순천향대학교
SOON CHUN HYANG UNIVERSITY

2024년 2학기

선형대수

3장

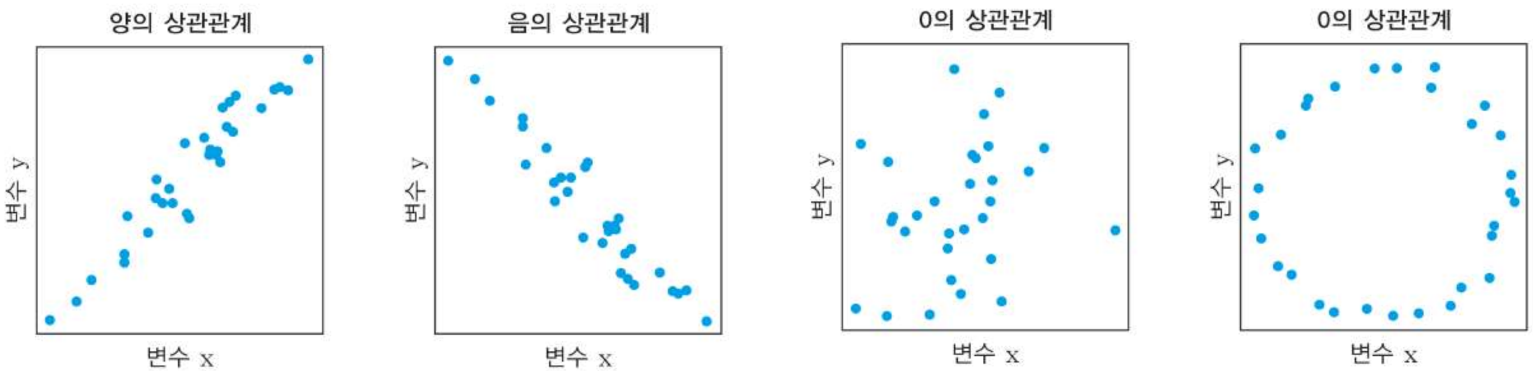
Chapter 03

- SECTION 03-1 상관관계와 코사인 유사도
- SECTION 03-2 시계열 필터링과 특징 탐지
- SECTION 03-3 k-평균 클러스터링

SCH

SECTION 03-1 상관관계와 코사인 유사도(1)

- 상관계수(correlation coefficient)
  - 두 변수 사이의 선형 관계를 정량화한 하나의 숫자
  - 상관계수의 범위는 -1부터 +1까지
    - -1은 완벽한 음의 관계, +1은 완벽한 양의 관계, 0은 선형 관계가 없음



양의 상관관계, 음의 상관관계, 0의 상관관계를 나타낸 데이터의 예제. 맨 오른쪽 그림은 상관관계가 선형적 척도라는 것을 나타냄 (상관계수가 0이라도 변수 간의 비선형 관계는 존재할 수 있음)

SECTION 03-1 상관관계와 코사인 유사도(2)

- 상관계수가 기대하는 범위 -1과 +1 사이에 존재하려면 정규화가 필요
  - 각 변수의 평균중심화
    - 평균중심화는 각 데이터값에서 평균값을 빼는 것
  - 벡터 노름 곱으로 내적을 나누기
    - 분할(divisive) 정규화는 측정 단위를 제거하고 상관계수 최대 크기를 |1|로 조정

피어슨 상관계수 공식

$$\rho = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

선형대수학 용어로 나타낸 피어슨 상관계수

$$\rho = \frac{\tilde{\mathbf{x}}^{\mathrm{T}}\tilde{\mathbf{y}}}{\|\tilde{\mathbf{x}}\|\ \|\tilde{\mathbf{y}}\|}$$

  - 코사인 유사도(cosine similarity)
    - 코사인 유사도에 대한 공식은 단순히 내적의 기하학적 공식으로 코사인 항을 구한 것
    - α는 **x**와 **y**의 내적

$$\cos(\theta_{x,y}) = \frac{\alpha}{\|\mathbf{x}\|\ \|\mathbf{y}\|}$$

실습1

- 상관계수 코드를 작성
  - 정답은 np.corrcoef()로 확인
  - myCorrCoef 함수를 만들고, 출력은 스칼라 값 (행렬X)

```
v1 = np.array([1, 2, 3])
v2 = np.array([2, 4, 5])

print(np.corrcoef(v1, v2), '₩n')
print(myCorrCoef(v1, v2))
```

```
[[1.         0.98198051]
 [0.98198051 1.        ]]

0.9819805060619659
```

실습2

- 코사인 유사도 코드를 작성
  - $\cos(\theta_{x,y})$ 의 값이 코사인 유사도

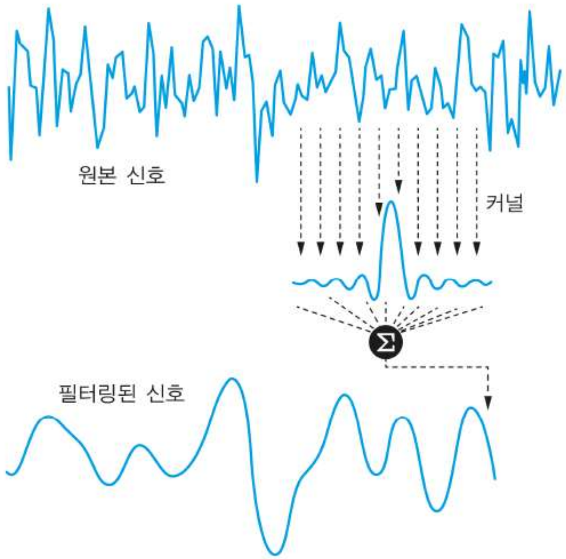
$$\cos(\theta_{x,y}) = \frac{\alpha}{\| \mathbf{x} \| \ \| \mathbf{y} \|}$$

```
#코사인 유사도
v1 = np.array([1, 2, 3])
v2 = np.array([2, 4, 5])



cos
```

```
0.9960238411119946
```

SECTION 03-2 시계열 필터링과 특징 탐지

- 내적은 시계열 필터링에도 사용
  - 커널과 시계열 신호 사이의 내적을 계산하는 것이 필터링 메커니즘
  - 필터링할 때 일반적으로 지역(local) 특징 탐지를 해야 하고 커널은 일반적으로 전체 시계열보다 훨씬 짧음
    - 커널과 동일한 길이의 짧은 데이터 조각과 커널 사이의 내적을 계산
    - 이 과정으로 필터링된 신호 구간에서 한 점이 생성되고 커널을 오른쪽으로 한 구간씩 이동시키면서 다른(또는 겹쳐진) 신호 조각과 내적을 계산 - 합성곱(convolution)



시계열 필터링 도식화

SECTION 03-3 *k*-평균 클러스터링(1)

- *k*-평균 클러스터링(k-means clustering)
  - 그룹 중심까지의 거리를 최소화하도록 다변량 데이터를 상대적으로 적은 수의 그룹 또는 범주로 분류하는 비지도 기법
  - *k*-평균 클러스터링 구현할 알고리즘
    1. 데이터 공간에서 임의의 *k*개 중심 점을 초기화. 여기서 중심은 클래스 또는 범주이며, 다음 단계에서는 각 데이터 관측치를 각 클래스에 할당
(중심은 임의의 차원의 수로 일반화된 형태입니다).
    2. 각 데이터 관측치와 각 중심 사이의 유클리드 거리를 계산
    3. 각 데이터 관측치를 가장 가까운 중심의 그룹에 할당
    4. 각 중심을 해당 중심에 할당된 모든 데이터 관측치의 평균으로 갱신
    5. 수렴 기준을 만족할 때까지 또는 *N* 회까지 2~4단계를 반복

SECTION 03-3 *k*-평균 클러스터링(2)

[1 단계] - 무작위로 클러스터 중심 *k*개를 데이터 생성

- *k*는 *k*-평균 클러스터링의 매개변수적 (여기서는 *k* = 3으로 고정)
- *k* 개 중심인 데이터 샘플을 무작위로 생성
- blur는 랜덤값 변동 폭을 설정

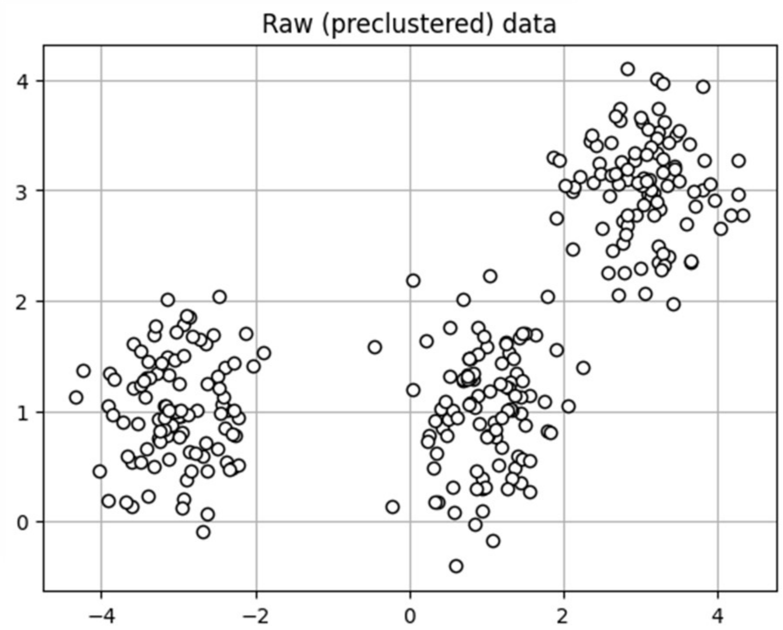
```
#k-평균 클러스터링
import matplotlib.pyplot as plt

## 데이터 개수
nPerClust = 100

# 랜덤값 변동 폭을 설정
blur = 0.5

# 중심점 3개, 데이터 생성을 위한 중심점
A = [  1, 1 ]
B = [ -3, 1 ]
C = [  3, 3 ]

# 데이터 생성. 각 (2, 100)
a = [ A[0]+np.random.randn(nPerClust)*blur , A[1]+np.random.randn(nPerClust)*blur ]
b = [ B[0]+np.random.randn(nPerClust)*blur , B[1]+np.random.randn(nPerClust)*blur ]
c = [ C[0]+np.random.randn(nPerClust)*blur , C[1]+np.random.randn(nPerClust)*blur ]
```

SECTION 03-3 *k*-평균 클러스터링(3)

[2 단계] – 무작위로 생성된 데이터를 그리기

- np.concatenate()의 2번째 인자가 axis=0 이면 행방향, axis=1이면 열방향으로 배열을 합침
- blur 값을 변경해 생성 데이터 값 확인

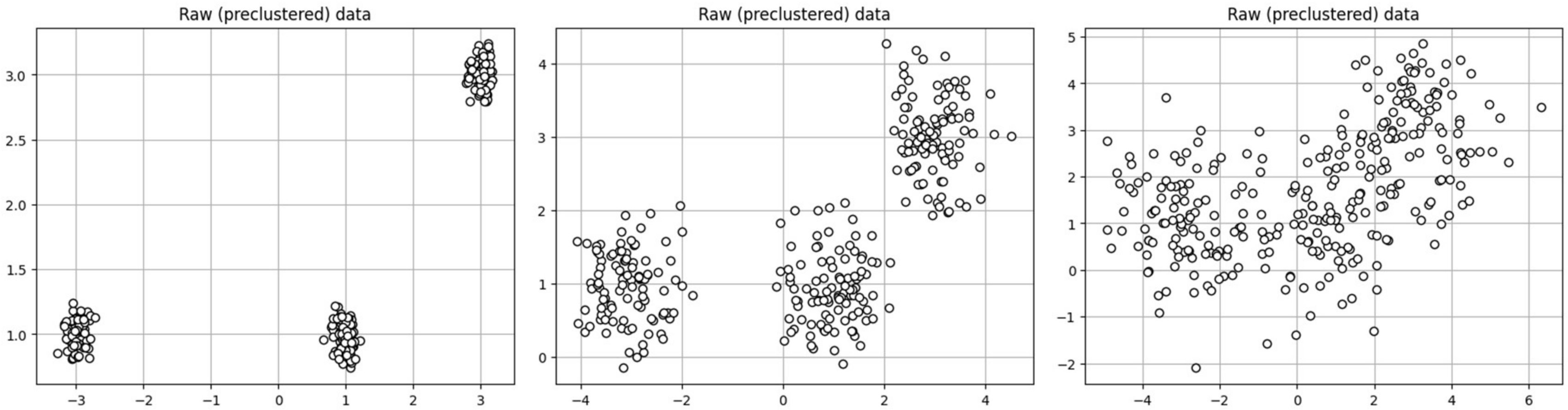
```
# 3개의 배열 합치기
data = np.transpose( np.concatenate((a,b,c),axis=1) ) #axis=1은 열방향

# axis에 따른 결과값 확인
tm0 = np.concatenate((a,b,c),axis=1)
print(np.shape(a), tm0.shape)
tm1 = np.concatenate((a,b,c),axis=0)
print(np.shape(a), tm1.shape)


# plot data
plt.plot(data[:,0],data[:,1],'ko',markerfacecolor='w')
plt.title('Raw (preclustered) data')

plt.grid()
plt.show()
```



SECTION 03-3 *k*-평균 클러스터링(3)

[2 단계] – 무작위로 생성된 데이터를 그리기

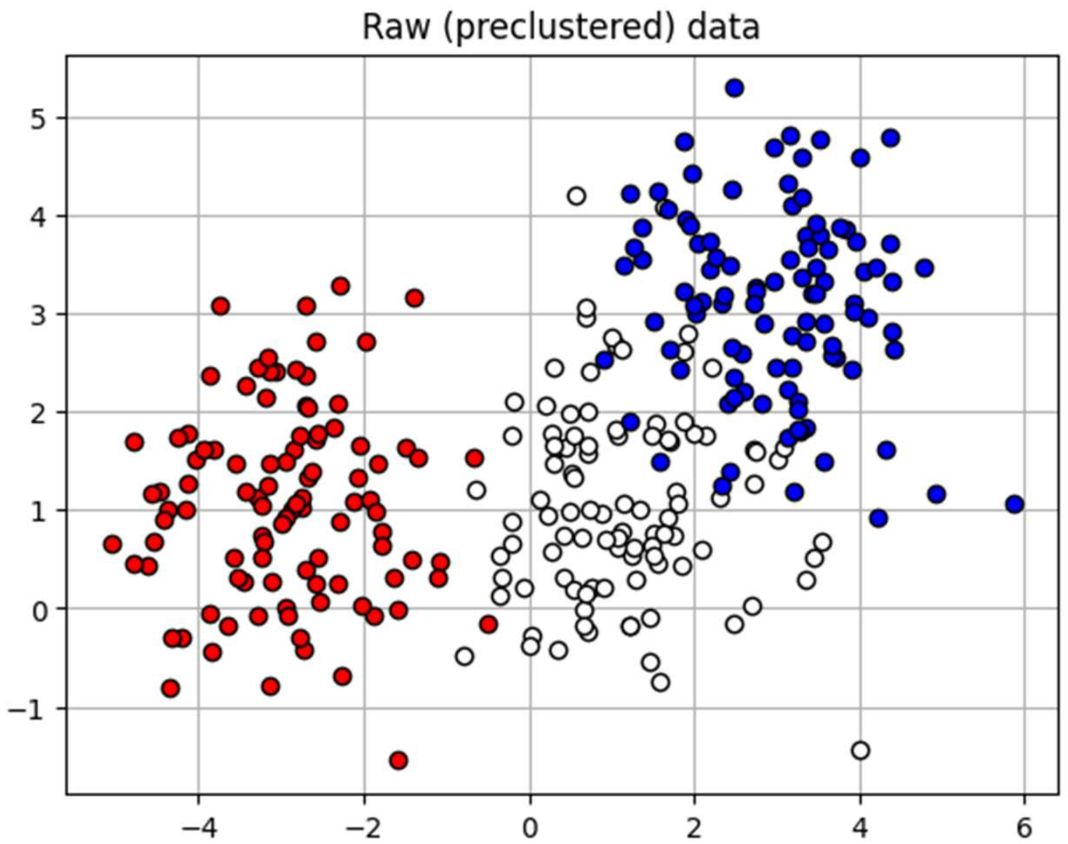
- np.concatenate()의 2번째 인자가 axis=0 이면 행방향, axis=1이면 열방향으로 배열을 합침
- blur 값을 변경해 생성 데이터 값 확인



blur = 0.1 　 blur = 0.5 　 blur = 1

실습3

■ 중심점에 따른 데이터 생성을 다른 색으로 표기



SECTION 03-3 *k*-평균 클러스터링(4)

[3 단계] – 랜덤 클러스터 중심을 k개 생성 (중복X)

- np.random.choice()로 랜덤하게 생성한 데이터 100개 중 3개를 추출
- 추출 시 중복되지 않도록 replace=False로 설정
- replace=True의 경우 중복 추출되면, default 값임

```
## 랜덤 클러스터 중심 초기화
k = 3

# 랜덤 클러스터 중심 생성
ridx = np.random.choice(range(len(data)), k, replace=False)
# replace=False는 중복되지 않도록 추출함, default: replace=True

centroids = data[ridx,:]
```

SECTION 03-3 *k*-평균 클러스터링(5)

[4 단계] – 데이터와 랜덤 클러스터 중심 그리기

- plt.subplots()는 여러 개의 그림을 모아서 출력
- flatten()함수는 멀티 배열 형태를 1차 배열로 변환

```
# 그래프 설정
fig,axs = plt.subplots(2,2,figsize=(6,6))
axs = axs.flatten() #1차 배열로 펴기
lineColors = [ [1,0,0],[0,1,0],[0,0,1] ] #'rgb'

# 데이터와 랜덤 클러스터 중심 그리기
axs[0].plot(data[:,0],data[:,1],'ko',markerfacecolor='w')
axs[0].plot(centroids[:,0],centroids[:,1],'ko')
axs[0].set_title('Iteration 0')
#axs[0].set_xticks([])
#axs[0].set_yticks([])
plt.show()
```

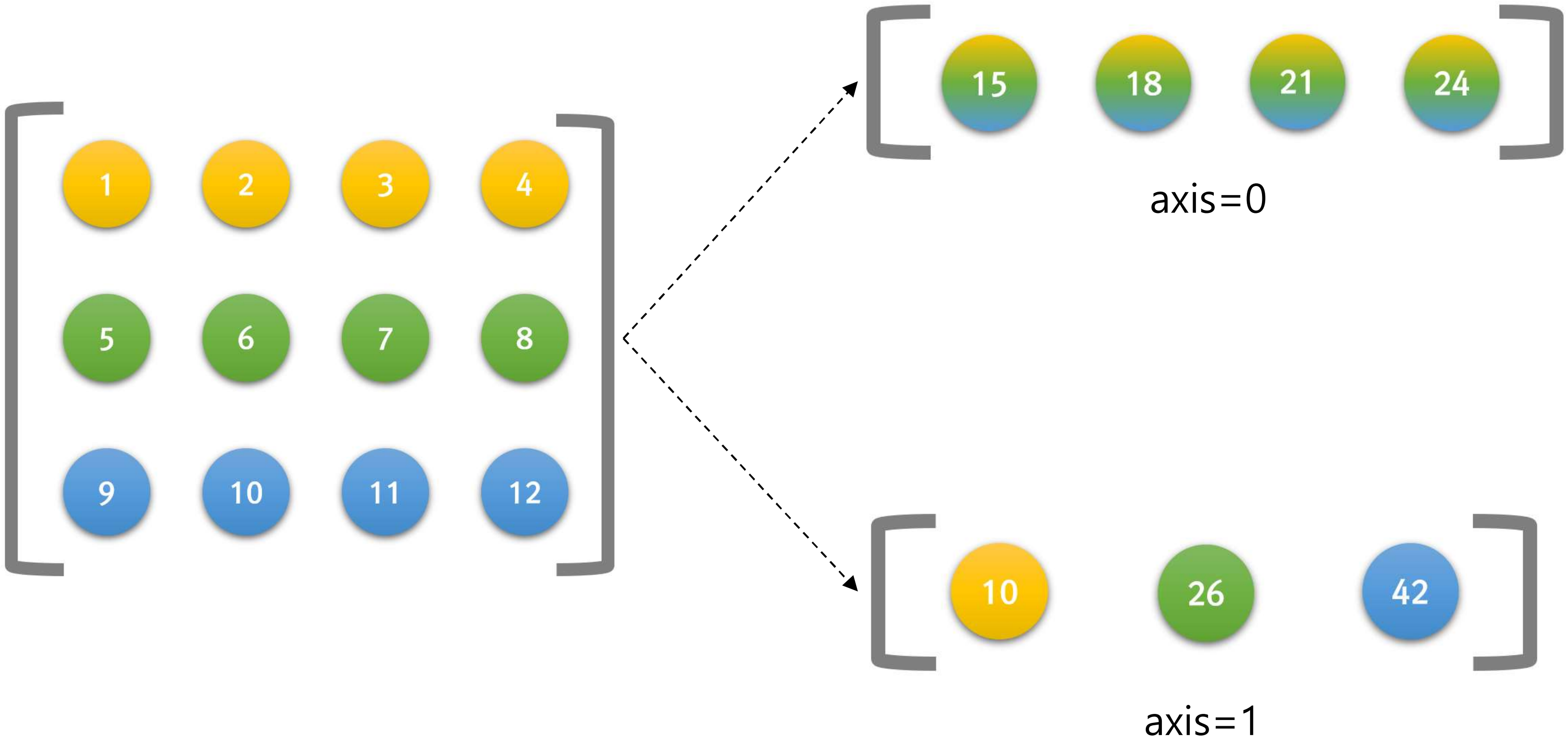
SECTION 03-3 *k*-평균 클러스터링(6)

[5 단계] – k-평균 클러스터링

- 3번 반복하면서 데이터의 클러스터를 분류하고, 클러스터 중심점을 다시 계산
- np.argmin()는 가장 작은 원소의 인덱스를 반환

```
# loop over iterations
for iteri in range(3):

  # 거리 계산
  dists = np.zeros((data.shape[0],k)) #(300, 3)
  for ci in range(k):
    dists[:,ci] = np.sum((data-centroids[ci,:])**2, axis=1) #브로드캐스팅 연산

  # 가장 가까운 클러스터 중심점 찾기
  groupidx = np.argmin(dists, axis=1)

  # 클러스터 중심점 다시 계산 (클러스터의 평균)
  for ki in range(k):
    centroids[ki,:] = [ np.mean(data[groupidx==ki,0]), np.mean(data[groupidx==ki,1]) ]

  # plot data points
  for i in range(len(data)):
    axs[iteri+1].plot([ data[i,0],centroids[groupidx[i],0] ],[ data[i,1],centroids[groupidx[i],1]
],color=lineColors[groupidx[i]])

  axs[iteri+1].plot(centroids[:,0],centroids[:,1],'ko')
  axs[iteri+1].set_title(f'Iteration {iteri+1}')
  axs[iteri+1].set_xticks([])
  axs[iteri+1].set_yticks([])
```

np.sum(), axis=0 or 1



SECTION 03-3 *k*-평균 클러스터링(6)

[5 단계] – k-평균 클러스터링

- data[groupidx==0,0]는 data의 x축의 값 만을 반환
- 이때 groupidx의 인덱스 값에 따라 0 인덱스의 값 만을 반환

```
# loop over iterations
for iteri in range(3):

  # 거리 계산
  dists = np.zeros((data.shape[0],k)) #(300, 3)
  for ci in range(k):
    dists[:,ci] = np.sum((data-centroids[ci,:])**2, axis=1) #브로드캐스팅 연산

  # 가장 가까운 클러스터 중심점 찾기
  groupidx = np.argmin(dists, axis=1)

  # 클러스터 중심점 다시 계산 (클러스터의 평균)
  for ki in range(k):
    centroids[ki,:] = [ np.mean(data[groupidx==ki,0]), np.mean(data[groupidx==ki,1]) ]

  # plot data points
  for i in range(len(data)):
    axs[iteri+1].plot([ data[i,0],centroids[groupidx[i],0] ],[ data[i,1],centroids[groupidx[i],1]
],color=lineColors[groupidx[i]])

  axs[iteri+1].plot(centroids[:,0],centroids[:,1],'ko')
  axs[iteri+1].set_title(f'Iteration {iteri+1}')
  axs[iteri+1].set_xticks([])
  axs[iteri+1].set_yticks([])
```