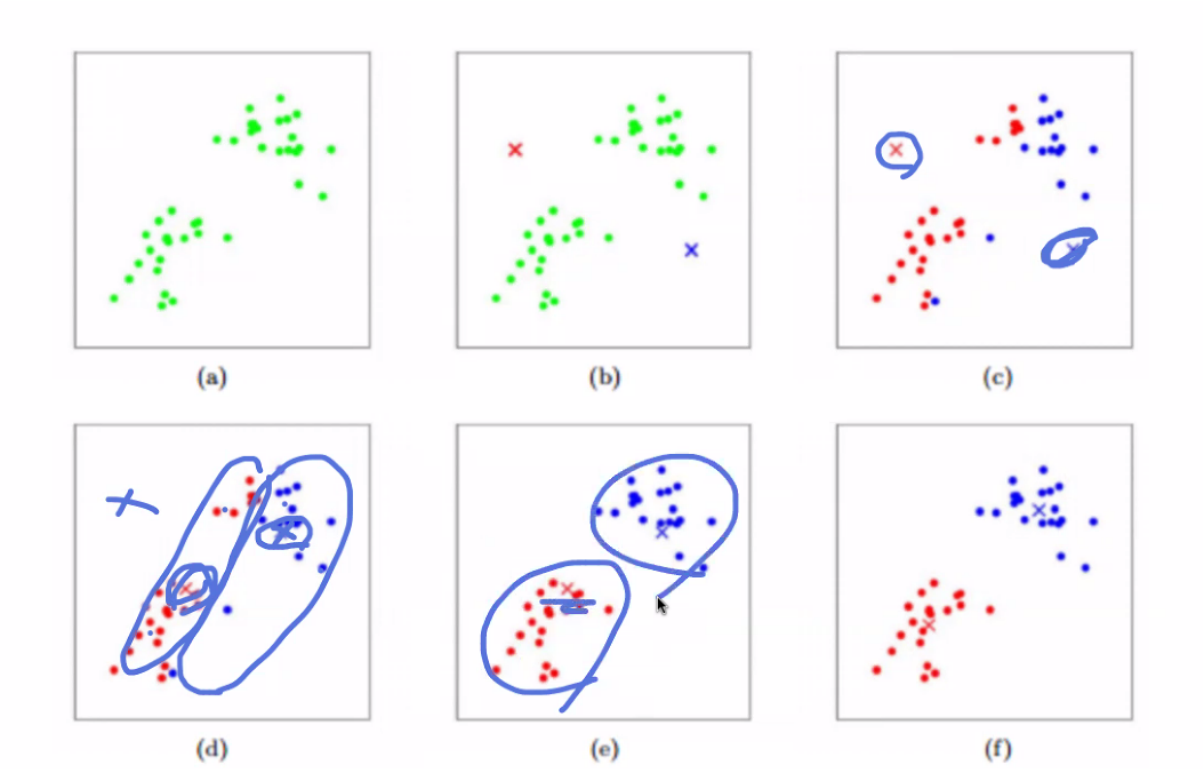
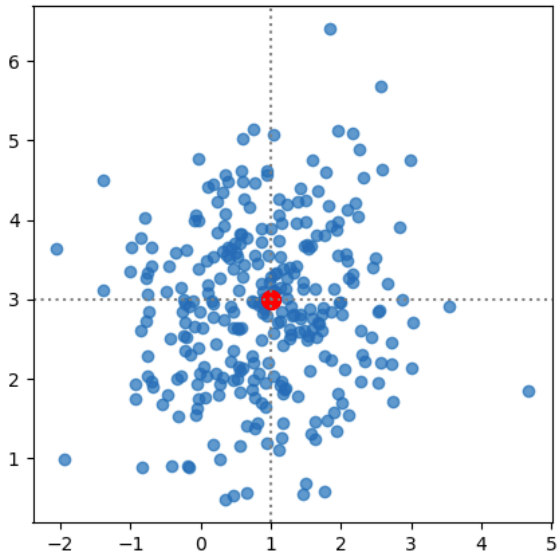
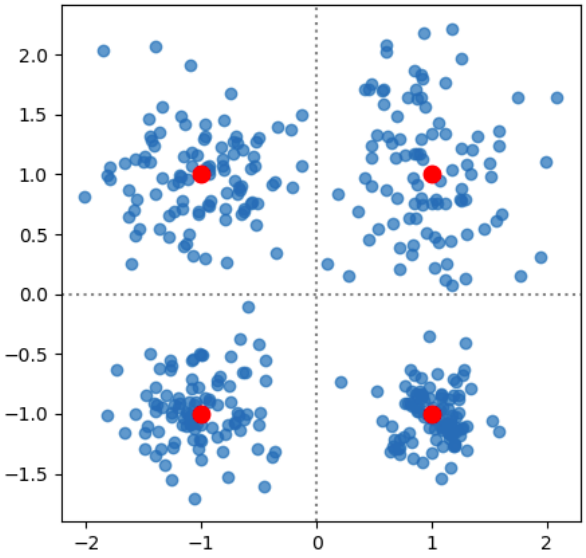
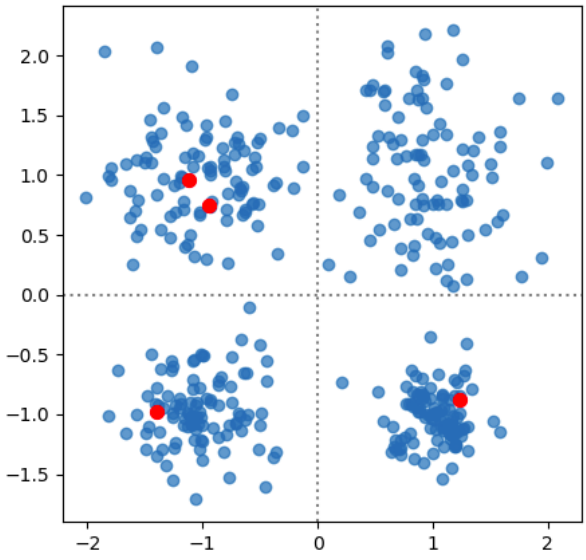
ML Day21 (Matplotlib)(K-means clustering)

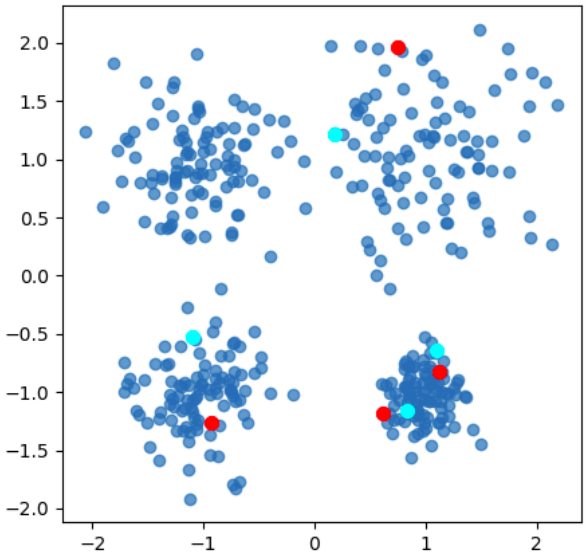


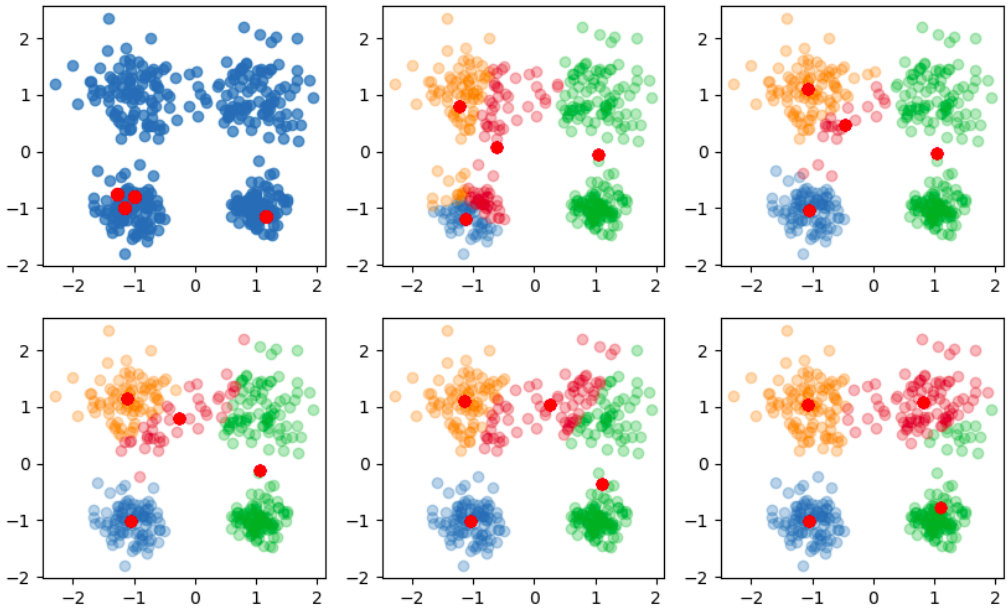
여기서 일단 k 값은 2다. 그래서 (b)에서 일단 중심점 2개를 아무 데나 찍고, (c)에서는 각 데이터들을 두 개 점 중 가까운 곳으로 할당한다. (d)에서는 그렇게 군집이 지정된 상태로 중심점을 업데이트 한다. 그리고 (e)에서는 업데이트 된 중심점과 각 데이터들의 거리를 구해서 군집을 다시 할당하는 거다.


6
5
4
3
2
1
−2
−1
0
1
2
3
4
5


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2

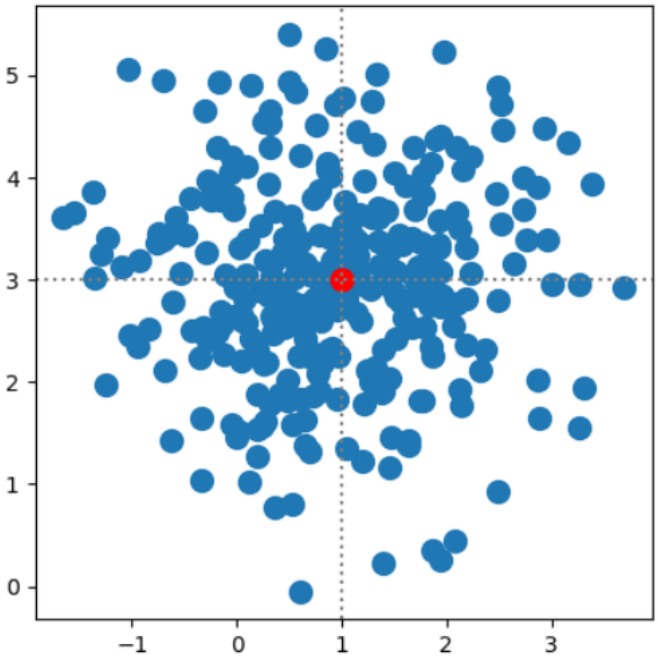

2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2
−1
0
1
2



K-means clustering

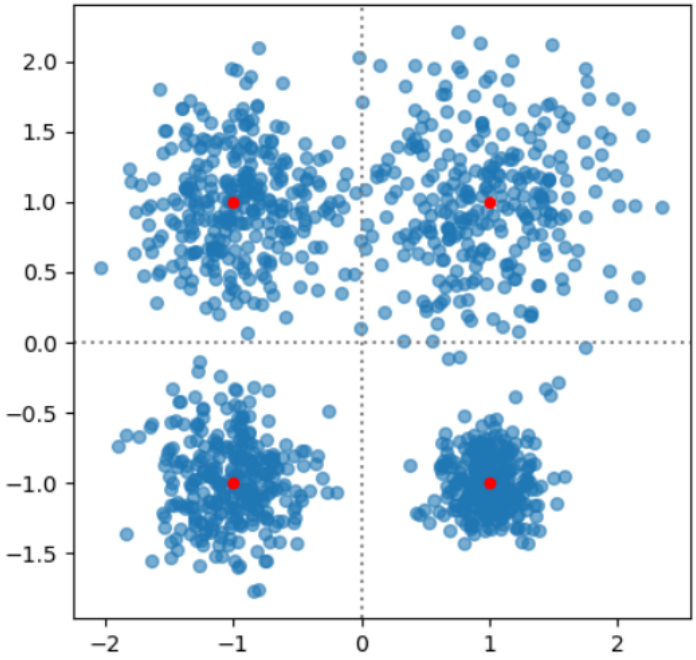
▼ **step (1)** - x_mean = 1, y_mean= 3을 가지는 1개의 scatter,

```
np.random.seed(0)

n_samples = 300
# means = [[1, 1], [-1, 1], [-1, -1], [1, -1]]
# stds = [0.5, 0.4, 0.3, 0.2]
x_mean, y_mean = 1, 3

data = np.random.normal(loc=[x_mean, y_mean], scale=1, size=(n_samples, 2))

fig, ax = plt.subplots(figsize=(5, 5))
ax.scatter(data[:, 0], data[:, 1], s=100)

ax.scatter(x_mean, y_mean, color='r', s=100)


ax.axvline(x=x_mean, color='gray', ls=':')
ax.axhline(y=y_mean, color='gray', ls=':')
plt.show()
```



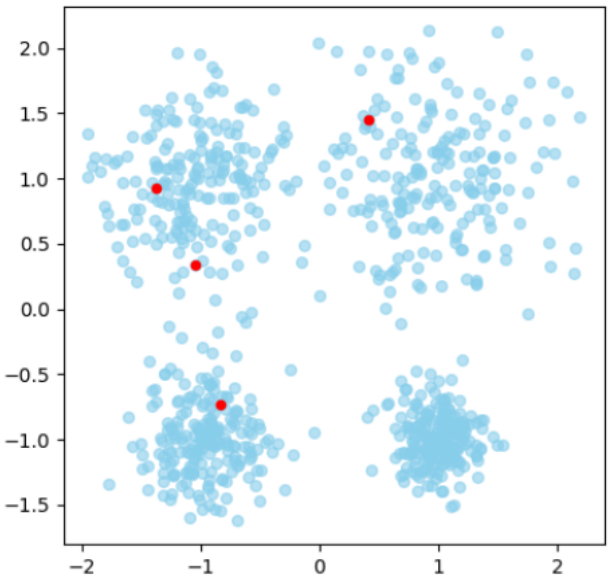
▼ **step (2)** - means = [[1, 1], [-1, 1], [-1, -1], [1, -1]], stds = [0.5, 0.4, 0.3, 0.2] 을 가지는 4개의 scatter

```
np.random.seed(0)

n_samples = 300
means = [[1, 1], [-1, 1], [-1, -1], [1, -1]]
stds = [0.5, 0.4, 0.3, 0.2]
# x_mean, y_mean = 1, 3

data1 = np.random.normal(loc=means[0], scale=stds[0], size=(n_samples, 2))
data2 = np.random.normal(loc=means[1], scale=stds[1], size=(n_samples, 2))
data3 = np.random.normal(loc=means[2], scale=stds[2], size=(n_samples, 2))
data4 = np.random.normal(loc=means[3], scale=stds[3], size=(n_samples, 2))


fig, ax = plt.subplots(figsize=(5, 5))

ax.scatter([data1[:, 0],data2[:, 0],data3[:, 0],data4[:, 0]], [data1[:, 1],data2[:, 1],data3[:, 1],data4[:, 1]], s=30, alpha=0.6)

for i in means:
    ax.scatter(i[0], i[1], color='r', s=20)

ax.axvline(x=0, color='gray', ls=':')
ax.axhline(y=0, color='gray', ls=':')
plt.show()
```



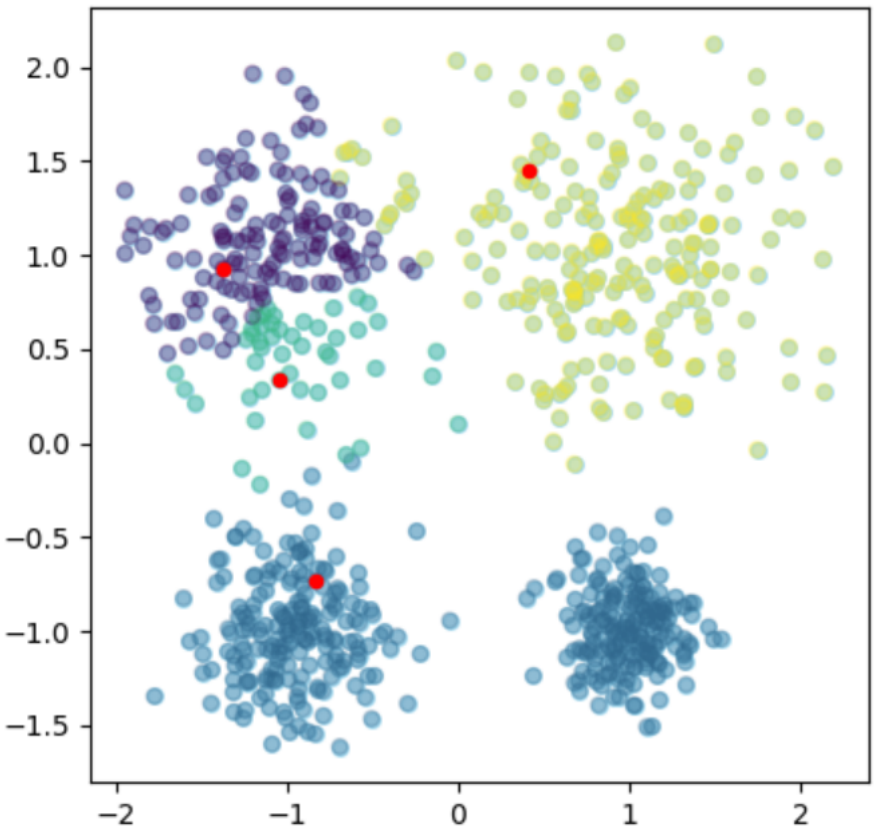
▼ **step (3) -** 초기 4개의 군집에서 값들을 뒤섞은 후 [:len_means]로 slicing하여 추출 후 scatter표현

```
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)
colors = ['orange', 'red', 'blue', 'green']
fig, ax = plt.subplots(figsize=(5, 5))


# 초기 4개의 군집 그리기, for문으로 len_means 만큼 돌아간 결과로 나온 data들을 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.scatter(data[:, 0], data[:, 1], c='skyblue', s=30, 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 4개의 표본 추출
ax.scatter(centeroid[:, 0], centeroid[:, 1], color='r', s=20, zorder=3)
```



▼ **step (4)** - Expectation step / 각 centroid와 scatter dot들 간의 Euclidean거리를 구한 후 근접하는 dot들의 색을 표현

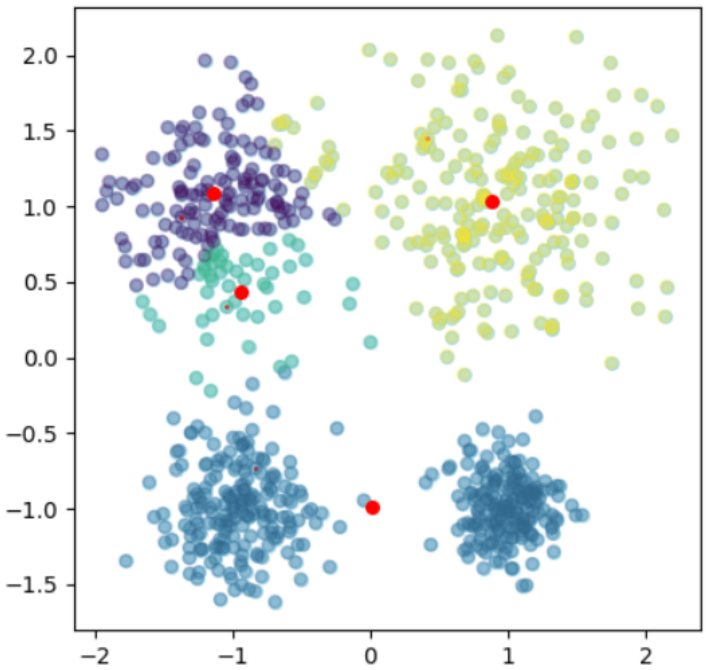
```
np.random.seed(0)

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, ax = plt.subplots(figsize=(5, 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.scatter(data[:, 0], data[:, 1], c='skyblue', s=30, 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 4개의 표본 추출
ax.scatter(centeroid[:, 0], centeroid[:, 1], color='r', s=20, zorder=3)


# 유클리디안 거리를 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.scatter(total_data[:, 0], total_data[:, 1], c=nearest_clus_li, s=30, alpha=0.3)


plt.show()
```



▼ **step (5)** - Maximization step / 색이 바뀐 각각의 군집들의 평균을 구하고, 그 평균값을 다시 centroid로 변경

```
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)
colors = ['orange', 'red', 'blue', 'green']
fig, ax = plt.subplots(figsize=(5, 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.scatter(data[:, 0], data[:, 1], c='skyblue', s=30, 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 4개의 표본 추출
ax.scatter(centeroid[:, 0], centeroid[:, 1], color='r', s=1)            # , zorder=3


# Expectation step (1) / 유클리디안 거리를 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.scatter(total_data[:, 0], total_data[:, 1], c=nearest_clus_li, s=30, alpha=0.3)


# 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 = []
```

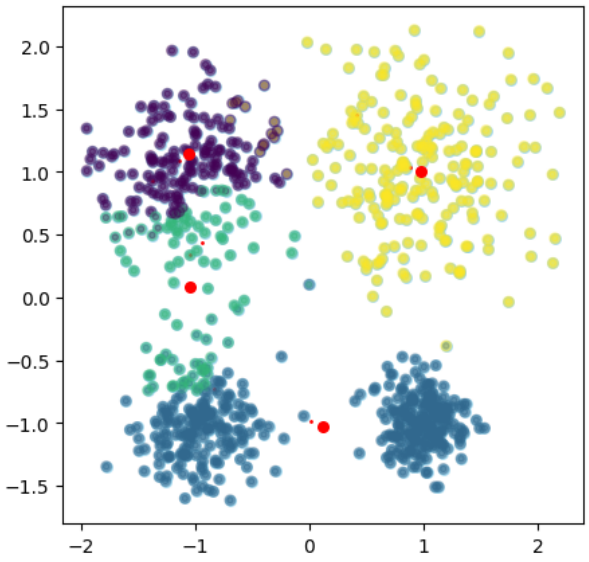
```
for xy in range(len_means):
    x_data = total_data[arr_nearest == xy][:, 0].mean()
    y_data = total_data[arr_nearest == xy][:, 1].mean()
    xy_data_li.append([x_data, y_data])
    xy = np.array(xy_data_li)
ax.scatter(x_data, y_data, color='r', s=30, zorder=3)
```



▼ **step (6)** - step4, step5를 반복

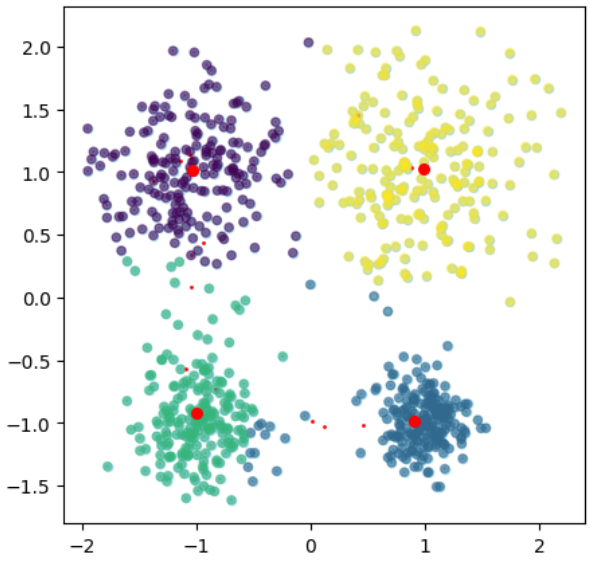
```
np.random.seed(0)

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(figsize=(5, 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.scatter(data[:, 0], data[:, 1], c='skyblue', s=30, 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 4개의 표본 추출
ax.scatter(centeroid[:, 0], centeroid[:, 1], color='r', s=1)            # , zorder=3


# Expectation step (1) / 유클리디안 거리를 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.scatter(total_data[:, 0], total_data[:, 1], c=nearest_clus_li, s=10, alpha=0.5)


# 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.scatter(xy_arr[:, 0], xy_arr[:, 1], color='r', s=1, 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.scatter(total_data[:, 0], total_data[:, 1], c=nearest_clus_li1, s=30, alpha=0.4, zorder=3)

# # 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.scatter(xy_arr1[:, 0], xy_arr1[:, 1], color='r', s=30, zorder=5)

plt.show()
```




```
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(figsize=(5, 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.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 4개의 표본 추출
ax.scatter(centeroid[:, 0], centeroid[:, 1], color='r', s=1)            # , zorder=3


# Expectation step (1) / 유클리디안 거리를 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.scatter(total_data[:, 0], total_data[:, 1], c=nearest_clus_li, s=1, alpha=0.1)


# 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.scatter(xy_arr[:, 0], xy_arr[:, 1], color='r', s=1, 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.scatter(total_data[:, 0], total_data[:, 1], c=nearest_clus_li1, s=1, alpha=0.1, 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.scatter(xy_arr1[:, 0], xy_arr1[:, 1], color='r', s=1, 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.scatter(total_data[:, 0], total_data[:, 1], c=nearest_clus_li2, s=1, alpha=0.1, 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.scatter(xy_arr2[:, 0], xy_arr2[:, 1], color='r', s=1, 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.scatter(total_data[:, 0], total_data[:, 1], c=nearest_clus_li3, s=20, alpha=0.5, 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.scatter(xy_arr3[:, 0], xy_arr3[:, 1], color='r', s=30, zorder=5)
plt.show()
```


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
2