**კლასტერიზაციის ალგორითმი**

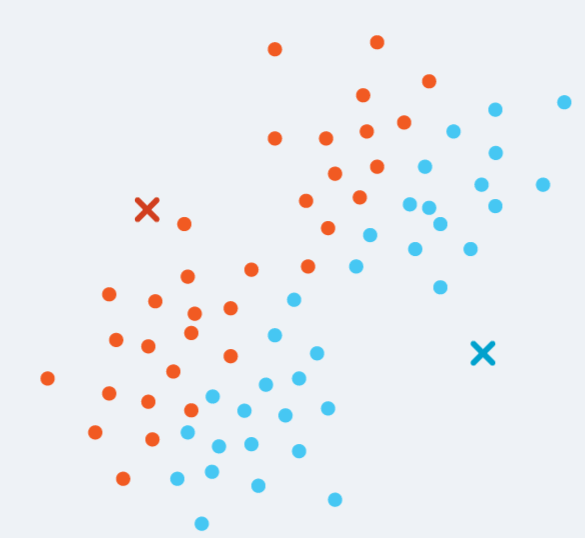
**(Unsupervised Learning)**

ვთქვათ მოცემული გვაქვს სიმრავლე ორგანზომილებიან სივრცეში

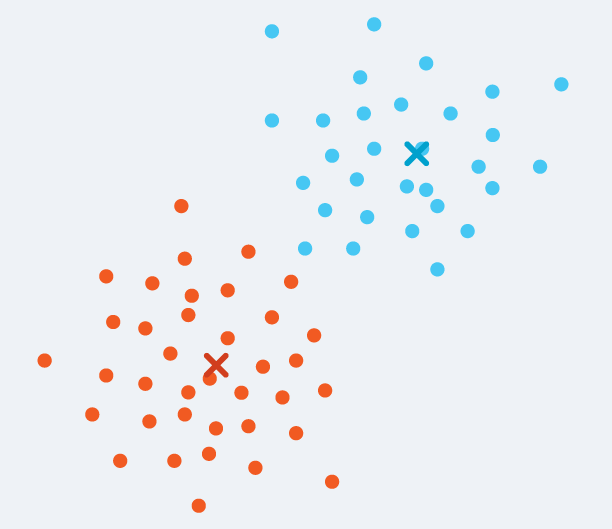
ჩვენი მიზანია დავაჯგუფოთ ისინი (ანუ სხვანაირად რომ ვთქვათ უნდა მოვახდინოთ მათი კლასტერიზაცია) წინასწარ განსაზღვრული ჯგუფების რაოდენობის მიხედვით.

(1) (2)

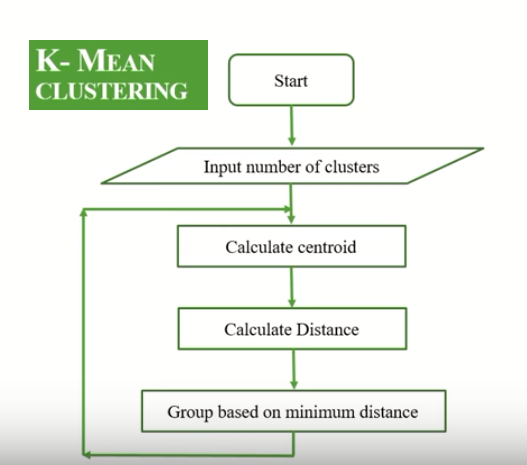




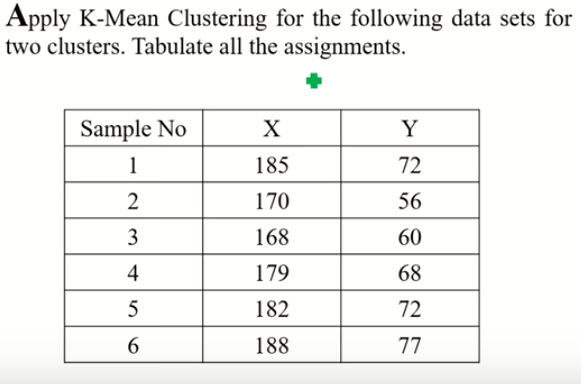
(3)



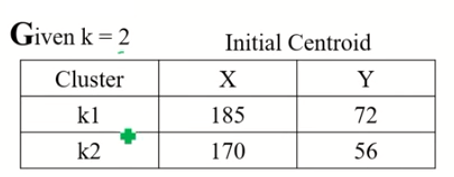
ალგორითმის აღწერა

****

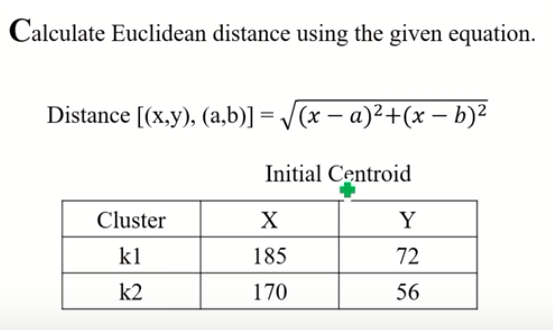
**განვიხილოთ შემდეგი მაგალითი:**

****

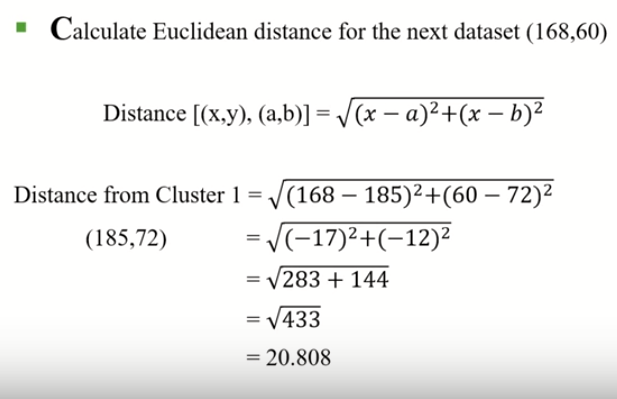
**ცენტროიდების როლში შემთხვევითად ავიღოთ 2 წერტილი**

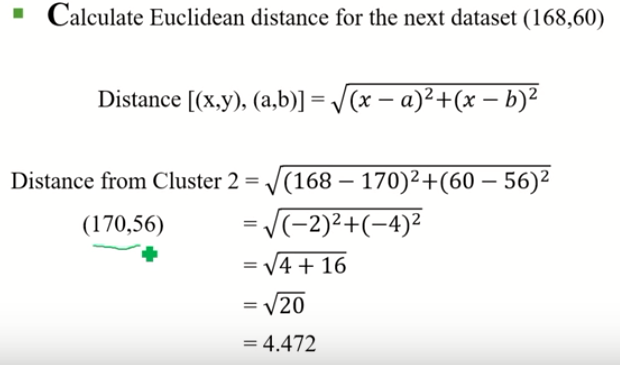
****

ცენტროიდებსა და ელემენტებს შორის მანძილი გამოვთვალოთ ევკლიდის სივრცის მაგალითზე

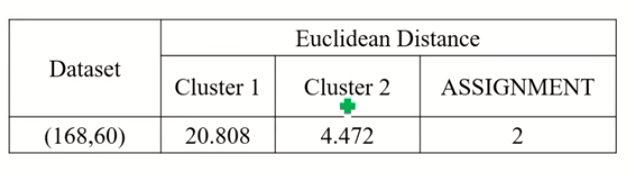
****

**ჩვენი მოცემული ყოველი ელემენტებისათვის დავთვალოთ მანძილები ცენტროიდებამდე და მივაკუთვნოთ იმ ცენტროიდს და შესაბამისად კლასტერს რომელ ცენტროიდამდეც მანძილი მინიმალურია.**

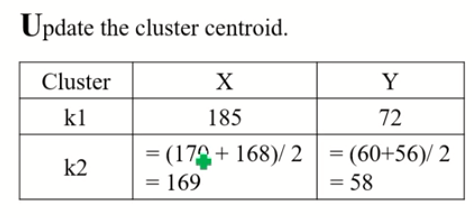
****

****

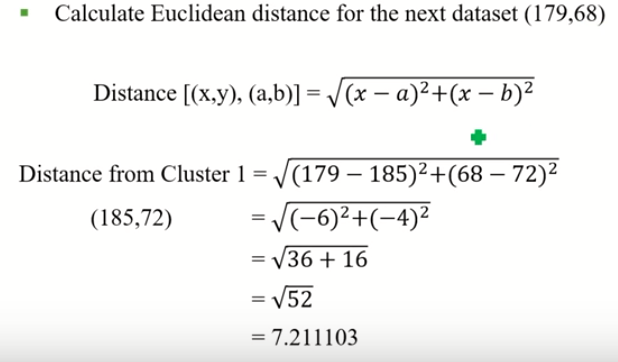
**მინიმალური მანძილი არის Cluster 2-მდე ამრიგად მივიღეთ შემდეგი:**

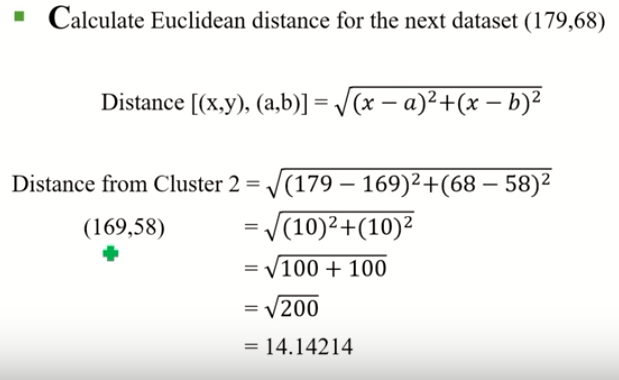
****

**ამის შემდეგ მოვახდინოთ ცენტროიდების ხელახალი გადათვლა:**

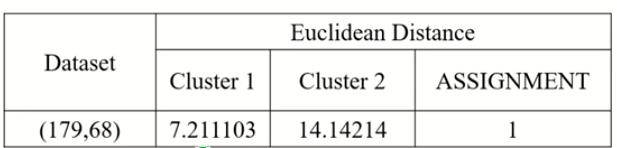
****

**ჩვენ დავთვალეთ ახალი ცენტროიდები. გადავიდეთ ეხლა შემდეგ წერტილზე (179, 68):**

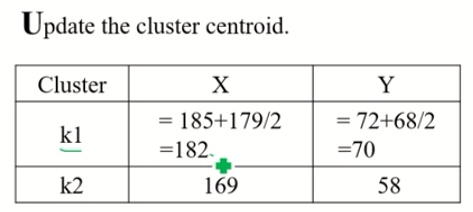
****

****

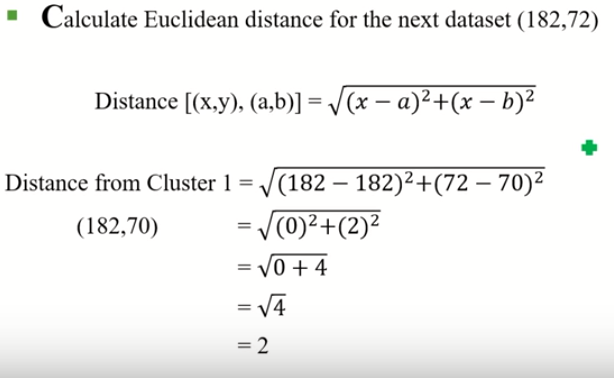
**ამ გამოთვლების შედეგად წერტილი (179, 68) განეკუთვნება Cluster1-ს:**

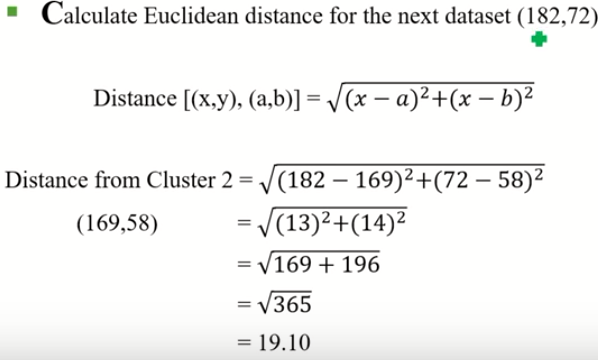
****

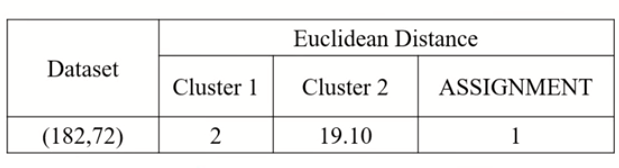
**მოვახდინოთ ცენტროიდების ხელახალი გადათვლა:**

****

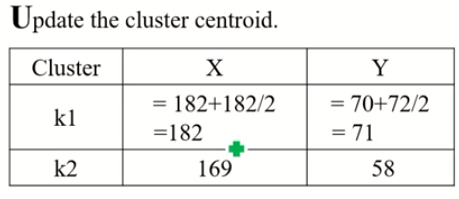
**გადავიდეთ შემდეგ წერტილზე:**

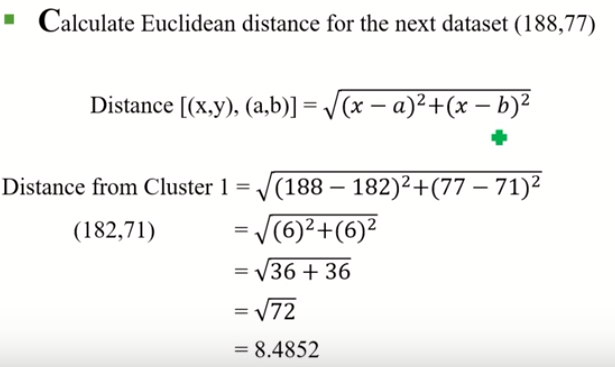
****

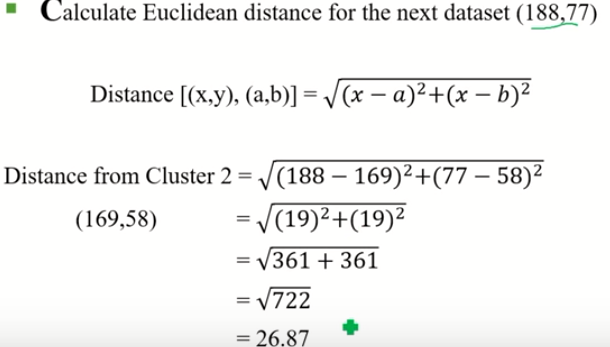
****

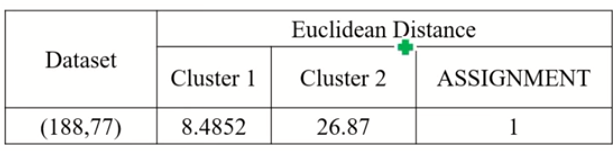
****

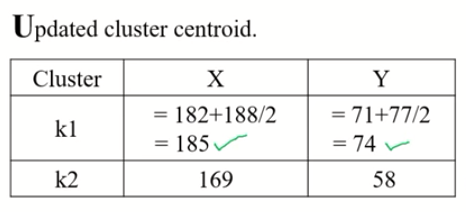
**ცენტროიდებისათვის კი ჩვენ მივიღებთ:**

****

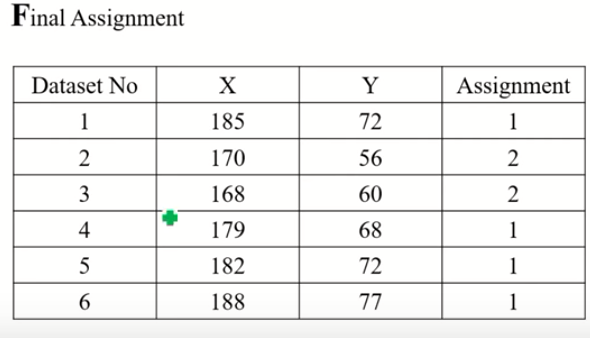
****

****

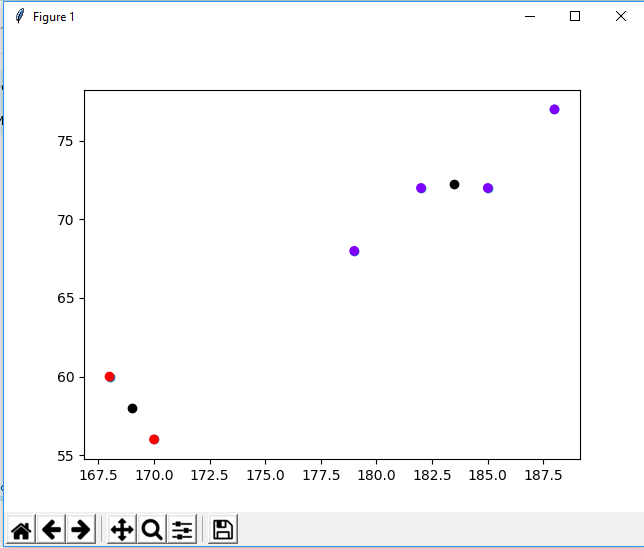
****

****

**საბოლოოდ ჩვენ მივიღებთ შემდეგ სურათს:**

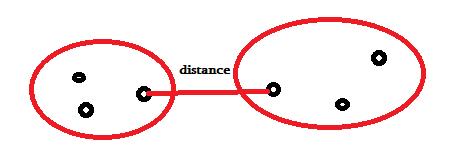
****

**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
**from** sklearn.cluster **import** KMeans  
  
X = np.array([[185,72],  
 [170,56],  
 [168,60],  
 [179,68],  
 [182,72],  
 [188,77]])  
  
plt.scatter(X[:,0],X[:,1], label=**'True Position'**)  
  
  
kmeans = KMeans(n\_clusters=2)  
kmeans.fit(X)  
  
print(kmeans.cluster\_centers\_)  
plt.scatter(X[:,0], X[:,1], c=kmeans.labels\_, cmap=**'rainbow'**)  
plt.scatter(kmeans.cluster\_centers\_[:,0] ,kmeans.cluster\_centers\_[:,1], color=**'black'**)  
plt.show()

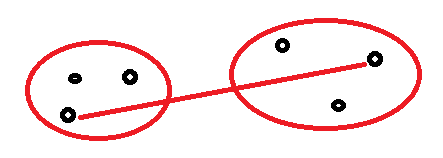
****

**იერარქიული კლასტერის აგება**

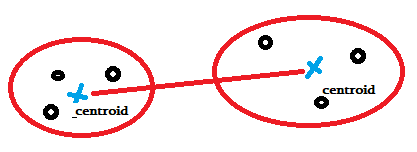
**Single Linkage** - This is the distance between the closest members of the two clusters



**Complete Linkage** - This is the distance between the members that are farthest apart

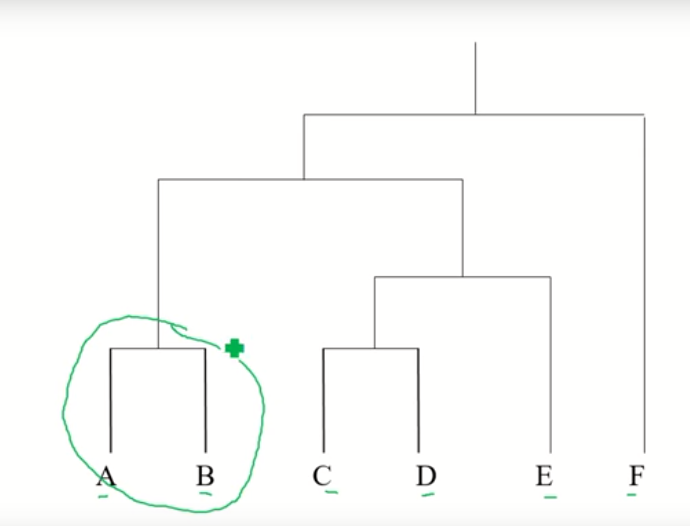


**Average Linkage** – This is the distance between centroids

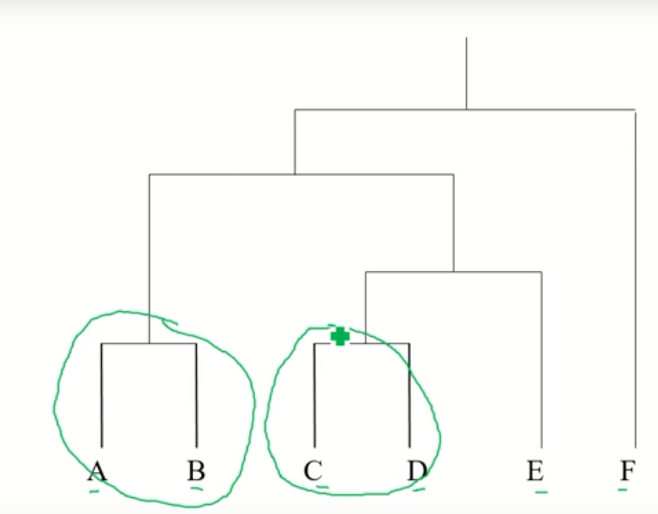




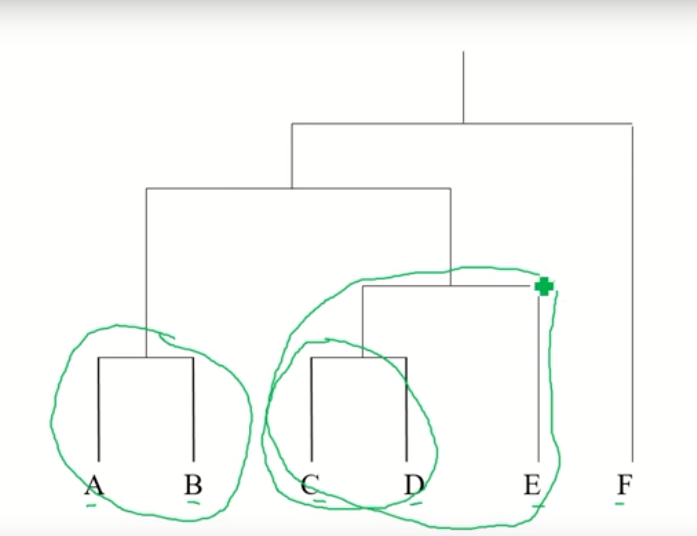
ყველაზე უახლოესი წევრები ქმნიან ერთ კლასტერს:



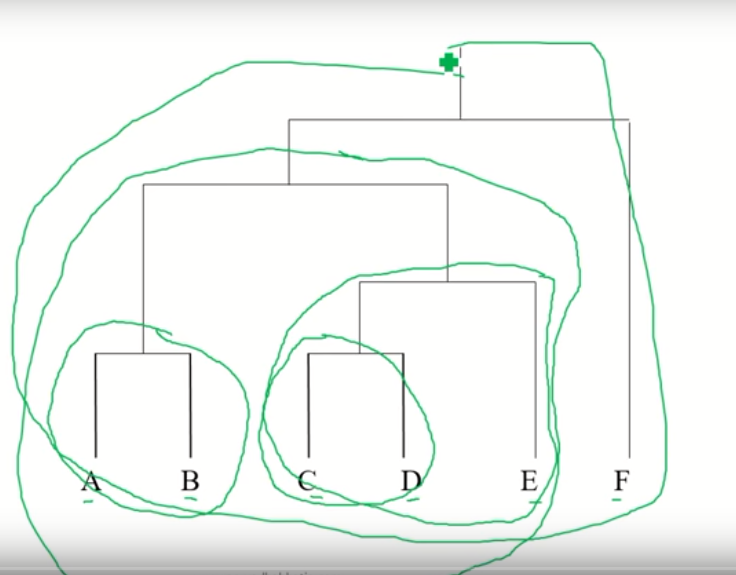
ჩვენ ვხედავთ შემდეგ კლასტერს:



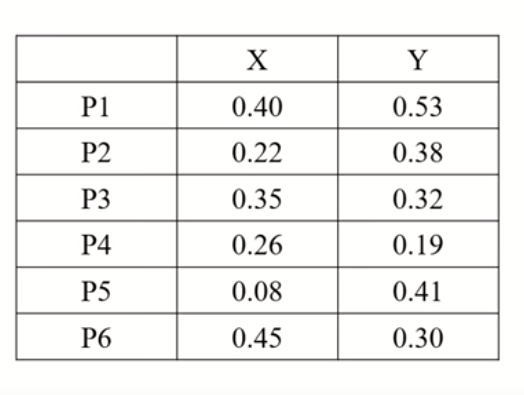
შემდეგ ჩვენ ერთმანეთთან ვაკავშირებთ C, D კლასტერს და E ელემენტს



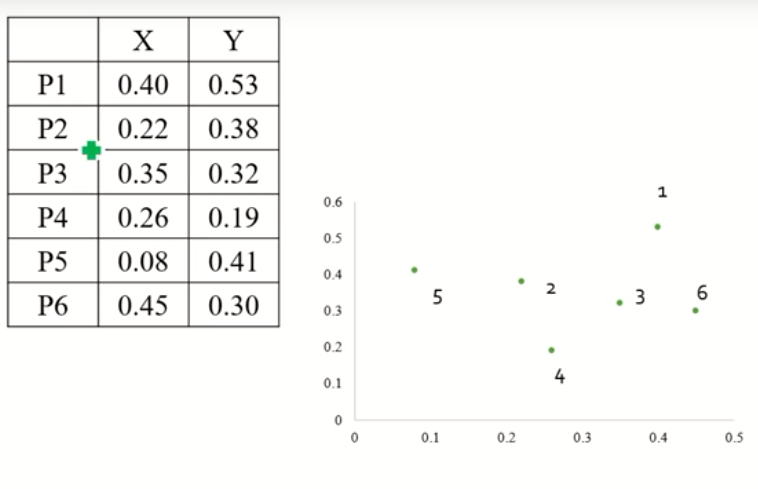
თუ ამ პროცესს უსასრულოდ გავაგრძელებთ მივიღებთ ერთ დიდ კლასტერს

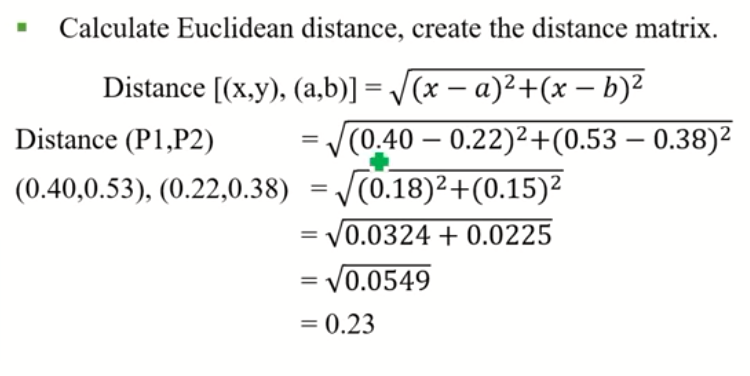


განვიხილოთ იერარქიული კლასტერის აგების ალგორითმი Single-Link ტექნოლოგიის გამოყენებით. ვთქვათ მოცემული გვაქვს შემდეგი წერტილები:

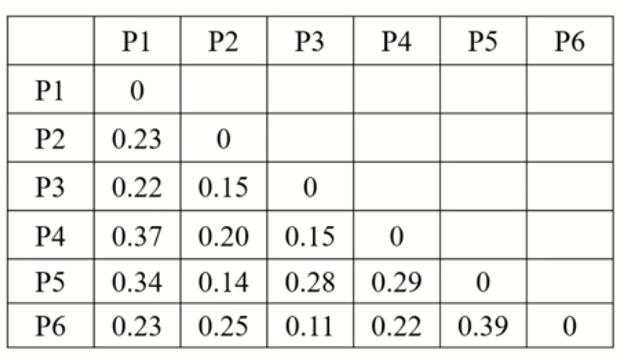


ავაგოთ შესაბამისი გრაფიკი:

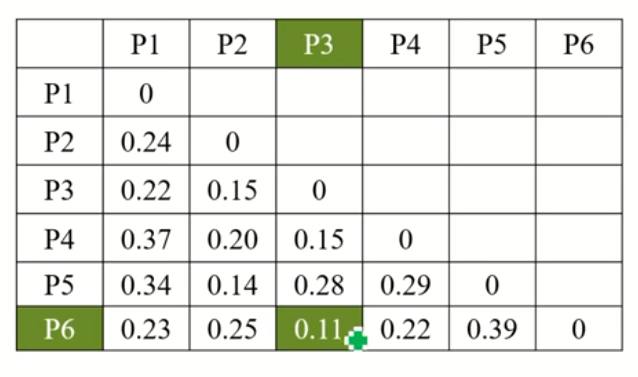




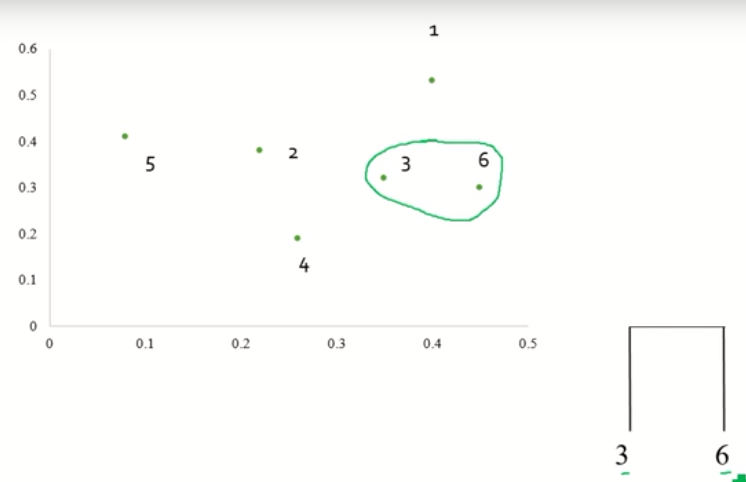
ავაგოთ ე.წ. Distance Matrix



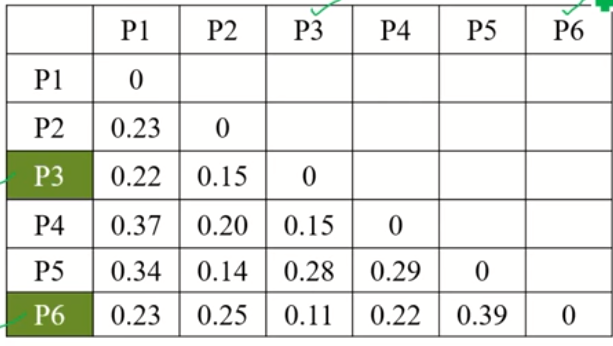
ვეძებთ პირველ კლასტერს იერარქიაში. ამისათვის მატრიცაში ვეძებთ მინიმალურ არა ნულის ტოლ ელემენტს



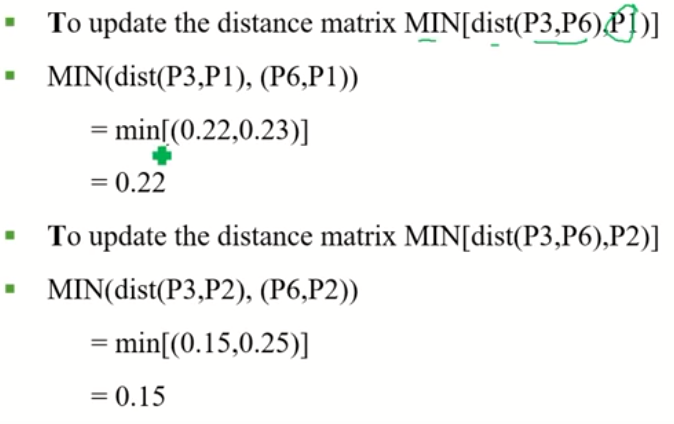
ამრიგად 0,11 არის მანძილი P6 და P3 წერტილებს შორის ამრიგად ჩვენ მივიღეთ P6 და P3 წერტილებისაგან შემდგარი კლასტერი. ნახაზზე მოცემულია კლასტერი და საწყისი დენდოგრამა:



შევცვალოთ ჩვენი საწყისი მატრიცა. ჩვენ ეხლა უნდა განვსაზღვროთ დისტანცია (P3 P6) კლასტერსა და P1 წერტილს შორის:



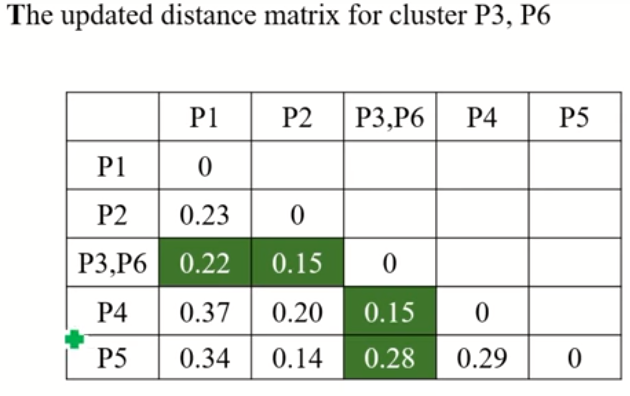
ვიპოვოთ



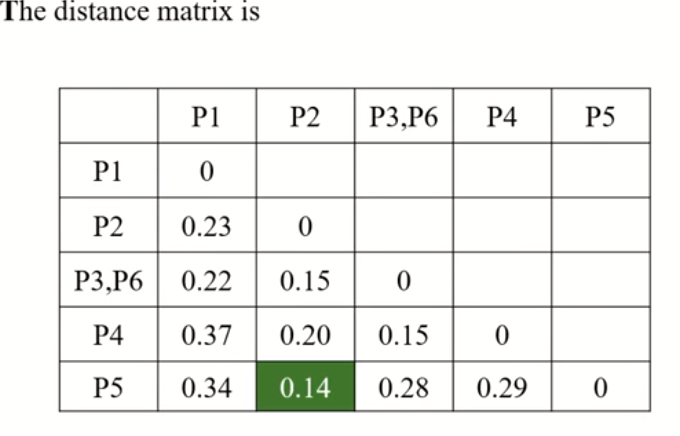
ანალოგიურად მივიღებთ



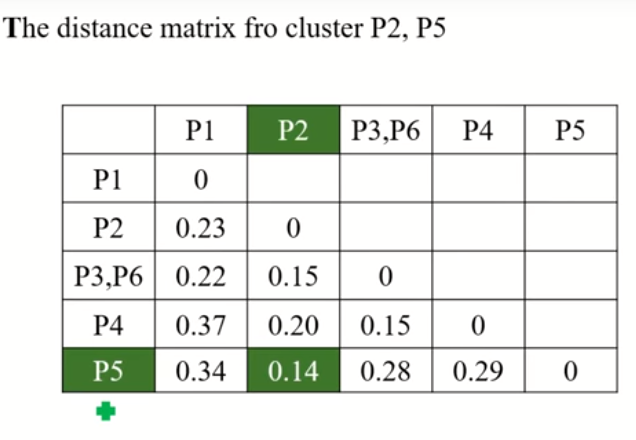
მივიღებთ ახამ მატრიცას (P3 P6) კლასტერისათვის



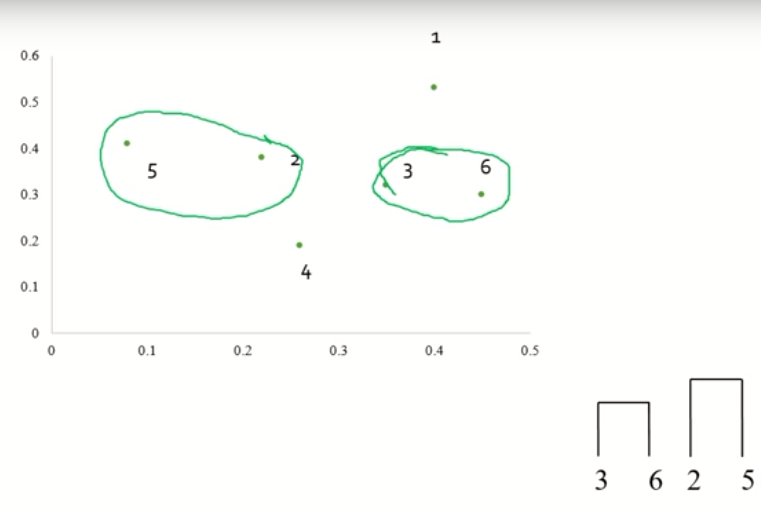
ჩვენ ისევ ვეძებთ მინიმალურ არანულოვან წევრს მატრიცაში

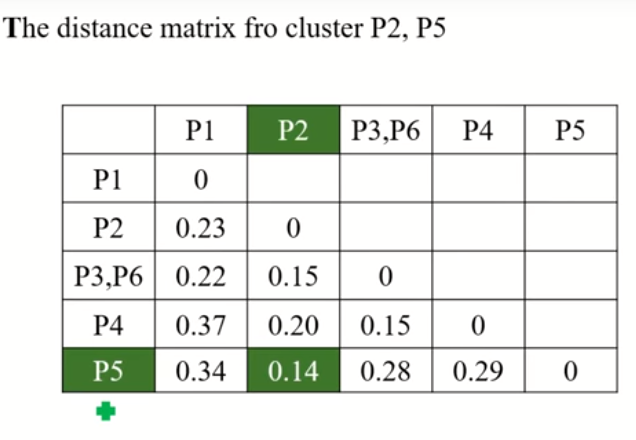
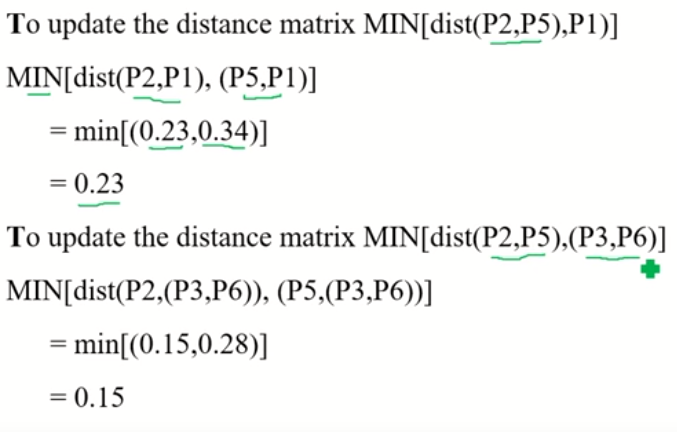


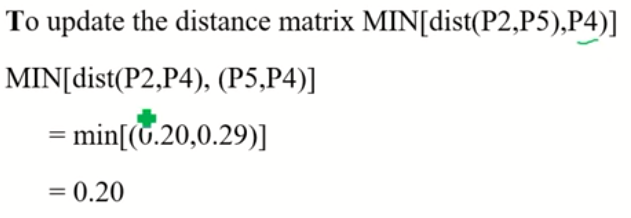
ეს არის მანძილი P2 P6 წერტილებს შორის:



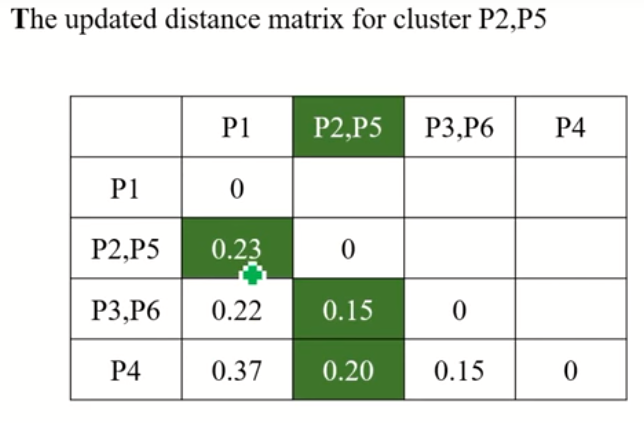
შემდეგ ნახაზზე ჩვენ უკვე მოცემული გვაქვს ორი კლასტერი და ორი დენდოგრამა:



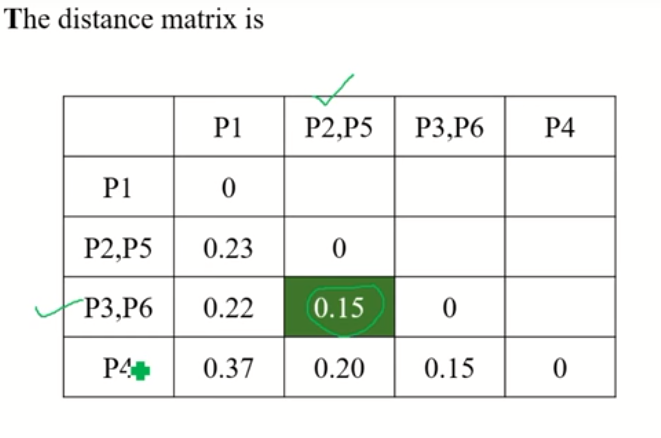




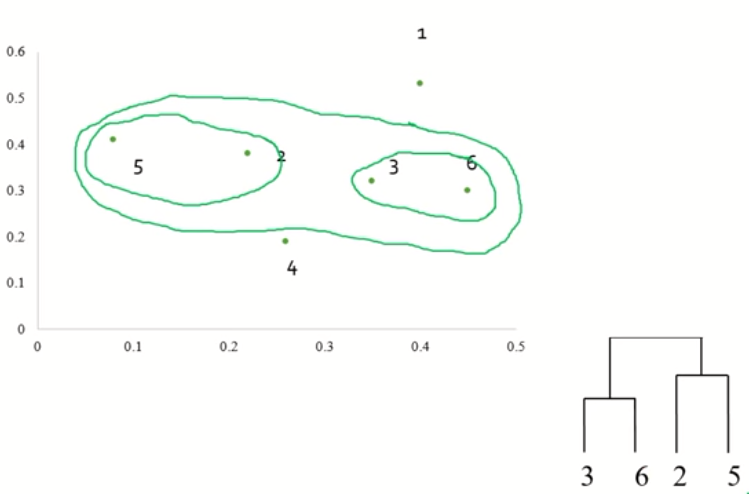
ამ გამოთვლების შედეგად ვიღებთ ახალ მატრიცას:



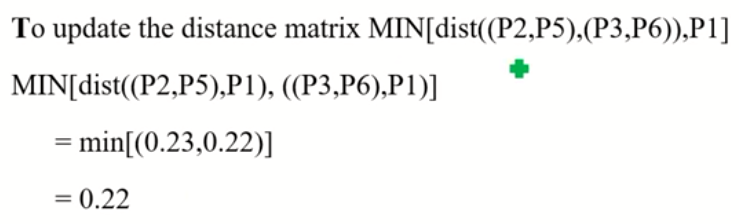
აქ ჩვენ ვხედავთ ორ ერთიდაიგივე მინიმალურ ელემენტს 0,15 და ავირჩიოთ ის რომელიც დგას ორი კლასტერის კვეთაში:



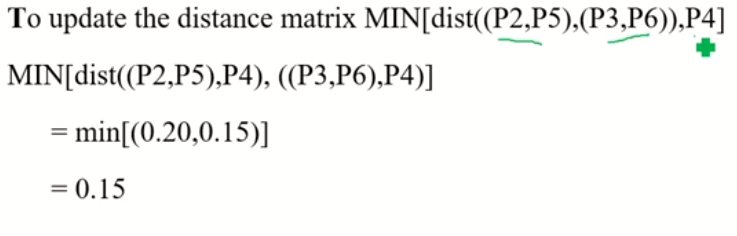
გრაფიკულად გვექნება შემდეგი:



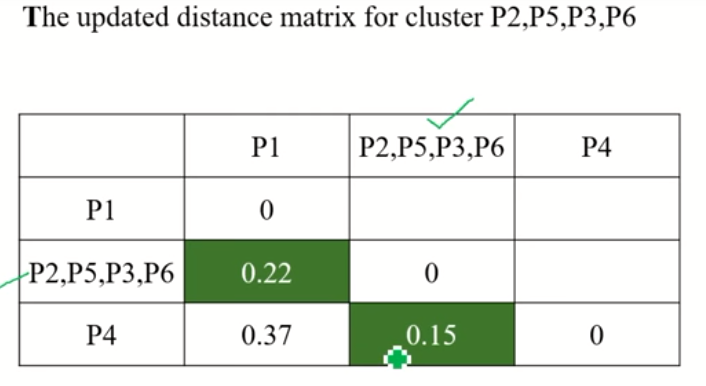
გვაქვს ახალი კლასტერი **((P2, P5)(P3, P6))**



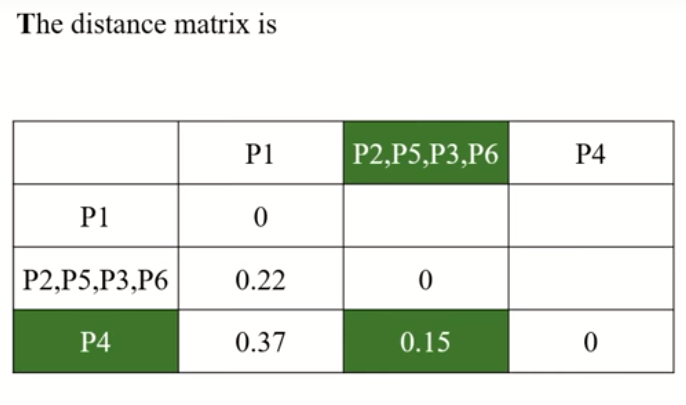
შემდეგი განახლებული მნიშვნელობა:

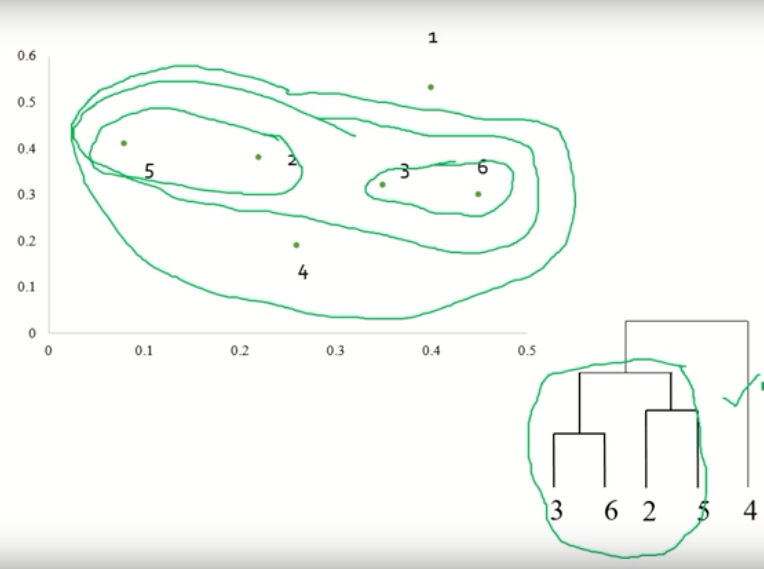


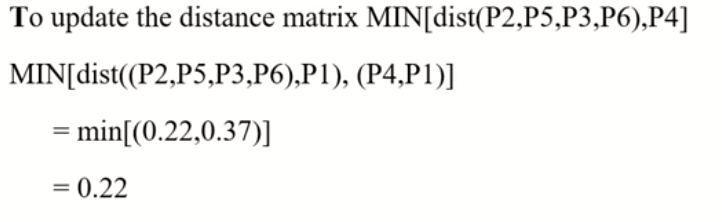
P2,P5,P3,P6 კლასტერის განახლებული მატრიცა



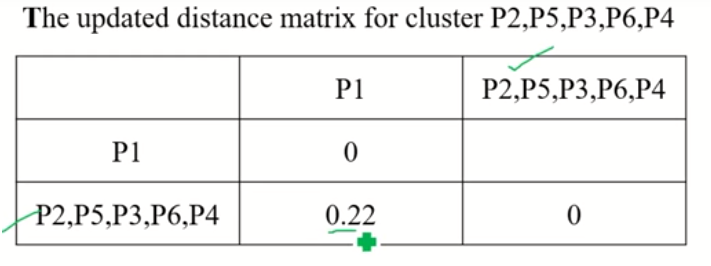
**აქ მინიმალური მნიშვნელობა არის 0,15 და ახალი კლასტერი ((P2,P5,P3,P6),P4)**



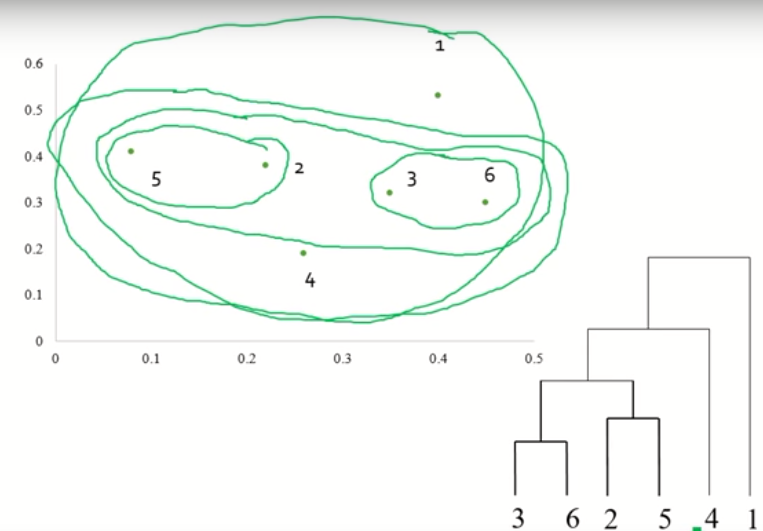




ამის შემდგომ მვიღებთ

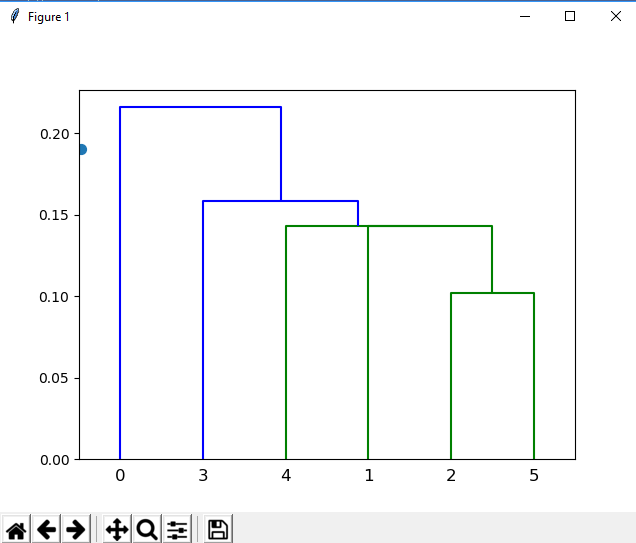


საბოლოოდ კი მივიღებთ შემდეგ დენდოგრამას:

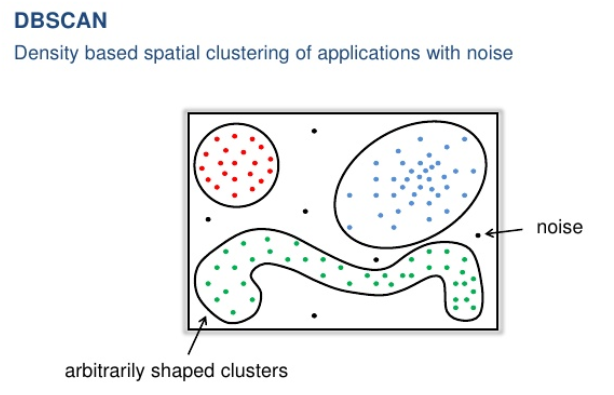


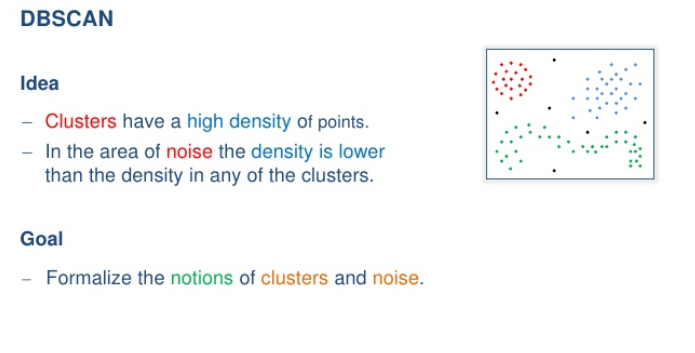
ამით ჩვენ ავაგეთ იერარქიული კლასტერი.

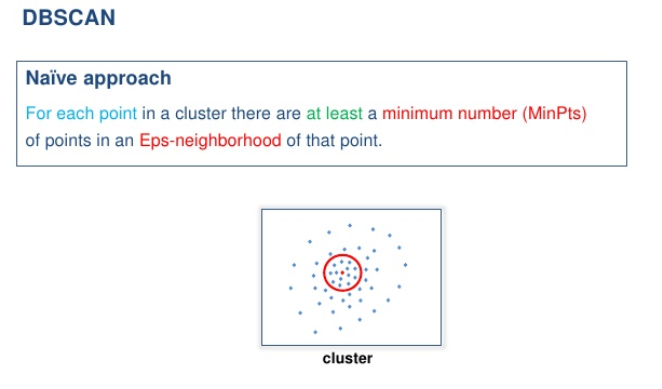
**import** numpy **as** np  
**from** scipy.cluster.hierarchy **import** linkage, dendrogram  
**import** matplotlib.pyplot **as** plt  
x=np.array([[0.40,0.53],[0.22,0.38],[0.35,0.32],[0.26,0.19],[0.08,0.41],[0.45,0.30]])  
  
plt.scatter(x[:,0],x[:,1], s=50)  
  
linkage\_matrix=linkage(x,**"single"**)  
  
dendogram = dendrogram(linkage\_matrix)  
  
print(linkage\_matrix)  
plt.show()

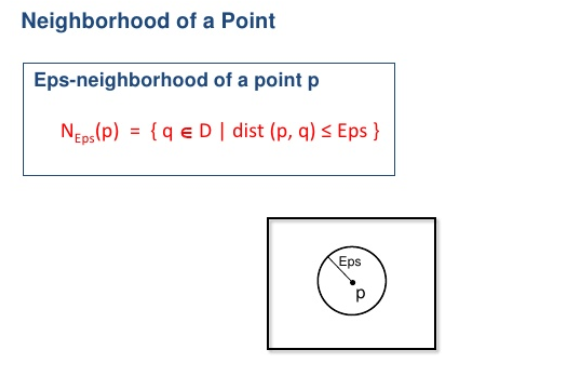


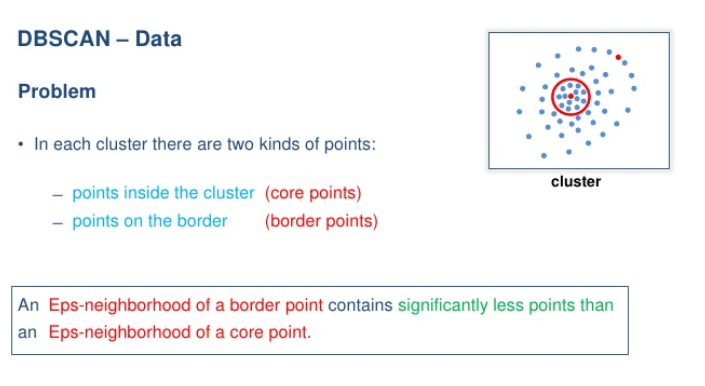
**Density-based clustering**

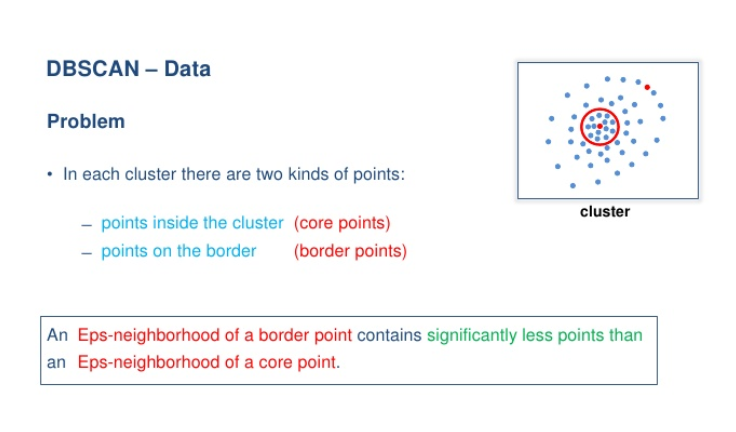


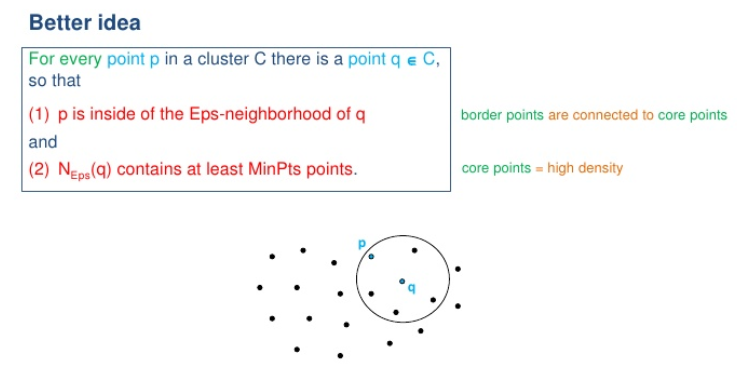


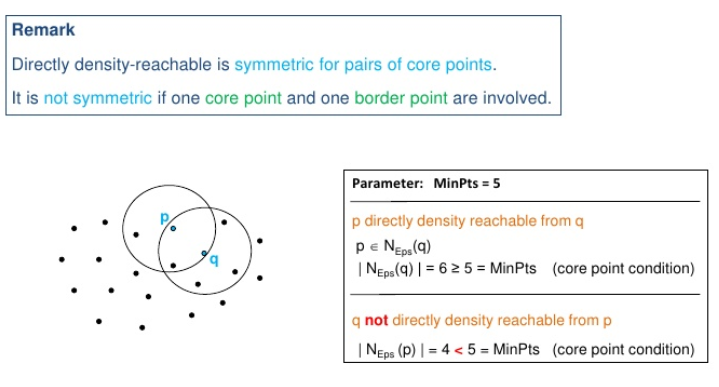


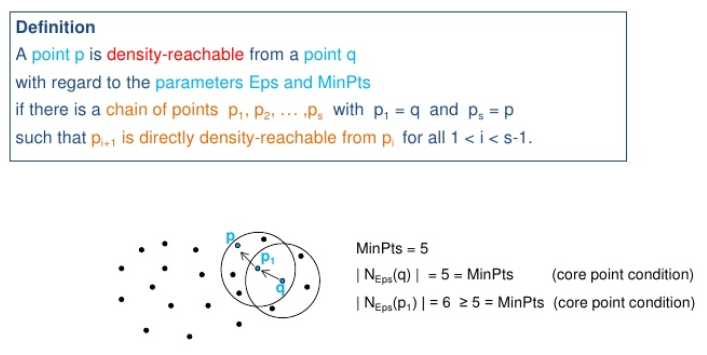


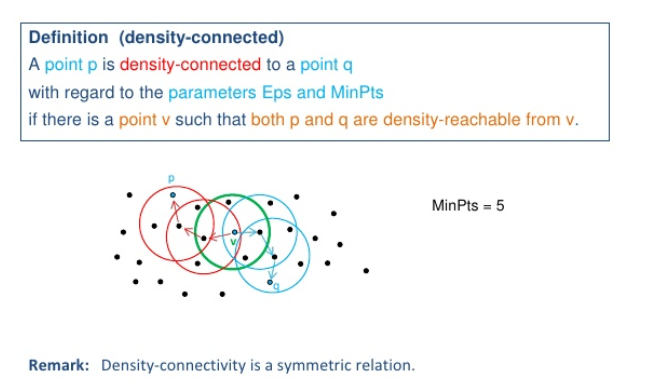


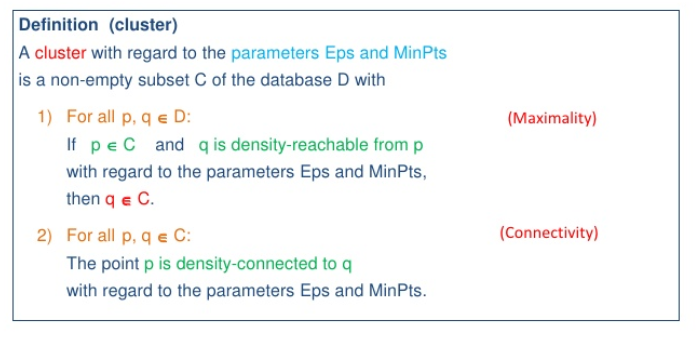


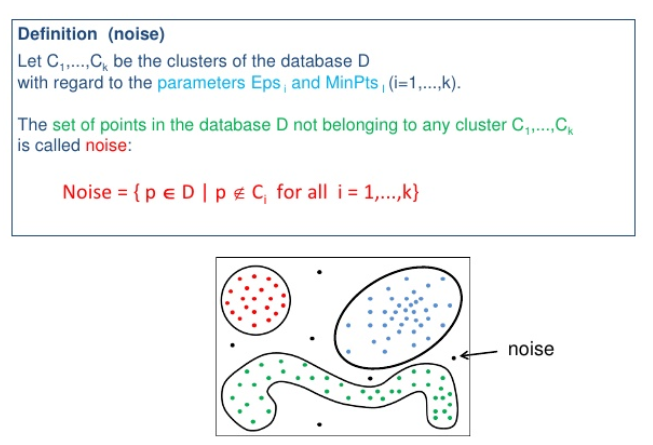


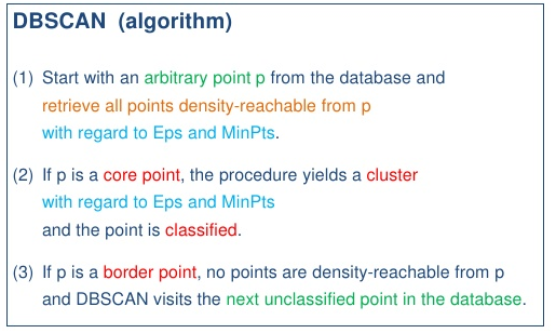












*###DBSCAN is somewhat slower than agglomerative clustering and k-Means, working by picking  
###a point to start with, but still scales to relatively large datasets.  
  
###There are two parameters in DBSCAN, min\_samples and eps. If there are at least min\_samples  
###many data points within a distance of eps to a given data point, it’s called a core sample.  
###Core samples that are closer than the distance eps are put into the same cluster by DBSCAN.  
###Points more than min-sample are within eps are labeled core, otherwise noise***from** sklearn.datasets **import** make\_moons  
**import** numpy **as** np  
**from** sklearn.datasets **import** make\_blobs  
**from** sklearn.cluster **import** DBSCAN  
**import** matplotlib.pyplot **as** plt  
**from** sklearn.model\_selection **import** train\_test\_split  
  
X, y = make\_blobs(random\_state=0, n\_samples=12)  
dbscan = DBSCAN()  
clusters = dbscan.fit\_predict(X)  
*##print(clusters)  
  
###All data points were assigned the label -1, which stands for noise.  
  
###Try changing eps and min\_samples to see the effect*fig, axes = plt.subplots(3, 4, figsize=(12, 6), subplot\_kw={**'xticks'**: (), **'yticks'**: ()})  
*###delete the axises value with subplot\_kw*colors = np.array([**'r'**, **'b'**, **'g'**, **'y'**])  
**for** i, min\_samples **in** enumerate([2, 3, 5]):  
 **for** j, eps **in** enumerate([1, 1.5, 2, 3]):  
 dbscan = DBSCAN(min\_samples=min\_samples, eps=eps)  
 clusters = dbscan.fit\_predict(X)  
 print(**'min\_samples: %d eps: %.1f clusters: %s'** % (min\_samples, eps, clusters))  
 sizes = 30 \* np.ones(X.shape[0])  
 sizes[dbscan.core\_sample\_indices\_] \*= 8 *###enlarge the core points by 8 times* axes[i, j].scatter(X[:, 0], X[:, 1], c=colors[clusters], s=sizes)  
 axes[i, j].set\_title(**'min\_samples: %d eps: %0.1f'** % (min\_samples, eps))  
fig.tight\_layout()  
plt.show()