Miniprojekt 4

Na początku stworzyłem moduł **prepare_data.py**, w którym oprócz wczytywania danych z pliku mogę przeskalować dane metodami: **min-max1, min-max2, standaryzacja.** Parametry skalowania, czyli średnia, odchylenie standardowe, minimum, maximum wyznaczam na podstawie zbioru treningowego, a następnie stosuję w procesie skalowania dla zbioru treningowego, testowego i walidacyjnego.

Następnie stworzyłem moduły dla rozważanych modeli:

- K-średnie (K-means),
- grupowanie hierarchiczne (hierarchical clustering),
- klasteryzacja spektralna (spectral clustering).

W folderze K-means zaimplementowałem algorytm k-średnich dla losowo wybranych współrzędnych początkowych centroidów, a także wersję k-means++, w której początkowe centroidy są bardziej porozrzucane.

W folderze hierarchical clustering zaimplementowałem natomiast algorytm grupowania hierarchicznego z wykorzystaniem metryki euklidesowej i 4 różnych metod łączenia klastrów:

- pojedyncze (single),
- pełne (complete),
- średnie (average),
- Centroidalne (centroid).

W folderze spectral clustering zaimplementowałem algorytm klasteryzacji spektralnej, w której do stworzenia grafu podobieństwa mogę zastosować otoczkę epsilonową z metryką euklidesową, jądro gaussowskie, a także odwrotność odległości. W moich rozważaniach wykorzystałem jednak głównie otoczkę epsilonową, ponieważ z łatwością mogłem znaleźć epsilon dający satysfakcjonujące wyniki.

Otrzymane klastry porównywałem z danymi klasyfikacjami y za pomocą miary **purity**, czyli dla każdego klastra wybierałem taką klasę z y, żeby zmaksymalizować liczbę poprawnych klasyfikacji, a następnie dzieliłem przez liczbę obserwacji.

$$purity(\Omega, C) = \frac{1}{m}\sum_{k} \max_{j} |\omega_k \cap c_j|$$

Oczywiście stosując tę miarę kilka klastrów mogło otrzymać tą samą etykietę.

K-means

Wyniki dla powyższego modelu sprawdzałem na 10 przebiegach algorytmu, zarówno w wersji k-means jak o kmeans++.

Liczbę klastrów wyznaczałem za pomocą reguły łokcia dla wykresu **inertia**, czyli sumy kwadratów odległości obserwacji od centroidu ich klastru.

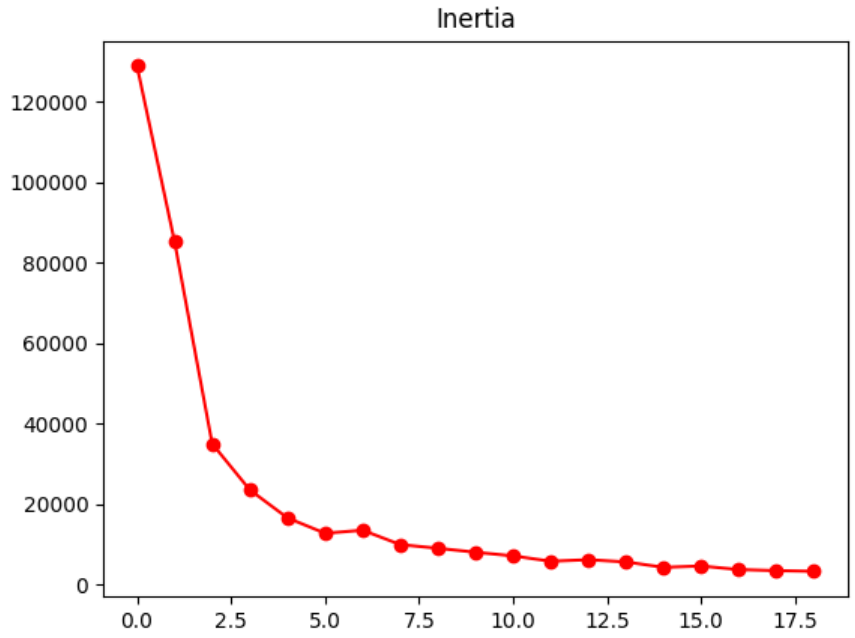
Jak będzie można zobaczyć w wynikach porównawczych z klasyfikacją y, algorytm kmeans++ często dawał lepsze wyniki od kmeans, a także był stabilniejszy w swojej klasteryzacji. Zdarzały się jednak przypadki - głownie w sytuacji gdzie oba algorytmy słabo przybliżały klasyfikację, w których algorytm kmeans okazywał się lepszy. Mogło się pewnie okazywać wtedy, że swoją początkową czystą losowością lepiej udawało mu się uformować z danych klastry.

Warto również zauważyć, że dla naszych danych algorytmom wystarczało średnio jedynie ok. 10 iteracji pętli, żeby zakończyć swoje działanie - tzn. wytworzyć centroidy, dla których klasteryzacja nie zmieniała się już w następnym obiegu.

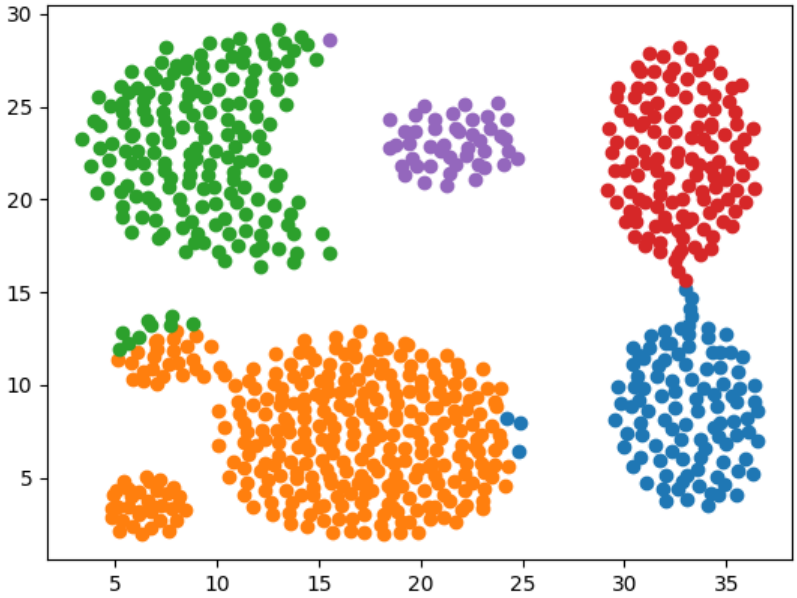
Omówimy teraz po kolei otrzymane wyniki dla naszych danych

1. Dane_2D_1.txt

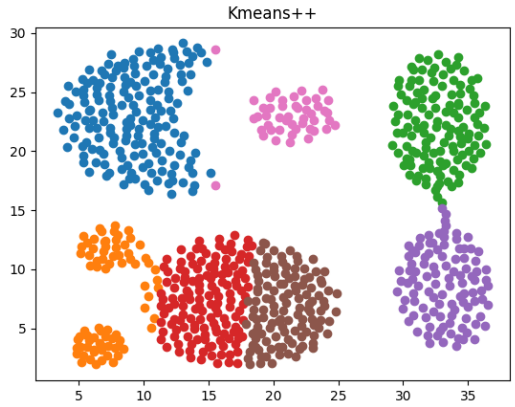
Stosując regułę łokcia dla wykresu błędu na tych danych można by wybrać jako k liczbę 4/5.

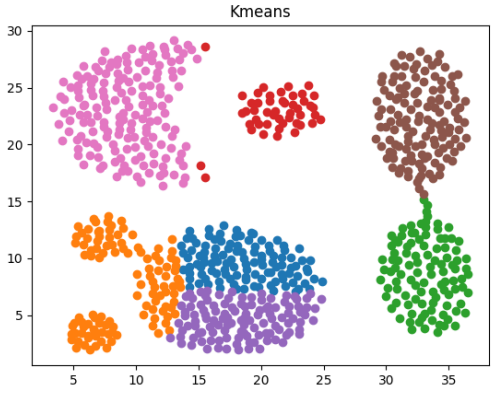


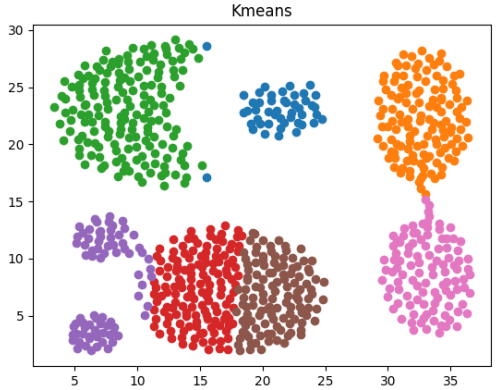
Jeżeli wybierzemy k=5 to otrzymamy już całkiem dobre przybliżenie klas, ale wtedy kształt w lewym dolnym rogu jest praktycznie zawsze 1 klastrem.



Wiedząc, że mamy tutaj 7 klas i biorąc k=7 w naszym algorytmie otrzymamy





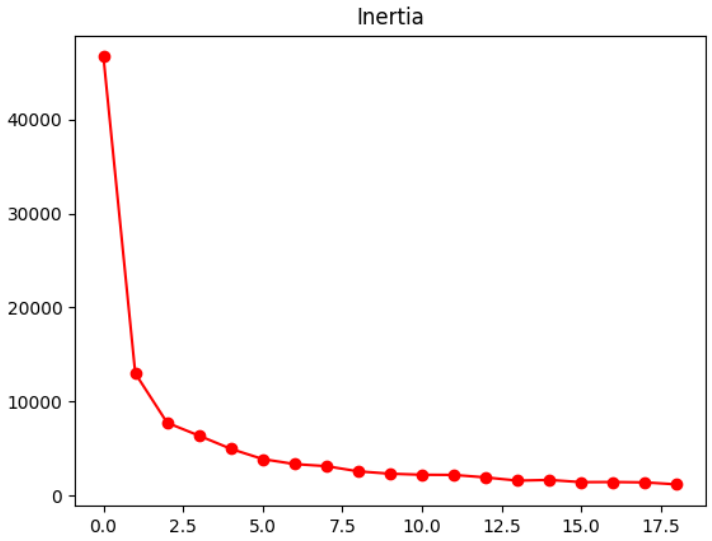


Jak można zauważyć klasteryzacja nie oddaje tu w 100% klasyfikacji. Problemem jest kształt w lewym dolnym rogu, który niezależnie od algorytmu i przebiegu jest dzielony w specyficzny sposób. Można tu zauważyć, że pomimo faktu dzielenie obserwacji na 7 klastrów wykorzystujemy jedynie 6 klas z y.

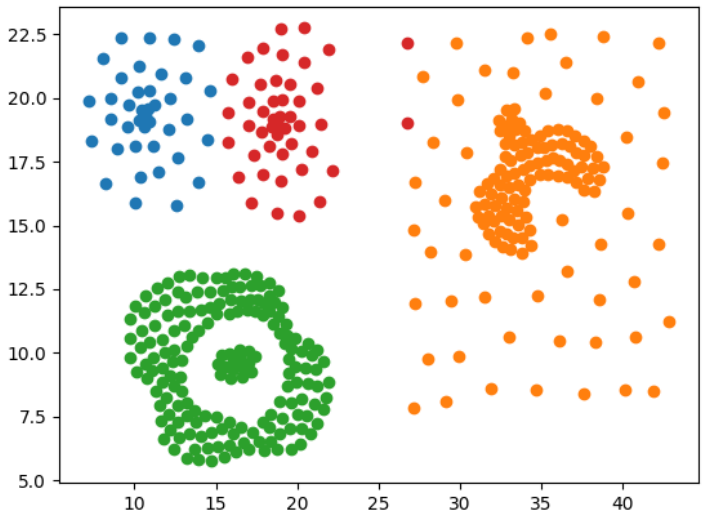
Wartość purity w tym przypadku jest taka sama dla kmeans i kmeans++ i jest równa = 0.939086.

2. Dane_2D_2.txt

W tym przypadku patrząc na wykres błędu możemy wybrać k=4.

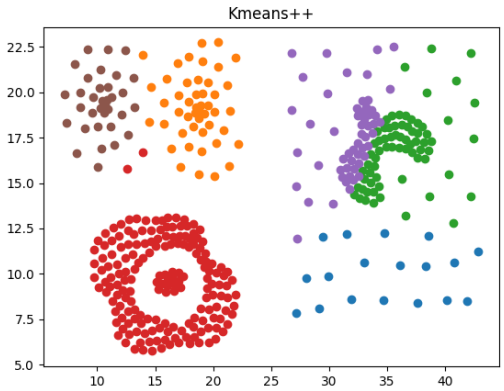


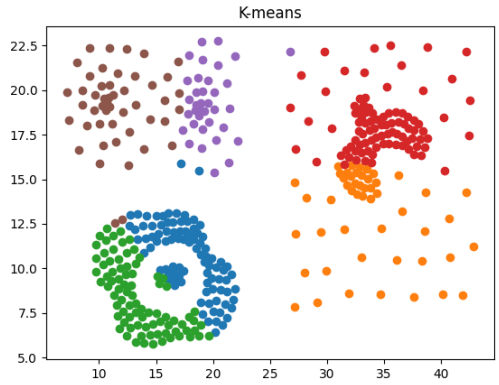
Otrzymamy wtedy następujący wykres



Ładnie zostały oddzielone 4 główne komponenty. Problematyczne w tym przypadku są te klasy, które na rysunku zawierają wewnątrz inną klasę. Tych klas nie uda nam się ładnie rozdzielić za pomocą tej klasteryzacji.

W tym pliku mamy 6 klas, dlatego też stosując nasze algorytmy dla k=6 dostaniemy

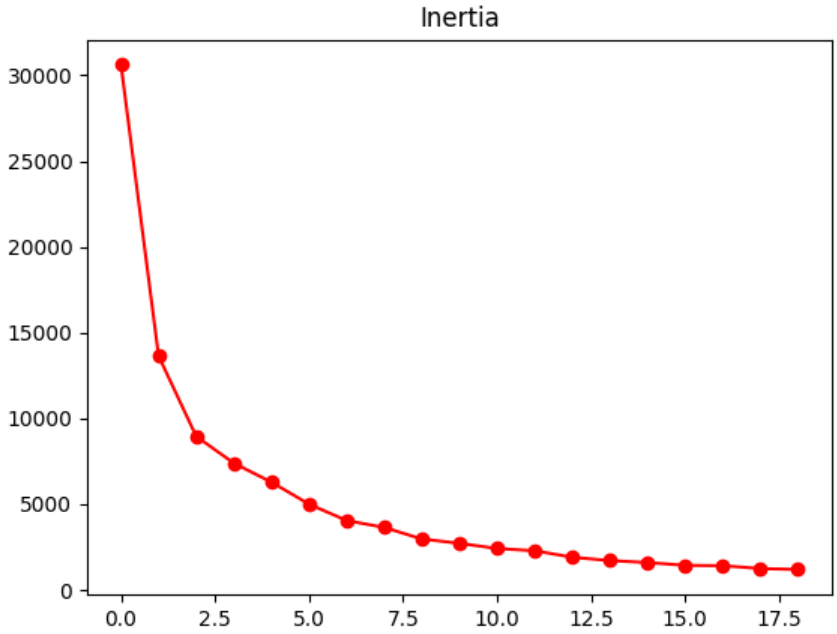




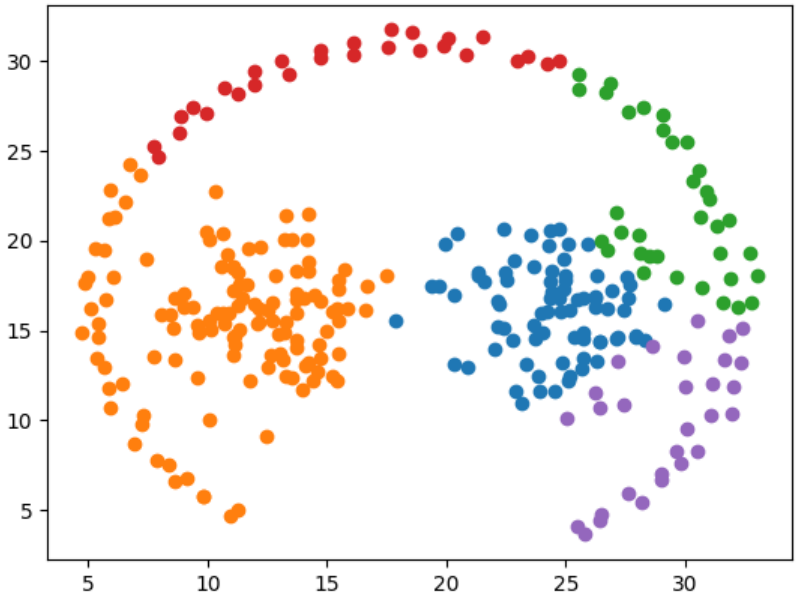
Algorytmy próbowały rozdzielić lewy dolny komponent na 2, a prawy na 2/3 części. Nie dawało to jednak jakiegoś pozytywnego rezultatu. Tym razem jednak kmeans++ było lepsze i osiągnęło purity 0.86967, a kmeans 0.83208.

3. Dane_2D_3.txt

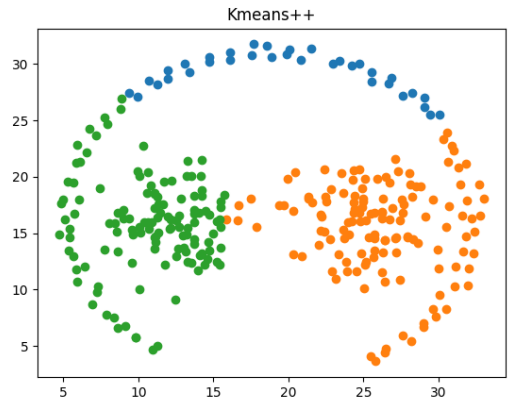
Z uwagi na specjalny kształt danych, który współgra dobrze z centroidami można zaobserwować na wykresie inertia, że ciężko zdecydować się na k, bo można go wybrać z przedziału 3 do 7

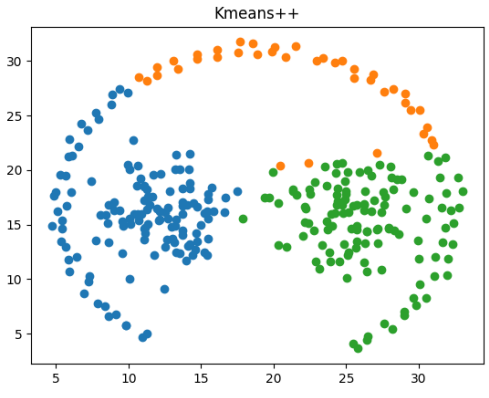


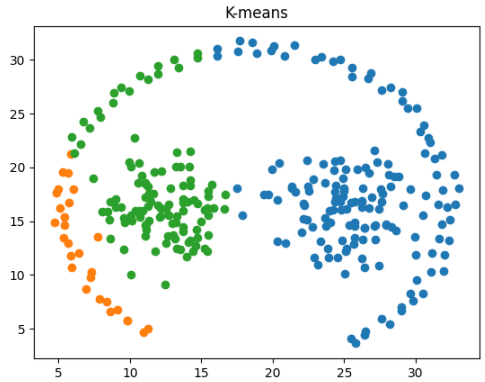
Dla k=5 mamy wykres

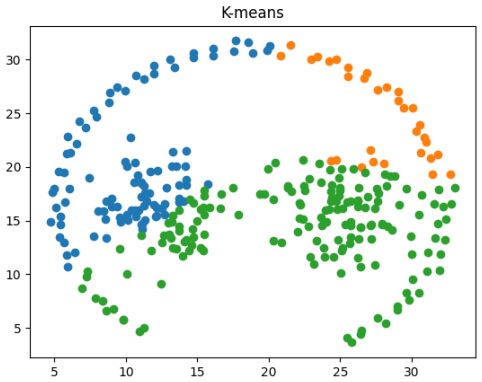


Tym razem problemem jest pierścień wokół 2 skupisk danych. Algorytm kmeans najchętniej podzieliłby ten pierścień na kilka mniejszych.
Stosując k=3, jak liczba pierwotnych klas dostaniemy





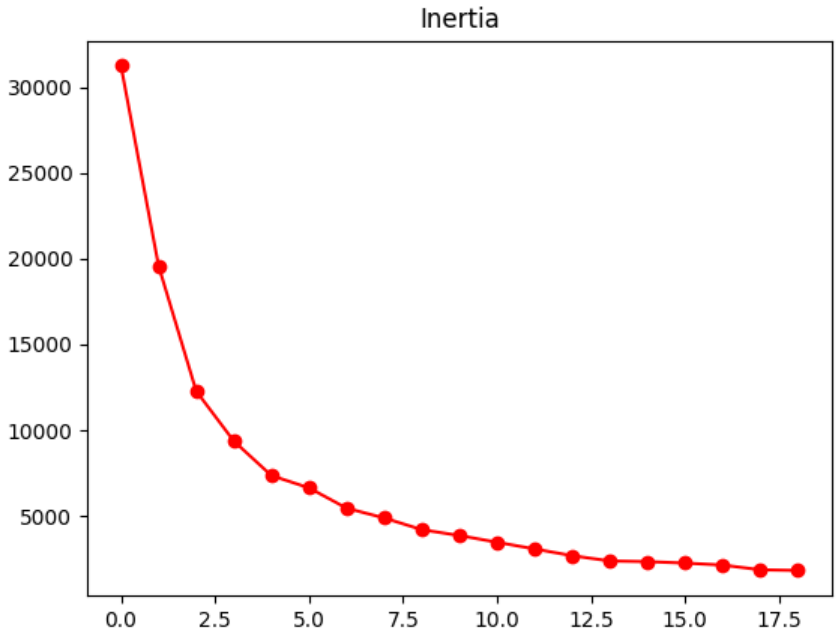


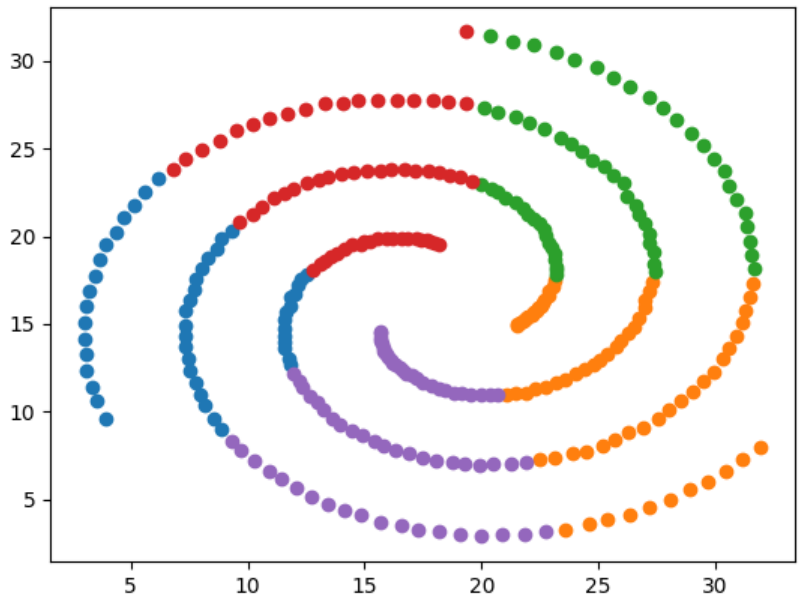


Jak można zauważyć algorytmy nie radzą sobie z tymi danymi, co skutkuje w niższym niż wcześniej purity – dla kmeans 0.74333, a kmeans++ 0.74666.

4. Dane_2D_4.txt

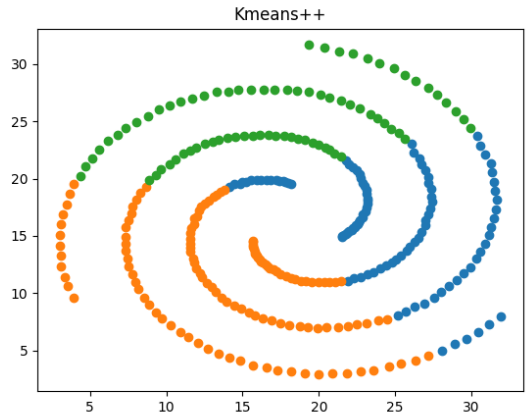
Dla tych danych jest jeszcze gorzej niż poprzednio. Z wykresu błędu możemy szacować k jako np. 5.

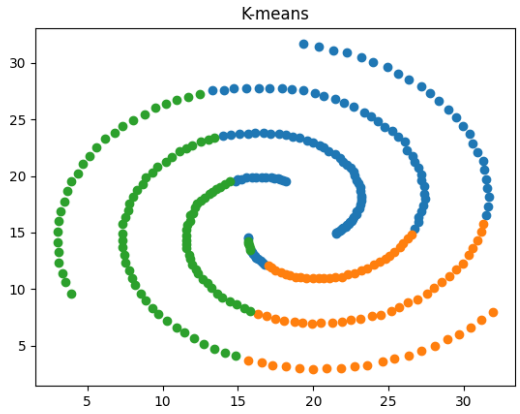




Otrzymany podział jest kompletnie inny od zamierzonego. W celu zminimalizowania inertia, algorytmy próbowały podzielić spirale na wycinki kołowe

Stosując k=3 jak oczekiwana liczba klas otrzymamy

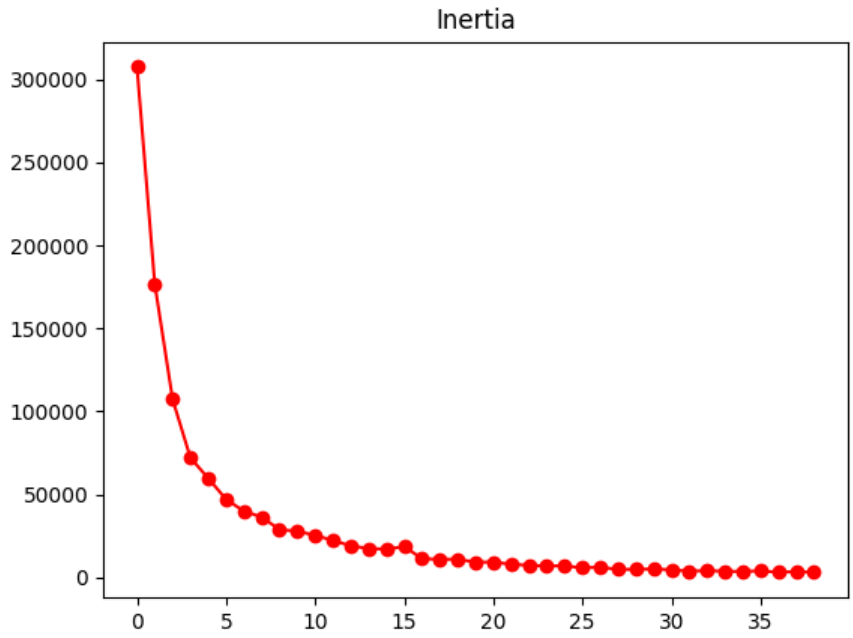




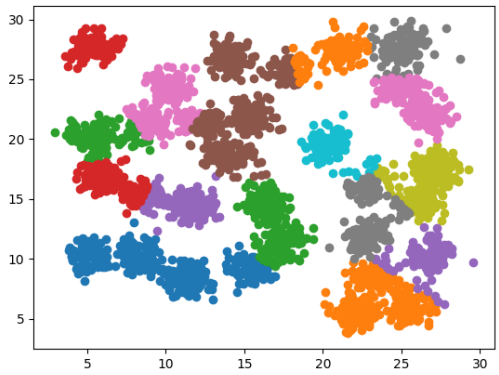
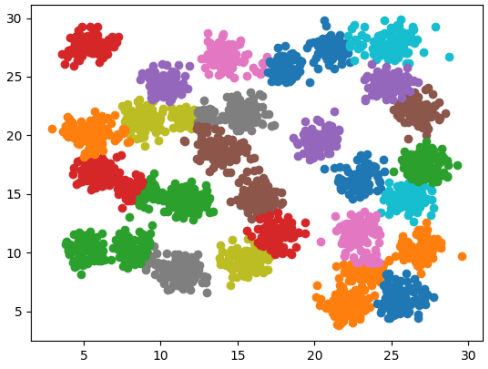
Wynik purity jak się można spodziewać jest naprawdę niski – dla kmeans jest odrobinę wyższy i wynosi 0.34935, a dla kmeans++ to 0.346153.

5. Dane_2D_5.txt

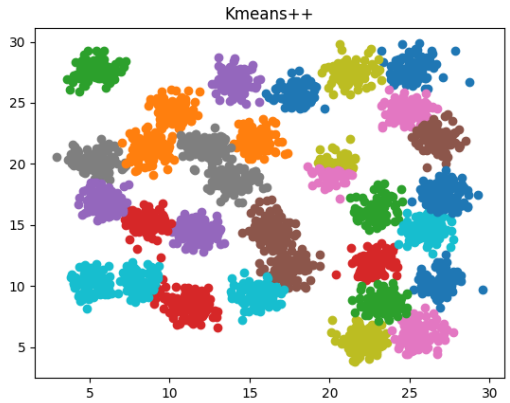
Plik ten jest znacząco większy od poprzednich, a także zawiera więcej klas. Biorąc pod uwagę skalę wykresu to patrząc na niego samego ciężej jest wybrać k według zasady łokcia. Trzeba też szczególniej przejrzeć wartości liczbowe, a także mieć na względzie, że zwykle chcemy wybrać jak najmniejsze k.

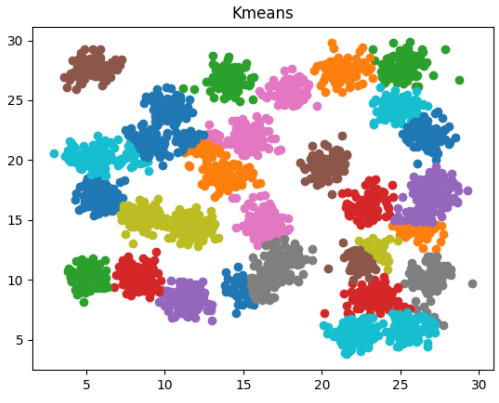


Po przeanalizowaniu danych liczbowych stwierdziłem, że ciekawymi kandydatami na k będzie 18 i 24

Już dla takich k można zaobserwować ładne dopasowywanie się klastrów do skupisk punktów. Analizując sam wykres błędu nie zgadłbym jednak, że mamy mieć tutaj aż 31 klas.

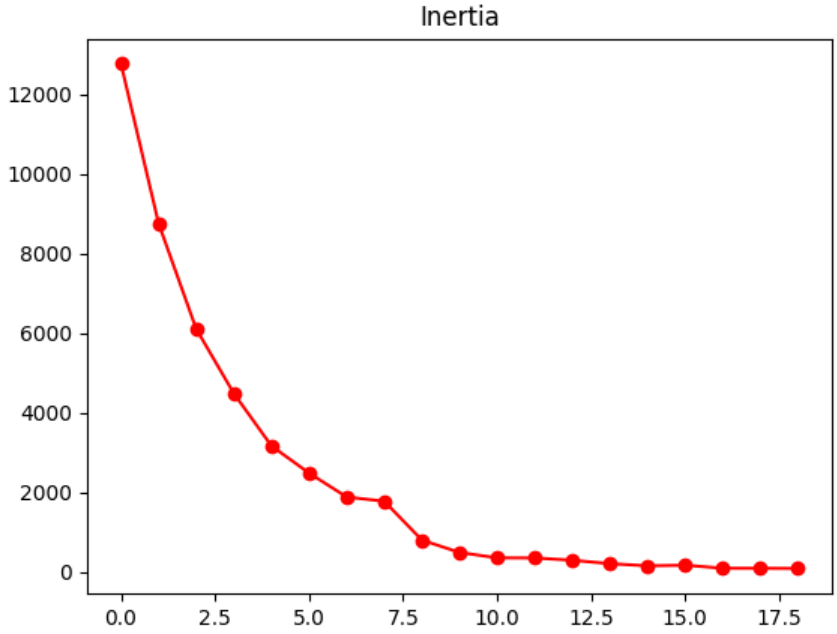




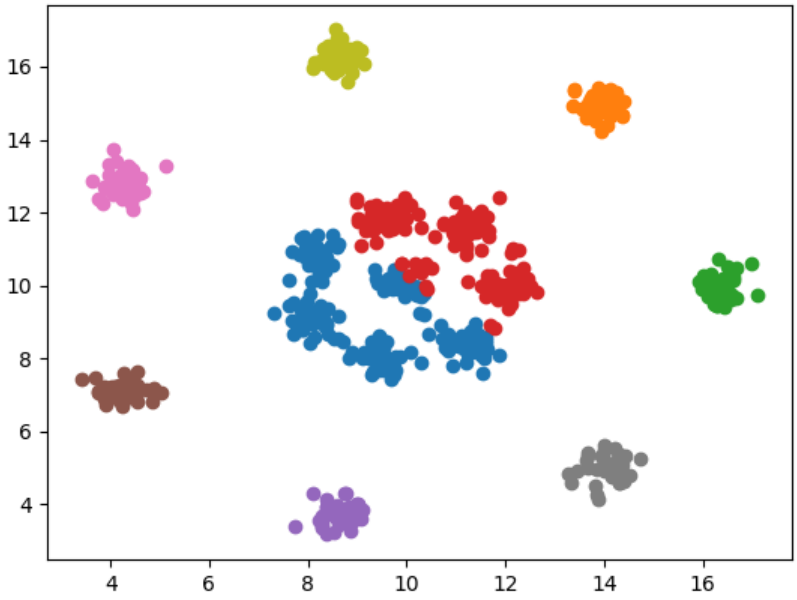
Po zastosowaniu takiego k, klastry w bardzo dobry sposób wpasowują się w kształt danych plamek, co skutkuje naprawdę wysokim współczynnikiem purity, biorąc pod uwagę rozmiar danych. Kmeans++ było lepsze i osiągnęło 0.947096, a kmeans natomiast 0.88225.

6. Dane_2D_6.txt

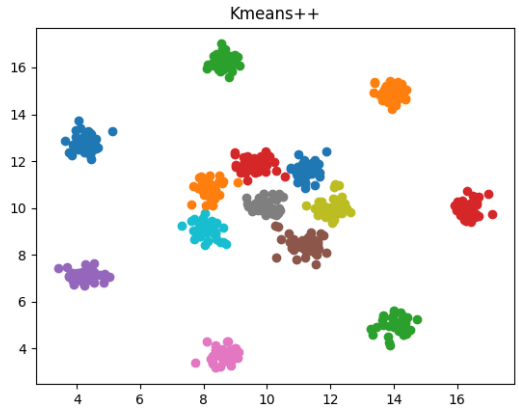
Te dane są trochę podobne do wcześniejszych pod względem wyglądu. Dlatego też nasze wyniki będą równie satysfakcjonujące co w 5. Plik pod względem rozmiaru jest mniejszy niż 5, ale cały czas jego liczba klas jest większa niż zwykle.

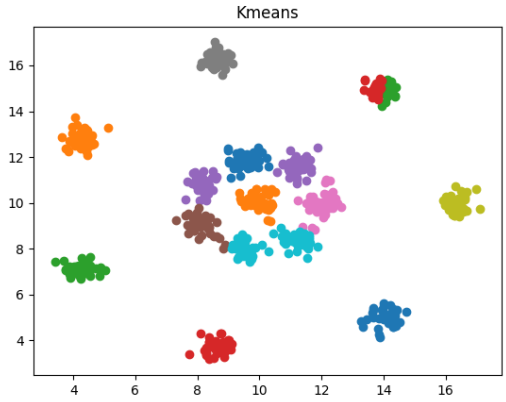


Widząc wykres od razu chciałem spróbować k=9, ponieważ można na nim zaobserwować spory spadek błędu.



Jak się okazuje jest to spowodowane faktem, że 7 klastrów dopasowuje się do zewnętrznych kół, a wewnętrzny fragment został podzielony na 2.
Stosując k=15 jak oczekiwana liczba klas dostaniemy.

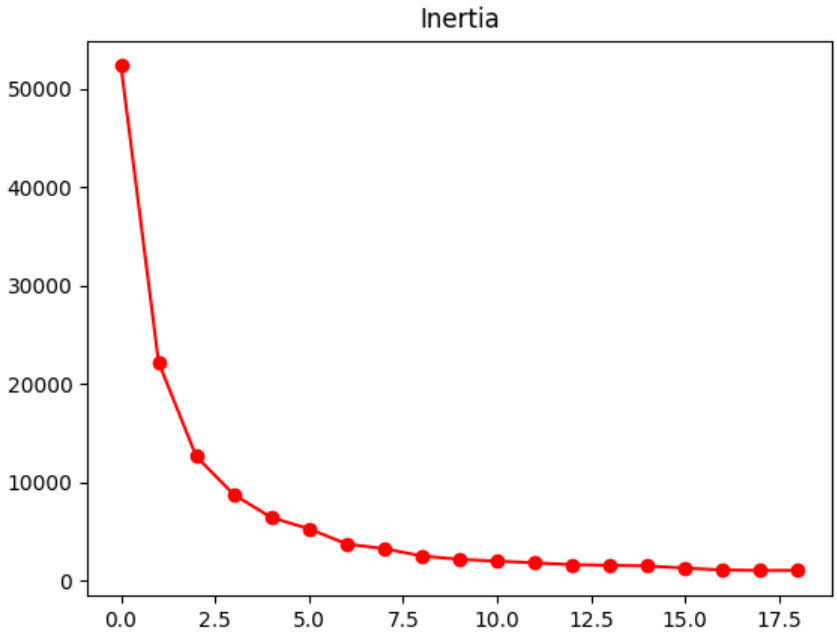


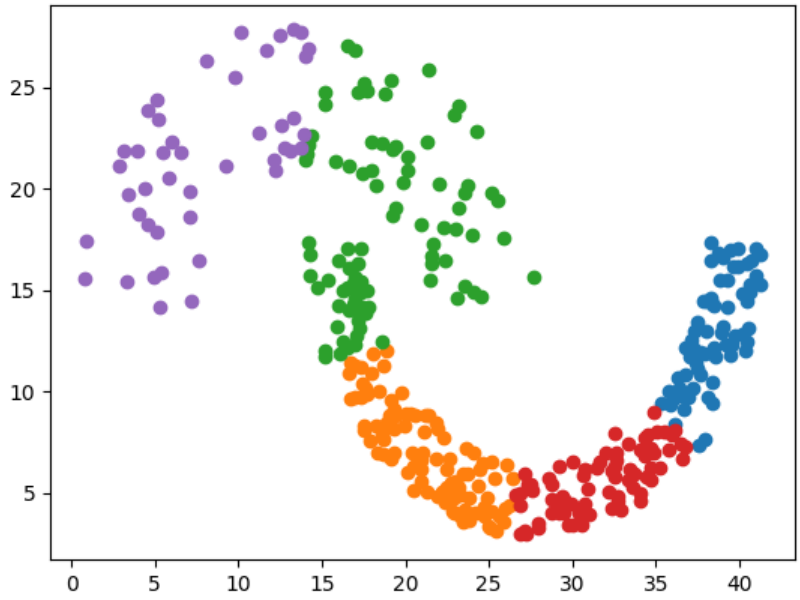


W tym przypadku klastry już naprawdę praktycznie idealnie wpasowują się w dane koła/plamki. Ma to odzwierciedlenie w purity, które wynosi dla Kmeans++ aż 0.99666, a dla kmeans też 0.926666.

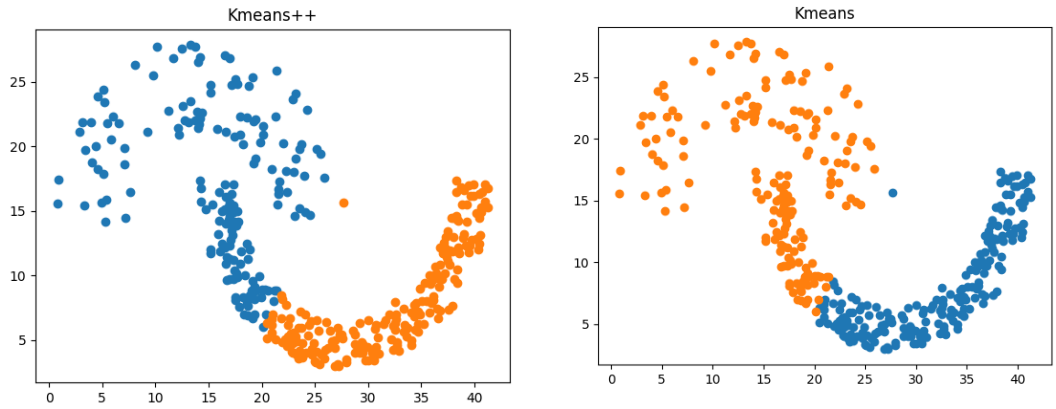
7. Dane_2D_7.txt

Tym razem mamy do czynienia z 2 półksiężycami. Stosując regułę łokcia można by wybrać k=5.





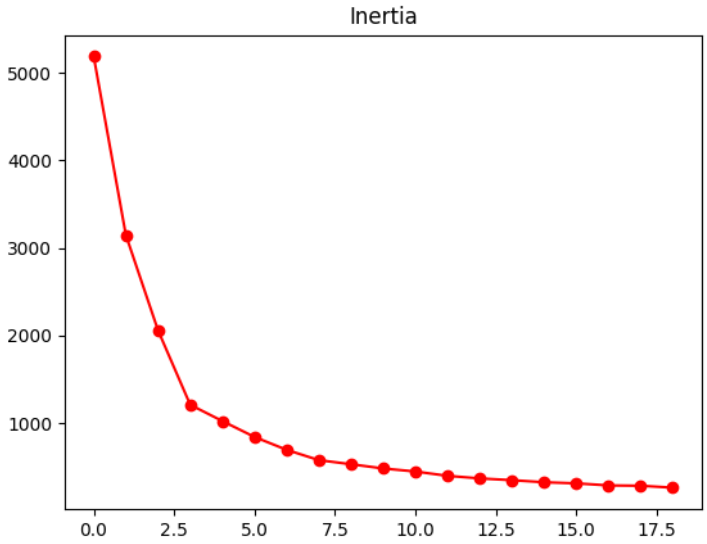
Jak można zauważyć, nasz algorytm z chęcią podzieliłby te półksiężyce na kilka mniejszych fragmentów, do których mógłby się łatwiej dopasować.
Wiemy jednak, że chcemy mieć tutaj k=2.



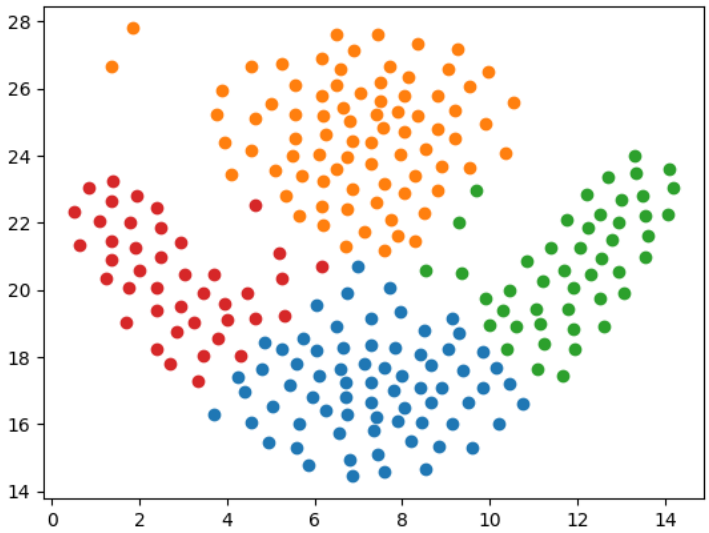
Jeżeli mamy do dyspozycji tylko 2 klasy to kmeans nie potrafi dobrze rozdzielić tych danych. Co ciekawe zarówno dla wersji kmeans jak i kmeans++ otrzymałem takie same najlepsze wyniki, które wynosiły 0.78552.

8. Dane_2D_8.txt

Teraz mamy do czynienia z kształtem przypominającym ludzika, w którym mamy 2 obserwacje odstające w lewym górnym rogu.

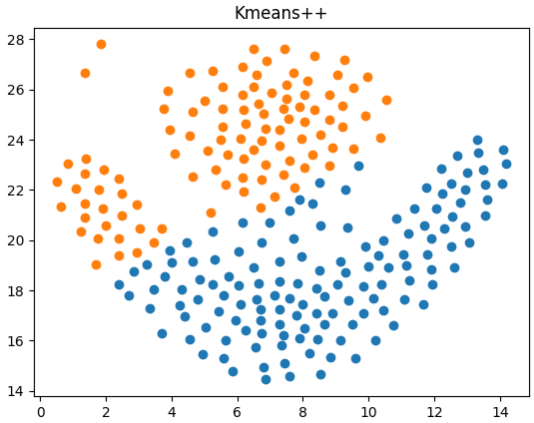


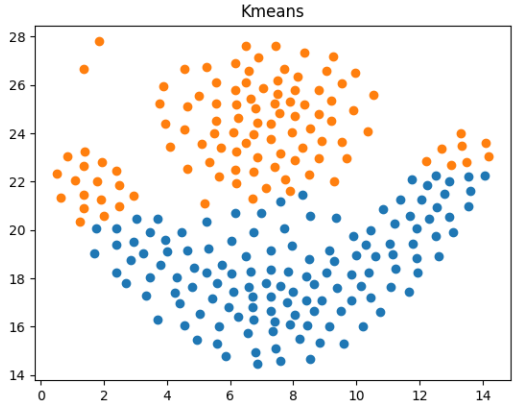
Z metody łokcia tym razem zdecydowałem się od razu na k=4.



W tym przypadku algorytm też chciał trochę podzielić ten kształt na kilka mniejszych, dla których bez większego problemu mógłby zdefiniować centroidy.

W tym przypadku mamy jednak oczekiwane k=2.

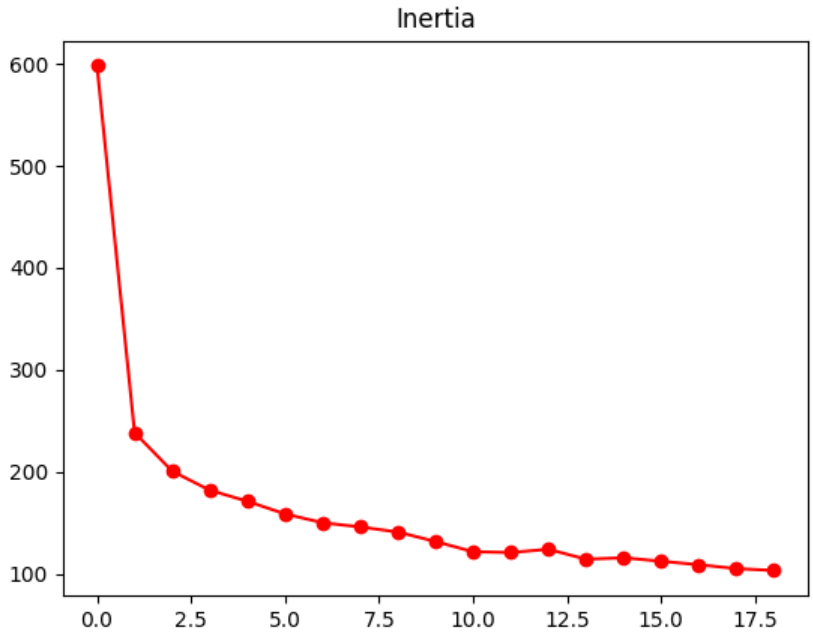




Dla k=2 jak można zaobserwować problemem są “dłonie”, których algorytmy nie są w stanie oddzielić od głowy. Mimo wszystko purity jest całkiem dobre i wynosi dla kmeans++ 0.858333, a dla kmeans jest lekko wyższe i jest równe 0.86666.

9. rp.data

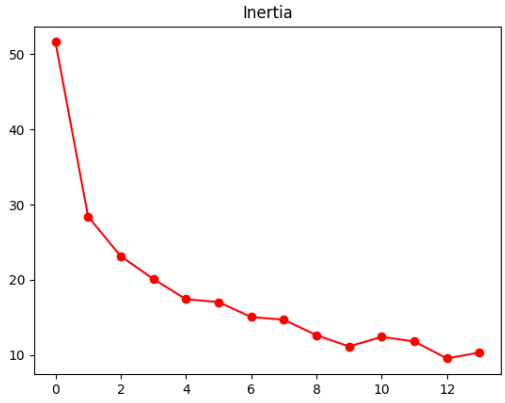
Sprawdziłem również w tym pliku jak wygląda wykres inertia po przeskalowaniu danych metodą min-max2

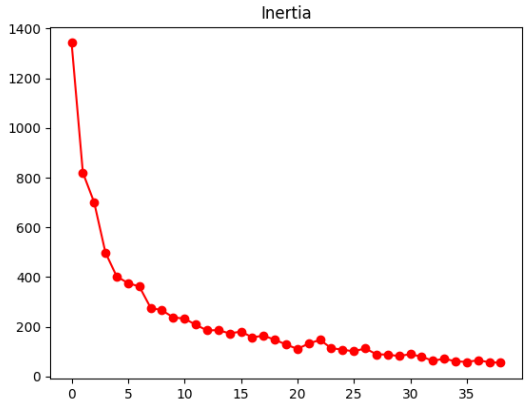


Co ciekawe wybór k=2 lub 3 byłby sensownym pomysłem. Co zgadza się z naszą interpretacją danych. Dla tych danych również otrzymaliśmy wysoki wynik purity dla k=2, który był równy aż 0.96193 dla kmeans i 0.96046 dla kmeans++. Te wyniki są dosyć zbliżone do wyników otrzymanych w miniprojekcie 3 dla naiwnego klasyfikatora bayesowskiego i regresji logistycznej.

10. Dane_9D.txt

Dla tych danych nie mamy dostarczonych opisów klas, a także ich wymiar jest większy niż 3, więc nie jesteśmy w stanie sobie ich zwizualizować w celu wybrania liczby klastrów. Możemy polegać tylko na wykresie błędu i regule łokcia.





Wybór k jest w tym przypadku ciężki, ponieważ w zależności od przebiegu algorytmu, wybranego skalowania, analizy liczbowej inertia różnie możemy interpretować jakie k wybrać z metody łokcia. Raz wygląda na to, że najlepiej wybrać k=4, a zaraz znowu najlepiej wygląda k=8, a k=6 też wygląda na sporej części wykresów dobrze.

Hierarchical clustering

Z uwagi na charakter algorytmu wyniki są dla 1 przebiegu algorytmu.

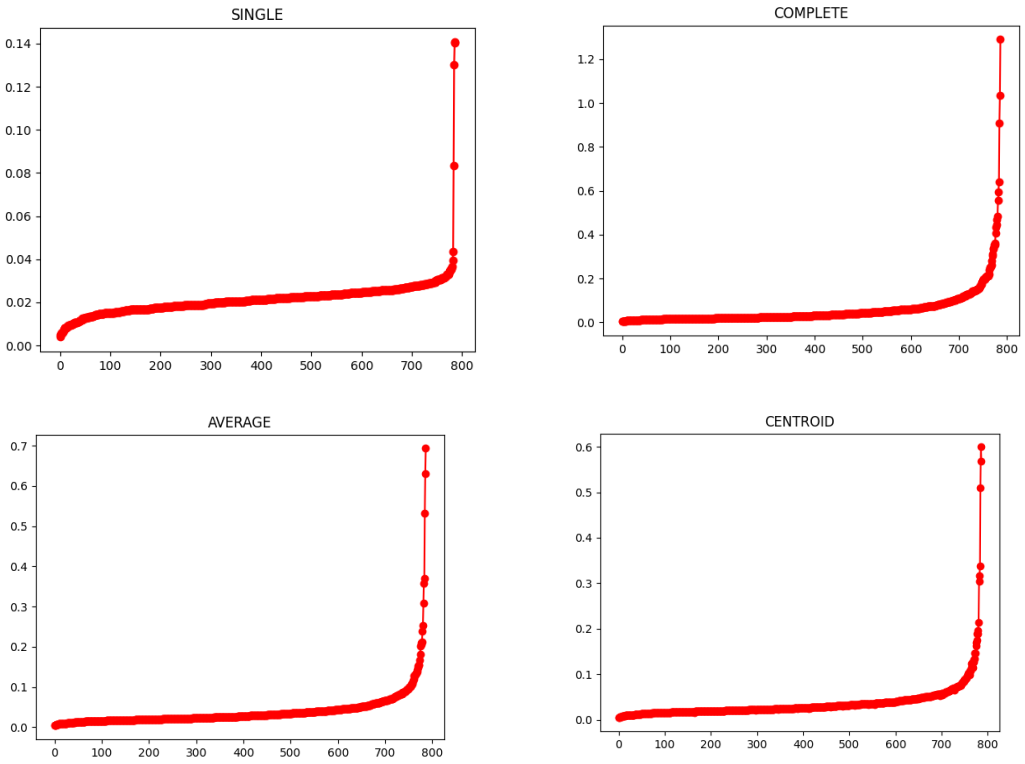
Podobnie jak w algorytmie K-means chcieliśmy wybrać liczbę klastrów dla algorytmu tak tutaj chcielibyśmy wiedzieć przy ilu klastrach skończyć algorytm. Dlatego też analizowałem wykres kolejno wybranych odległości w czasie algorytmu i na tej podstawie próbowałem zauważyć moment, w którym odległości się zwiększają.

Jak już wcześniej wspomniałem zaimplementowałem algorytm w 4 wersjach w zależności od wyboru metody łączenia klastrów.

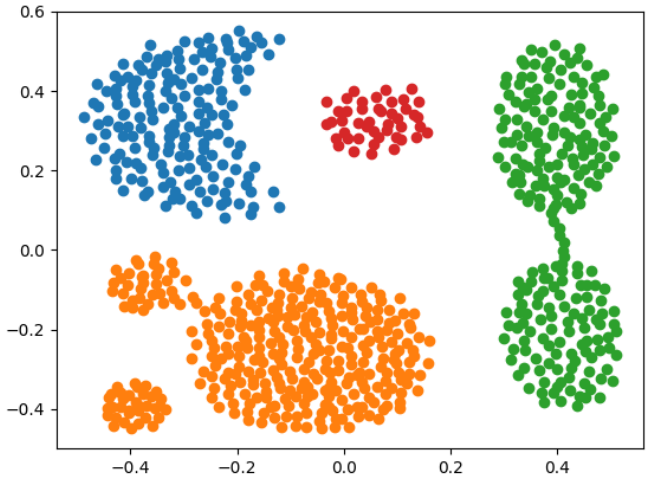
Przejdźmy teraz do otrzymanych wyników

1. Dane_2D_1.txt

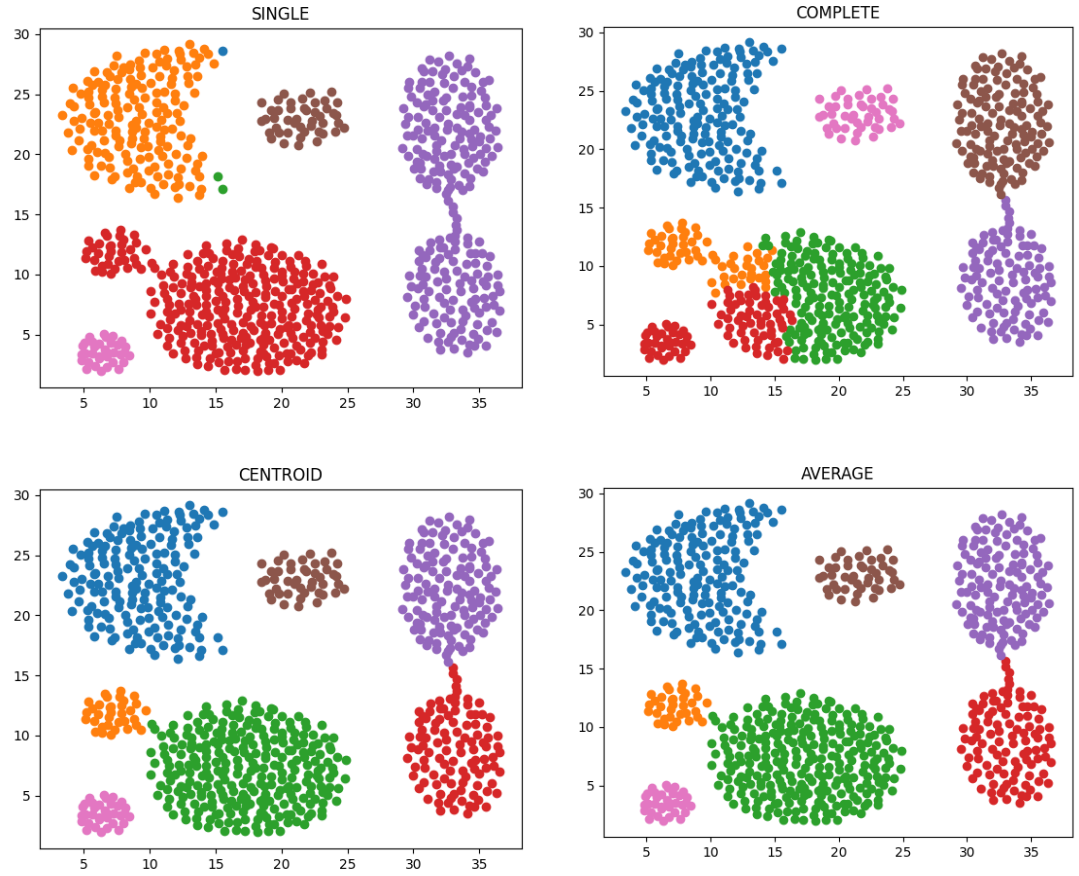
Analizując wykres odległości można zauważyć, że głównie celowalibyśmy w 6/7 klastrów dla metod complete, average, centroid. Tylko single mielibyśmy mniej – 4.



Wynik 6 jest zbliżony realnej liczby klas otrzymanej dla pliku 1. Natomiast wykres dla 4 klastrów single to



Dane zostały po prostu rozdzielone na 4 osobne segmenty.
Wyniki dla k=7 wyglądały natomiast następująco.



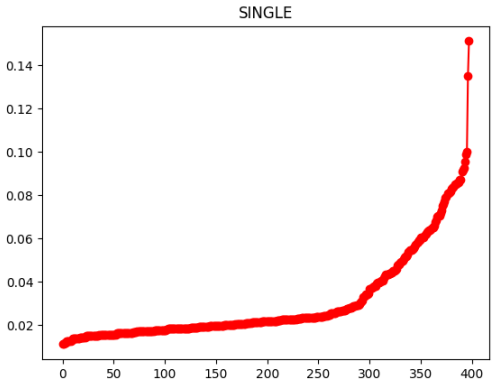
Jak można zauważyć z wykresów single zasadniczo rozdzielił 5 klas. Udało mu się oddzielić mniejszy lewy dolny fragment od całego lewego dolnego co wcześniej nie udało się kmeans, natomiast co ciekawe nie oddzielił fragmentu po prawej. Average i centroid rozdzieliły nasze dane praktycznie w 100%, a

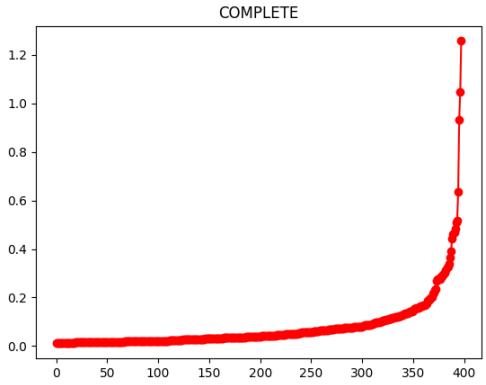
complete starał się, ale podobnie jak kmeans poległ na lewym dolnym fragmencie.

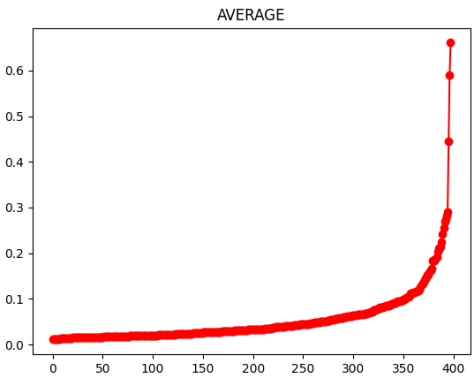
Single uzyskał purity 0.82741, complete 0.9175, a centroid i average w obu przypadkach fenomenalne 0.99619

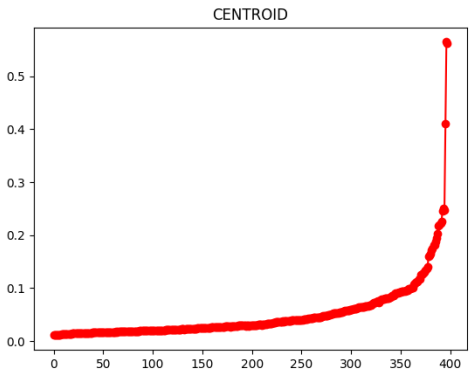
2. Dane_2D_2.txt

W tym przypadku można szacować k dla single 3, complete 5, average 4, centroid 5.

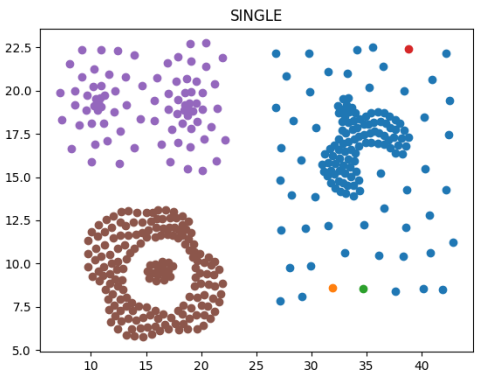


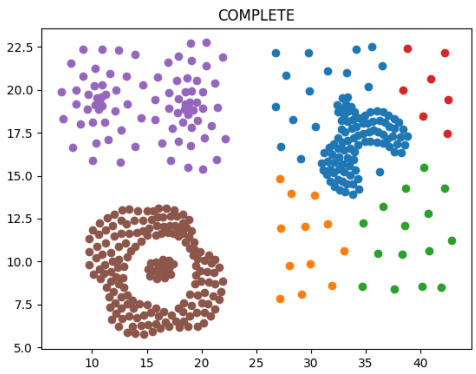


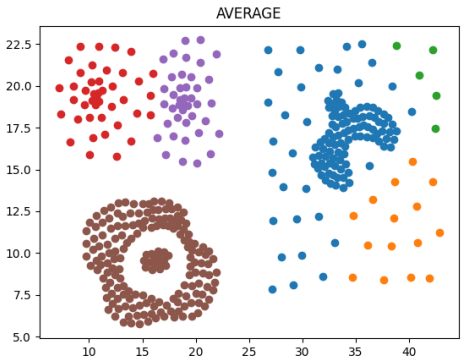


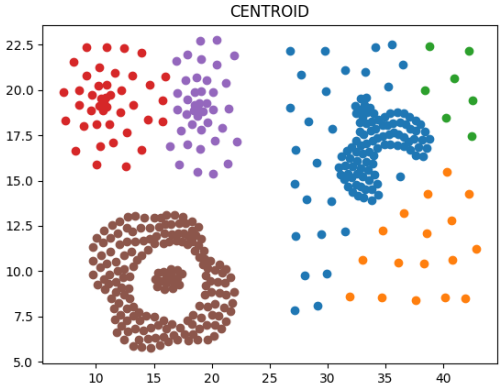


Takie wybory k mają swoje odzwierciedlenie dla wykresów dla liczby klas równej 6.







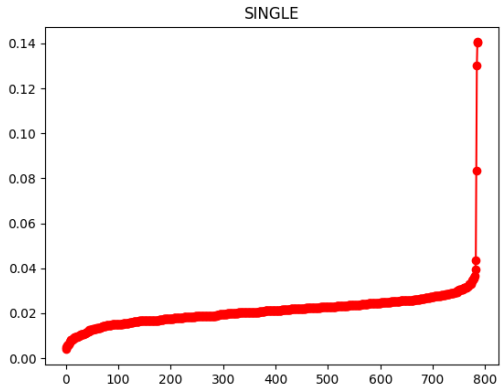


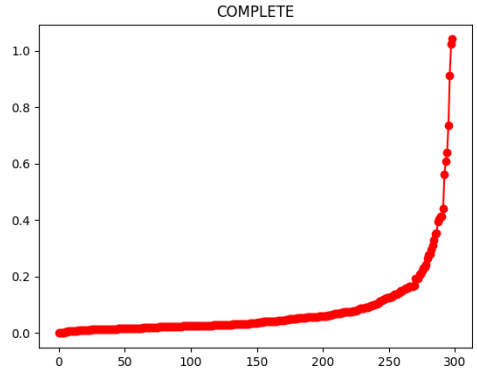
Single właśnie udało się wydzielić tylko 3 klastry, complete w sumie to 4, ale zrobił to w dość specyficzny sposób, a centroid i average są zbliżone natomiast do wykresów z kmeans.

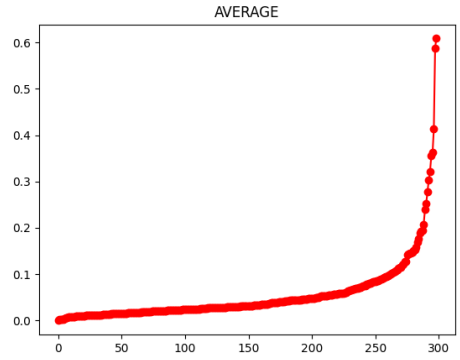
Single uzyskał tutaj 0.746867, complete 0.824561, average 0.8746867, a centroid 0.884711.

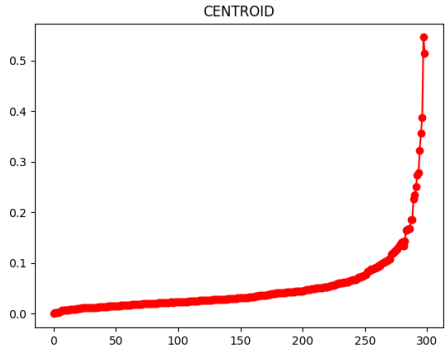
3. Dane_2D_3.txt

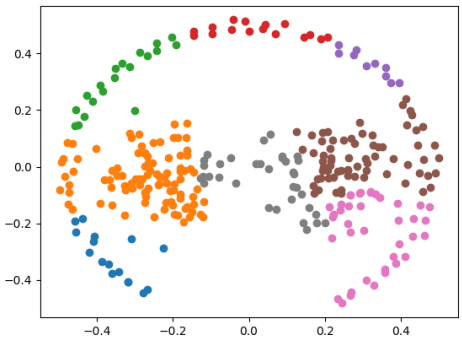
Podobnie jak kmeans nasze metody chciałyby podzielić pierścień na kilka mniejszych fragmentów - single na 4, complete na 8, average na co najmniej 4, centroid na co najmniej 6

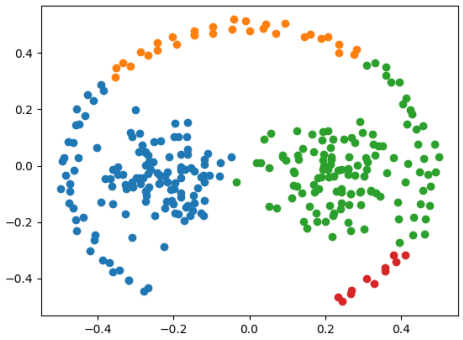


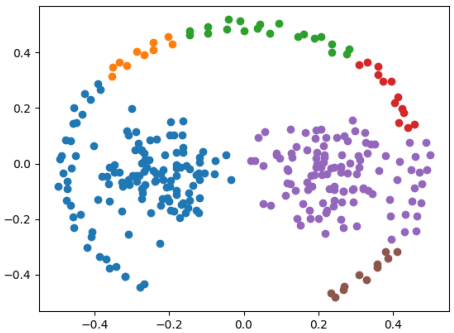






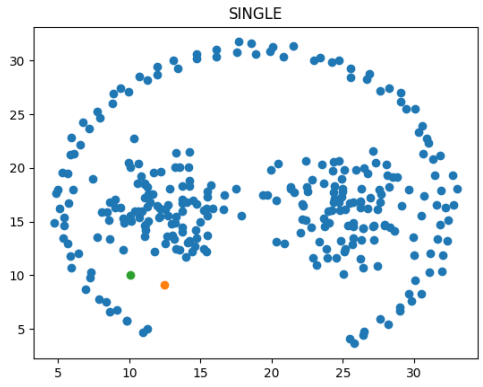


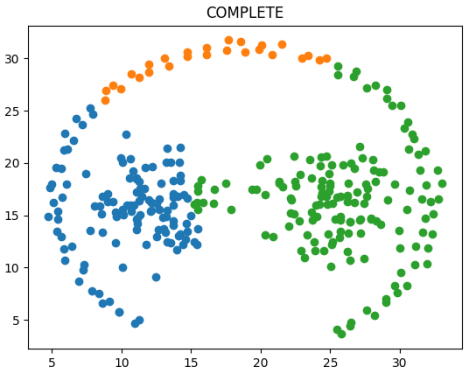


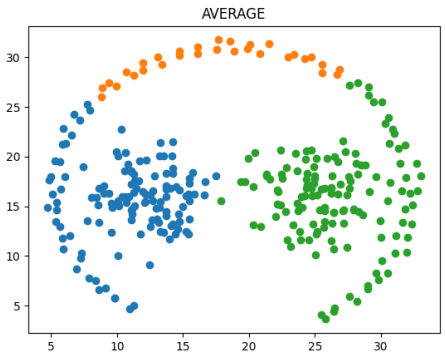


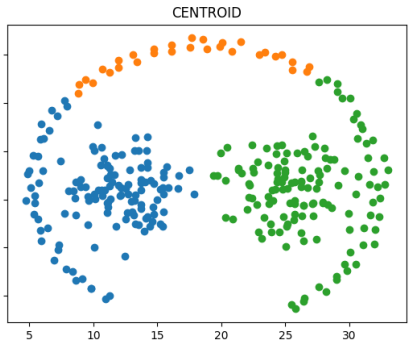
Od lewej kolejno complete k=8, average k=4, centroid k=6

Wyniki dla k=3 wyglądają w taki sposób







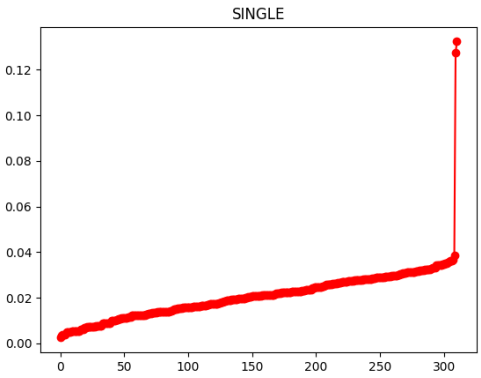


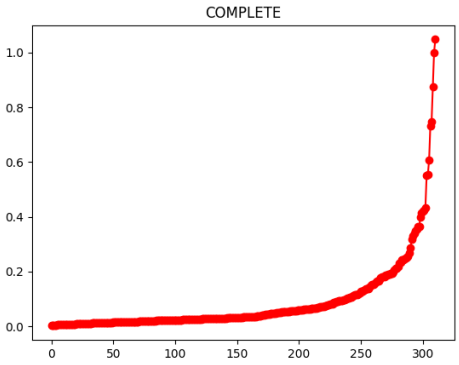
Single odstaje od reszty, w ogóle nie poradził sobie z klasteryzacją i tak właściwie zwrócił 1 klaster. Pozostałe metody dawały wyniki na podobnym poziomie (podobnym także do kmeans).

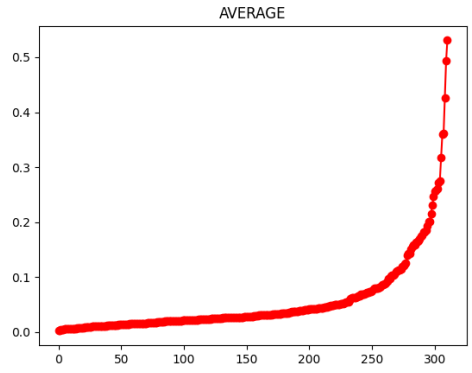
Wyniki jaki osiągnęły to single - 0.3733333, complete - 0.68333, average - 0.73, centroid - 0.73333

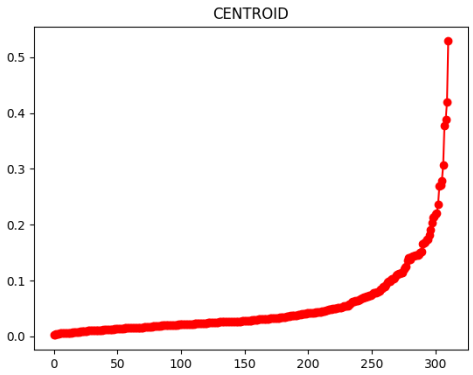
4. Dane_2D_4.txt

Wnioski dla tych danych będą bardzo zbliżone do kmeans – wysokie k w celu rozbicia spirali na jak najwięcej części, z 1 dość sporym wyjątkiem.



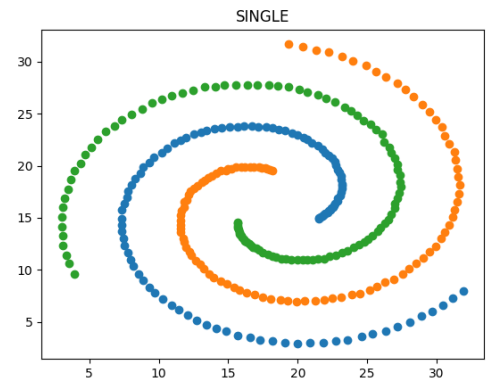


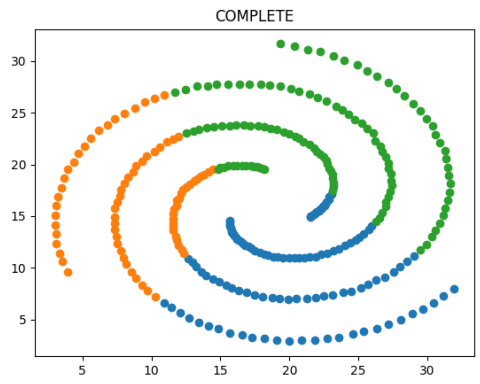


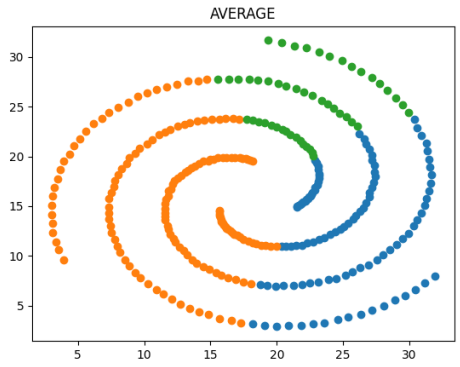


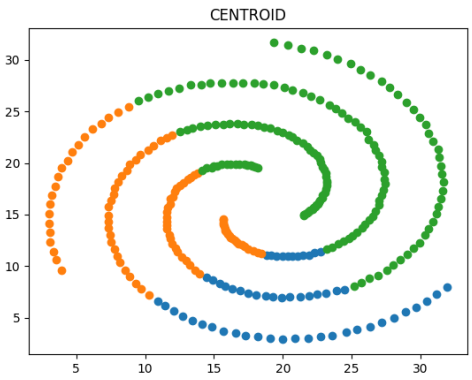
Z wykresów można odczytać, że complete chce 10, average 6, centroid 8, a co ciekawe single 3.

Dla k = 3 wykresy są następujące







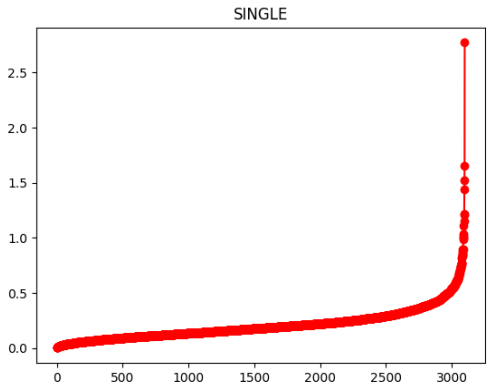


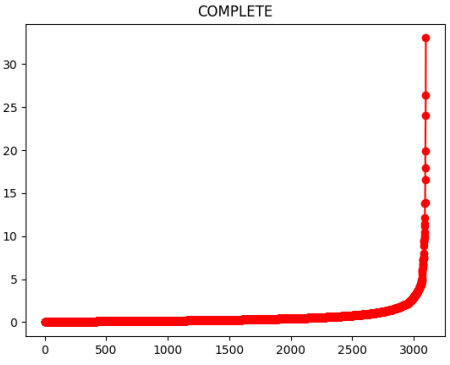
Single uzyskał dla tego przypadku 100% skuteczności - udało mu się idealnie wyznaczyć klastry, pozostałe metody jak już wspomniałem zwróciły wyniki podobne do kmeans.

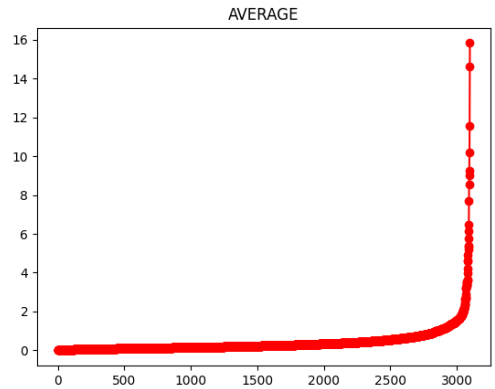
Purity dla single – 1.0, complete - 0.3782, average - 0.368589, centroid - 0.4038

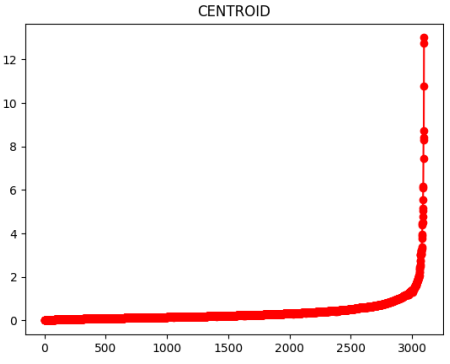
5. Dane_2D_5.txt

Generalnie metody, podobnie jak kmeans dobrze sobie radziły z tymi danymi. Niestety tym razem single poległ.



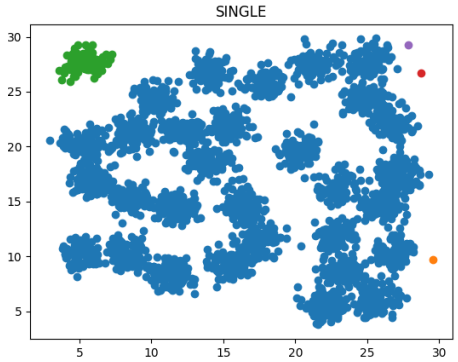


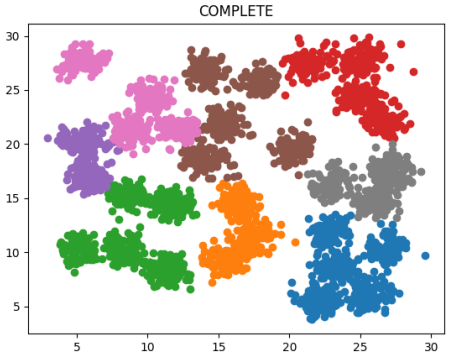


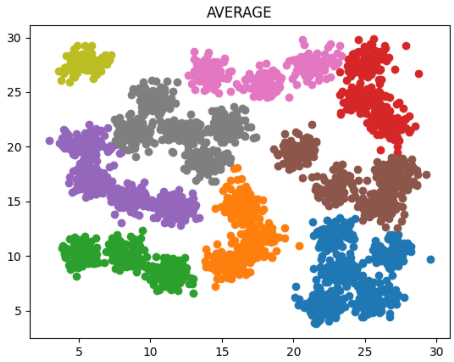


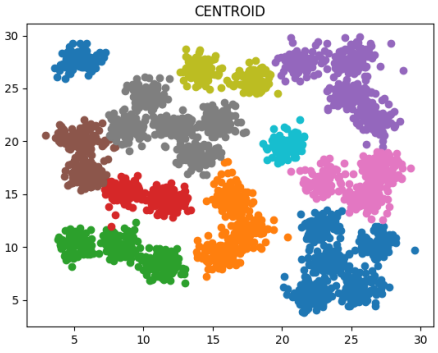
Wyniki, które moglibyśmy odczytać z tych wykresów są znacznie mniejsze od liczby klas 31 – single ma 5, complete 8, average 9, centroid 11.

Wykresy odpowiednich metod dla danych powyżej k prezentują się następująco

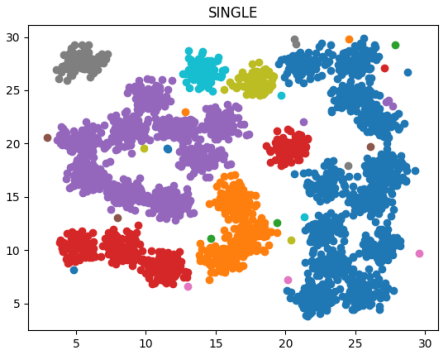


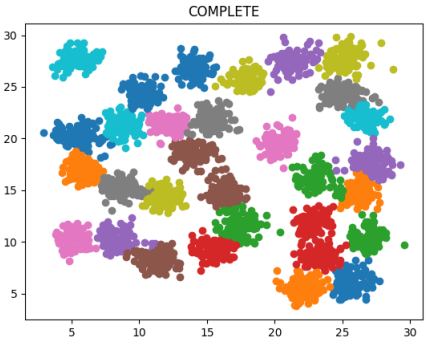


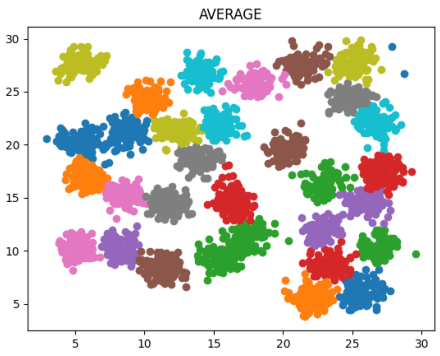


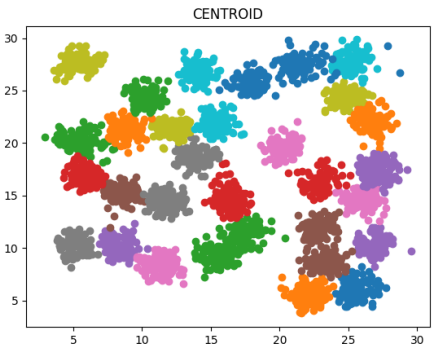


Natomiast dla k=31 wygląda to tak





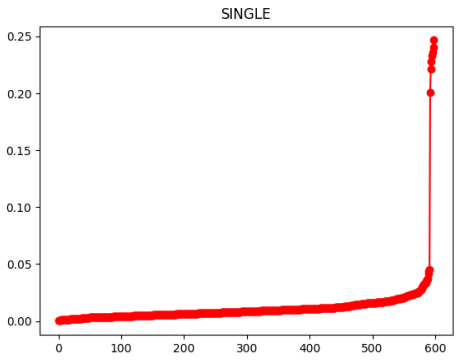


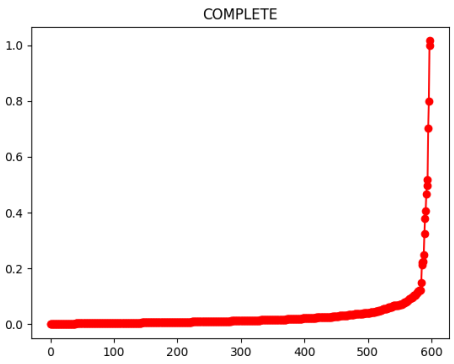


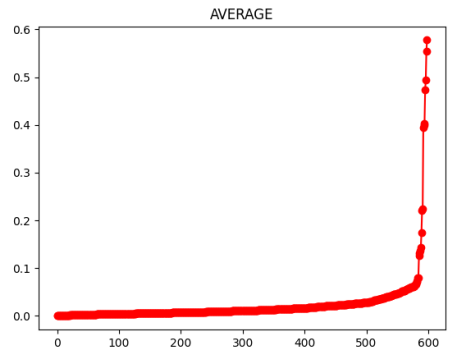
Najlepsze purity uzyskał complete i wyniosło ono 0.961935, następnie był centroid z wynikiem 0.9387, kolejny average z 0.9380 i ostatni single z zaledwie 0.2651.

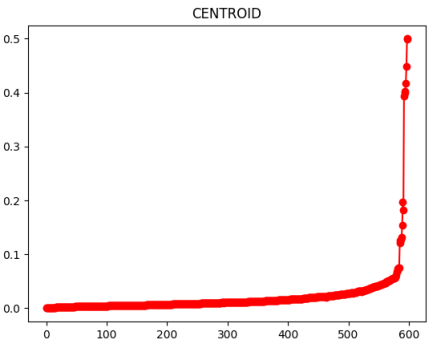
6. Dane_2D_6.txt

Tutaj metody też porodziły sobie super z wyłączeniem single. Co ciekawe przyglądając się wykresowi można by stwierdzić, że k=15 jest dobre dla naszych danych w przypadku complete, average i centroidu. Single natomiast ma k=8.

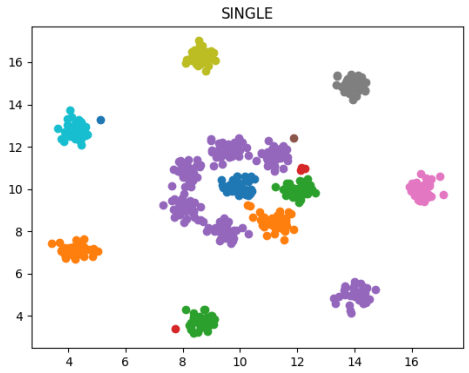


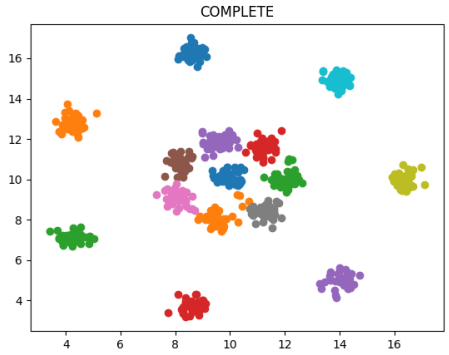


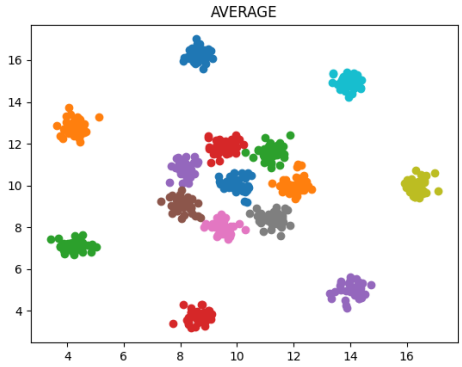


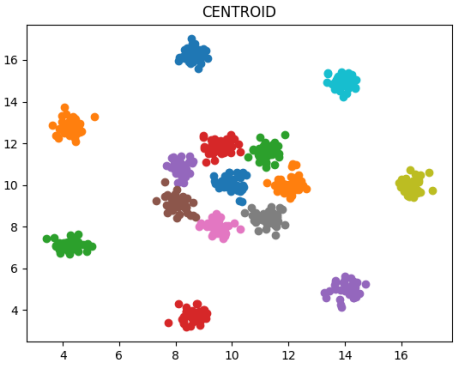


Dla k=15 wygląda to następująco:





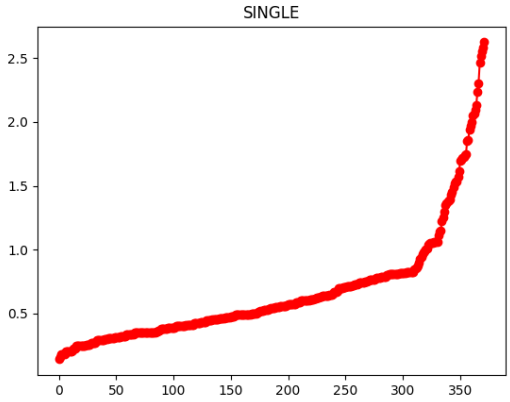


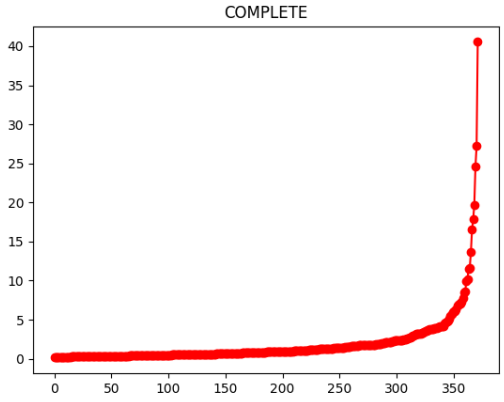


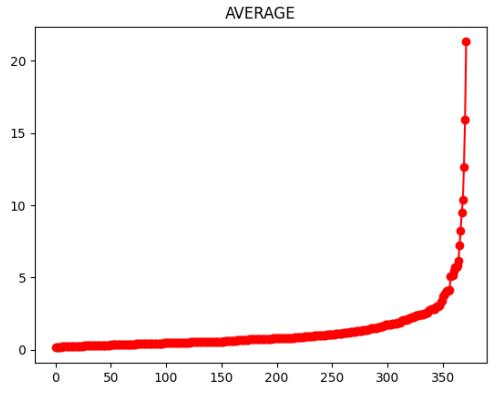
Single uzyskał tu wynik 0.73166, a pozostałe 3 metody dały świetny wynik – complete 0.99, a centroid i average jeszcze więcej - 0.995

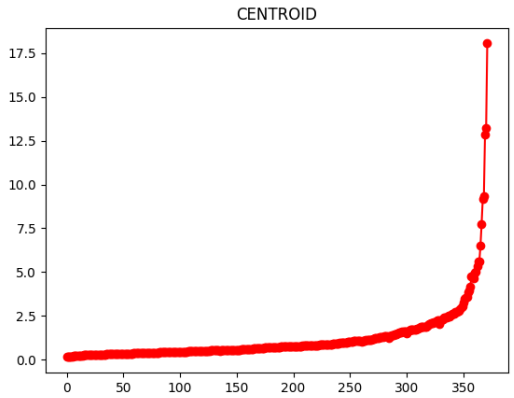
7. Dane_2D_7.txt

Podobnie jak kmeans, nasze metody chciałyby móc podzielić półksiężyce na więcej części - single na 7, a pozostałe na 8

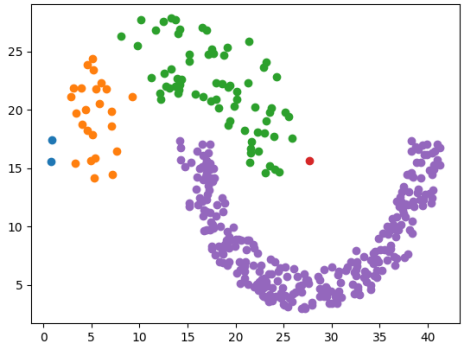


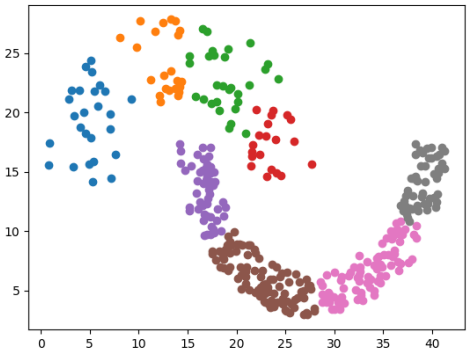


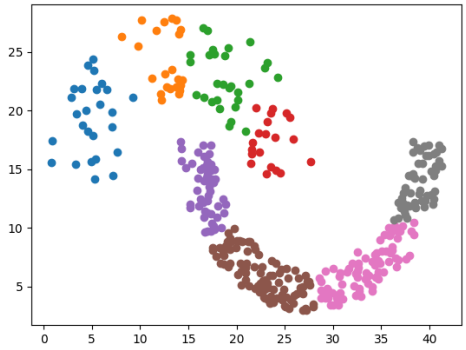


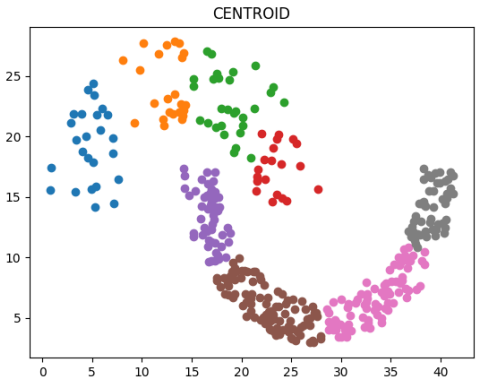


Wykresy wtedy wyglądają tak (w kolejności jak zwykle):

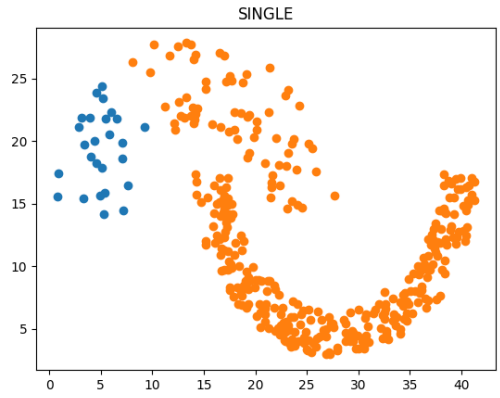


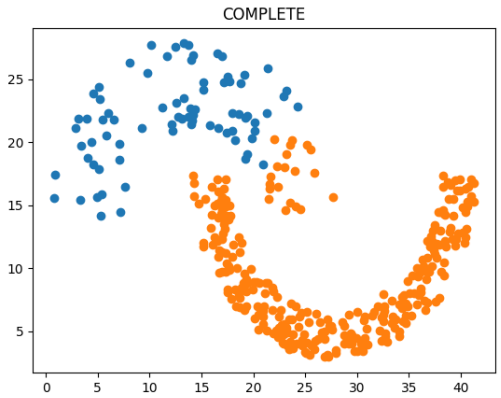


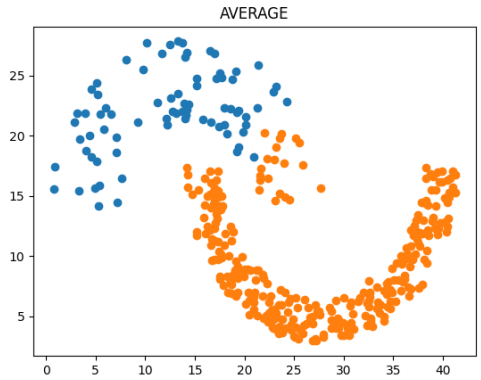


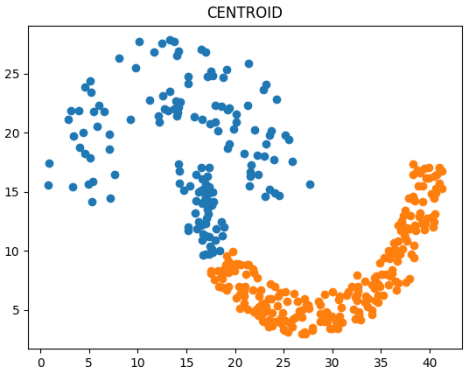


Gdy jednak zastosujemy k=2 to już nie tak dobrze (nie udaje się rozdzielić półksiężyców w 100%)





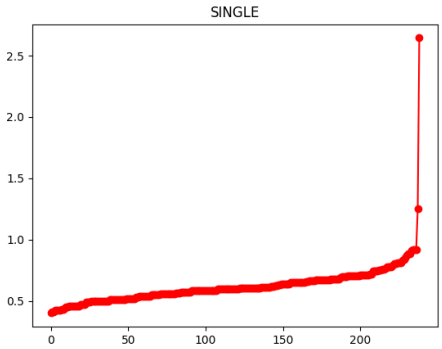


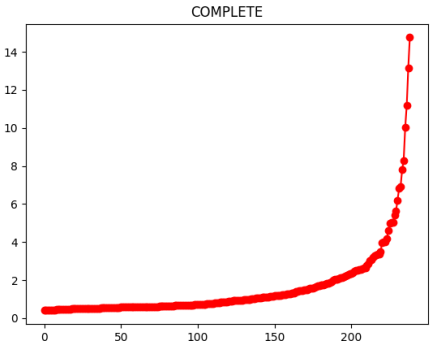


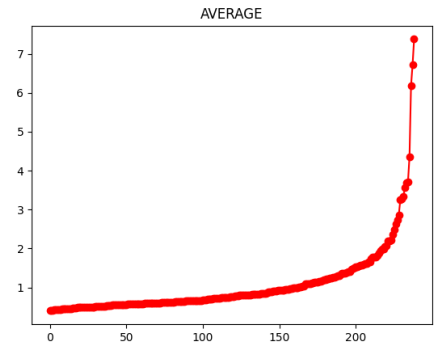
Otrzymane wyniki są następujące - single – 0.80965, complete i average po równo - 0.94638, a centroid 0.8605

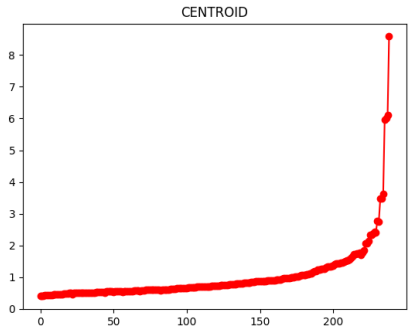
8. Dane_2D_8.txt

Metody w tym przypadku, tak jak w kmeans chciałyby podzielić ludzika na kilka części.

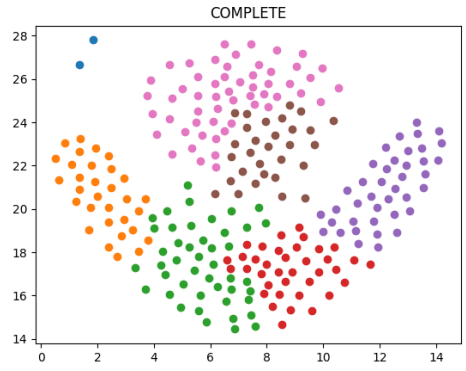


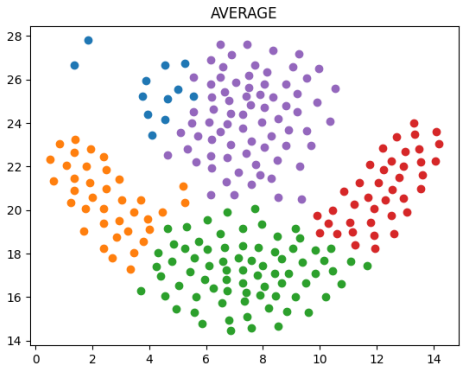


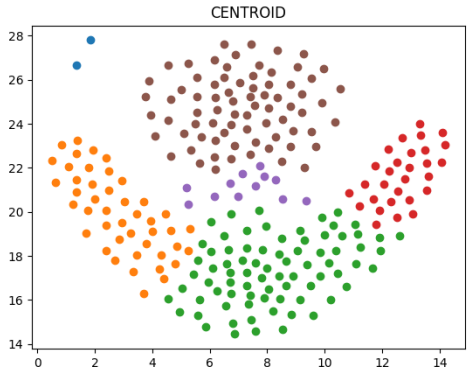




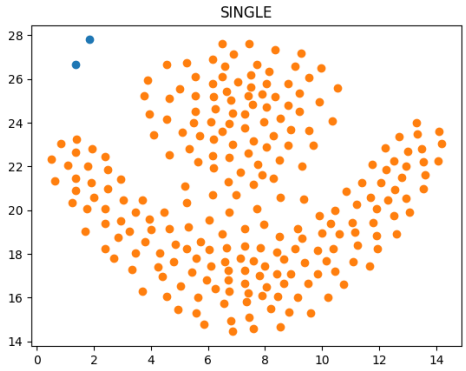
Single tylko na 3, ale complete na 7, average na 5, centroid na 6.

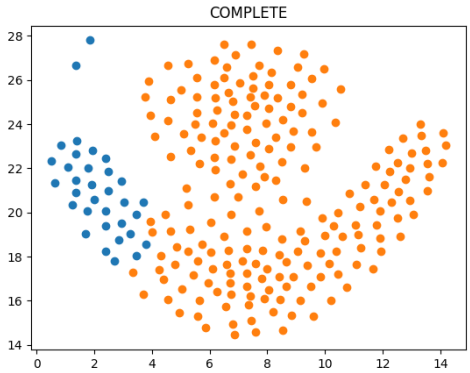


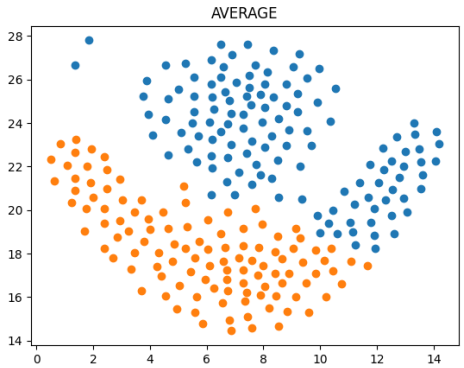


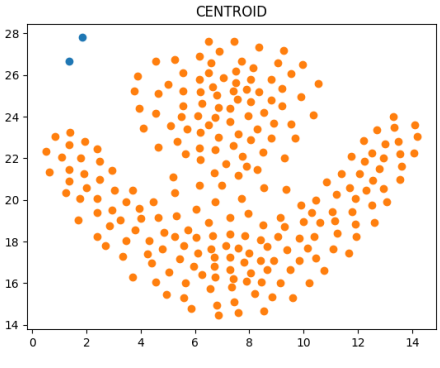


Dla k = 2 wykresy te oprócz average się popsują, do takiego stopnia, że dla centroidu 1 klastrem będą 2 kropki w lewym górnym, a 2 klastrem pozostała reszta.





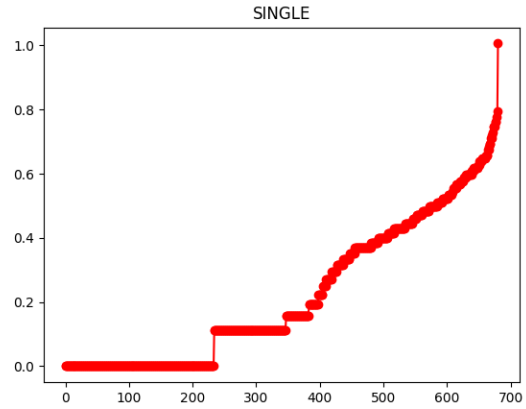


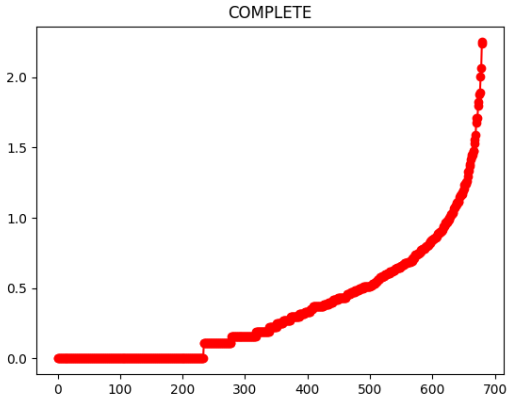


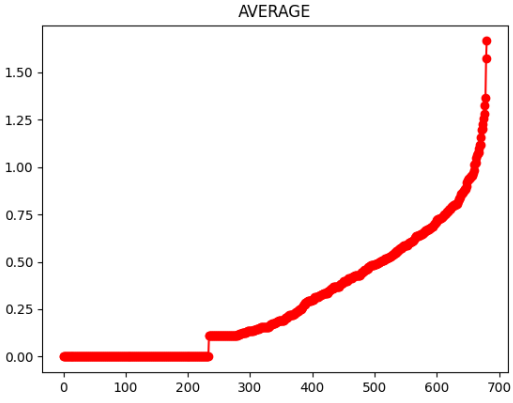
Wyniki prezentują się następująco - single i centroid to 0.645833, complete 0.6375, a average 0.833333

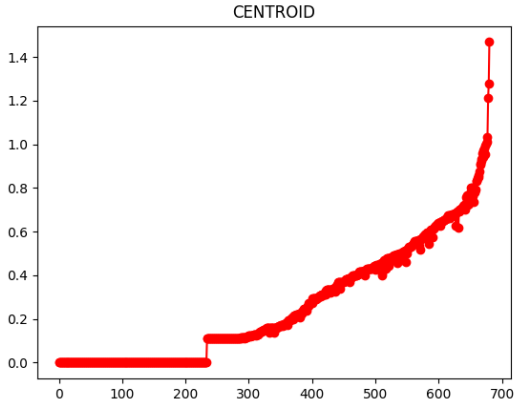
9. rp.data

Tutaj również popatrzyłem co pokazałyby nasze wykresy.







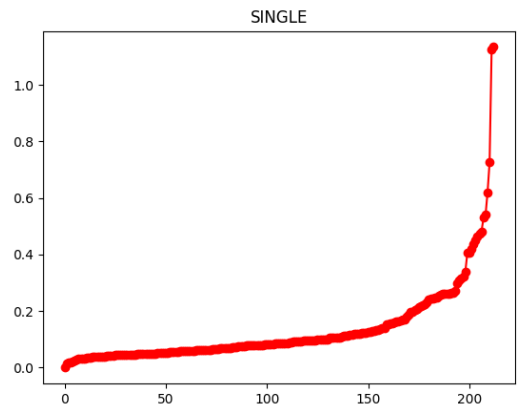


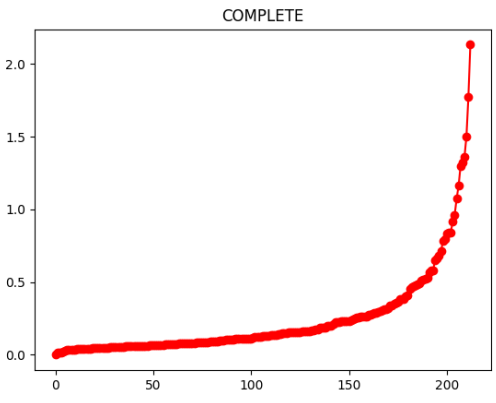
Otrzymane k dałyby bardzo zbliżone wyniki do właściwego tzn. Single 2, average 3, centroid 4, complete można by różnie interpretować, ale raczej powyżej 4. Co ciekawe na wykresach można zauważyć, że w tym pliku było sporo obserwacji mających takie same współrzędne.

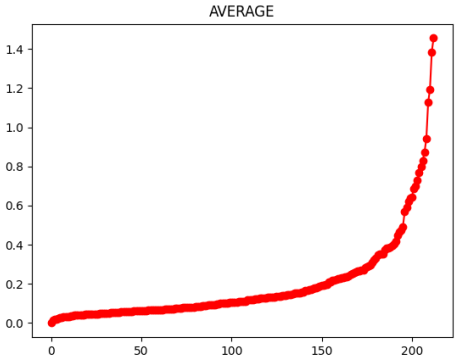
Wyniki uzyskane dla single to 0.65153, complete 0.85065, average 0.94289, a natomiast centroid aż 0.97071.

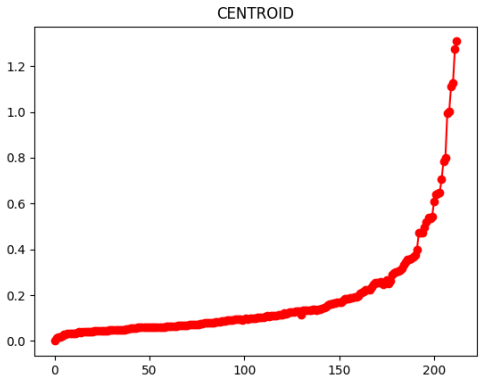
10 Dane_9D.txt

Dla naszych danych bez etykiet też możemy przeanalizować liczbę potencjalnych klastrów.









Dla single zdecydowalibyśmy się pewnie na ok. 7, complete 8, average 6, centroidu też w okolicy 6/8.

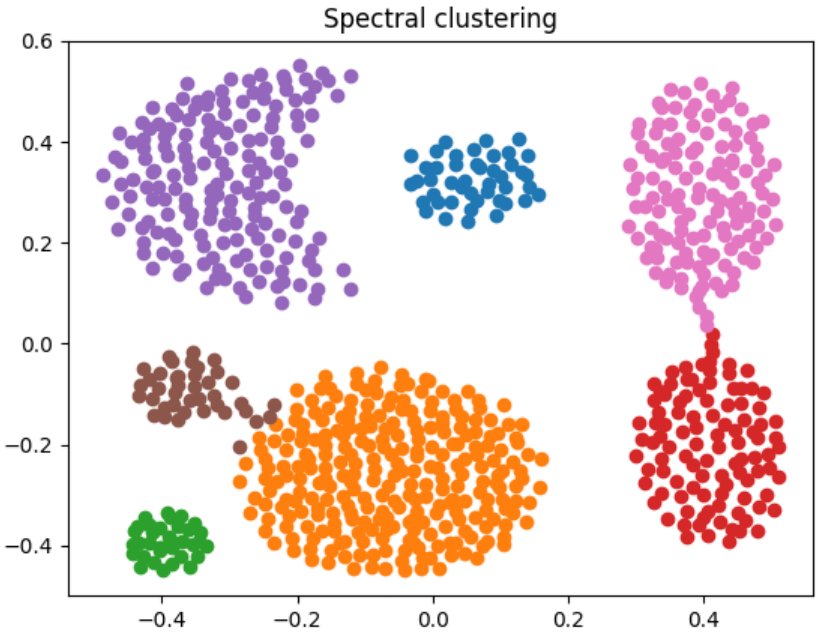
Spectral clustering

Dla klasteryzacji specjalnej gdy liczba klas była większa od 2 to wykonywałem kilka przebiegów algorytmu kmeans++ albo próbowałem wykorzystać grupowanie hierarchiczne.

Dla tego algorytmu sprawdzałem tylko takie k, które były równe liczbie w zestawie danych.

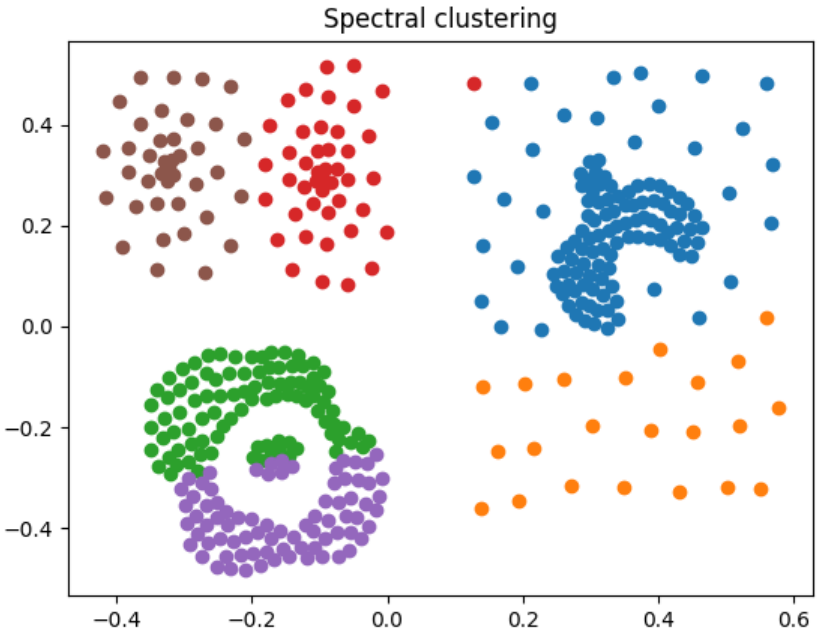
1. Dane_2D_1.txt

Dla epsilona 0.1 osiągnąłem tu wykorzystując kmeans++ purity 0.991116



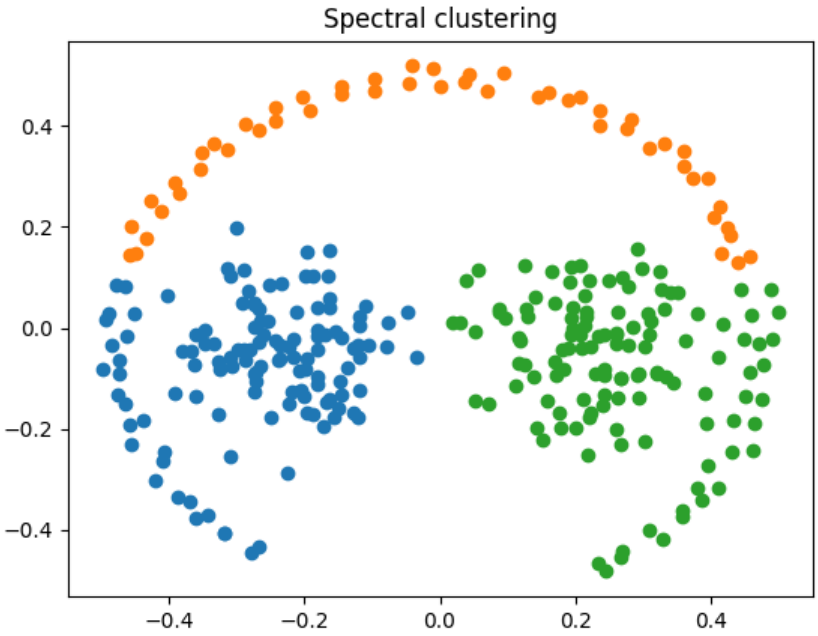
2. Dane_2D_2.txt

Dla epsilona 0.2 osiągnąłem tutaj wykorzystując kmeans++ purity 0.887218, co jest najwyższym wynikiem, ale nadal poniżej 0.9. W żadnym algorytmie nie udało mi się rozdzielić sensownie lewego dolnego fragmentu na 2, ani prawego na 2 klastry.



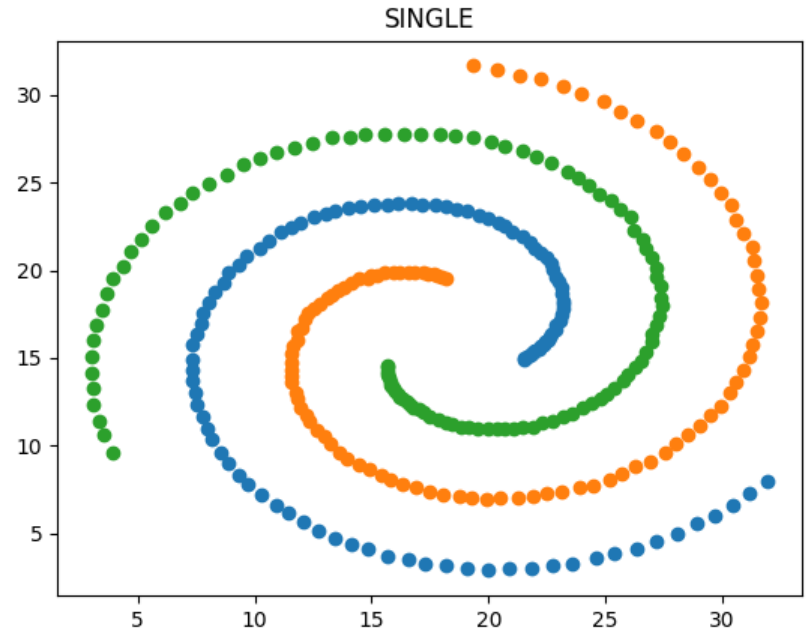
3. Dane_2D_3.txt

Dla epsilona 0.1 osiągnąłem tutaj wykorzystując kmeans++ purity 0.8033333, co jest też najwyższym wynikiem spośród wszystkich innych, ale ledwie co powyżej 0.8. Problemem tutaj może być taki fakt, że np. Po prawej stronie odległości w pierścieniu są większe niż odległości punktów z prawej kulki.



4. Dane_2D_4.txt

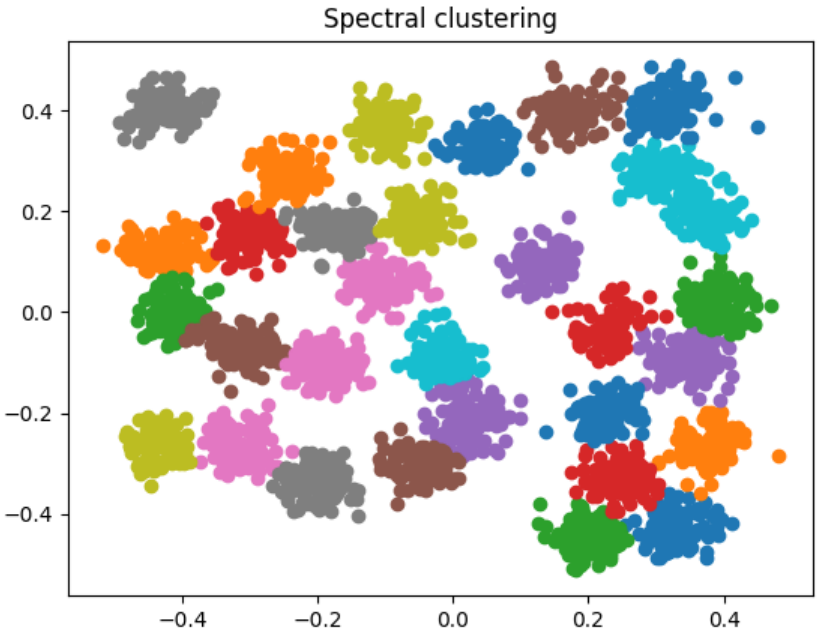
Dla epsilona 0.1 osiągnąłem tutaj wykorzystując kmeans++ purity 1.0



(Wykres wzięty jak widać z single, ale afekt jest taki sam)

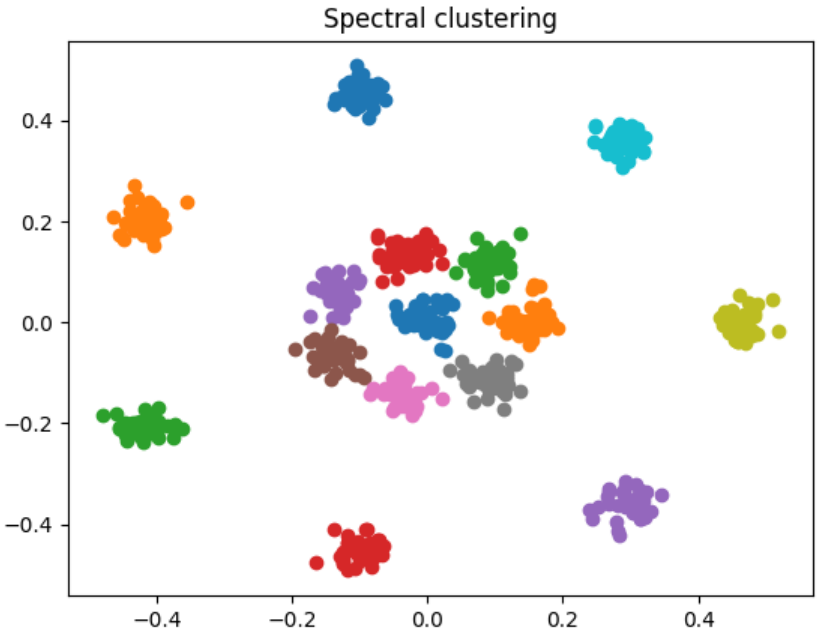
5. Dane_2D_5.txt

Dla epsilona 0.1 i centroidu uzyskałem tutaj purity 0.962258



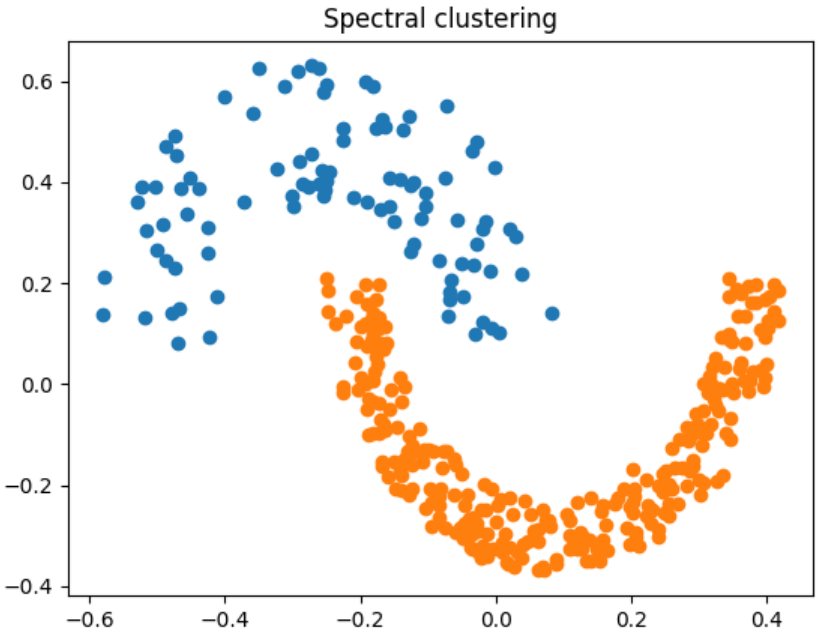
6. Dane_2D_6.txt

Dla epsilona 0.06 i np. average uzyskałem tutaj purity 0.9966666



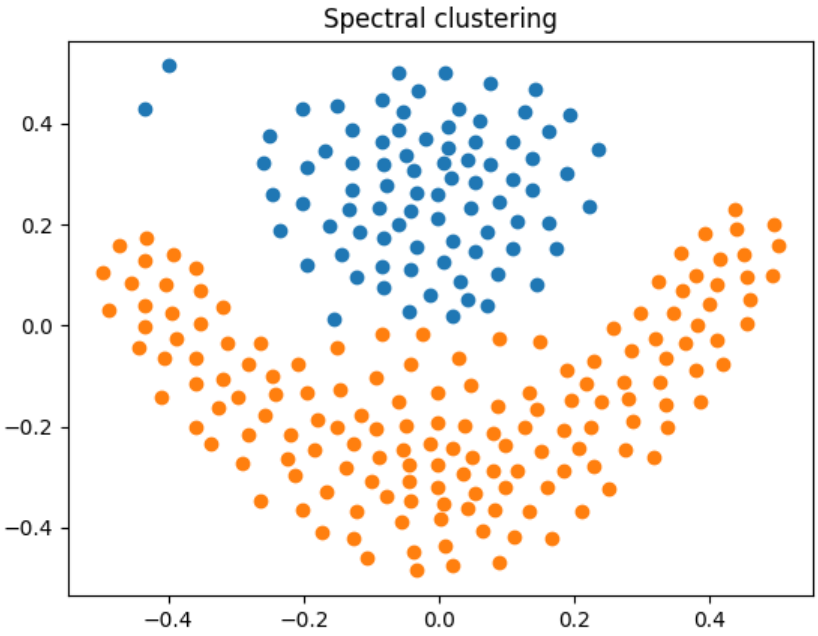
7. Dane_2D_7.txt

Stosując tutaj klasteryzację spektralną udało się dla epsilona 0.1 uzyskać purity 1.0



8. Dane_2D_8.txt

Dla epsilona 0.26 uzyskałem tutaj purity 0.979166, co jest zdecydowanie najlepszym wynikiem.



9. rp.data

Dla epsilona 0.32 uzyskałem tu purity 0.9443, co nie jest akurat najlepszym wynikiem. Może to wynika z faktu, że jest sporo obserwacji o takich samych współrzędnych.

Podsumowując wykorzystując klasteryzację spektralną jesteśmy w stanie dobrze sklasteryzować nawet przypadki, które były wcześniej ciężkie do sklasteryzowania. W szczególnych przypadkach może on jednak nie być najlepszy. Pozostałe algorytmy bardzo dobrze sprawowały się gdy dane były skupione w mniej więcej kulistych klastrach. Gdy pojawiały się kształty spirali, pierścieni, półksiężyców to mogły mieć z takimi kształtami problem.