와 clustering함수의 retur
    xy_data_li = []
    for xy in range(K):
        x_data = total_data[arr_nearest == xy][:, 0].mean()
        y_data = total_data[arr_nearest == xy][:, 1].mean()
        xy_data_li.append([x_data, y_data])
        xy_arr = np.array(xy_data_li)
        centeroid = xy_arr                                              # update되는 centeroid인 xy_arr를 cen
    return centeroid                                                    # centeroid값을 return

K = 4

t_data = get_dataset()                                                  # get_dataset()함수를 통해 return되는
initial_data = get_initial_centeroids(t_data, K)                        # 초기 centeroids값을 initial_data 변
clustering(t_data, initial_data)
axes[0][0].scatter(initial_data[:, 0], initial_data[:, 1], color='r', s=20, zorder=3)
axes[0][0].scatter(t_data[:, 0], t_data[:, 1], c='steelblue', s=20, alpha=0.6)

plt.show()
```




```
# 선생님 code / 함수 선언

import numpy as np


def get_dataset():
    n_cluster_samples = 100
```

```
    means = [[1, 1], [-1, 1], [-1, -1], [1, -1]]
    stds = [0.5, 0.4, 0.3, 0.2]

    n_clusters = len(means)
    data = []
    for cluster_idx in range(n_clusters):
        cluster_data = np.random.normal(loc=means[cluster_idx],
                                        scale=stds[cluster_idx],
                                        size=(n_cluster_samples, 2))
        data.append(cluster_data)
    data = np.vstack(data)
    return data


def get_initial_centroids(dataset, K):
    n_data = dataset.shape[0]

    random_idx = np.arange(n_data)
    np.random.shuffle(random_idx)
    centroid_idx = random_idx[:K]

    centroids = dataset[centroid_idx]
    return centroids


def cal_which_cluster(dataset, centroids, sample_idx):
    sample = dataset[sample_idx]

    dists = []
    for centroid in centroids:
        dist = np.sum((sample - centroid) ** 2)
        dists.append(dist)

    which_cluster = np.argmin(dists)
    return which_cluster


def clustering(dataset, centroids, K):
    n_data = dataset.shape[0]
    clusters = [[] for _ in range(K)]

    for sample_idx in range(n_data):
        which_cluster = cal_which_cluster(dataset, centroids, sample_idx)
        clusters[which_cluster].append(dataset[sample_idx])

    clusters = [np.array(cluster) for cluster in clusters]
    return clusters


def update_centroids(clusters):
    centroids = np.array([np.mean(cluster, axis=0)
                          for cluster in clusters])
    return centroids
```

```
# 선생님 code

import numpy as np
import matplotlib.pyplot as plt

from utils import get_dataset              #
from utils import get_initial_centroids
from utils import clustering
from utils import update_centroids


K = 4
dataset = get_dataset()
centroids = get_initial_centroids(dataset, K)

fig, axes = plt.subplots(2, 3, figsize=(10, 6))
axes[0, 0].scatter(dataset[:, 0], dataset[:, 1], alpha=0.7)
axes[0, 0].scatter(centroids[:, 0], centroids[:, 1], color='r', s=50)

for ax in axes.flat[1:]:  #axes.flatten()[1:]
    clusters = clustering(dataset, centroids, K)
    centroids = update_centroids(clusters)

    for cluster_idx in range(K):
        cluster = clusters[cluster_idx]
        ax.scatter(cluster[:, 0], cluster[:, 1], alpha=0.3)
        ax.scatter(centroids[:, 0], centroids[:, 1], color='r')

plt.show()
```