Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

Predmet / Subject

– Umelá inteligencia / Artificial intelligence –

- Dokumentácia / Documentation -

Zadanie č.3

Ak. Rok / Academic term: 2022/2023, zimný semester

Cvičiaci / Instructors:
Ing. Martin Komák, PhD.

Študent / Student:
Adam Grík

Bratislava, 2022.

Obsah / Content:

1 Zadanie

Máme 2D priestor, ktorý ma rozmery X a Y, v intervaloch od -5000 do +5000. Tento 2D priestor vyplníme 20 bodmi, pričom každý bod má náhodne zvolenú polohu pomocou súradníc X a Y. Každý bod má unikátne súradnice.

Po vygenerovaní 20 náhodných bodov vygenerujeme ďalších 40 000 bodov, avšak tieto body nebudú generované úplne náhodne, ale spôsobom takým, že ako prvé vyberieme náhodne jeden bod z doteraz všetkých vytvorených bodov, a následne k súradniciam X a Y, pridám náhodne vygenerovaný offset od -100 a +100, čo budú súradnice nášho nového bodu, a takto postupujem až kým ich nebude 40 000.

Mojou úloho v zadaní č. 3 bolo naprogramovať zhlukovač pre 2D priestor, ktorý analyzuje 2D priestor so všetkými jeho bodmi a rozdelí tento priestor na k zhlukov (klastrov).
Za úlohu som mal implementovať rôzne verzie zhlukovačov, konkrétne týmito algoritmami:

- K-means, kde stred je centroid
- K-means, kde stred je medoid
- Divízne zhlukovanie, kde stred je centroid

Ďalšou úlohou bolo tak isto vyhodotiť úspešnosť zhlukovača. Za úspešný zhlukovač môžeme považovať taký, v ktorom žiaden klaster nemá priemernú vzdialenosť bodov od stredu viac ako 500.

Samozrejme pre každé riešenie je potrebná aj 2D vizualizácia.

Dané algoritmy budem implementovať v programovacom jazyku Python.

2 Algoritmy

2.1 K-means (stred - centroid)

Ak už máme vytvorené pole našich 40 000 bodov tak pri algoritme k-means postupujeme následovne:

- Zvolíme si k počet centroidov na náhodných súradniciach v celom našom 2D priestore – v mojom prípade som testoval počty centoridov 10, 15 a 20
- Každý zo 40 000 bodov priradíme k tomu centroidu, ku ktorému má najbližšiu vzdialenosť – v mojom prípade som používal *Euclidian distance,* pričom nám vznikne k počet klastrov
- Následne pre každý klaster vypočítame novú pozíciu centroidu, a to tak že si vypočítame priemer zo všetkých X súradníc, a priemer zo všetkých Y súradníc, následne nový centroid vznikne na novýc X a Y súradniciach
- Opäť opakujeme postup od bodu č. 2, kde každý bod priradíme k najbližšiemu centroidu

Tento postup opakujeme až kým sa nám neprestanú meniť resp. sa neustália pozície jednotlivých centroidov.

2.2 K-means (stred - medoid)

Ak už máme vytvorené pole našich 40 000 bodov tak pri algoritme k-means postupujeme následovne:

- Zvolíme si k počet medoidov, a to tak že vyberieme k počet náhodných bodov z našich 40 000 vytvorených
- Každý zo 40 000 bodov priradíme k tomu medoidu, ku ktorému má najbližšiu vzdialenosť – v mojom prípade som používal *Euclidian distance,* pričom nám vznikne k počet klastrov
- Následne pre každý klaster vypočítame novú pozíciu medoidu, a to tak, že medoid je bod ktorý má najmenšiu priemernu vzdialenosť ku všetkým bodom v danom klastri
- Opäť opakujeme postup od bodu č. 2, kde každý bod priradíme k najbližšiemu medoidu

Tento postup opakujeme až kým sa nám neprestanú meniť resp. sa neustália pozície jednotlivých medoidov.

2.3 Divízne zhlukovanie (stred - centroid)

Ak už máme vytvorené pole našich 40 000 bodov tak pri algoritme divízneho zhlukovania postupujeme následovne:

- Ako prvé si náš 2D priestor rozdelíme na polku, čiže nám vzniknú prvé dva klastre
- Ako ďalšie si zvolíme ten klaster ktorý ma väčšiu priemernú vzdialenosť bodov od jeho stredu
- Ako ďalšie si v tomto (väčšom) klastri náhodne zvolíme dva centroidy
- Následne vykonávame celý proces k-means kde stred je centroid, pritom hodnota k, čo predstavuje počet náhodne zvolených klastrov bude vždy 2
- Po tomto procese budeme mať o jeden klaster viacej, čiže opäť ideme všetky body priraďovať tomu klastru ku ktorému má najmenšiu vzdialenosť
- Opäť opakujeme celý postup od bodu č.2, čím nám bude vznikať stále viacej a viacej klastrov

Tento postup opakujeme dovtedy, až kým všetky z vytvorených klastrov nemajú menšiu priemernú vzdialenošt od stredu ako 500.

3 Popis implementácie

3.1 Vygenerovanie bodov

Všetky body v mojom riešení sú ukladané v poliach, to znamená, že ak mám polia napr. x[] a y[], tak x[0] a y[0] budú súradnice prvého bodu, x[1] a y[1] budú súradnice druhého bodu atď..

```
def create_points(menu):
    x_randoms = range(-5000, 5000)
    x = random.sample(x_randoms, 20)

    y_randoms = range(-5000, 5000)
    y = random.sample(y_randoms, 20)

    # print(x)
    # print(y)


    for i in range(0, 40000):
        # get random x
        randoms = range(0, len(x))
        my_random = random.sample(randoms,1)

        for j in range (0,len(x)):
            if(j == my_random[0]):
                new_x = x[j]
                x_offset = range(-100, 100)
                get_offset_x = random.sample(x_offset, 1)
                new_x = new_x + get_offset_x[0]
                x.append(new_x)

                new_y = y[j]
                y_offset = range(-100, 100)
                get_offset_y = random.sample(y_offset, 1)
                new_y = new_y + get_offset_y[0]
                y.append(new_y)
                break
```

Ako prvé si náhodne generujem mojich 40 020 bodov, a to tak, že najprv si náhodne zvolím 20 bodov, v rozmedzí od -5000 a 5000. Následne si v cykle for, ktorý beží 40 tisíc krát vygenerujem ďalších 40 tisíc bodov, a to tak že si náhodne zvolím jeden bod z už doteraz vygenerovaných bodov a následne tomuto bodu pridám k obidvom súradnicam aj X aj Y, náhodný offset od -100 do 100, čím mi vzniknú súradnice nového bodu, ktorý následne pridám do môjho poľa súradníc, až kým takto nevytvorím všetkých 40 000 bodov.

3.2 K-means (stred - centroid)

3.2.1 Generovanie centroidov

```python
def get_first_20_centroids(x_values, y_values):
    centroids = []
    x = []
    y = []

    while (len(x) < 20 and len(y) < 20):
        x_randoms = range(-5000, 5000)
        x_axis = random.sample(x_randoms, 1)

        y_randoms = range(-5000, 5000)
        y_axis = random.sample(y_randoms, 1)

        pom = 0
        for i in range(0, len(x)):
            if((x[i] != x_axis[0] and y[i] != y_axis[0]) or (x[i] == x_axis[0] and y[i] != y_axis[0]) or (x[i] != x_axis[0] and y[i] == y_axis[0])
                continue
            if(x[i] == x_axis[0] and y[i] == y_axis[0]):
                pom = 1
                break

        if(pom == 0):
            x.append(x_axis[0])
            y.append(y_axis[0])
            continue
        else:
            continue


    centroids.append(x)
    centroids.append(y)

    return x, y
```

Pri algoritme k-means, kde stred je centroid si ako prvé vytváram 20 náhodných centroidov. Vytváram si ich v cykle while, ktorý mi beží až kým ich nebude vytvorených 20, a generujem ich tak že si náhodne zvolím súradnice X a Y, pričom tieto súradnice následne porovnávam so súradnicami ďalších bodov a centroidov, aby sa nestalo, že napr. dva centroidy budú mať rovnaké súradnice, čiže sa budú nachádzať na tom istom mieste.

3.2.2 Priradenie bodov k centroidom, a zmena pozície centroidov

```python
def assign_points_to_centroids(centroids_x, centroids_y, x, y, value):
    clusters_x = [[centroids_x[0]], [centroids_x[1]], [centroids_x[2]], [centroids_x[3]], [centroids_x[4]],
                  [centroids_x[5]], [centroids_x[6]], [centroids_x[7]], [centroids_x[8]], [centroids_x[9]],
                  [centroids_x[10]], [centroids_x[11]], [centroids_x[12]], [centroids_x[13]], [centroids_x[14]],
                  [centroids_x[15]], [centroids_x[16]], [centroids_x[17]], [centroids_x[18]], [centroids_x[19]]]
                  # [centroids_x[20]], [centroids_x[21]], [centroids_x[22]], [centroids_x[23]], [centroids_x[24]],
                  # [centroids_x[25]], [centroids_x[26]], [centroids_x[27]], [centroids_x[28]], [centroids_x[29]]]


    clusters_y = [[centroids_y[0]], [centroids_y[1]], [centroids_y[2]], [centroids_y[3]], [centroids_y[4]],
                  [centroids_y[5]], [centroids_y[6]], [centroids_y[7]], [centroids_y[8]], [centroids_y[9]],
                  [centroids_y[10]], [centroids_y[11]], [centroids_y[12]], [centroids_y[13]], [centroids_y[14]],
                  [centroids_y[15]], [centroids_y[16]], [centroids_y[17]], [centroids_y[18]], [centroids_y[19]]]
                  # [centroids_y[20]], [centroids_y[21]], [centroids_y[22]], [centroids_y[23]], [centroids_y[24]],
                  # [centroids_y[25]], [centroids_y[26]], [centroids_y[27]], [centroids_y[28]], [centroids_y[29]]]
```

Jednotlivé klastre si ukladám do dvojrozmerného poľa, pričom vždy prvé súradnice v jednotlivých klastrov budú predstavovať súradnice centroidov. To znamená, že napr. premenná clusters_x[0][0] bude súradnica X pre prvý centroid a premenná clusters_y[0][0], bude súradnica Y pre prvý centroid, všetky ostatné súradnice už budú patriť jednotlivým bodom daného klastra.

```
# BEFORE
clusters_x, clusters_y = assign_points_into_clusters(clusters_x, clusters_y, x, y)

if(value == 'k'):
    print_average_points_distance(clusters_x, clusters_y, 'k', 'p')
    show_plot(clusters_x, clusters_y, len(clusters_x))
```

Následne ako prvé si priradím pomocou funkcie assign_points_into_cluster(), všetky body k jednotlivým centroidom, podľa toho ku ktorému majú najmenšiu vzdialenosť.

Potom už len vypíšem priemernú dĺžku bodov od jeho stredu jednotlivých klastrov.

```
# AFTER
sums = []
for m in range(0,100):
    for i in range(0, len(clusters_x)):
        xx = 0
        yy = 0

        for j in range(1, len(clusters_x[i])):
            xx = xx + clusters_x[i][j]
            yy = yy + clusters_y[i][j]

        average_x = xx / len(clusters_x[i])
        average_y = yy / len(clusters_y[i])

        clusters_x[i][0] = average_x
        clusters_y[i][0] = average_y

    for i in range(0,len(clusters_x)):
        del clusters_x[i][1:]
        del clusters_y[i][1:]

    clusters_x, clusters_y = assign_points_into_clusters(clusters_x, clusters_y, x, y)
```

Ako ďalšie si v cykle for nájdem nové súradnice centroidu v každom klastri, a to tak že, si spočítam najprv všetky X súradnice, potom Y súradnice, a tieto dve hodnoty vydelím počtom všetkých súradníc, čo znamená že si ako keby jednoducho vypočítam stred daného klastra, tým mi vznikne centroid s novými súradnicami, následne premažem všetky klaste s tým že súradnice nových centroidov mi v daných poliach ostanú, ako ďalšie opäť priraďujem všetky body k daným centroidom, pomocou funkcie assign_points_into_clusters().

```
if (value == 'k'):
    big_sum = print_average_points_distance(clusters_x, clusters_y, 'k', 'p')
else:
    pom = ''
    big_sum = print_average_points_distance(clusters_x, clusters_y, 'k', pom)

sums.append(big_sum)

if(m > 0):
    if(int(sums[m]) == int(sums[m-1])):
        break
```

Ako posledné už len v cykle vypočítam a vypíšem priemerné vzdialenosti všetkých bodov od stredu v každom klastri, a ďalšie si ešte vypočítam aj priemer všetkých týchto vzdialeností, kde následne porovnávam či tento priemer bol rovnaký ako priemer predošlej iterácie, ak áno to znamená že centroidy už nezmenili svoju pozíciu od tej predošlej, z čoho vyplýva že cyklus môžeme ukončiť.

```
if (value == 'k'):
    show_plot(clusters_x, clusters_y, len(clusters_x))

return clusters_x, clusters_y
```

Nakoniec už len zobrazíme vizualizáciu bodov a klastrov v našom 2D priestore.

3.3 K-means (stred - medoid)

3.3.1 Generovanie medoidov

```
def get_medoids(x_values, y_values, number):
    x = []
    y = []


    randoms = range(0, len(x_values))
    my_random = random.sample(randoms, number)
    for j in range(0, len(x_values)):
        for m in range(0, len(my_random)):
            if(j == my_random[m]):
                x.append(x_values[j])
                y.append(y_values[j])

    # print(x)
    # print(y)

    return x, y
```

Ako prvé si náhodne vygenerujem 20 medoidov, a to tak že jednoducho si náhodne zvolím 20 X a Y súradníc, z mojich 40 tisíc vytvorených, ktoré následne priradím novým bodom, čo budú reprezentovať prvotné medoidy.

3.3.2 Priradenie bodov k medoidom, a zmena pozície medoidov

```
def assign_points_to_medoids(medoids_x, medoids_y, x, y, value):
    clusters_x = [[medoids_x[0]], [medoids_x[1]], [medoids_x[2]], [medoids_x[3]], [medoids_x[4]],
                  [medoids_x[5]], [medoids_x[6]], [medoids_x[7]], [medoids_x[8]], [medoids_x[9]],
                  [medoids_x[10]], [medoids_x[11]], [medoids_x[12]], [medoids_x[13]], [medoids_x[14]]]
                  # [medoids_x[15]], [medoids_x[16]], [medoids_x[17]], [medoids_x[18]], [medoids_x[19]]]
                  # [medoids_x[20]], [medoids_x[21]], [medoids_x[22]], [medoids_x[23]], [medoids_x[24]],
                  # [medoids_x[25]], [medoids_x[26]], [medoids_x[27]], [medoids_x[28]], [medoids_x[29]]]

    clusters_y = [[medoids_y[0]], [medoids_y[1]], [medoids_y[2]], [medoids_y[3]], [medoids_y[4]],
                  [medoids_y[5]], [medoids_y[6]], [medoids_y[7]], [medoids_y[8]], [medoids_y[9]],
                  [medoids_y[10]], [medoids_y[11]], [medoids_y[12]], [medoids_y[13]], [medoids_y[14]]]
                  # [medoids_y[15]], [medoids_y[16]], [medoids_y[17]], [medoids_y[18]], [medoids_y[19]]]
                  # [medoids_y[20]], [medoids_y[21]], [medoids_y[22]], [medoids_y[23]], [medoids_y[24]],
                  # [medoids_y[25]], [medoids_y[26]], [medoids_y[27]], [medoids_y[28]], [medoids_y[29]]]


    # BEFORE
    clusters_x, clusters_y = assign_points_into_clusters(clusters_x, clusters_y, x, y)

    if(value == 'k'):
        print_average_points_distance(clusters_x, clusters_y, 'k', 'p')
        show_plot(clusters_x, clusters_y, len(clusters_x))
```

Začiatok funkcie je totožný s funkciou ktorá rieši k-means, kde stred je centroid (viď kapitola 3.2.2).

```
# AFTER
sums = []
for m in range(0,100):
    for i in range(0, len(clusters_x)):
        xx = 0
        yy = 0

        for j in range(1, len(clusters_x[i])):
            xx = xx + clusters_x[i][j]
            yy = yy + clusters_y[i][j]

        average_x = xx / len(clusters_x[i])
        average_y = yy / len(clusters_y[i])

        a = (average_x, average_y)
        b = (clusters_x[i][1], clusters_y[i][1])
        dist = distance.euclidean(a, b)
        min = dist
        index = 0
        for j in range(2, len(clusters_x[i])):
            b = (clusters_x[i][j], clusters_y[i][j])
            dist = distance.euclidean(a, b)

            if (dist < min):
                min = dist
                index = j

        clusters_x[i][0] = clusters_x[i][index]
        clusters_y[i][0] = clusters_y[i][index]
```

Ako ďalšie mám opäť cyklus for ako v predošlej funkcií, a opäť si v ňom pre každý klaster vypočítam najprv všetky X súradnice, potom Y súradnice, a tieto dve hodnoty vydelím počtom všetkých súradníc, čo znamená že si ako keby jednoducho vypočítam stred, následne ale ešte prechádzam všetky body v danom klastri a ten bod, ktorý sa nachádza najbližšie ku stredu vznikne ako náš nový medoid.

```
for i in range(0,len(clusters_x)):
    del clusters_x[i][1:]
    del clusters_y[i][1:]

assign_points_into_clusters(clusters_x, clusters_y, x, y)
```

Následne opäť len premažem všetky klastre, pričom mi v klastroch ostanú iba súradnice nových medoidov, a následne opäť priradím všetky body k daným medoidom pomocou funkcie assign_points_into_clusters().

```
    if (value == 'k'):
        big_sum = print_average_points_distance(clusters_x, clusters_y, 'k', 'p')
    else:
        pom = ''
        big_sum = print_average_points_distance(clusters_x, clusters_y, 'k', pom)

    sums.append(big_sum)

    if(m > 0):
        if(int(sums[m]) == int(sums[m-1])):
            break

if(value == 'k'):
    show_plot(clusters_x, clusters_y, len(clusters_x))

return clusters_x, clusters_y
```

Ďalší postup už je totožný s postupom vo funkcií ktorá rieši k-mean, kde stred je centroid (viď kapitola 3.2.2.).

3.4 Divízne zhlukovanie (stred - centroid)

```
def division_clustering(x_values, y_values):
    clusters_x = []
    clusters_y = []

    first_cluster_x = []
    first_cluster_y = []

    second_cluster_x = []
    second_cluster_y = []

    for i in range(0, len(x_values)):
        if(-5000 < x_values[i] < 0):
            first_cluster_x.append(x_values[i])
            first_cluster_y.append(y_values[i])
        if(0 < x_values[i] < 5000):
            second_cluster_x.append(x_values[i])
            second_cluster_y.append(y_values[i])

    clusters_x.append(first_cluster_x)
    clusters_x.append(second_cluster_x)
    clusters_y.append(first_cluster_y)
    clusters_y.append(second_cluster_y)
```

Ako prvé si ako keby rozdelím 2D plochu na polku podľa osi X, čím mi vzniknú prvotné dva klastre.

```
# FIRST ASSIGN POINTS
for i in range(0, len(clusters_x)):
    xx = 0
    yy = 0

    for j in range(1, len(clusters_x[i])):
        xx = xx + clusters_x[i][j]
        yy = yy + clusters_y[i][j]

    average_x = xx / len(clusters_x[i])
    average_y = yy / len(clusters_y[i])

    clusters_x[i][0] = average_x
    clusters_y[i][0] = average_y

    for j in range(0, len(clusters_x)):
        del clusters_x[i][1:]
        del clusters_y[i][1:]

    clusters_x, clusters_y = assign_points_into_clusters(clusters_x, clusters_y, x_values, y_values)
```

Následne si v týchto dvoch klastroch nájdem pozíciu centroidov ako v predošlom algoritme k-means, kde stred bol medoid, a potom opäť volám funkciu assign_points_into_clusters(), ktorá mi vlastne priradí body do prvotných dvoch klastrov.

```
# LOOP ASSIGN POINTS
for m in range(0,100):
    pom = ''
    maximum = print_average_points_distance(clusters_x, clusters_y, 'd', pom )

    if(m == 0):
        show_plot(clusters_x, clusters_y, len(clusters_x))

    x_min = min(clusters_x[maximum])
    x_max = max(clusters_x[maximum])
    y_min = min(clusters_y[maximum])
    y_max = max(clusters_y[maximum])

    centroids_x, centroids_y = get_2_centroids(x_max, x_min, y_max, y_min)

    new_centroids_x, new_centroids_y = assign_points_to_centroids(centroids_x, centroids_y, clusters_x[maximum], clusters_y[maximum], 'd')

    del clusters_x[maximum]
    del clusters_y[maximum]
```

Následne mám opäť cyklus for, v ktorom si ako prvé nájdem klaster, ktorý má najväčšiu priemernú vzdialenosť bodov od jeho stredu, ako ďalšie si v tomto danom klastri nájdem dva náhodne zvolené centroidy, následne volám funkciu assign_points_to_centroids (fungovanie celej tejto funkcie je popísane v kapitole 3.2.2, kde popisujem algoritmus k-means, kde stred je centroid) pričom v tomto prípade budem pracovať iba s dvomi centroidmi, kde mi následne táto funkcia vráti súradnice týchto dvoch centroidov.
Ako ďalšie premažem celý klaster, ktorý bol označený ako najväčší klaster.

```
for i in range(0, len(clusters_x)):
    del clusters_x[i][1:]
    del clusters_y[i][1:]

for i in range(0,2):
    clusters_x.append(new_centroids_x[i])
    clusters_y.append(new_centroids_y[i])

clusters_x, clusters_y = assign_points_into_clusters(clusters_x, clusters_y, x_values, y_values)

sums = print_average_points_distance(clusters_x, clusters_y, 's', 'p')
```

Následne premažem všetky ostatné klastre, pričom súradnice centroidov mi ostali. Ako ďalšie priradím dva nové centroidy, ktoré mi vrátila funkcia vyššie k ostaným centroidom, a potom opäť volám funkciu assign_points_into_clusters(), ktorá mi priradí každý bod k centroidu, ku ktorému má najmenšiu vzdialenosť.
Následne už len vypíšem priemerné vzdialenosti všetkých bodov od stredu v každom doteraz vytvorenom klastri.

```
the_500 = 0
for i in range(0,len(sums)):
    if(sums[i] > 500):
        the_500 = 1
        break

if(the_500 == 0):
    print("This are final clusters !")
    show_plot(clusters_x, clusters_y, len(clusters_x))
    break

show_plot(clusters_x, clusters_y, len(clusters_x))
```

Ako posledné v cykle už len zisťujem či všetky priemerné vzdialenosti všetkých bodov od stredu v každom doteraz vytvorenom klastri sú menšie ako 500, ak hej tak spĺňam podmienku, cyklus končí a už len graficky zobrazím 2D priestor, s danými bodmi a zhlukmi. Ak podmienka ešte nie je splnená to znamená že ešte existujú klastre, ktoré majú väčšiu priemernú vzdialenost všetkých bodov od stredu ako 500, tak cyklus znova opakujem.
Takýmto spôsobom budem vytvárať stále nové a menšie klastre, až kým nebude splnená daná podmienka.

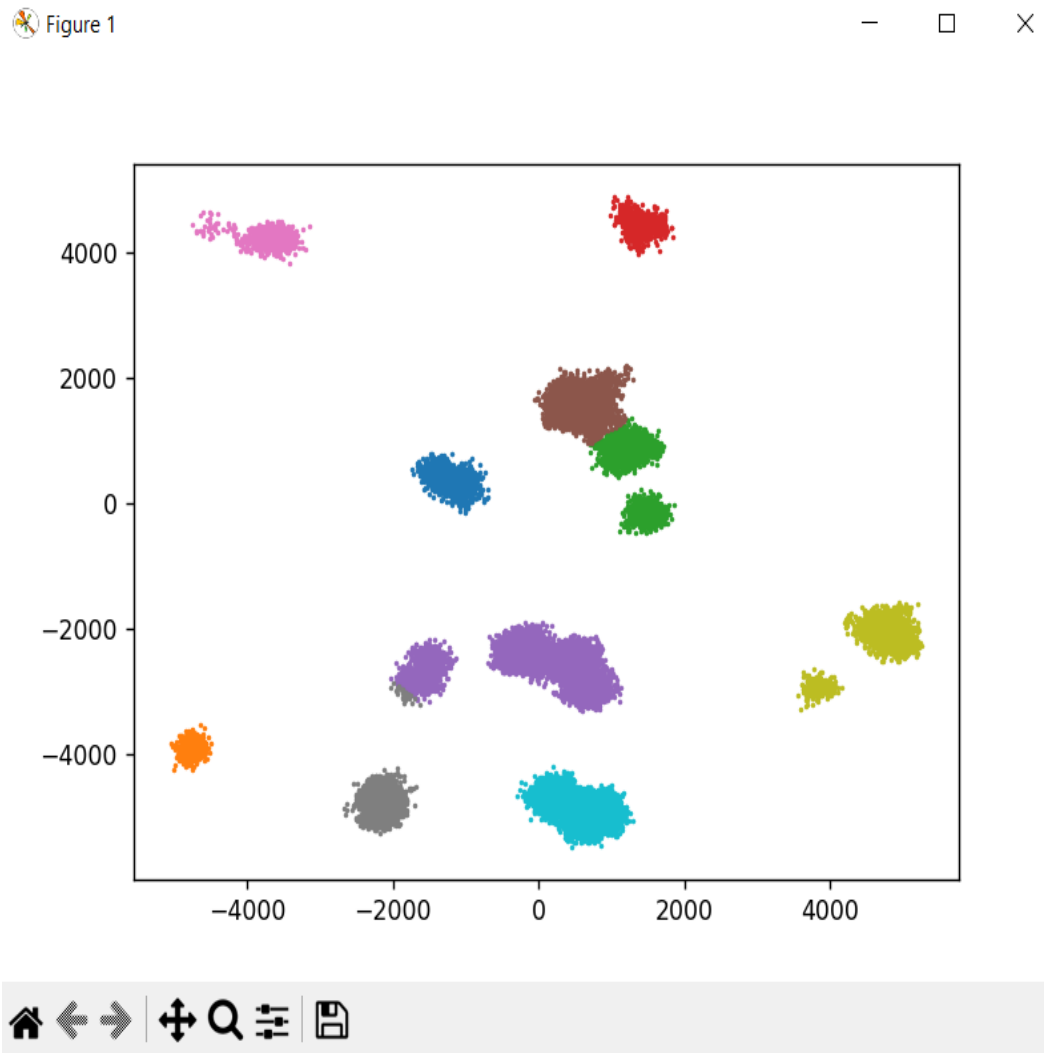
4 Testovanie

4.1 K-means (stred - centroid)

4.1.1 K = 10

Takto vyzerá výsledok testovania pre algoritmus k-means, kde stred je centorid, pričom hodnotu k sme mali nastavenú na 10, čo znamená že sme pracovali s 10 centroidmi.

Aj keď úloha bola stavaná viacmenej na to aby vyhovovala 20 centroidom, tak môžeme vidieť že už pri 10 centroidoch algoritmus dosiahol naozaj veľmi dobré výsledky, kde iba jeden kluster mal priemernú vzdialenosť všetkých bodov od stredu väčšiu ako 500.

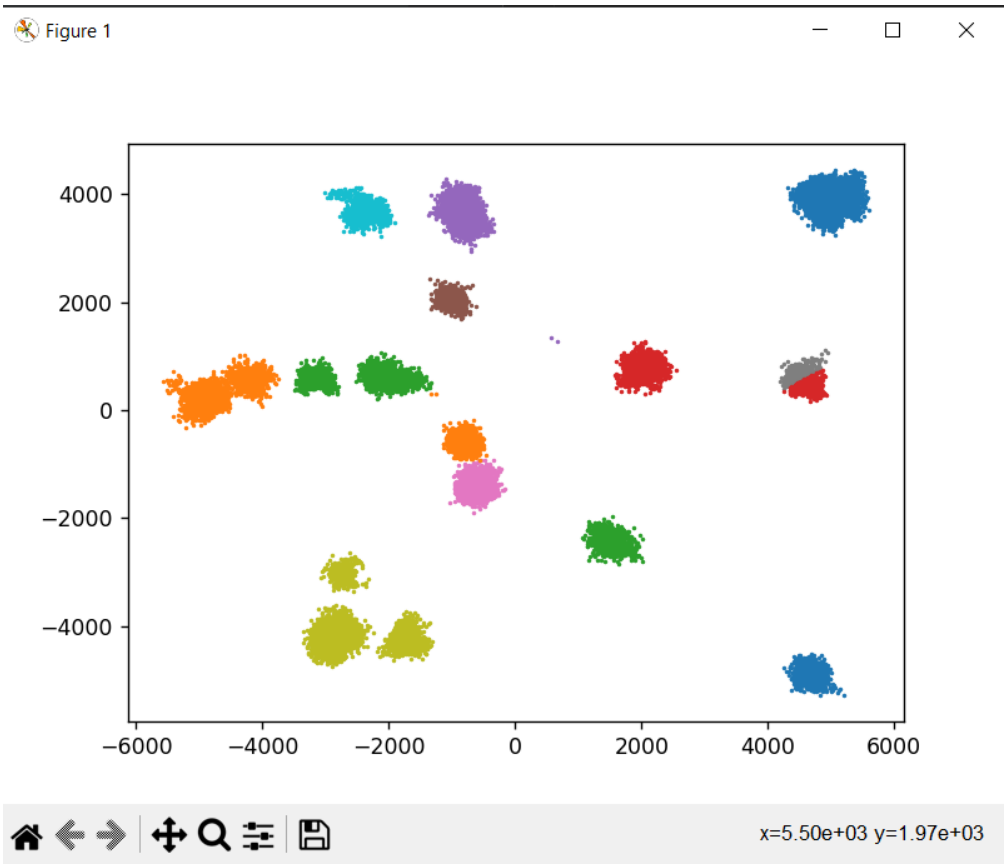



```
208.20202730837306
139.51694302695117
432.7009793250015
187.8345372332617
618.5213203137554
214.13235982392175
193.59000328663348
196.71799392663573
285.2554643977258
331.90814672219307
SUM =  280.8379775364453
```

4.1.2 K = 15

Aj pri hodnote K=15, môžeme vidieť opäť veľmi dobré výsledky testovania, v tomto prípade už žiaden klaster nemá priemernú vzdialenosť bodov od stredu väčšiu ako 500.

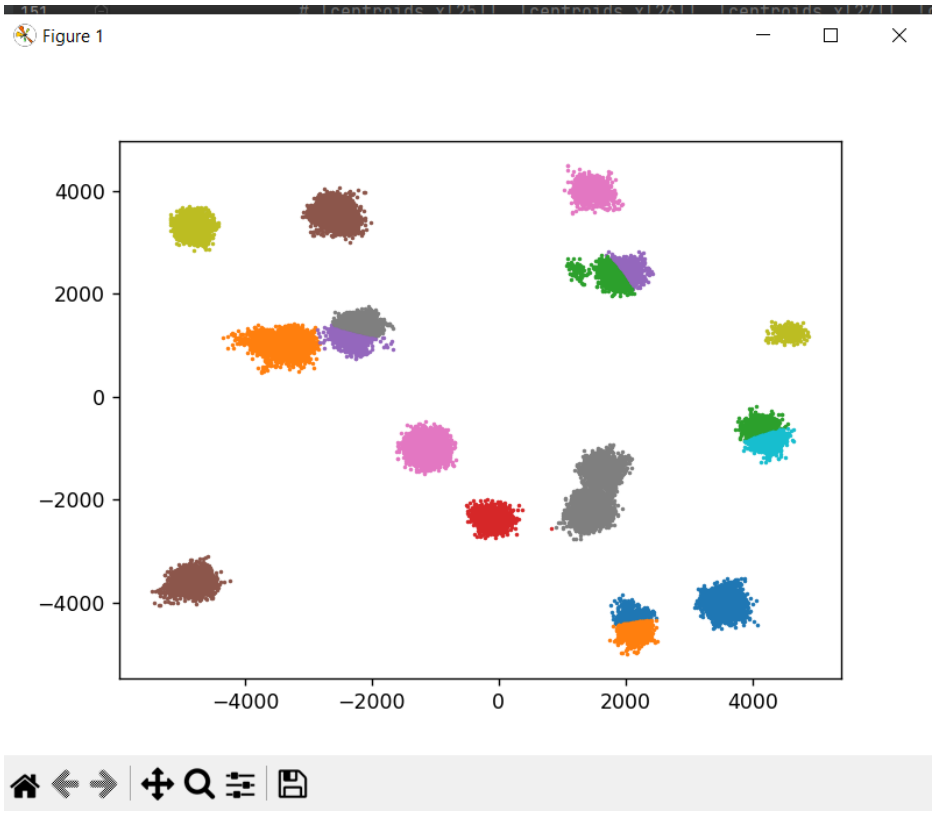
Avšak v tomto prípade už tu môžeme vidieť jeden zhluk, ktorý je rozdelený na dva klastre, ale pritom by mohol celý zhluk ako jeden klaster.




```
216.25041830345722
389.6687254036464
498.21759990531865
182.17925851165617
218.73171592945744
141.03689943706084
169.3480828227756
132.43378574109437
440.6685822249394
187.2878740061075
167.9894469936473
155.9489898279126
191.2784170056531
135.5167963876217
326.4147274576494
SUM =  236.86475466386645
```

4.1.3 K = 20

Pri hodnote K = 20 už môžeme vidieť že dokonca nám vznikli klastre resp. centroidy, ku ktorým neboli priradené žiadne body. Opäť je tam ešte viacej ako v predošlom testovaní takých zhlukov, ktoré sú rozdelené do dvoch klastrov, ale mohli by byť rozdelené iba do jedného klastra, z čoho vyplýva že z môjho pohľadu je už hodnota K = 20 zbytočne vysoká.

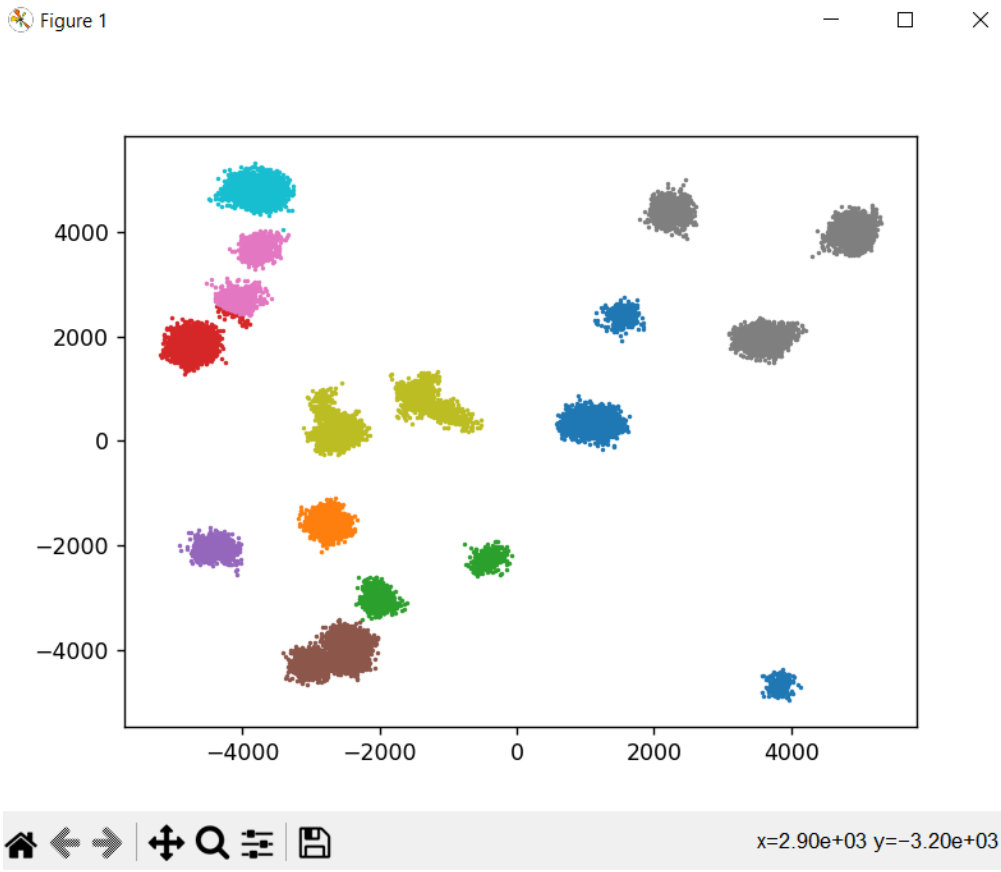



```
199.0190105322106
262.3876254587401
142.37582437953648
0.0
136.703453877956
171.03351418706708
186.52350838965742
420.9205566978624
169.27937557678928
0.0
127.62089553671822
151.3143899995218
167.40895945388252
172.83866013296932
169.54143558048412
197.45388859199977
189.56586887691358
148.7024786949468
138.34964779776894
149.42493720401293
SUM =  165.02320154845185
```

4.2 K-means (stred - medoid)

4.2.1 K = 10

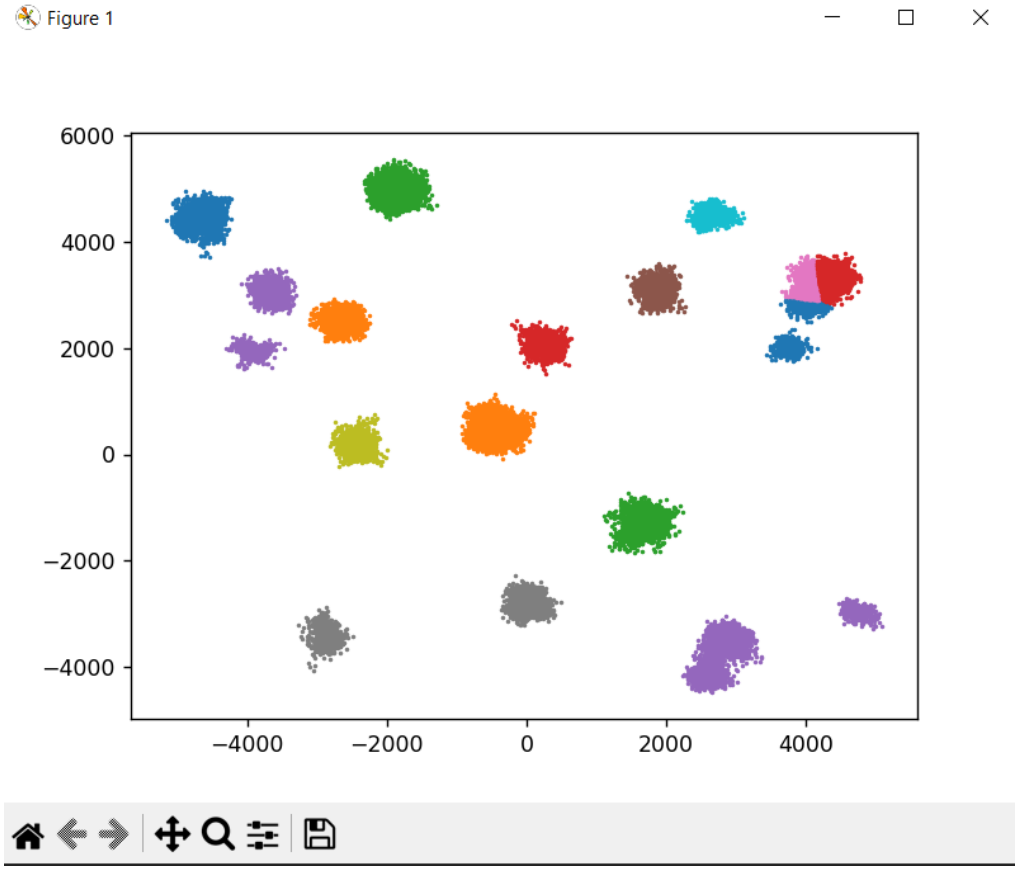
V prípade K-means, kde stred je medoid už testovanie nedosahuje také dobré výsledky. A to z dôvodu, ktorý môžeme vidieť aj v tomto prípade. V tomto prípade veľmi záleží na tom ako nám náhodne vygeneruje prvotné medoidy, pretože sa môže stať že niektoré medoidy nám vygeneruje príliš vedľa seba, pričom zase na druhej strane môže byť vygenerovaný medoid, ktorý bude musieť obhospodáriť príliš veľkú plochu, a tým pádom bude mať aj vysoké číslo čo sa týka priemernej vzdialenosti všetkých bodov od stredu, aj keď v tomto prípade sú výsledky testovania pomerne dobré.




```
695.0496839584949
177.07139732686488
691.4062312287001
179.39784033852845
159.44393635888935
316.5164023399858
508.22960026942917
1460.3891086758072
661.455238280759
184.9083147416215
SUM =  503.386775351908
```

4.2.2 K = 15

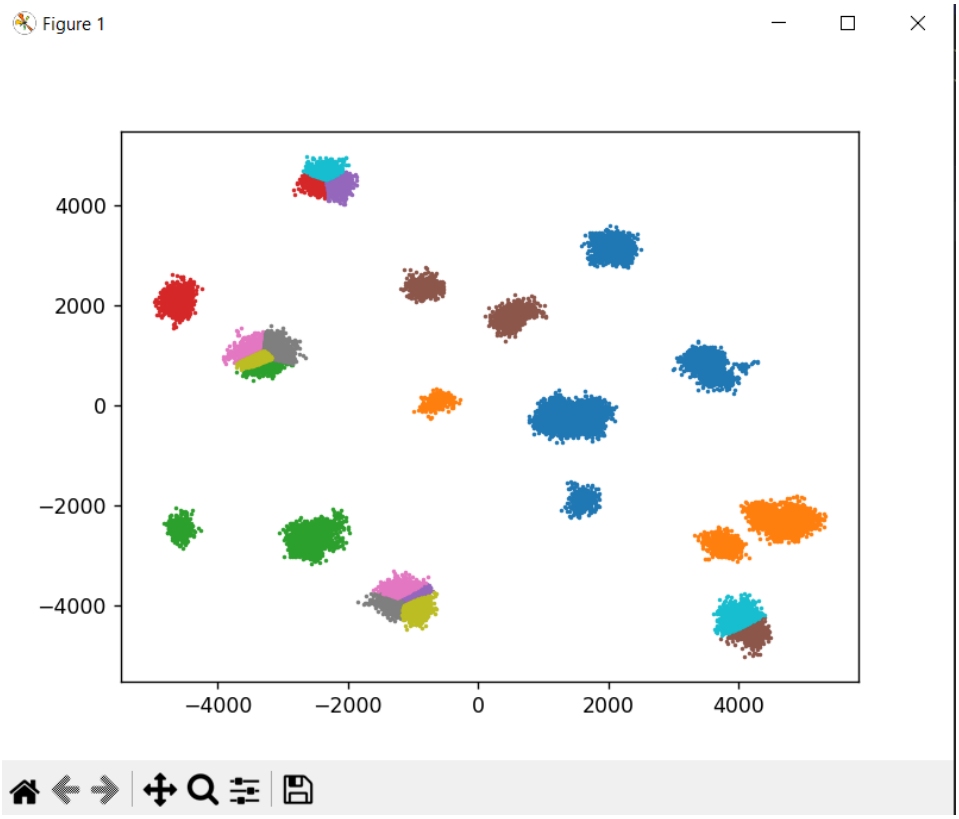
Aj v tomto prípade môžeme vidieť, že žiadny klaster nemá priemernú vzdialenosť všetkých bodov od stredu väčšiu ako 500. Aj napriek tomu že v tomto prípade testovania nám to vyšlo pomerne pekne, tak vidíme že jeden zhluk, ktorý mohol byť ako jeden klaster je rozdelený dokonca až do troch klastrov, ale tieto situácie sú ovplynené len tým ako blízko boli náhodne vygenerované medoidy.




```
393.6617642479053
168.04053281434045
190.7591688003785
169.5992039742916
327.5330881677825
169.99085947503036
177.0764043966344
1123.238394396722
171.80621973556845
151.99664147184396
190.44870598362562
218.78490626196452
186.1488691179387
159.53866781883718
484.35760184549514
SUM =  285.5320685672239
```

4.2.3 K = 20

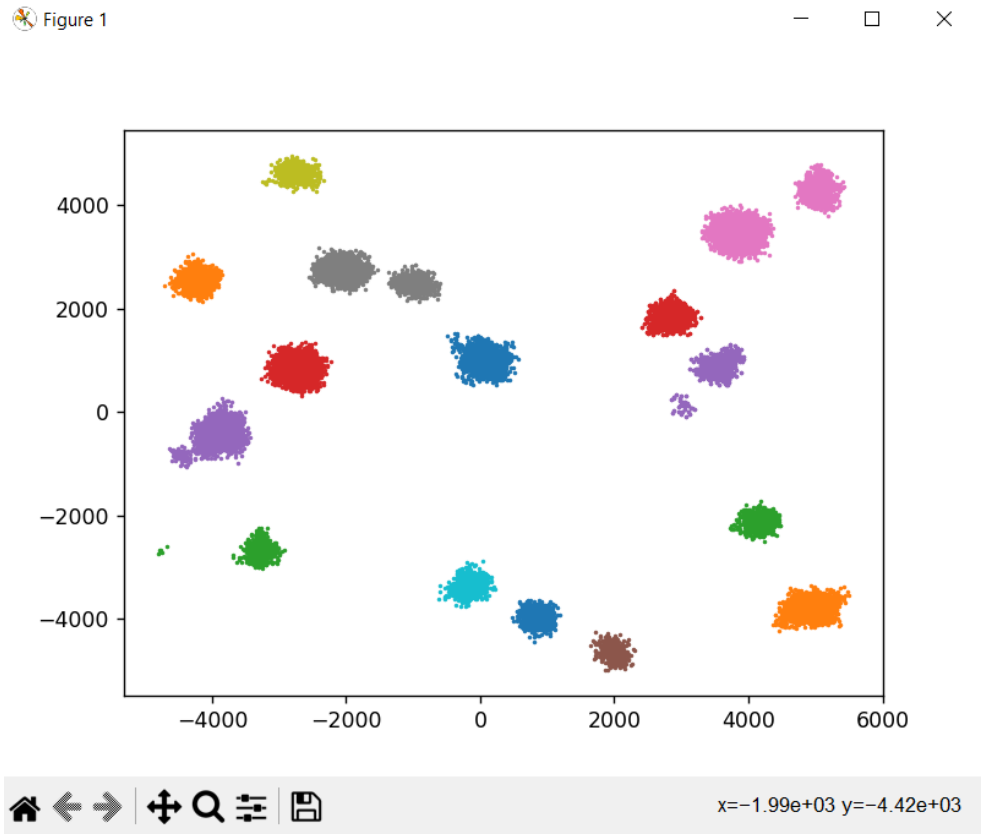
Podľa môjho názoru je táto hodnota pre k-means, kde stred je medoid až príliš vysoká, pretože ako môžeme vidieť už sa nám tu veľmi často vyskytujú dva extrémy. Na jednej strane máme už pomerne dosť zhlukov rozdelených do príliš veľa klastrov, pričom to mohol byť jeden maximálne dva klastre, a zase na druhej strane máme klastre ktoré sú príliš veľké. Táto situácia je len z toho dôvodu že náhodne generujeme počiatočné pozície medoidov, čo má za následok to, že niekde tých medoidov môže byť viac na jednej kope, inde zase môže byť na veľkej vygenerovaný iba jeden medoid.




```
180.49702674737603
444.8007757113863
566.3399845740023
176.26728147014373
102.92154440757764
143.08303604158667
117.93740934033616
120.03481048410066
122.14979301501039
147.0710668497279
897.0515833139559
138.99170596320775
124.32560476962773
110.67102730186477
121.29343960998831
617.8084435729446
123.69743402310982
129.2063781577054
91.11010090241453
116.18155694134727
SUM =  229.57200015987064
```

4.3 Divízne zhlukovanie (stred - centroid)

Z môjho pohľadu je tento typ zhlukovania najefektívnejší, pretože algoritmus si vytvorí presne toľko klastrov, koľko potrebuje nato aby mal každý priemernú vzdialenosť bodov od stredu menšiu ako 500. Ako môžeme aj vidieť vo výsledkoch testovania, algoritmus si vytvoril 15 klastrov, a všetky majú priemernú vzdialenosť bodov od stredu menšiu ako 500.




```
191.82441299738076
169.38859900571128
167.724805049849
193.34419880080787
188.39379243077673
154.6040425249611
318.49224219165313
341.0605460006573
157.66479949635055
165.2756985559526
152.9059976504478
199.3248473911914
174.79675321356973
159.6998214929186
197.31414583892513
SUM =  195.45431350941018
-----------------
This are final clusters !
```

4.4 Zhodnotenie a porovnanie testovania

Ak by som mal porovnať a vybrať z daných troch implementovaných algoritmov, tak podľa môjho názoru najlepšie a najefektívnejšie pracuje algoritmus divízneho zhlukovania, pretože si vytvorí presne toľko klastrov koľko potrebuje na splnenie požiadavok, čo znamená že nech sú body a zhluky akokoľvek rozmiestnené vždy splní podmienku úspešného zhlukovača, pretože si to dokáže upravovať podľa seba.

Pričom na druhej strane pri algoritmoch k-means je to do veľkej miery ovplyvnené aj tým ako máme rozmiestnené body a zhluky, ale hlavne tým ako sa nám náhodne rozmiestnia počiatočné centroidy alebo medoidy. Vďaka týmto faktorom nám následne môžu vzniknúť dva extrémy, ktoré už boli spomínané aj vyššie. Prvý extrém je ten, že ak je viacero centroidov alebo medoidov vygenerovaných príliš vedľa seba, bude to mať za príčinu to, že zhluk, ktorý mohol byť ako jeden klaster je rozdelený do dvoch alebo dokonca viacerých klastrov, na druhej strane môže byť vygenerovaný centroid alebo medoid, ktorý obhospodaruje príliš veľkú časť priestoru, čo bude mať za následok to, že hodnota priemeru vzdialeností všetkých bodov od stredu bude príliš vysoká.

Dokonca, k-means, kde stred je medoid, je ešte znevýhodnený o tú vec, že prvotné medoidy vyberáme z už vytvorených bodov, čo znamená že je ešte väčšia pravdepodobnosť výskytu bodov, ktoré sa nachádzajú príliš vedľa seba.

Takže, z môjho pohľadu by som algoritmus divízneho zhlukovania označil za najlepší a najefektívnejší, na druhej strane algoritmus k-means, kde stred je medoid by som označil za najmenej úspešný a efektívny.

5 Záver

Implementácia daných algortimov funguje bez akýchkoľvek problémov. Myslím si, že výsledky testovania splnili očakávania. Program je implementovaný čo najefektívnejšie, tak aby bežal čo najrýchlejšie a využíval čo najmenej pamäti.