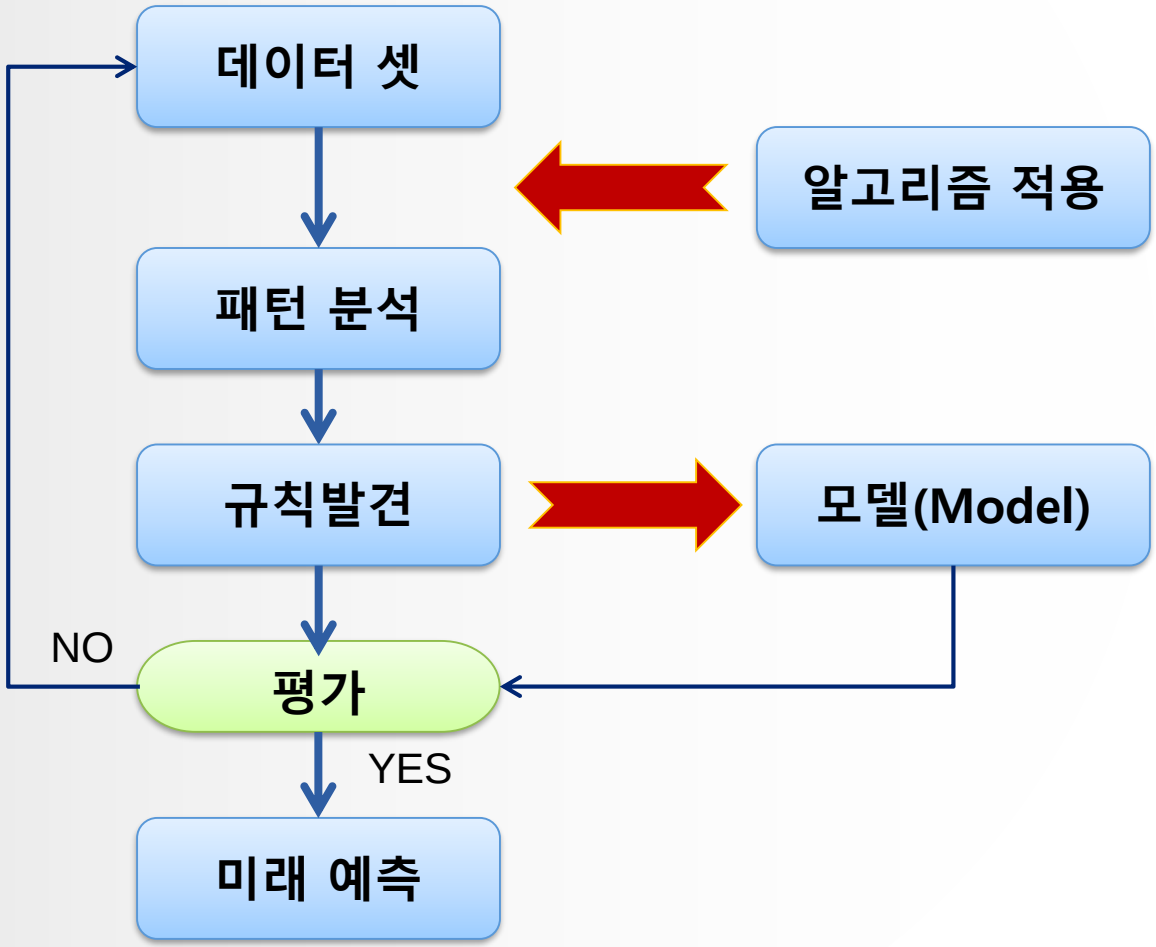
Chap10.
군집예측 모형

작성자 : 김진성

목차

1. 비지도학습(unSupervised Learning) 절차

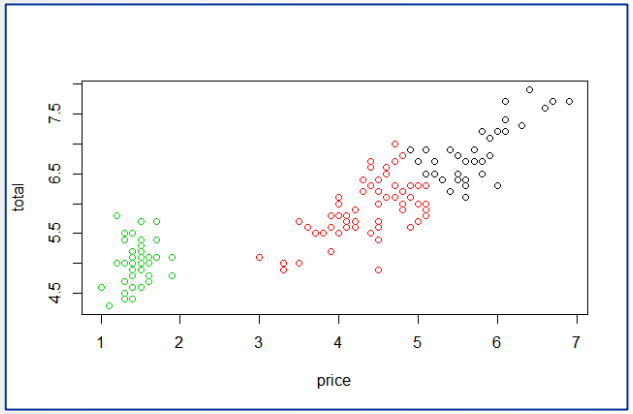


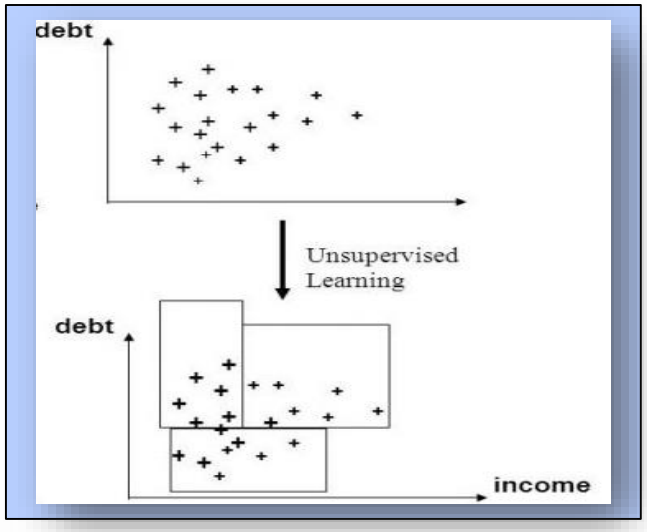
2. 군집 분석

- **유사도가 높은 데이터 군집화**
  - 유사도(유클리드안 거리식)가 높은 데이터끼리 그룹화
  - 계층형 클러스터링과 비계층형 클러스터링으로 분류

✓ K-means : 비계층적 군집분석

✓ Hierarchical : 계층적 군집분석





군집화 알고리즘 예

1) 군집 분석 개요

- **군집분석 특징**
  - 종속변수(y변수)가 없는 데이터 마이닝 기법
  - 유클리드 거리 기반 유사 객체 묶음
  - 고객 DB -> 알고리즘 적용 -> 패턴 추출(rule) -> 근거리 모형으로 군집형성
  - 계층적 군집분석(탐색적), 비계층적 군집분석(확인적)
  - 주요 알고리즘 : k-means, hierarchical

1) 군집 분석 개요

- **군집분석 특징**
  - 전체적인 데이터 구조를 파악하는데 이용
  - 관측대상 간 유사성을 기초로 비슷한 것 끼리 그룹화(Clustering)
  - 유사성 = 유클리드 거리
  - 분석결과에 대한 가설 검정 없음(타당성 검증 방법 없음)
  - 분야 : 사회과학, 자연과학, 공학 분야
  - 척도 : 등간, 비율척도(연속적인 양)

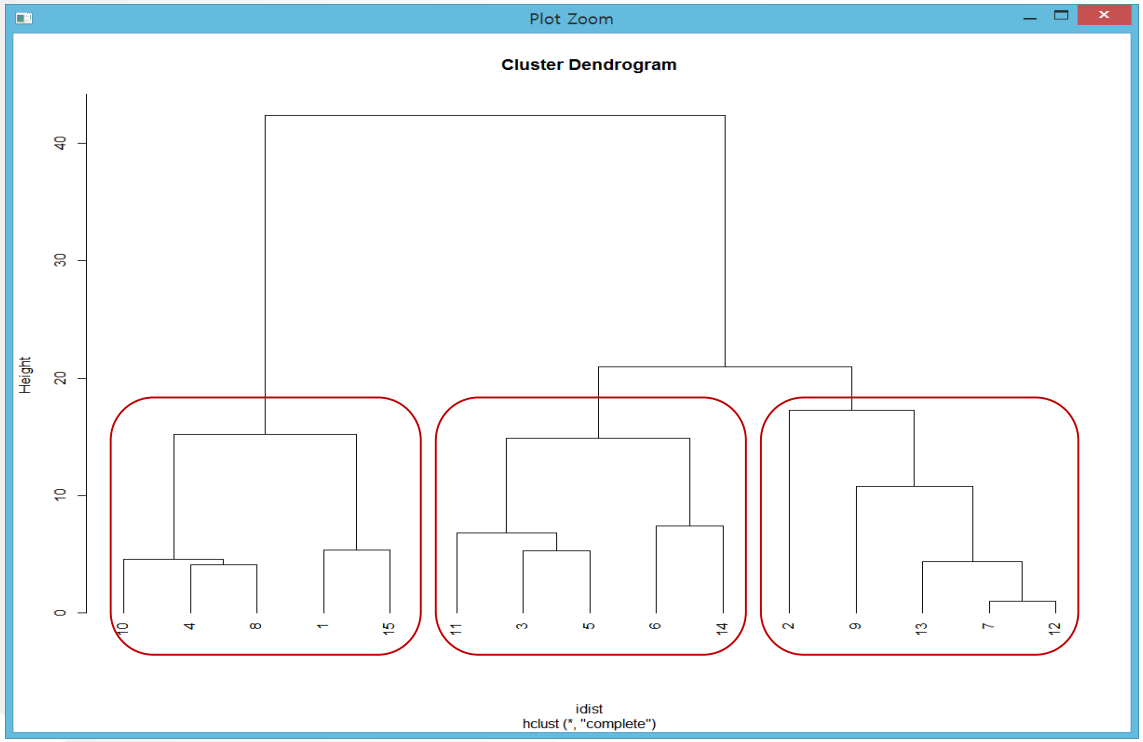
- **유클리드 거리 계산식**

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

❖ 관측대상 p와 q의 대응하는 변량값의 차가 작으면, 두 관측대상은 유사하다고 정의하는 식

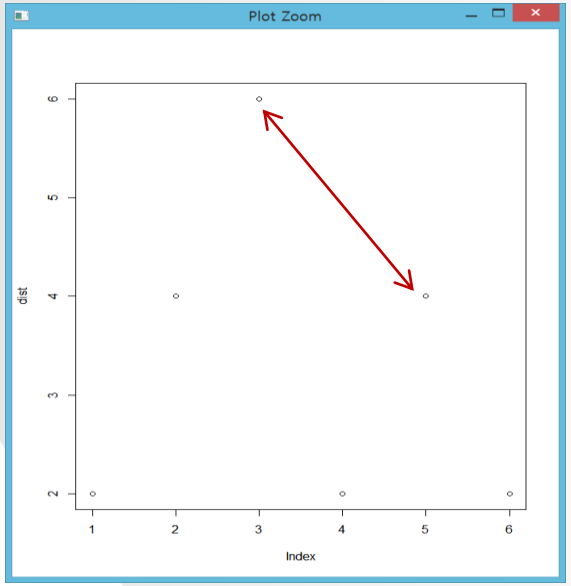
1) 군집 분석 개요

- **군집 분석 결과**
  - 평균결합방식을 적용한 덴드로그램(Dendrogram)
  - 가로축 : 학생번호, 세로축 : 상대적 거리
  - 군집수는 사용자가 정할 수 있음(2집단, 3집단 등)



2) 유클리드 거리

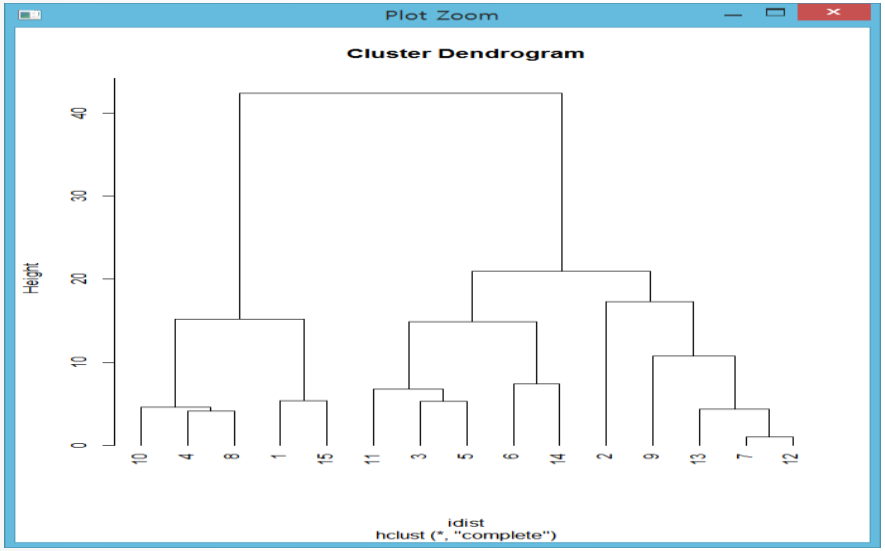
- **유클리드 거리(Euclidean distance)**
  - 두 점 사이의 거리를 계산하는 방법
  - 이 거리를 이용하여 유클리드 공간 정의



$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

3) 계층적 군집 분석

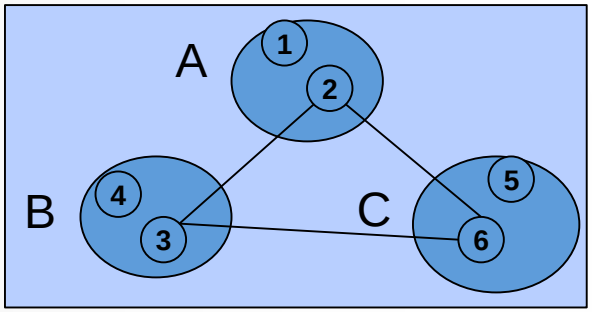
- **계층적 군집분석**
  - 유클리드 거리를 이용한 군집분석 방법
  - 계층적(hierarchical)으로 군집 결과 도출
  - 탐색적 군집분석



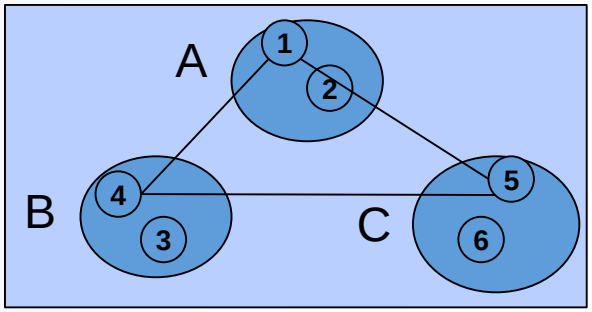
3) 계층적 군집 분석

군집화 방식

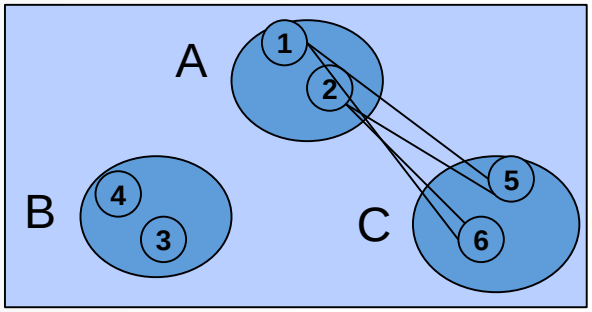
1. 단일기준결합방식 :
 각 군집에서 중심으로부터 거리가 가까운 것(2,3,6) 1개씩 비교하여 가장 가까운 것 끼리 군집화
2. 완전기준결합방식 :
 각 군집에서 중심으로부터 가장 먼 대상(1,4,5) 끼리 비교하여 가장 가까운 것 끼리 군집화
3. 평균기준결합방식 :
 한 군집 안에 속해 있는 모든 대상과 다른 군집에 속해있는 모든 대상의 쌍 집합에 대한 거리를 평균 계산하여 가장 가까운 것 끼리 군집화
(1 -> 5,6 평균, 2 -> 5, 6 평균)



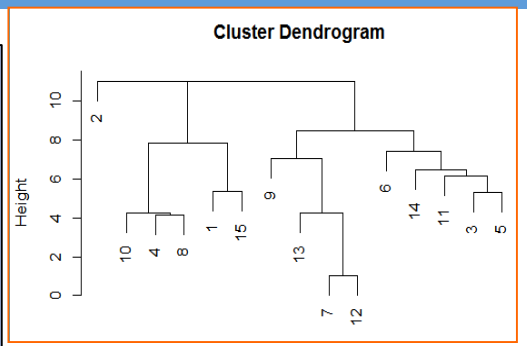
단일기준결합방식

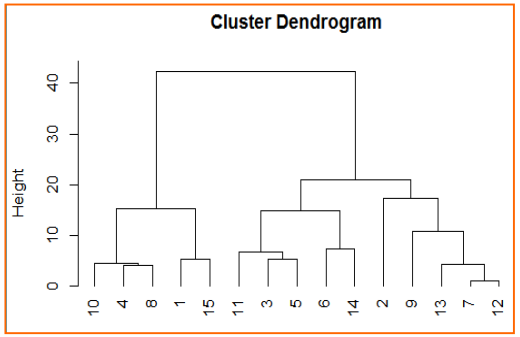


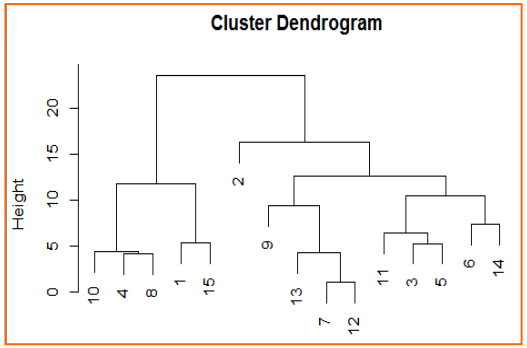
완전기준결합방식



평균기준결합방식



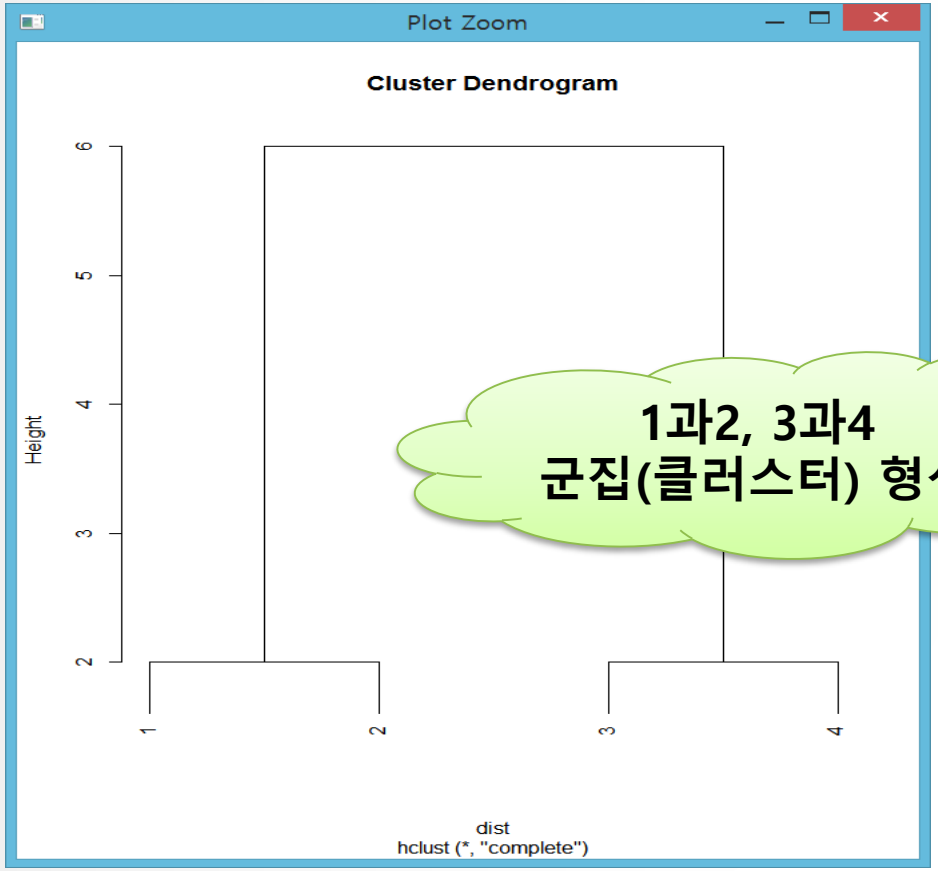




덴드로그램

3) 계층적 군집 분석

- 계층적 군집분석 결과 : 벤드로그램(dendrogram)



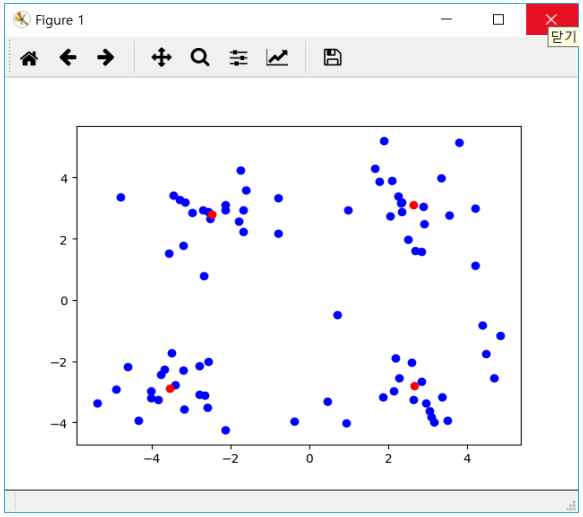
1과2, 3과4
군집(클러스터) 형성

4) 비 계층적 군집 분석

- **비계층적 군집 분석(k-means)**
  - 확인적 군집분석 방법
  - 계층적 군집분석법 보다 속도 빠름
  - 군집의 수를 알고 있는 경우 이용
  - K는 미리 정하는 군집 수
  - 계층적 군집화의 결과에 의거하여 군집 수 결정
  - 변수 보다 관측대상 군집화에 많이 이용
  - 군집의 중심(Cluster Center) 사용자가 정함

4) 비 계층적 군집 분석

- **k-평균 군집분석 알고리즘**
  - 단계1. k값을 초기값으로 k개 centroid 선정
  - 단계2. 각 데이터 포인터를 가장 가까운 centroid에 할당
  - 단계3. centroid에 할당된 모든 데이터 포인트의 중심 위치 계산(centroid 재조정)
  - 단계4. 재조정된 centroid와 가장 가까운 데이터 포인트 할당
  - 단계5. centroid 재조정이 발생되지 않을 때 까지 (or 지정한 수) 3~4단계 반복



kMeans 알고리즘 예

```
# data set 생성 함수
def loadDataSet(fileName):
    dataMat = []
    fr = open(fileName)
    for line in fr.readlines():
        curLine = line.strip().split('₩t')

        # float casting -> list append
        blankList = [] # 빈 list
        for data in curLine :
            fltCasting = float(data)
            blankList.append(fltCasting)

        dataMat.append(blankList)
    return dataMat

# data set 생성
dataSet = mat(loadDataSet('data/testSet.txt'))
print(dataSet)
print(type(dataSet))
print(shape(dataSet))
# dataMat 각 column의 최소값/최대값
print('min =', min(dataSet[:, 0]))
print('max =', max(dataSet[:, 0]))
```

```
[[ 1.658985  4.285136]
 [-3.453687  3.424321]
 [ 4.838138 -1.151539]
 [-5.379713 -3.362104]
 [ 0.972564  2.924086]
 [-3.567919  1.531611]
 [ 0.450614 -3.302219]
 [-3.487105 -1.724432]
 [ 2.668759  1.594842]
 [-3.156485  3.191137]
 [ 3.165506 -3.999838]
 [-2.786837 -3.099354]
 [ 4.208187  2.984927]
 [-2.123337  2.943366]
 [ 0.704199 -0.479481]
 [-0.39237  -3.963704]
 [ 2.831667  1.574018]
 [-0.790153  3.343144]
 [ 2.943496 -3.357075]
 [-3.195883 -2.283926]
 [ 2.336445  2.875106]
 [-1.786345  2.554248]
 [ 2.190101 -1.90602 ]
 [-3.403367 -2.778288]
 [ 1.778124  3.880832]
 [-1.688346  2.230267]
 [ 2.592976 -2.054368]
 [-4.007257 -3.207066]
 [ 2.257734  3.387564]
 [-2.679011  0.785119]
 [ 0.939512 -4.023563]
 [-3.674424 -2.261084]
 [ 2.046259  2.735279]
 [-3.18947   1.780269]
 [ 4.372646 -0.822248]
 [-2.579316 -3.497576]
 [ 1.889034  5.1904  ]
 [-0.798747  2.185588]
 [ 2.83652  -2.658556]
 [-3.837877 -3.253815]
 [ 2.096701  3.886007]
 [-2.709034  2.923887]
 [ 3.367037 -3.184789]
 [-2.121479 -4.232586]
 [ 2.329546  3.179764]
 [-3.284816  3.273099]
 [ 3.091414 -3.815232]
 [-3.762093 -2.432191]
 [ 3.542056  2.778832]
 [-1.736822  4.241041]
 [ 2.127073 -2.98368 ]
 [-4.323818 -3.938116]
 [ 3.792121  5.135768]
 [-4.786473  3.358547]
 [ 2.624081 -3.260715]
 [-4.009299 -2.978115]
 [ 2.493525  1.96371 ]
 [-2.513661  2.642162]
 [ 1.864375 -3.176309]
 [-3.171184 -3.572452]
 [ 2.89422   2.489128]
 [-2.562539  2.884438]
 [ 3.491078 -3.947487]
 [-2.565729 -2.012114]
 [ 3.332948  3.983102]
 [-1.616805  3.573188]
 [ 2.280615 -2.559444]
 [-2.651229 -3.103198]
 [ 2.321395  3.154987]
 [-1.685703  2.939697]
 [ 3.031012 -3.620252]
 [-4.599622 -2.185829]
 [ 4.196223  1.126677]
 [-2.133863  3.093686]
 [ 4.668892 -2.562705]
 [-2.793241 -2.149706]
 [ 2.884105  3.043438]
 [-2.967647  2.848696]
 [ 4.479332 -1.764772]
 [-4.905566 -2.91107 ]]
```

<class 'numpy.matrixlib.defmatrix.**matrix**'>
(80, 2)
min = [[-5.379713]]
max = [[4.838138]]

```
# 유클리드안 거리 계산 함수
def distEclud(vecA, vecB):
    return sqrt(sum(power(vecA - vecB, 2)))

# 첫번째 행과 두번째 행 거리계산 예
print('distEclud:',distEclud(dataSet[0], dataSet[1]))
# distEclud: 5.18463281668
```

<<첫번째 행과 두번째 행 data>>

[1.658985 4.285136]
 [-3.453687 3.424321]

<<유클리드안 거리 계산 절차>>

x - y= [5.112672 0.860815]

(x - y)^2= [26.13941498 0.74100246]

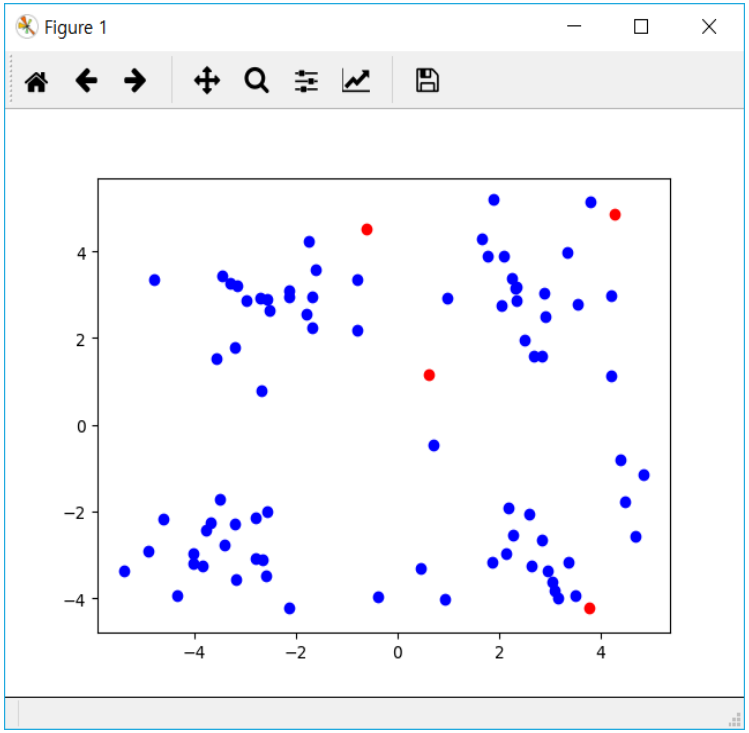
sum(x - y)^2= 26.8804174438

sqrt(sum(x - y)^2))= 5.18463281668

```
# cluster 중심 생성 함수
def randCent(dataSet, k):
    n = shape(dataSet)[1] # data set column 수
    centroids = mat(zeros((k,n)))#create centroid mat->(2,2)

    # cluster 중심 생성 : 각 차원의 범위 내에서 군집의 중심 random 생성
    for j in range(n):
        print('j=', j)
        minJ = min(dataSet[:,j])
        rangeJ = float(max(dataSet[:,j]) - minJ)
        print('rangeJ:',rangeJ)

        r = random.rand(k,1) # (2,1)
        centroids[:,j] = mat(minJ + rangeJ * r)
        print('after centroids :\n', centroids)

    return centroids

print('randCent')
centroids = randCent(dataSet, 4) # k=4
```

```
j= 0
rangeJ: 10.217851
after centroids :
 [[ 3.98461833  0.        ]
 [-4.31874837  0.        ]
 [-1.33869911  0.        ]
 [-0.34828341  0.        ]]
j= 1
rangeJ: 9.422986000000002
after centroids :
 [[ 3.98461833  4.59333469]
 [-4.31874837  3.22459462]
 [-1.33869911  2.28310017]
 [-0.34828341 -3.2559881 ]]
```

```
# cluster centroid 산점도 그리기

# raw data
x_data = [x[:,0] for x in dataSet] # 1칼럼
y_data = [x[:,1] for x in dataSet] # 2칼럼

flg = plt.figure() # 차트 플롯 생성
chart = flg.add_subplot(1,1,1) # 행,열,위치
chart.scatter(x_data, y_data, c='b')

# centroid data
x_cent = [x[:,0] for x in centroids] # 1칼럼
y_cent = [x[:,1] for x in centroids] # 2칼럼
chart.scatter(x_cent, y_cent, c='r')
plt.show()
```



```python
# k 평균 군집화 알고리즘
def kMeans(dataSet, k, cnt = 1):
    m = shape(dataSet)[0] # 80
    clusterAssment = mat(zeros((m, 2))) # (80, 2) - centroid 저장 matrix
    centroids = randCent(dataSet, k) # cluster 중심 생성(0~3)
    clusterChanged = True
    while clusterChanged:
        clusterChanged = False
        # 각 점을 가장 가까운 중심에 할당
        for i in range(m): # 0 ~ 80
            minDist = inf; minIndex = -1
            for j in range(k): # 0 ~ 3
                # 4개의 centroids와 i번째 dataSet 거리 계산
                distJI = distEclud(centroids[j,:], dataSet[i,:])
                # 가장 거리가 가까운 값으로 minDist, minIndex 수정
                if distJI < minDist:
                    minDist = distJI; minIndex = j
            # 기존 centroid와 평균으로 수정된 centroid가 다른경우
            if clusterAssment[i,0] != minIndex : # 반복 조건
                clusterChanged = True
            # cluster의 centroid 저장 : [minIndex, 최소거리제곱]
            clusterAssment[i, :] = minIndex, minDist**2

        # cluster 중심 다시 계산
        for center in range(k): # 0 ~ 3
            # 동일한 cluster data set 가져오기
            ptsInClust = dataSet[nonzero(clusterAssment[:,0]==center)[0]]
            # 동일 cluster의 칼럼 평균으로 중심(centroid) 수정
            centroids[center,:] = mean(ptsInClust, axis=0) # 칼럼 합계
    return centroids, clusterAssment
```

80 x 4 = 320반복
Data set 전체를 대상으로 4개의 centroid와 비교하여 가장 가까운 centroid index를 clusterAssment에 저장

전체 관측치 대상 centroid 분류

clusterAssment

```
[[  2.          1.1363227 ]
 [  0.         15.10966678
```

centroids: 1

```
[[-4.8444312    3.60819483]
 [-3.69039561   3.45376638]
 [ 3.70771631  -0.863704   ]
 [ 4.20764089  -1.91250924]]
```

centroids: 2

```
[[ -5.08309300e+00   -1.77850000e-03]
 [ -2.65056492e+00    4.85776865e-01]
 [  2.24124913e+00    1.81565896e+00]
 [  2.68885006e+00   -3.07701372e+00]]
```

centroids: 3

```
[[-4.10367677  -2.40353962]
 [-2.49391804   1.1466944 ]
 [ 2.38384745   2.95623936]
 [ 2.7481024   -2.90572575]]
```

centroids: 4

```
[[-3.53973889  -2.89384326]
 [-2.46154315   2.78737555]
 [ 2.6265299    3.10868015]
 [ 2.65077367  -2.79019029]]
```

[해설] 4개의 중심(centroid)이 4회 반복하여 k-평균에 수렴된다.(결과는 random)

```
print('final centroids')
print(centroids)

print('clusterAssment')
print(clusterAssment)
```

final centroids

```
[[-3.53973889 -2.89384326]
 [-2.46154315  2.78737555]
 [ 2.6265299   3.10868015]
 [ 2.65077367 -2.79019029]]
```

최종 선정된 4개의 centroid 좌표

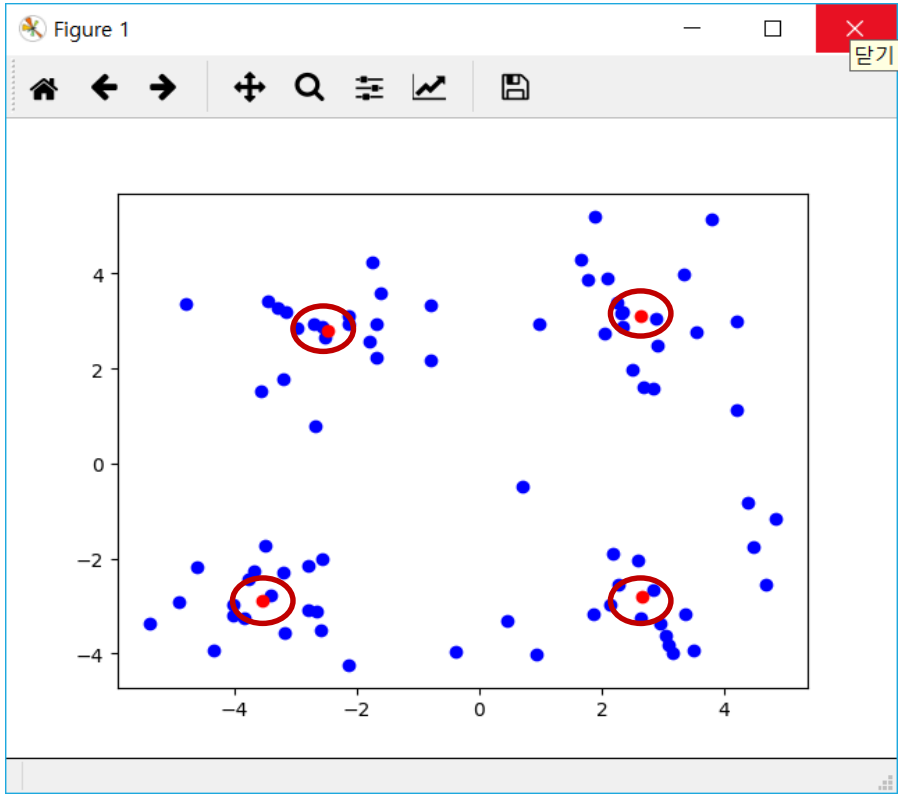
80개 관측치 별 소속 centroid 색인과 최소거래 제곱

clusterAssment

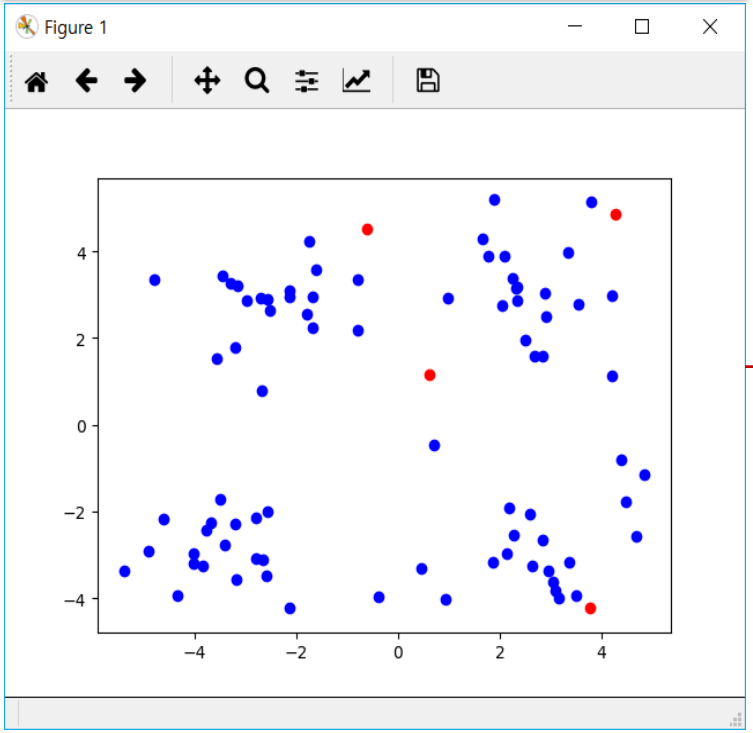
```
[[ 2.          1.1363227 ]
 [ 0.         15.10966678]
 [ 1.          1.77894835]
 [ 0.         13.05952193]
 [ 2.          0.67431094]
 [ 0.          4.06569593]
 [ 3.          3.33397996]
 [ 0.          1.67339718]
 [ 2.          3.51462445]
 [ 0.         13.31021684]
 [ 3.          1.13529043]
 [ 0.          7.18249411]
 [ 2.          6.15926403]
 [ 0.         12.78567924]
 [ 1.          8.36799659]
 [ 3.          7.40114722]
 [ 2.          3.9127034 ]
 [ 2.          6.40498401]
 [ 3.          0.45347939]
 [ 0.          3.34103091]
 [ 2.          0.47880971]
 [ 0.         11.15144227]
 [ 1.          2.22569954]
 [ 0.          5.42075933]
 [ 2.          0.43560221]
 [ 0.          9.59838408]
 [ 1.          1.55921974]
 [ 0.          8.16847865]
 [ 2.          0.29776667]
 [ 0.          1.84622208]
 [ 3.          2.14443412]
 [ 0.          3.45335301]
 [ 2.          0.33213566]
 [ 0.          5.0044968 ]
 [ 1.          0.89633092]
 [ 0.          9.67348248]
 [ 2.          3.89722376]
 [ 2.          7.50813432]
 [ 3.          0.89456762]
 [ 0.          8.19360567]
 [ 2.          0.56965913]
 [ 0.         11.6990146 ]
 [ 3.          1.25165147]
 [ 0.         15.49073083]
 [ 2.          0.35200481]
 [ 0.         13.91217217]
 [ 3.          0.82778395]
 [ 0.          4.18562902]
 [ 2.          3.45195472]
 [ 2.         13.11100262]
 [ 3.          0.20993966]
 [ 0.         13.31624491]
 [ 2.          7.88267073]
 [ 0.         16.97310858]
 [ 3.          0.14811802]
 [ 0.          6.96454337]
 [ 2.          2.15464654]
 [ 0.         10.11592038]
 [ 3.          0.22529984]
 [ 0.          9.71326666]
 [ 2.          1.874607  ]
 [ 0.         11.60829487]
 [ 3.          1.7643597 ]
 [ 0.          2.86292319]
 [ 2.          3.12381318]
 [ 2.         11.37636644]
 [ 3.          0.73728448]
 [ 0.          7.34171072]
 [ 2.          0.34513403]
 [ 2.         11.79980406]
 [ 3.          0.61568374]
 [ 0.          4.86369735]
 [ 1.          5.89919385]
 [ 0.         13.80717602]
 [ 1.          3.20595858]
 [ 0.          3.05895604]
 [ 2.          1.3460239 ]
 [ 0.          0.99318189]
 [ 1.          1.26564187]
 [ 0.          8.82913224]]
```

```
# cluster 산점도 그리기
print('x y scatter plotting')
x_data = [x[:,0] for x in dataSet] # 1칼럼

y_data = [x[:,1] for x in dataSet] # 2칼럼

flg = plt.figure() # 차트 플롯 생성
chart = flg.add_subplot(1,1,1) # 행,열,위치

chart.scatter(x_data, y_data, c='b')

print('x y scatter centroids plotting')
x_cent = [x[:,0] for x in centroids] # 1칼럼

y_cent = [x[:,1] for x in centroids] # 2칼럼
chart.scatter(x_cent, y_cent, c='r')
plt.show()
```

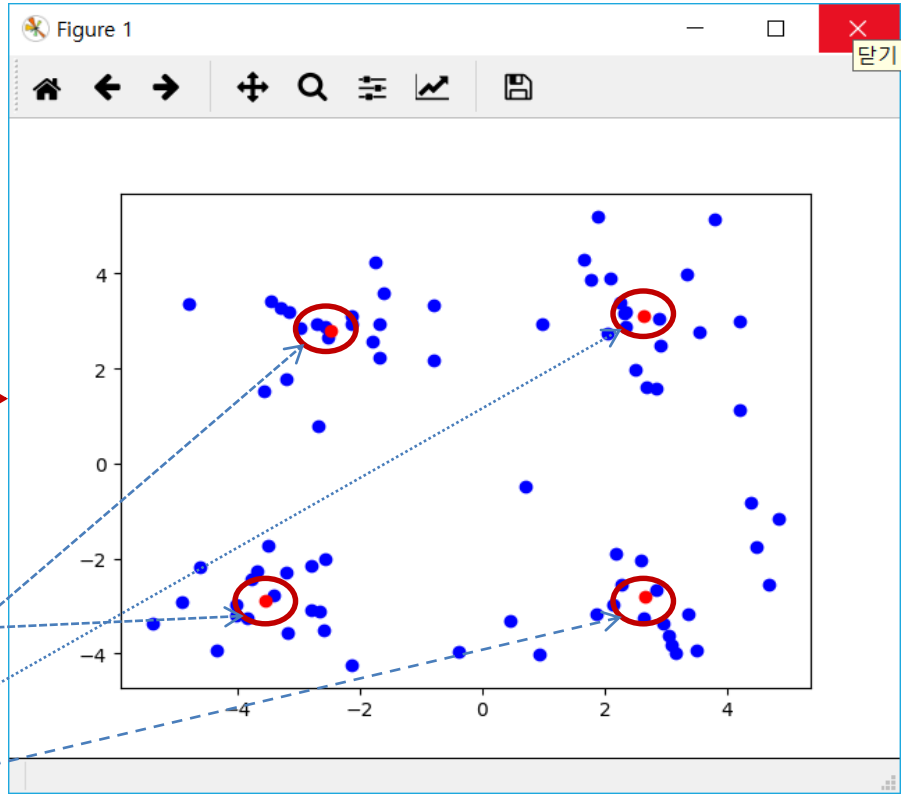
K=4 centroids



K=4 초기 random centroid



kMeans centroids

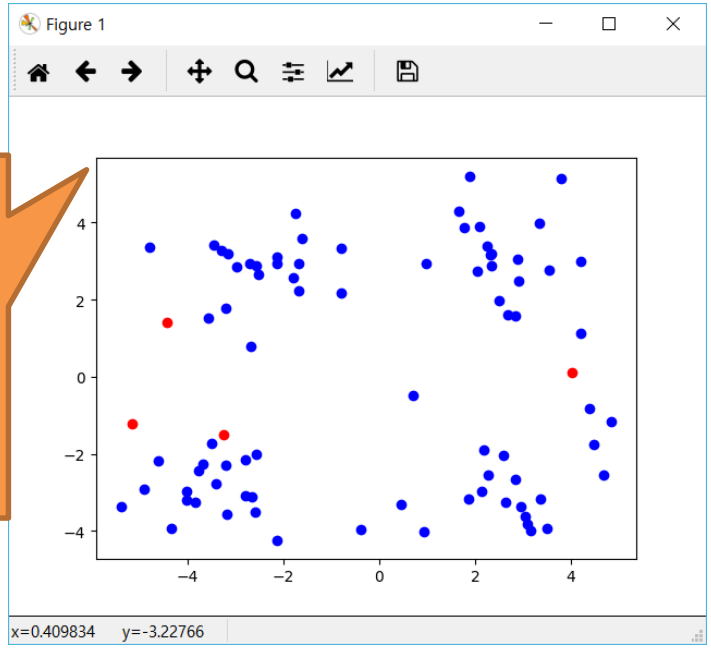


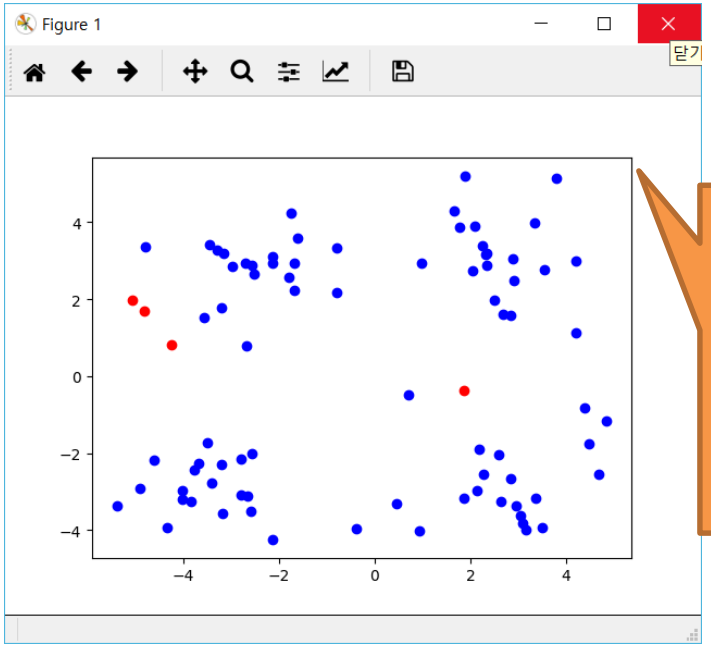

```
final centroids
[[-3.53973889 -2.89384326]
 [-2.46154315  2.78737555]
 [ 2.6265299   3.10868015]
 [ 2.65077367 -2.79019029]]
```

K 평균 수렴 과정

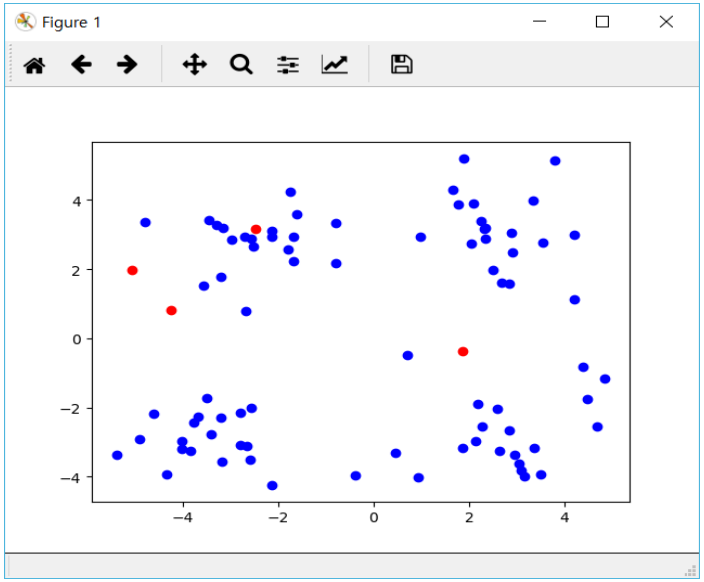
동일한 cluster(k개) 단위로 평균 계산하여 centroid 수정

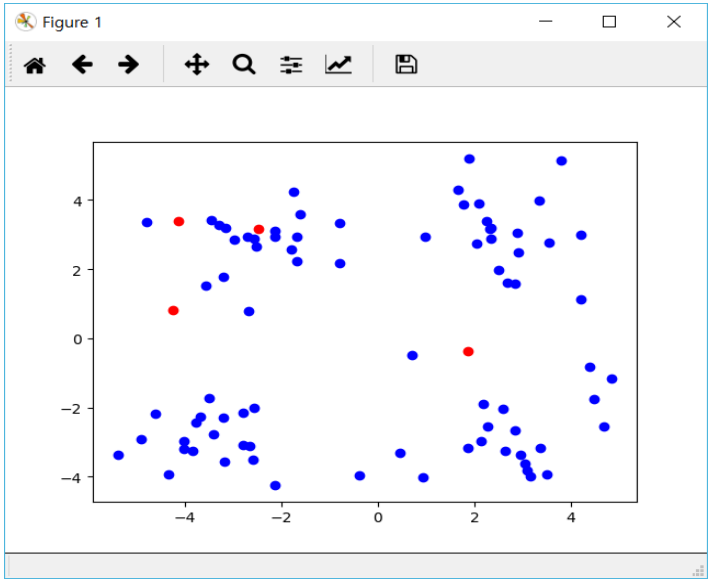
K=4 random 생성 가장 가까운 centroid 할당





동일 cluster 단위로 평균계산 centroid 수정





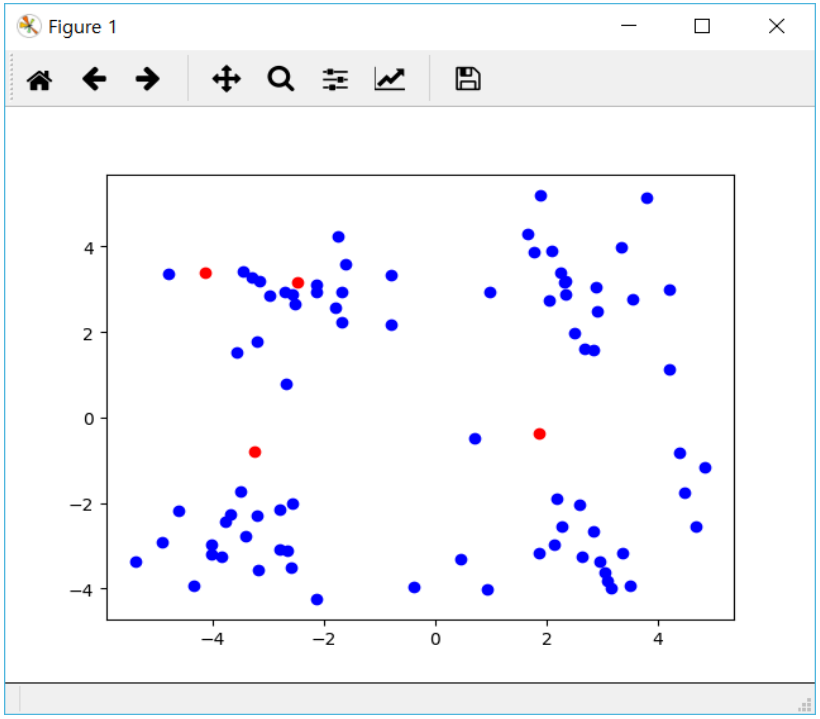


Figure 1
4
2
0
−2
−4
−4
−2
0
2
4

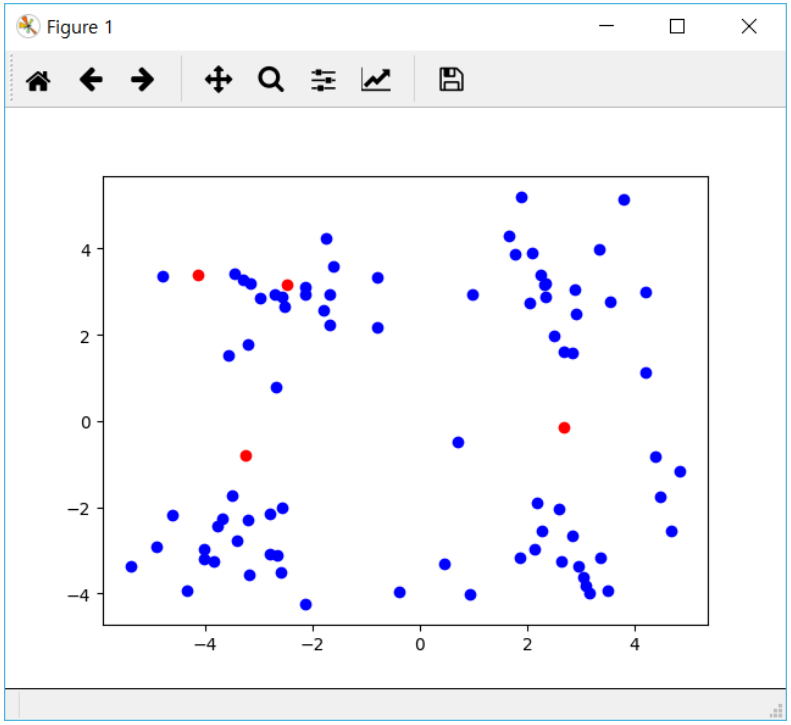


Figure 1
4
2
0
−2
−4
−4
−2
0
2
4

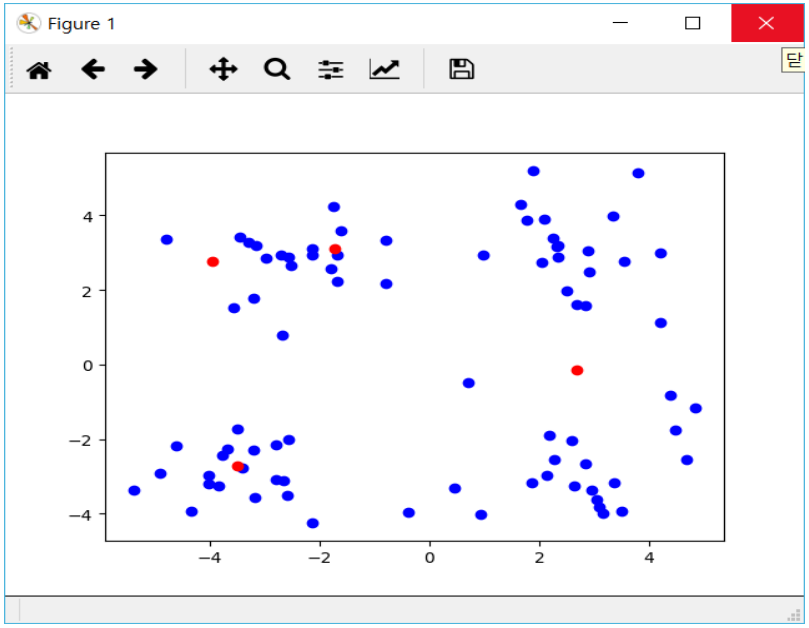


Figure 1
4
2
0
−2
−4
−4
−2
0
2
4

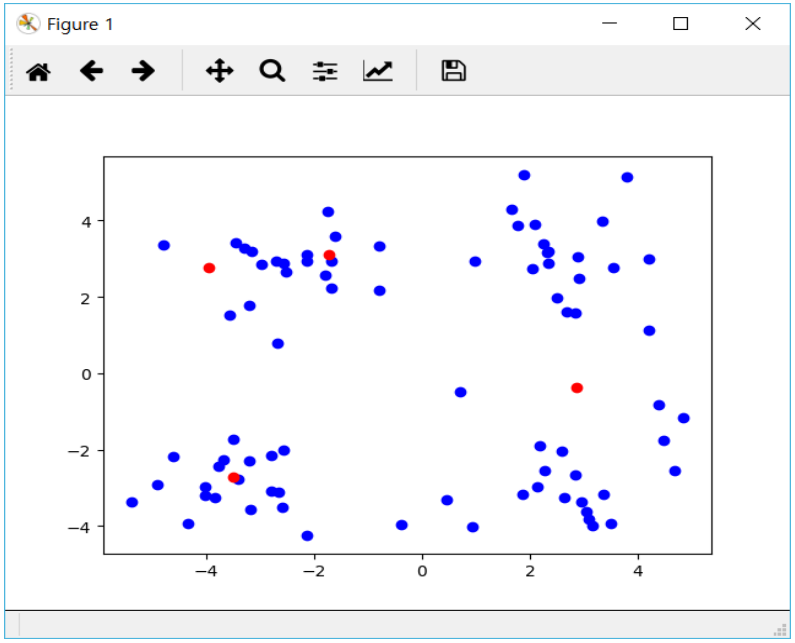


Figure 1
4
2
0
−2
−4
−4
−2
0
2
4

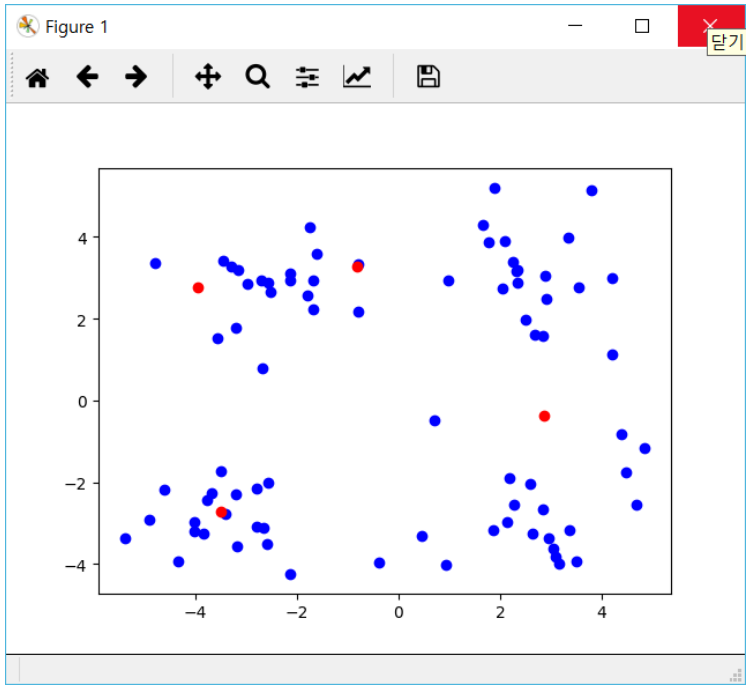


Figure 1
닫기

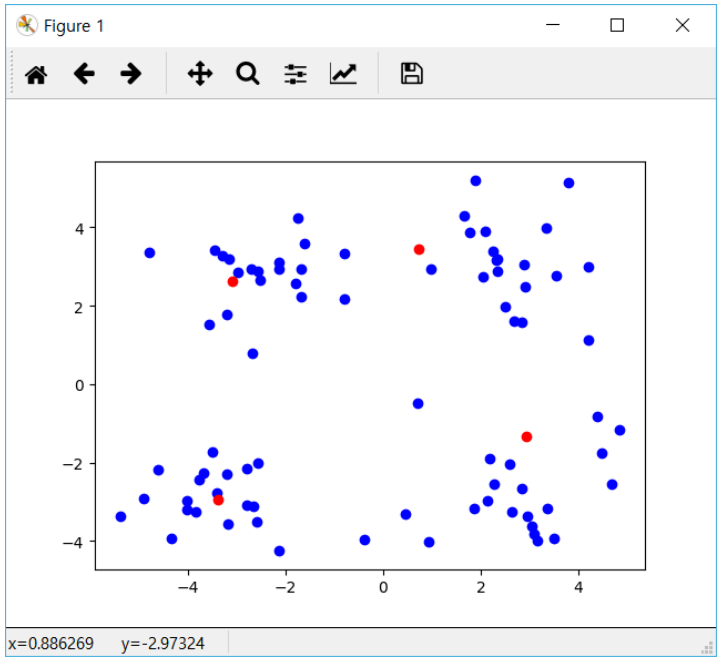


Figure 1
x=0.886269 y=-2.97324

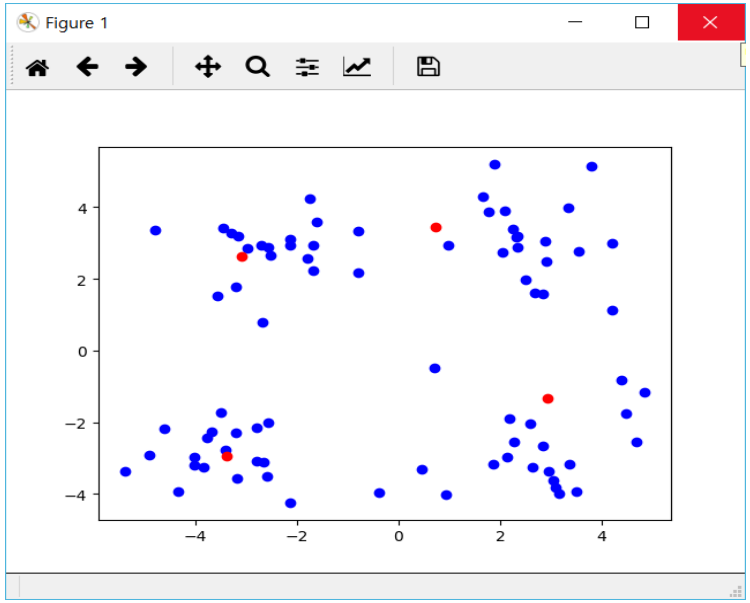


Figure 1

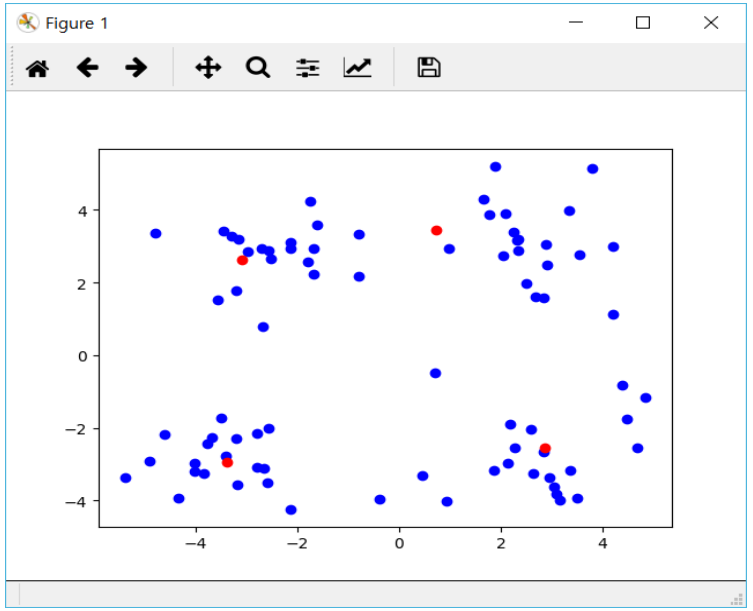
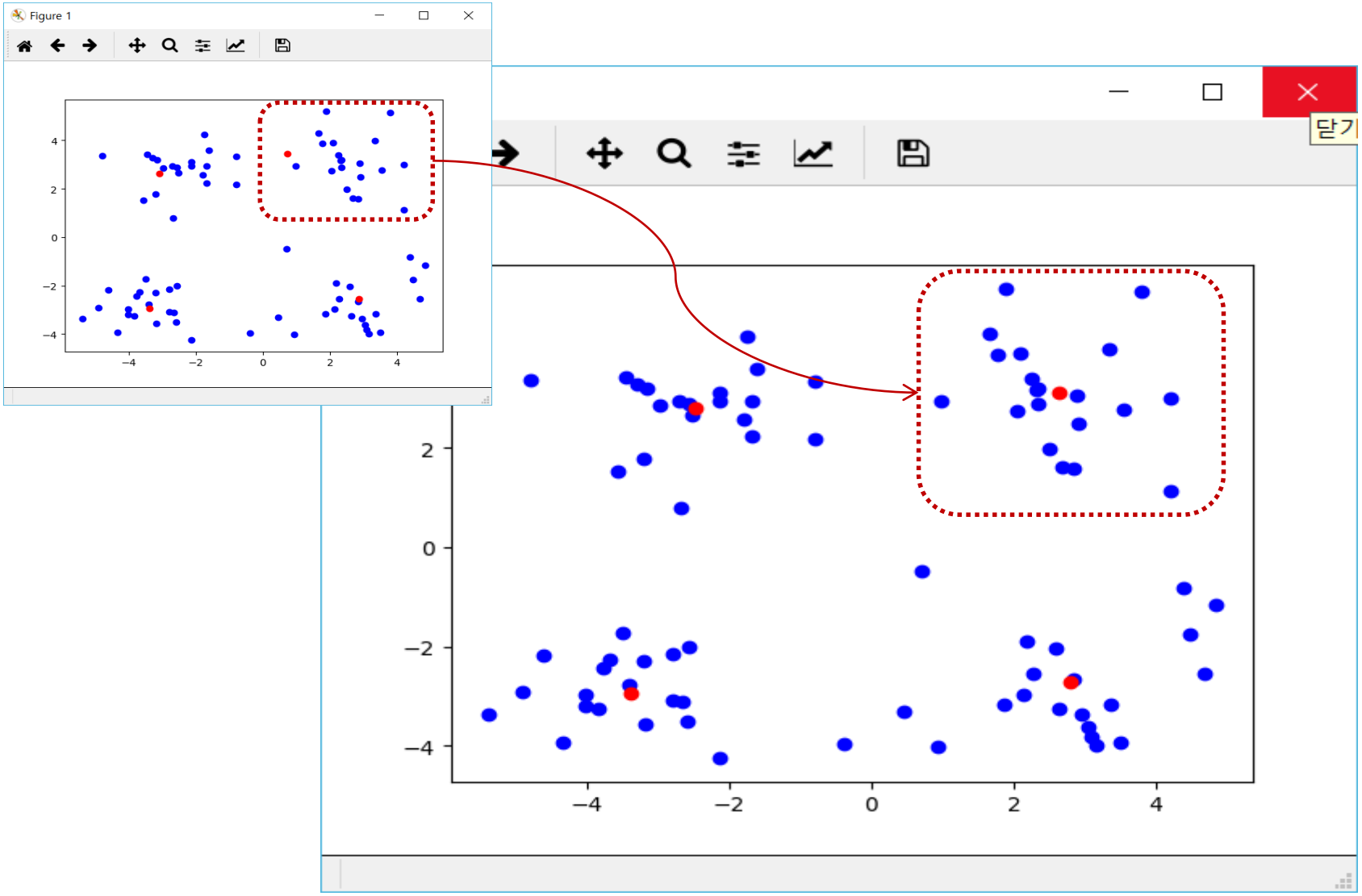


Figure 1

전체 관측 80개의 기존 centroid와 평균으로 계산된 신규 centroid가 동일한 경우 kMeans 완료됨



K=4인 경우 4개의 군집을 색으로 구분