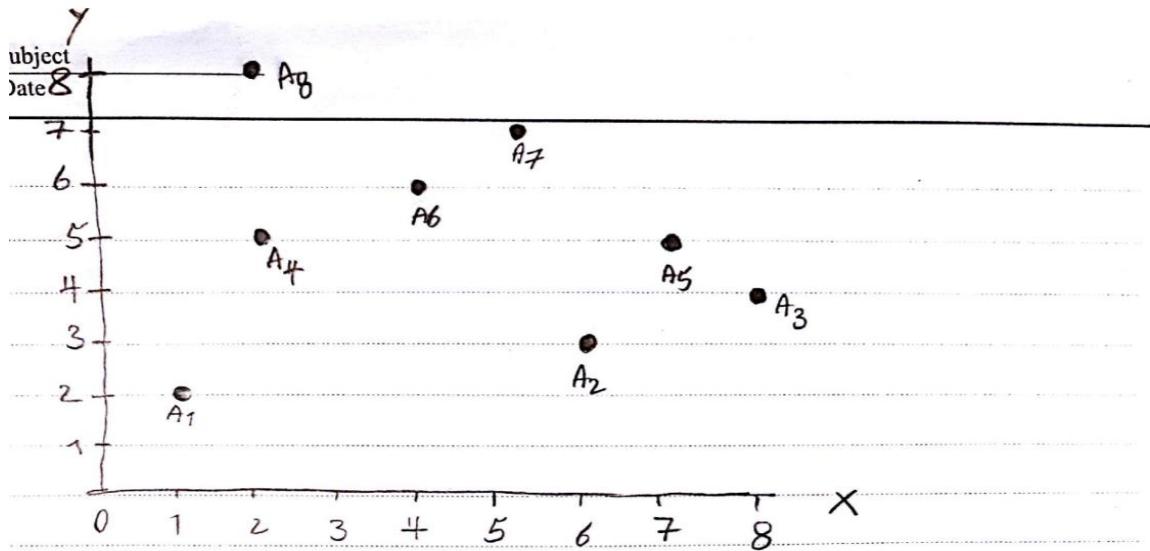


(1)

(الف)



$$C_1 = \left( \frac{8+2+2}{3}, \frac{4+5+8}{3} \right) = \left( 4, \frac{17}{3} \right) \quad \text{(الخط 1)}$$

$$C_2 = \left( \frac{5+6+7}{3}, \frac{3+5+7}{3} \right) = (6, 5)$$

$$C_3 = \left( \frac{1+4}{2}, \frac{2+6}{2} \right) = \left( \frac{5}{2}, 4 \right)$$

8 خطوط

|           |       |       |       |       |       |       |       |      |
|-----------|-------|-------|-------|-------|-------|-------|-------|------|
| $D^0 = 1$ | 4.74  | 3.33  | 4.33  | 2.1   | 3.07  | 0.33  | 1.66  | 3.07 |
| 2         | 5.8   | 2     | 2.2   | 4     | 1     | 2.23  | 2.23  | 5    |
| 3         | 2.5   | 3.64  | 5.5   | 7.11  | 4.6   | 2.5   | 3.9   | 4.03 |
| $A_1$     | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ | $A_7$ | $A_8$ |      |

$$G^0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$A_1 \ A_2 \ A_3 \ A_4 \ A_5 \ A_6 \ A_7 \ A_8$

g1:  $A_6, A_7, A_8$ g2:  $A_2, A_3, A_5$ g3:  $A_1, A_4$

$$C_1 = \left( \frac{4+5+2}{3}, \frac{6+7+8}{3} \right) = \left( \frac{11}{3}, 7 \right)$$

$$C_2 = \left( \frac{6+8+7}{3}, \frac{3+4+5}{3} \right) = (7, 4)$$

$$C_3 = \left( \frac{1+2}{2}, \frac{2+5}{2} \right) = \left( \frac{3}{2}, \frac{7}{2} \right)$$

$$D^1 = \begin{array}{c|ccccccccc} 1 & 5.66 & 4.62 & 5.2 & 2.6 & 3.88 & 1.05 & 1.33 & 1.94 \\ 2 & 6.32 & 1.41 & 1 & 5.09 & 1 & 3.6 & 3.6 & 6.4 \\ 3 & 7.5 & 4.5 & 6.51 & 1.5 & 5.7 & 3.5 & 4.9 & 4.5 \end{array}$$

$$G^1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{array}{l} g_1 : A_6, A_7, A_8 \\ g_2 : A_2, A_3, A_5 \\ g_3 : A_1, A_4 \end{array}$$

جذب هم و هم تبع ایس

(٤)

$$C1=A1, C2=A2, C3=A3$$

|    | X | Y | C1 | C2 | C3 |
|----|---|---|----|----|----|
| A1 | 1 | 2 | -  | -  | -  |
| A2 | 6 | 3 | -  | -  | -  |
| A3 | 8 | 4 | -  | -  | -  |
| A4 | 2 | 5 | 4  | 6  | 7  |

|    |   |   |   |   |    |
|----|---|---|---|---|----|
| A5 | 7 | 5 | 9 | 3 | 2  |
| A6 | 4 | 6 | 7 | 5 | 6  |
| A7 | 5 | 7 | 9 | 5 | 6  |
| A8 | 2 | 8 | 7 | 9 | 10 |

.C1 میره به A4 ,A8

.C2 میره به A6,A7

.C3 میره به A5

$$\text{COST} = (4+7) + (5+5) + 2 = 23$$

A4 رو به عنوان نقطه تصافی در نظر میگیریم برا خوشه ی سوم.

|    | X | Y | C1 | C2 | C3 |
|----|---|---|----|----|----|
| A1 | 1 | 2 | -  | -  | -  |
| A2 | 6 | 3 | -  | -  | -  |
| A3 | 8 | 4 | 9  | 3  | 7  |
| A4 | 2 | 5 | -  | -  | -  |
| A5 | 7 | 5 | 9  | 3  | 5  |
| A6 | 4 | 6 | 7  | 5  | 3  |
| A7 | 5 | 7 | 9  | 5  | 5  |
| A8 | 2 | 8 | 7  | 9  | 3  |

.C2 میره A3,A5,A7

.C3 میره A6,A8

$$\text{NEW COST} = (3+3+5) + (3+3) = 17$$

17<23

.C3 میشه A4 پس.

: مرحله 2

A5 به عنوان خوشه دوم قرار میگیرد.

|    | X | Y | C1 | C2 | C3 |
|----|---|---|----|----|----|
| A1 | 1 | 2 | -  | -  | -  |
| A2 | 6 | 3 | 6  | 3  | 5  |
| A3 | 8 | 4 | 9  | 2  | 7  |
| A4 | 2 | 5 | -  | -  | -  |
| A5 | 7 | 5 | -  | -  | -  |
| A6 | 4 | 6 | 7  | 4  | 3  |
| A7 | 5 | 7 | 10 | 4  | 5  |
| A8 | 2 | 8 | 7  | 8  | 3  |

خوشه ی دوم A2,A3,A7

خوشه ی سوم A6,A8

$$\text{COST} = 3+2+4+3+3 = 15$$

پس A5 به عنوان خوشه ی دوم قرار میگیرد.

: مرحله 3

A6 را به عنوان مرکز خوشه ی اول قرار میدهیم.

|    | X | Y | C1 | C2 | C3 |
|----|---|---|----|----|----|
| A1 | 1 | 2 | 6  | 9  | 4  |
| A2 | 6 | 3 | 5  | 3  | 5  |
| A3 | 8 | 4 | 6  | 2  | 7  |
| A4 | 2 | 5 | -  | -  | -  |
| A5 | 7 | 5 | -  | -  | -  |
| A6 | 4 | 6 | -  | -  | -  |
| A7 | 5 | 7 | 2  | 4  | 5  |
| A8 | 2 | 8 | 4  | 8  | 3  |

خوشة اول. A7,A6

خوشه دوم. A2,A3,A5

خوشه سوم. A1,A8,A4

$$\text{COST} = 4+3+2+2+3=14$$

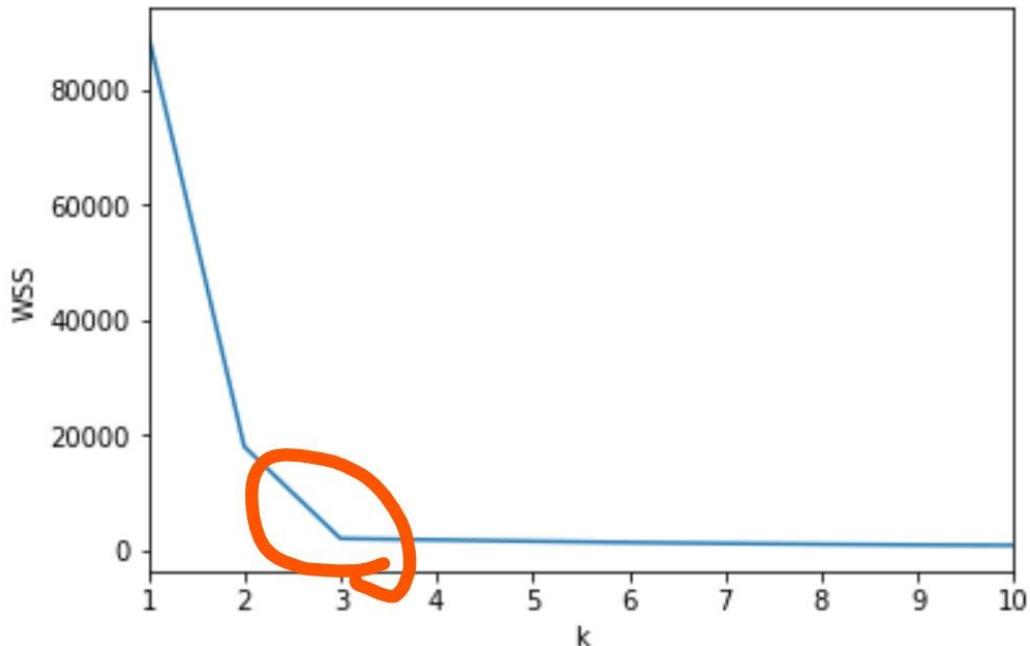
$$15 > 14$$

## The Elbow Method (2

برای مقادیر مختلف  $k$  ، مجموع خطاهای مجاز (WSS) را محاسبه کنید.  $k$  ای انتخاب میشود که WSS برای اولین بار شروع به کاهش می کند. در نمودار  $k-WSS$  ، این به صورت آرنج قابل مشاهده است.

مربع خطای هر نقطه، مربع فاصله نقطه از مرکز خوشه پیش بینی شده آن است.

WSS مجموع این خطاهای مربع برای همه نقاط است.  
از هر معیار فاصله ای مانند فاصله اقلیدسی یا فاصله منهتن می توان استفاده کرد.



(الف) می توان یک medoid را به عنوان نقطه ای در خوش تعریف کرد که عدم شباهت آن با سایر نقاط خوش حداقل است.

فاصله ای medoid متعلق به خوش بی  $C_i$  با نقاط  $P_i$  حساب میشود:

$$c = \sum_{Ci} \sum_{Pi \in Ci} |Pi - Ci|$$

مزایا: پیاده سازی راحت، سریع است چون مراحل ثابتی دارد، به outlier کمتر توجه نمیشود نسبت بقیه ای الگو ها.

عیب اصلی الگوریتم های K-Medoid این است که برای خوش بندی گروه های غیر کروی (شکل دلخواه) از اشیا مناسب نیست. دلیل این امر این است که به حداقل رساندن فاصله بین اجسام غیر medoid و medoid (مرکز خوش) متکی است - به طور خلاصه ، از فشردگی به عنوان معیار های خوش بندی به جای اتصال استفاده می کند.

ممکن است نتایج مختلفی را برای اجرایهای مختلف در یک مجموعه داده به دست آورد زیرا اولین k تا medoid به طور تصادفی انتخاب می شوند.

(ب)

CLARA به جای یافتن medoid برای کل مجموعه داده ها ، نمونه کوچکی از داده ها را با اندازه ثابت در نظر می گیرد و الگوریتم PAM را برای تولید یک مجموعه بهینه از Medoid های medoid برای نمونه اعمال می کند. دقت medoid حاصل با میانگین عدم تشابه بین هر شی در کل مجموعه داده

و Medoid خوشه آن اندازه گیری می شود که به عنوان تابع هزینه تعریف میشود.

CLARA به منظور به حداقل رساندن bias نمونه برداری ، چندین بار فرایندهای نمونه گیری و خوشه بندی را تکرار می کند. نتایج نهایی خوشه بندی با مجموعه ای از Medoid با حداقل هزینه مطابقت دارد.

مزایا: به طور کلی پیچیدگی نسبتاً کم و کارایی بالا محاسباتی معایب: برای داده های غیر محدب مناسب نیست ، نسبتاً حساس به outliers local optimal به راحتی به کشیده می شود.

ج) داده های گروهی متراکم را در یک خوشه واحد قرار می دهد. ویژگی خوشه بندی DBSCAN قوی بودن نسبت به نقاط خوشه ها را مشخص کنیم ، اینجا نیازی به تعداد خوشه ها نیست که باید از قبل گفته شود.

DBSCAN فقط به دو پارامتر نیاز دارد: minPoints و Epsilon. اپسیلون شعاع دایره ای است که در اطراف هر نقطه داده ایجاد می شود تا چگالی را بررسی کند و

`minPoints` حداقل تعداد نقاط داده مورد نیاز درون آن دایره است تا به عنوان نقطه هسته طبقه بندی شود.

مزایا:

مشخص کردن تعداد خوشه های داده نیاز نیست.

می تواند هر شکلی را پیدا کند حتی اگر خوشه توسط خوشه دیگری احاطه شده باشد.

به راحتی می تواند نقاط `outliers` افتاده را در مجموعه داده ها پیدا کند.

حساسیت زیادی به `noise` ندارد ، به این معنی است که `noise` را تحمل می کند.

معیاب:

کیفیت نتیجه به اندازه گیری فاصله مورد استفاده به عملکرد `regionQuery` بستگی دارد.

نقاط مرزی بسته به ترتیب پردازش ممکن است در هر خوشه ای قرار داشته باشند بنابراین کاملاً قطعی نیست.

وقتی هزینه محاسبه نزدیکترین همسایه زیاد باشد ممکن است `computation time` بالا باشد.

برای ابعاد بالاتر می تواند در اجرا کند باشد.

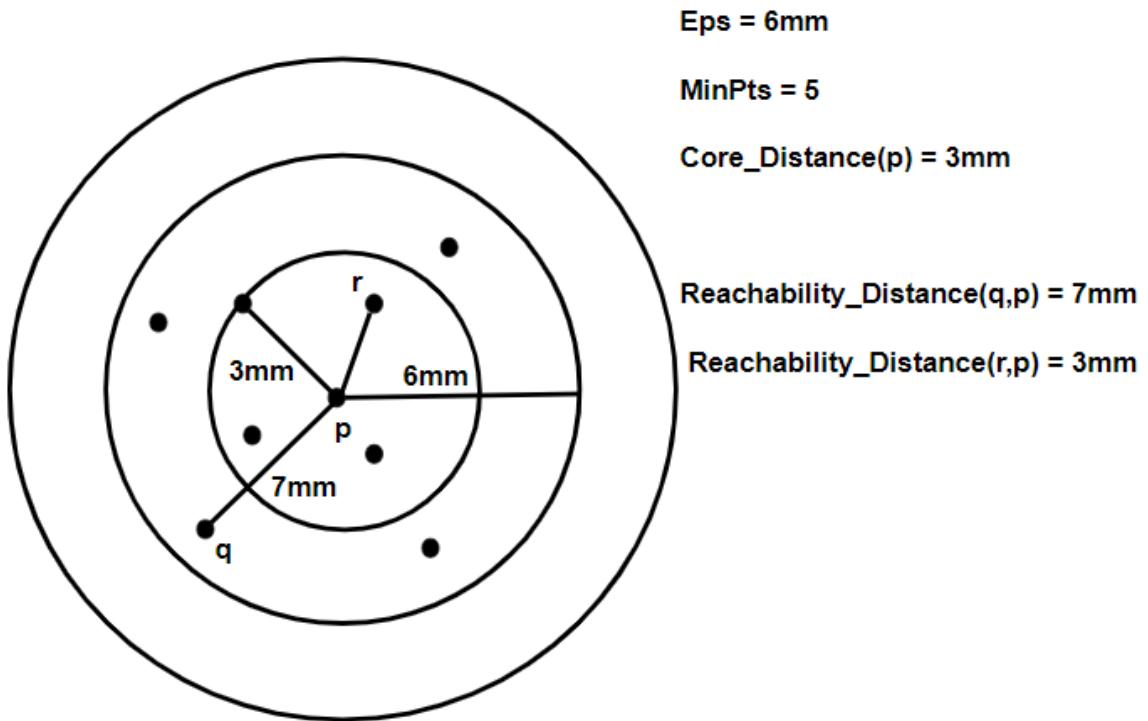
سازگاری تنوع در تراکم محلی کمتر است.

د) OPTICS همون DBSCAN است با دو پارامتر اضافه:

فاصله هسته: حداقل مقدار شعاع مورد نیاز برای طبقه بندی یک نقطه داده شده به عنوان نقطه اصلی است. اگر نقطه داده شده یک هسته اصلی نباشد ، فاصله هسته تعریف نشده است.

فاصله دسترسی: با توجه به داده دیگر تعریف می شود.

فاصله قابل دسترسی بین یک نقطه  $p$  و  $q$  حداقل فاصله اصلی  $p$  و فاصله اقلیدسی (یا برخی از معیارهای فاصله دیگر) بین  $p$  و  $q$  است. اگر  $q$  یک نقطه اصلی نباشد ، فاصله دسترسی قابل تعریف نیست.



معایب: OPTICS حافظه‌ی بیشتری نیاز دارد به خاطر  $\min$  heap برای تشخیص نقطه‌ی بعدی برای فاصله‌ی دسترسی. به قدرت محاسباتی بیشتری هم نیاز دارد چون نسبت به DBSCAN از پارامترهای دیگه‌ای هم استفاده میکنه برای query هاش.

اگر بین خوشه‌ها افت تراکم وجود نداشته باشد، کار نمیکند.  
مزایا: این روش داده‌های داده شده را به خوشه‌ها تفکیک نمی‌کند. این فقط یک نمودار از بر اساس فاصله‌ی دسترسی تولید می‌کند و تفسیر ما است که نقاط را بر این اساس خوشه‌بندی کنیم. پس یک عدد از قبل مشخص شده برای تعداد خوشه‌ها نمیخواهد.

خوشه ها به هر شکلی میتوانند باشند.  
Outlier ها را به خوبی تشخیص میدهد.

Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) یک الگوریتم خوشه بندی است که می تواند با تولید یک خلاصه کوچک و جمع و جور از مجموعه داده بزرگ که اطلاعات را تا حد ممکن حفظ می کند (Clustering Feature Tree)، مجموعه داده های بزرگ را خوشه بندی کند. این خلاصه کوچک به جای خوشه بندی مجموعه داده بزرگ، خوشه بندی می شود.

معایب: فقط می تواند صفات متریک (قابل اندازه گیری) را پردازش کند. صفت متریک هر صفتی است که مقادیر آن را می توان در فضای اقلیدسی نشان داد، به عنوان مثال، هیچ ویژگی طبقه ای قابل نمایش نیست.

با افزایش اندازه مجموعه داده ها مقیاس خوبی ندارند و اغلب به پارامتر سازی مناسب نیاز دارند که تهیه آنها معمولاً دشوار است

مزایا: اول، نیازی نیست که از قبل تعداد خوشه های مورد انتظار را بدانیم. دوم، بدون محاسبات خوشه بندی نهایی، الگوریتم سریع BIRCH حتی سریعتر خواهد شد

و) Chameleon یک الگوریتم خوش بندی دو فازی است. در ابتدا یک نمودار نزدیکترین همسایه k-mean (با استفاده از الگوریتم تقسیم نمودار) ایجاد می کند که فقط شامل پیوندهایی بین یک نقطه و نزدیکترین همسایگان آن است. فاز دوم از الگوریتم خوش بندی سلسله مراتبی تجمعی استفاده میکند برای پیدا کردن خوشه های واقعی که معمولاً با هم ترکیب می شوند تا خوشه های فرعی را تشکیل دهند.

مزایا: الگوریتم افزایشی  
معایب: پیچیدگی زمانی الگوریتم CHAMELEON در حد بالا ابعاد  $O(n^2)$  است.

2) الف) نادرست - تمام نقاطی که در فاصله اپسیلون از هر نقطه (نه هسته) خوش قرار دارند را پیدا کرده و آنها را به خوش اضافه میکند. تمام نقاطی را که از فاصله اپسیلون از تمام نقاط تازه اضافه شده پیدا کند آنها را به خوش اضافه میکند.

ب) درست - چون بر اساس فاصله  $\epsilon$  کار میکند، اگر این فاصله درست انتخاب شده باشد، outliers در خوش ها قرار نمیگیرند.

ج) نادرست -  $O(n^2)$  است چون فاصله‌ی بین هر دو object موجود را حساب می‌کند.

د) درست - نیازی به این کار ندارد و تعداد خوش‌ها را خودش حدس می‌زند.

(5) الف) DBSCAN - چون فاصله‌های گروه‌ها از هم نزدیک است، kmean قاطی می‌کند گروه‌ها را و یک نقطه میانگین قرار میدهد و چند نقطه از چند گروه در یک خوش‌قرار می‌گیرند که اشتباه است.

ب) dbscan - اینجا چون density ضعیف است با kmean به مشکل میخورد مثلا برای خوش سبز که داده‌هایش پراکنده‌تر است، در یک خوش‌قرار نمی‌گیرند بر اساس density.

ج) DBSCAN و kmean - با kmean مرکز خوش‌ها رو بدست می‌اوریم و با dbscan همزمان نقاط متغق به آن خوش‌ها را. با dbscan خالی گروه‌های متصل به هم یکی می‌شوند و با kmean خالی نصف گروه قرمز در گروه سبز قرار می‌گرفت.

د) DBSCAN- واضحا فقط بر اساس تراکم است، پس فقط با DBSCAN انجام می‌پذیرد.

(6)

|   | A    | B    | C    | D    | E    | (5) |
|---|------|------|------|------|------|-----|
| A | 0    | 1.23 | 2.44 | 0.85 | 2.04 |     |
| B | 1.23 | 0    | 0.74 | 1.2  | 0.98 |     |
| C | 2.44 | 0.74 | 0    | 1.34 | 1.4  |     |
| D | 0.85 | 1.2  | 1.34 | 0    | 0.87 |     |
| E | 2.04 | 0.98 | 1.4  | 0.87 | 0    |     |

1)  $D_1(B, C) = 0.74, \delta(B, u) = D_1(B, C)/2 = 0.37$   
 $\delta(A, u)$

matrix update

$$D_2((B, C), A) = \min(\overbrace{D_1(B, A)}^{1.23}, \overbrace{D_1(C, A)}^{2.44}) = 1.23$$

$$D_2((B, C), D) = \min(\overbrace{D_1(B, D)}^{1.2}, \overbrace{D_1(C, D)}^{1.34}) = 1.2$$

$$D_2((B, C), E) = \min(\overbrace{D_1(B, E)}^{0.98}, \overbrace{D_1(C, E)}^{1.4}) = 0.98$$

|       | A    | (B,C) | D    | E    |  |
|-------|------|-------|------|------|--|
| A     | 0    | 1.23  | 0.85 | 2.04 |  |
| (B,C) | 1.23 | 0     | 1.2  | 0.98 |  |
| D     | 0.85 | 1.2   | 0    | 0.87 |  |
| E     | 2.04 | 0.98  | 0.87 | 0    |  |

$$D_2(A, D) = 0.85$$

$$\delta(D, v) = \delta(A, v) = 0.425$$

$$D_3((A, D), (B, C)) = \min_{2.04} (D_2(A, (B, C)), D_2(D, (B, C)) = \delta(u, v) = 0.425 - 0.37 = 0.055$$

$$D_3((A, D), E) = \min(D_2(A, E), D_2(D, E)) = \min(0.87, 1.2) = 0.87$$

$$D_3((A, D), E) = \min(D_2(A, E), D_2(D, E)) = \min(0.87, 1.2) = 0.87$$

DATA

|       | (A,D) | (B,C) | E    |
|-------|-------|-------|------|
| (A,D) | 0     | 1.2   | 0.87 |
| (B,C) | 1.2   | 0     | 0.98 |
| E     | 0.87  | 0.98  | 0    |

$$D_3((A,D), E) = 0.87$$

$$\delta((A,D), r) = \delta(E, r) = 0.435$$

$$\delta(r, r) = 0.435 - 0.425 = 0.01$$

$$D_4((A,D), E), (B,C)) = \min(D_3((A,D), (B,C)), D_3(E, (B,C)))$$

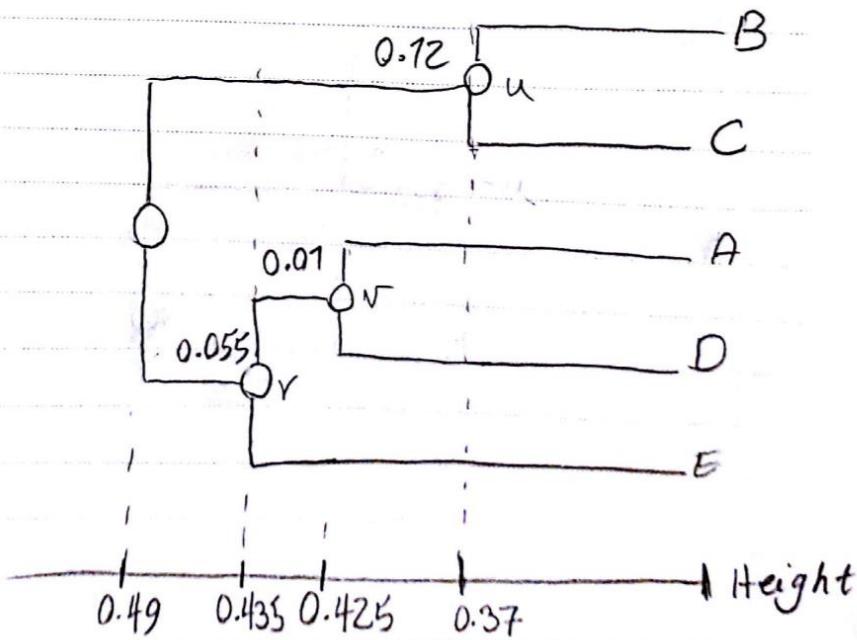
$$= \min(1.2, 0.98) = 0.98$$

|          | ((A,D), E) | (B,C) |
|----------|------------|-------|
| (A,D), E | 0          | 0.98  |
| (B,C)    | 0.98       | 0     |

$$\delta((B,C), h) = \delta((A,D), E), h) = 0.49$$

$$\delta(r, h) = 0.49 - 0.435 = 0.055$$

Dendrogram



$$D_1(B, C) = 0.74$$

Complete-link

$$\delta(B, u) = \delta(C, u) = 0.37$$

$$D_2((B, C), A) = \max(D_1(B, A), D_1(C, A)) = 2.44$$

$$D_2((B, C), D) = \max(D_1(B, D), D_1(C, D)) = 1.34$$

$$D_2((B, C), E) = \max(D_1(B, E), D_1(C, E)) = 1.4$$

|        | A    | (B, C) | D    | E    |
|--------|------|--------|------|------|
| A      | 0    | 2.44   | 0.85 | 2.04 |
| (B, C) | 2.44 | 0      | 1.34 | 1.4  |
| D      | 0.85 | 1.34   | 0    | 0.87 |
| E      | 2.04 | 1.4    | 0.87 | 0    |

$$D_2(A, D) = 0.85$$

$$\delta(A, v) = \delta(D, v) = 0.425$$

$$\delta(u, v) = 0.425 - 0.37 =$$

$$2.44 - 0.055$$

$$D_3((A, D), (B, C)) = \max(D_2((A, (B, C)), D_2(D, (B, C)))$$

$$D_3((A, D), E) = \max(D_2(\overline{A, E}), D_2(\overline{D, E})) = 2.04$$

|        | (A, D) | (B, C) | E    |
|--------|--------|--------|------|
| (A, D) | 0      | 2.44   | 2.04 |
| (B, C) | 2.44   | 0      | 1.4  |
| E      | 2.04   | 1.4    | 0    |

$$D_3((B, C), E) = 1.4$$

$$\delta(r, E) = \delta(r, (B, C)) = 0.7$$

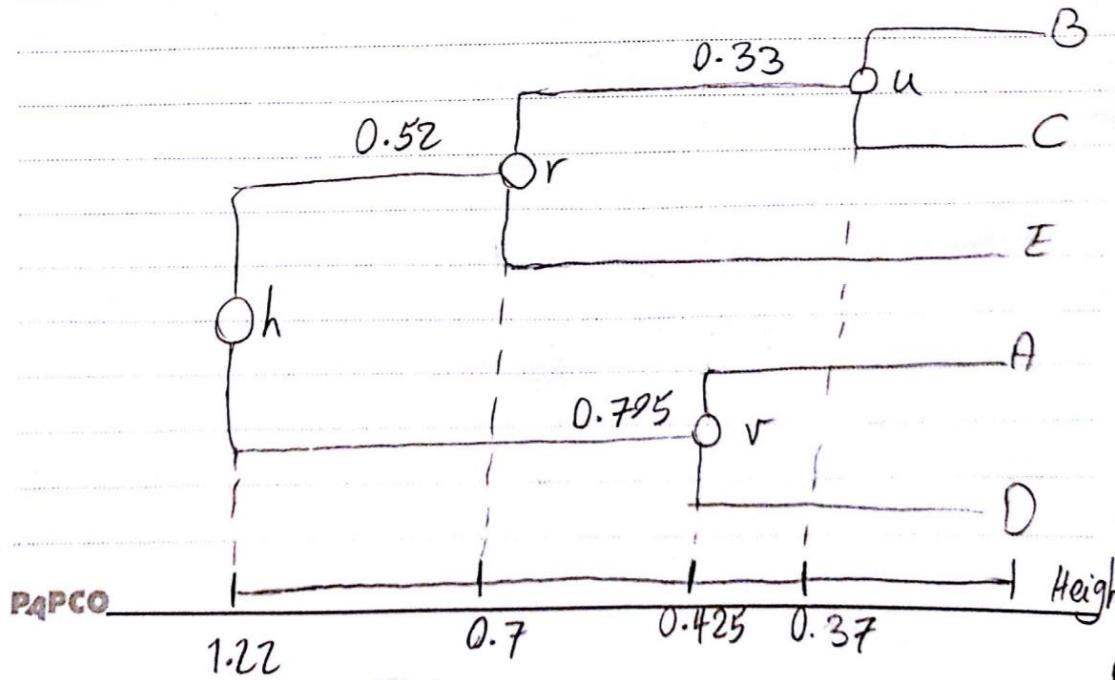
$$\delta(r, v) = 0.7 - 0.425 = 0.275$$

$$D_4 \left( \underbrace{((B, C), E), (A, D)}_{2.44} \right) = \max(D_3(E, (A, D)),$$

$$D_3 \left( ((B, C), (A, D)) \right) = 2.44$$

|             | (A, D) | ((B, C), E) | $\delta(((B, C), E), h) = \delta((A, D), h)$ |
|-------------|--------|-------------|--|
| (A, D)      | 0      | 2.44        | $= 1.22$                                     |
| ((B, C), E) | 2.44   | 0           | $\delta(h, r) = 1.22 - 0.7 = 0.52$           |

Dendrogram



# سوالات عملی

تمام کدها با شماره سوال ضمیمه شده است.

1) روی worm امتحان شده

کد kmean کاملا کامنت گذاری شده است.

```
39         #Updating Centroids by taking mean of Cluster it belongs to
40     temp_cent = x[points==idx].mean(axis=0)
41     centroids.append(temp_cent)
42
43     centroids = np.vstack(centroids) #Updated Centroids
44
45     distances = cdist(x, centroids , 'euclidean')
46     points = np.array([np.argmin(i) for i in distances])
47
48     return points
49
50     #return np.mean(distances)
51 #Load Data
52 worms = pd.read_csv('worms.csv')
53 print(worms)
54 worms = worms.iloc[:, 1:]
55 #worms = worms.to_numpy()
56
57 #Applying our function
58
59 #Visualize the results
60 wormsnsp = worms.to_numpy()
61 print(wormsnsp)
62 label = kmeans(wormsnsp,8,100)
63
64 u_labels = np.unique(label)
65 for i in u_labels:
66     plt.scatter(wormsnsp[label == i , 0] , wormsnsp[label == i , 1] , s=0.001, label =
67 plt.legend()
68 plt.show()
69
70 #worms.plot(x ='X' , y='Y' , s=0.001 ,kind = 'scatter')
71 #plt.legend()
72 #plt.show()
73
74
75
```

In [45]:

```
[105595 385595 5567.9 3585.4
105596 105596 3383.9 3897.8
105597 105597 3259.6 2970.7
105598 105598 2688.7 2958.6
105599 105599 4095.2 2341.0]
```

Out [45]:

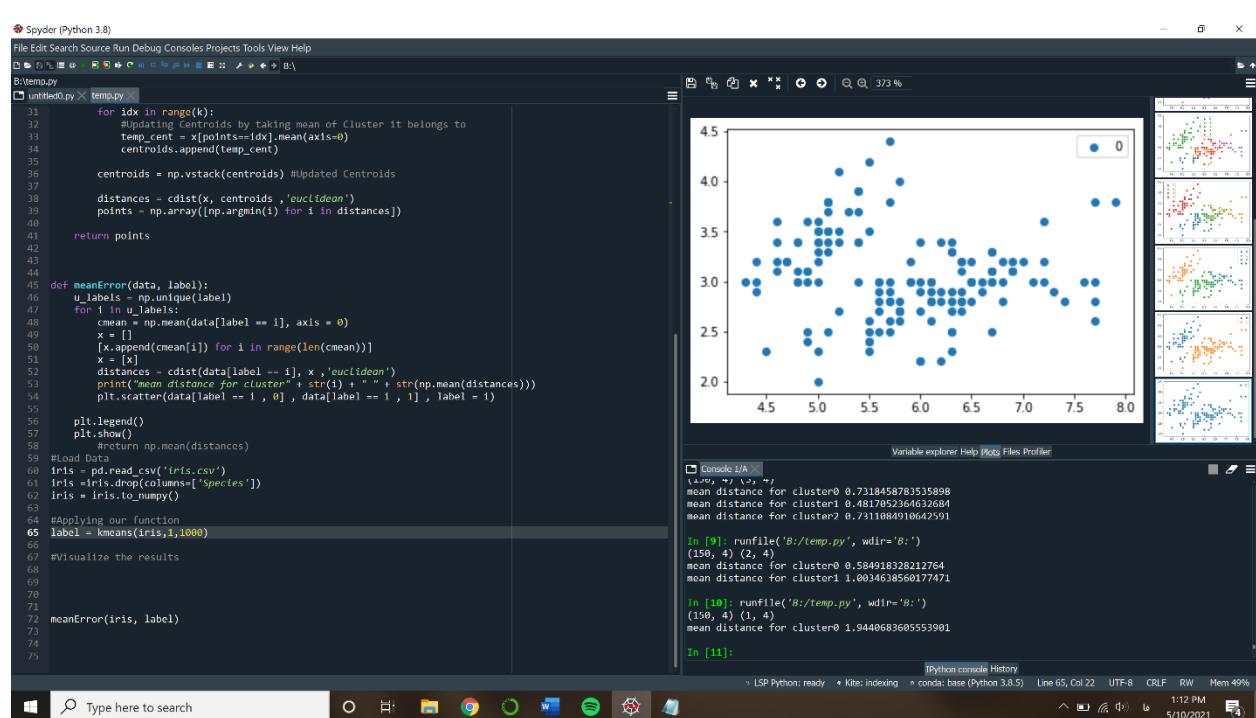
```
[105598 rows x 3 columns]
[[4260.1 3476.1
[4276.3 3475.6]
[4260.2 3475.3]
...
[3259.6 2970.7]
[2688.7 2998.6]
[4095.2 2344.1]
(105600, 2) (8, 2)]
```

LSP Python: ready \* Kite: indexing \* conda: base (Python 3.8.5) Line 60, Col 27 UTF-8 CRLF RW Mem 60% 11:51 AM 4/23/2021

ابتدا به طور تصادفی مراکز انتخاب میشوند، سپس نقاط که با کمترین فاصله از مراکز هستند `initiate` میشوند، سپس به تعداد `Iteration` ها مراکز با میانگین گیری آپدیت میشوند و فاصله های جدید محاسبه میشوند و دوباره نقاط جدید مقدار

دهی میشوند. در نهایت نقاط لیبل گذاری شده برگردانده میشود.

2) با یک حلقه روی تعداد خوشه ها شروع میشود، برای هر خوشه، میانگین خوشه، مرکز را بدست میاوریم، سپس فاصله ای اقلیدسی هر میانگین با مقادیر خوشه را بدست میاوریم.



K=2

Spyder (Python 3.8)

```

31     for idx in range(k):
32         #Updating Centroids by taking mean of Cluster it belongs to
33         temp_cent = x[points==idx].mean(axis=0)
34         centroids.append(temp_cent)
35
36     centroids = np.vstack(centroids) #Updated Centroids
37
38     distances = cdist(x, centroids, 'euclidean')
39     points = np.array([np.argmin(i) for i in distances])
40
41     return points
42
43
44
45 def meanError(data, label):
46     u_labels = np.unique(label)
47     for i in u_labels:
48         cmean = np.mean(data[label == i], axis = 0)
49         x = []
50         [x.append(cmean[i]) for i in range(len(cmean))]
51         x = [x]
52         distances = cdist(data[label == i], x, 'euclidean')
53         print("mean distance for cluster" + str(i) + " " + str(np.mean(distances)))
54         plt.scatter(data[label == i , 0] , data[label == i , 1] , label = i)
55
56     plt.legend()
57     plt.show()
58     return np.mean(distances)
59
#Load Data
60 iris = pd.read_csv('iris.csv')
61 iris = iris.drop(columns=['Species'])
62 iris = iris.to_numpy()
63
64 #Applying our function
65 label = kmeans(iris,2,1000)
66
67 #Visualize the results
68
69
70
71
72 meanError(iris, label)
73
74
75

```

Variable explorer Help Jupyter Files Profiler

Console 1/A

```

mean distance for cluster0 0.4817052364632684
mean distance for cluster1 0.7318458783535898
mean distance for cluster2 0.4817052364632684
mean distance for cluster3 0.7311084910642591

```

In [8]: runfile('B:/temp.py', wdir='B:')

```

(150, 4) (3, 4)
mean distance for cluster0 0.7318458783535898
mean distance for cluster1 0.4817052364632684
mean distance for cluster2 0.7311084910642591
mean distance for cluster3 0.584918328212764
mean distance for cluster0 0.584918328212764
mean distance for cluster1 1.003463856017747
mean distance for cluster2 0.7311084910642591
mean distance for cluster3 0.4817052364632684

```

In [9]:

LSP Python: ready    kite: indexing    conda: base (Python 3.8.5) Line 65, Col 22    UTF-8    CRLF    RW    Mem 50%

1:11 PM 5/10/2021

# K=3

Spyder (Python 3.8)

```

31     for idx in range(k):
32         #Updating Centroids by taking mean of Cluster it belongs to
33         temp_cent = x[points==idx].mean(axis=0)
34         centroids.append(temp_cent)
35
36     centroids = np.vstack(centroids) #Updated Centroids
37
38     distances = cdist(x, centroids, 'euclidean')
39     points = np.array([np.argmin(i) for i in distances])
40
41     return points
42
43
44
45 def meanError(data, label):
46     u_labels = np.unique(label)
47     for i in u_labels:
48         cmean = np.mean(data[label == i], axis = 0)
49         x = []
50         [x.append(cmean[i]) for i in range(len(cmean))]
51         x = [x]
52         distances = cdist(data[label == i], x, 'euclidean')
53         print("mean distance for cluster" + str(i) + " " + str(np.mean(distances)))
54         plt.scatter(data[label == i , 0] , data[label == i , 1] , label = i)
55
56     plt.legend()
57     plt.show()
58     return np.mean(distances)
59
#Load Data
60 iris = pd.read_csv('iris.csv')
61 iris = iris.drop(columns=['Species'])
62 iris = iris.to_numpy()
63
64 #Applying our function
65 label = kmeans(iris,3,1000)
66
67 #Visualize the results
68
69
70
71
72 meanError(iris, label)
73
74
75

```

Variable explorer Help Jupyter Files Profiler

Console 1/A

```

mean distance for cluster0 0.5266313680552394
mean distance for cluster1 0.6999847088484026
mean distance for cluster2 0.569026876742275
mean distance for cluster3 0.4817052364632684

```

In [7]: runfile('B:/temp.py', wdir='B:')

```

(150, 4) (3, 4)
mean distance for cluster0 0.5266313680552394
mean distance for cluster1 0.6999847088484026
mean distance for cluster2 0.569026876742275
mean distance for cluster3 0.4817052364632684

```

In [8]: runfile('B:/temp.py', wdir='B:')

```

(150, 4) (3, 4)
mean distance for cluster0 0.7318458783535898
mean distance for cluster1 0.4817052364632684
mean distance for cluster2 0.7311084910642591
mean distance for cluster3 0.4817052364632684

```

In [9]:

LSP Python: ready    kite: indexing    conda: base (Python 3.8.5) Line 65, Col 22    UTF-8    CRLF    RW    Mem 50%

1:11 PM 5/10/2021

# K=4

```

31
32     for idx in range(k):
33         #Updating Centroids by taking mean of Cluster it belongs to
34         temp_cent = x[points==idx].mean(axis=0)
35         centroids.append(temp_cent)
36
37     centroids = np.vstack(centroids) #Updated Centroids
38
39     distances = cdist(x, centroids, 'euclidean')
40     points = np.array([np.argmin(i) for i in distances])
41
42     return points
43
44
45 def meanError(data, label):
46     u_labels = np.unique(label)
47     for i in u_labels:
48         cmean = np.mean(data[label == i], axis = 0)
49         x = []
50         [x.append(cmean[i]) for i in range(len(cmean))]
51         x = [x]
52         distance = cdist(data[label == i], x, 'euclidean')
53         print("mean distance for cluster" + str(i) + " " + str(np.mean(distances)))
54         plt.scatter(data[label == i , 0] , data[label == i , 1] , label = i)
55
56     plt.legend()
57     plt.show()
58     return np.mean(distances)
59
#Load Data
60 iris = pd.read_csv('iris.csv')
61 iris = iris.drop(columns=['Species'])
62 iris = iris.to_numpy()
63
64 #Applying our function
65 label = kmeans(iris,4,1000)
66
67 #visualize the results
68
69
70
71
72 meanError(iris, label)
73
74
75

```

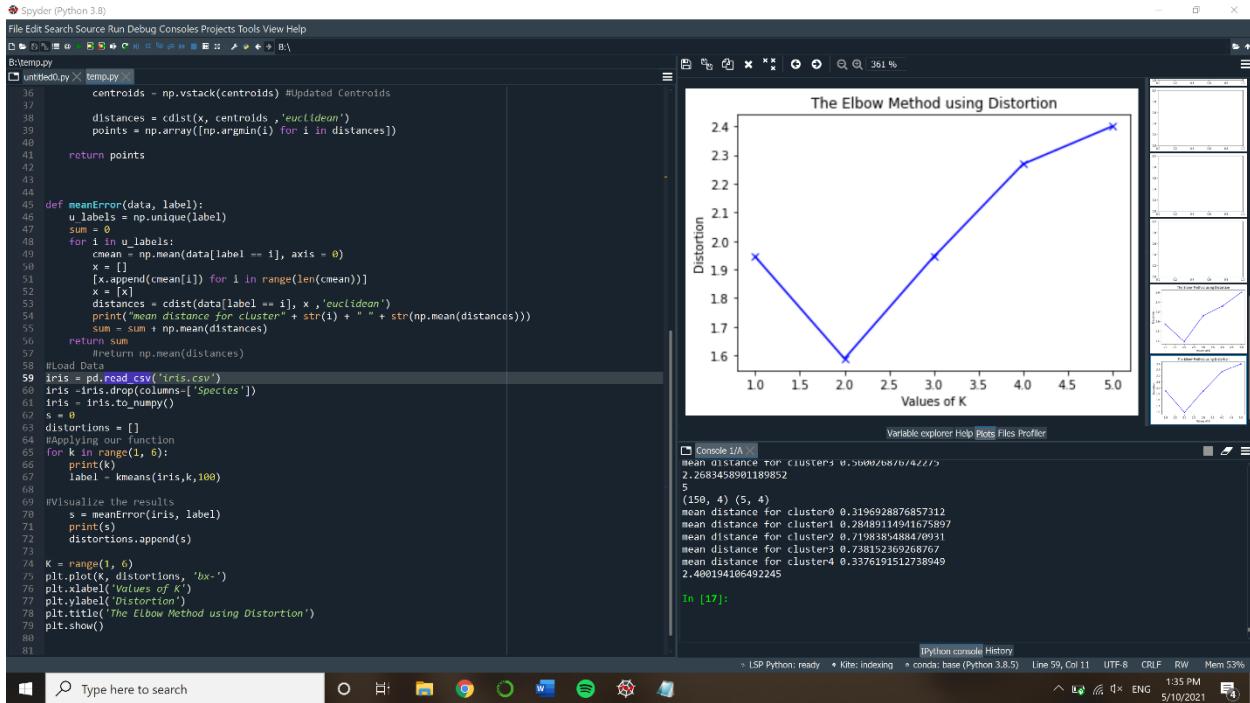
K=5

```

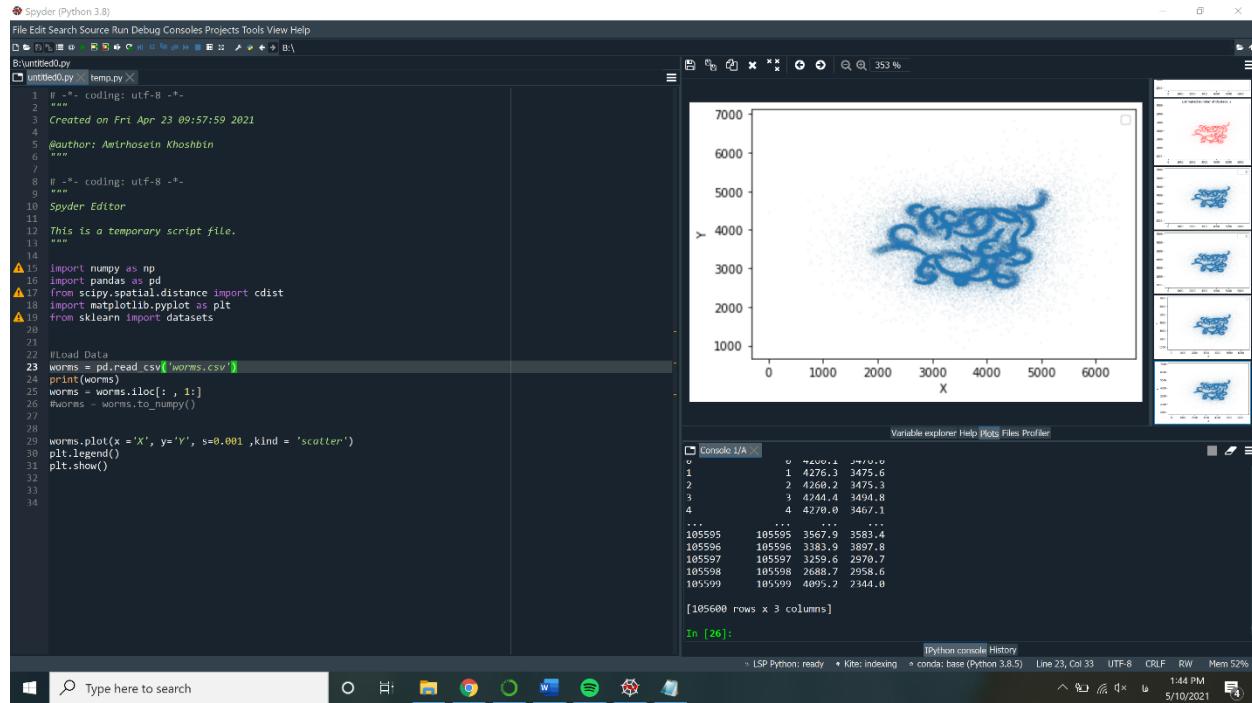
31
32     for idx in range(k):
33         #Updating Centroids by taking mean of Cluster it belongs to
34         temp_cent = x[points==idx].mean(axis=0)
35         centroids.append(temp_cent)
36
37     centroids = np.vstack(centroids) #Updated Centroids
38
39     distances = cdist(x, centroids, 'euclidean')
40     points = np.array([np.argmin(i) for i in distances])
41
42     return points
43
44
45 def meanError(data, label):
46     u_labels = np.unique(label)
47     for i in u_labels:
48         cmean = np.mean(data[label == i], axis = 0)
49         x = []
50         [x.append(cmean[i]) for i in range(len(cmean))]
51         x = [x]
52         distance = cdist(data[label == i], x, 'euclidean')
53         print("mean distance for cluster" + str(i) + " " + str(np.mean(distances)))
54         plt.scatter(data[label == i , 0] , data[label == i , 1] , label = i)
55
56     plt.legend()
57     plt.show()
58     return np.mean(distances)
59
#Load Data
60 iris = pd.read_csv('iris.csv')
61 iris = iris.drop(columns=['Species'])
62 iris = iris.to_numpy()
63
64 #Applying our function
65 label = kmeans(iris,5,1000)
66
67 #visualize the results
68
69
70
71
72 meanError(iris, label)
73
74
75

```

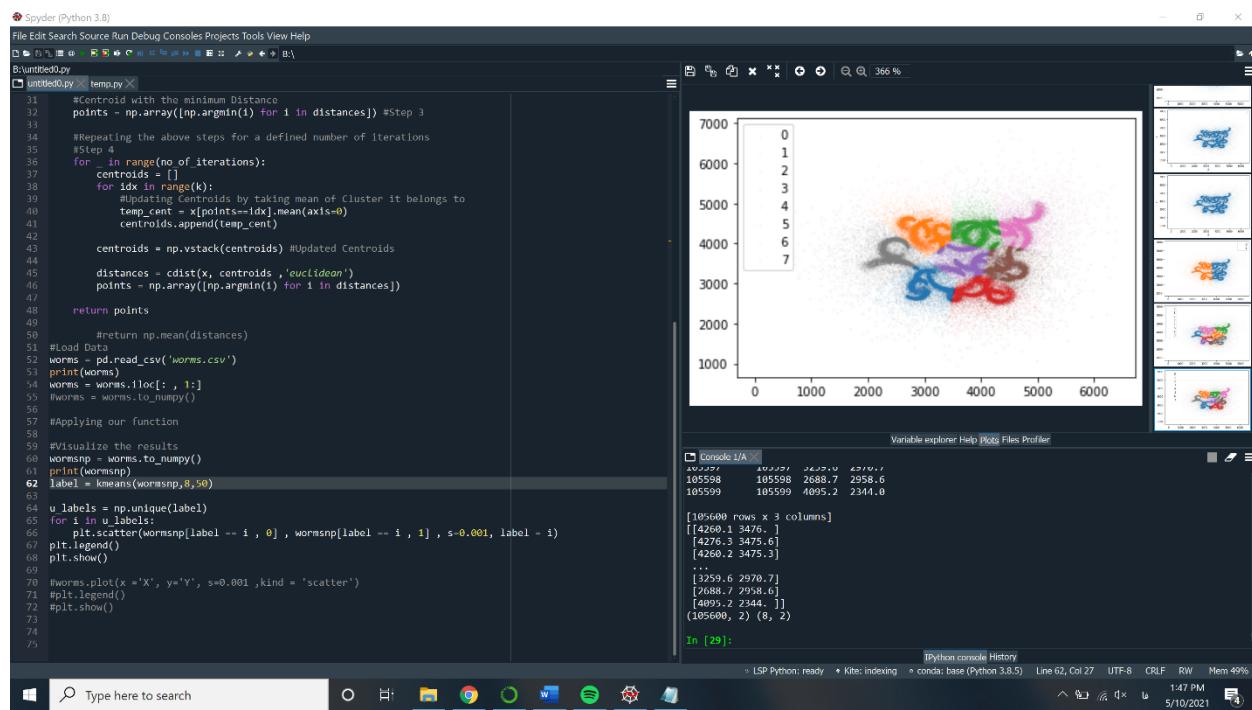
## ۴) متغیر y جمع MAE خوش‌ها است، واضح است با $k=2$ distortion کمترین است.



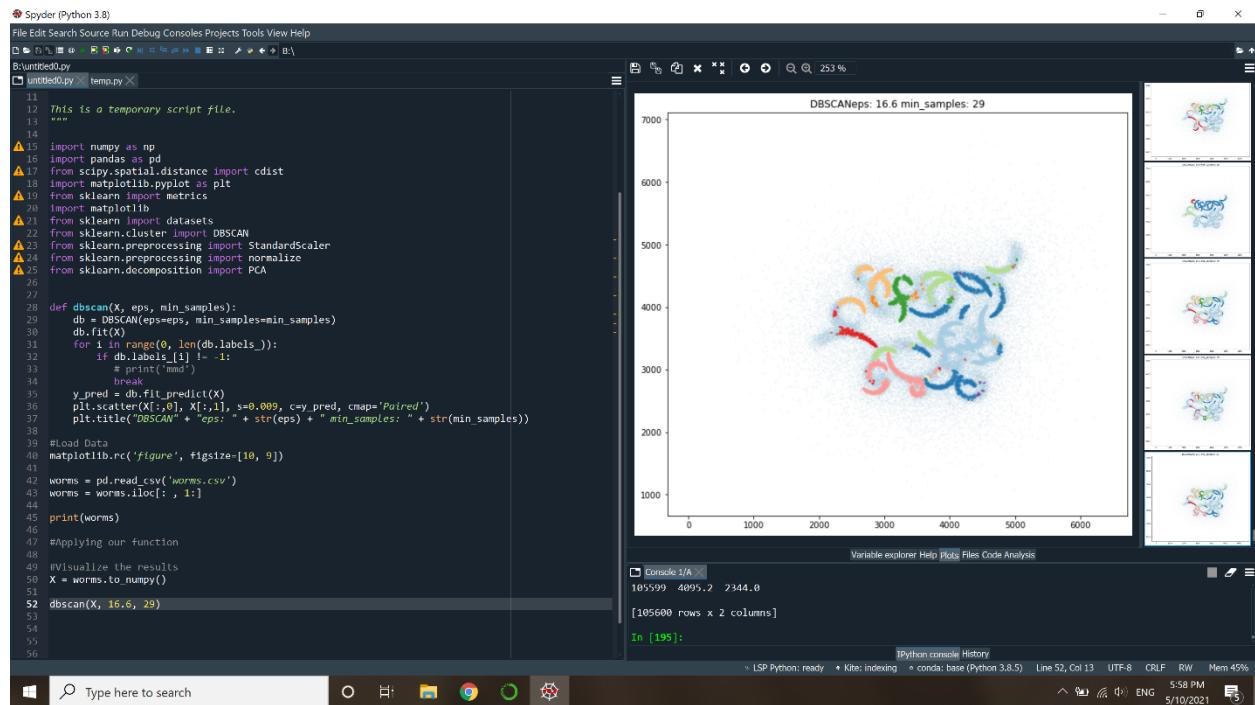
(5)



6) با  $k=8$  خوشبندی شده است.



7) با چیزی اینجوری  
min\_sample=29 و eps=16.6 درآمد.



The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a script named 'temp.py' containing Python code for performing DBSCAN clustering on 'worms.csv' data. The code includes imports for numpy, scipy, matplotlib, sklearn, and PCA, followed by a function 'dbscan' that performs the clustering and plots the results. The plot window on the right shows a scatter plot titled 'DBSCANeps: 16.6 min\_samples: 29'. The plot displays several distinct clusters of points colored in red, green, blue, and orange, representing different worm species. The x-axis ranges from 0 to 6000, and the y-axis ranges from 1000 to 7000. Below the plot, the IPython console shows the command 'dbscan(X, 16.6, 29)' and its output: '[105600 rows x 2 columns]'. The status bar at the bottom indicates the system is ready, with Python 3.8.5, Line 52, Col 13, UTF-8, CRLF, and Mem 45%.

```

Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
B:\untitled0.py temp.py
11
12 This is a temporary script file.
13 """
14
15 import numpy as np
16 from scipy.spatial.distance import cdist
17 import matplotlib.pyplot as plt
18 from sklearn import metrics
19 import matplotlib
20 from sklearn import datasets
21 from sklearn.cluster import DBSCAN
22 from sklearn.preprocessing import StandardScaler
23 from sklearn.preprocessing import normalize
24 from sklearn.decomposition import PCA
25
26
27 def dbscan(X, eps, min_samples):
28     db = DBSCAN(eps=eps, min_samples=min_samples)
29     db.fit(X)
30     for i in range(0, len(db.labels_)):
31         if db.labels_[i] == -1:
32             # print("mnd")
33             break
34
35     y_pred = db.fit_predict(X)
36     plt.scatter(X[:,0], X[:,1], s=0.009, c=y_pred, cmap='Paired')
37     plt.title("DBSCAN" + "eps: " + str(eps) + " min_samples: " + str(min_samples))
38
39 #Load Data
40 matplotlib.rcParams['figure', figsize=[10, 9])
41
42 worms = pd.read_csv('worms.csv')
43 worms = worms.iloc[:, 1:]
44
45 print(worms)
46
47 #Applying our function
48
49 #Visualize the results
50 X = worms.to_numpy()
51
52 dbscan(X, 16.6, 29)

```